Database Systems for Management Third edition

James F. Courtney

David B. Paradice

Kristen L. Brewer

Julia C. Graham

For any questions about this text, please email: drexel@uga.edu

The Global Text Project is funded by the Jacobs Foundation, Zurich, Switzerland

This book is licensed under a Creative Commons Attribution 3.0 License

CHAPTER CONTENTS

The Goals of Normalization
Modification Anomalies
Deletion Anomalies
Insertion Anomalies
Update Anomalies
The Role of Assumptions in Identifying Anomalies
Avoiding Problems Due to Dependence
Functional Dependence
Full Functional Dependence
Transitive Dependence
Multivalued Dependence
Decomposing Poor Designs
First, Second, Third, and Boyce-Codd Normal Forms
First Normal Form
Second Normal Form
Third Normal Form
Boyce-Codd Normal Form
Avoiding Problems with Multiple Values
Fourth Normal Form
Avoiding Problems with Decompositions
Fifth Normal Form
Domain-Key Normal Form
Side Effects of Normalization

CHAPTER 5
Normalized Database Design

This chapter focuses on developing robust database designs. In Chapter 3 we presented tools and methods for designing a database system. In Chapter 4 we overviewed the major models and discussed how to create schemas for each. In this chapter, we present relational examples that will help us distinguish between good and bad database designs. Although, we use the relational model to illustrate concepts, the techniques we present here are applicable regardless of the approach taken to implement a database.

THE GOALS OF NORMALIZATION

There are many possible designs for the data structures used in a database system. In the long run, some designs prove better than others. We consider one design superior to another if (1) it makes the relationships in the database easier to understand and simpler to maintain, (2) it is more amenable to processing new requests for data, (3) it reflects the semantics, or meaning, of the situation being modeled, (4) it prevents the storage of invalid information, and (5) it can be represented efficiently using available hardware and software.

Whenever data items have been haphazardly grouped into records (or attributes into relations), one or more of the three properties mentioned above is compromised. When data items that are logically unrelated (or logically independent) are aggregated, users become confused. Such groupings also make maintenance difficult because the relationships being maintained are unclear. Finally, new requests for data may require

substantial work (1) to determine if the data item being supplied is the one desired, or (2) to create new (possibly more logical) groupings.

When these events occur, the database system may quickly become regarded as ineffective or useless. Experience has shown that most problems can be traced to improper, or unsound, conceptual database designs. Such designs represent monetary losses to management; they require frequent restructuring of the database to accommodate new demands and longer times to implement changes, and they result in ineffective use of the organization's data asset.

Normalization is a technique that structures data in ways that help reduce or prevent problems. The normalization process results in logically consistent record structures that are easy to understand and simple to maintain. Various levels of normalization may be obtained. Criteria that define the levels of normalization are called **normal forms.**

In this chapter we examine database designs in order to identify their weak and strong points. We present this material as an intuitive approach to the concept of normalization. Consequently, you may find that you have already developed a "sixth sense" for some of the material discussed in this chapter.

Rarely is a manager required to recite the criteria of the various normal forms. Nevertheless, a manager must be familiar with the ramifications of poor database design. As the criteria for higher levels of normalization are met, database system designers are able to say with confidence that certain problems (to be discussed shortly) will likely not exist.

Kristen Brewer 5/18/09 1:42 PM
Deleted: never

Several types of problems that may result from poor database design are presented in this chapter. Each of several sources of these problems is examined in detail, and in each case alternative database designs are developed.

MODIFICATION ANOMALIES

To help recognize the differences among alternative database designs, it is useful to examine how mistakes may be inadvertently introduced into a database. Mistakes may occur during insertion, deletion, or modification of data in the database. When the mistake is a consequence of the design of the database, a **modification anomaly** is said to exist. Figures 5-1 through 5-4 will be used to illustrate modification anomalies and the processes that resolve them.

Deletion Anomalies

Consider the relation in Figure 5-1. This table contains data about employees, the department in which they work, the projects on which they are working, and the manager of the project. The primary key of a data structure is underlined. The data structure in Figure 5-1 suffers from a number of problems. If Baker decides to leave the company, and the tuple or record containing the Baker data is deleted from this database, all data recording that Cody is the manager of the project to identify new investment possibilities is lost. The removal of one logical entity which results in loss of information about an unrelated logical entity is called a **deletion anomaly.** An anomaly (in general) is a deviation from whatever is expected. This deletion has unexpected results: data regarding a project manager is lost when data about an employee is deleted.

Another example of a deletion anomaly can be seen in the table in Figure 5-2. This data structure contains data regarding travel plans for members of a private travel club. This organization maintains vacationing facilities worldwide. Notice that if W. Earp decides to cancel the vacation in London and the corresponding

tuple or record is deleted, all data regarding the cost of traveling to London is lost.

A third example can be seen in the medical data shown in Figure 5-3. In this design, the admitting doctor's office address is lost when patient data is deleted. One could argue that this data is no longer needed after a patient's data is deleted. But a reentry of the doctor's office address is highly likely, and thus a data structure that maintains this information is preferable.

A final example is shown in Figure 5-4. This data structure contains data on insurance agents and their clients. As you can see, should policy owners Lucas and Hammock cancel their policies, all information about insurance agent Jones is lost. As in the medical data example, a data structure that maintains this data is preferable.

Insertion Anomalies

The data structures in Figures 5-1 through 5-4 have a second kind of defect. Suppose that a new project is planned for the organization illustrated in Figure 5-1, and one wants to add data regarding the manager of that project. Because E-Number is part of the primary key and key attributes may not be null, adding this data is impossible until at least one person has been assigned to work on the project.

The insertion of data about one logical entity which requires the insertion of data about an unrelated logical entity is called an **insertion anomaly.** In this case, inserting data about a project manager requires insertion of an unrelated piece of data--a worker assigned to the project.

The same type of insertion anomaly can be seen in Figure 5-2. These designs require that a member of the club be planning to vacation in a particular location in order for cost data about that location to be inserted in the data structure. Consequently, adding cost data about travel to a new location is not possible.

In Figure 5-3, data on a new doctor's office cannot be inserted until a patient is admitted by the doctor. The data about a doctor's office is important even when the doctor has no patients, but the current design precludes maintaining this information. A data structure design that allows insertion of this data regardless of the doctor's current patient load is preferable.

Figure 5-4 shows the same type of problem. An insurance agent's address cannot be inserted until the agent has a client. As with the medical data, this restriction is unacceptable.

Update Anomalies

Refer again to Figure 5-1. Suppose that a project gets a new manager. The design in Figure 5-1 requires a change in several tuples. The modification of the information for one logical entity which requires more than one modification to a relation is called an **update anomaly.**

The same sort of problem exists in Figure 5-2. If the cost of travel to Madrid changes, the modification must be made in more than one place. Similarly, a modification to an admitting doctor's office address (or an employer's address) in Figure 5-3 requires changes in many occurrences. The same problem is evident with an insurance agent's address in Figure 5-4.

In these examples, the update anomaly does not appear to be a costly design error, because very few rows are shown in the figures. The flaw becomes very much greater if thousands of occurrences are maintained. Imagine the impact of such a design error in a student information management system for a major

university with 30,000 student records.

The Role of Assumptions in Identifying Anomalies

In identifying modification anomalies as we have in these examples, we rely on some unstated assumptions. Such assumptions should be made explicit to avoid inaccurately identifying anomalies. In Figure 5-1, for example, we assume that all employees have one, unique employee number. We also assume that projects have unique project numbers and that each has only one manager. We do not assume that employees have unique names, but we do assume that organizations (i.e., employers) have unique names. These are reasonable assumptions.

Consider the data in Figure 5-2. Can we conclude from this relation that the destination data item always determines the cost attribute? Although such a conclusion appears evident, we cannot reach this conclusion based on the data in the relation. We must rely on an underlying assumption that the cost for travel to a particular destination is always the same.

Such assumptions depend on the underlying policies of the organization. You should *always* confirm relationships between data items by examining the underlying assumptions. Data structures are time-invariant, but data items assume various values over time. Consequently, as a designer of a database structure, you must consider the underlying (time-invariant) relationships to identify modification anomalies.

Kristen Brewer 5/18/09 1:51 PM
Formatted: Font:Italic, No underline

AVOIDING PROBLEMS DUE TO DEPENDENCE

The modification anomalies discussed above all stem from the fact that the value of one data item can be determined from the value of a second data item, while the other data items in the data structure are logically independent of this relationship. The name of a project in Figure 5-1, for example, can be determined from the number of the project. The value of the project name data item "depends" on the value of the project number data item.

