- What normalization is and what role it plays in the database design process
- About the normal forms 1NF, 2NF, 3NF, BCNF, and 4NF
- How normal forms can be transformed from lower normal forms to higher normal forms
- That normalization and ER modeling are used concurrently to produce a good database design
- That some situations require denormalization to generate information efficiently

Good database design must be matched to good table structures. We learn to evaluate and design good table structures to control data redundancies, thereby avoiding data anomalies. The process that yields such desirable results is known as normalization.

In order to recognize and appreciate the characteristics of a good table structure, it is useful to examine a poor one. Therefore, we begins by examining the characteristics of a poor table structure and the problems it creates. Wy then learn how to correct a poor table structure. This methodology will yield important dividends: you will know how to design a good table structure and how to repair an existing poor one.

We will discover not only that data anomalies can be eliminated through normalization, but also that a properly normalized set of table structures is actually less complicated to use than an unnormalized set. In addition, we will learn that the normalized set of table structures more faithfully reflects an organization's real operations.

DATABASE TABLES AND NORMALIZATION

If the database tables are treated as though they are files in a file system, the relational database management system (RDBMS) never has a chance to demonstrate its superior data-handling capabilities.

Yet it is possible to create poor table structures even in a good database design. So 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).

Although normalization is a very important database design ingredient, you should not assume that the highest level of normalization is always the most desirable. Generally, the higher the normal form, the more relational join operations are required to produce a specified output and the more resources are required by the database system to respond to end-user queries. A successful design must also consider end-user demand for fast performance. Therefore, you will occasionally be expected to denormalize some portions of a database design in order to meet performance requirements. Denormalization produces a lower normal form; that is, a 3NF will be converted to a 2NF through denormalization. However, the price you pay for increased performance through denormalization is greater data redundancy.

THE NEED FOR NORMALIZATION

Normalization is typically used in conjunction with the entity relationship modeling. There are two common situations in which database designers use normalization. When designing a new database structure based on the business requirements of the end users, the database designer will construct a data model using a technique such as Crow's Foot notation ERDs. After the initial design is complete, the designer can use normalization to analyze the relationships that exist among the attributes within each entity, to determine if the structure can be improved through normalization. Alternatively, database designers are often asked to modify existing data structures that can be in the form of flat files, spreadsheets, or older database structures. Again, through an analysis of the relationships among the attributes or fields in the data structure, the database designer can use the normalization process to improve the existing data structure to create an appropriate database design. Whether designing a new database structure or modifying an existing one, the normalization process is the same.

To get a better idea of the normalization process, consider the simplified database activities of a construction company that manages several building projects. Each project has its own project number, name, employees assigned to it, and so on. Each employee has an employee number, name, and job classification, such as engineer or computer technician.

THE NORMALIZATION PROCESS

We will learn how to use normalization to produce a set of normalized tables to store the data that will be used to generate the required information. The objective of normalization is to ensure that each table conforms to the concept of well-formed relations—that is, 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 are 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 are uniquely identifiable by a primary key value.
- Each table is void of insertion, update, or deletion anomalies. This is to ensure the integrity and consistency of the data.

To accomplish the objective, the normalization process takes you through the steps that lead to successively higher normal forms.

The concept of keys is central to the discussion of normalization. Recall that a candidate key is a minimal (irreducible) superkey. The primary key is the candidate key that is selected to be the primary means used to identify the rows in the table. Although normalization is typically presented from the perspective of candidate keys, for the sake of simplicity while initially explaining the normalization process, we will make the assumption that for each table there is only one candidate key, and therefore, that candidate key is the primary key.

From the data modeler's point of view, the objective of normalization is to ensure that all tables are at least in third normal form (3NF). Even higher-level normal forms exist. However, 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. More often than not, such higher normal forms increase joins (slowing performance) without adding any value in the elimination of data redundancy. Some very specialized applications, such as statistical research, might require normalization beyond the 4NF, but those applications fall outside the scope of most business operations. Because we focuses on practical applications of database techniques, the higher-level normal forms are not covered.

