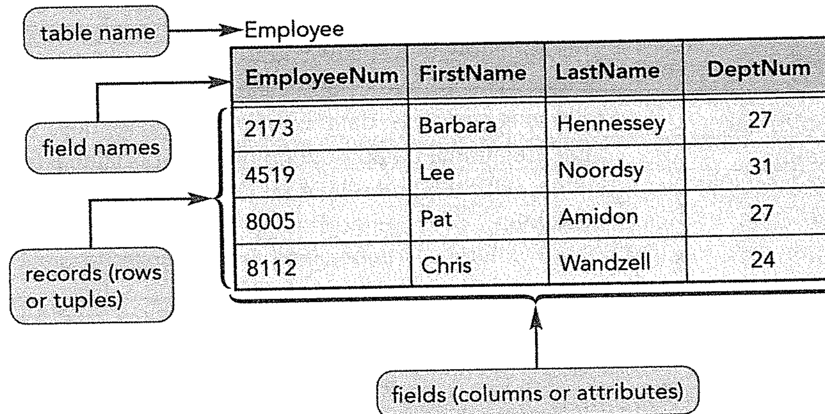
Tables

A relational database stores its data in tables. A **table** is a two-dimensional structure made up of rows and columns. The terms table, **record** (row), and **field** (column) are the popular names for the more formal terms **relation** (table), **tuple** (row), and **attribute** (column), as shown in Figure A-1.

Figure A-1 **A table (relation) consisting of records and fields**



The Employee table shown in Figure A-1 is an example of a relational database table, a two-dimensional structure with the following characteristics:

- Each row is unique. Because no two rows are the same, you can easily locate and update specific data. For example, you can locate the row for EmployeeNum 8005 and change the FirstName value, Pat, the LastName value, Amidon, or the DeptNum value, 27.
- The order of the rows is unimportant. You can add or view rows in any order. For example, you can view the rows in LastName order instead of EmployeeNum order.
- Each table entry contains a single value. At the intersection of each row and column, you cannot have more than one value. For example, each row in Figure A-1 contains one EmployeeNum value, one FirstName value, one LastName value, and one DeptNum value.
- The order of the columns is unimportant. You can add or view columns in any order.
- Each column has a unique name called the **field name**. The field name allows you to access a specific column without needing to know its position within the table.
- Each row in a table describes, or shows the characteristics of, an entity. An **entity** is a person, place, object, event, or idea for which you want to store and process data. For example, EmployeeNum, FirstName, LastName, and DeptNum are characteristics of the employees of a company. The Employee table represents all the employee entities and their characteristics. That is, each row of the Employee table describes a different employee of the company using the characteristics of EmployeeNum, FirstName, LastName, and DeptNum. The Employee table includes only characteristics of employees. Other tables would exist for the company's other entities. For example, a Department table would describe the company's departments and a Position table would describe the company's job positions.

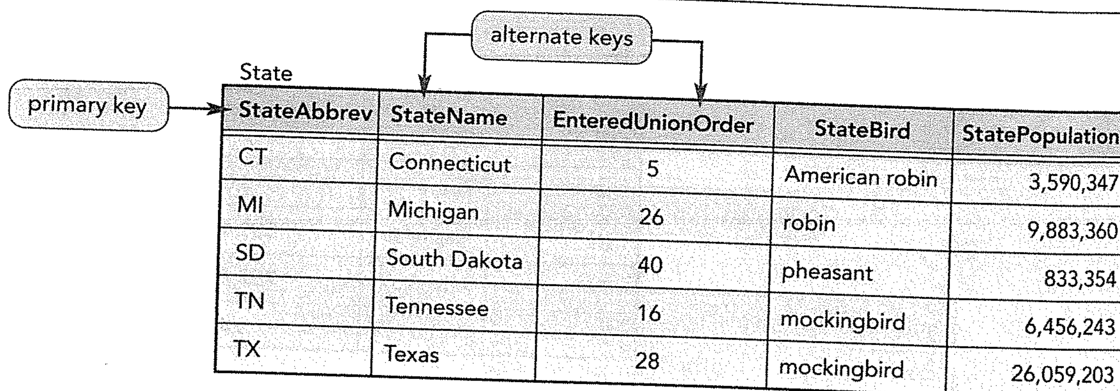
Knowing the characteristics of a table leads directly to a definition of a relational database. A **relational database** is a collection of tables (relations).

Note that this book uses singular table names, such as Employee and Department, but some people use plural table names, such as Employees and Departments. You can use either singular table names or plural table names, as long as you consistently use the style you choose.

Keys

Primary keys ensure that each row in a table is unique. A **primary key** is a column, or a collection of columns, whose values uniquely identify each row in a table. In addition to being *unique*, a primary key must be *minimal* (that is, contain no unnecessary extra columns) and must not change in value. For example, in Figure A-2 the State table contains one record per state and uses the StateAbbrev column as its primary key.

Figure A-2 **A table and its keys**



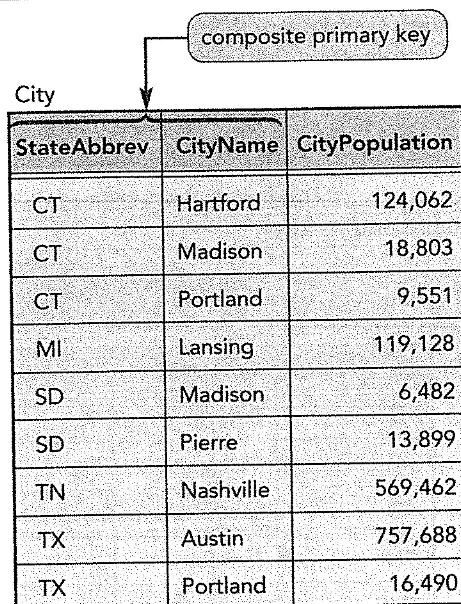
StateAbbrev	StateName	EnteredUnionOrder	StateBird	StatePopulation
CT	Connecticut	5	American robin	3,590,347
MI	Michigan	26	robin	9,883,360
SD	South Dakota	40	pheasant	833,354
TN	Tennessee	16	mockingbird	6,456,243
TX	Texas	28	mockingbird	26,059,203

Could any other column, or collection of columns, be the primary key of the State table?

- Could the StateBird column serve as the primary key? No, because the StateBird column does not have unique values (for example, the mockingbird is the state bird of more than one state).
- Could the StatePopulation column serve as the primary key? No, because the StatePopulation column values change periodically and are not guaranteed to be unique.
- Could the StateAbbrev and StateName columns together serve as the primary key? No, because the combination of these two columns is not minimal. Something less, such as the StateAbbrev column by itself, can serve as the primary key.
- Could the StateName column serve as the primary key? Yes, because the StateName column has unique values. In a similar way, you could select the EnteredUnionOrder column as the primary key for the State table. One column, or a collection of columns, that can serve as a primary key is called a **candidate key**. The candidate keys for the State table are the StateAbbrev column, the StateName column, and the EnteredUnionOrder column. You choose one of the candidate keys to be the primary key, and each remaining candidate key is called an **alternate key**. The StateAbbrev column is the State table's primary key in Figure A-2, so the StateName and EnteredUnionOrder columns become alternate keys in the table.

Figure A-3 shows a City table containing the fields StateAbbrev, CityName, and CityPopulation.

Figure A-3 **A table with a composite key**



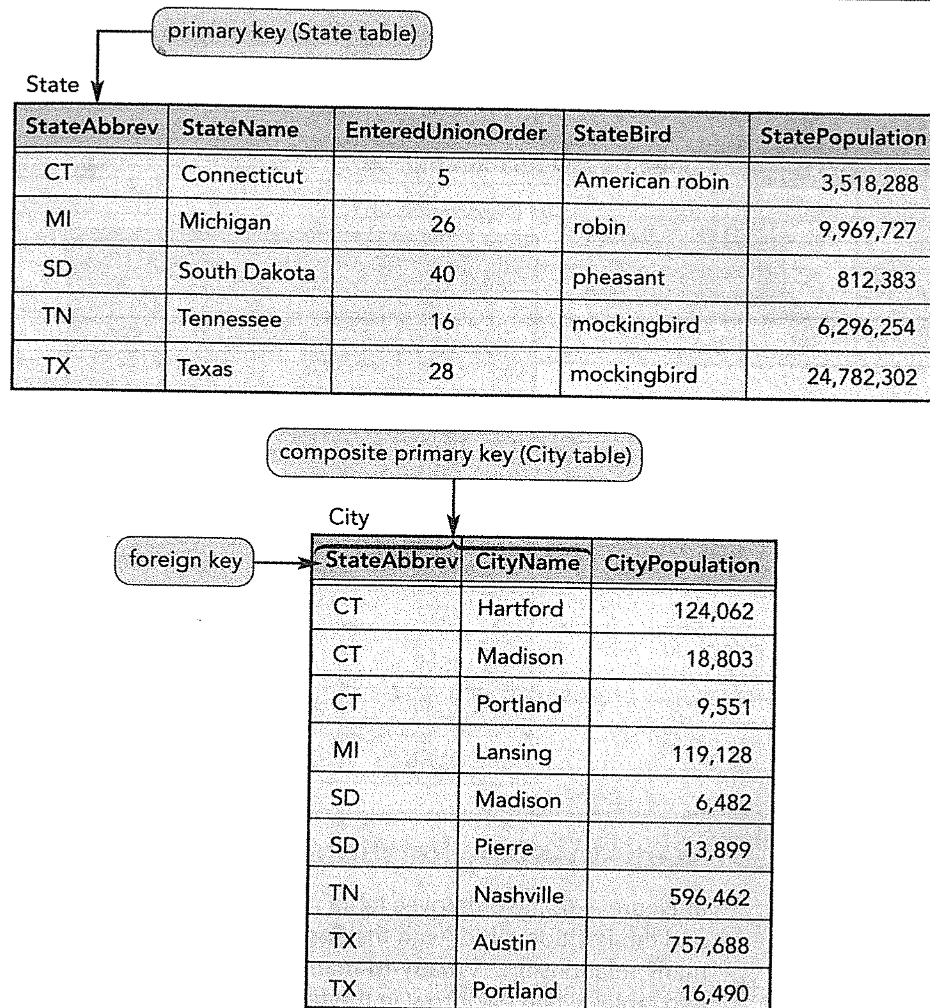
City

StateAbbrev	CityName	CityPopulation
CT	Hartford	124,062
CT	Madison	18,803
CT	Portland	9,551
MI	Lansing	119,128
SD	Madison	6,482
SD	Pierre	13,899
TN	Nashville	569,462
TX	Austin	757,688
TX	Portland	16,490

What is the primary key for the City table? The values for the CityPopulation column periodically change and are not guaranteed to be unique, so the CityPopulation column cannot be the primary key. Because the values for each of the other two columns are not unique, the StateAbbrev column alone cannot be the primary key and neither can the CityName column (for example, there are two cities named Madison and two cities named Portland). The primary key is the combination of the StateAbbrev and CityName columns. Both columns together are needed to identify—uniquely and minimally—each row in the City table. A multiple-column key is called a **composite key** or a **concatenated key**. A multiple-column primary key is called a **composite primary key**.