We introduce here a notation using bubble charts that can be used to diagram this notion of dependence. To indicate when attributes that compose a key have relationships with non-key attributes, we circle attributes that compose a concatenated primary key. Figure 5-5 shows an arrow from the P-Number data item to the Manager data item. The direction of the arrow indicates which item depends on the other. In this case, Manager depends on P-Number. The diagram also indicates that P-Number determines Project name.

In Figure 5-2 there is a relationship between Destination and Cost of traveling to that destination which exists independently of who travels there. Travel to Madrid, for example, always costs $1,500, regardless of who plans to go to Madrid. Travel to Los Angeles is always $900. This is diagrammed in Figure 5-6.

Figure 5-7 shows the dependency diagram for the medical data structures. In this diagram, some of the arrows come from the circle denoting the primary key, indicating that the key determines the data item. Medical Record Number and Admission Date and Admission Time *taken together* determine, for example, Admitting Doctor.

Kristen Brewer 5/18/09 1:51 PM
Formatted: Font:Italic, No underline

Kristen Brewer 5/18/09 1:51 PM
Formatted: Font:Italic

Figure 5-8 shows the dependence diagram for the insurance data structures. Here the primary key is composed of a single data item, so there is no confusion about what is determined by the components of the primary key.

Problems that exist in the data structures in Figure 5-1 can be found by examining the dependency diagrams of data items relative to the primary key. Notice that we must combine two data items (employee number and project number) to form a key for this data structure. The project name, however, can be completely determined if we know just the project number. The project name is not dependent on which employee is currently assigned to the project. Whenever such a combination of conditions exists, modification anomalies are also present. In general, a design in which the non-key attributes are dependent on all of the key attributes, and not just any subset of them, is better.

Functional Dependence

From the previous examples, we can introduce the concept of functional dependence. **Functional dependence** exists between two data items in a data structure if the value of one data item implies the value of a second data item. We say that the first data item "determines" the second, and we frequently refer to the first data item as a **determinant.**

Note that there is no requirement for unique values in this discussion. Yates, for example, appears as a manager of two different projects in Figure 5-1. A common misconception about functional dependencies is that the attribute being determined cannot occur more than once in the data. Actually, the important point is that the same manager name appears for every instance of a particular project name. That the manager's name appears more than once is irrelevant.

There is a second common misunderstanding about functional dependence. When a data item (or collection of data items) determines another data item, the second data item does not necessarily determine the first. In the data structures mentioned above, that Yates is a manager does not determine the name of the project Yates manages.

Full Functional Dependence

In cases where the value of a combination of data items determines the value of another data item, we wish to distinguish further between the times when the entire combination of values is required to make the determination and when a subset of the combination of values is adequate. A data item is said to be **fully functionally dependent** on a combination of data items if it is functionally dependent on the combination of data items and not functionally dependent on any proper subset of the combination of data items. A proper subset of a set is any subset of that set except the set itself and the empty set.

If we examine the design in Figure 5-5, we can see that E-Number alone determines employee Name (because an employee has a unique employee number). Because the primary key is the combination of E-Number and P-Number, but a subset of the primary key determines the value of employee Name, employee Name is not fully functionally dependent on the primary key. Figure 5-6 indicates that the Cost of travel to a Destination is dependent only on the Destination and is not influenced by who is traveling or when the trip occurs. Hence Cost is not fully functionally dependent on the primary key. In Figure 5-7, however, Diagnosis is determined by *all* of the primary key and not by any subset of the key. Therefore Diagnosis is fully functionally dependent on the primary key.

Kristen Brewer 5/18/09 1:43 PM
Formatted: Font:Italic, No underline

Transitive Dependence

If the value of one data item determines the value of a second data item that is not a candidate key, and the value of the second data item determines the value of a third data item, then a **transitive dependence**

exists between the data items. The dependency diagram in Figure 5-7 can be used to demonstrate transitive dependence. Because Medical Record Number determines Employer and Employer determines Employer's Address (each patient has one employer and each employer has only one address), a transitive dependence exists between Medical Record Number and Employer's Address. Because of this transitive relationship, the first data item determines the third. We note that combinations of data items may be considered, but we'll restrict the consideration to the cases where full functional dependence holds.

Multivalued Dependence

When one value of a data item is associated with a collection of values of a second data item, a **multivalued dependence** exists and we say that the first data item multidetermines the second. Functional dependence is a special case of multivalued dependence. We examine multivalued dependencies in greater detail below.

Decomposing Poor Designs

Now that we have developed the concept of functional dependence, we have a firm foundation on which to build our techniques for designing good data structures.

New data structures can be derived from the ASSIGNMENT data structure shown in Figure 5-1. The result is presented in Figure 5-9. These tables do not suffer from the problems identified earlier. New projects can be created without the need to assign employees. Employees can be added without affecting information about the project. Also, departmental information on employees has been separated into a separate table.

Figure 5-10 shows the dependency diagram for the revised data structures. Compared to the diagram in Figure 5-6, these are much simpler.

Figure 5-11 shows the result of decomposing the structures in Figure 5-2. In this design it is possible to add new destinations before anyone plans to visit them. Altering customer data without affecting data about travel costs is also possible. There are only a small number of updates required to change the cost associated with traveling to a specific location. Figure 5-12 contains the corresponding revised dependency diagram, which shows clearly that the Cost data item is no longer determined by a subset of a primary key.

Figure 5-13 shows a decomposition of the medical data in Figure 5-3. Adding a new doctor's address is now possible. Similarly, data for an employer can now be added without regard to patient data. In this design, many patient records can "share" an employer's data record. This reduces the data redundancy in maintaining an employer record for each patient. The dependency diagrams in Figure 5-14 reflect these changes.

FIRST, SECOND, THIRD, AND BOYCE-CODD NORMAL FORMS

The process of eliminating anomalies results in normalization of the database design. The data structures meet certain criteria known as **normal forms.** There is a sequence of normal forms, each one adding more constraints to a data structure as we progress from the first set of criteria, called **first normal form,** through the highest set, called **domain-key normal form.** The first four normal forms deal with functional dependence.

First Normal Form

A data structure is in **first normal form** if it can be represented as a flat file structure. Flat files contain no repeating groups. A *repeating group* is a data item that can occur as a collection of values. In the case of our travel example, for instance, each customer may have many trips. Figure 5-15a shows a typical non-flat data structure. The data structure in this example contains data for purchase orders and items being ordered. A purchase order can contain any number of items ordered. Hence "item ordered" is a repeating group. Figure 5-15b shows the equivalent flat data structure.

Kristen Brewer 5/18/09 1:57 PM
Formatted: Font:Italic

A benefit gained by converting data structures to first normal form is that the converted structures are much easier to process. Data structures containing repeating groups must also contain either data indicating how many occurrences of the repeating item are contained in the structure or a special mark indicating when the last item occurs. Flat file structures have fixed, or constant, characteristics.

Second Normal Form

If a data structure is in first normal form and all non-key data items are fully functionally dependent on the primary key, then the structure is in **second normal form**. In the design in Figure 5-1 a non-key attribute (employee Name) is determined by a subset of the primary key (E-Number), so this structure is not in second normal form. By comparison, the redesign shown in Figure 5-9 meets the criteria for second normal form. In the ASSIGNMENT relation of Figure 5-9, there are no non-key data items. Therefore non-key data items cannot possibly be dependent on a subset of the primary key. Consequently, the ASSIGNMENT relation is in at least second normal form. The other two tables have primary keys composed of a single data item, so a non-key data item cannot be determined by a proper subset of the primary key. We can therefore conclude that these data structures are also in (at least) second normal form.

Figure 5-12 presents the revised dependence diagrams for the travel club tables from Figure 5-11. MEMBER-DATA and TRAVEL-COSTS contain no multiple-data item keys, so they must be in second normal form. TRAVEL-PLANS contains no non-key data items, so it must also be in second normal form.

The dependence diagrams for the revised medical relations (Figure 5-14) and the insurance relations (Figure 5-8) show clearly that these data structures are in second normal form. In most cases, the primary key is a single data item, thus precluding a violation of full functional dependence. In the ADMISSION data structure (Figure 5-14), the diagram shows that the only determinant is the entire primary key.

Third Normal Form

If a data structure is in second normal form and contains no transitive dependencies, then the structure is in **third normal form**. If we return again to the medical record data structures in Figure 5-3, we now have two reasons why these structures are not in third normal form: They are not in second normal form and we have also identified a transitive dependence. The transitive dependence was that Medical Record Number determined Employer and Employer determined Employer's Address.

In Figure 5-9 we know immediately that if the ASSIGNMENT data structures are in second normal form they must also be in third normal form. The reason is that we need at least three attributes to define a transitive dependence. Because we have only two attributes, no transitive dependence exists. We can conclude that the ASSIGNMENT data structures are in at least third normal form.

