Aktu

Database Management System

BCS-501

Unit-3

Syllabus

Data Base Design & Normalization: Functional dependencies, normal forms, first, second, 8 third normal forms, BCNF, inclusion dependence, loss less join decompositions, normalization using FD, MVD, and JDs, alternative approaches to database design

Topics Covered

- **Introduction to Database Design & Normalization**
- **What is a Functional Dependency? - (Aktu 23-24)**
- **Armstrong's Axioms - (Aktu 21-22)**
- **Types of Functional Dependencies**
- **Canonical Cover in FDs -(Aktu 23-24)**
- **Normalization and Normal Forms**
- **Step-by-Step Explanation of Normal Forms (1st,2nd,3rd & BCNF)**
- **Prime & Non Prime Attributes**
- **How to Find Candidate Key**
- **Inclusion Dependency**
- **Lossless Join Decomposition**
- **Multivalued Dependencies (MVDs)**
- **Join Dependency (JD)**
- **4NF & 5NF**
- **Alternate Approaches to Database Design**

Introduction to Database Design & Normalization

What is Database Design?

It is the process of organizing data into structured formats that ensure consistency, efficiency, and accuracy. It involves defining:

- **Schema**: The logical structure of the database, including tables, columns, and relationships.
- **Constraints**: Rules to maintain data integrity.

- **Relationships**: How data in one table is related to data in another (e.g., primary and foreign keys).

What is Normalization?

It is a systematic process in database design to organize data into multiple tables to:

- **Reduce redundancy**: Avoid duplicate data storage.
- **Ensure data integrity**: Prevent anomalies like inconsistent updates or deletions.

It involves dividing large tables into smaller, well-structured tables while maintaining relationships between them. This process adheres to a set of rules, called normal forms, to achieve an efficient and error-free database design.

Importance of Normalization

- Eliminates Redundancy
- Improves Data Integrity
- Facilitates Scalability

Normalized databases are easier to manage and scale as the amount of data grows.

Example

Unnormalized Table:

StudentID	Name	Course	Instructor	InstructorPhone
101	Alice	Math	Dr. Smith	9876543210
101	Alice	Physics	Dr. Brown	8765432109

Problems:

- **Redundancy:** "Alice" and her details are repeated.
- **Update Anomaly:** Changing "Dr. Smith's" phone number requires multiple updates.
- **Delete Anomaly:** Deleting "Math" for Alice might remove her instructor details.

Normalized Table:

Table 1 (Students):

StudentID	Name
101	Alice

CourseID	Course	Instructor
C1	Math	Dr. Smith
C2	Physics	Dr. Brown

Table 2 (Courses):

Table 3 (Instructors):

Instructor	Phone
Dr. Smith	9876543210
Dr. Brown	8765432109

Table 4 (Enrollments):

StudentID	CourseID
101	C1
101	C2

What is a Functional Dependency?

AKTU- 2023-24

A Functional Dependency (FD) is a rule that defines how one column in a table determines the value of another column.

Denoted as X → Y, where:

- X (Determinant): The column(s) whose values decide another column.
- Y (Dependent): The column whose value depends on X.

StudentID	Name	Course
101	Alice	Math
102	Bob	Physics
103	Charlie	Chemistry

- **StudentID → Name**: If you know the StudentID, you can find the Name.
- **Course → Name**: This is not true, as the same course could be taken by multiple students.

Why is Functional Dependency Important?

- **Organize Data:** By identifying relationships between columns, we can design better databases.
- **Remove Redundancy:** Prevents storing unnecessary repeated data.
- **Avoid Errors:** Reduces issues like incorrect or inconsistent data.

AKTU- 2021-22

Armstrong's Axioms / Inference Rules for Functional Dependencies

Armstrong's axioms are basic rules to derive all possible functional dependencies (FDs) from a given set of FDs. There are three main axioms:

1. Reflexivity

- **Rule:** If a set of attributes (Y) is a subset of another set of attributes (X), then X → Y holds true.
- **Meaning:** Any attribute or group of attributes determines itself or its subset.

Roll_No	Name	Dept
101	Alice	Science
102	Bob	Arts

- **{Roll_No, Name} → Name**: Since Name is a part (subset) of {Roll_No, Name}, this FD is valid.
- **Roll_No → Roll_No**: Any column always determines itself.

