Normalization | Meerza Ahmed

Tuesday, September 12, 2023 1:00 PM

Lecture 4: Link | Syllabus

Class Week 4/15

Database Systems by Coronel & Morris

Chapter 6: Normalization of Data Tables

* 6.1 *Database Tables and Normalization*

How do you recognize a poor table structure, and how do you produce a good table? The answer to both questions involves normalization. Normalization is a process for evaluating and correcting table structures to minimize data redundancies, thereby reducing the likelihood of data anomalies. Normalization works through a series of stages called normal forms. The first three stages are described as first normal form (1NF), second normal form (2NF), and third normal form (3NF). From a structural point of view, 2NF is better than 1NF, and 3NF is better than 2NF. For most purposes in business database design, 3NF is as high as you need to go in the normalization process. However, you will discover that properly designed 3NF structures also meet the requirements of fourth normal form (4NF).

* 6.3 *The Normalization Process*

The objective of normalization is to ensure that each table conforms to the concept of well-formed relations—in other words, tables that have the following characteristics :

- Each table represents a single subject. For example, a COURSE table will contain only data that directly pertain to courses. Similarly, a STUDENT table will contain only student data.
- No data item will be unnecessarily stored in more than one table (in short, tables have minimum controlled redundancy). The reason for this requirement is to ensure that the data is updated in only one place.
- All nonprime attributes in a table are dependent on the primary key—the entire primary key and nothing but the primary key. The reason for this requirement is to ensure that the data is uniquely identifiable by a primary key value.
- Each table is void of insertion, update, or deletion anomalies, which ensures the integrity and consistency of the data.

NORMAL FORMS

NORMAL FORM	CHARACTERISTIC	SECTION
First normal form (1NF)	Table format, no repeating groups, and PK identified	6-3a
Second normal form (2NF)	1NF and no partial dependencies	6-3b
Third normal form (3NF)	2NF and no transitive dependencies	6-3c
Boyce-Codd normal form (BCNF)	Every determinant is a candidate key (special case of 3NF)	6-6a
Fourth normal form (4NF)	3NF and no independent multivalued dependencies	6-6b

* Normal forms such as the fifth normal form (5NF) and domain-key normal form (DKNF) are not likely to be encountered in a business environment and are mainly of theoretical interest. Such higher normal forms usually increase joins, which slows performance without adding any value in the elimination of data redundancy

FUNCTIONAL DEPENDENCE CONCEPTS

CONCEPT	DEFINITION
Functional dependence	The attribute *B* is fully functionally dependent on the attribute *A* if each value of *A* determines one and only one value of *B*. Example: PROJ_NUM → PROJ_NAME (read as *PROJ_NUM functionally determines PROJ_NAME*) In this case, the attribute PROJ_NUM is known as the determinant attribute, and the attribute PROJ_NAME is known as the dependent attribute.
Functional dependence (generalized definition)	Attribute *A* determines attribute *B* (that is, *B* is functionally dependent on *A*) if all (generalized definition) of the rows in the table that agree in value for attribute *A* also agree in value for attribute *B*.
Fully functional dependence (composite key)	If attribute *B* is functionally dependent on a composite key *A* but not on any subset of that composite key, the attribute *B* is fully functionally dependent on *A*.

* It is crucial to understand these concepts because they are used to derive the set of functional dependencies for a relation. The normalization process works one relation at a time, identifying the dependencies on that relation and normalizing the relation. Normalization starts by identifying the dependencies of a given relation and progressively breaking up the relation (table) into a set of new relations (tables) based on the identified dependencies

* Two types of functional dependencies are important in normalization; partial dependencies and transitive dependencies.

A partial dependency exists when there is a functional dependence in which the determinant is only part of the primary key (remember the assumption that there is only one candidate key). For example, if (A, B) → (C, D), B → C, and (A, B) is the primary key, then the functional dependence B → C is a partial dependency because only part of the primary key (B) is needed to determine the value of C. Partial dependencies tend to be straightforward and easy to identify

A transitive dependency exists when there are functional dependencies such that X → Y, Y → Z, and X is the primary key. In that case, the dependency X → Z is a transitive dependency because X determines the value of Z via Y. Unlike partial dependencies, transitive dependencies are more difficult to identify among a set of data.