Functional Dependence

Before outlining the normalization process, it's a good idea to review the concepts of determination and functional dependence.

It is crucial to understand these concepts because they are used to derive the set of functional dependencies for a given relation. The normalization process works one relation at a time, identifying the dependencies on that relation and normalizing the relation. As you will see in the following sections, 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 that are of special interest in normalization are 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 we are assuming 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 rather 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 easier way to identify transitive dependencies. A transitive dependency will 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. Hence, throughout the discussion of the normalization process, the existence of a functional dependence among nonprime attributes will be considered a sign of a transitive dependency. To address the problems related to transitive dependencies, changes to the table structure are made based on the functional dependence that signals the transitive dependency's existence. Therefore, to simplify the description of normalization, from this point forward we will refer to the signaling dependency as the transitive dependency.

CONVERSION TO FIRST NORMAL FORM
Because the relational model views data as part of a table or a collection of tables in which all key values must be identified.

A relational table must not contain repeating groups. The existence of repeating groups provides evidence that the table 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. In addition, the dependencies must be identified to diagnose the normal form. Identification of the normal form will let you know where you are in the normalization process. The normalization process 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

Step 3: Identify All Dependencies

Dependency diagrams are very helpful in getting a bird's-eye view of all of the relationships among a table's attributes, and their use makes it less likely that you will overlook an important dependency.

As you examine, note the following dependency diagram features:

1. The primary key attributes are bold, underlined, and shaded in a different color.

2. The arrows above the attributes indicate all desirable dependencies, that is, dependencies that are based on the primary key.

3. The arrows below the dependency diagram indicate less desirable dependencies. Two types of such dependencies exist:

a. Partial dependencies. A dependency based on only a part of a composite primary key is a partial dependency.

b. Transitive dependencies. In other words, a transitive dependency is a dependency of one nonprime attribute on another nonprime attribute. The problem with transitive dependencies is that they still yield data anomalies.

All relational tables satisfy the 1NF requirements. The problem with the 1NF table is that it contains partial dependencies—that is, 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. (If the information requirements seem to dictate the use of partial dependencies,

CONVERSION TO SECOND NORMAL FORM
Converting to 2NF is done 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.

Step 1: Make New Tables 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. It is important that the determinants remain in the original table because they will be the foreign keys for the relationships that are needed to relate these new tables to the original table. For the construction of our revised dependency diagram, write each key component on a separate line; then write the original (composite) key on the last line.

Each component will become the key in a new table. In other words, the original table is now divided into three tables.

Step 2: Reassign Corresponding Dependent Attributes

The dependencies for the original key components are found by examining the arrows below the dependency diagram. The attributes that are dependent in a partial dependency are removed from the original table and placed

in the new table with its determinant. Any attributes that are not dependent in a partial dependency will remain in the original table. In other words, the tables that result from the conversion to 2NF are given appropriate names.

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.

CONVERSION TO THIRD NORMAL FORM
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

Place the dependent attributes in the new tables with their determinants and remove them from their original tables.

Draw a new dependency diagram to show all of the tables you have defined in Steps 1 and 2. Name the table to reflect its contents and function. Check all of the tables to make sure that each table has a determinant and that no table contains inappropriate dependencies.

Note that this conversion has eliminated the original table's transitive dependency; the tables are now said to be in third normal form (3NF).

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.

Be aware, however, that while the technique is the same, it is imperative that 2NF be achieved before moving on to 3NF; be certain to resolve the partial dependencies before resolving the transitive dependencies. Recall, however, the assumption that was made at the beginning of the discussion of the normalization process—that each table has only one candidate key, which is the primary key. If a table has multiple candidate keys, then the overall process remains the same, but there are additional considerations.