The StateAbbrev column in the City table is also a foreign key. A **foreign key** is a column, or a collection of columns, in one table in which each column value must match the value of the primary key of some table or must be null. A **null** is the absence of a value in a particular table entry. A null value is not blank, nor zero, nor any other value. You give a null value to a column value when you do not know its value or when a value does not apply. As shown in Figure A-4, the values in the City table's StateAbbrev column match the values in the State table's StateAbbrev column. Thus, the StateAbbrev column, the primary key of the State table, is a foreign key in the City table. Although the field name StateAbbrev is the same in both tables, the names could be different. As a rule, experts use the same name for a field stored in two or more tables to broadcast clearly that they store similar values; however, some exceptions exist.

Figure A-4 **StateAbbrev as a primary key (State table) and a foreign key (City table)**



State

StateAbbrev	StateName	EnteredUnionOrder	StateBird	StatePopulation
CT	Connecticut	5	American robin	3,518,288
MI	Michigan	26	robin	9,969,727
SD	South Dakota	40	pheasant	812,383
TN	Tennessee	16	mockingbird	6,296,254
TX	Texas	28	mockingbird	24,782,302

City

StateAbbrev	CityName	CityPopulation
CT	Hartford	124,062
CT	Madison	18,803
CT	Portland	9,551
MI	Lansing	119,128
SD	Madison	6,482
SD	Pierre	13,899
TN	Nashville	596,462
TX	Austin	757,688
TX	Portland	16,490

A **nonkey field** is a field that is not part of the primary key. In the two tables shown in Figure A-4, all fields are nonkey fields except the StateAbbrev field in the State and City tables and the CityName field in the City table. "Key" is an ambiguous word because it can refer to a primary, candidate, alternate, or foreign key. When the word "key" appears alone, however, it means primary key and the definition for a nonkey field consequently makes sense.

Relationships

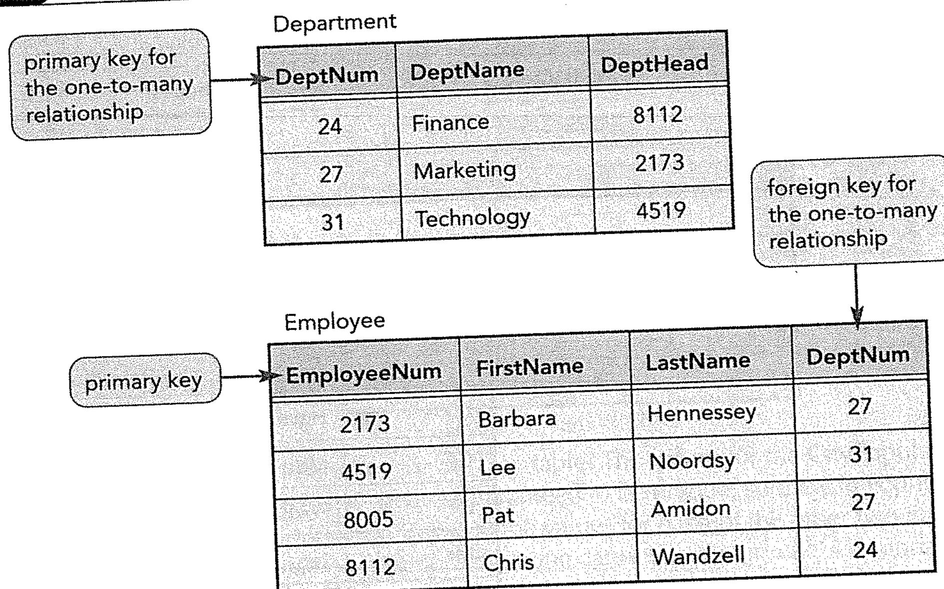
In a database, a table can be associated with another table in one of three ways: a one-to-many relationship, a many-to-many relationship, or a one-to-one relationship.

One-to-Many Relationship

The Department and Employee tables, shown in Figure A-5, have a one-to-many relationship. A **one-to-many relationship** (abbreviated **1:M** or **1:N**) exists between two tables when each row in the first table (sometimes called the **primary table**) matches many rows in the second table and each row in the second table (sometimes called the **related table**) matches at most one row in the first table. "Many" can mean zero

rows, one row, or two or more rows. As Figure A-5 shows, the DeptNum field, which is a foreign key in the Employee table and the primary key in the Department table, is the common field that ties together the rows of the two tables. Each department has many employees; and each employee works in exactly one department or hasn't been assigned to a department, if the DeptNum field value for that employee is null.

Figure A-5 **A one-to-many relationship**



Department

DeptNum	DeptName	DeptHead
24	Finance	8112
27	Marketing	2173
31	Technology	4519

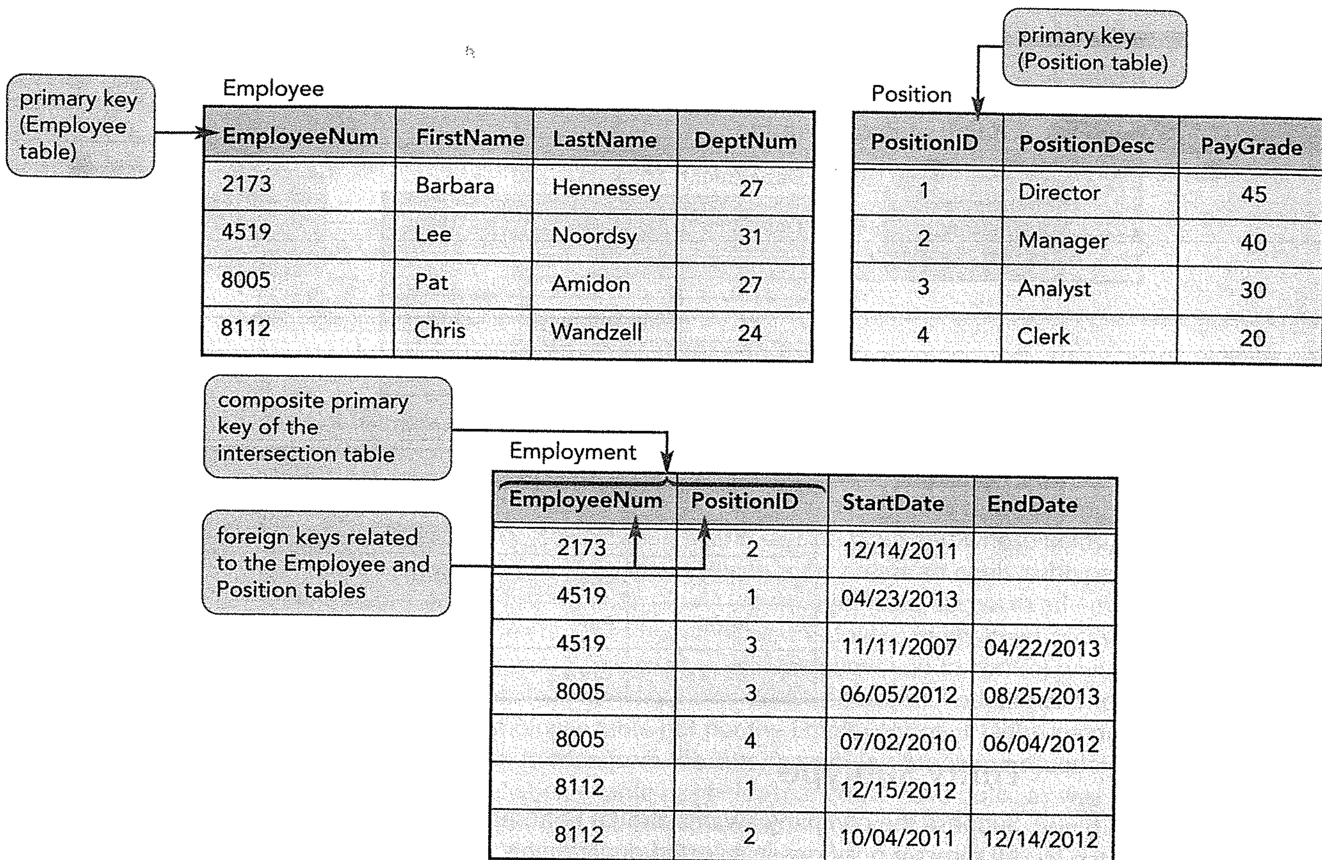
Employee

EmployeeNum	FirstName	LastName	DeptNum
2173	Barbara	Hennessey	27
4519	Lee	Noordsy	31
8005	Pat	Amidon	27
8112	Chris	Wandzell	24

Many-to-Many Relationship

In Figure A-6, the Employee table (with the EmployeeNum field as its primary key) and the Position table (with the PositionID field as its primary key) have a many-to-many relationship. A **many-to-many relationship** (abbreviated as **M:N**) exists between two tables when each row in the first table matches many rows in the second table and each row in the second table matches many rows in the first table. In a relational database, you must use a third table (often called an **intersection table**, **junction table**, or **link table**) to serve as a bridge between the two many-to-many tables; the third table has the primary keys of the two many-to-many tables as its primary key. The original tables now each have a one-to-many relationship with the new table. The EmployeeNum and PositionID fields represent the primary key of the Employment table that is shown in Figure A-6. The EmployeeNum field, which is a foreign key in the Employment table and the primary key in the Employee table, is the common field that ties together the rows of the Employee and Employment tables. Likewise, the PositionID field is the common field for the Position and Employment tables. Each employee may serve in many different positions within the company over time, and each position in the company will be filled by different employees over time.