2. Augmentation

- **Rule:** If X → Y is a valid FD, then X + Z → Y + Z is also valid.
- **Meaning:** Adding the same attribute(s) (Z) to both sides of a valid FD doesn't change the dependency.

Roll_No	Name	Dept	Building
101	Alice	Science	A1
102	Bob	Arts	A2

- **Roll_No → Name:** If you know Roll_No, you can determine Name.
- **By augmentation, Roll_No + Dept → Name + Dept is also valid.**

3. Transitivity

- **Rule**: If X → Y and Y → Z are valid FDs, then X → Z is also valid.
- **Meaning:** If one attribute depends on a second, and the second depends on a third, then the first indirectly determines the third.

Roll_No	Dept	Building
101	Science	A1
102	Arts	A2

- **Roll_No → Dept: Knowing Roll_No gives the Dept.**
- **Dept → Building: Knowing Dept gives the Building.**
- **By transitivity, Roll_No → Building is valid.**

Other Properties Derived from Axioms:

- **Union: If X → Y and X → Z, then X → YZ.**
- **Decomposition: If X → YZ, then X → Y and X → Z.**

Types of Functional Dependencies

1. Trivial Functional Dependency

A dependency is trivial if the dependent column is already part of the determinant.

StudentID	Name
101	Alice
102	Bob

- **{StudentID, Name} → Name:** This is trivial because Name is already part of the determinant {StudentID, Name}.
- **StudentID → StudentID:** This is also trivial.

2. Non-Trivial Functional Dependency

A dependency is non-trivial if the dependent column is not part of the determinant.

StudentID	Name	Course
101	Alice	Math
102	Bob	Physics

- **StudentID → Name:** Knowing StudentID gives us the Name, which is not part of StudentID.

3. Multivalued Functional Dependency

It occurs when one column determines multiple columns that are independent of each other.

StudentID	Hobby	Skill
101	Painting	Python
101	Dancing	Python

- **StudentID → {Hobby, Skill}, but Hobby and Skill do not depend on each other.**

4. Transitive Functional Dependency

A dependency is transitive if one column depends on another through a third column.

StudentID	Class	Teacher
101	10A	Mr. Smith
102	10B	Ms. Johnson

- **StudentID → Class and Class → Teacher, so StudentID → Teacher is transitive.**

5. Partial Functional Dependency

It occurs when a non-key attribute depends only on part of a composite key, not the whole key.

CourseID	Semester	Credits
C101	Fall	3
C102	Spring	4

- **If {CourseID, Semester} is the composite key, but only CourseID → Credits, it's a partial dependency.**

6. Fully Functional Dependency

A dependency is fully functional when a column depends on the entire composite key, not just part of it.

OrderID	ProductID	Quantity
O101	P001	10
O102	P002	5

- **{OrderID, ProductID}** → **Quantity** depends on the full composite key, so it's fully functional.

Canonical Cover in Functional Dependencies

AKTU- 2023-24

It is the simplified version of a set of Functional Dependencies (FDs). It removes unnecessary dependencies and makes the FD set smaller without changing its meaning. This helps in tasks like normalization, decomposition, and checking dependency preservation.

Steps to Calculate Canonical Cover

Let's break it down step by step with a simple example:

Given: Functional Dependencies (FDs)

We have a relation $R(A, B, C, D)$ with the following FDs:

1. $A \rightarrow BC$
2. $B \rightarrow C$
3. $A \rightarrow B$
4. $AB \rightarrow C$

Step 1: Split RHS of FDs

If any FD has multiple attributes on the right-hand side, split it into multiple FDs with one attribute on the right side.

- $A \rightarrow BC$ becomes:
  - $A \rightarrow B$
  - $A \rightarrow C$

Now the updated FDs are:

1. $A \rightarrow B$
2. $A \rightarrow C$
3. $B \rightarrow C$
4. $A \rightarrow B$
1. 5. $AB \rightarrow C$

Step 2: Remove Redundant FDs

Check if any FD is unnecessary (i.e., it can be derived from the others).

- **FD $A \rightarrow B$ is repeated, so remove the duplicate.**

Updated FDs:

1. $A \rightarrow B$
2. $A \rightarrow C$
3. $B \rightarrow C$
4. $AB \rightarrow C$

Step 3: Remove Redundant Attributes from LHS

Check if any attribute in the left-hand side of an FD is unnecessary.

