Chapter 1: Database Management

In This Chapter

- **Discovering the basics of databases**
- **Figuring out how to manipulate data**
- **Understanding database programming**

Database management is all about storing organized information and knowing how to retrieve it again. Although the idea of storing and retrieving data is simple in theory, managing databases can get complicated in a hurry. Not only can data be critical, such as bank records, but data retrieval may be time-sensitive as well. After all, retrieving a person's medical history in a hospital emergency room is useless if that information doesn't arrive fast enough to tell doctors that the patient has an allergic reaction to a specific antibiotic.

Because storing and retrieving information is so important, one of the most common and lucrative fields of computer programming is database management. Database management involves designing and programming ways to store and retrieve data. Because nearly every business from agriculture to banking to engineering requires storing and retrieving information, database management is used throughout the world.

The Basics of Databases

A database acts like a big bucket where you can dump in information. The two most important parts of any database is storing information and yanking it back out again. Ideally, storing information should be just as easy as retrieving it no matter how much data you may need to store or retrieve.

To store and retrieve data, computer scientists have created three types of database designs:

- **Free-form**
- **Flat-file**
- **Relational**

Free-form databases

Free-form databases are designed to make it easy to store and retrieve information. A free-form database acts like a scratch pad of paper where you can scribble any type of data, such as names and addresses, recipes, directions to your favorite restaurant, pictures, or a list of items that you want to do the next day. A free-form database gets its name because it gives you the freedom to store dissimilar information in one place, as shown in Figure 1-1.

Being able to store anything in a database can be convenient, but that convenience is like the freedom to throw anything you want in a closet, such as pants, books, written reports, and photographs. With such a wide variety of stuff dumped in one place, finding what you need can be much more difficult.

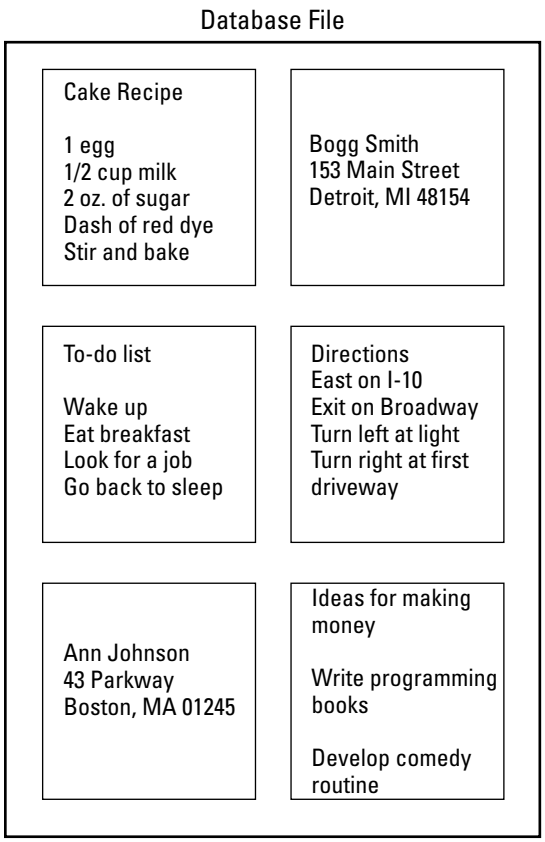
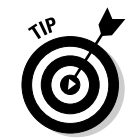


Figure 1-1: A free-form database can store randomly structured information.

To retrieve data from a free-form database, you need to know at least part of the data you want to find. So if you stored a name and phone number in a free-form database, you could find it again by just typing part of the name you want to find (such as typing **Rob** to find the name *Robert*). If you stored a recipe in a free-form database, you could find it by typing one of the ingredients of that recipe, such as **milk**, **shrimp**, or **carrots**.