Fortunately, there is an effective way to identify transitive dependencies: they occur only when a functional dependence exists among nonprime attributes. In the previous example, the actual transitive dependency is X → Z. However, the dependency Y → Z signals that a transitive dependency exists

* 6.3-a *Conversion to First Normal Form*

A relational table must not contain repeating groups. The existence of repeating groups fails to meet even the lowest normal form requirements, thus reflecting data redundancies. Normalizing the table structure will reduce the data redundancies. If repeating groups do exist, they must be eliminated by making sure that each row defines a single entity.

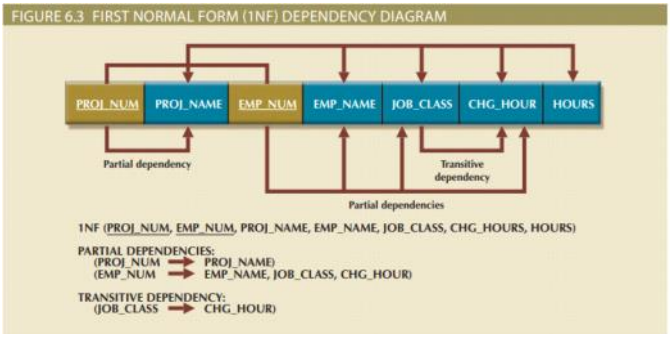
Normalization starts with a simple three-step procedure:

Step 1: Eliminate the Repeating Groups - Start by presenting the data in a tabular format, where each cell has a single value and there are no repeating groups. To eliminate the repeating groups, eliminate the nulls by making sure that each repeating group attribute contains an appropriate data value.

Step 2: Identify the Primary Key - For example, the PROJ_NUM values can identify any of the employees. To maintain a primary key that will uniquely identify *any attribute value, the new key must be composed of a combination of PROJ_NUM and EMP_NUM.*

Step 3: Identify All Dependencies - Dependency exists between two nonprime attributes; therefore, it is a signal that a transitive dependency exists, and we will refer to it as a transitive dependency.

FIGURE 6.3 FIRST NORMAL FORM (1NF) DEPENDENCY DIAGRAM



* All relational tables satisfy the 1NF requirements. The problem with the 1NF table structure shown in Figure 6.3 is that it contains partial dependencies—dependencies based on only a part of the primary key. While partial dependencies are sometimes used for performance reasons, they should be used with caution. This is because a table that contains partial dependencies is still subject to data redundancies, and therefore to various anomalies. The data redundancies occur because every row entry requires duplication of data.

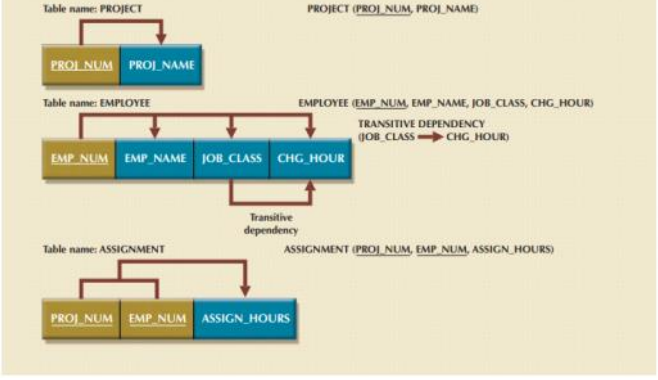
* 6.3-b *Conversion to Second Normal Form*

Conversion to 2NF occurs only when the 1NF has a composite primary key. If the 1NF has a single-attribute primary key, then the table is automatically in 2NF. The 1NF-to2NF conversion is simple.
Starting with the 1NF format displayed in Figure 6.3, you take the following steps :

Step 1: Make a new table to eliminate partial dependencies - For each component of the primary key that acts as a determinant in a partial dependency, create a new table with a copy of that component as the primary key. While these components are placed in the new tables, it is important that they also remain in the original table as well. The determinants must remain in the original table because they will be the foreign keys for the relationships needed to relate these new tables to the original table.

