**Normalization Concepts**

### **Anomalies in Database Design**

Anomalies in database design refer to unexpected issues that arise when performing common operations such as inserting, updating, or deleting data. These anomalies occur due to **data redundancy** and **poor table structure** and lead to inconsistent or incomplete data. There are three main types of anomalies:

#### **1. Insertion Anomaly**

An insertion anomaly occurs when inserting a new record requires unnecessary additional data to be provided.

* **Example:** In a university database where student data, course data, and enrollment data are stored in the same table, inserting a new course (say, Course C4) would require student and enrollment data, even if there are no students enrolled yet. This is because the table’s primary key may depend on a combination of student and course details, forcing the user to enter redundant or unnecessary information.
* **Problem:** This can lead to inefficiency, where data that should not be necessary for an operation is required, leading to partial or incorrect data entries.

#### **2. Update Anomaly**

An update anomaly occurs when modifying a single fact in the database requires updating multiple rows.

* **Example:** If the name of a course (say, Course C1) is stored in several rows of a table where both student and course data are mixed, updating the course name would require changing it in every row where it appears. If a course has 50 students enrolled, you need to update 50 rows, increasing the risk of **data inconsistencies** if any rows are missed.
* **Problem:** This leads to data duplication, and if all rows are not updated consistently, the database will contain conflicting information.

#### **3. Deletion Anomaly**

A deletion anomaly occurs when deleting a record inadvertently removes other related but necessary information.

* **Example:** In the same university database, if a student (S1) drops a course, and that student was the only person enrolled in the course, deleting their enrollment may inadvertently delete all information about the course itself (e.g., Course C1 and Offering O1), leading to the **loss of data** that should have been retained.
* **Problem:** Deleting one piece of data unintentionally removes other valuable data, leading to incomplete information.

**Conclusion on Anomalies:** These anomalies occur because of **data redundancy** in improperly structured tables. The goal of normalization is to eliminate such redundancies by organizing data in a way that avoids these anomalies.

### **Functional Dependency**

**Functional dependency** (FD) is a fundamental concept in relational database theory. It describes the relationship between two sets of attributes in a table. In simple terms, a functional dependency exists when the value of one attribute (or a set of attributes) uniquely determines the value of another attribute.

#### **1. Definition of Functional Dependency**

If attribute A determines attribute B, we say that **B is functionally dependent on A** (denoted as A → B). This means that for each unique value of A, there is only one corresponding value of B.

* **Example:** In a student database, if StudentID uniquely determines the StudentName, then there is a functional dependency StudentID → StudentName. Each student has one unique ID, and for each ID, there is a corresponding student name.

#### **2. Types of Functional Dependencies**

There are different types of functional dependencies that are useful for normalization:

* **Full Functional Dependency:** Attribute B is fully dependent on A if B is dependent on all parts of a composite key.
  + **Example:** In a table with a composite primary key (StudentID, CourseID), the Grade attribute is fully functionally dependent on the combination of both StudentID and CourseID.
* **Partial Functional Dependency:** Attribute B is partially dependent on A if it is dependent on only part of a composite key.
  + **Example:** If StudentName is dependent only on StudentID (part of a composite key), then we have a partial dependency, which is not desirable in normalized forms.
* **Transitive Functional Dependency:** Attribute B is transitively dependent on A if B is dependent on C, and C is dependent on A (i.e., A → C → B).
  + **Example:** If StudentID → DepartmentID and DepartmentID → DepartmentName, then StudentID transitively determines DepartmentName.

#### **3. Importance of Functional Dependencies in Normalization**

Functional dependencies are critical for the process of normalization. They help in identifying the **relationships between attributes** and guide the breaking down of tables into smaller, related tables to avoid redundancy and anomalies.

### **Normal Forms:** Normalization is the process of structuring a database to reduce redundancy and improve data integrity. A **normal form** defines the level of refinement a table must achieve to reduce redundancy and avoid anomalies. Each successive normal form builds on the previous one and imposes stricter rules for data organization.

#### **1. First Normal Form (1NF)**

A table is in **First Normal Form (1NF)** if:

* All attributes contain **atomic values** (indivisible units of data).
* There are no **repeating groups** of data.

**Example:** A student table should not have multiple columns for courses (Course1, Course2, etc.). Instead, each course should be a separate row with a student ID and course ID.