For example, if a table has multiple candidate keys and one of those candidate keys is a composite key, the table can have partial dependencies based on this composite candidate key, even when the primary key chosen is a single attribute. In those cases, following the process described above, those dependencies would be perceived as transitive dependencies and would not be resolved until 3NF. The simplified process described above will allow the designer to achieve the correct result, but through practice, you should recognize all candidate keys and their dependencies as such, and resolve them appropriately. The existence of multiple candidate keys can also influence the identification of transitive dependencies. Previously, a transitive dependency was defined to exist when one nonprime attribute determined another nonprime attribute. In the presence of multiple candidate keys, the definition of a nonprime attribute as an attribute that is not a part of any candidate key is critical. If the determinant of a functional dependence is not the primary key but is a part of another candidate key, then it is not a nonprime attribute and does not signal the presence of a transitive dependency.

IMPROVING THE DESIGN

The table structures are cleaned up to eliminate the troublesome partial and transitive dependencies. You can now focus on improving the database's ability to provide information and on enhancing its operational characteristics. Please note that for space issues, each section presents just one example—the designer must apply the principle to all remaining tables in the design. Remember that normalization cannot, by itself, be relied on to make good designs. Instead, normalization is valuable because its use helps eliminate data redundancies.

Evaluate PK Assignments
A surrogate key, as you should recall, is an artificial PK introduced by the designer with the purpose of simplifying the assignment of primary keys to tables. Surrogate keys are usually numeric, they are often automatically generated by the DBMS, they are free of semantic content (they have no special meaning), and they are usually hidden from the end users.

Evaluate Naming Conventions
It is best to adhere to the naming conventions.

Refine Attribute Atomicity
It is generally good practice to pay attention to the atomicity requirement. An atomic attribute is one that cannot be further subdivided. Such an attribute is said to display atomicity. Such a task would be very difficult if the name components were within a single attribute. In general, designers prefer to use simple, single-valued attributes as indicated by the business rules and processing requirements.

Identify New Attributes
If the EMPLOYEE table were used in a real-world environment, several other attributes would have to be added.

Identify New Relationships
The designer must take care to place the right attributes in the right tables by using normalization principles.

Refine Primary Keys as Required for Data Granularity
Granularity refers to the level of detail represented by the values stored in a table's row. Data stored at their lowest level of granularity are said to be atomic data. as explained earlier.

Maintain Historical Accuracy

Evaluate Using Derived Attributes
From a strictly database point of view, such derived attribute values can be calculated when they are needed to write reports or invoices. However, storing the derived attribute in the table makes it easy to write the application software to produce the desired results. Also, if many transactions must be reported and/or summarized, the availability of the derived attribute will save reporting time. (If the calculation is done at the time of data entry, it will be completed when the end user presses the Enter key, thus speeding up the process.)

SURROGATE KEY CONSIDERATIONS

Although this design meets the vital entity and referential integrity requirements, the designer must still address some concerns. For example, a composite primary key might become too cumbersome to use as the number of attributes grows. (It becomes difficult to create a suitable foreign key when the related table uses a composite primary key. In addition, a composite primary key makes it more difficult to write search routines.) Or a primary key attribute might simply have too much descriptive content to be usable. When, for whatever reason, the primary key is considered to be unsuitable.

At the implementation level, a surrogate key is a system-defined attribute generally created and managed via the DBMS. Usually, a system-defined surrogate key is numeric, and its value is automatically incremented for each new row. For example, Microsoft Access uses an AutoNumber data type, Microsoft SQL Server uses an identity column, and Oracle uses a sequence object.

It is worth repeating that database design often involves trade-offs and the exercise of professional judgment. In a real-world environment, you must strike a balance between design integrity and flexibility. Unfortunately, that limitation is likely to be undesirable from a managerial point of view. In any case, frequent data audits would be appropriate.

HIGHER-LEVEL NORMAL FORMS

Tables in 3NF will perform suitably in business transactional databases. However, there are occasions when higher normal forms are useful.

THE BOYCE-CODD NORMAL FORM (BCNF)

A table is in Boyce-Codd normal form (BCNF) when every determinant in the table is a candidate key. Clearly, when a table contains only one candidate key, the 3NF and the BCNF are equivalent.

Putting that proposition another way, BCNF can be violated only when the table contains more than one candidate key.

