ADVANCE DATABASE MANAGEMENT SYSTEM

Chapter 7
Normalization

Bikash Khadka Shah

MCSE, MSDA, OCP, RHCE, RHCVA, CCNA, CEH

9841766620 | 9801076620

Functional Dependencies

- The functional dependency is a relationship that exists between two attributes. It typically exists between the primary key and non-key attribute within a table.
  - X → Y
  - The left side of FD is known as a determinant, the right side of the production is known as a dependent.

In the context of the given tables, let's define a few functional dependencies based on their attributes.

Example Functional Dependencies:

Students Table:

•Student_ID → Student_Name

•The Student_ID uniquely determines the Student_Name. Each student has a unique ID linked to their name.

•Student_ID → Email, Phone

•The Student_ID uniquely determines both the Email and Phone for a student.

- Subject_Code → Course_Name, Credit_Hours
  - The Subject_Code uniquely determines both the Course_Name and Credit_Hours in the Major_Subject table. Each subject code corresponds to a specific course name and credit hours.

- Trivial functional dependency

A trivial functional dependency in a database refers to a situation where an attribute or set of attributes functionally determines itself or a subset of itself. In other words, it's an FD that is always true by definition. It doesn't provide any additional information as the determined attribute(s) are already present in the determining attribute(s). – A → B has trivial functional dependency if B is a subset of A.

  – The following dependencies are also trivial

- A → A, B → B

- Trivial Functional Dependency: Student_ID → Student_ID
- In the Students table, the attribute Student_ID trivially determines itself. This is essentially stating that knowing a Student_ID gives you the same Student_ID. However, this doesn't provide any new information since the determining attribute is the same as the determined attribute

Full-functional Dependency

- A fully-functional dependency (FFD) occurs when an attribute or set of attributes functionally determine another non-prime attribute in a table, and no subset of those attributes has the same functional dependency on that non-prime attribute.

- Let's consider an example based on the Enrollment table:

- Suppose we have the following functional dependencies:

- Student_ID, Subject_Code → Grade
- Student_ID → Subject_Code
- Here, the dependency Student_ID, Subject_Code → Grade indicates that the grade obtained by a student in a specific subject is determined by both the Student_ID and Subject_Code.

Transitive Dependency

- Transitive dependency occurs when an attribute is functionally dependent on another non-prime attribute, which is not its primary key. In simpler terms, if A → B and B → C, then A → C is a transitive dependency.

- Let's consider an example based on the Students table:

- Suppose we have the following functional dependencies:

- **Student_ID → Student_Name**
- **Student_ID → Email**
- Here, both **Student_Name** and **Email** are functionally dependent on **Student_ID,** which is the primary key of the **Students** table.
- Now, if we introduce another hypothetical functional dependency:

- **Student_Name → Email**
- This dependency indicates that the Email attribute is functionally dependent on Student_Name.

- Since we have the dependencies:

- **Student_ID → Student_Name** (1st FD) • **Student_Name → Email** (hypothetical 3rd FD)
- We can infer transitive dependency:

- **Student_ID → Student_Name → Email**

Multivalued Dependency

- A multivalued dependency (MVD) occurs when there's a relationship between attributes within a table where one attribute set (or a set of attributes) determines multiple independent sets of attributes.

- Let's consider an example based on a modified Students table with additional attributes:

- Student_ID
- Courses_Enrolled (representing the courses a student is enrolled in)
- Books_Required (representing the books required for each course)
- Now, assume we have the following MVD:
- Student_ID →→ Courses_Enrolled, Books_Required
- This MVD indicates that for a given Student_ID, there can be multiple sets of Courses_Enrolled and multiple corresponding sets of Books_Required, and these sets are independent of each other for that particular student.

- Multivalued dependencies are essential to identify relationships where a set of attributes determines multiple sets of attributes independently within a table. They help in database normalization to maintain data integrity and reduce redundancy by breaking tables into smaller ones based on these dependencies.

Student_ID	Courses_Enrolled	Books_Required
1	[C1, C2]	[B1, B2]
1	[C3]	[B3]
2	[C1, C3]	[B1, B3]
3	[C2]	[B2]

Partial Dependency

- Partial dependency refers to a situation where a non-prime attribute in a table is functionally dependent on only a part of a composite primary key, rather than on the entire primary key.
- Partial Dependency occurs when a nonprime attribute is functionally dependent on part of a candidate key.
- The 2nd Normal Form (2NF) eliminates the Partial Dependency. Let's consider an example using the Enrollment table:

Suppose the table has the following attributes:

- Student_ID (part of the composite primary key)
- Subject_Code (part of the composite primary key)
- Grade
- Attendance
- And let's assume the following functional dependencies:

- Student_ID, Subject_Code → Grade
- Student_ID → Attendance
- Here, Grade is fully functionally dependent on the composite key (Student_ID,

Subject_Code), meaning for a given combination of Student_ID and

Subject_Code, there's a unique Grade. This dependency represents a scenario where Grade depends on both the student and the subject enrolled.