Step 2: Reassign Corresponding Dependent Attributes - The attributes that are dependent in a partial dependency are removed from the original table and placed in the new table with the dependency's determinant. Any attributes that are not dependent in a partial dependency will remain in the original table. In other words, the three tables that result from the conversion to 2NF are given appropriate names.

FIGURE 6.4 SECOND NORMAL FORM (2NF) CONVERSION RESULTS



* The results of Steps 1 and 2 are displayed in Figure 6.4. At this point, most of the anomalies discussed earlier have been eliminated. For example, if you now want to add, change, or delete a PROJECT record, you need to go only to the PROJECT table and make the change to only one row. Because a partial dependency can exist only when a table's primary key is composed of several attributes, a table whose primary key consists of only a single attribute is automatically in 2NF once it is in 1NF. Figure 6.4 still shows a transitive dependency, which can generate anomalies. For example, if the charge per hour changes for a job classification held by many employees, that change must be made for each of those employees. If you don't update some of the employee records that are affected by the change, employees with the same job description will generate different hourly charges.

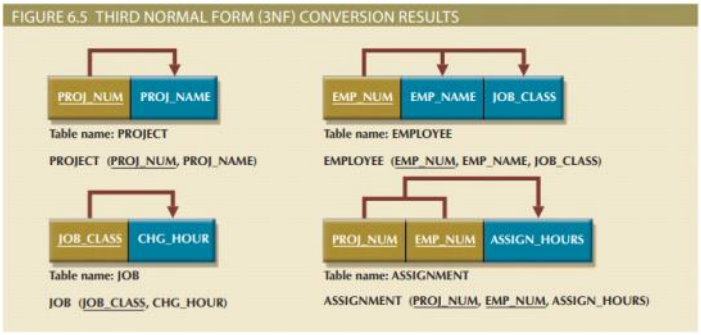
* 6.3-b *Conversion to Third Normal Form*

The data anomalies created by the database organization in Figure 6.4 are easily eliminated by completing the following two steps:

Step 1: Make New Tables to Eliminate Transitive Dependencies - For every transitive dependency, write a copy of its determinant as a primary key for a new table. A determinant is any attribute whose value determines other values within a row. If you have three different transitive dependencies, you will have three different determinants. As with the conversion to 2NF, it is important that the determinant remain in the original table to serve as a foreign key

Step 2: Reassign Corresponding Dependent Attributes - Identify the attributes that are dependent on each determinant identified in Step 1. Place the dependent attributes in the new tables with their determinants and remove them from their original tables.

FIGURE 6.5 THIRD NORMAL FORM (3NF) CONVERSION RESULTS



* It is interesting to note the similarities between resolving 2NF and 3NF problems. To convert a table from 1NF to 2NF, it is necessary to remove the partial dependencies. To convert a table from 2NF to 3NF, it is necessary to remove the transitive dependencies. No matter whether the "problem" dependency is a partial dependency or a transitive dependency, the solution is the same: create a new table for each problem dependency, The determinant of the problem dependency remains in the original table and is placed as the primary key of the new table. The dependents of the problem dependency are removed from the original table and placed as nonprime attributes in the new table.

* 6.6-b *Fourth Normal Form (4NF)*

You might encounter poorly designed databases, or you might be asked to convert spreadsheets into a database format in which multiple multivalued attributes exist. If you follow the proper design procedures illustrated in this book, you should not encounter the problems.

The discussion of 4NF and beyond is largely academic if you make sure that your tables conform to the following two rules:

1. All attributes must be dependent on the primary key, but they must be independent of each other.
2. No row may contain two or more multivalued facts about an entity

* 6.8 *Denormalization*

It is important to remember that the optimal relational database implementation requires that all tables be at least in third normal form (3NF). A good relational DBMS excels at managing normalized relations—that is, relations void of any unnecessary redundancies that might cause data anomalies. Although the creation of normalized relations is an important database design goal, it is only one of many such goals. Good database design also considers processing (or reporting) requirements and processing speed.

