The **Boyce-Codd Normal Form (BCNF)** is a stricter version of the Third Normal Form (3NF) in database normalization. A table is in BCNF if, for every functional dependency $A \rightarrow B$, A is a superkey (i.e., A uniquely identifies every row in the table).

Easiest Understandable Example of BCNF

Example 1: A Table Violating BCNF

Imagine a table `StudentCourse` with the following attributes:

StudentID	CourseID	Instructor
1	Math101	Dr. Smith
2	Math101	Dr. Smith
3	CS101	Dr. Lee

Functional Dependencies (FDs):

1. $\text{CourseID} \rightarrow \text{Instructor}$ (Each course is taught by one instructor.)
2. $\text{StudentID}, \text{CourseID} \rightarrow \text{Instructor}$ (Primary key: $\text{StudentID}, \text{CourseID}$.)

3.5NF

Problem:

- $\text{CourseID} \rightarrow \text{Instructor}$ violates BCNF because CourseID is not a superkey.

Steps to Convert to BCNF

To fix the violation, we split the table:

1. **First Table** (`CourseInstructor`):

CourseID	Instructor
Math101	Dr. Smith
CS101	Dr. Lee

   - This table captures the dependency $\text{CourseID} \rightarrow \text{Instructor}$.

2. **Second Table** (`StudentCourse`):

StudentID	CourseID
1	Math101
2	Math101
3	CS101

   - This table ensures that $\text{StudentID, CourseID}$ is the key.

Final Tables in BCNF

1. **CourseInstructor** satisfies BCNF because CourseID is a superkey.
2. **StudentCourse** satisfies BCNF because $\text{StudentID, CourseID}$ is a superkey.

Multivalued Dependency (MVD):

A **multivalued dependency** occurs when one attribute in a table determines a set of values for another attribute, independently of other attributes. It extends the concept of functional dependency but applies to sets of values.

Formal Definition:

If A and B are attributes in a table, and $A \to\to B$ is a multivalued dependency, then:

- For each value of A, there exists a set of values for B, independent of other attributes in the table.

In simpler terms:

- Knowing A tells you all possible values of B, regardless of any other attribute in the table.

Example of Multivalued Dependency:

Table: `StudentActivities`

StudentID	Sport	Club
1	Football	Music Club
1	Cricket	Music Club
1	Football	Drama Club
1	Cricket	Drama Club

Multivalued Dependency:

- $\text{StudentID} \to\to \text{Sport}$: The sports a student participates in are independent of the clubs they are part of.
- $\text{StudentID} \to\to \text{Club}$: The clubs a student joins are independent of the sports they play.

4th Normal Form (4NF):

A table is in **4th Normal Form (4NF)** if:

1. It is in **Boyce-Codd Normal Form (BCNF).**
2. It has no **non-trivial multivalued dependencies.**

Non-Trivial MVD:

An MVD $A \rightarrow\rightarrow B$ is non-trivial if:

- B is not a subset of A, and
- $A \cup B$ does not form all attributes in the table.

Steps to Achieve 4NF:

1. Identify the multivalued dependencies in the table.
2. Decompose the table into smaller tables to isolate the multivalued dependencies.

Decomposing the Example Table into 4NF:

Original Table:

StudentID	Sport	Club
1	Football	Music Club
1	Cricket	Music Club
1	Football	Drama Club
1	Cricket	Drama Club

Step 1: Decompose into Two Tables

1. Table 1: StudentSports

StudentID	Sport
1	Football
1	Cricket

2. Table 2: StudentClubs

StudentID	Club
1	Music Club
1	Drama Club

Now, each table independently satisfies 4NF because:

- In `StudentSports`, there is no multivalued dependency; $StudentID$ determines $Sport$.
- In `StudentClubs`, there is no multivalued dependency; $StudentID$ determines $Club$.

Scenario:

Imagine a table `EmployeeProjects` where an employee can work on multiple projects **and** also have multiple skills. The projects and skills are independent of each other.

Original Table: `EmployeeProjects`

EmployeeID	Project	Skill
1	ProjectA	Java
1	ProjectB	Java
1	ProjectA	Python
1	ProjectB	Python

SOLUTION ????

Join Dependency and 5th Normal Form (5NF)

Join Dependency (JD):

A **join dependency** occurs when a table can be decomposed into two or more smaller tables and then rejoined to recreate the original table **without any loss of information.** This generalizes the concept of functional dependencies.

For a table R, a join dependency $*(R_1, R_2, \ldots, R_n)*$ exists if R can be decomposed into $R_1, R_2, \ldots, R_n$, and reconstructing R by joining $R_1, R_2, \ldots, R_n$ is lossless.