Figure A-6 A many-to-many relationship



Employee

EmployeeNum	FirstName	LastName	DeptNum
2173	Barbara	Hennessey	27
4519	Lee	Noordsy	31
8005	Pat	Amidon	27
8112	Chris	Wandzell	24

Position

PositionID	PositionDesc	PayGrade
1	Director	45
2	Manager	40
3	Analyst	30
4	Clerk	20

Employment

EmployeeNum	PositionID	StartDate	EndDate
2173	2	12/14/2011	
4519	1	04/23/2013	
4519	3	11/11/2007	04/22/2013
8005	3	06/05/2012	08/25/2013
8005	4	07/02/2010	06/04/2012
8112	1	12/15/2012	
8112	2	10/04/2011	12/14/2012

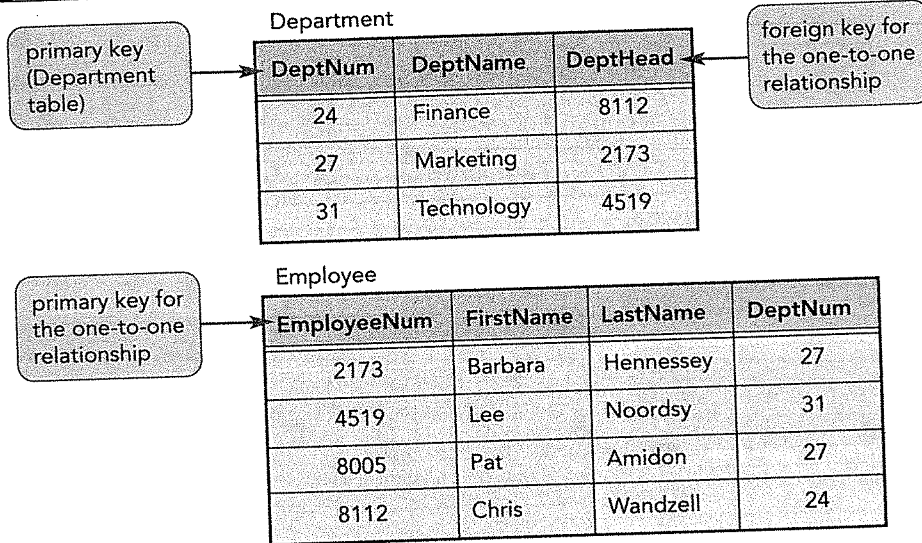
One-to-One Relationship

In Figure A-5, recall that there's a one-to-many relationship between the Department table (the primary table) and the Employee table (the related table). Each department has many employees, and each employee works in one department. The DeptNum field in the Employee table serves as a foreign key to connect records in that table to records with matching DeptNum field values in the Department table.

Furthermore, each department has a single employee who serves as the head of the department, and each employee either serves as the head of a department or simply works in a department without being the department head. Therefore, the Department and Employee tables not only have a one-to-many relationship, but these two tables also have a second relationship, a one-to-one relationship. A **one-to-one relationship** (abbreviated **1:1**) exists between two tables when each row in each table has at most one matching row in the other table. As shown in Figure A-7, each DeptHead field value in the Department table represents the employee number in the Employee table of the employee who heads the department. In other words, each DeptHead field value in the Department table matches exactly one EmployeeNum field value in the Employee table. At the same time, each EmployeeNum field value in the Employee table matches at most one DeptHead field value in the Department table—matching one DeptHead field value if the employee is a department head, or matching zero DeptHead field values if the employee is not a department head. For this one-to-one relationship, the EmployeeNum field in the Employee table and the DeptHead field in the Department table are the fields that link the two tables, with the DeptHead field serving as a foreign key in the Department table and the EmployeeNum field serving as a primary key in the Employee table.

Some database designers might use EmployeeNum instead of DeptHead as the field name for the foreign key in the Department table because they both represent the employee number for the employees of the company. However, DeptHead better identifies the purpose of the field and would more commonly be used as the field name.

Figure A-7 **A one-to-one relationship**



Department

DeptNum	DeptName	DeptHead
24	Finance	8112
27	Marketing	2173
31	Technology	4519

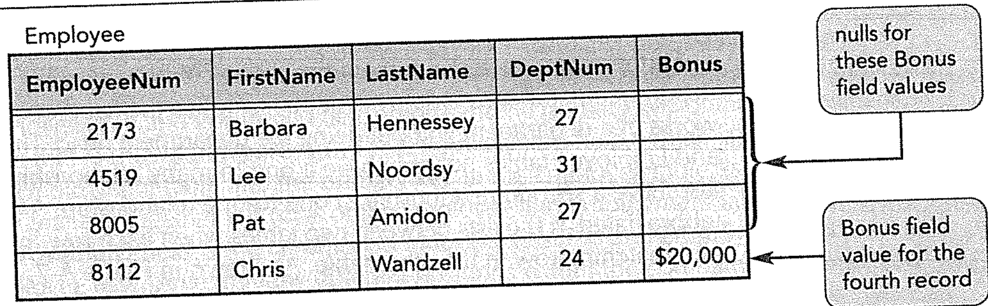
Employee

EmployeeNum	FirstName	LastName	DeptNum
2173	Barbara	Hennessey	27
4519	Lee	Noordsy	31
8005	Pat	Amidon	27
8112	Chris	Wandzell	24

Entity Subtype

Suppose the company awards annual bonuses to a small number of employees who fill director positions in selected departments. As shown in Figure A-8, you could store the Bonus field in the Employee table because a bonus is an attribute associated with employees. The Bonus field would contain either the amount of the employee's bonus (record 4 in the Employee table) or a null value for employees without bonuses (records 1 through 3 in the Employee table).

Figure A-8 **Bonus field added to the Employee table**

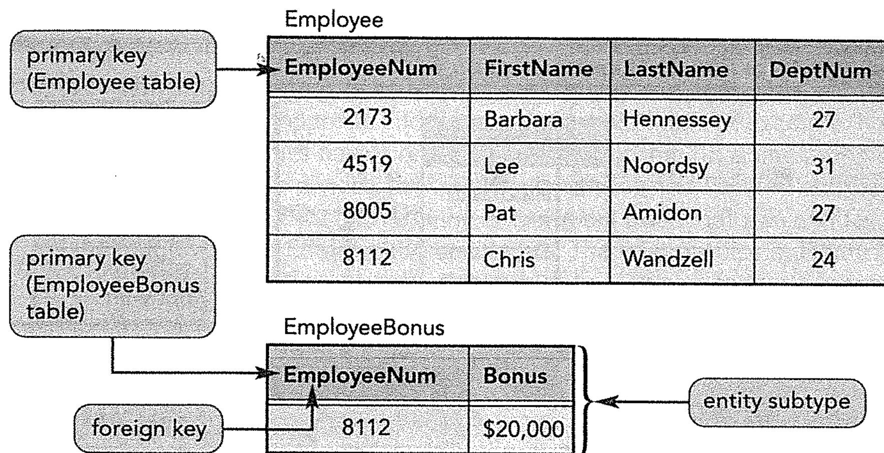


Employee

EmployeeNum	FirstName	LastName	DeptNum	Bonus
2173	Barbara	Hennessey	27	
4519	Lee	Noordsy	31	
8005	Pat	Amidon	27	
8112	Chris	Wandzell	24	$20,000

Figure A-9 shows an alternative approach, in which the Bonus field is placed in a separate table, the EmployeeBonus table. The EmployeeBonus table's primary key is the EmployeeNum field, and the table contains one row for each employee earning a bonus. Because some employees do not earn a bonus, the EmployeeBonus table has fewer rows than the Employee table. However, each row in the EmployeeBonus table has a matching row in the Employee table, with the EmployeeNum field serving as the common field; the EmployeeNum field is the primary key in the Employee table and is a foreign key in the EmployeeBonus table.

Figure A-9 **Bonus values stored in a separate table, an entity subtype**


Employee

EmployeeNum	FirstName	LastName	DeptNum
2173	Barbara	Hennessey	27
4519	Lee	Noordsy	31
8005	Pat	Amidon	27
8112	Chris	Wandzell	24

EmployeeBonus

EmployeeNum	Bonus
8112	$20,000

The EmployeeBonus table, in this situation, is called an **entity subtype**, a table whose primary key is a foreign key to a second table and whose fields are additional fields for the second table. Database designers create an entity subtype in two situations. In the first situation, some users might need access to all employee fields, including employee bonuses, while other employees might need access to all employee fields except bonuses. Because most DBMSs allow you to control which tables a user can access, you can specify that some users can access both tables and that other users can access the Employee table but not the EmployeeBonus table, keeping the employee bonus information hidden from the latter group. In the second situation, you can create an entity subtype when a table has fields that could have nulls, as was the case for the Bonus field stored in the Employee table in Figure A-8. You should be aware that database experts debate the validity of the use of nulls in relational databases, and many experts insist that you should never use nulls. This warning against nulls is partly based on the inconsistent way different RDBMSs treat nulls and partly due to the lack of a firm theoretical foundation for how to use nulls. In any case, entity subtypes are an alternative to the use of nulls.