The problem with normalization is that as tables are decomposed to conform to normalization requirements, the number of database tables expands. Therefore, in order to generate information, data must be put together from various tables. Joining a large number of tables takes additional input/output (I/O) operations and processing logic, thereby reducing system speed. Most relational database systems are able to handle joins very efficiently. However, rare and occasional circumstances may allow some degree of denormalization so processing speed can be increased

COMMON DENORMALIZATION EXAMPLES

CASE	EXAMPLE	RATIONALE AND CONTROLS
Redundant data	Storing ZIP and CITY attributes in the AGENT table when ZIP determines CITY (see Figure 2.2)	Avoid extra join operations Program can validate city (drop-down box) based on the zip code
Derived data	Storing STU_HRS and STU_CLASS (student classification) when STU_HRS determines STU_CLASS (see Figure 3.28)	Avoid extra join operations Program can validate classification (lookup) based on the student hours
Preaggregated data (also derived data)	Storing the student grade point average (STU_GPA) aggregate value in the STUDENT table when this can be calculated from the ENROLL and COURSE tables (see Figure 3.28)	Avoid extra join operations Program computes the GPA every time a grade is entered or updated STU_GPA can be updated only via administrative routine
Information requirements	Using a temporary denormalized table to hold report data; this is required when creating a tabular report in which the columns represent data that are stored in the table as rows (see Figures 6.17 and 6.18)	Impossible to generate the data required by the report using plain SQL No need to maintain table Temporary table is deleted once report is done Processing speed is not an issue

* 6.8 *Data Modeling Checklist*

The data-modeling checklist shown in Table 6.7 will help ensure that you perform data-modeling tasks successfully based on the concepts and tools you have learned in this text

DATA-MODELING CHECKLIST

BUSINESS RULES

- Properly document and verify all business rules with the end users.
- Ensure that all business rules are written precisely, clearly, and simply. The business rules must help identify entities, attributes, relationships, and constraints.
- Identify the source of all business rules, and ensure that each business rule is justified, dated, and signed off by an approving authority.

DATA MODELING

Naming conventions: All names should be limited in length (database-dependent size).

- Entity names:
  - Should be nouns that are familiar to business and should be short and meaningful
  - Should document abbreviations, synonyms, and aliases for each entity
  - Should be unique within the model
  - For composite entities, may include a combination of abbreviated names of the entities linked through the composite entity
- Attribute names:
  - Should be unique within the entity
  - Should use the entity abbreviation as a prefix
  - Should be descriptive of the characteristic
  - Should use suffixes such as _ID, _NUM, or _CODE for the PK attribute
  - Should not be a reserved word
  - Should not contain spaces or special characters such as @, !, or &
- Relationship names:
  - Should be active or passive verbs that clearly indicate the nature of the relationship

Entities:

- Each entity should represent a single subject.
- Each entity should represent a set of distinguishable entity instances.
- All entities should be in 3NF or higher. Any entities below 3NF should be justified.
- The granularity of the entity instance should be clearly defined.
- The PK should be clearly defined and support the selected data granularity.

Attributes:

- Should be simple and single-valued (atomic data)
- Should document default values, constraints, synonyms, and aliases
- Derived attributes should be clearly identified and include source(s)
- Should not be redundant unless this is required for transaction accuracy, performance, or maintaining a history
- Nonkey attributes must be fully dependent on the PK attribute

Relationships:

- Should clearly identify relationship participants
- Should clearly define participation, connectivity, and document cardinality

ER model:

- Should be validated against expected processes: inserts, updates, and deletions
- Should evaluate where, when, and how to maintain a history
- Should not contain redundant relationships except as required (see attributes)
- Should minimize data redundancy to ensure single-place updates
- Should conform to the minimal data rule: All that is needed is there, and all that is there is needed.