Appendix H
Database Design

Introduction

This appendix presents a method for designing a database to satisfy a set of requirements. In designing a database, you must identify the tables in the database, the columns in the tables, the primary keys of the tables, and the relationships between the tables.

The appendix begins by examining some important concepts concerning relational databases and then presents the design method. To illustrate the process, the appendix presents the requirements for the JSP Recruiters database. The appendix then applies the design method to those requirements to produce the database design. The appendix concludes by examining normalization, a process that you can use to identify and fix potential problems in database designs.

Relational Databases

A **relational database** is a collection of tables similar to the tables for JSP Recruiters that appear in Figure H–1 on the next page. The Client table contains information about the clients to which JSP Recruiters provides recruiting services. JSP assigns each client to a specific recruiter. The Recruiter table contains information about the recruiters to whom these clients are assigned.

The Seminar table lists the specific seminars that the recruiters at JSP Recruiters offer to their clients. Each seminar has a number and a description. The table also includes the number of hours for which the seminar usually is offered and the increments, that is, the standard time blocks, in which the seminar usually is offered. The first row, for example, indicates that seminar S01 is Assessing Patient Satisfaction. The seminar typically is offered in 2-hour increments for a total of 8 hours.

The Seminar Offerings table contains a client number, a seminar number, the total number of hours for which the seminar is scheduled, and the number of hours the client already has spent in the seminar. The second record shows that client BH72 currently has scheduled seminar S03 (Medical Powers of Attorney). The seminar is scheduled for 12 hours, of which 6 hours already have been spent.

The formal term for a table is relation. A **relation** essentially is a two-dimensional table. If you study the tables shown in Figure H–1 on the next page, you might see that there are certain restrictions you can place on relations. Each column in a table should have a unique name, and entries in each column should match this column name. For example, in the Postal Code column, all entries in fact should *be* postal codes. In addition, each row should be unique. After all, if two rows in a table contain identical data, the second row does not provide any information that you do not already have. In addition, for maximum flexibility, the order in which columns and rows appear in a table should be immaterial. Finally, a table's design is less complex if you restrict each position in the table to a single entry; that is, you do not permit multiple entries (often called **repeating groups**) in the table. These restrictions lead to the following definition:

Client

Client Number	Client Name	Street	City	State	Postal Code	Amount Paid	Current Due	Recruiter Number
AC34	Alys Clinic	134 Central	Berridge	CO	80330	$0.00	$17,500.00	21
BH72	Berls Hospital	415 Main	Berls	CO	80349	$29,200.00	$0.00	24
FD89	Ferb Dentistry	34 Crestview	Berridge	CO	80330	$21,000.00	$12,500.00	21
FH22	Family Health	123 Second	Tarleton	CO	80409	$0.00	$0.00	24
MH56	Munn Hospital	76 Dixon	Mason	CO	80356	$0.00	$43,025.00	24
PR11	Peel Radiology	151 Valleyview	Fort Stewart	CO	80336	$31,750.00	$0.00	21
RM32	Roz Medical	315 Maple	Berls	CO	80349	$0.00	$0.00	27
TC37	Tarleton Clinic	451 Hull	Tarleton	CO	80409	$18,750.00	$31,500.00	27
WL56	West Labs	785 Main	Berls	CO	80349	$14,000.00	$0.00	24

Recruiter

Recruiter Number	Last Name	First Name	Street	City	State	Postal Code	Rate	Commission
21	Kerry	Alyssa	261 Pointer	Tourin	CO	80416	0.10	$17,600.00
24	Reeves	Camden	3135 Brill	Denton	CO	80412	0.10	$19,900.00
27	Fernandez	Jaime	265 Maxwell	Charleston	CO	80380	0.09	$9,450.00
34	Lee	Jan	1827 Oak	Denton	CO	80413	0.08	$0.00

Seminar

Seminar Number	Seminar Description	Hours	Increments
S01	Assessing Patient Satisfaction	8	2
S02	HIPAA Fundamentals	4	2
S03	Medical Powers of Attorney	12	4
S04	OSHA Fundamentals	4	2
S05	Using Basic Medical Terminology	16	4
S06	Working in Teams	16	4
S07	Coping with Crisis Situations	8	2
S08	Personal Hygiene in the Medical Environment	2	1

Seminar Offerings

Client Number	Seminar Number	Total Hours	Hours Spent
BH72	S02	4	0
BH72	S03	12	6
FH22	S04	4	2
FH22	S07	8	4
MH56	S06	16	8
MH56	S07	8	4
PR11	S05	16	4
TC37	S01	8	2
TC37	S07	10	2
TC37	S08	2	0

Figure H–1

A relation is a two-dimensional table in which:

1. The entries in the table are single-valued; that is, each location in the table contains a single entry.
2. Each column has a distinct name (technically called the attribute name).
3. All values in a column are values of the same attribute (that is, all entries must correspond to the column name).
4. The order of columns is immaterial. You can view the columns in any order you want.