Entity-Relationship Diagrams

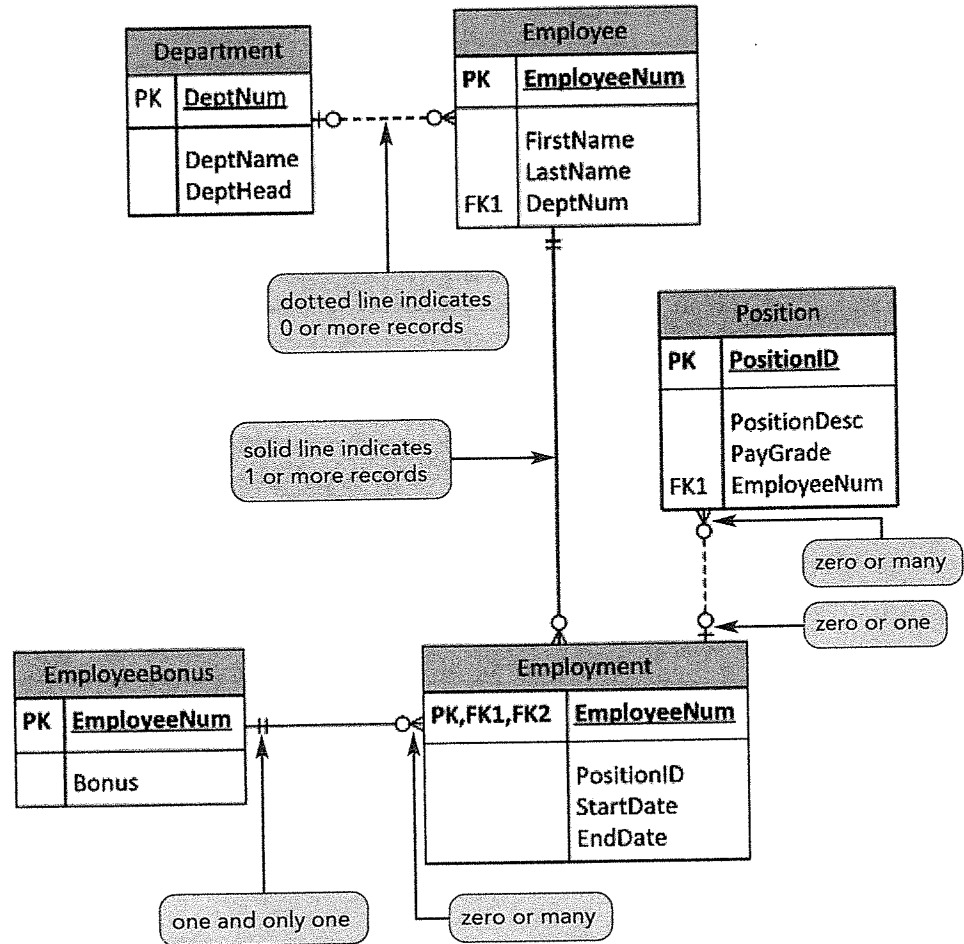
A common shorthand method for describing tables is to write the table name followed by its fields in parentheses, underlining the fields that represent the primary key and identifying the foreign keys for a table immediately after the table. Using this method, the tables that appear in Figures A-5 through A-7 and Figure A-9 are described in the following way:

Department (DeptNum, DeptName, DeptHead)
    Foreign key: DeptHead to Employee table
Employee (EmployeeNum, FirstName, LastName, DeptNum)
    Foreign key: DeptNum to Department table
Position (PositionID, PositionDesc, PayGrade)
Employment (EmployeeNum, PositionID, StartDate, EndDate)
    Foreign key: EmployeeNum to Employee table
    Foreign key: PositionID to Position table
EmployeeBonus (EmployeeNum, Bonus)
    Foreign key: EmployeeNum to Employee table

Another popular way to describe tables *and their relationships* is with entity-relationship diagrams. An **entity-relationship diagram (ERD)** shows a database's entities and the relationships among the entities in a symbolic, visual way. In an ERD, an entity

and a table are equivalent. Figure A-10 shows an entity-relationship diagram for the tables that appear in Figures A-5 through A-7 and Figure A-9.

Figure A-10 **An entity-relationship diagram (ERD)**



ERDs have the following characteristics:

- A table is represented by a rectangle that contains the table name and lists the field names. Within each rectangle, the primary key is identified with the abbreviation PK, and any foreign keys are designated with FK. Required fields are formatted in bold.
- Relationships are identified by lines joining the tables. A solid relationship line between two tables indicates there could be 1 or more related records. A dotted relationship line between two tables indicates there could be 0 or more related records.

- At the ends of each relationship line, symbols identify the minimum and maximum possible number of related records from each entity in the relationship. A single perpendicular line represents 1 record, a circle represents 0 records, and a group of three branching lines—known as a crow's foot—represents many records. A one-to-many relationship is represented by a 1 at one end of the relationship line and a crow's foot at the opposite end of the relationship line. For example, the Department and Employee tables have a one-to-many relationship. In a similar manner, a many-to-many relationship exists between the Employee and Position entities and one-to-one relationships exist between the Department and Employee entities and between the Employee and EmployeeBonus entities. The relationships in Figure A-10 illustrate all the possible designations for the ends of lines except for "one or many," which is represented by a single perpendicular line with a crow's foot.

Integrity Constraints

A database has **integrity** if its data follows certain rules; each rule is called an **integrity constraint**. The ideal is to have the DBMS enforce all integrity constraints. If a DBMS can enforce some integrity constraints but not others, the other integrity constraints must be enforced by other programs or by the people who use the DBMS. Integrity constraints can be divided into three groups: primary key constraints, foreign key constraints, and domain integrity constraints.

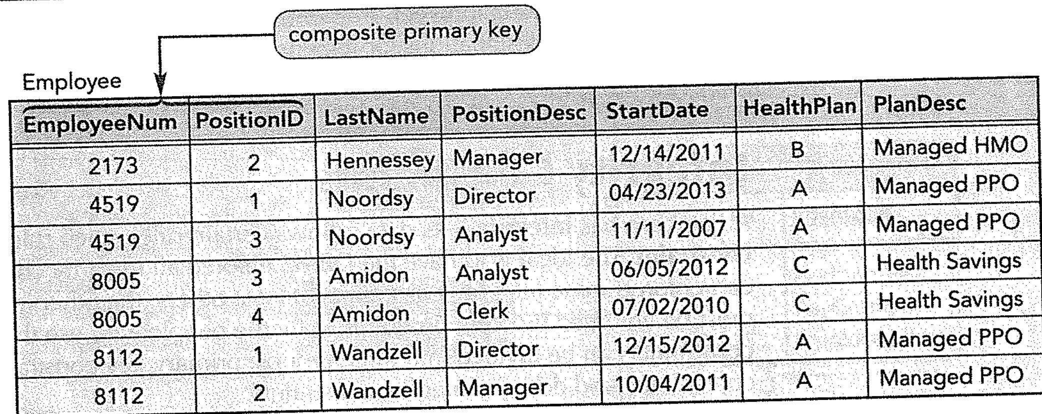
- One primary key constraint is inherent in the definition of a primary key, which says that the primary key must be unique. The **entity integrity constraint** says that the primary key cannot be null. For a composite key, none of the individual fields can be null. The uniqueness and nonnull properties of a primary key ensure that you can reference any data value in a database by supplying its table name, field name, and primary key value.
- Foreign keys provide the mechanism for forming a relationship between two tables, and referential integrity ensures that only valid relationships exist. **Referential integrity** is the constraint specifying that each nonnull foreign key value must match a primary key value in the primary table. Specifically, referential integrity means that you cannot add a row containing an unmatched foreign key value. Referential integrity also means that you cannot change or delete the related primary key value and leave the foreign key orphaned. In some RDBMSs, when you create a relationship, you can specify one of these options: restricted, cascades, or nullifies. If you specify **restricted** and then change or delete a primary key, the DBMS updates or deletes the value only if there are no matching foreign key values. If you choose **cascades** and then change a primary key value, the DBMS changes the matching foreign key values to the new primary key value, or, if you delete a primary key value, the DBMS also deletes the matching foreign key rows. If you choose **nullifies** and then change or delete a primary key value, the DBMS sets all matching foreign key values to null.
- A **domain** is a set of values from which one or more fields draw their actual values. A **domain integrity constraint** is a rule you specify for a field. By choosing a data type for a field, you impose a constraint on the set of values allowed for the field. You can create specific validation rules for a field to limit its domain further. As you make a field's domain definition more precise, you exclude more and more unacceptable values for the field. For example, in the State table, shown in Figures A-2 and A-4, you could define the domain for the EnteredUnionOrder field to be a unique integer between 1 and 50 and the domain for the StateBird field to be any text string containing 25 or fewer characters.

Dependencies and Determinants

Just as tables are related to other tables, fields are also related to other fields. Consider the modified Employee table shown in Figure A-11. Its description is:

Employee (EmployeeNum, PositionID, LastName, PositionDesc, StartDate, HealthPlan, PlanDesc)

Figure A-11 A table combining fields from three tables



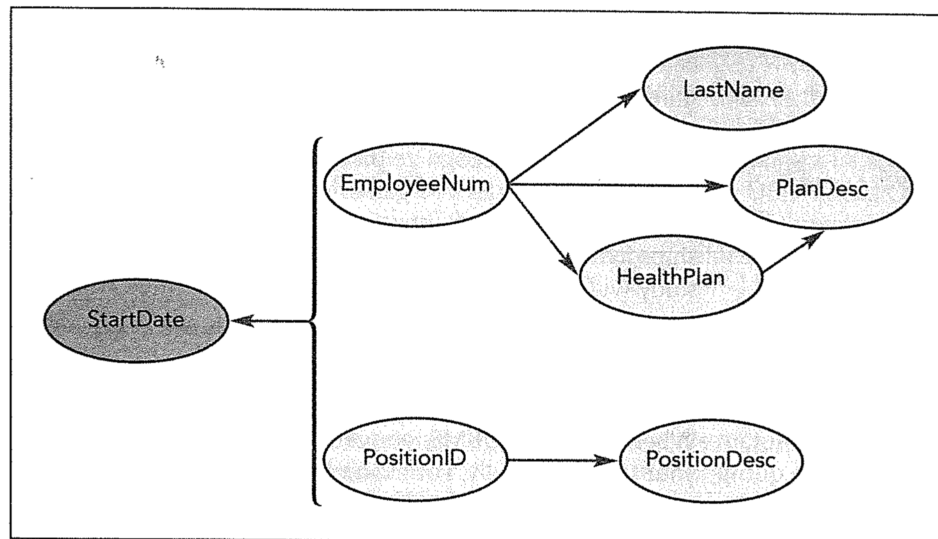
EmployeeNum	PositionID	LastName	PositionDesc	StartDate	HealthPlan	PlanDesc
2173	2	Hennessey	Manager	12/14/2011	B	Managed HMO
4519	1	Noordsy	Director	04/23/2013	A	Managed PPO
4519	3	Noordsy	Analyst	11/11/2007	A	Managed PPO
8005	3	Amidon	Analyst	06/05/2012	C	Health Savings
8005	4	Amidon	Clerk	07/02/2010	C	Health Savings
8112	1	Wandzell	Director	12/15/2012	A	Managed PPO
8112	2	Wandzell	Manager	10/04/2011	A	Managed PPO

The modified Employee table combines several fields from the Employee, Position, and Employment tables that appeared in Figure A-6. The EmployeeNum and LastName fields are from the Employee table. The PositionID and PositionDesc fields are from the Position table. The EmployeeNum, PositionID, and StartDate fields are from the Employment table. The HealthPlan and PlanDesc fields are new fields for the Employee table, whose primary key is now the combination of the EmployeeNum and PositionID fields.