Free-form databases have two big disadvantages:

- **They're clumsy for retrieving information.** For example, suppose you stored the name *Robert Jones* and his phone number *555-9378.* The only way to retrieve this information is by typing part of this data, such as **Rob**, **555**, or **nes**. If you type **Bob**, the free-form database doesn't find *Robert.* So it's possible to store information in a free-form database and never be able to find it again, much like storing a cherished photo album in an attic and forgetting exactly where it might be.
- **They can't sort or filter information.** If you want to see the phone numbers of every person stored in a free-form database, you can't. If you want to see only information in the free-form database that you stored in the past week, you can't do that either.



Because free-form databases are so limited in retrieving information, they're best used for simple tasks, such as jotting down notes or ideas but not for storing massive amounts of critical information. To store data with the ability to sort, search, and filter data to view only specific types of information, you need a flat-file database.

Flat-file databases

The biggest difference between a free-form database and a flat-file database is that a flat-file database imposes *structure.* Whereas a free-form database lets you type random information in any order, flat-file databases force you to add information by first defining the structure of your data and then adding the data itself.

Before you can store data, you must design the structure of the database. This means defining what type of data to store and how much room to allocate for storing it. So you might decide to store someone's first name and last name and allocate up to 20 characters for each name.

Each chunk of data that you want to record, such as a first name, is a *field.* A group of fields is a *record.* If you want to store names and telephone numbers, each name and telephone number is a field, and each name and its accompanying telephone number make up a single record, as shown in Figure 1-2.

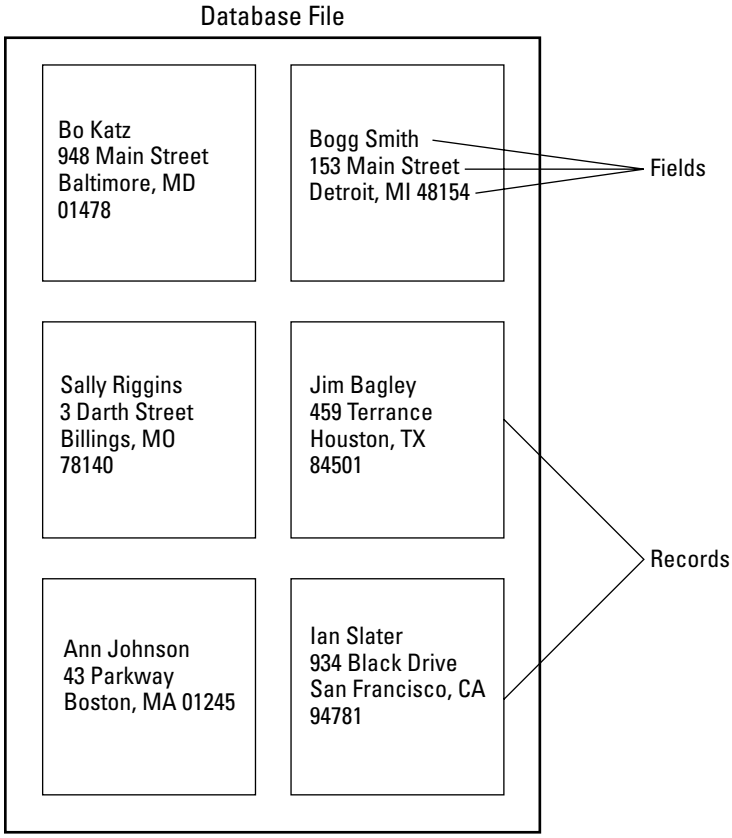


Figure 1-2: A flat-file database stores data in fields and records.

Flat-file databases impose a structure on the type of information you can store to make retrieving information much easier later. However, you need to design the structure of your database carefully. If you define the database to store only first and last names, you can't store any other information other than first and last names.

Designing the size and types of fields can be crucial in using the database later. If you create a Name field but allocate only ten characters to hold that data, the name *Bob Jones* fits but another name, such as *Daniel Jonathan Perkins,* cuts off.

Another problem is how you define your fields. You could store names in one big field or separate them into three separate fields for holding first, middle, and last names. Using a single field to store a name might initially look simpler, but separating names in different fields is actually more useful because this allows the database to sort by first, middle, or last name.