5th Normal Form (5NF):

A table is in **5NF** (or **Project-Join Normal Form**) if:

1. It is in **4th Normal Form (4NF)**.
2. It has no **non-trivial join dependencies**.

Non-Trivial Join Dependency:

A join dependency is **non-trivial** if the table cannot be recreated without decomposing into multiple tables.

The goal of 5NF is to eliminate redundancy caused by **join dependencies** while ensuring no loss of data.

Example of Join Dependency and 5NF

Original Table: `EmployeeProjectsRoles`

EmployeeID	Project	Role
1	ProjectA	Developer
1	ProjectA	Tester
1	ProjectB	Developer
2	ProjectA	Tester

Join Dependencies:

- $(\text{EmployeeID}, \text{Project})$
- $(\text{EmployeeID}, \text{Role})$
- $(\text{Project}, \text{Role})$

Problem:

The redundancy exists because the relationships between `EmployeeID`, `Project`, and `Role` are independent:

1. An employee can be associated with multiple projects.
2. An employee can have multiple roles.
3. A project can involve multiple roles.

Decomposing into 5NF:

To remove redundancy and satisfy 5NF, decompose the table into three smaller tables:

1. **Table 1: EmployeeProjects**

EmployeeID	Project
1	ProjectA
1	ProjectB
2	ProjectA

2. **Table 2: EmployeeRoles**

EmployeeID	Role
1	Developer
1	Tester
2	Tester

3. **Table 3: ProjectRoles**

Project	Role
ProjectA	Developer
ProjectA	Tester
ProjectB	Developer

Final Tables in 5NF:

- Each table contains information about a single relationship.
- Joining all three tables recreates the original data without redundancy.
- This satisfies 5NF because no non-trivial join dependencies remain.

Inference Rules for Functional Dependencies

- **Armstrong's axioms are used to conclude functional dependencies on a relational database.**
- The inference rule is a type of assertion. It can apply to a set of FD(functional dependency) to derive other FD.
- Using the inference rule, we can derive additional functional dependency from the initial set.

<u>The Functional dependency has 6 types of inference rule:</u>

1. Reflexive Rule (IR_1)

2. Augmentation Rule (IR_2)

3. Transitive Rule (IR_3)

4. Union Rule (IR_4)

5. Decomposition Rule (IR_5)

6. Pseudo transitive Rule (IR6)

Armstrong's Axioms

1. **Reflexivity Rule**

   If $Y \subseteq X$, then $X \to Y$.

   (A set of attributes determines its own subset.)

   Example:

   If $X = \{A, B, C\}$, then $\{A, B, C\} \to \{A, B\}$.

2. **Augmentation Rule**

   If $X \to Y$, then $XZ \to YZ$ for any Z.

   (Adding attributes to both sides of a functional dependency preserves validity.)

   Example:

   If $A \to B$, then $AC \to BC$.

3. **Transitivity Rule**

   If $X \to Y$ and $Y \to Z$, then $X \to Z$.

   (A chain of dependencies can be combined into a single dependency.)

   Example:

   If $A \to B$ and $B \to C$, then $A \to C$.

Additional Inference Rules (Derived from Armstrong's Axioms):

4. **Union Rule**

   If $X \to Y$ and $X \to Z$, then $X \to YZ$.

   (A single determinant can determine multiple attributes.)

   Example:

   If $A \to B$ and $A \to C$, then $A \to BC$.

5. **Decomposition Rule**

   If $X \to YZ$, then $X \to Y$ and $X \to Z$.

   (A determinant of a composite attribute can determine each component individually.)

   Example:

   If $A \to BC$, then $A \to B$ and $A \to C$.

6. **Pseudotransitivity Rule**

   If $X \to Y$ and $WY \to Z$, then $WX \to Z$.

   (Combining dependencies with overlapping attributes.)

   Example:

   If $A \to B$ and $BC \to D$, then $AC \to D$.

Closure of a Set of Attributes:

- The closure of a set of attributes X, denoted X^+, is the set of all attributes that can be functionally determined by X using the given FDs and inference rules.

Steps to Find Attribute Closure:

1. Start with $X^+ = X$.
2. For each FD $A \to B$:
   - If $A \subseteq X^+$, then add B to X^+.
3. Repeat until no more attributes can be added to X^+.