In the Employee table, each field is related to other fields. To determine field relationships, you ask "Does a value for a particular field give me a single value for another field?" If the answer is Yes, then the two fields are **functionally** related. For example, a value for the EmployeeNum field determines a single value for the LastName field, and a value for the LastName field depends on the value of the EmployeeNum field. In other words, EmployeeNum functionally determines LastName, and LastName is functionally dependent on EmployeeNum. In this case, EmployeeNum is called a determinant. A **determinant** is a field, or a collection of fields, whose values determine the values of another field. A field is functionally dependent on another field (or a collection of fields) if that other field is a determinant for it.

You can graphically show a table's functional dependencies and determinants in a **bubble diagram**; a bubble diagram is also called a **data model diagram** or a **functional dependency diagram**. Figure A-12 shows the bubble diagram for the Employee table shown in Figure A-11.

Figure A-12 **A bubble diagram for the modified Employee table**



You can read the bubble diagram in Figure A-12 as follows:

- The EmployeeNum field is a determinant for the LastName, HealthPlan, and PlanDesc fields.
- The PositionID field is a determinant for the PositionDesc field.
- The StartDate field is functionally dependent on the EmployeeNum and PositionID fields together.
- The HealthPlan field is a determinant for the PlanDesc field.

Note that EmployeeNum and PositionID together serve as a determinant for the StartDate field and for all fields that depend on the EmployeeNum field alone and the PositionID field alone. Some experts include these additional fields and some don't. The previous list of determinants does not include these additional fields.

An alternative way to show determinants is to list the determinant, a right arrow, and then the dependent fields, separated by commas. Using this alternative, the determinants shown in Figure A-12 are:

EmployeeNum → LastName, HealthPlan, PlanDesc
PositionID → PositionDesc
EmployeeNum, PositionID → StartDate
HealthPlan → PlanDesc

Only the StartDate field is functionally dependent on the table's full primary key, the EmployeeNum and PositionID fields. The LastName, HealthPlan, and PlanDesc fields have partial dependencies because they are functionally dependent on the EmployeeNum field, which is part of the primary key. A **partial dependency** is a functional dependency on part of the primary key, instead of the entire primary key. Does another partial dependency exist in the Employee table? Yes, the PositionDesc field has a partial dependency on the PositionID field.

Because the EmployeeNum field is a determinant of both the HealthPlan and PlanDesc fields, and the HealthPlan field is a determinant of the PlanDesc field, the HealthPlan and PlanDesc fields have a transitive dependency. A **transitive dependency** is a functional dependency between two nonkey fields, which are both dependent on a third field.

How do you know which functional dependencies exist among a collection of fields, and how do you recognize partial and transitive dependencies? The answers lie with the questions you ask as you gather the requirements for a database application. For each field and entity, you must gain an accurate understanding of its meaning and relationships in the context of the application. **Semantic object modeling** is an entire area of study within the database field devoted to the meanings and relationships of data.

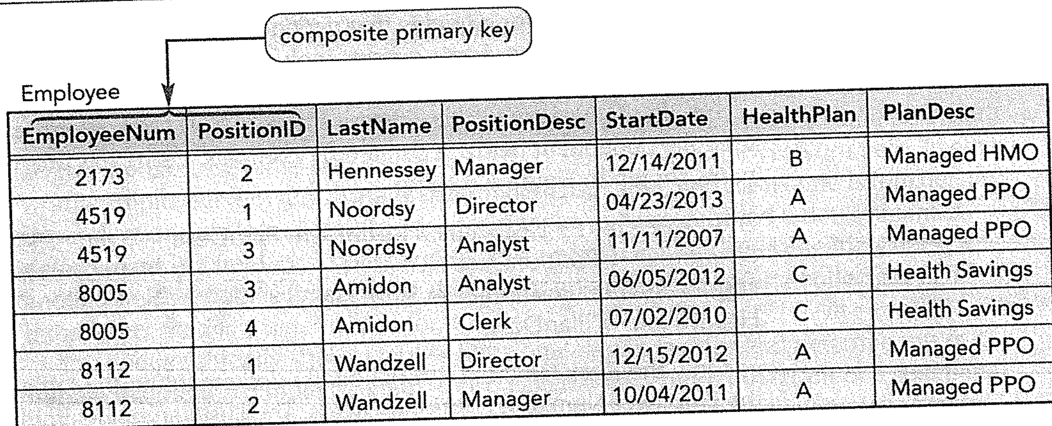
Anomalies

When you use a DBMS, you are more likely to get results you can trust if you create your tables carefully. For example, problems might occur with tables that have partial and transitive dependencies, whereas you won't have as much trouble if you ensure that your tables include only fields that are directly related to each other. Also, when you remove data redundancy from a table, you improve that table. **Data redundancy** occurs when you store the same data in more than one place.

The problems caused by data redundancy and by partial and transitive dependencies are called **anomalies** because they are undesirable irregularities of tables. Anomalies are of three types: insertion, deletion, and update.

To examine the effects of these anomalies, consider the modified Employee table that is shown again in Figure A-13.

Figure A-13 A table with insertion, deletion, and update anomalies



EmployeeNum	PositionID	LastName	PositionDesc	StartDate	HealthPlan	PlanDesc
2173	2	Hennessey	Manager	12/14/2011	B	Managed HMO
4519	1	Noordsy	Director	04/23/2013	A	Managed PPO
4519	3	Noordsy	Analyst	11/11/2007	A	Managed PPO
8005	3	Amidon	Analyst	06/05/2012	C	Health Savings
8005	4	Amidon	Clerk	07/02/2010	C	Health Savings
8112	1	Wandzell	Director	12/15/2012	A	Managed PPO
8112	2	Wandzell	Manager	10/04/2011	A	Managed PPO

- An **insertion anomaly** occurs when you cannot add a record to a table because you do not know the entire primary key value. For example, you cannot add the new employee Cathy Corbett with an EmployeeNum of 3322 to the Employee table if you do not know her position in the company. Entity integrity prevents you from leaving any part of a primary key null. Because the PositionID field is part of the primary key, you cannot leave it null. To add the new employee, your only option is to make up a PositionID field value, until you determine the correct position. This solution misrepresents the facts and is unacceptable, if a better approach is available.

- A **deletion anomaly** occurs when you delete data from a table and unintentionally lose other critical data. For example, if you delete EmployeeNum 2173 because Hennessey is no longer an employee, you also lose the only instance of HealthPlan B in the database. Thus, you no longer know that HealthPlan B is the "Managed HMO" plan.
- An **update anomaly** occurs when a change to one field value requires the DBMS to make more than one change to the database, and a failure by the database to make all the changes results in inconsistent data. For example, if you change a LastName, HealthPlan, or PlanDesc field value for EmployeeNum 8005, the DBMS must change multiple rows of the Employee table. If the DBMS fails to change all the rows, the LastName, HealthPlan, or PlanDesc field now has different values in the database and is inconsistent.

Normalization

Database design is the process of determining the content and structure of data in a database in order to support some activity on behalf of a user or group of users. After you have determined the collection of fields users need to support an activity, you need to determine the precise tables needed for the collection of fields and then place those fields into the correct tables. Understanding the functional dependencies of all fields; recognizing the anomalies caused by data redundancy, partial dependencies, and transitive dependencies when they exist; and knowing how to eliminate the anomalies are all crucial to good database design. Failure to eliminate anomalies leads to data redundancy and can cause data integrity and other problems as your database grows in size.

The process of identifying and eliminating anomalies is called **normalization**. Using normalization, you start with a collection of tables, apply sets of rules to eliminate anomalies, and produce a new collection of problem-free tables. The sets of rules are called **normal forms**. Of special interest for our purposes are the first three normal forms: first normal form, second normal form, and third normal form. First normal form improves the design of your tables, second normal form improves the first normal form design, and third normal form applies even more stringent rules to produce an even better design. Note that normal forms beyond third normal form exist; these higher normal forms can improve a database design in some situations but won't be covered in this section.

First Normal Form

Consider the Employee table shown in Figure A-14. For each employee, the table contains EmployeeNum, which is the primary key; the employee's first name, last name, health plan code and description; and the ID, description, pay grade, and start date of each position held by the employee. For example, Barbara Hennessey has held one position, while the other three employees have held two positions. Because each entry in a table must contain a single value, the structure shown in Figure A-14 does not meet the requirements for a table, or relation; therefore, it is called an **unnormalized relation**. The set of fields that includes the PositionID, PositionDesc, PayGrade, and StartDate fields, which can have more than one value, is called a **repeating group**.

Figure A-14 **Repeating groups of data in an unnormalized Employee table**

Employee

EmployeeNum	PositionID	FirstName	LastName	PositionDesc	PayGrade	StartDate	HealthPlan	PlanDesc
2173	2	Barbara	Hennessey	Manager	40	12/14/2011	B	Managed HMO
4519	1 3	Lee	Noordsy	Director Analyst	45 30	04/23/2013 11/11/2007	A	Managed PPO
8005	3 4	Pat	Amidon	Analyst Clerk	30 20	06/05/2012 07/02/2010	C	Health Savings
8112	1 2	Chris	Wandzell	Director Manager	45 40	12/15/2012 10/04/2011	A	Managed PPO

First normal form addresses this repeating-group situation. A table is in **first normal form (1NF)** if it does not contain repeating groups. To remove a repeating group and convert to first normal form, you expand the primary key to include the primary key of the repeating group, forming a composite key. Performing the conversion step produces the 1NF table shown in Figure A-15.

Figure A-15 **After conversion to 1NF**

Employee

EmployeeNum	PositionID	FirstName	LastName	PositionDesc	PayGrade	StartDate	HealthPlan	PlanDesc
2173	2	Barbara	Hennessey	Manager	40	12/14/2011	B	Managed HMO
4519	1	Lee	Noordsy	Director	45	04/23/2013	A	Managed PPO
4519	3	Lee	Noordsy	Analyst	30	11/11/2007	A	Managed PPO
8005	3	Pat	Amidon	Analyst	30	06/05/2012	C	Health Savings
8005	4	Pat	Amidon	Clerk	20	07/02/2010	C	Health Savings
8112	1	Chris	Wandzell	Director	45	12/15/2012	A	Managed PPO
8112	2	Chris	Wandzell	Manager	40	10/04/2011	A	Managed PPO

The alternative way to describe the 1NF table is:

Employee (EmployeeNum, PositionID, FirstName, LastName, PositionDesc, PayGrade, StartDate, HealthPlan, PlanDesc)

The Employee table is now a true table and has a composite key. The table, however, suffers from insertion, deletion, and update anomalies. (As an exercise, find examples of the three anomalies in the table.) The EmployeeNum field is a determinant for the FirstName, LastName, HealthPlan, and PlanDesc fields, so partial dependencies exist in the Employee table. It is these partial dependencies that cause the anomalies in the Employee table, and second normal form addresses the partial-dependency problem.