Although such a rigid structure might seem to make flat-file databases harder to use, it does make flat-file databases easier to search and sort information. Unlike free-form databases that may contain anything, every record in a flat-file database contains the exact same type of information, such as a name, address, and phone number. This makes it possible to search and sort data.

If you want to find the telephone number of *Robert Jones,* you could tell the flat-file database to show you all the records that contain a first name beginning with the letter R. If you want to sort your entire database alphabetically by last name, you can do that, too.

A flat-file database gets its name because it can work only with one file at a time. This makes a flat-file database easy to manage but also limits its usefulness. If you have a flat-file database containing names and addresses and a second flat-file database containing names and telephone numbers, you might have identical names stored in the two separate files. Change the name in one flat-file database and you need to change that same name in the second flat-file database.

Relational databases

For storing simple, structured information, such as names, addresses, and phone numbers (similar to a Rolodex file), flat-file databases are adequate. However, if you need to store large amounts of data, you're better off using a *relational database*, which is what the majority of database programs offer.

Like a flat-file database, you can't store anything in a relational database until you define the number and size of your fields to specify exactly what type of information (such as names, phone numbers, and e-mail addresses) that you want to save.

Unlike flat-file databases, relational databases can further organize data into groups, or *tables.* Whereas a free-form database stores everything in a file and a flat-file database stores everything in file, but organizes it into fields; a relational database stores everything in a file that's divided into tables, which are further divided into fields, as shown in Figure 1-3.

Think of database tables as miniature flat-file databases that can connect with each other.

Just as storing a name in separate First Name and Last Name fields gives you more flexibility in manipulating your data, grouping data in separate tables also give you more flexibility in manipulating and sharing information.

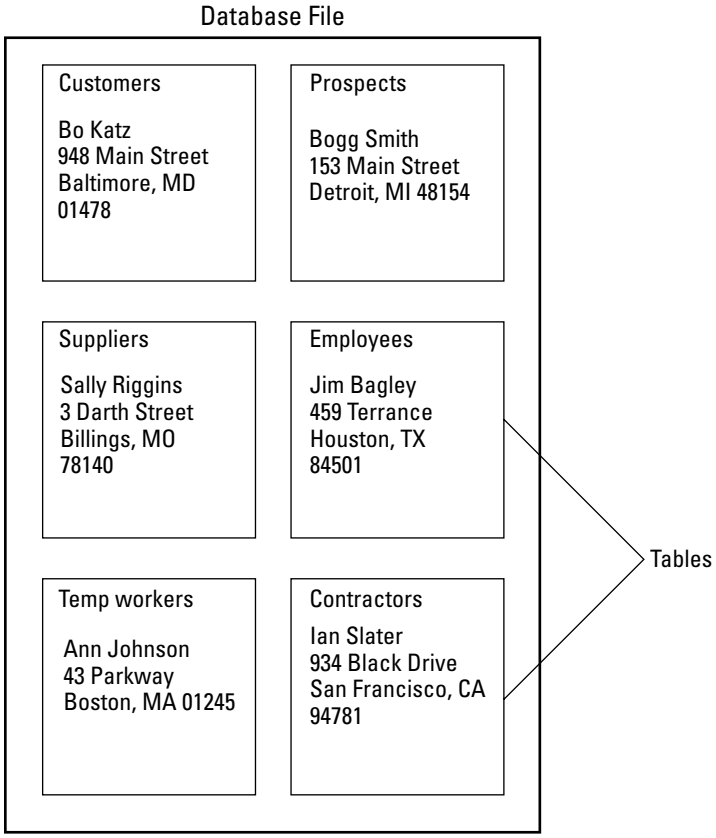


Figure 1-3: A relational database is divided into tables.

Suppose you have a list of employees that includes names, addresses, and telephone numbers. Now you may want to organize employees according to the department where they work. With a flat-file database, you'd have to create a separate file and store duplicate names in these separate databases, as shown in Figure 1-4.

Every time you add a new employee, you'd have to update both the employee database and the specific department database that defines where he works. If an employee leaves, you'd have to delete his name from two separate databases as well. With identical information scattered between two or more databases, keeping information updated and accurate is difficult.