5. Each row is distinct; that is, no two rows are identical.
6. The order of rows is immaterial. You can view the rows in any order you want.

A relational database is a collection of relations. Rows in a table (relation) often are called **records** or **tuples**. Columns in a table (relation) often are called **fields** or **attributes**.

To depict the structure of a relational database, you can use a commonly accepted shorthand representation: you write the name of the table and then, within parentheses, list all of the columns in the table. Each table should begin on a new line. If the entries in the table occupy more than one line, the entries that appear on the next line should be indented so it is clear that they do not constitute another table. Using this method, you would represent the JSP Recruiters database as follows:

Client (Client Number, Client Name, Street, City, State, Postal Code, Amount Paid, Current Due, Recruiter Number)
Recruiter (Recruiter Number, Last Name, First Name, Street, City, State, Postal Code, Rate, Commission)
Seminar (Seminar Number, Seminar Description, Hours, Increments)
Seminar Offerings (Client Number, Seminar Number, Total Hours, Hours Spent)

The JSP Recruiters database contains some duplicate column names. For example, the Recruiter Number column appears in *both* the Recruiter table *and* the Client table. Suppose a situation exists wherein the two columns might be confused. If you write Recruiter Number, how would the computer or another individual know which Recruiter Number column in which table you intended to use? When duplicate column names exist in a database, you need to indicate the column to which you are referring. One common approach to this problem is to write both the table name and the column name, separated by a period. Thus, you would write the Recruiter Number column in the Client table as Client.Recruiter Number and the Recruiter Number column in the Recruiter table as Recruiter.Recruiter Number. Technically, when you combine a column name with a table name, you say that you **qualify** the column names. It *always* is acceptable to qualify column names, even if there is no possibility of confusion. If confusion may arise, however, it is *essential* to qualify column names.

Functional Dependence

In the JSP Recruiters database (Figure H–1), a given client number in the database will correspond to a single client because client numbers are unique. Thus, you could look up a client number and find a single name that corresponds to it. No ambiguity exists. If you know a value for an attribute guarantees that you also know a single value for a second attribute, the second attribute is said to be **functionally dependent** on the first. Thus, Client Name is functionally dependent on Client Number because if you know a value for Client Number, you automatically know a single value for Client Name. In other words, Client Number determines Client Name. This often is indicated as Client Number → Client Name. The attribute following the arrow is functionally dependent on the attribute preceding the arrow.

If you were given a city and asked to find a single client's name, however, you could not do it. Given Berridge as the city, for example, you would find two client names (Alys Clinic and Ferb Dentistry). Formally, you would say the Client Name is *not* functionally dependent on City.

In the Recruiter table, Last Name is functionally dependent on Recruiter Number. If you are given a value for Recruiter Number, for example 24, you always will find a *single* last name, in this case Reeves, associated with it.

In the Client table, Client Name is not functionally dependent on Recruiter Number. Given Recruiter Number 21, for example, you would not be able to find a single client name, because 21 appears on more than one row in the table.

In the Seminar Offerings table, Hours Spent is not functionally dependent on Client Number. Client Number does not give enough information. For example, in the first row, Client Number is BH72 and Hours Spent is 0. In the second row, however, the Client Number also is BH72, but the Hours Spent is 6. Hours Spent also is not functionally dependent on Seminar Number, because Seminar Number does not give enough information. For example, in the fourth row, Seminar Number is S07 and Hours Spent is 4. In the ninth row, however, Seminar Number also is S07, but Hours Spent is 2.

Hours Spent actually is functionally dependent on the combination (formally called the **concatenation**) of Client Number and Seminar Number. Given a client number *and* a seminar number, you can determine a single value for Hours Spent. This would be written as Client Number, Seminar Number → Hours Spent.

Primary Key

The **primary key** of a table (relation) is the column or minimum collection of columns that uniquely identifies a given row in that table. In the Recruiter table, the recruiter's number uniquely identifies a given row. Any recruiter number appears on only one row of the table. Thus, Recruiter Number is the primary key. Similarly, Client Number is the primary key of the Client table and Seminar Number is the key of the Seminar table.