Observe that all projects have a unique name and that each project has only one manager. Project number determines project name and project name determines manager. However, a transitive dependency does not exist between the data items because project name is a candidate key. Therefore, the PROJECT relation is in third normal form.

The dependency diagram shown in Figure 5-7 for the MEDICAL-DATA data structure indicates it is not in third normal form because a transitive dependency exists (Medical Record Number determines Employer determines Employer's Address). This design is not even in second normal form because a subset of the key determines some non-key attributes. To achieve third normal form, the design would require a further decomposition of the MEDICAL-DATA data structure as shown in Figure 5-14.

Figure 5-12 does not expose any transitive dependency in the travel club data structure. Similarly, Figure 5-8 shows no transitive dependency exists in the insurance data structure.

Boyce-Codd Normal Form

The original development of the normal forms stopped at third normal form. However, since that time other normal forms have been identified from research into dependencies when the primary key is composed of a collection of attributes (a concatenated key). The next normal form we will discuss was identified by R. F. Boyce and E. F. Codd, and bears their names.

If a data structure is in third normal form and all determinants are candidate keys, then the data structure is in **Boyce-Codd normal form**. In the design of Figure 5-1 we can identify that project number is a determinant (of project name and manager) but is not a candidate key because project number will not uniquely determine a tuple. Hence we could immediately rule out the possibility that the data structure in Figure 5-1 is in Boyce-Codd normal form. In our alternative design shown in Figure 5-10, the PROJECT data structure is in Boyce-Codd normal form because the project name determines the manager and the project name is a candidate key. A **candidate key** is an *attribute* or collection of attributes that uniquely indentifies a *tuple*. The ASSIGNMENT data structure is in Boyce-Codd normal form because there are no non-key data items.

Kristen Brewer 5/18/09 2:00 PM
Formatted: Font:Not Bold, Italic

Kristen Brewer 5/18/09 1:59 PM
Formatted: Font:Italic

The decomposition of the travel club data structure shown in Figure 5-12 shows TRAVEL-PLANS in Boyce-Codd normal form because there are no non-key data items. Also, TRAVEL-COSTS and MEMBER-DATA are in Boyce-Codd normal form because the only determinant in each is the primary key.

Similarly, all of the other decompositions have achieved Boyce-Codd normal form. In Figure 5-14, all determinants are primary keys. The same is true in Figure 5-8, the diagram for the insurance data structure.

Boyce-Codd normal form is not a natural by-product of third normal form. Figure 5-16a presents a relation (of travel data) in third normal form, which is not in Boyce-Codd normal form. A key assumption in this example is that the airline travels to only one destination. (This is quite reasonable and frequently the case when dealing with commuter airlines in small university towns.)

Kristen Brewer 5/18/09 1:44 PM
Deleted: form which

The relation in Figure 5-16a is clearly in first normal form; it is in second normal form because no subset of the key (Customer and Destination combined) determines Carrier, and no transitive dependency exists. However, Carrier determines Destination (hence Carrier is a determinant) and Carrier is not a candidate key. Therefore, the relation is not in Boyce-Codd normal form.

All of the modification anomalies exist. If the tuple with Customer number 6155 is removed, the information that Reliable Airlines travels to Austin is lost. If another Carrier should be added to the table, it cannot be

added until it has a customer. If an airline changes its name, the change must be reflected in multiple occurrences in the table. Thus, third normal form still has modification anomalies.

Kristen Brewer 5/18/09 1:44 PM
Deleted: inthe

The decomposition in Figure 5-16b resolves these issues. Customer 6155's data can be removed without affecting the database. A new Carrier can be added (to OPTION) without customer data being required. Also, an airline's name can change without requiring multiple updates to the database.

The normal forms discussed thus far are summarized in Table 9-1. A general heuristic to follow is that when more than one relationship is represented in a data structure, there is quite likely to be some type of modification anomaly inherent in the structure. In all of these cases, modification anomalies have been resolved by decomposing the original table into smaller tables. When you decompose into smaller tables, you must remember to embed the appropriate part of the primary key from the original table in the new table. (In some cases, the entire primary key must appear in the new table.)

At first glance it may appear that transforming a database design into a higher normal form requires duplicating data. Actually, although the number of data item *types* in the database may be increased, the number of data item *values* stored in the database is actually reduced. Hence there is a net savings in storage requirements for the database.

Kristen Brewer 5/18/09 1:52 PM
Formatted: Font:Italic, No underline

Kristen Brewer 5/18/09 1:52 PM
Formatted: Font:Italic, No underline

When a data structure is in Boyce-Codd normal form (or higher), it does not suffer modification anomalies due to functional dependence. Boyce-Codd normal form is therefore a desirable database design goal.

AVOIDING PROBLEMS WITH MULTIPLE VALUES

There are also problems in data structure designs, which are not due to functional dependence. Figure 5-17 presents a relation containing employee numbers, project numbers, and computer codes. Assume that an employee may use the computer for any project on which he or she is working, and that there is no restriction on the number of computer accounts an employee may have. The primary key must be the concatenated key shown in order to reflect these assumptions.

Kristen Brewer 5/18/09 1:44 PM
Deleted: designs which

Now suppose that employee number 100 has another computer account authorized. Figure 5-18 shows the modified database. Due to the initial assumptions, adding one tuple may lead to erroneous inferences. This data seems to indicate that employee number 100 uses two computer accounts for project 23796 and only one for project 34548. The problem is caused partially by the fact that the attributes Project and User-code can assume multiple values for a given employee. More important, there is no logical dependence between the account numbers and the projects. As before, the root of the problem is in the attempt to represent more than one relationship in a data structure.

Figure 5-19 presents another example of the problems that can be caused by multiple values. This table contains data about a person's music preferences and the automobiles that the person owns. Clarke enjoys jazz and rock music and owns a sports car and a truck. Suppose that Clarke buys a van. If the data is updated as shown in Figure 5-20, there appears to be an implication that Clarke prefers jazz music while driving the van. Two occurrences must be added, as in Figure 5-21, to maintain a logically consistent relationship. The lack of any logical relationship between music preference and automobile type causes these problems.

Note also the deletion and update anomalies in this example. If Clarke sells the sports car, more than one occurrence must be deleted to reflect this. If Clarke's music preference switches from rock to classical, more than one occurrence must be modified to accurately represent this change.

Problems with multiple values arise when there are two or more logically independent relationships within a

data structure. In the first example, Project and User-code have logically independent relationships with Employee. An employee may work on many projects and use many computer accounts, but there is no logical relationship between projects and computer accounts. In the second example, Music Preference and Automobile Owned exist in logically independent relationships with Name. A person may enjoy many types of music and own several automobiles, but there is no logical relationship between music preference and type of automobile owned.

Fourth Normal Form

A comprehensive discussion of the normal forms beyond the Boyce-Codd normal form is beyond the scope of this text. It is instructional, however, to present examples and general definitions for these normal forms. These examples illustrate the potential problems in structures that are in Boyce-Codd normal form.

Fourth normal form addresses the issue of problems caused by multivalued dependence. If a data structure is in Boyce-Codd normal form, and either (1) it contains no multivalued dependencies or (2) all its multivalued dependencies are also functional dependencies, then the data structure is in **fourth normal form**.

Many discussions of fourth normal form include definitions of multivalued dependence. Problems with multivalued dependence arise only when there are three or more *logically independent* data items within a data structure. In our example (Figure 5-18), Project and User-code are logically independent. Employee multidetermines Project and multidetermines User-code, but there is no dependence between Project and User-code. Note that the existence of a multivalued dependence does not imply the existence of a problem. Consider, for example, Figure 5-22, which shows a table containing the attributes Employee, Skill, and Music Type. In this case we assume that there is a relationship between skill and music type. The employee Borst, for example, is a trained critic of classical music and has composed classical, jazz, and rock music. The multiple values do not cause a problem because there is a relationship between skill and music type. Only when the multiple values are independent is there a problem.

Kristen Brewer 5/18/09 1:44 PM
Formatted: Font:Italic, No underline

AVOIDING PROBLEMS WITH DECOMPOSITIONS

There are problems in relations that have nothing to do with either dependence among attributes or multiple values. Consider the information embodied by the relation in Figure 5-23. At first glance it may seem that a reasonable alternative design could be as shown in Figure 5-24. But notice the result of combining the two data structures in Figure 5-25 (perhaps by joining the relations) where EMPLOYEE Manager equals MANAGER Name. New data has been introduced. The combination of values "Adams, Yates, 23438" and the combination "Clarke, Yates, 26197" do not exist in the original data of Figure 5-24. In the relational model, projections that produce spurious information on joining are called **loss projections.** We refer to the general case as **loss decompositions.** Database designs should strive for non-loss decompositions -- decompositions that do not produce incorrect information when recombined.