Relational databases solve this problem by dividing data into tables with a table grouping the minimum amount of data possible. So, one table might contain names and employee ID whereas a second table might contain only employee names and department names, as shown in Figure 1-5.

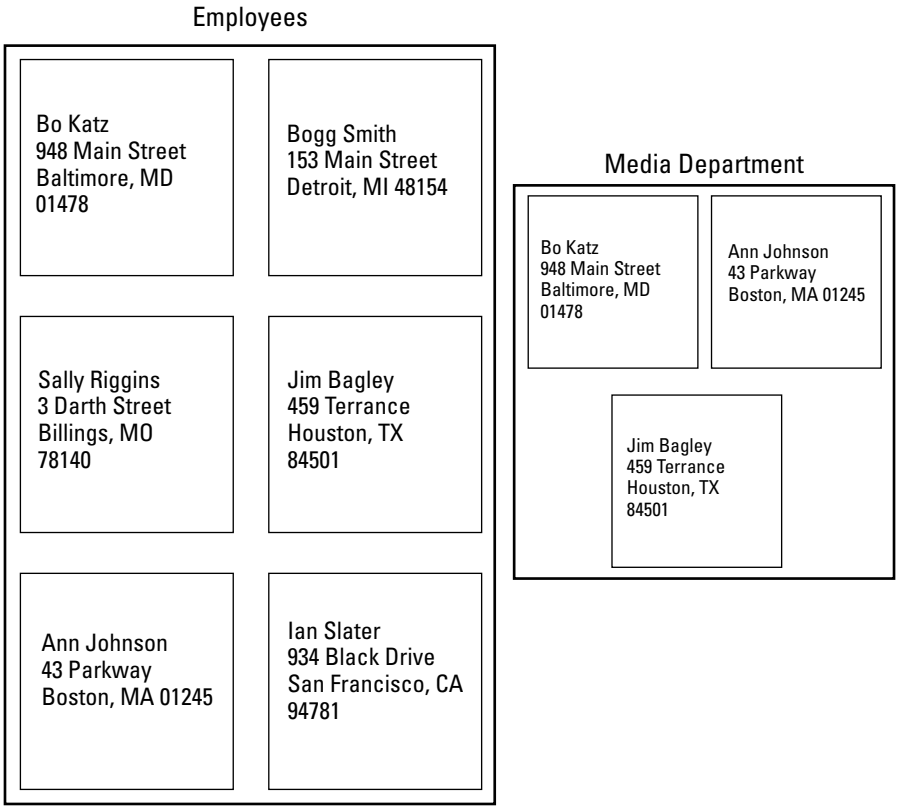


Figure 1-4: Flat-file databases must store duplicate data in separate files.

Table

Name	Employee ID
Bill Adams	4Y78
Sally Tarkin	8U90
Johnny Brown	4T33
Doug Hall	4A24
Yolanda Lee	9Z49
Sam Collins	1Q55
Randy May	2E03
Al Neander	4M79
Kal Baker	2B27

Table

Name	Department
Bill Adams	Public relations
Sally Tarkin	Human resources
Johnny Brown	Engineering
Doug Hall	Engineering
Yolanda Lee	Human resources
Sam Collins	Engineering
Randy May	Public relations
Al Neander	Public relations
Kal Baker	Human resources

Figure 1-5: Tables separate data into pieces.

A column in a table represents a single field, often called an *attribute*. A row in a table represents a single record, often called a *tuple*.

What makes tables useful is that you can link them together. So whereas one table may appear to contain names and addresses while a second table might also contain names and departments, the two tables are actually sharing information. Instead of having to type a name twice in both tables, you need to type the name only once, and the link between separate tables automatically keeps that information updated and accurate in all other linked tables.

By linking or relating tables together, you can combine data in different ways. If you have a list of customers stored in one table and a list of sales in another table, you can relate these two tables to show which customers are buying which products, or which products are most popular in specific sales regions. Basically, relating tables together allows you to create *virtual* databases by sharing and combining data from separate database tables. By combining data from separate tables, you can uncover hidden information behind your data, as shown in Figure 1-6.