- **Let's analyze $AB \rightarrow C$:**
  - **Test if $A \rightarrow C$ can already imply $AB \rightarrow C$.**
  - **Yes, because $A \rightarrow C$ is already present.**

So, $AB \rightarrow C$ is redundant. Remove it.

Updated FDs:

1. $A \rightarrow B$
2. $A \rightarrow C$
3. $B \rightarrow C$

Step 4: Final Canonical Cover
The final set of simplified FDs is:

- $A \rightarrow B$
- $A \rightarrow C$
- $B \rightarrow C$

Normalization and Normal Forms

Normalization is the process of organizing data in a database to reduce redundancy and improve data integrity. It involves dividing a database into smaller tables and defining relationships between them to ensure consistency and eliminate anomalies like update, delete, and insert anomalies.

Importance of Normalization:

- **Minimizes Redundancy:** Reduces duplicate data.
- **Improves Data Integrity:** Ensures that data is accurate and consistent.
- **Simplifies Maintenance:** Makes it easier to update data.
- **Optimizes Queries:** Improves database performance by reducing unnecessary joins.

Step-by-Step Explanation of Normal Forms

1.First Normal Form (1NF):

A relation is in 1NF if:

- Each column contains atomic (indivisible) values.
- Each record is unique.

Example: Improper 1NF Table

StudentID	Name	Courses
1	Alice	Math, Physics
2	Bob	Chemistry

Solution (Convert to 1NF):

StudentID	Name	Course
1	Alice	Math
1	Alice	Physics
2	Bob	Chemistry

Prime and Non-Prime Attributes

Prime Attributes

- A prime attribute is an attribute that is part of at least one candidate key of a relation.
- **Candidate Key**: A minimal set of attributes that can uniquely identify every tuple (row) in a relation.

Non-Prime Attributes

- A non-prime attribute is an attribute that is not part of any candidate key.

Example

Consider a relation $R(A, B, C, D)$ with the following **Functional Dependencies (FDs):**

1. $AB \rightarrow C$
2. $C \rightarrow D$

Step 1: Find the Candidate Key(s):

- AB is a **candidate key** because it can uniquely identify all attributes in the relation.

Step 2: Identify Prime and Non-Prime Attributes:

- **Prime Attributes:** A, B (because they are part of the candidate key AB).
- **Non-Prime Attributes:** C, D (because they are not part of any candidate key).

How to Find Candidate Key

Given Relation:

$R(A, B, C, D)$

Functional Dependencies:

1. $AB \rightarrow C$
2. $C \rightarrow D$

Step 1: List All Attributes

$R = \{A, B, C, D\}$

Step 2: Identify Closures

- $AB^+ = \{A, B, C, D\}$: AB **determines all attributes.**
- $A^+ = \{A\}$: A **does not determine all attributes.**
- $B^+ = \{B\}$: B **does not determine all attributes.**
- $C^+ = \{C, D\}$: C **determines** D**, but not** A **or** B**.**

Step 3: Check Superkeys

- AB **is a superkey because** AB^+ **contains all attributes.**

Step 4: Identify Candidate Key

- AB **is the candidate key because it is minimal and can determine all attributes.**

Tips

1. Always start by calculating closures.
2. Look for minimal sets of attributes that can uniquely identify all attributes.
3. Eliminate redundancy to identify candidate keys.

2.Second Normal Form (2NF):

A table is in Second Normal Form (2NF) if:

- It is in First Normal Form (1NF)
- It has no partial dependencies, meaning no **non-prime attribute depends on a part of a composite key.**

Steps to Achieve 2NF

1. Start with a 1NF table.

Ensure no multi-valued or repeating attributes exist.

2. Check for Partial Dependencies.
Identify if any non-prime attribute depends on only a portion of a composite key.

3. Remove Partial Dependencies.
Create new tables to separate these dependencies.

4. Reorganize the Table.
Ensure that every non-prime attribute depends on the whole candidate key.

Example
Table: Student_Subject

Roll_No	Subject_Code	Subject_Name	Student_Name
1	MATH01	Math	Alice
2	PHYS01	Physics	Bob
1	PHYS01	Physics	Alice
2	MATH01	Math	Bob

Candidate Key: $\{Roll\_No, Subject\_Code\}$
Non-Prime Attributes: $Subject\_Name, Student\_Name$