- However, the Attendance attribute is only functionally dependent on Student_ID, not on the entire composite key (Student_ID, Subject_Code). This scenario indicates a partial dependency, where Attendance depends on a part of the composite key (Student_ID) but not on the entire composite key.

Normalization

- Normalization in database management is a systematic approach aimed at simplifying table structures by breaking them down to eliminate data

redundancy and undesirable characteristics such as insertion, update, and delete anomalies.

- This process involves multiple steps to organize data into tabular form, eliminate duplicate entries, and establish relationships between tables.
- The goal of normalization is to reduce data repetition, organize large datasets into smaller, more manageable tables, and establish proper relationships between these tables to maintain data integrity and efficiency.

Normalization

- Normalization essentially involves a two-step process:

1. Removing repeating groups: This entails breaking down complex tables into simpler forms by identifying and removing repeating groups of data.
2. Eliminating duplicate data: Once repeating groups are addressed, the next step involves removing duplicate entries from the relational tables, ensuring that each piece of data is stored only once.

Need of Normalization

- The goal of normalization is to create a set of relational tables that are free of redundant data and that can be consistently and correctly modified.
  - It removes redundant data from a relational table.
  - It maintains consistency in the table.
  - It converts the complex table into simple one.
  - It tries to remove the anomalies as we proceed to Higher Normal Forms.

First Normal Form (1NF)

- A relation will be 1NF if it contains an atomic value.
- It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
- First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

Let's consider the Major_Subject table for checking if it satisfies the First Normal Form:

The Major_Subject table typically contains the following attributes:

- Subject_Code (Primary Key)

- Course_Name
- Credit_Hours
- Looking at the table, each attribute appears to contain atomic values. For instance:
- Subject_Code ,Course_Name,Credit_Hours each seem to hold single values.
- There are no arrays, lists, or repeating groups within a single attribute.
- Therefore, based on the given information, the **Major_Subject** seems to satisfy the First Normal Form (1NF) as it holds atomic values in each cell of the table without any repeating groups.

For Example Lab Only

- CREATE TABLE sample_major_subject AS SELECT FROM major_subject ;

```
-- Inserting records with redundant data into the Sample_Major_Subject table
INSERT INTO Sample_Major_Subject (Subject_Code, Course_Name,
Credit_Hours)
VALUES
(1, 'Mathematics', 4),
(2, 'Physics', 3),
(3, 'Chemistry', 4),
(1, 'Mathematics', 4),
(4, 'Biology', 3);
```

Subject_Code	Course_Name	Credit_Hours
1	Mathematics	4
2	Physics	3
3	Chemistry	4
1	Mathematics	4
4	Biology	3

- Although the table FIRST is in 1NF it contains redundant data.
- Although the table FIRST is in 1NF it contains redundant data.

- Redundancy causes update anomalies in Sample_Major_Subject table:
- Insert Anomalies:

Inserting redundant records leads to data inconsistency and duplication.

Duplicate entries necessitate managing and updating the same information separately, causing unnecessary replication of data.

- Delete Anomalies:

Deleting duplicate or redundant records risks unintentional loss of non-redundant data associated with those duplicates.

Removing one instance of redundancy might result in the loss of other valid and relevant information linked to that duplicate entry.

- Update Anomalies:

Redundant data across multiple records causes inconsistencies during updates.

Updating information in some instances of redundant data while leaving others unchanged leads to inconsistent and unreliable data.

For example, if attributes are updated in only a subset of duplicate records, it creates inconsistency within the dataset.

Second Normal Form (2NF)

The Second Normal Form (2NF) is a database normalization rule that builds upon the First Normal Form (1NF) by addressing issues related to composite keys and partial dependencies within a table.

For a table to satisfy the Second Normal Form (2NF), it must fulfill the following criteria:

- Be in 1NF.
- Have no partial dependencies, meaning all non-prime attributes are fully functionally dependent on the entire primary key.

- The Second Normal Form (2NF) builds upon the First Normal Form (1NF) and is particularly concerned with removing partial dependencies within a table that has a composite primary key.
- The process for transforming a 1NF table to 2NF is:
  - Identify any determinants other than the composite key, and the columns they determine.
  - Create and name a new table for each determinant and the unique columns it determines.
  - Move the determined columns from the original table to the new table. The determinate becomes the primary key of the new table.
  - Delete the columns you just moved from the original table except for the determinate which will serve as a foreign key.
  - The original table may be renamed to maintain semantic meaning.

Let's consider the Major_Subject table:

- Subject_Code (Primary Key)
- Course_Name
- Credit_Hours

For 2NF:

- **1NF Compliance:**
- Ensure that the table is already in the First Normal Form (1NF), meaning each column holds atomic values and there are no repeating groups.
- **Removing Partial Dependencies:**
- Check for any partial dependencies where non-prime attributes depend on only part of the primary key.

- In this scenario, if Course_Name and Credit_Hours both depend solely on the Subject_Code (part of the composite key) and not on any subset of the key, the table satisfies 2NF.
- Tables in 2NF but not in 3NF still contain modification anomalies.
- In the example of SECOND, they are:
  - INSERT. The fact that a particular city has a certain status (Rome has a status of 50) cannot be inserted until there is a supplier in the city.
  - DELETE. Deleting any row in SUPPLIER destroys the status information about the city as well as the association between supplier and city.