Table

Name	Employee ID
Bill Adams	4Y78
Sally Tarkin	8U90
Johnny Brown	4T33
Doug Hall	4A24
Yolanda Lee	9Z49
Sam Collins	1Q55
Randy May	2E03
Al Neander	4M79
Kal Baker	2B27

Table

Name	Department
Bill Adams	Public relations
Sally Tarkin	Human resources
Johnny Brown	Engineering
Doug Hall	Engineering
Yolanda Lee	Human resources
Sam Collins	Engineering
Randy May	Public relations
Al Neander	Public relations
Kal Baker	Human resources

Name	Employee ID	Department
Bill Adams	4Y78	Public relations
Sally Tarkin	8U90	Human resources
Johnny Brown	4T33	Engineering
Doug Hall	4A24	Engineering
Yolanda Lee	9Z49	Human resources
Sam Collins	1Q55	Engineering
Randy May	2E03	Public relations
Al Neander	4M79	Public relations
Kal Baker	2B27	Human resources

Figure 1-6: Relational databases let you combine data from different tables.

Tables divide data into groups, but taken on a larger scale, it's possible to divide an entire database into multiple databases that are physically separate. Such databases are *distributed databases.*

A company might use a distributed database to keep track of all its employees. A branch office in Asia might have a database of employees in Singapore, another branch in Europe might have a database of employees in England, and a third branch in America might have a database of employees in California. Combining these separate databases would create a single database of all the company's employees.

Manipulating Data

After you define the structure of a database by organizing information in tables and fields, the next step is to write commands for modifying and manipulating that information. This can be as simple as adding data to a specific table or as complicated as retrieving data from three different tables, reorganizing this information alphabetically by last name, and displaying this list on the screen with mathematical calculations showing sales results for each person and a total amount for an entire department and company.

The three basic commands for manipulating data are *Select, Project,* and *Join.* The Select command retrieves a single row or tuple from a table. So if you want to retrieve someone's name to find her e-mail address, you could use the Select command, as shown in Figure 1-7.

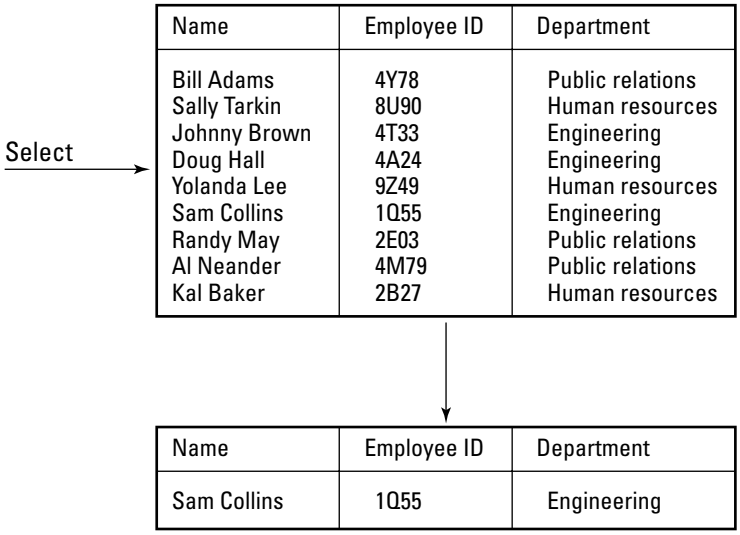


Figure 1-7: The Select command retrieves a single record or tuple.

Besides retrieving a single record or tuple, the Select command can retrieve multiple tuples, such as a list of all employees who work in a certain department.

The Project command retrieves the entire column or attribute from a database table. This can be useful when you just want to view certain information, such as the names of employees along with the department where they work, as shown in Figure 1-8.

The Project command acts like a filter, hiding data that you don't want to see and displaying only data that you do want to see. Combining the Select and Project commands can find just the names and e-mail addresses of all employees who work in a certain department.