The primary key of the Seminar Offerings table consists of two columns, Client Number and Seminar Number. Total Hours and Hours Spent neither are dependent on just Client Number nor are they dependent on just Seminar Number. Rather, they are dependent on the combination of Client Number and Seminar Number. Thus, neither Client Number nor Seminar Number alone can be the primary key. Both columns are required.

The primary key provides an important way of distinguishing one row in a table from another. In the shorthand representation, you underline the column or collection of columns that comprise the primary key for each table in the database. Thus, the complete shorthand representation for the JSP Recruiters database is as follows:

Client (Client Number, Client Name, Street, City, State, Postal Code, Amount Paid, Current Due, Recruiter Number)

Recruiter (Recruiter Number, Last Name, First Name, Street, City, State, Postal Code, Rate, Commission)

Seminar (Seminar Number, Seminar Description, Hours, Increments)

Seminar Offerings (Client Number, Seminar Number, Total Hours, Hours Spent)

Occasionally (but not often) there might be more than one possibility for the primary key. For example, if the JSP Recruiters database included the recruiter's Social Security number in the Recruiter table, either the recruiter number or the Social Security number could serve as the primary key. In this case, both columns are referred to as candidate keys. Similarly to a primary key, a **candidate key** is a column or collection of columns on which all columns in the table are functionally dependent — the definition for primary key really defines candidate key as well. From all the candidate keys, the user chooses one to be the primary key.

Database Design

This section presents a specific database design method, given a set of requirements that the database must support. The section then presents a sample of such requirements and illustrates the design method by designing a database to satisfy these requirements.

Design Method

The following steps illustrate how to design a database for a set of requirements.

1. Read through the requirements and identify the entities (objects) involved. Assign names to the entities. If, for example, the design involves departments and employees, you could assign the names Department and Employee. If the design involves customers, orders, and parts, you could assign the names Customer, Order, and Part.
2. Identify a unique identifier for each entity. For example, if one of the entities is parts, you would determine what it takes to uniquely identify each individual part. In other words, what enables the organization to distinguish one part from another? For a part entity, it may be Part Number. For a customer entity, it may be Customer Number. If there is no such unique identifier, it probably is a good idea to add one. Perhaps the previous system was a manual one in which customers were not assigned numbers, in which case this would be a good time to add customer numbers to the system. If there is no natural candidate for a primary key, you can add an AutoNumber field (like the ID field that Access automatically adds when you create a new table).
3. Identify the attributes for all the entities. These attributes will become the columns in the tables. It is possible that more than one entity has the same attribute. At JSP Recruiters, for example, clients and recruiters both have the attributes of street address, city, state, and postal code. To clarify this in your planning, you can follow the name of the attribute with the corresponding entity in parentheses. Thus, Street (Client) would be the street address of a client, whereas Street (Recruiter) would be the street address of a recruiter.
4. Identify the functional dependencies that exist among the attributes.
5. Use the functional dependencies to identify the tables. You do this by placing each attribute with the attribute or minimum combination of attributes on which it is functionally dependent. The attribute or attributes on which all other attributes in the relation are dependent will be the primary key of a relation. The remaining attributes will be the other columns in the relation. Once you have determined all the columns in the relation, you can assign an appropriate name to the relation.
6. Identify any relationships between tables by looking for matching columns.

The following sections illustrate the design process by designing the database for JSP Recruiters. The next section gives the requirements that this database must support, and the last section creates a database design based on those requirements.

BTW

Physical Database Design

Database design includes both logical design and physical design. Logical database design includes determining the fields (attributes) for the various tables (relations). With physical database design, you need to determine the appropriate data type for each field. For example, should the Client Number field be a Text field or a Number field? What format would be best for each data type? To do so, use the guidelines given on pages AC 10 and 301.

BTW

Boolean Operators

The Boolean operators (AND, OR, and NOT) have a special restriction on the expressions involved. Each expression must be either true or false. When using the Boolean operator AND, for example, the expressions before and after the word AND are examined to determine whether they are true or false. If both are true, the result is true. Otherwise, the result is false. Typically, the expressions involve either a field and a value ([Amount Paid]>20000) or two fields ([Hours Spent]<[Total Hours]). There is one type of field that is appropriate for the Boolean operators without needing to be part of an expression — the Yes/No field. For example, there might be a Yes/No field in the Recruiter table called Bonus that is set to Yes (true) if the recruiter is eligible for a bonus and set to No (false) if the Recruiter is not. This field could be used in a Boolean expression by itself (Bonus AND [Commission]>15000). This expression would be true if Bonus is Yes and the value in the Commission field is greater than 15,000. It would be false otherwise. For details on determining when to use a Yes/No field, see the Plan Ahead on page AC 301.