Step 1: Check for Partial Dependencies

- **Subject_Name** depends only on **Subject_Code** (a part of the composite key).
  - $Subject\_Code \rightarrow Subject\_Name \rightarrow$ **Partial Dependency**
- **Student_Name** depends only on **Roll_No** (a part of the composite key).
  - $Roll\_No \rightarrow Student\_Name \rightarrow$ **Partial Dependency**

Step 2: Remove Partial Dependencies

Decompose into two tables:

1. Student Table

Roll_No	Student_Name
1	Alice
2	Bob

2. Subject Table

Subject_Code	Subject_Name
MATH01	Math
PHYS01	Physics

3. Student_Subject Table (Relationship Table)

Roll_No	Subject_Code
1	MATH01
1	PHYS01
2	MATH01
2	PHYS01

HomeWork for 2nd NF

StudentID	CourseID	StudentName	CourseName
1	101	Alice	Math
1	102	Alice	Physics

3.Third Normal Form (3NF):

A table is in Third Normal Form (3NF) if:

- It is in Second Normal Form (2NF).
- There are no transitive dependencies, meaning no non-prime attribute depends on another non-prime attribute.

Steps to Achieve 3NF

1.Start with a 2NF table.
Ensure there are no partial dependencies.

2.Check for Transitive Dependencies.
Identify if any non-prime attribute depends on another non-prime attribute.

3.Remove Transitive Dependencies.
Create new tables to separate these dependencies.

4.Reorganize the Table.
Ensure that every non-prime attribute is directly dependent on the key, not on another non-prime attribute.

Example
Table: Employee

Emp_ID	Dept_ID	Dept_Name	Dept_Location
1	D01	HR	New York
2	D02	IT	San Francisco
3	D01	HR	New York
4	D03	Admin	Chicago

Candidate Key: Emp_ID
Non-Prime Attributes: $Dept_ID, Dept_Name, Dept_Location$

Step 1: Check for Transitive Dependencies

- $Emp\_ID \rightarrow Dept\_ID$
- $Dept\_ID \rightarrow Dept\_Name, Dept\_Location$

Problem:

- $Dept\_Name$ **and** $Dept\_Location$ **are dependent on** $Dept\_ID$**, not directly on Emp_ID**

Step 2: Remove Partial Dependencies

Decompose into two tables:

1. Employee Table

Emp_ID	Dept_ID
1	D01
2	D02
3	D01
4	D03

2. Department Table

Dept_ID	Dept_Name	Dept_Location
D01	HR	New York
D02	IT	San Francisco
D03	Admin	Chicago

4.Boyce-Codd Normal Form (BCNF):

A table is in BCNF if:

- It is in Third Normal Form (3NF).
- For every functional dependency (X → Y), X (determinant) must be a superkey.

Key Terms

- **Superkey:** An attribute or a set of attributes that can uniquely identify a row.
- **Determinant:** The left-hand side (X) of a functional dependency $X \rightarrow Y$.

Steps to Achieve BCNF

1. **Check if the Table is in 3NF.**
   Start from a 3NF table.
2. **Verify Functional Dependencies.**
   For each dependency $X \rightarrow Y$, check if X is a superkey.
3. **Decompose if Needed.**
   If X is not a superkey, decompose the table into smaller tables until every determinant is a superkey.

Example
Table: Student

Roll_No	Subject	Teacher
1	Math	Mr. A
2	Physics	Mr. B
3	Math	Mr. A
4	Chemistry	Mr. C

Functional Dependencies:

1. $Roll\_No \rightarrow Subject$
2. $Subject \rightarrow Teacher$

Candidate Key: $Roll\_No$
Problem:

- $Subject \rightarrow Teacher$: $Subject$ **is not a superkey.**
- **This violates BCNF.**

Step 1: Decompose the Table

- **Split into two tables:**

1. Student_Subject Table

Roll_No	Subject
1	Math
2	Physics
3	Math
4	Chemistry

2. Subject_Teacher Table

Subject	Teacher
Math	Mr. A
Physics	Mr. B
Chemistry	Mr. C

Step 2: Verify BCNF

1. **In Student_Subject,** $Roll\_No \rightarrow Subject$**, and** $Roll\_No$ **is a superkey.**
2. **In Subject_Teacher,** $Subject \rightarrow Teacher$**, and** $Subject$ **is a superkey.**

AKTU- 2023-24

Q: Why do we normalize databases?

Normalization reduces redundancy, improves data integrity, avoids anomalies, and makes databases easier to maintain.