Most designers consider the BCNF to be a special case of the 3NF. In fact, if the techniques shown here are used, most tables conform to the BCNF requirements once the 3NF is reached. So how can a table be in 3NF and not be in BCNF? To answer that question, you must keep in mind that a transitive dependency exists when one nonprime attribute is dependent on another nonprime attribute.

In other words, a table is in 3NF when it is in 2NF and there are no transitive dependencies. But what about a case in which a nonkey attribute is the determinant of a key attribute? That condition does not violate 3NF, yet it fails to meet the BCNF requirements because BCNF requires that every determinant in the table be a candidate key.

A + B ^ C, D

A + C ^ B, D

C ^ B

Notice that this structure has two candidate keys: (A + B) and (A + C). (The condition C ^ B indicates that a nonkey attribute determines part of the primary key—and that dependency is n ot transitive or partial because the dependent is a prime attribute!) .Yet the condition C ^ B causes the table to fail to meet the BCNF requirements.

To convert the table into table structures that are in 3NF and in BCNF, first change the primary key to A + C. That is an appropriate action because the dependency C ^ B means that C is, in effect, a superset of B. At this point, the table is in 1NF because it contains a partial dependency, C ^ B. Next, follow the standard decomposition procedures to produce.

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. For example, consider the possibility that an employee can have multiple assignments and can also be involved in multiple service organizations. Suppose employee 10123 does volunteer work for the Red Cross and United Way. In addition, the same employee might be assigned to work on three projects:

Table 1

Table name: VOLUNTEER_V1

EMP_NUM	ORG_CODE	ASSIGN_NUM
10123	RC	1
10123	UW	3
10123		4

Table name: VOLUNTEER_V2

EMP_NUM	ORG_CODE	ASSIGN_NUM
10123	RC	
10123	UW	
10123		1
10123		3
10123		4

Table name: VOLUNTEER_V3

EMP_NUM	ORG_CODE	ASSIGN_NUM
10123	RC	1
10123	RC	3
10123	UW	4

There is a problem with the tables in Figure 1. The attributes ORG_CODE and ASSIGN_NUM each may have many different values. In normalization terminology, this situation is referred to as a multivalued dependency. A multivalued dependency occurs when one key determines multiple values of two other attributes and those attributes are independent of each other. (One employee can have many service entries and many assignment entries. Therefore, one EMP_NUM can determine multiple values of ORG_CODE and multiple values of ASSIGN_NUM; however, ORG_CODE and ASSIGN_NUM are independent of each other.) The presence of a multivalued dependency means that if versions 1 and 2 are implemented, the tables are likely to contain quite a few null values; in fact, the tables do not even have a viable candidate key. (The EMP_NUM values are not unique, so they cannot be PKs. No combination of the attributes in table versions 1 and 2 can be used to create a PK because some of them contain nulls.) Such a condition is not desirable, especially when there are thousands of employees, many of whom may have multiple job assignments and many service activities. Version 3 at least has a PK, but it is composed of all of the attributes in the table. In fact, version 3 meets 3NF requirements, yet it contains many redundancies that are clearly undesirable.

The solution is to eliminate the problems caused by the multivalued dependency. You do this by creating new tables for the components of the multivalued dependency. In this example, the multivalued dependency is resolved by creating the ASSIGNMENT and SERVICE_V1 tables depicted in Figure 2. Note that in Figure 2, neither the ASSIGNMENT nor the SERVICE_V1 table contains a multivalued dependency. Those tables are said to be in 4NF.