Requirements for the JSP Recruiters Database

The JSP Recruiters database must support the following requirements:

1. For a client, JSP needs to maintain the client number, name, street address, city, state, postal code, amount paid, and the amount that currently is due. They also need the total amount, which is the sum of the amount already paid and the current due.
2. For a recruiter, store the recruiter number, last name, first name, street address, city, state, postal code, rate, and commission.
3. For a seminar, store the seminar number, seminar description, hours, and increments. In addition, for each offering of the seminar, store the number of the client for whom the seminar is offered, the total number of hours planned for the offering of the seminar, and the number of hours already spent. The total hours may be the same as the normal number of hours for the seminar, but it need not be. This gives JSP the flexibility of tailoring the offering of the seminar to the specific needs of the client.
4. Each client has a single recruiter to which the client is assigned. Each recruiter may be assigned many clients.
5. A client may be offered many seminars and a seminar may be offered to many clients.

Database Design Example

The following represents the application of the design method for the JSP Recruiters requirements.

1. There are three entities: clients, recruiters, and seminars. The names assigned to them are Client, Recruiter, and Seminar, respectively.
2. The unique identifier for clients is the client number. The unique identifier for recruiters is the recruiter number. The unique identifier for seminars is the seminar number. The names assigned to these identifiers are Client Number, Recruiter Number, and Seminar Number, respectively.
3. The attributes are:

   Client Number
   Client Name
   Street (Client)
   City (Client)
   State (Client)
   Postal Code (Client)
   Amount Paid
   Current Due
   Recruiter Number
   Last Name
   First Name
   Street (Recruiter)
   City (Recruiter)
   State (Recruiter)
   Postal Code (Recruiter)
   Rate
   Commission
   Seminar Number

Seminar Description
Hours
Increments
Total Hours
Hours Spent

Remember that parentheses after an attribute indicate the entity to which the attribute corresponds. Thus, Street (Client) represents the street address of a client in a way that distinguishes it from Street (Recruiter), which represents the street address of a recruiter.

Question: Why isn't Total Amount included?

Answer: Total Amount, which is Amount Paid plus Current Due, can be calculated from other columns. You can perform this calculation in queries, forms, and reports. Thus, there is no need to include it as a column in the Client table. Further, by including it, you introduce the possibility of errors in the database. For example, if Amount Paid is $5,000, Current Due is $2,000, and yet you set Total Amount equal to $8,000, you have an error. You also need to be sure to change Total Amount appropriately whenever you change either Amount Paid or Current Due. If Total Amount is not stored, but rather is calculated when needed, you avoid all these problems.

4. The functional dependencies among the attributes are:

Client Number → Client Name, Street (Client), City (Client), State (Client), Postal Code (Client), Amount Paid, Current Due, Recruiter Number

Recruiter Number → Last Name, First Name, Street (Recruiter), City (Recruiter), State (Recruiter), Postal Code (Recruiter), Rate, Commission

Seminar Number → Seminar Description, Hours, Increments

Client Number, Seminar Number → Total Hours, Hours Spent

Question: Why is Total Hours listed with Client Number and Seminar Number rather than just with Seminar Number?

Answer: If the Total Hours was required to be the same as the number of hours for the seminar, then it indeed would be listed with Seminar Number. (It would not vary from one client to another.) Because JSP wants the flexibility of tailoring the number of hours for which a particular seminar is offered to the specific needs of the client, Total Hours also is dependent on Client Number.

The client's name, street address, city, state, postal code, amount paid, and current due are dependent only on Client Number. Because a client has a single recruiter, the recruiter number is dependent on Client Number as well. The recruiter's last name, first name, street address, city, state, postal code, rate, and commission are dependent only on Recruiter Number. A seminar description, the number of hours for the seminar, and the increments in which the seminar is offered are dependent only on Seminar Number. The total hours for a particular seminar offering as well as the hours already spent are dependent on the combination of Client Number and Seminar Number.

5. The tables are:

   Client (Client Number, Client Name, Street, City, State, Postal Code, Amount Paid, Current Due, Recruiter Number)

   Recruiter (Recruiter Number, Last Name, First Name, Street, City, State, Postal Code, Rate, Commission)

   Seminar (Seminar Number, Seminar Description, Hours, Increments)

   Seminar Offerings (Client Number, Seminar Number, Total Hours, Hours Spent)

   The primary keys are underlined.