AKTU- 2022-23

Q: List all prime and non-prime attributes

- **Relation R(A, B, C, D, E)**
- **FDs:** $AB \rightarrow C, B \rightarrow E, C \rightarrow D$
- **Candidate Key:** AB.

Answer:
Prime Attributes: A, B.
Non-Prime Attributes: C, D, E.

Q: Find Key for R and Decompose into 2NF and 3NF

AKTU- 2022-23

Relation R: {A, B, C, D, E, F, G, H, I, J}
Functional Dependencies:
AB-> C
A-> DE
B->F
F->GH
D->IJ

Step 1: Find the Candidate Key

Given Functional Dependencies:

1. $AB \rightarrow C$
2. $A \rightarrow DE$
3. $B \rightarrow F$
4. $F \rightarrow GH$
5. $D \rightarrow IJ$

Attributes: $A, B, C, D, E, F, G, H, I, J$

Step-by-Step Process:

- AB **determines** C.
- A **determines** D, E **and** B **determines** F.
- F **determines** G, H **and** D **determines** I, J.

By combining, AB **can determine all other attributes (**$AB^+ = \{A, B, C, D, E, F, G, H, I, J\}$.

Candidate Key: AB **is the only candidate key.**

Step 2: Decompose into 2NF

To achieve 2NF, remove partial dependencies (when a non-prime attribute depends only on a part of the candidate key).

Partial Dependencies:

1. $A \rightarrow DE$ **(Depends on part of the key** A**).**
2. $B \rightarrow F$ **(Depends on part of the key** B**).**

Decomposition into 2NF Relations:

1. **Relation** $R1(A, D, E)$**: Includes** $A \rightarrow DE$.
    - **Attributes:** A, D, E.
    - **Key:** A.
2. **Relation** $R2(B, F)$**: Includes** $B \rightarrow F$.
    - **Attributes:** B, F.
    - **Key:** B.
3. **Relation** $R3(F, G, H)$**: Includes** $F \rightarrow GH$.
    - **Attributes:** F, G, H.
    - **Key:** F

4. **Relation $R4(D,I,J)$: Includes $D \rightarrow IJ$.**
    - **Attributes:** D,I,J.
    - **Key:** D.
5. **Relation $R5(AB,C)$: Includes $AB \rightarrow C$.**
    - **Attributes:** A,B,C.
    - **Key:** AB.

Step 3: Decompose into 3NF

3NF eliminates transitive dependencies, where a non-prime attribute depends on another non-prime attribute.

Steps for 3NF Decomposition:

- **Check each relation from 2NF:**
    1. $R_1(A,D,E)$: **No transitive dependency, already in 3NF.**
    2. $R_2(B,F)$: **No transitive dependency, already in 3NF.**
    3. $R_3(F,G,H)$: **No transitive dependency, already in 3NF.**
    4. $R_4(D,I,J)$: **No transitive dependency, already in 3NF.**
    5. $R_5(A,B,C)$: **No transitive dependency, already in 3NF.**

Final Decomposed Relations

The 3NF decomposition results in the following relations:

1. $R_1(A,D,E)$ **with** $A \rightarrow DE$.
2. $R_2(B,F)$ **with** $B \rightarrow F$.
3. $R_3(F,G,H)$ **with** $F \rightarrow GH$.
4. $R_4(D,I,J)$ **with** $D \rightarrow IJ$.
5. $R_5(A,B,C)$ **with** $AB \rightarrow C$.

Subscribe Multi Atoms & Multi Atoms Plus

What is Inclusion Dependency?

An Inclusion Dependency is a constraint that specifies that the values in one set of columns in a table (relation) must appear in another set of columns in another table (or the same table). It expresses a subset relationship between attributes.

Student Table

student_id	course_id
101	CS101
102	CS102

Course Table

course_id	course_name
CS101	Computer Basics
CS102	Data Structures

- The course_id column in Students must match a value in the course_id column in Courses.
- This is an Inclusion Dependency.

It ensures that every course a student takes (in the Students table) actually exists in the list of courses (in the Courses table).

- **Constraint:** $\pi_{course_id}(Student) \subseteq \pi_{course_id}(Course)$

Why Is It Important?

1. **Keeps Data Consistent**: It prevents errors, like assigning a student to a course that doesn't exist.
2. **Maintains Relationships**: It keeps the "connections" between tables intact.
3. **Avoids Data Loss:** If a course is removed, any student enrolled in that course can also be removed safely, keeping the database clean.