Second Normal Form

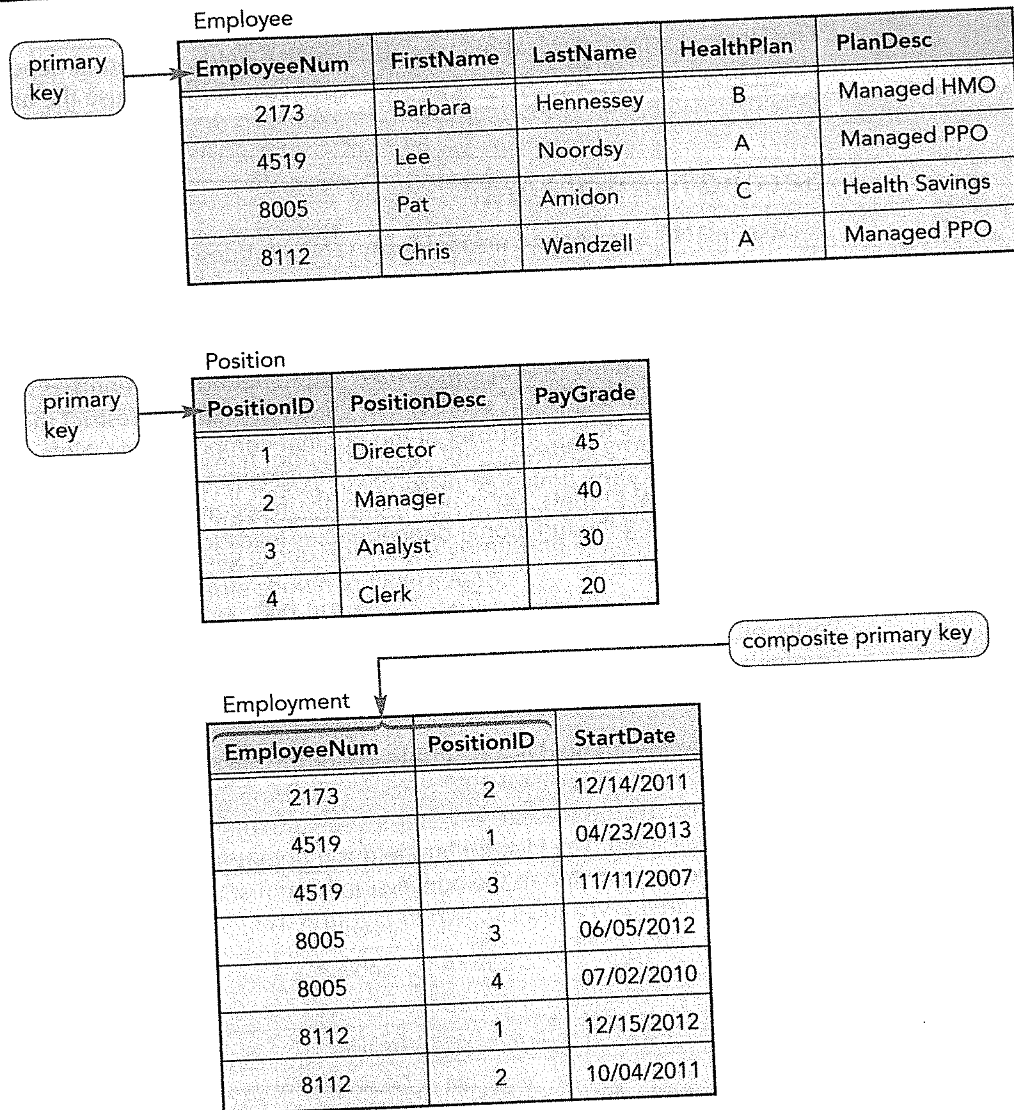
A table in 1NF is in **second normal form (2NF)** if it does not contain any partial dependencies. To remove partial dependencies from a table and convert it to second normal form, you perform two steps. First, identify the functional dependencies for every field in the table. Second, if necessary, create new tables and place each field in a table such that the field is functionally dependent on the entire primary key, not part of the primary key. If you need to create new tables, restrict them to tables with a primary key that is a subset of the original composite key. Note that partial dependencies occur only when you have a composite key; a table in first normal form with a single-field primary key is automatically in second normal form.

First, identifying the functional dependencies leads to the following determinants for the Employee table:

EmployeeNum → FirstName, LastName, HealthPlan, PlanDesc
PositionID → PositionDesc, PayGrade
EmployeeNum, PositionID → StartDate
HealthPlan → PlanDesc

The EmployeeNum field is a determinant for the FirstName, LastName, HealthPlan, and PlanDesc fields. The PositionID field is a determinant for the PositionDesc and PayGrade fields. The composite key EmployeeNum and PositionID is a determinant for the StartDate field. The HealthPlan field is a determinant for the PlanDesc field. Performing the second step in the conversion from first normal form to second form produces the three 2NF tables shown in Figure A-16.

Figure A-16 After conversion to 2NF



Employee

EmployeeNum	FirstName	LastName	HealthPlan	PlanDesc
2173	Barbara	Hennessey	B	Managed HMO
4519	Lee	Noordsy	A	Managed PPO
8005	Pat	Amidon	C	Health Savings
8112	Chris	Wandzell	A	Managed PPO

Position

PositionID	PositionDesc	PayGrade
1	Director	45
2	Manager	40
3	Analyst	30
4	Clerk	20

Employment

EmployeeNum	PositionID	StartDate
2173	2	12/14/2011
4519	1	04/23/2013
4519	3	11/11/2007
8005	3	06/05/2012
8005	4	07/02/2010
8112	1	12/15/2012
8112	2	10/04/2011

The alternative way to describe the 2NF tables is:

Employee (EmployeeNum, FirstName, LastName, HealthPlan, PlanDesc)
Position (PositionID, PositionDesc, PayGrade)
Employment (EmployeeNum, PositionID, StartDate)
    Foreign key: EmployeeNum to Employee table
    Foreign key: PositionID to Position table

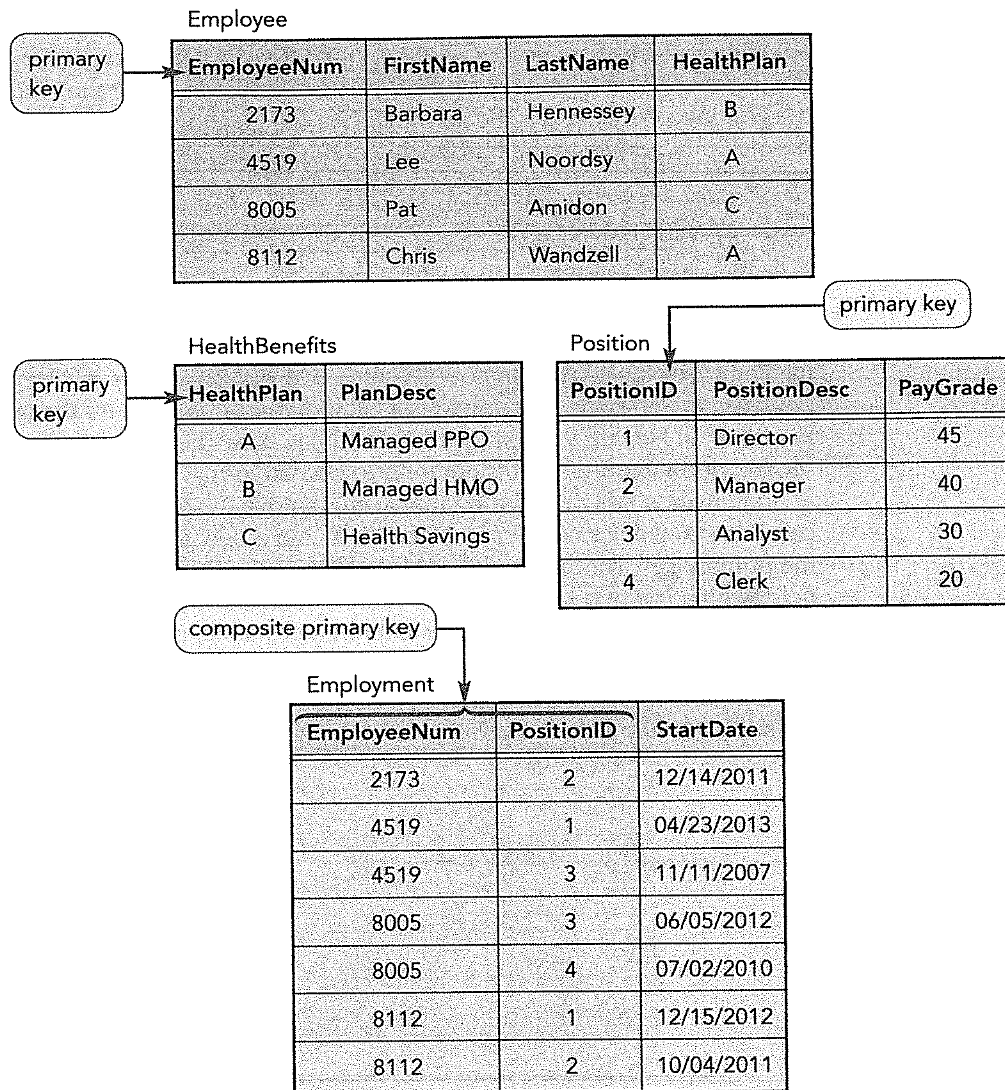
All three tables are in second normal form. Do anomalies still exist? The Position and Employment tables show no anomalies, but the Employee table suffers from anomalies caused by the transitive dependency between the HealthPlan and PlanDesc fields. (As an exercise, find examples of the three anomalies caused by the transitive dependency.) That is, the HealthPlan field is a determinant for the PlanDesc field, and the EmployeeNum field is a determinant for the HealthPlan and PlanDesc fields. Third normal form addresses the transitive-dependency problem.