**Goal of 1NF:** Eliminate duplicate entries and ensure that the data is stored in individual cells.

#### **2. Second Normal Form (2NF)**

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

* It is already in 1NF.
* There are **no partial dependencies**, meaning every non-key attribute is fully dependent on the primary key.

**Example:** In a table with a composite key (StudentID, CourseID), attributes like StudentName should not depend on only StudentID. These partial dependencies need to be moved to a separate table.

**Goal of 2NF:** Ensure that non-key attributes depend only on the entire primary key and not on a part of it.

#### **3. Third Normal Form (3NF)**

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

* It is already in 2NF.
* There are **no transitive dependencies**, meaning non-key attributes do not depend on other non-key attributes.

**Example:** If StudentID → DepartmentID and DepartmentID → DepartmentName, the DepartmentName should be placed in a separate table to remove the transitive dependency.

**Goal of 3NF:** Ensure that non-key attributes are not indirectly dependent on the primary key.

#### **4. Boyce-Codd Normal Form (BCNF)**

A table is in **Boyce-Codd Normal Form (BCNF)** if:

* It is already in 3NF.
* Every determinant (i.e., an attribute that determines another attribute) is a **candidate key**.

**Example:** If both ProfessorID and CourseID determine the classroom, but CourseID alone can also determine the classroom, this violates BCNF, as CourseID is not a candidate key in this case.

**Goal of BCNF:** Strengthen the rules for functional dependencies, ensuring that any attribute determining other attributes is a candidate key.

**1.Modification Anomalies:**

**Motivation for Eliminating Unwanted Redundancy through Normalization**

In any database, especially large-scale systems like a university database, unwanted redundancy leads to several issues. The normalization process is crucial for eliminating this redundancy, thereby avoiding the associated problems. The goal of normalization is to ensure efficient data handling, prevent data anomalies, and ensure that each fact is stored in only one place. This leads to a more reliable and efficient database structure.

### **Modification Anomalies**

Modification anomalies occur when data redundancy in a database leads to unexpected side effects during typical data operations (insert, update, delete). These anomalies create inefficiencies and data inconsistencies. The three types of modification anomalies are as follows:

#### **1. Insertion Anomaly**

An **insertion anomaly** arises when extra data is required to insert a new record, even if it is irrelevant to the main operation being performed.

* **Example:** In a university database, inserting a new course (e.g., Course C4) requires information about a student enrolled in the course. This is because the primary key in the current table is a combination of Student Number and Offer Number. Without this student and offer data, the new course cannot be inserted, creating an unnecessary dependency on unrelated information.
* **Problem:** This forces users to enter irrelevant or redundant information just to insert a new record, making the system inefficient.

#### **2. Update Anomaly**

An **update anomaly** occurs when changing a single piece of information requires updating multiple rows in the table.

* **Example:** Suppose you want to update the description of a course (e.g., Course C2). If 10 students are enrolled in Course C2, you must update the description in 10 different rows, leading to redundancy and a high risk of inconsistency if any rows are missed.
* **Problem:** Requiring updates in multiple places increases the likelihood of errors and can cause inconsistent data if all rows aren’t updated correctly.

#### **3. Deletion Anomaly**

A **deletion anomaly** occurs when deleting one piece of information results in the unintended loss of other important data.

* **Example:** If a student (e.g., Student S1) enrolled in Offering O1 is deleted, the data related to the offering (e.g., Offering O1 and Course C1) may also be deleted, even though it’s still needed for other students or purposes.
* **Problem:** This leads to unintended data loss, which can compromise the integrity and completeness of the database.

### **Importance of Avoiding Modification Anomalies**

Good database design aims to avoid modification anomalies by eliminating redundant data. The key principle is to store each piece of information in only one place. Doing so prevents unwanted side effects during insert, update, or delete operations. A well-designed database ensures that users can modify rows without creating unnecessary dependencies or risking data integrity.

For example, in a well-structured university database, a user should be able to insert a new course without needing to insert data about students or course offerings. Likewise, when a student is deleted from the database (e.g., upon graduation), the associated course information should not be inadvertently lost.

### **Normalization Process**

The **normalization process** involves restructuring a database to eliminate redundant data and avoid modification anomalies. This process ensures that every fact is stored in only one place, and there is a clear relationship between the data. Normalization breaks down large tables into smaller, related tables to reduce redundancy and improve data integrity.