6. The following are the relationships between the tables:

   a. The Client and Recruiter tables are related using the Recruiter Number fields.

   b. The Client and Seminar Offerings tables are related using the Client Number fields.

   c. The Seminar and Seminar Offerings tables are related using the Seminar Number fields.

Question: Is there a relationship between the Client table and the Seminar table? If so, what type of relationship is it?

Answer: There actually is a many-to-many relationship between the Client table and the Seminar table. A client can schedule many seminars and a seminar can be offered to many clients. The typical way a many-to-many relationship between two tables is implemented is by creating a third table whose primary key is the combination of the primary keys of the individual tables. In this case, the Seminar Offerings table plays exactly that role.

Normalization

After you create your database design, you should analyze it to make sure the design is free of potential problems. To do so, you use a process called normalization. The **normalization** process enables you to identify the existence of potential problems. This process also supplies methods for correcting these problems.

The normalization process involves converting tables into various types of **normal forms**. A table in a particular normal form possesses a certain desirable set of properties. Several normal forms exist, the most common being first normal form (1NF), second normal form (2NF), and third normal form (3NF). The forms create a progression in which a table that is in 1NF is better than a table that is not in 1NF; a table that is in 2NF is better than one that is in 1NF; and so on. The goal of normalization is to take a table or collection of tables and produce a new collection of tables that represents the same information but is free of problems.

First Normal Form

A relation (table) that contains a **repeating group** (or multiple entries for a single row) is called an **unnormalized relation**. Removal of repeating groups is the starting point in the goal for tables that are as free of problems as possible. In fact, in most database management systems, tables cannot contain repeating groups. A table (relation) is in **first normal form** (**1NF**) if it does not contain repeating groups.

In designing a database, you may have created a table with a repeating group. For example, you might have created a Seminar Offerings table, in which the primary key is the Client Number and there is a repeating group consisting of Seminar Number, Total Hours, and Hours Spent. In the example, there is one row per client with Seminar Number, Total Hours, and Hours Spent repeated as many times as necessary (Figure H–2).

Seminar Offerings

Client Number	Seminar Number	Total Hours	Hours Spent
BH72	S02	4	0
	S03	12	6
FH22	S04	4	2
	S07	8	4
MH56	S06	16	8
	S07	8	4
PR11	S05	16	4
TC37	S01	8	2
	S07	10	2
	S08	2	0

Figure H–2

In the shorthand notation, you represent a repeating group by enclosing the repeating group within parentheses. The Seminar Offerings table in Figure H–2 would be represented as:

Seminar Offerings (Client Number, (Seminar Number, Total Hours, Hours Spent))

Conversion to First Normal Form

To convert the table to 1NF, remove the repeating group symbol to give the following:

Seminar Offerings (Client Number, Seminar Number, Total Hours, Hours Spent)

The corresponding example of the new table is shown in Figure H–3.

Seminar Offerings

Client Number	Seminar Number	Total Hours	Hours Spent
BH72	S02	4	0
BH72	S03	12	6
FH22	S04	4	2
FH22	S07	8	4
MH56	S06	16	8
MH56	S07	8	4
PR11	S05	16	4
TC37	S01	8	2
TC37	S07	10	2
TC37	S08	2	0

Figure H–3

Note that the first row of the unnormalized table (Figure H–2 on the previous page) indicates that client BH72 currently is being offered both seminar S02 and seminar S03. In the normalized table, this information is represented by *two* rows, the first and the second (Figure H–3 on the previous page). The primary key for the unnormalized Seminar Offerings table was the Client Number only. The primary key for the normalized table now is the combination of Client Number and Seminar Number.

In general, when converting a non-1NF table to 1NF, the primary key typically will include the original primary key concatenated with the key of the repeating group; that is, the column that distinguishes one occurrence of the repeating group from another within a given row in the table. In this case, Seminar Number is the key to the repeating group and thus becomes part of the primary key of the 1NF table.

Second Normal Form

Even though the following table is in 1NF, problems may exist that will cause you to want to restructure the table. You might have created the following Seminar Offerings table:

Seminar Offerings (Client Number, Client Name, Seminar Number, Seminar Description, Total Hours, Hours Spent)

with the functional dependencies:

Client Number → Client Name

Seminar Number → Seminar Description

Client Number, Seminar Number → Total Hours, Hours Spent

This notation indicates that Client Number alone determines Client Name, Seminar Number alone determines Seminar Description, but it requires *both* a Client Number *and* a Seminar Number to determine either Total Hours or Hours Spent. Consider the sample of this table shown in Figure H–4.