Lossless Join Decomposition

When we split a large table (relation R) into smaller tables (like $R1$ and $R2$), **lossless join decomposition** ensures that **no data is lost** when we combine (join) these smaller tables back to reconstruct the original table.

Why is it Important?

- If decomposition is **not lossless**, some data may be missing or duplicated when we try to reconstruct the original table.
- A lossless decomposition guarantees that the original data can always be obtained by a natural join operation on the decomposed tables.

Conditions for Lossless Join Decomposition

To check if the decomposition of R into $R1$ and $R2$ is lossless, the following conditions must hold:

1.Union of Attributes:

The combined attributes of $R1$ and $R2$ must equal all the attributes of R. Mathematically:

$$\text{Attributes}(R1) \cup \text{Attributes}(R2) = \text{Attributes}(R)$$

If this condition fails, the decomposition is not lossless.

2.Intersection of Attributes:

There must be at least one common attribute between $R1$ and $R2$.
Mathematically:

$$\text{Attributes}(R1) \cap \text{Attributes}(R2) \neq \Phi$$

If $R1$ and $R2$ have no common attributes, they can't be joined to reconstruct R.

3.Key Property of Common Attributes:

The common attribute(s) must form a key in at least one of the smaller tables ($R1$ or $R2$).
Mathematically:

$$\text{Attributes}(R1) \cap \text{Attributes}(R2) \rightarrow \text{Attributes}(R1) \quad \text{or} \quad \text{Attributes}(R1) \cap \text{Attributes}(R2) \rightarrow \text{Attributes}(R2)$$

This ensures that the common attribute(s) can uniquely identify tuples in at least one of the decomposed tables.

Aktu-2022-23

Q. Given the following set of FDs on schema R (V,W,X,Y,Z)

{Z→V, W→Y, XY→Z, V→WX}

State whether the following decomposition are loss-less-join decompositions or not.
(i) R1=(V,W,X) , R2=(V,Y,Z)
(ii) R1=(V,W,X), R2=(X,Y,Z)

Decomposition (i): $R1 = (V, W, X),\ R2 = (V, Y, Z)$

1. **Condition 1:**
   Union of attributes:

   $R1 \cup R2 = (V, W, X) \cup (V, Y, Z) = (V, W, X, Y, Z)$

   ✅ **Matches R.**

2. **Condition 2:**
   Intersection of attributes:

   $R1 \cap R2 = (V, W, X) \cap (V, Y, Z) = (V)$

   ✅ **Not empty.**

3. **Condition 3:**
   Common attribute V is a key:

   - **In $R1$:** $V \rightarrow WX$ ✅

   ✅ **Lossless join!**

Decomposition (ii): $R1 = (V, W, X)$, $R2 = (X, Y, Z)$

1. **Condition 1:**
   Union of attributes:

   ✅ **Matches** R.
2. **Condition 2:**
   Intersection of attributes:

   ✅ **Not empty.**
3. **Condition 3:**
   Common attribute X **is not a key in** $R1$ **or** $R2$**:**
   - X **alone doesn't determine all attributes in** $R1$ **or** $R2$.

   ❌ **Not lossless join!**

Final Answer

1. **Decomposition (i): Lossless join** ✅
2. **Decomposition (ii): Not lossless join** ❌

Multivalued Dependencies (MVDs)

- A **Multivalued Dependency (MVD)** exists in a relation when one attribute determines multiple independent values of another attribute. Unlike functional dependencies (FDs), where one attribute determines exactly one value of another, MVD allows multiple values.
- **Notation:**
  If $A \twoheadrightarrow B$, this means A determines all possible combinations of values for B, independent of other attributes.
- **Example:**
  Consider a relation **Student(StuID, Course, Hobby):**
  - A student can have **multiple courses**.
  - A student can have **multiple hobbies**.
  - These two attributes $Course$ and $Hobby$ are **independent** of each other.

  MVD: $StuID \rightarrow\!\!\rightarrow Course$ and $StuID \rightarrow\!\!\rightarrow Hobby$

Trivial vs Non-Trivial MVD:

- **Trivial MVD:** If X and Y overlap completely or their union equals the whole table.
- **Non-Trivial MVD:** If X and Y are separate and unrelated.