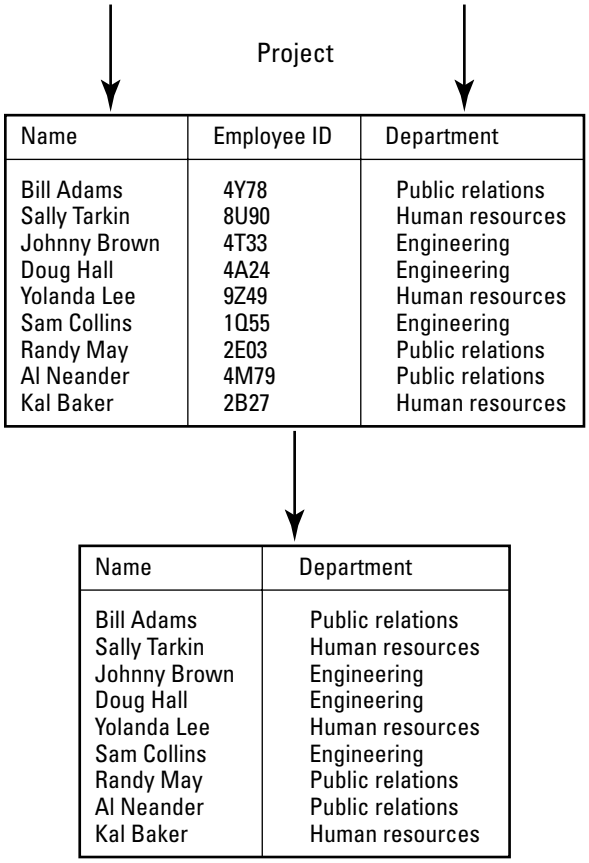


Figure 1-8: The Project command retrieves an entire column or attribute.

The Join command combines separate tables together to create a virtual table that can show new information. For example, a Join command might combine a table of products, and a table of customers with a table of sales people can show which sales person is best at selling certain products and which products are most popular with customers, as shown in Figure 1-9.

The Select, Project, and Join commands are generic terms. Every database uses its own terms for performing these exact same actions, so be aware of these terminology differences.

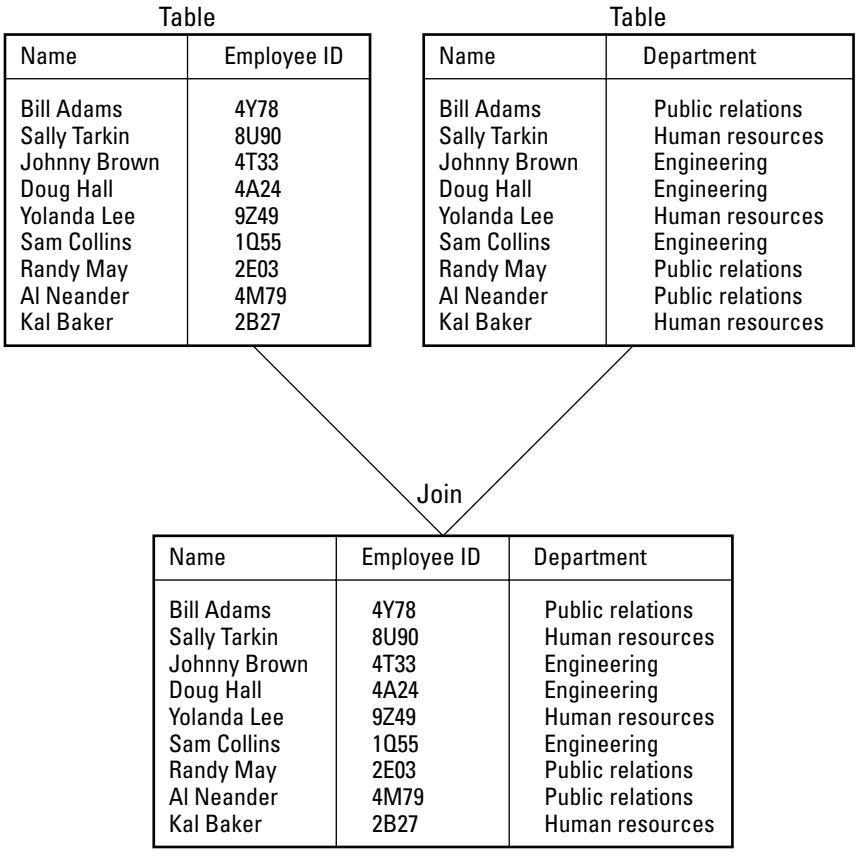


Figure 1-9: The Join command matches two tables together.