Example:

In this simplified Major_Subject table, Course_Name and Credit_Hours are attributes that seem to be fully functionally dependent on the entire Subject_Code (part of the composite primary key). There are no partial dependencies observed, meaning all non-prime attributes are fully determined by the entire primary key, and the table satisfies the Second Normal Form (2NF).

Subject_Code	Course_Name	Credit_Hours
1	Mathematics	4
2	Physics	3
3	Chemistry	4

4	Biology	3

Third Normal form (3NF)

- 1NF Compliance:
- Ensure both tables are in the First Normal Form (1NF), meaning each column holds atomic values without repeating groups.
- 2NF Compliance:
- Confirm that the tables are in the Second Normal Form (2NF), which implies no partial dependencies exist, and all non-prime attributes are fully functionally dependent on the entire primary key.
- 3NF Criteria:
- Eliminate any transitive dependencies, ensuring that non-prime attributes do not depend on other non-prime attributes.

•The process of transforming a table into 3NF is:

•Step 1: Confirm 1NF: Ensure each column holds atomic values without repeating groups. Based on the provided structure, assuming it's already in 1NF.

•Step 2: Check 2NF: Verify that there are no partial dependencies; all non-prime attributes (Course_Name and Credit_Hours) are fully dependent on the entire primary key (Subject_Code). If there are no partial dependencies, it's in 2NF.

•Step 3: Assess for 3NF: Look for any transitive dependencies, especially between non-prime attributes. If Course_Name or Credit_Hours depend on each other or any other non-primary attribute, resolve this by separating the table into distinct

tables, such as creating a Courses table with Subject_Code as the primary key and Course_Name, Credit_Hours as attributes.

Boyce-Codd Normal Form(BCNF)

- A relation is said to be in BCNF if
  - It is already in 3NF, and
  - The only determinate are candidate keys
  - Boyce-Codd Normal Form (BCNF) is a stricter form of normalization that ensures further reduction of redundancies and anomalies in a database table. A table is in BCNF if, for every non-trivial functional dependency $X \rightarrow Y$, X is a superkey.

- Let's consider a hypothetical scenario from the school management database
- Example: Course-Professor Allocation
- Consider a table that allocates professors to courses:

Course_Code	Professor_ID	Professor_Name	Course_Name
CSE101	P101	Prof. Smith	Computer Science I
MATH202	P102	Prof. Johnson	Calculus II
PHY301	P101	Prof. Smith	Physics III

To check for BCNF:

- Check for 3NF:
- Ensure the table is already in the Third Normal Form (3NF).
- Check for BCNF:
- Examine if each determinant (left-hand side of the functional dependency) is a superkey.

In the above example:

- Course_Code uniquely determines Professor_Name and Course_Name, forming a superkey.
- Professor_ID uniquely determines Professor_Name, also forming a superkey.

Therefore, in this example, the table complies with BCNF as each determinant (attributes on the left side of the functional dependencies) is a superkey,

ensuring there are no non-trivial functional dependencies where the determinant is not a superkey.

Fourth Normal form (4NF)

- A relational table is in the fourth normal form
- (4NF) if
  - It is in BCNF, and
  - All multivalued dependencies are also functional dependencies.

- Decompose the table given earlier in “Multivalued Dependency”

In the context of a school management database, let's consider a hypothetical scenario involving the Students and Courses tables:

Students Table

- Attributes: Student_ID (Primary Key), Student_Name, Address, Phone, Email

Courses Table

- Attributes: Course_ID (Primary Key), Course_Name, Subject_Code

Example of a 4NF Scenario:

Let's assume we have an additional table called Student_Courses that aims to track courses taken by students:

Student_Courses Table

- Attributes: Student_ID (Foreign Key), Course_ID (Foreign Key), Semester

In this scenario:

- Student_ID and Course_ID together form the composite primary key.
- There's a multi-valued dependency where a student can take multiple courses, and a course can be taken by multiple students, resulting in a many-to-many relationship.

To achieve 4NF:

- Ensure that there are no non-trivial multi-valued dependencies.
- Decompose the table to remove multi-valued dependencies and store data in a manner that avoids redundancy.

Decomposition Example:

To achieve 4NF, the Student_Courses table might be decomposed into two tables:

Student_Enrollment:

- Attributes: Student_ID (Foreign Key), Semester
- This table contains information about students and the semester they are enrolled in.

Course_Enrollment:

- Attributes: Course_ID (Foreign Key), Semester
- This table stores information about courses offered in different semesters.

- This decomposition allows for a more normalized structure by separating multivalued dependencies into distinct tables, reducing redundancy, and ensuring that each table satisfies 4NF.

Fifth Normal form (5NF)

- A relation is said to be in 5NF if and only if it satisfies 4NF and no join dependency exists.
- A relation is said to have join dependency if it can be recreated by joining multiple sub relations and each of these sub relations has a subset of the attributes of the original relation.
- In other words, a relation is in 5NF, only if it is already in 4NF and it cannot be decomposed further.