A non-loss decomposition for this data structure is shown in Figure 5-26. Note that these data structures reconstruct the original data relationships.

Fifth Normal Form

Fifth normal form addresses the issue of loss and non-loss decompositions. In the relational model, this is yet another type of dependence: join dependence. A **join dependence** is a constraint that requires a relation to be the join of its projections. We refer to the general case as a **decomposition dependence.**

A relation is in **fifth normal form** if the data relationships embodied in the data structure cannot be

reconstructed from data structures containing fewer data items. If the data structure can be decomposed only into smaller records that all have the same key, then the data structure is in fifth normal form (Kent, 1983). A data structure in fifth normal form also satisfies all lower normal forms.

A **constraint** is simply a rule or condition that tuples of a relation must satisfy. Constraints include referential and entity integrity, domain constraints, and functional dependencies.

DOMAIN-KEY NORMAL FORM

The final normal form we discuss is domain-key normal form, also called sixth normal form. Domain-key normal form was defined in 1981 by R. Fagin. Fagin stated that a relation "is in **domain-key normal form** if every constraint can be inferred by simply knowing the set of attribute names and their underlying domains, along with the set of keys."

An explanation of domain-key normal form requires definition of two more types of dependence. A **key dependence** implies no changes to the relation that would result in the key no longer being a unique identifier of tuples in the relation are allowed. Figure 5-27a shows an occurrence of a relation with the key dependence (or constraint) explicitly stated.

The second dependence is **domain dependence.** In domain dependence the values for an attribute must come from a specific domain. In Figure 5-27a the domain dependencies for the Actual Cost attribute and the Expected Cost attribute are also stated explicitly.

The tuple "New billing system, 23760, Baker, 1000, 10000" in Figure 5-27b would violate domain-key normal form if added to the PROJECT relation, because P-Number would no longer be a key. The tuple "New ad campaign, 34004, Kanter, 0, -1000" in Figure 5-27c would violate domain-key normal form if it were added to the PROJECT relation, because -1000 is not a valid value for Expected Cost (costs should be positive). We also note that any tuple deletion that would result in either a key or domain dependence being violated also violates domain-key normal form.

As you can see, the definition of domain-key normal form is unlike that of the prior normal form definitions in that it says nothing about traditional dependencies (functional, multivalued, and join) but deals instead with the basic concepts of domains, keys, and constraints. Fagin showed that, after modification of the traditional normal forms to consider the consequences of domain sizes, the traditional normal forms are all implied by domain-key normal form. In other words, if you pick the right key and define domains properly, the system will most likely be appropriate.

Domain-key normal form has practical as well as theoretical implications. From a practical viewpoint, it is much easier for database software to deal with domains and keys than to deal with functional, transitive, or multivalued dependencies. Hence future database systems may be able to incorporate checks for normal form consistency based on the domain-key approach to normalization. This could ease the burden of the design phase that is so critical to effective database system use.

SIDE EFFECTS OF NORMALIZATION

By now you may have decided that normalization is without fault. For the most part, we encourage this view. But as is frequently the case, the solution to one set of problems introduces new problems.

Relationship Constraints We noted above that the decomposition of data structures into smaller structures of higher normal form results in a duplication of data item types. Every time a data structure is decomposed for the sake of normalization, the determinant data item (or items) ends up in two structures, acting as the

primary key in one of them. We saw this in the decomposition of the TRAVEL-CLUB data structure in Figure 5-2 into the TRAVEL-PLANS and TRAVEL-COSTS data structures in Figure 5-12. The determinant data item Destination became the key in the TRAVEL-COSTS relation.

The Destination attribute in TRAVEL-PLANS is now a foreign key. If referential integrity is maintained in a database after normalization, then all values of foreign keys must either be null or exist as values of the corresponding primary key. As explained earlier, the decomposition process requires that an appropriate part of the primary key in the original relation (perhaps all of it) be included in the new relations that are formed. Therefore, as primary keys occur in more places, more opportunities exist for referential integrity to be compromised. Consequently, the decomposition process inherently yields a set of referential integrity constraints involving the data structures created in the decomposition.

The overhead necessary to maintain these constraints must be balanced against the ease with which users can interact with the database system. As in many cases, this balance is a design decision, which must be made by the information analyst.

Kristen Brewer 5/18/09 1:45 PM
Deleted: decision which

Retrieval Inefficiency The increase in data structures inherent in the normalization process can also adversely affect the retrieval efficiency of the database system, especially in a microcomputer-based system. Consequently, a database system designer who knows that the database system will reside on a microcomputer may choose to keep the data structures in less than Boyce-Codd normal form.

In such a case, the data structures are frequently indexed on several keys, allowing access to the data in the structure in many ways. Fortunately, many microcomputer database systems provide a programming-like language that can be used to protect against the modification anomalies inherent in the lower normal forms.

THE USER'S VIEW

Issues of normalization and consistency are important concerns for database system designers, and the design decisions made in these areas directly affect database system users. The database is a model of the organization, and as such must change to reflect the changes in the organization. When a database changes, however, preservation of all existing relevant relationships is necessary. This need may be at odds with proposed changes to the relationships modeled in the database, as when rearrangement of the data item groupings in the database becomes necessary. When normalization is a goal of database design from the start, the likelihood of changes being required that will have a detrimental effect on existing applications programs is reduced.

One of the most challenging tasks in database system development is to determine an efficient implementation of the database. Conversion to a normalized database results in minimizing the amount of data stored in most (but not all) cases. One might not expect this, since normalization requires decomposing tables. But whereas some duplication of data *types* may be incurred, there is usually a reduction in the number of occurrences of the data type, which must be stored. Hence there is a net reduction in the total storage required.

Kristen Brewer 5/18/09 1:52 PM
Formatted: Font:Italic, No underline

Kristen Brewer 5/18/09 1:45 PM
Deleted: type which

The largest advantage to the user is that normalized databases are easier to maintain, due largely to the fact that data relationships in a normalized database are much more clearly and simply represented. This means that new applications will be easier to implement, which the user will see in terms of faster response to requests for changes. On the other hand, users might find a normalized database is difficult to comprehend. The easiest database for users to understand would consist of a single, universal relation that

contained everything. No joins would be required and only one table name is needed! Also, as discussed above, normalization can impact retrieval performance.

Metropolitan National Bank

You now have some basis for evaluating the preliminary implementation designs that you developed earlier. For each relation in your relational implementation, identify all functional dependencies and determine the normal form for the relation. Redesign as necessary to achieve Boyce Codd normal form. If you must redesign, be sure to revise your Entity-Relationship model if necessary to reflect any changes in *relationships*. In revising your original implementation designs, be careful about the degree to which you allow the normalization process to direct your redesign efforts. You must carefully weigh the trade-offs between resolving anomalies and maintaining a design that supports the priorities outlined in earlier chapters.

Kristen Brewer 5/18/09 1:52 PM
Formatted: Font:Italic, No underline

Discuss the design decisions you make in a report that accompanies the final design. Compare your normalized design to the performance criteria developed in earlier chapters. Does your normalized design still provide the response times required to adequately support the MetNat decision-making environment? State explicitly how your implementation succeeds or fails in this regard. If you feel that a problem exists, could utility programs be developed to mitigate this problem? State explicitly what these utility programs will provide and when they should be executed. If you feel that the design cannot be adequately adjusted using utility programs, state why you feel the problem cannot be resolved.

CHAPTER SUMMARY

Modification anomalies are the result of poor database design. These anomalies include deletion anomalies, insertion anomalies, and update anomalies. Deletion anomalies occur when the deletion of a data item in a data structure results in loss of unrelated information from the database. Insertion anomalies occur when one data item cannot be added to a data structure without the addition of an unrelated data item. Update anomalies occur when many updates to a database must occur in order for a single data item to be updated.

The most common source of these problems is dependence among attributes. Modification anomalies are likely to occur when more than one relationship is modeled in a relation. A second source of modification anomalies exists when logically independent attributes occur in a relation and these attributes are allowed to assume multiple values. Both sources of modification anomalies can be avoided by decomposing the data structure into smaller, more logically consistent, data structures.

In some rare cases in a relational database implementation, improper use of projection may be a third potential source of problems. These problems occur when the join of the projections results in a relation that contains data not in the original relation. Non-loss projections do not have this property. Projection must be used carefully to correct modification anomalies in a relational database design.

Normalization is a process that removes anomalies from data structures. Criteria that define the many levels of normalization are known as normal forms. Each normal form builds on the criteria of the prior forms to construct more stable data structures. Although the normalization process produces more instances of an attribute type, it actually results in a reduction in storage requirements because fewer actual data values must be stored in the database. Unfortunately, the introduction of normalized data structures increases the need for concern regarding referential integrity between data structures. Also, in microcomputer-based systems, normalization may have to be compromised to increase retrieval efficiency.