Example 1

Given Table Attributes:

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

Functional Dependencies (FDs):

1. $A \to B$
2. $B \to C$
3. $AC \to D$
4. $D \to E$

Find Closure of $X = \{A\}$:

1. **Start:**
   $X^+ = \{A\}$.
2. **Apply FD 1 ($A \to B$):**
   Since $A \subseteq X^+$, add B:
   $X^+ = \{A, B\}$.
3. **Apply FD 2 ($B \to C$):**
   Since $B \subseteq X^+$, add C:
   $X^+ = \{A, B, C\}$.
4. **Apply FD 3 ($AC \to D$):**
   Since $A, C \subseteq X^+$, add D:
   $X^+ = \{A, B, C, D\}$.
5. **Apply FD 4 ($D \to E$):**
   Since $D \subseteq X^+$, add E:
   $X^+ = \{A, B, C, D, E\}$.
6. **No More FDs Apply:**
   X^+ is now stable: $\{A, B, C, D, E\}$.

REFER SLIDE 60 , 61 FOR MORE EXAMPLES ON CLOSURE

Equivalence and Minimal Cover

1. Equivalence of Functional Dependencies (FDs):

Two sets of FDs F and G are **equivalent** if:

1. Every FD in F can be inferred from G, and
2. Every FD in G can be inferred from F.

To check equivalence, compute the **closure of attributes** for both sets of FDs and verify they produce the same results.

2. Minimal Cover (Canonical Cover):

A **minimal cover** for a set of FDs is a simplified version that is:

- Equivalent to the original FDs.
- Contains no redundant attributes or FDs.
- Each FD has a single attribute on the right-hand side.

Steps to Find Minimal Cover:

1. Split FDs:

   Convert each FD into a form where the right-hand side has only one attribute.

   For example:

   $A \rightarrow BC$ becomes $A \rightarrow B$ and $A \rightarrow C$.

2. Remove Redundant Attributes from the Left-Hand Side:

   For each FD $A \rightarrow B$, check if removing an attribute from A still implies B. If yes, remove it.

3. Eliminate Redundant FDs:

   Check if removing a dependency $A \rightarrow B$ still preserves equivalence. If yes, remove it.

Example of Minimal Cover:

Given FDs:

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

Step 1: Split into Single Attributes on RHS:

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

Step 2: Remove Redundant Attributes from LHS:

No attributes can be removed from the LHS.

Step 3: Eliminate Redundant FDs:

- Check $A \rightarrow C$: It can be inferred from $A \rightarrow B$ and $B \rightarrow C$. Remove $A \rightarrow C$.

Minimal Cover:

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

Consider the following decomposition for the relation schema:
R = {A, B, C, D, E, F, G, H, I,}. Determine whether the decomposition D has (i) the dependency preservation property and (ii) the lossless join property, with respect to
F = { AB -> C , A -> DE, B -> F, F -> GH, D -> IJ}.
D = {R1, R2, R3, R4, R5};
R1 = {A, B, C, D},
R2 = {D, E},
R3 = {B, F}, R4 = {F, G, H}, R5 = {D, I, J}.

Properties of Relational Decomposition

Relational decomposition splits a relation into smaller relations while preserving key properties.

Key Properties:

1. **Lossless Join:** A decomposition is **lossless** if it allows the original relation to be reconstructed without loss of data.
   Condition: For a decomposition R_1 and R_2, $R_1 \cap R_2$ must be a superkey in at least one of them.
2. **Dependency Preservation:** A decomposition is **dependency-preserving** if all functional dependencies are enforceable on the decomposed relations without requiring a join.
3. **Redundancy Elimination:** Decomposition should minimize redundancy in data storage.

Algorithms for Relational Database Schema Design

1. Lossless Decomposition Algorithm:

Ensures that decomposition satisfies the **lossless join property**.

Steps:

1. Identify the candidate keys of the relation.
2. Check if the intersection of decomposed relations is a superkey in at least one of them.
3. Decompose relations iteratively to meet the lossless join condition.

3. 3NF Decomposition Algorithm:

Produces a schema in **3rd Normal Form (3NF)** while ensuring **lossless join** and **dependency preservation**.

Steps:

1. Identify a minimal cover of FDs.
2. Create a relation for each FD $A \rightarrow B$ in the minimal cover.
3. Ensure that each relation contains a candidate key of the original relation.
4. Verify that the decomposition satisfies the lossless join and dependency preservation properties.

4. BCNF Decomposition Algorithm:

Produces a schema in **Boyce-Codd Normal Form (BCNF)**.

Steps:

1. Check if the given schema is in BCNF.
2. If not, decompose the relation into smaller relations by removing violations of BCNF.
3. Ensure each decomposed relation is in BCNF.
4. Verify the lossless join property for each decomposition.