Seminar

Client Number	Client Name	Seminar Number	Seminar Description	Total Hours	Hours Spent
BH72	Berls Hospital	S02	HIPAA Fundamentals	4	0
BH72	Berls Hospital	S03	Medical Powers of Attorney	12	6
FH22	Family Health	S04	OSHA Fundamentals	4	2
FH22	Family Health	S07	Coping with Crisis Situations	8	4
MH56	Munn Hospital	S06	Working in Teams	16	8
MH56	Munn Hospital	S07	Coping with Crisis Situations	8	4
PR11	Peel Radiology	S05	Using Basic Medical Terminology	16	4
TC37	Tarleton Clinic	S01	Assessing Patient Satisfaction	8	2
TC37	Tarleton Clinic	S07	Coping with Crisis Situations	10	2
TC37	Tarleton Clinic	S08	Personal Hygiene in the Medical Environment	2	0

Figure H–4

The name of a specific client, BH72 for example, occurs multiple times in the table. This redundancy causes several problems. It certainly is wasteful of space, but that is not nearly as serious as some of the other problems. These other problems are called **update anomalies** and they fall into four categories:

1. **Update.** A change to the name of client BH72 requires not one change to the table, but several — you must change each row in which BH72 appears. This certainly makes the update process much more cumbersome; it is more complicated logically and takes longer to update.
2. **Inconsistent data.** There is nothing about the design that would prohibit client BH72 from having two or more different names in the database. The first row, for example, might have Berls Hospital as the name, whereas the second row might have Berl Hospital.
3. **Additions.** There is a real problem when you try to add a new seminar and its description to the database. Because the primary key for the table consists of both Client Number and Seminar Number, you need values for both of these to add a new row. If you have a client to add but there are as yet no seminars scheduled for it, what do you use for a seminar number? The only solution would be to make up a dummy seminar number and then replace it with a real seminar number once the client requests a seminar. This is not an acceptable solution.
4. **Deletions.** In Figure H–4, if you delete client BH72 from the database, you also *lose* all the information about seminar S02. For example, you would no longer know that the description of seminar S02 is HIPAA Fundamentals.

These problems occur because there is a column, Client Name, that is dependent on only a portion of the primary key, Client Number, and *not* on the complete primary key. The problem with Seminar Description is that it is dependent on only the Seminar Number. This leads to the definition of second normal form. Second normal form represents an improvement over first normal form because it eliminates update anomalies in these situations. In order to understand second normal form, you need to understand the term nonkey column.

A column is a **nonkey column** (also called a **nonkey attribute**) if it is not a part of the primary key. A table (relation) is in **second normal form** (**2NF**) if it is in first normal form and no nonkey column is dependent on only a portion of the primary key.

Note that if the primary key of a table contains only a single column, the table automatically is in second normal form. In that case, there could not be any column dependent on only a portion of the primary key.

Conversion to Second Normal Form

To correct the problems, convert the table to a collection of tables in second normal form. Then name the new tables. The following is a method for performing this conversion.

First, take each subset of the set of columns that make up the primary key, and begin a new table with this subset as its primary key. For the Seminar Offerings table, this would give:

(Client Number)

(Seminar Number)

(Client Number, Seminar Number)

Next, place each of the other columns with the appropriate primary key; that is, place each one with the minimal collection of columns on which it depends. For the Seminar Offerings table, this would yield:

(Client Number, Client Name)

(Seminar Number, Seminar Description)

(Client Number, Seminar Number, Total Hours, Hours Spent)

Each of these new tables now can be given a name that is descriptive of the meaning of the table, such as Client, Seminar, and Seminar Offerings. Figure H–5 shows samples of the tables involved.

Client

Client Number	Client Name
BH72	Berls Hospital
FH22	Family Health
MH56	Munn Hospital
PR11	Peel Radiology
TC37	Tarleton Clinic

Seminar

Seminar Number	Seminar Description
S01	Assessing Patient Satisfaction
S02	HIPAA Fundamentals
S03	Medical Powers of Attorney
S04	OSHA Fundamentals
S05	Using Basic Medical Terminology
S06	Working in Teams
S07	Coping with Crisis Situations
S08	Personal Hygiene in the Medical Environment

Seminar Offerings

Client Number	Seminar Number	Total Hours	Hours Spent
BH72	S02	4	0
BH72	S03	12	6
FH22	S04	4	2
FH22	S07	8	4
MH56	S06	16	8
MH56	S07	8	4
PR11	S05	16	4
TC37	S01	8	2
TC37	S07	10	2
TC37	S08	2	0

Figure H–5

The new design eliminates the update anomalies. A client name occurs only once for each client, so you do not have the redundancy that you did in the earlier design. Changing the name of a client now is a simple process involving a single change. Because the name of a client occurs in a single place, it is not possible to have multiple names for the same client in the database at the same time.