QUESTIONS AND EXERCISES

1. What is a deletion anomaly? an insertion anomaly? an update anomaly? Why are modification anomalies avoided?

2. Which type of modification anomaly is worse? Why?

3. When might a database designer decide that modification anomalies should not be resolved? What additional burden does this place on users of the database? What additional processing may be required of applications programs that use the database?

4. What is the meaning of "dependence" as used in this chapter?

5. Can a relation with only two attributes ever have problems due to dependence?

6. Can a relation with only two attributes ever have problems due to multiple values?

7. Consider your answers to Questions 5 and 6. Should relational database designers strive to make all relations binary?

8. Give an example of a relation with problems due to multiple values.

9. Show non-loss projections of the relation in Figure 5-16.

10. Join the projections in your answer to Question 9 to demonstrate that your answer is correct.

NOTE: Many of the following exercises appeared in Chapter 3. In that chapter, the problems called for E-R models and data structure diagrams. Your solutions to those earlier exercises will assist you here. If you did not create E-R models or data structure diagrams, you should do so now.

11. Consider an INVESTMENT database that contains data items for Investor's name, Amount invested, Stock, Broker's name, Broker's office address, and Broker's telephone. The following assumptions may be made: investors may invest in many stocks; a broker has only one office; and each broker's office has only one telephone number. Design data structures in Boyce-Codd normal form modeling these data relationships.

Kristen Brewer 5/18/09 1:46 PM
Deleted: which model

12. Consider a database of TRANSFER-STUDENT data intended to maintain data about students transferring into a university. The database contains the following data items: Student's name, Student's address, Student's telephone number, Prior university attended, GPA at the prior university, Hours earned at the prior university, Last date at the prior university, Student's sex, Student's date of birth, Guardian's name, Guardian's address, and Relationship of guardian to the student. The following assumptions may be made: students have one address and one telephone number; guardians have one address and one telephone number; and a student may have attended more than one prior university. Design data structures in Boyce-Codd normal form modeling these data relationships.

Kristen Brewer 5/18/09 1:46 PM
Deleted: which

Kristen Brewer 5/18/09 1:47 PM
Deleted: model

13. A database for a local garage is needed. The database contains data items for Customer's name, Customer's address, Customer's work telephone, Customer's home telephone, Date of work done, Automobile make, Automobile model, Description of work done, Parts needed to complete work, Charge for parts needed, Charge for labor performed, and Total charge. For warranty reasons, data must be

maintained in the database for at least ninety days; therefore a customer may have several records in the database at any time. Identical parts have only one cost, but different parts have different costs (e.g., all tires cost the same and all engines cost the same, but a tire and an engine do not cost the same). Design data structures in Boyce-Codd normal form modeling these data relationships.

Kristen Brewer 5/18/09 1:46 PM
Deleted: which model

14. A small database for a restaurant is needed. The database should contain data for Customer's name, Reservation date, Reservation time, Seating preference (booth or table), Area preference (indoor or outdoor), Number of people dining, and Telephone number where the customer can be reached if needed prior to arrival. Assume that a customer makes only one reservation per date, but do not assume that the customer's name can alone act as a primary key. Design data structures in Boyce-Codd normal form modeling these data relationships. Be prepared to discuss any additional assumptions you make.

Kristen Brewer 5/18/09 1:47 PM
Deleted: which

15. A cable television company desires a database containing data on its customers. The following data items are needed: Customer name, Customer address, Customer telephone number, Account number, Service provided (basic, movie channel A, movie channel B, movie channel C, and/or children's channel), the Charge for each service, Equipment used by the customer that belongs to the cable television company (decoder, remote control, etc.), an Identifying number for each piece of equipment, and the Extra charge (if any) for any equipment used by the customer. Assume that a customer may have more than one account number (some customers have rental properties and use the cable service to make their properties more attractive). Design data structures in Boyce-Codd normal form modeling these data relationships.

Kristen Brewer 5/18/09 1:47 PM
Deleted: which

16. Consider the following relations and functional dependencies. Assume in part (a) that the key for R1 is AC, the key for R2 is A, and the key for R3 is AC. In part (b), assume that the key for R1 is S and the key for R2 is SU. What normal form are the relations currently in? Convert all relations to Boyce-Codd normal form.

a.
R1 (A,B,C,D)
R2 (A,B,E)
R3 (A,C,D,F)
A -> B
AC -> D
A -> E
B -> CD
D -> F

b.
R1 (S,T,U,V,W)
R2 (S,U,Y)
S -> TU
S -> V
S -> W
SU -> Y
TU -> W
U -> W
U -> S

17. Suppose you have a data structure for a dance studio. The fields in the data structure are dancer identification number, dancer name, dancer address, dancer telephone number, class identification number, day that the class meets, and time that the class meets. Assume that each student takes one class, and each class meets only once a week. Assume also that dancer identification number is the key. What normal form is the data structure currently? Decompose this data structure into at least Boyce-Codd normal form.

18. Consider a database used to support a soccer referees association. Suppose the association schedules referees for games in three youth leagues in the county, for any high school game in the county, and for any college game played at a university in the county. Currently, the database consists of only one data structure that contains the following information: game number (unique), date of game, time of game, location of game, referee assigned to game, telephone number of referee (in case the game is cancelled), home team, visiting team, league in which the teams compete, and telephone number of the league office. Data for games for several weeks in advance is maintained in the database. In the case of youth games, a referee may be assigned to more than one game per day. Draw a dependency diagram of this data and decompose the data into data structures in at least Boyce-Codd normal form.

19. Suppose you have been asked to design a database for a company that manages magazine subscriptions. This company mails thousands of advertisements each year to households throughout the country offering hundreds of magazine subscriptions at discount subscription rates. The company also has a sweepstakes each year in which many prizes are awarded to lucky respondents. Design a database that will maintain the following data: subscriber's name and address, magazines subscribed, expiration dates of subscriptions, prizes awarded to subscriber (if any), date of last advertisement sent to subscriber, addresses of all locations that have received an advertisement regardless of response, and magazines offered and terms of subscription for each magazine.

FOR FURTHER READING

Normalization as discussed by the originator of the relational approach is described in:

Codd, E. F. "Further Normalization of the Relational Database Model." In Data Base Systems, Courant Computer Science Symposia 6, Prentice-Hall, Englewood Cliffs, New Jersey, 1972, pp. 65-98.

For a very readable (and applied) discussion of the normal forms criteria, see

Kent, William. "A Simple Guide to Five Normal Forms in Relational Database Theory," Communications of the ACM, Vol. 26, No. 2, February 1983, pp. 120-125.

Domain-key normal form was first defined in the following work by Fagin:

Fagin, R. "A Normal Form for Relational Databases That Is Based on Domains and Keys," ACM Transactions on Database Systems, Vol. 6, No. 3, September 1981.

FIGURE 9-1
Business Relation

FIGURE 5-2
Travel Club Relation

FIGURE 5-3
Medical Relation

FIGURE 5-3 (continued)

FIGURE 5-4
Insurance Relation

FIGURE 5-5
Dependency Diagram for Figure 5-1

FIGURE 5-6
Dependency Diagram for Figure 5-2

FIGURE 5-7
Dependency Diagram for Figure 5-3

FIGURE 5-8
Dependency Diagram for Figure 5-4

FIGURE 5-9
Normalized Relations for the Data in Figure 5-1

FIGURE 5-10
Revised Dependency Diagrams for the Data Structure in Figure 5-9

Figure 5-11
Normalized Relations for the Data in Figure 5-2

Kristen Brewer 5/18/09 1:48 PM
Deleted: FIgure

FIGURE 5-12
Revised Dependency Diagrams for the Data Structure in Figure 5-11

FIGURE 5-13
Normalized Relations for Figure 5-3

FIGURE 5-13 (continued)

FIGURE 5-14
Revised Dependence Diagrams for the Data Structure in Figure 5-13

FIGURE 5-15
File Structures. (a) Non-flat file structures have repeating groups; (b) converting the non-flat file in (a) to a flat file eliminates repeating data items.

FIGURE 5-16
Third Normal Form and Boyce-Codd Normal Form. (a) A relation in third normal form but not in Boyce-Codd normal form; (b) a Boyce-Codd normalization of (a).

FIGURE 5-17
Relation Containing Multiple Values

FIGURE 5-18
A Tuple Has Been Added to the Data. Does employee 100 really use two computer accounts for project 23796 and only one account for project 34548?