Third Normal Form

A table in 2NF is in **third normal form (3NF)** if every determinant is a candidate key. This definition for 3NF is referred to as **Boyce-Codd normal form (BCNF)** and is an improvement over the original version of 3NF. What are the determinants in the Employee table? The EmployeeNum and HealthPlan fields are the determinants; however, the EmployeeNum field is a candidate key because it's the table's primary key, and the HealthPlan field is not a candidate key. Therefore, the Employee table is in second normal form, but it is not in third normal form.

To convert a table to third normal form, remove the fields that depend on the non-candidate-key determinant and place them into a new table with the determinant as the primary key. For the Employee table, the PlanDesc field depends on the HealthPlan field, which is a non-candidate-key determinant. Thus, you remove the PlanDesc field from the table, create a new HealthBenefits table, place the PlanDesc field in the HealthBenefits table, and then make the HealthPlan field the primary key of the HealthBenefits table. Note that only the PlanDesc field is removed from the Employee table; the HealthPlan field remains as a foreign key in the Employee table. Figure A-17 shows the database design for the four 3NF tables.

Figure A-17 After conversion to 3NF



Employee

EmployeeNum	FirstName	LastName	HealthPlan
2173	Barbara	Hennessey	B
4519	Lee	Noordsy	A
8005	Pat	Amidon	C
8112	Chris	Wandzell	A

HealthBenefits

HealthPlan	PlanDesc
A	Managed PPO
B	Managed HMO
C	Health Savings

Position

PositionID	PositionDesc	PayGrade
1	Director	45
2	Manager	40
3	Analyst	30
4	Clerk	20

Employment

EmployeeNum	PositionID	StartDate
2173	2	12/14/2011
4519	1	04/23/2013
4519	3	11/11/2007
8005	3	06/05/2012
8005	4	07/02/2010
8112	1	12/15/2012
8112	2	10/04/2011

The alternative way to describe the 3NF relations is:

Employee (EmployeeNum, FirstName, LastName, HealthPlan)
  Foreign key: HealthPlan to HealthBenefits table
HealthBenefits (HealthPlan, PlanDesc)
Position (PositionID, PositionDesc, PayGrade)
Employment (EmployeeNum, PositionID, StartDate)
  Foreign key: EmployeeNum to Employee table
  Foreign key: PositionID to Position table

The four tables have no anomalies because you have eliminated all the data redundancy, partial dependencies, and transitive dependencies. Normalization provides the framework for eliminating anomalies and delivering an optimal database design, which you should always strive to achieve. You should be aware, however, that experts sometimes denormalize tables to improve database performance—specifically, to decrease the time it takes the database to respond to a user's commands and requests. Typically, when you denormalize tables, you combine separate tables into one table to reduce the need for the DBMS to join the separate tables to process queries and other informational requests. When you denormalize a table, you reintroduce redundancy to the table. At the same time, you reintroduce anomalies. Thus, improving performance exposes a database to potential integrity problems. Only database experts should denormalize tables, but even experts first complete the normalization of their tables.

Natural, Artificial, and Surrogate Keys

When you complete the design of a database, your tables should be in third normal form, free of anomalies and redundancy. Some tables, such as the State table (see Figure A-2), have obvious third normal form designs with obvious primary keys. The State table's description is:

State (StateAbbrev, StateName, EnteredUnionOrder, StateBird, StatePopulation)

Recall that the candidate keys for the State table are StateAbbrev, StateName, and EnteredUnionOrder. Choosing the StateAbbrev field as the State table's primary key makes the StateName and EnteredUnionOrder fields alternate keys. Primary keys such as the StateAbbrev field are sometimes called natural keys. A **natural key** (also called a **logical key** or an **intelligent key**) is a primary key that consists of a field, or a collection of fields, that is an inherent characteristic of the entity described by the table and that is visible to users. Other examples of natural keys are the ISBN (International Standard Book Number) for a book, the SSN (Social Security number) for a U.S. individual, the UPC (Universal Product Code) for a product, and the VIN (vehicle identification number) for a vehicle.

Is the PositionID field, which is the primary key for the Position table (see Figure A-17), a natural key? No, the PositionID field is not an inherent characteristic of a position. Instead, the PositionID field has been added to the Position table only as a way to identify each position uniquely. The PositionID field is an **artificial key**, which is a field that you add to a table to serve solely as the primary key and that is visible to users.

Another reason for using an artificial key arises in tables that allow duplicate records. Although relational database theory and most experts do not allow duplicate records in a table, consider a database that tracks donors and their donations. Figure A-18 shows a Donor table with an artificial key of DonorID and with the DonorFirstName and DonorLastName fields. Some cash donations are anonymous, which accounts for the fourth record in the Donor table. Figure A-18 also shows the Donation table with the DonorID field, a foreign key to the Donor table, and the DonationDate and DonationAmt fields.

Figure A-18 Donor and Donation tables

Donor

DonorID	DonorFirstName	DonorLastName
1	Christina	Chang
2	Franco	Diaz
3	Angie	Diaz
4		Anonymous
5	Tracy	Burns

Donation

DonorID	DonationDate	DonationAmt
1	10/12/2015	$50.00
1	09/30/2016	$50.00
2	10/03/2016	$75.00
4	10/10/2016	$50.00
4	10/10/2016	$50.00
4	10/11/2016	$25.00
5	10/13/2016	$50.00

What is the primary key of the Donation table? No single field is unique, and neither is any combination of fields. For example, on 10/10/2016, two anonymous donors (DonorID value of 4) donated $50 each. You need to add an artificial key, DonationID for example, to the Donation table. The addition of the artificial key makes every record in the Donation table unique, as shown in Figure A-19.

Figure A-19 Donation table after adding DonationID, an artificial key

Donation

DonationID	DonorID	DonationDate	DonationAmt
1	1	10/12/2015	$50.00
2	1	09/30/2016	$50.00
3	2	10/03/2016	$75.00
4	4	10/10/2016	$50.00
5	4	10/10/2016	$50.00
6	4	10/11/2016	$25.00
7	5	10/13/2016	$50.00

The descriptions of the Donor and Donation tables now are:

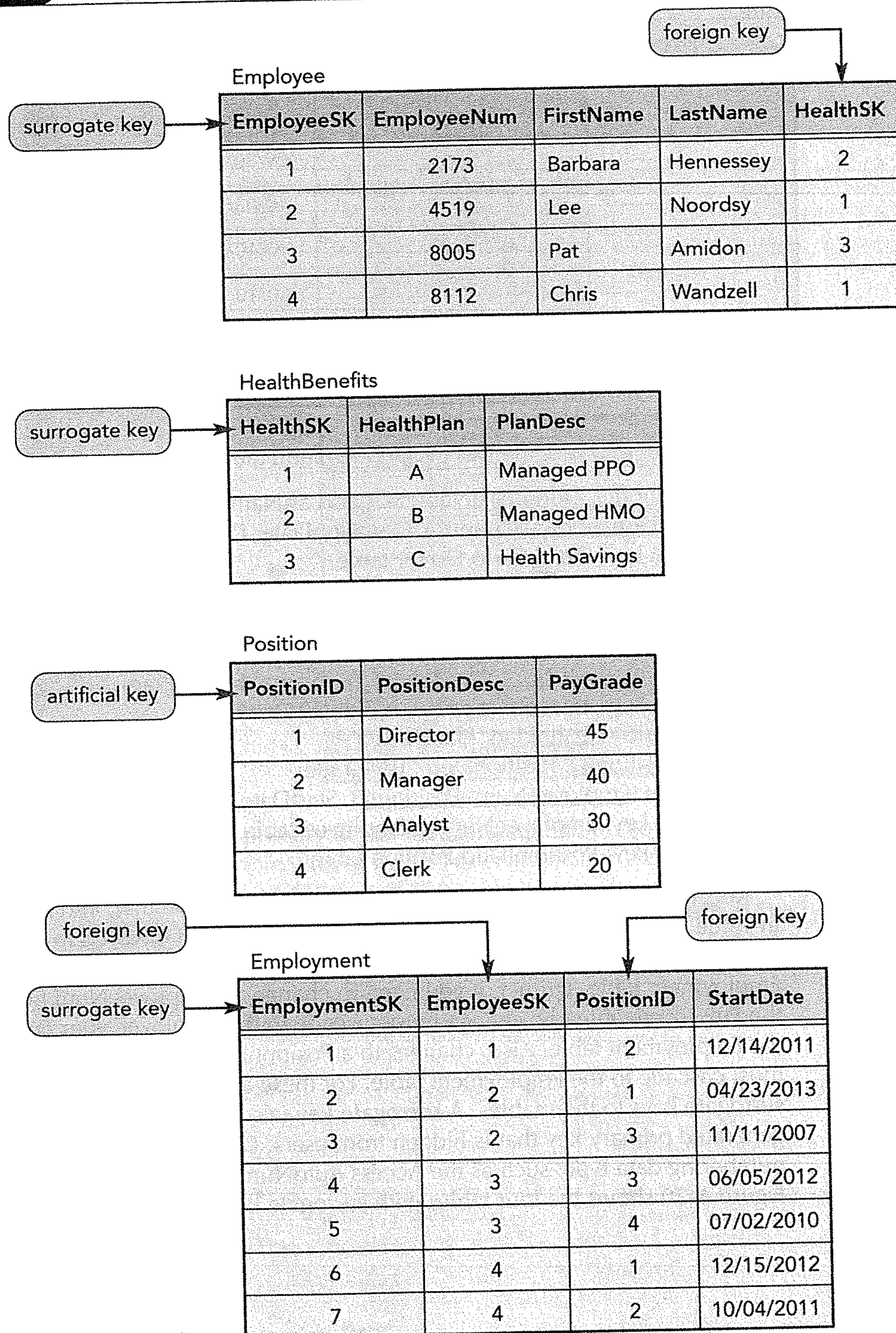
Donor (DonorID, DonorFirstName, DonorLastName)
Donation (DonationID, DonorID, DonationDate, DonationAmt)
  Foreign key: DonorID to Donor table

For another common situation, consider the 3NF tables you reviewed in the previous section (see Figure A-17) that have the following descriptions:

Employee (EmployeeNum, FirstName, LastName, HealthPlan)
  Foreign key: HealthPlan to HealthBenefits table
HealthBenefits (HealthPlan, PlanDesc)
Position (PositionID, PositionDesc, PayGrade)
Employment (EmployeeNum, PositionID, StartDate)
  Foreign key: EmployeeNum to Employee table
  Foreign key: PositionID to Position table

Recall that a primary key must be unique, must be minimal, and must not change in value. In theory, primary keys don't change in value. However, in practice, you might have to change EmployeeNum field values that you incorrectly entered in the Employment table. Further, if you need to change an EmployeeNum field value in the Employee table, the change must cascade to the EmployeeNum field values in the Employment table. Also, changes to a PositionID field value in the Position table must cascade to the Employment table. For these and other reasons, many experts add surrogate keys to their tables. A **surrogate key** (also called a **synthetic key**) is a system-generated primary key that is hidden from users. Usually you can use an automatic numbering data type, such as the Access AutoNumber data type, for a surrogate key. Figure A-20 shows the four tables with surrogate keys added to each table.