Writing database commands

Every relational database program includes commands for manipulating data. These commands essentially form a proprietary programming language specific to that database program. Microsoft Access uses a programming language called VBA (Visual Basic for Applications) whereas FileMaker uses a language called FileMaker Script. Most databases actually consist of separate files with one file containing the actual data and a second file containing programs for manipulating that data.

The main difference between a general purpose language, like C++, and a database language is that the database language needs only to define what data to use and how to manipulate it, but the database program (or the *database engine*) takes care of the actual details. Database languages only need to define what to do, not how to do it.

The SQL language

Although every relational database program comes with its own language, the most popular language for manipulating large amounts of data is SQL (Structured Query Language). SQL is used by many different database programs, such as those sold by Oracle, Microsoft, and Sybase. If you're going to work with databases as a programmer, you have to figure out SQL.

SQL commands, like all database programming languages, essentially hide the details of programming so you can focus on the task of manipulating data. To retrieve names from a database table named Employees, you could use this SQL command:

```
SELECT FirstName, LastName FROM Employees
```

To selectively search for certain names, you could use this variation:

```
SELECT FirstName, LastName FROM Employees
WHERE FirstName = 'Richard'
```

To add new information to a table, you could use this command:

```
INSERT INTO Employees
VALUES ('John', 'Smith', '555-1948')
```

To delete information from a table, you could use the delete command, such as:

```
DELETE FROM Employees
WHERE LastName = 'Johnson'
```

To modify a phone number, you could use this command:

```
UPDATE Employees
```

```
SET PhoneNumber = '555-1897'
WHERE LastName = 'Smith'
```

An ordinary user can type simple database commands to retrieve information from a database, but because most users don't want to type a series of commands, it's usually easier for someone else to write commonly used database commands and then store these commands as miniature programs. Then instead of forcing the user to type commands, the user can just choose an option and the database will run its program associated with that option, such as sorting or searching for information.

Data integrity

With small databases, only one person may use the database at a time. However with large databases, it's possible for multiple people to access the database at the same time. The biggest problem with multi-user databases is data integrity.

Data integrity is insuring that data is accurate and updated, which can cause a problem when multiple users are modifying the same data. An airline reservation system might let multiple travel agents book seats on the same airplane, but the database must make sure that two travel agents don't book the same seat at the same time.

To prevent two users from modifying the same data, most database programs protect data by letting only the first user modify the data and locking others out. While the first user is modifying the data, no one else can modify that same data.

Locking can prevent two users from changing data at the same time, but sometimes, changing data may require multiple steps. To change seats on an airline reservation system, you may need to give up one seat (such as seat 14F) and take another one (such as seat 23C). But in the process of giving up one seat, it's possible that another user could take the second seat (23C) before the first user can, which would leave the first user with no seats at all.

To prevent this problem, database programs can lock all data that a user plans to modify, such as preventing anyone from accessing seats 14F and 23C. Another solution to this problem is a *rollback.* If a second user takes seat 23C before the first user can get to it, the database program can rollback its changes and give the first user back the original seat 14F.

Multi-user databases have algorithms for dealing with such problems, but if you're creating a database from scratch, these are some of the many problems you need to solve, which explains why most people find it easier just to use an existing database program rather than write their own database program.

Data mining

Large chunks of data are *data warehouses. Data mining* simply looks at separate databases to find information that's not obvious in either database. For example, one database might contain tax information, such as names, addresses, and Social Security numbers. A second database might contain airline passenger information, such as names, addresses, and flight numbers. A third database might contain telephone calling records that contain names, addresses, and phone numbers called.