To add a new client, you create a new row in the Client table and thus there is no need to have a seminar offering already scheduled for that client. Also, deleting client BH72 has nothing to do with the Seminar table and, consequently, does not cause seminar S02 to be deleted. Thus, you still have its description (HIPAA Fundamentals) in the database. Finally, you have not lost any information in the process. The data in the original design can be reconstructed from the data in the new design.

Third Normal Form

Problems still can exist with tables that are in 2NF as illustrated in the following Client table:

Client (Client Number, Client Name, Street, City, State, Postal Code, Amount Paid, Current Due, Recruiter Number, Last Name, First Name)

The functional dependencies in this table are:

Client Number → Client Name, Street, City, State, Postal Code, Amount Paid, Current Due, Recruiter Number

Recruiter Number → Last Name, First Name

Client Number determines all the other columns. In addition, Recruiter Number determines Last Name and First Name.

Because the primary key of the table is a single column, the table automatically is in second normal form. As the sample of the table shown in Figure H–6 demonstrates, however, this table has problems similar to those encountered earlier, even though it is in 2NF. In this case, it is the last name and first name of a recruiter that can occur many times in the table (see recruiter 21, Alyssa Kerry, for example).

BTW

3NF

The actual definition of third normal form is that no nonkey column is functionally dependent on any other nonkey column. The relation in Figure H-6 also violates this definition because the nonkey column Last Name is functionally dependent on the nonkey column Recruiter Number.

Client

Client Number	Client Name	...	Amount Paid	Current Due	Recruiter Number	Last Name	First Name
AC34	Alys Clinic	...	$0.00	$17,500.00	21	Kerry	Alyssa
BH72	Berls Hospital	...	$29,200.00	$0.00	24	Reeves	Camden
FD89	Ferb Dentistry	...	$21,000.00	$12,500.00	21	Kerry	Alyssa
FH22	Family Health	...	$0.00	$0.00	24	Reeves	Camden
MH56	Munn Hospital	...	$0.00	$43,025.00	24	Reeves	Camden
PR11	Peel Radiology	...	$31,750.00	$0.00	21	Kerry	Alyssa
RM32	Roz Medical	...	$0.00	$0.00	27	Fernandez	Jaime
TC37	Tarleton Clinic	...	$18,750.00	$31,500.00	27	Fernandez	Jaime
WL56	West Labs	...	$14,000.00	$0.00	24	Reeves	Camden

Figure H–6

This redundancy results in the same set of problems described previously with the Seminar Offerings table. In addition to the problem of wasted space, you have similar update anomalies, as follows:

1. **Updates.** A change to the name of a recruiter requires not one change to the table, but several. Again the update process becomes very cumbersome.
2. **Inconsistent data.** There is nothing about the design that would prohibit a recruiter from having two different names in the database. On the first row, for example, the name for recruiter 21 might read Alyssa Kerry; whereas on the third row (another row on which the recruiter number is 21), the name might be Stacy Webb.

3. **Additions.** In order to add recruiter 39, whose name is Tati Angelo, to the database, she must have at least one client. If she has not yet been assigned any clients, either you cannot record the fact that her name is Tati Angelo or you have to create a fictitious client for her to represent. Again, this is not a desirable solution to the problem.
4. **Deletions.** If you were to delete all the clients of recruiter 24 from the database, then you also would lose all information concerning recruiter 24.

These update anomalies are due to the fact that Recruiter Number determines Last Name and First Name, but Recruiter Number is not the primary key. As a result, the same Recruiter Number and consequently the same Last Name and First Name can appear on many different rows.

You have seen that 2NF is an improvement over 1NF, but to eliminate 2NF problems, you need an even better strategy for creating tables in the database. Third normal form provides that strategy. Before looking at third normal form, however, you need to become familiar with the special name that is given to any column that determines another column (like Recruiter Number in the Client table).

Any column (or collection of columns) that determines another column is called a **determinant**. Certainly the primary key in a table is a determinant. In fact, by definition, any candidate key is a determinant. (Remember that a candidate key is a column or collection of columns that could function as the primary key.) In this case, Recruiter Number is a determinant, but it certainly is not a candidate key, and that is the problem.

A table is in **third normal form** (**3NF**) if it is in second normal form and if the only determinants it contains are candidate keys. This definition is not the original definition of third normal form. This more recent definition, which is preferable to the original, also is referred to as **Boyce-Codd normal form** (**BCNF**). However, in this text, it simply is referred to as third normal form.