FIGURE 5-19
Relation Containing Multiple Values

FIGURE 5-20
An Additional Tuple Demonstrates a Problem Caused by Multiple, Independent Values. Does Clarke really enjoy only jazz music only while in the van?

FIGURE 5-21
The Relation for Figure 5-20. A second tuple is needed to maintain logical consistency.

FIGURE 5-22
The Multiple Values in This Data Are Related, So There Are No Anomalies

FIGURE 5-23
A Typical Relation Used in Project Management

FIGURE 5-24
An Apparently Harmless Decomposition of the Relation in Figure 5-23

FIGURE 5-25
The Result of Joining the Relations in Figure 5-24. Note that tuples that did not exist in the original relation have been created.

FIGURE 5-26
A Set of Non-loss Projections of the Data in Figure 5-23

FIGURE 5-27
Domain-Key Normal Form. (a) Relation that currently satisfies domain-key normal form; (b) violation of domain-key normal form key dependence; (c) violation of domain-key normal form domain dependence.

FIGURE 5–1 BUSINESS Relation

ASSIGNMENT

E-Number	Name	Department	Project	P-Number	Manager
1001	Adams	Accounting	New billing system	26716	Yates
1001	Adams	Accounting	Mailing list maintenance	23835	Kanter
1002	Baker	Finance	Identify new investments	43873	Cody
1003	Clarke	Accounting	New billing system	26716	Yates
1003	Clarke	Accounting	Purchasing system design	34761	Yates

FIGURE 5–5
Dependency Diagram for the Data Structure in Figure 5–1

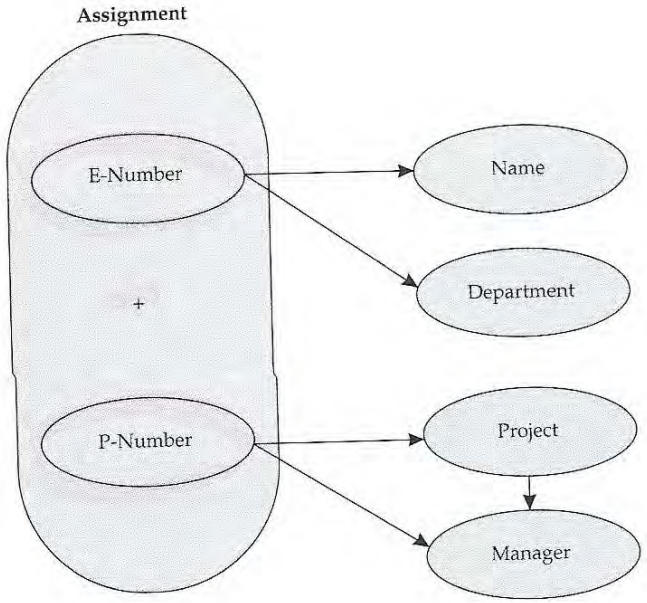


FIGURE 5–6
Dependency Diagram for the Data Structure in Figure 5–2

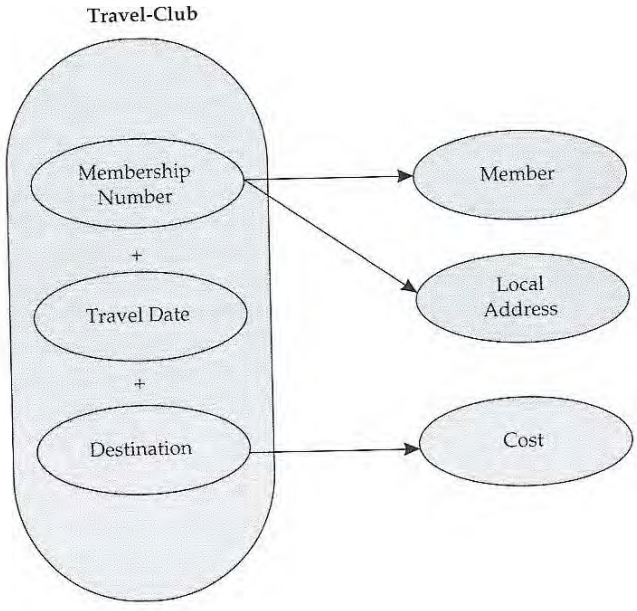


FIGURE 5–7 Dependency Diagram for the Data Structure in Figure 5–3

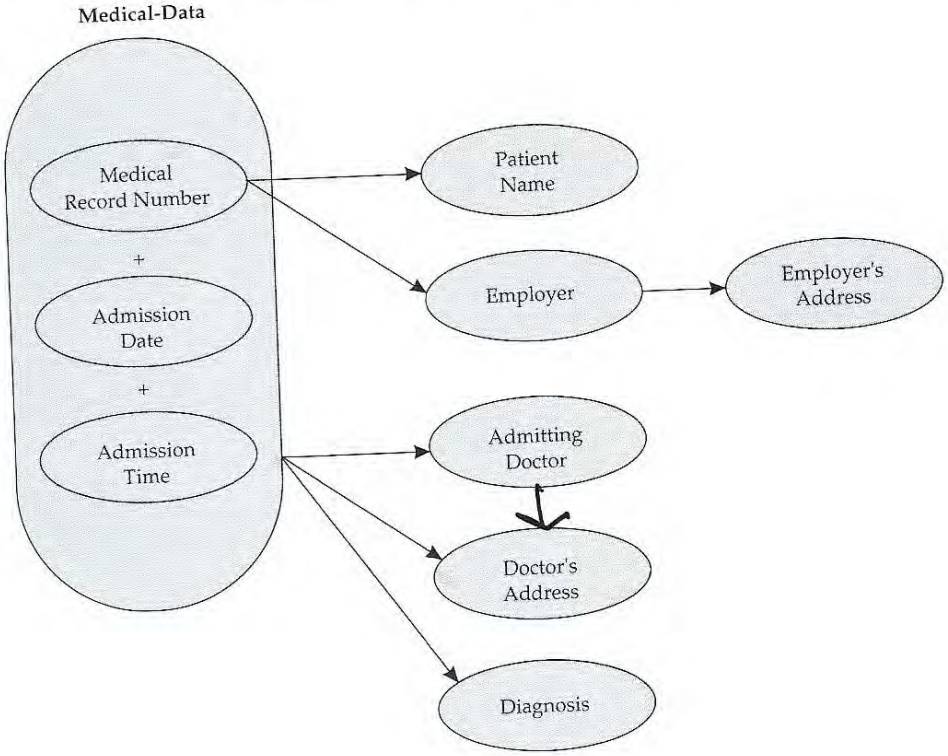


FIGURE 5–8
Dependency Diagram for the Data Structure in Figure 5–4

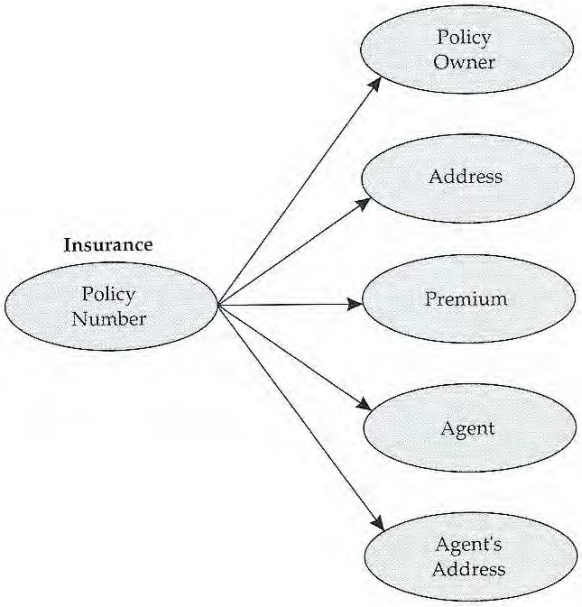


FIGURE 5–9
Normalized Relations for the Data in Figure 5–1

ASSIGNMENT

E-Number	P-Number
1001	26716
1001	23835
1002	43873
1003	26716
1003	34761

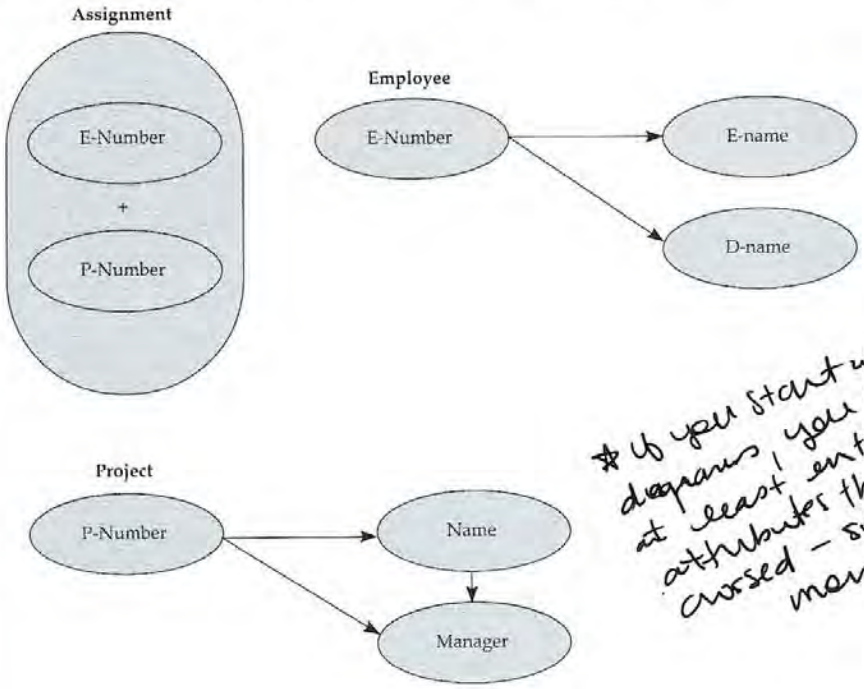
EMPLOYEE

E-Number	Name	Department
1001	Adams	Accounting
1002	Baker	Finance
1003	Clarke	Accounting