Table 2

Table name: PROJECT

PROJ_CODE	PROJ_NAME	PROJ_BUDGET
1	BeThere	1023245.00
2	BlueMoon	20198608.00
3	GreenThumb	3234456.00
4	GoFast	5674000.00
5	GoSlow	1002500.00

Table name: EMPLOYEE

EMP_NUM	EMP_LNAME
10121	Rogers
10122	O'Leery
10123	Panera
10124	Johnson

Table name: ASSIGNMENT

ASSIGN_NUM	EMP_NUM	PROJ_CODE
1	10123	1
2	10121	2
3	10123	3
4	10123	4
5	10121	1
6	10124	2
7	10124	3
8	10124	5

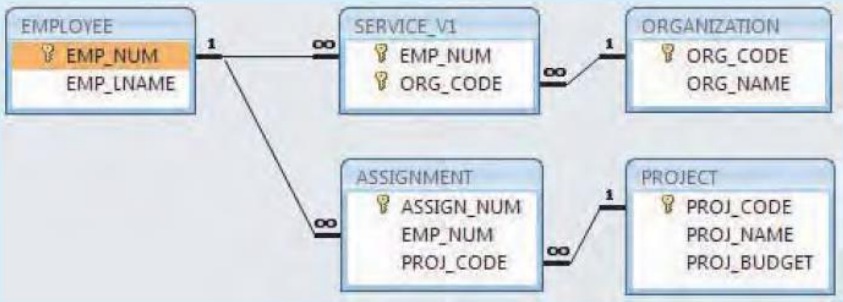
Table name: ORGANIZATI

ORG_CODE	ORG_NAME
RC	Red Cross
UW	United Way
WF	Wildlife Fund

Table name: SERVICE_V1

EMP_NUM	ORG_CODE
10123	RC
10123	UW
10123	WF

The relational diagram



If you follow the proper design procedures illustrated in this book, you shouldn't encounter the previously described problem. Specifically, the discussion of 4NF 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.

NORMALIZATION AND DATABASE DESIGN

But even the best database designers are known to make occasional mistakes that come to light during normalization checks. However, many of the real-world databases you encounter will have been improperly designed or burdened with anomalies if they were improperly modified over the course of time. And that means you might be asked to redesign and modify existing databases that are, in effect, anomaly traps. Therefore, you should be aware of good design principles and procedures as well as normalization procedures.

First, an ERD is created through an iterative process. You begin by identifying relevant entities, their attributes, and their relationships. Then you use the results to identify additional entities and attributes. The ERD provides the big picture, or macro view, of an organization's data requirements and operations.

Second, normalization focuses on the characteristics of specific entities; that is, normalization represents a micro view of the entities within the ERD. And as you learned in the previous, the normalization process might yield additional entities and attributes to be incorporated into the ERD. Therefore, it is difficult to separate the normalization process from the ER modeling process; the two techniques are used in an iterative and incremental process.

To illustrate the proper role of normalization in the design process, let's reexamine the operations of the contracting company whose tables were normalized in the preceding sections. Those operations can be summarized by using the following business rules:

- The company manages many projects.
- Each project requires the services of many employees.
- An employee may be assigned to several different projects.
- Some employees are not assigned to a project and perform duties not specifically related to a project. Some employees are part of a labor pool, to be shared by all project teams. For example, the company's executive secretary would not be assigned to any one particular project.
- Each employee has a single primary job classification. That job classification determines the hourly billing rate.
- Many employees can have the same job classification. For example, the company employs more than one electrical engineer.

Given that simple description of the company's operations, two entities and their attributes are initially defined:

The design process is now on the right track. The ERD represents the operations accurately, and the entities now reflect their conformance to 3NF. The combination of normalization and ER modeling yields a useful ERD, whose entities may now be translated into appropriate table structures.

DENORMALIZATION

It's 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.

NORMALIZATION OF DATABASE TABLES

Keep in mind that the advantage of higher processing speed must be carefully weighed against the disadvantage of data anomalies. On the other hand, some anomalies are of only theoretical interest. For example, should people in a real-world database environment worry that a ZIP_CODE determines CITY in a CUSTOMER table whose primary key is the customer number? Is it really practical to produce a separate table for

ZIP (ZIP_CODE, CITY)

to eliminate a transitive dependency from the CUSTOMER table? (Perhaps your answer to that question changes if you are in the business of producing mailing lists.) As explained earlier, the problem with denormalized relations and redundant data is that the data integrity could be compromised due to the possibility of data anomalies (insert, update, and deletion anomalies). The advice is simple: use common sense during the normalization process.