### **Impact of Poor Database Design**

A poorly designed database with excessive redundancy can lead to numerous modification anomalies. For example, imagine a university database where a single table stores student, course, offering, and enrollment data, and the primary key is a combination of Student Number and Offer Number. In such a design, the following issues can occur:

* **Redundant Data:** Multiple rows contain repetitive data about students, courses, and offerings.
* **Complex Operations:** Simple operations like updating a course description require changes in multiple places, increasing the risk of errors.
* **Inefficiency:** Users must provide unnecessary information to insert or update data, leading to inefficiency.

### **Examples of Modification Anomalies in Poorly Designed Tables**

Let’s revisit examples of modification anomalies based on a university database with a single table where Student Number and Offer Number form the primary key.

#### **1. Insertion Anomaly Example**

To insert a new course (Course C4), it is necessary to provide a student number and offer number, even if no students have enrolled in the course yet. This is because the table’s primary key relies on both Student Number and Offer Number, meaning no row can exist with null values in either of these columns.

#### **2. Update Anomaly Example**

To change the description of a course (Course C2), two rows must be updated. If 10 students are enrolled in Course C2, then 10 rows must be changed. This creates redundancy and increases the risk of inconsistent data if one or more rows are not properly updated.

#### **3. Deletion Anomaly Example**

If a student (Student S1) enrolled in Offering O1 is deleted, information about Offering O1 and Course C1 may also be lost. This results in losing important data about the course offering, even though other students may still be enrolled in that course.

### **Solutions to Avoid Modification Anomalies**

There are two common approaches to avoid or minimize modification anomalies:

1. **Using Default Primary Keys:** Some users try to avoid insertion anomalies by using default primary keys or dummy values. However, this solution is temporary and does not eliminate the root problem.
2. **Code Workarounds:** Database analysts may write custom code to prevent the loss of important data. While this approach can work, it is inefficient and leads to increased complexity in database operations.

### **Better Solution: Normalization**

A better and more sustainable solution to prevent modification anomalies is **normalization**. Normalization involves restructuring the database to reduce redundancy, ensure data consistency, and eliminate modification anomalies. By separating the data into different tables based on relationships and dependencies, modification anomalies can be avoided.

For example, rather than storing all data (students, courses, enrollments) in a single table, normalization breaks this data into several smaller, logically related tables. This ensures that each table contains only the information directly related to its primary key, thereby preventing unnecessary dependencies between unrelated data.

### **Transaction Processing vs. Business Intelligence Processing**

At the start of this lesson, the question was raised about which type of processing environment has more motivation to avoid modification anomalies. The answer is that **transaction processing** is more sensitive to modification anomalies than **business intelligence processing**.

* **Transaction Processing:** This type of processing, which involves frequent updates, inserts, and deletes, strongly prefers a well-normalized database design (e.g., the five-table design discussed in earlier modules). A well-structured database minimizes modification anomalies and makes transaction processing more efficient and reliable.
* **Business Intelligence Processing:** Business intelligence, which focuses on querying and reporting, may prefer a denormalized design with fewer tables. A single table design is easier to query and may be sufficient for analysis purposes, even if it introduces some redundancy.

**2. Functional Dependencies:**

## **Introduction to Functional Dependencies**

Functional dependencies (FDs) are foundational concepts in database normalization that help to define relationships between attributes in a table. Understanding FDs is essential for ensuring that database designs are efficient and free from unwanted redundancy.

## **Definition of Functional Dependency**

A functional dependency is a constraint that specifies a relationship between two sets of attributes in a table. It can be expressed as X→YX \rightarrow YX→Y, which means that for each unique value of XXX, there is exactly one corresponding value of YYY. This relationship is similar to a mathematical function, where a single input (X) produces a single output (Y). For example, in a university database, if a student's number (S1) determines their city (C1), we express this as:

* **Student Number** (S1) → **Student City** (C1)

This implies that for each student number, there is at most one associated city.

## **Analogy to Unique Constraints**

Functional dependencies can be thought of as unique constraints within the context of relational databases. Specifically, if two attributes XXX and YYY are combined in a table, and XXX determines YYY, then XXX must have unique values. For instance:

* If the attribute **Offer Number** uniquely identifies the **Off Year**, then we can say:
  + **Offer Number** (O1) → **Off Year** (2020)

Here, the offer number must be unique in the table to ensure a proper functional dependency exists.