PROJECT

P-Number	Project	Manager
26716	New billing system	Yates
23835	Mailing list maintenance	Kanter
43873	Identify new investments	Cody
34761	Purchasing system design	Yates

FIGURE 5–10 Revised Dependency Diagrams for the Data Structures in Figure 5–9



* If you start w/ E-R diagrams, you have at least 1 entities & attributes that aren't crossed – somewhat normalized

FIGURE 5–11 Normalized Relations for the Data in Figure 5–2

MEMBER-DATA

Membership Number	Member	Local Address
3246	W. Earp	Laramie
5498	B. Jones	Los Angeles
8730	R. Steele	Hollywood
0653	D. Tracy	New York
6593	J. Friday	San Francisco

TRAVEL-PLANS

Membership Number	Destination	Travel Date
3246	London	7/10
5498	Madrid	7/12
8730	Los Angeles	6/30
0653	Madrid	7/05
6593	Los Angeles	7/04

TRAVEL-COSTS

Destination	Cost
London	$1200
Madrid	$1500
Los Angeles	$ 900

FIGURE 5–12 Revised Dependency Diagrams for the Data Structures in Figure 5–11

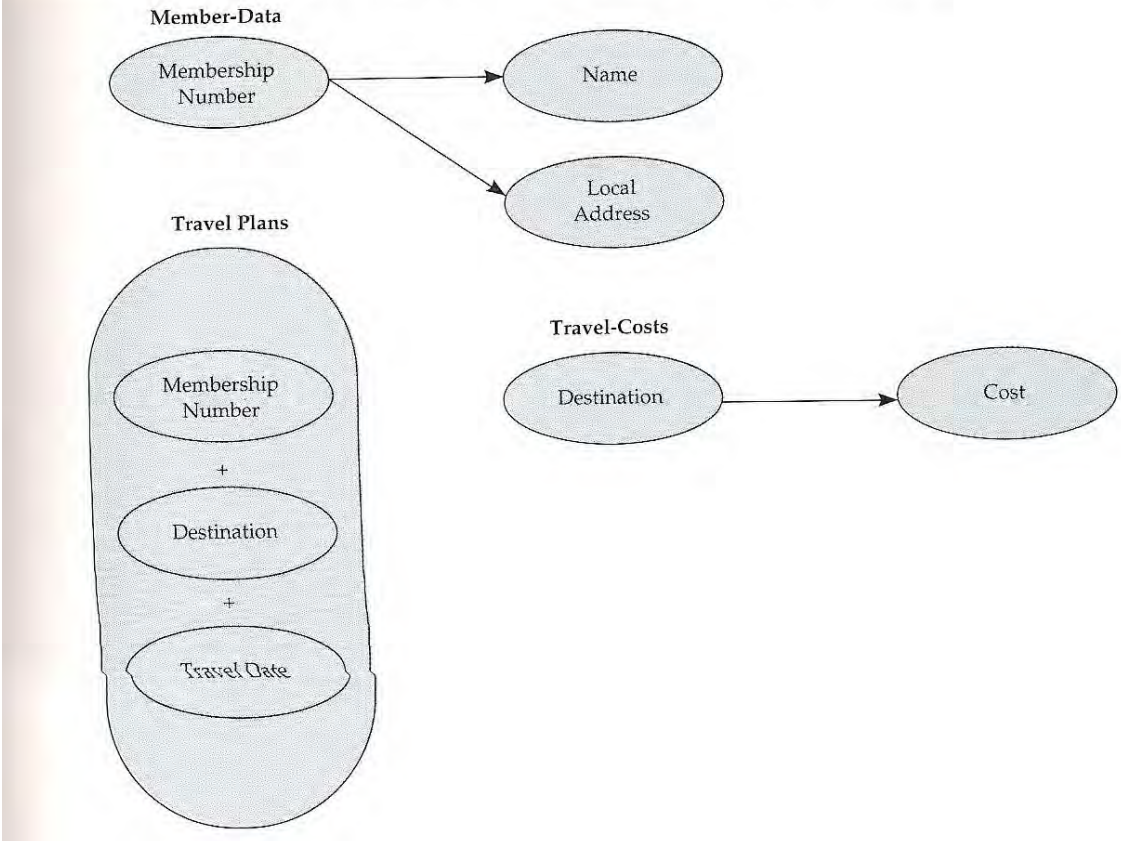


FIGURE 5–13 Normalized Relations for the Data in Figure 5–3

ADMISSION

Medical Record Number	Admission Date	Admission Time	Diagnosis	Admitting Doctor
76154	7/09	6:04	Cardiac Arrest	Barnum
65429	7/09	10:46	Heat Exhaustion	Fraser
27635	7/09	13:20	Stress: job	Barnum
65283	7/09	18:14	Childbirth	Barnum

DOCTORS

Admitting Doctor	Doctor's Address
Barnum	129 Medical Pkwy.
Fraser	104 Medical Pkwy.

EMPLOYER

Patient's Employer	Employer's Address
Billco	3562 S. Industrial Blvd.
P & R Co.	1100 Northcreek Suite 200
Gaines HS	1209 Washington Ave.

PATIENT

Medical Record Number	Patient Name
76154	Grimes
65429	Robinson
27635	Lisauckis
65283	Hargrove

EMPLOYED-BY

Medical Record Number	Patient's Employer
76154	Billco
65429	P & R Co.
27635	Gaines HS
65283	N/A

FIGURE 5–14 Revised Dependency Diagrams for the Data Structures in Figure 5–13

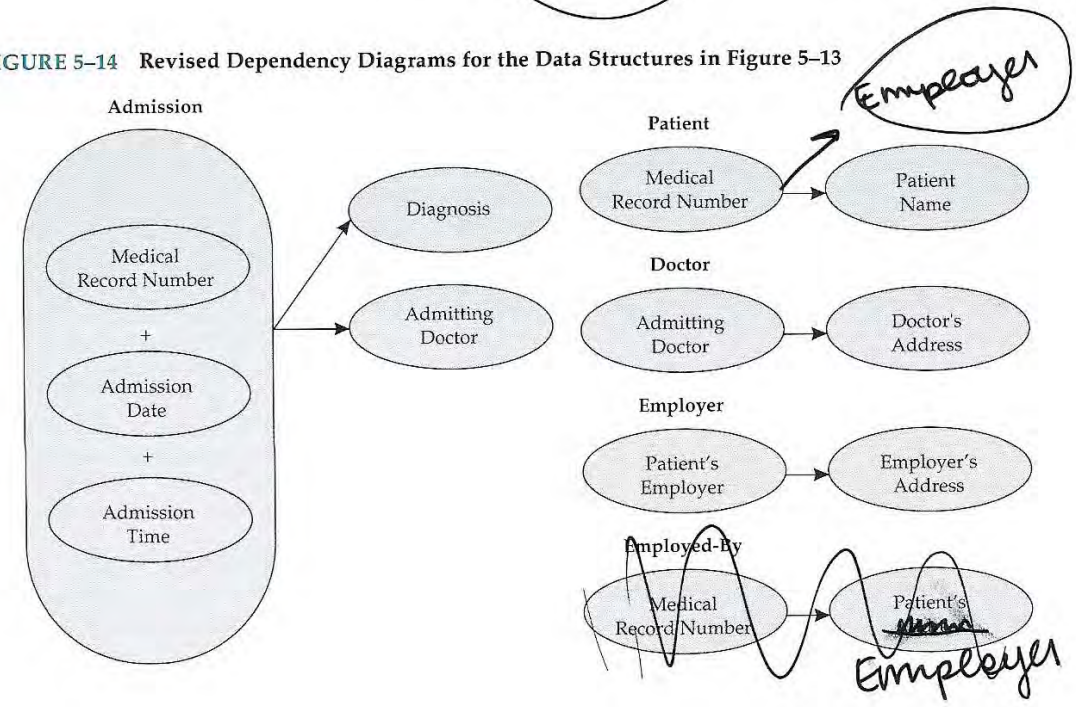
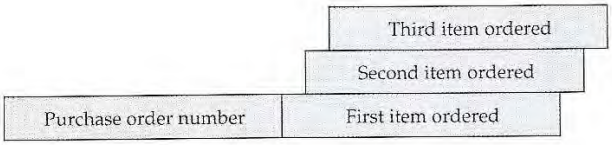