As seen in the faculty evaluation report, the conflicts between design efficiency, information requirements, and performance are often resolved through compromises that may include denormalization. In this case, and assuming there is enough storage space, the designer's choices could be narrowed down to:

Although 2NF tables cannot always be avoided, the problem of working with tables that contain partial and/or transitive dependencies in a production database environment should not be minimized. Aside from the possibility of troublesome data anomalies being created, unnormalized tables in a production database tend to suffer from these defects:

• Data updates are less efficient because programs that read and update tables must deal with larger tables.

• Indexing is more cumbersome. It is simply not practical to build all of the indexes required for the many attributes that might be located in a single unnormalized table.

• Unnormalized tables yield no simple strategies for creating virtual tables.

Remember that good design cannot be created in the application programs that use a database. Also keep in mind that unnormalized database tables often lead to various data redundancy disasters in production databases such as the ones examined thus far. In other words, use denormalization cautiously and make sure that you can explain why the unnormalized tables are a better CHOICE in certain situations than their normalized counterparts.

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:
  1. Should be nouns that are familiar to business and should be short and meaningful
  2. Should document abbreviations, synonyms, and aliases for each entity
  3. 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:
  1. Should be unique within the entity
  2. Should use the entity abbreviation as a prefix
  3. Should be descriptive of the characteristic
  4. Should use suffixes such as _ID, _NUM, or _CODE for the PK attribute
  5. Should not be a reserved word
  6. Should not contain spaces or special characters such as @, !, or &
- Relationship Names:
  1. 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 deletes
- 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."

SUMMARY

W Normalization is a technique used to design tables in which data redundancies are minimized. The first three normal forms (1NF, 2NF, and 3NF) are most commonly encountered. From a structural point of view, higher normal forms are better than lower normal forms, because higher normal forms yield relatively fewer data

redundancies in the database. Almost all business designs use 3NF as the ideal normal form. A special, more restricted 3NF known as Boyce-Codd normal form, or BCNF, is also used.

- A table is in 1NF when all key attributes are defined and when all remaining attributes are dependent on the primary key. However, a table in 1NF can still contain both partial and transitive dependencies. (A partial dependency is one in which an attribute is functionally dependent on only a part of a multiattribute primary key. A transitive dependency is one in which one attribute is functionally dependent on another nonkey attribute.) A table with a single-attribute primary key cannot exhibit partial dependencies.
- A table is in 2NF when it is in 1NF and contains no partial dependencies. Therefore, a 1NF table is automatically in 2NF when its primary key is based on only a single attribute. A table in 2NF may still contain transitive dependencies.
- A table is in 3NF when it is in 2NF and contains no transitive dependencies. Given that definition of 3NF, the Boyce-Codd normal form (BCNF) is merely a special 3NF case in which all determinant keys are candidate keys. When a table has only a single candidate key, a 3NF table is automatically in BCNF.
- A table that is not in 3NF may be split into new tables until all of the tables meet the 3NF requirements.
- Normalization is an important part—but only a part—of the design process. As entities and attributes are defined during the ER modeling process, subject each entity (set) to normalization checks and form new entity (sets) as required. Incorporate the normalized entities into the ERD and continue the iterative ER process until all entities and their attributes are defined and all equivalent tables are in 3NF.
- A table in 3NF might contain multivalued dependencies that produce either numerous null values or redundant data. Therefore, it might be necessary to convert a 3NF table to the fourth normal form (4NF) by splitting the table to remove the multivalued dependencies. Thus, a table is in 4NF when it is in 3NF and contains no multivalued dependencies.
- The larger the number of tables, the more additional I/O operations and processing logic required to join them. Therefore, tables are sometimes denormalized to yield less I/O in order to increase processing speed. Unfortunately, with larger tables, you pay for the increased processing speed by making the data updates less efficient, by making indexing more cumbersome, and by introducing data redundancies that are likely to yield data anomalies. In the design of production databases, use denormalization sparingly and cautiously.
- The Data-Modeling Checklist provides a way for the designer to check that the ERD meets a set of minimum requirements.