## **Importance of Specifying Functional Dependencies**

The specification of functional dependencies is crucial as it drives the normalization process. Unlike primary key and foreign key constraints, the assertion of functional dependencies cannot be automated. It requires a deep understanding of business requirements and rules. These constraints dictate what conditions should generally be true in a well-structured database.

## **The Structure of Functional Dependencies**

Functional dependencies can be organized and represented in two primary ways: lists and diagrams.

### **Listing Functional Dependencies**

A structured list organizes functional dependencies by their left-hand sides (determinants). For example:

* **Student Number** determines **Student Class**
* **Offer Number** determines **Course Number**

By organizing these dependencies, database designers can easily identify which attributes should be grouped together in a table to eliminate redundancy.

### **Functional Dependency Diagrams**

Functional dependency diagrams visually represent the relationships between attributes. In such diagrams, lines connect determinants to their dependent attributes. While diagrams can illustrate complex relationships, they may become cumbersome with many dependencies, making lists often more manageable.

## **Falsifying Functional Dependencies**

While a functional dependency can be asserted based on understanding and analysis, it cannot be proven solely by examining rows of a table. However, it can be falsified through example rows.

### **What Does Falsification Mean?**

Falsifying a functional dependency means demonstrating that the proposed relationship does not hold true across the data. This involves finding instances where the same value of XXX corresponds to multiple values of YYY.

### **Example of Falsification**

For instance, consider the functional dependency:

* **Student Number** determines **Offer Number**

To falsify this dependency, we look for two rows in a sample table where the student number is the same but the offer numbers differ. If we find:

* Row 1: **Student Number** S1 → **Offer Number** O1
* Row 2: **Student Number** S1 → **Offer Number** O2

This falsifies the functional dependency, indicating that **Student Number** does not uniquely determine **Offer Number**.

**Key Points**

1. **Functional Dependency Definition**: A relationship where one attribute set determines another.
2. **Unique Constraint Analogy**: Functional dependencies act similarly to unique constraints in ensuring uniqueness in data.
3. **Specification Importance**: Specifying functional dependencies is essential for effective normalization and cannot be automated.
4. **Organization of FDs**: Functional dependencies can be organized into lists or diagrams for clarity.
5. **Falsification Concept**: Falsifying functional dependencies demonstrates potential inaccuracies in assumed relationships.
6. **Practical Application**: Falsification serves as a communication tool with users to validate functional dependencies against actual data.

**3. Normal Forms:**

## **Introduction to Normalization**

Normalization is a fundamental concept in database design aimed at eliminating unwanted redundancy in a relational database schema. This lesson will focus on Boyce-Codd Normal Form (BCNF), a specific type of normal form that addresses functional dependencies in relational databases. The lesson begins with a trivia question: **Who is considered the father of the relational data model?** The answer is **Dr. Ted Codd**, whose pioneering work laid the foundation for relational databases.

## **Normalization Process Overview**

Normalization is a structured process that removes redundancy by organizing data within a database. The key steps in this process include:

1. **Input**: The process begins with a list of functional dependencies and unique columns in the table design.
2. **Detecting Violations**: The next step involves identifying any violations of the allowable patterns of functional dependencies. If a violation is detected, the original table is split into smaller, more normalized tables.
3. **Revised Design**: The new tables should adhere to the allowable patterns of functional dependencies specified by the normal form.

### **Importance of Functional Dependencies**

Functional dependencies are crucial to the normalization process as they define the relationships between different attributes in a table. A functional dependency is a relationship where one attribute uniquely determines another attribute. For successful normalization, the functional dependency list must be **complete and minimal**.

* **Complete Functional Dependencies**: This means all necessary dependencies must be included.
* **Minimal Functional Dependencies**: This ensures that no redundant dependencies are present.