Conversion to Third Normal Form

You now have identified the problem with the Client table: it is not in 3NF. You now need a scheme to correct the deficiency in the Client table and in all tables having similar deficiencies. Such a method follows.

First, for each determinant that is not a candidate key, remove from the table the columns that depend on this determinant, but do not remove the determinant. Next, create a new table containing all the columns from the original table that depend on this determinant. Finally, make the determinant the primary key of this new table.

In the Client table, for example, Last Name and First Name are removed because they depend on the determinant Recruiter Number, which is not a candidate key. A new table is formed, consisting of Recruiter Number as the primary key, Last Name and First Name. Specifically:

Client (Client Number, Client Name, Street, City, State, Postal Code,
Amount Paid, Current Due, Recruiter Number, Last Name, First Name)

is replaced by:

Client (Client Number, Client Name, Street, City, State, Postal Code,
Amount Paid, Current Due, Recruiter Number)

and

Recruiter (Recruiter Number, Last Name, First Name)

Figure H–7 shows samples of the tables involved.

Client

Client Number	Client Name	...	Amount Paid	Current Due	Recruiter Number
AC34	Alys Clinic	...	$0.00	$17,500.00	21
BH72	Berls Hospital	...	$29,200.00	$0.00	24
FD89	Ferb Dentistry	...	$21,000.00	$12,500.00	21
FH22	Family Health	...	$0.00	$0.00	24
MH56	Munn Hospital	...	$0.00	$43,025.00	24
PR11	Peel Radiology	...	$31,750.00	$0.00	21
RM32	Roz Medical	...	$0.00	$0.00	27
TC37	Tarleton Clinic	...	$18,750.00	$31,500.00	27
WL56	West Labs	...	$14,000.00	$0.00	24

Recruiter

Recruiter Number	Last Name	First Name
21	Kerry	Alyssa
24	Reeves	Camden
27	Fernandez	Jaime

Figure H–7

This design corrects the previously identified problems. A recruiter's name appears only once, thus avoiding redundancy and making the process of changing a recruiter's name a very simple one. It is not possible with this design for the same recruiter to have two different names in the database. To add a new recruiter to the database, you add a row in the Recruiter table so it is not necessary to have a pre-existing client for the recruiter. Finally, deleting all the clients of a given recruiter will not remove the recruiter's record from the Recruiter table, so you retain the recruiter's name; all the data in the original table can be reconstructed from the data in the new collection of tables. All previously mentioned problems indeed have been solved.

In the Lab

Design, create, modify, and/or use a database following the guidelines, concepts, and skills presented in this appendix.

Lab 1: Designing a Database

Instructions: Answer the following questions in the format specified by your instructor.

1. List three relational table characteristics that are violated in the table shown in Figure H–8.

Orders

Date	Part Number	Number Ordered	Date
11/01/2008	AX12	11	11/12/2008
11/01/2008	BT04 BZ66	1 1	11/09/2008 Friday
11/03/2008	CB03	4	11/13/2008
11/03/2008	CX11	2	11/12/2008
11/04/2008	AZ52 BA74	2 4	11/12/2008 $182.91
11/04/2008	BT04	1	11/15/2008
11/03/2008	CX11	2	11/12/2008
11/04/2008	CZ81	2	11/13/2008

Figure H–8

2. The following table is a student's first attempt to create a database design:

   Student (StudentID, Student Name, Credit Hours, AdvisorID, Advisor Name, (CourseID, Course Name, Grade))

   a. Identify the functional dependencies.

   b. Convert this table to an equivalent collection of tables in 3NF.

3. The following table concerns invoice information. For a given invoice (identified by invoice number), there will be a single customer. The customer's number, name, and address appear on the invoice as well as the invoice date. Also, there may be several different items appearing on the invoice. For every item that appears, the item number, description, price, and number shipped will be displayed. Convert this table into an equivalent collection of tables in 3NF.

   Invoice (Invoice Number, Customer Number, Customer Name,

   Street Address, City, State, Postal Code, Country, Invoice Date, (Item Number,

   Item Description, Price, Number Shipped))

4. Consider the following set of requirements for Sun City Limo:

   For each driver, the company keeps track of the driver's name, driver's license number, home telephone number, and cell telephone number.

   For each limo, the company keeps track of a unique limousine ID number, limousine color, and number of passengers the limo can hold.

   One driver can be assigned to more than one limo.

Based on these requirements, create a set of 3NF relations.