Figure A-20 Using surrogate keys



Employee

EmployeeSK	EmployeeNum	FirstName	LastName	HealthSK
1	2173	Barbara	Hennessey	2
2	4519	Lee	Noordsy	1
3	8005	Pat	Amidon	3
4	8112	Chris	Wandzell	1

HealthBenefits

HealthSK	HealthPlan	PlanDesc
1	A	Managed PPO
2	B	Managed HMO
3	C	Health Savings

Position

PositionID	PositionDesc	PayGrade
1	Director	45
2	Manager	40
3	Analyst	30
4	Clerk	20

Employment

EmploymentSK	EmployeeSK	PositionID	StartDate
1	1	2	12/14/2011
2	2	1	04/23/2013
3	2	3	11/11/2007
4	3	3	06/05/2012
5	3	4	07/02/2010
6	4	1	12/15/2012
7	4	2	10/04/2011

The HealthSK field replaces the HealthPlan field as a foreign key in the Employee table, and the EmployeeSK field replaces the EmployeeNum field in the Employment table. Now when you change an incorrectly entered EmployeeNum field value in the Employee table, you don't need to cascade the change to the Employment table. Likewise, when you change an incorrectly entered HealthPlan field value in the HealthBenefits table, you don't need to cascade the change to the Employee table.

As you design a database, you should *not* consider the use of surrogate keys, and you should use an artificial key only for the rare table that has duplicate records. At the point when you implement a database, you might choose to use artificial and surrogate keys, but be aware that database experts debate their use and effectiveness. You need to consider the following tradeoffs between natural and surrogate keys:

- You use surrogate keys to avoid cascading updates to foreign key values. Surrogate keys can also replace lengthier foreign keys when those foreign keys reference composite fields.
- You don't need a surrogate key for a table whose primary key is not used as a foreign key in another table because cascading updates is not an issue.
- Tables with surrogate keys require more joins than do tables with natural keys. For example, if you need to know all employees with a HealthPlan field value of A, the surrogate key in Figure A-20 requires that you join the Employee and HealthBenefits tables to answer the question. Using natural keys as shown in Figure A-17, the HealthPlan field appears in the Employee table, so no join is necessary.
- Although surrogate keys are meant to be hidden from users, they cannot be hidden from users who create SQL statements and use other ad hoc tools.
- Because you need a unique index for the natural key and a unique index for the surrogate key, your database size is larger and index maintenance takes more time when you use a surrogate key. On the other hand, a foreign key using a surrogate key is usually smaller than a foreign key using a natural key, especially when the natural key is a composite key, so those indexes are smaller and faster to access for lookups and joins.

Microsoft Access Naming Conventions

In the early 1980s, Microsoft's Charles Simonyi introduced an identifier naming convention that became known as Hungarian notation. Microsoft and other companies use this naming convention for variable, control, and other object naming in Basic, Visual Basic, and other programming languages. When Access was introduced in the early 1990s, Stan Leszynski and Greg Reddick adapted Hungarian notation for Microsoft Access databases; their guidelines became known as the Leszynski/Reddick naming conventions. In recent years, the Leszynski naming conventions, the Reddick naming conventions, and other naming conventions have been published. Individuals and companies have created their own Access naming conventions, but many are based on the Leszynski/Reddick naming conventions, as are the naming conventions covered in this section.

An Access database can contain thousands of objects, fields, controls, and other items, and keeping track of their names and what they represent is a difficult task. Consequently, you should use naming conventions that identify the type and purpose of each item in an Access database. You can use naming conventions that identify items generally or very specifically.

For an object, include a prefix tag to identify the type of object, as shown in Figure A-21. In each example in Figure A-21, the final object name consists of a three-character tag prefixed to the base object name. For example, the form name of frmEmployeesAndPositions consists of the frm tag and the EmployeesAndPositions base form name.

PRACTICE

Review Assignments

1. What are the formal names for a table, for a row, and for a column? What are the popular names for a row and for a column?
2. What is a domain?
3. What is an entity?
4. What is the relationship between a primary key and a candidate key?
5. What is a composite key?
6. What is a foreign key?
7. Look for and describe an example of a one-to-one relationship, an example of a one-to-many relationship, and an example of a many-to-many relationship in a newspaper, magazine, book, or everyday situation you encounter.
8. When do you use an entity subtype?
9. What is the entity integrity constraint?
10. What is referential integrity?
11. What does the cascades option, which is used with referential integrity, accomplish?
12. What are partial and transitive dependencies?
13. What three types of anomalies can be exhibited by a table, and what problems do they cause?
14. Figure A-24 shows the Employee, Position, and Employment tables with primary keys EmployeeNum, PositionID, and both EmployeeNum and PositionID, respectively. Which two integrity constraints do these tables violate and why?

Figure A-24 **Integrity constraint violations**

Employee

EmployeeNum	FirstName	LastName	HealthPlan
2173	Barbara	Hennessey	B
4519	Lee	Noordsy	A
8005	Pat	Amidon	C
8112	Chris	Wandzell	A

Position

PositionID	PositionDesc	PayGrade
1	Director	45
2	Manager	40
3	Analyst	30
4	Clerk	20

Employment

EmployeeNum	PositionID	StartDate
2173	2	12/14/2011
4519	1	04/23/2013
4519		11/11/2007
8005	3	06/05/2012
8005	4	07/02/2010
8112	1	12/15/2012
9876	2	10/04/2011

15. The State and City tables, shown in Figure A-4, are described as follows:
    State (StateAbbrev, StateName, EnteredUnionOrder, StateBird, StatePopulation)
    City (StateAbbrev, CityName, CityPopulation)
    Foreign key: StateAbbrev to State table
    Add the field named CountyName for the county or counties in a state containing the city to this database, justify where you placed it (that is, in an existing table or in a new one), and draw the entity-relationship diagram for all the entities. Counties for some of the cities shown in Figure A-4 are Travis and Williamson counties for Austin TX; Hartford county for Hartford CT; Clinton, Eaton, and Ingham counties for Lansing MI; Davidson county for Nashville TN; Hughes county for Pierre SD; and Nueces and San Patricio counties for Portland TX.
16. Suppose you have a table for a dance studio. The fields are dancer's identification number, dancer's name, dancer's address, dancer's telephone number, class identification number, day that the class meets, time that the class meets, instructor name, and instructor identification number. Assume that each dancer takes one class, each class meets only once a week and has

one instructor, and each instructor can teach more than one class. In what normal form is the table currently, given the following alternative description?

Dancer (DancerID, DancerName, DancerAddr, DancerPhone, ClassID, ClassDay, ClassTime, InstrName, InstrID)

Convert this relation to 3NF and represent the design using the alternative description method.

17. Store the following fields for a library database: AuthorCode, AuthorName, BookTitle, BorrowerAddress, BorrowerName, BorrowerCardNumber, CopiesOfBook, ISBN (International Standard Book Number), LoanDate, PublisherCode, PublisherName, and PublisherAddress. A one-to-many relationship exists between publishers and books. Many-to-many relationships exist between authors and books and between borrowers and books.
    a. Name the entities for the library database.
    b. Create the tables for the library database and describe them using the alternative method. Be sure the tables are in third normal form.
    c. Draw an entity-relationship diagram for the library database.
18. In the database shown in Figure A-25, which consists of the Department and Employee tables, add one record to the end of the Employee table that violates both the entity integrity constraint and the referential integrity constraint.

Figure A-25 Creating integrity constraint violations

Department

DeptID	DeptName	Location
M	Marketing	New York
R	Research	Houston
S	Sales	Chicago

Employee

EmployeeID	EmployeeName	DeptID
1111	Sue	R
2222	Pam	M
3333	Bob	S
4444	Chris	S
5555	Pat	R
6666	Meg	R

19. Consider the following table:

    Patient (PatientID, PatientName, BalanceOwed, DoctorID, DoctorName, ServiceCode, ServiceDesc, ServiceFee, ServiceDate)

    This is a table concerning data about patients of doctors at a clinic and the services the doctors perform for their patients. The following dependencies exist in the Patient table:

    PatientID → PatientName, BalanceOwed

    DoctorID → DoctorName

    ServiceCode → ServiceDesc, ServiceFee

PatientID, DoctorID, ServiceCode → PatientName, BalanceOwed, DoctorName, ServiceDesc, ServiceFee, ServiceDate

a. Based on the dependencies, convert the Patient table to first normal form.

b. Next, convert the Patient table to third normal form.

20. Suppose you need to track data for mountain climbing expeditions. Each member of an expedition is called a climber, and one of the climbers is named to lead an expedition. Climbers can be members of many expeditions over time. The climbers in each expedition attempt to ascend one or more peaks by scaling one of the many faces of the peaks. The data you need to track includes the name of the expedition, the leader of the expedition, and comments about the expedition; the first name, last name, nationality, birth date, death date, and comments about each climber; the name, location, height, and comments about each peak; the name and comments about each face of a peak; comments about each climber for each expedition; and the highest height reached and the date for each ascent attempt by a climber on a face with commentary.

    a. Create the tables for the expedition database and describe them using the alternative method. Be sure the tables are in third normal form.

    b. Draw an entity-relationship diagram for the expedition database.

21. What is the difference among natural, artificial, and surrogate keys?

22. Why should you use naming conventions for the identifiers in a database?