### **Key Points About Normalization**

1. **Table Splitting**: Normalization splits a table into smaller tables when violations of the normal form are found. This process helps maintain data integrity and reduces redundancy.
2. **Functional Dependency Completeness**: A complete and minimal list of functional dependencies is essential for effective normalization, although the derivation of unique columns from these dependencies is not covered in this course.
3. **Hierarchy of Normal Forms**: Several normal forms have been developed to eliminate redundancies in database design. Each normal form introduces stricter rules regarding functional dependencies. The progression is as follows:
   * **First Normal Form (1NF)**: The foundation of normalization.
   * **Second Normal Form (2NF)**: Builds upon 1NF by addressing partial dependencies.
   * **Third Normal Form (3NF)**: Further refines the structure by eliminating transitive dependencies.
   * **Boyce-Codd Normal Form (BCNF)**: A stricter version of 3NF that ensures every determinant is unique in the table.

## **Boyce-Codd Normal Form (BCNF)**

BCNF is considered one of the most critical rules in normalization practice. It is designed to address specific kinds of redundancy and ensure data integrity within relational databases.

### **Definition of BCNF**

BCNF requires that every determinant in a functional dependency must be unique in the table. A functional dependency violates BCNF if the determinant is not unique.

* **Determinant**: The attribute or combination of attributes on the left-hand side of a functional dependency that uniquely determines another attribute on the right-hand side.

### **Examples of BCNF Violations**

Using a sample table from previous lessons, the **Big University Database**, several functional dependencies were identified, many of which violated BCNF. The primary key of this table is a combination of **student number** and **offer number**.

1. **Unique Columns**: The table also has a unique combination of **student email** and **offer number**.
2. **Identifying Violations**: The initial functional dependencies included several that did not have unique determinants, indicating violations of BCNF.

### **Procedure for Refining Table Design**

The lesson presents a simplified procedure to ensure that a table design complies with BCNF:

1. **Organizing Functional Dependencies**: The first step is to group functional dependencies by their determinants. This organization helps clarify violations of BCNF.
2. **Defining Tables**: Each functional dependency group becomes a table, with the determinant serving as the primary key. Foreign keys should also be added to relate tables appropriately.
3. **Merging Tables**: To avoid excessive table splitting, tables may be merged if one contains a subset of columns from another. This step helps streamline the database structure while maintaining compliance with BCNF.

### **Merging Tables Example**

When two columns mutually determine each other (e.g., **student number** and **student email**), the normalization process creates two separate tables. In such cases, these tables should be merged into one, ensuring that all determinants remain unique in the merged table.

## **Importance of Normalization in Business Intelligence vs. Transaction Processing**

### **Normalization Overview**

Normalization is the process of organizing data within a database to reduce redundancy and improve data integrity. In transaction processing, it is crucial to minimize modification anomalies to maintain accurate and efficient data operations. However, in business intelligence processing, which emphasizes data reporting and analysis, the need for strict normalization is often less critical.

### **Key Differences**

1. **Transaction Processing**:
   * Involves frequent updates, inserts, and deletions.
   * Aims to avoid modification anomalies to ensure data consistency during transactions.
   * Generally requires a normalized database structure to maintain accuracy and reliability.
2. **Business Intelligence Processing**:
   * Focuses on data analysis and reporting rather than frequent modifications.
   * Avoidance of modification anomalies becomes a secondary concern.
   * May utilize denormalized data structures to enhance query performance, as the reporting environment does not rely heavily on constant data updates.

## **Reflective Objectives of Normalization**

### **Role of Normalization**

Normalization serves as both a refinement and an initial design tool within the database development process. Understanding its role helps database designers make informed decisions about how to structure data effectively.

### **Importance of Normalization**

* **Refinement Tool**:
  + After creating an Entity-Relationship Diagram (ERD), normalization techniques can be applied to analyze tables for redundancies.
  + This step helps ensure that the database schema is efficient and minimizes data anomalies.
* **Initial Design Tool**:
  + When starting with normalization, designers identify functional dependencies directly, applying procedures like BCNF (Boyce-Codd Normal Form) to create an effective data structure.
  + Initial normalization ensures a strong foundation for the database before further refinement.

## **Practical Application of Normalization Techniques**

### **Conceptual Data Modeling**

Normalization can be applied effectively during conceptual data modeling. This involves using the ERD to intuitively group related attributes and identify potential redundancies without extensive formalization of functional dependencies.

### **Functional Dependencies and Keys**

* Identifying functional dependencies is essential for understanding how attributes relate to primary keys in tables. The designer must focus on ensuring that primary keys determine other columns effectively.
* Recognizing and documenting functional dependencies prevents the oversight of unique columns and redundant data, leading to a more structured database.