Example:

- Trivial: $Teacher \twoheadrightarrow Teacher, Subject$.
- Non-Trivial: $Teacher \twoheadrightarrow Subject$ (when Subject doesn't overlap with Teacher).

Join Dependency (JD):

- **What is JD?**
- - JD specifies that a relation RR can be decomposed into smaller relations $R_1, R_2, \ldots, R_n$ such that the original relation can be perfectly reconstructed (lossless join).
- **How is it denoted?**
  - $JD(R_1, R_2, \ldots, R_n)$: $R_1, R_2, \ldots, R_n$ are subsets of R.
- **When is JD Trivial?**
  - If any one of the relations $R_1, R_2, \ldots, R_n$ is equal to the entire relation R, the JD is trivial.

- **Example of JD:**
  - Consider $R(Faculty, Subject, Committee)$.
    - Decompose R into:
      - $R_1(Faculty, Subject)$ and $R_2(Faculty, Committee)$.
    - Using a **lossless join**, R can be reconstructed:
      - $R = \Pi_{Faculty, Subject}(R_1) \bowtie \Pi_{Faculty, Committee}(R_2)$.

4NF (Fourth Normal Form)

- A table is in 4NF if it is in BCNF (Boyce-Codd Normal Form) and does not have any non-trivial Multivalued Dependencies (MVDs).
- In simple terms, 4NF eliminates redundancy caused by independent multivalued facts.

Example:

Teacher	Subject	Committee
John	Math	Placement
John	Science	Placement
John	Math	Scholarship
John	Science	Scholarship

- **Problem:**
  - The **Subject** and **Committee** are independent of each other.
  - $Teacher \twoheadrightarrow Subject$ and $Teacher \twoheadrightarrow Committee$ are **independent multivalued dependencies**.

Decomposition into 4NF:

Teacher and Subject:

Teacher	Subject
John	Math
John	Science

Teacher and Committee:

Teacher	Committee
John	Placement
John	Scholarship

5NF (Fifth Normal Form)

- A table is in 5NF if it is in 4NF and cannot be decomposed further without losing data.
- 5NF deals with Join Dependencies (JD).

Example:

Student	Course	Teacher
Alice	Math	Mr. Smith
Alice	Science	Mrs. Johnson
Bob	Math	Mr. Smith
Bob	Science	Mrs. Johnson

- **Problem:**

  This table has a join dependency: The relationship between Student, Course, and Teacher can be split into three smaller relationships.

Decomposition into 5NF:

Student and Course:

Student	Course
Alice	Math
Alice	Science
Bob	Math
Bob	Science

Course and Teacher:

Course	Teacher
Math	Mr. Smith
Science	Mrs. Johnson

Student and Teacher:

Student	Teacher
Alice	Mr. Smith
Alice	Mrs. Johnson
Bob	Mr. Smith
Bob	Mrs. Johnson

Alternate Approaches to Database Design

Database design typically follows the normalization approach, but there are alternative methods that may be more appropriate depending on the application's requirements. These approaches focus on optimizing database performance, reducing redundancy, and ensuring data integrity, sometimes diverging from strict normalization principles.

1. Denormalization

- **What it is:** Combines multiple tables into one to make data retrieval faster.
- **When to use:** For read-heavy applications where performance is more important than reducing redundancy.
- **Example:** Instead of separate Customer and Orders tables, combine them into one table with all details.
- **Advantage:** Faster reads.
- **Disadvantage:** Data redundancy increases.

2. Schema-less Design (NoSQL)

- **What it is**: No fixed structure for the database; uses flexible formats like JSON.
- **When to use**: For unstructured or frequently changing data, like in social media apps.
- **Example**: Store a customer and their orders in a single document.
- **Advantage:** Flexible and scales easily.
- **Disadvantage:** Queries can become complex.

3. Agile Database Design

- **What it is:** Start with a simple design and add features as needed over time.
- **When to use:** In rapidly changing projects like startups.
- **Example:** Begin with just a Users table and add Orders or Addresses tables later.
- **Advantage:** Adapts to new requirements quickly.
- **Disadvantage:** May need rework later.

4. Graph-Based Design

- **What it is:** Stores data as nodes (items) and edges (relationships).
- **When to use:** For highly interconnected data like social networks.
- **Example:** A User node connected to Friends or Posts nodes.
- **Advantage:** Great for relationship-heavy queries.
- **Disadvantage:** Not ideal for standard tabular data.