FIGURE 5–15a
A Nonflat File Structure



(a)

FIGURE 5–15b
The Flat File Equivalent of Figure 5–15a

Purchase order number	First item ordered
Purchase order number	Second item ordered
Purchase order number	Third item ordered

(b)

FIGURE 5–16a
A Relation in Third Normal Form But Not in Boyce-Codd Normal Form

FLIGHT

Customer	Destination	Carrier
1283	Austin	Friendly
2873	Dallas	Speedy
7625	Austin	Friendly
6155	Austin	Reliable
1283	Dallas	Speedy

(Note: Carrier → Destination)

FIGURE 5–16b
A Boyce-Codd Normalization of Figure 5–16a

RESERVATION

Customer	Carrier
1283	Friendly
2873	Speedy
7625	Friendly
6155	Reliable
1283	Speedy

OPTION

Carrier	Destination
Friendly	Austin
Speedy	Dallas
Reliable	Austin

TABLE 5–1 A Summary of the Normal Form Criteria through Boyce-Codd Normal Form (Note that each normal form builds on the prior one)

Normal Form	Criteria
First normal form	The data structure is representable as a flat file.
Second normal form	The data structure is in first normal form and all non-key attributes are fully functionally dependent on the primary key.
Third normal form	The data structure is in second normal form and there are no transitive dependencies.
Boyce-Codd normal form	The data structure is in third normal form and every determinant is a candidate key.

FIGURE 5–17
Relation Containing Multiple Values

COMPUTER-ACCOUNT

Employee	Project	User-Code
100	23796	User100
100	34548	User100
102	23487	User102

FIGURE 5–18
A Tuple Has Been Added to the Data
Does employee 100 really use two computer accounts for project 23796 and only one account for project 34548?

COMPUTER-ACCOUNT

Employee	Project	User-Code
100	23796	User100
100	34548	User100
102	23487	User102
100	23796	User100A

FIGURE 5–19
Relation Containing Multiple Values

MUSIC-AUTO-DATA

Name	Music Preference	Automobile Owned
Clarke	Jazz	Sports car
Clarke	Rock	Sports car
Clarke	Jazz	Truck
Clarke	Rock	Truck

FIGURE 5–20
An Additional Tuple Occurrence Demonstrates a Problem Caused by Multiple, Independent Values
Does Clarke really enjoy only jazz music only while in the van?

MUSIC-AUTO-DATA

Name	Music Preference	Automobile Owned
Clarke	Jazz	Sports car
Clarke	Rock	Sports car
Clarke	Jazz	Truck
Clarke	Rock	Truck
Clarke	Jazz	Van

FIGURE 5–21
The Relation of Figure 5–20
A second tuple is needed to maintain logical consistency.

MUSIC-AUTO-DATA

Name	Music Preference	Automobile Owned
Clarke	Jazz	Sports car
Clarke	Rock	Sports car
Clarke	Jazz	Truck
Clarke	Rock	Truck
Clarke	Jazz	Van
Clarke	Rock	Van

FIGURE 5–22
The Multiple Values in This Data Are Related, So There Are No Anomalies

MUSIC-SKILL

Employee	Skill	Music Type
Borst	Composer	Classical
Borst	Composer	Jazz
Borst	Composer	Rock
Borst	Critic	Classical
Myles	Composer	Classical
Myles	Composer	Jazz

FIGURE 5–23
A Typical Relation Used in Project Management

ASSIGNMENT

Employee	Manager	Project
Adams	Yates	26197
Baker	Kanter	21051
Clarke	Yates	23438
Dexter	Barnes	34687

FIGURE 5–24
An Apparently Harmless Decomposition of the Relation in Figure 5–23

EMPLOYEE

Name	Manager
Adams	Yates
Baker	Kanter
Clarke	Yates
Dexter	Barnes

MANAGER

Name	Project
Yates	26197
Yates	23438
Kanter	21051
Barnes	34687

FIGURE 5–25
The Result of Joining the Relations in Figure 5–24
Note that tuples that did not exist in the original relation have been created.

ASSIGNMENT

Employee	Manager	Project
Adams	Yates	26197
Adams	Yates	23438
Baker	Kanter	21051
Clarke	Yates	23438
Clarke	Yates	26197
Dexter	Barnes	34687

FIGURE 5–26
A Set of Nonloss Projections of the Data in Figure 5–23

PROJECT-STAFF

Employee	Project
Adams	26197
Baker	21051
Clarke	23438
Dexter	34687

PROJECT-MANAGEMENT

Project	Manager
26197	Yates
21051	Kanter
23438	Yates
34687	Barnes

express relationships by foreign keys

FIGURE 5–27a
A Relation in Domain-Key Normal Form

PROJECT

Name	P-Number	Manager	Actual Cost	Expected Cost
New billing system	23760	Yates	1000	10000
Common stock issue	28765	Baker	3000	4000
Resolve bad debts	26713	Kanter	2000	1500
New office lease	26511	Yates	5000	5000

Key is: P-Number
Domain of Actual cost is {x : x > = 0}
Domain of Expected Cost is {x : x > = 0}

FIGURE 5–27b
A Relation that Violates Domain-Key Normal Form Key Dependence

PROJECT

Name	P-Number	Manager	Actual Cost	Expected Cost
New billing system	23760	Yates	1000	10000
Common stock issue	28765	Baker	3000	4000
Resolve bad debts	26713	Kanter	2000	1500
New office lease	26511	Yates	5000	5000
New billing system	23760	Baker	1000	10000

FIGURE 5–27c
Domain-Key Normal Form
A Relation that Violates Domain-Key Normal Form Domain Dependence

PROJECT

Name	P-Number	Manager	Actual Cost	Expected Cost
New billing system	23760	Yates	1000	10000
Common stock issue	28765	Baker	3000	4000
Resolve bad debts	26713	Kanter	2000	1500
New office lease	26511	Yates	5000	5000
New ad campaign	34004	Kanter	0	−1000

FIGURE 5–2 **TRAVEL-CLUB Relation**

TRAVEL-CLUB

Membership Number	Member	Local Address	Destination	Cost	Travel Date
3246	W. Earp	Laramie	London	$1200	7/10
5498	B. Jones	Los Angeles	Madrid	$1500	7/12
8730	R. Steele	Hollywood	Los Angeles	$ 900	6/30
0653	D. Tracy	New York	Madrid	$1500	7/05
6593	J. Friday	San Francisco	Los Angeles	$ 900	7/04

FIGURE 5–3 **MEDICAL Relation**

MEDICAL-DATA

Medical Record Number	Patient Name	Admission Date	Admission Time	Diagnosis	Admitting Doctor	Doctor's Address	Patient's Employer	Employer's Address
76154	Grimes	7/09	6:04	Cardiac Arrest	Barnum	129 Medical Pkwy.	Bilco	3562 S. Industrial Blvd.
65429	Robinson	7/09	10:46	Heat Exhaustion	Fraser	104 Medical Pkwy.	P & R Co.	1100 Northcreek, Ste. 200
27635	Lisauckis	7/09	13:20	Stress:Job	Barnum	129 Medical Pkwy.	Gaines HS	1209 Washington Ave.
55283	Hargrove	7/09	18:14	Childbirth	Barnum	129 Medical Pkwy.	N/A	N/A

FIGURE 5–4 **INSURANCE Relation**

Insurance Policy Owner	Address	Policy Number	Agent	Agent's Address	Premium
Smith	124 Main St.	8906253	Brown	505 Georgia Ave.	$300
Lucas	1704 Hadlock St.	7856253	Jones	818 Enfield St.	$450
Murphy	1210 Marcel Ave.	3482674	Brown	505 Georgia Ave.	$380
Wade	750 Leslie Ave.	6282748	Brown	707 Texas Ave.	$510
Hammock	31 Venetian Dr.	7621938	Jones	818 Enfield St.	$280