## **Balancing Normalization and Query Performance**

### **Avoiding Modification Anomalies**

While avoiding modification anomalies is a primary goal of normalization, it is essential to balance this with the need for efficient query performance, particularly in business intelligence environments.

### **Denormalization**

Denormalization is the purposeful violation of normal forms, typically BCNF. It can simplify table designs for ease of querying while increasing complexity when modifications are required.

#### **Reasons for Denormalization:**

1. **Functional Dependencies**:
   * Some functional dependencies may not significantly impact the overall data structure and can be ignored in certain contexts.
   * Ignoring non-critical dependencies can lead to more straightforward table designs without major risks of anomalies.
2. **Performance Improvements**:
   * In many scenarios, denormalization can enhance performance by reducing the number of join operations needed during query execution.
   * In business intelligence, where query performance is a top priority, denormalization may be favored over strict normalization.

## **Case Study: Zip Code Functional Dependency**

### **Practical Complexity of Functional Dependencies**

An example provided is the functional dependency where a zip code determines a city and state. However, this dependency can be complex due to the nature of zip codes, especially in sparsely populated areas where boundaries may cross city and state lines.

### **Implications of Ignoring Dependencies**

* If a database does not require maintaining independent zip code tables, these dependencies can be safely ignored.
* However, in cases where sales tax calculations or other operations depend on accurate zip code data, it becomes essential to maintain these relationships to avoid potential discrepancies.

**4. Normalization Problems:**

## **Modification Anomalies**

Modification anomalies are issues that arise during the insertion, updating, or deletion of data in a database. These anomalies can lead to inconsistencies and inefficiencies. There are three primary types of modification anomalies:

### **1. Insertion Anomaly**

* **Definition**: This anomaly occurs when certain data cannot be added to a table without the presence of other data.
* **Example**: In the big university database table, if a new student (S3) wishes to enroll, they cannot be inserted without specifying an offer number. This is due to the primary key being a combination of student number and offer number.

### **2. Update Anomaly**

* **Definition**: This anomaly occurs when a single data change requires multiple rows to be updated.
* **Example**: Changing the city of student S1 from Seattle to Hong Kong requires updating multiple rows if S1 is enrolled in 10 different courses. This means 10 rows must be altered, increasing the risk of errors.

### **3. Deletion Anomaly**

* **Definition**: This anomaly occurs when the deletion of a data entry inadvertently removes other important data.
* **Example**: If the enrollment record for student S2 in offer O3 is deleted, it may also remove course C3 and offering O3 from the database, leading to loss of critical information.

## **Functional Dependency Falsification**

Functional dependencies express relationships between data in a database, helping to establish rules for data organization. Falsification of functional dependencies indicates redundancy or inconsistency.

### **Identifying Functional Dependencies**

In the context of the big university database, possible functional dependencies where "student city" acts as the determinant include:

1. **Student City → Offer Number**
2. **Student City → Off Term**
3. **Student City → Student Number**

### **Falsification Examples**

Falsification occurs when the same determinant leads to different values in the dependent column.

* **Example 1**: For the functional dependency Student City → Offer Number, if two students from Seattle (same city) are linked to different offer numbers (O1 and O2), this falsifies the dependency.
* **Example 2**: For the dependency Student City → Off Term, if two students from Bothell have different off terms (spring vs. fall), this indicates a violation.
* **Non-Falsified Dependency**: The dependency Student Number → Student City may not be falsified if two rows share the same student number, confirming that the relationship is valid.

## **Conversion and Normalization**

The final problem in this lesson combines the conversion from an ERD to a table design with normalization practices.

### **Steps in Conversion**

1. **Entity Type Rule**: Define tables for each entity (e.g., students, interviewers, interviews).
2. **One-to-Many Relationship Rule**: Establish foreign keys based on relationships (e.g., linking interviewers to interviews).
3. **Identifying Relationship Rule**: Recognize primary key components in composite keys.

### **Normalization Process**

* **Functional Dependencies**: While primary keys imply functional dependencies, normalization requires identifying and addressing those not implied by primary keys.
* **BCNF Violations**: Check for violations of Boyce-Codd Normal Form (BCNF). For example, if Advisor Number → Advisor Name is found in the student table, it indicates that the table needs to be split to eliminate redundancy, placing the advisor's details in a separate table.