By themselves, these separate databases may seem to have no connection, but link the tax database with an airline passenger database and you can tell which passengers traveled to certain countries and reported an income less than $25,000. Just by combining these two databases, you can flag any suspicious travel arrangements. If someone reports income under $25,000, but has made ten different trips to Saudi Arabia, Venezuela, and the Philippines, that could be a signal that something isn't quite right.

Now toss in the telephone calling database and you can find everyone who reported less than $25,000 income, made ten or more overseas trips to other countries, and made long-distance phone calls to those same countries. Retrieving this type of information from multiple databases is what data mining is all about.

Data mining finds hidden information stored in seemingly innocuous databases. As a result, data mining can be used to track criminals (or anti-government activists) and identify people most likely to have a genetic disposition to certain diseases (which can be used for preventative treatment or to deny them health insurance). With so many different uses, data mining can be used for both helpful and harmful purposes.

Database Programming

At the simplest level, a database lets you store data and retrieve it again. For storing a list of people you need to contact regularly, a Rolodex-type database can be created and used with no programming. However, if you want to store large amounts of data and manipulate that information in different ways, you may need to write a program.

The three parts of a database program include the *user interface*, the *database management system* (which contains commands for manipulating data), and the actual *information* stored in a database, as shown in Figure 1-10.

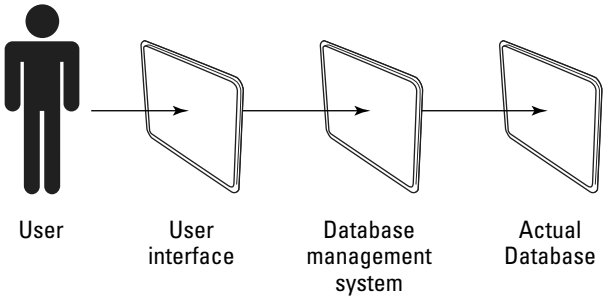


Figure 1-10: The three parts of a database program.

The user interface lets people use the data without having to know how the data is stored or how to write commands to manipulate the data. The database stores the actual information, such as dividing data into tables and fields. The commands for manipulating that data may include printing, searching, or sorting through that data, such as searching for the names of all customers who recently ordered over $10,000 worth of products in the past month.

Here are three ways to write a database program. The first way is to use an ordinary programming language, such as C++ or BASIC. The problem with using a programming language like C++ is that you have to create all three parts of a database from scratch. Although this gives you complete control over the design of the database, it also takes time.

As a simpler alternative, many programmers buy a database toolkit, written in their favorite programming language, such as C++. This toolkit takes care of storing and manipulating data, so all you need to do is design the database structure (tables and fields) and create the user interface.

A second way to write a database program is to start with an existing relational database program and use its built-in programming language to create a user interface and the commands for manipulating the data. The advantage of this approach is that you don't have to create an entire database management system from scratch, but the disadvantage is that you have to know the proprietary language of that particular database, such as Microsoft Access or FileMaker.

A third way to write a database program involves a combination of existing database programs and general-purpose programming languages, like C++:

1. **Use a database program to design the structure of the data and then you use the database's programming language to write commands for manipulating that data.**

2. **Use your favorite programming language, such as C++ or BASIC, to create a user interface for that database.**

   This approach takes advantage of the database program's strengths (designing database structures and manipulating data) while avoiding its weakness in designing user interfaces. General-purpose languages, like C++ or BASIC, are much better for designing user interfaces, which can make your database program much easier to use.

If you have a lot of time on your hands, create an entire database from scratch with C++ or BASIC. However, if you don't want the hassle of creating an entire database management system yourself, buy a commercial database program and customize it using the database program's own programming language. This second approach is the most common solution for creating database programs.

If you find a database programming language too clumsy or too restrictive for designing user interfaces, write your own user interface in your favorite programming language and slap it on an existing database program. This may involve the hassle of integrating your user interface (written in C++) with the database file and data manipulating commands created by a database program (such as Microsoft Access).

Ultimately, database programming involves making data easy to access and retrieve no matter which method you choose. Because storing information is so crucial in any industry, database programming will always be in demand. If you figure out how to design and program databases, you'll always have plenty of work to choose from.