PART OF THE PICTURE: Database Systems

BY KEITH VANDER LINDEN, CALVIN COLLEGE

One of our local supermarket chains keeps detailed records of its products, sales, customers, and supplies. They can tell you the current price of any item in any of their stores, who supplied it, and how long it's been on the shelf. They can tell you how many people bought tortilla chips at the rock-bottom sale price last weekend and whether they also bought salsa and bean dip to go with them. Not surprisingly, this level of record keeping produces a staggering volume of information, commonly called **data.** Why do they go to all this trouble? The answer—they have to make business decisions and these decisions are based on data. The more accurate and detailed the data, the better the decisions can be. Was the rock-bottom sale price for the chips too high, or too low? Did they make an overall profit? How many units of chips should they buy next time? The data can help answer these questions, and thus are critical to business in a competitive market.

This use of data is not unique to the grocery business. Banks keep records of our accounts and our transactions, universities and colleges keep records of our tuition costs and our performance, airlines keep records on which planes are flying where and who has paid to ride on them. As in the case of the supermarket, these data sets can become very large, and must be maintained for long periods of time. Furthermore, they must be conveniently accessible to many people. It is useful, therefore, to store and maintain them on computers. The data sets themselves, when stored on a computer, are commonly called **databases,** and the programs designed to maintain them are called **database management systems.**

The current chapter has discussed files, and it is not hard to see that their ability to store large amounts of data in a persistent manner is important for database systems. Section 8.1 gave an example of how files can be used to maintain a small meteorological database. As we work with that database, however, a problem arises: The database will change, as all realistic databases do. We may, for example, want to add temperature readings to the pressure readings already contained there. Adding this information to the file is easy enough, but we must also modify the code by adding an additional variable to store the temperature, an extra input command to read the value, and we must know the order in which the pressure and temperature readings are stored in the file. Another complication would arise if we wanted to add information on the particular instruments used to collect the readings. This would probably require a separate file of information for each of the instruments, new code to read and process this file, and some additional code allowing us to record which readings were taken with which instruments.

What started as a fairly simple problem, with a small driver program and a single data file, has become much more complex. In addition, this complexity is likely to increase as the database grows, and as more and more people want to access it. Database systems are designed to address these problems. Although varied, they tend to provide a number of common facilities.

- *High-level views of the data*—Databases allow programmers to view the data at a higher level, ignoring some of the details of the location and format of the files.
- *Access routines*—A high-level view of the data wouldn't be of much use if the programmer couldn't retrieve and manipulate the data. Database systems, therefore, provide what is called a **query language,** which provides a set of operations that a programmer can use when accessing and manipulating a database.
- *Support for large databases*—Databases tend to be large, frequently too large to fit into a computer's main memory. For example, the supermarket database mentioned earlier maintains approximately one *terabyte* of data, that's 10^{12} bytes or 1,000 gigabytes. Database systems are designed to manipulate large databases such as this without reading them into memory all at once.
- *Security*—It is frequently important to restrict access to sensitive or proprietary data in a database. Most database systems provide this capability.
- *Data sharing*—The data stored in a database is often of interest to many different people. Several people may, therefore, want to retrieve or modify the data at the same time. Database systems typically provide a check-in/check-out protocol for the data, much like the protocol for checking out books at a public library. This ensures that only one person can manipulate the data at a time.

THE RELATIONAL MODEL

Different models may be used to design a database system, each of which attempts to provide the features just mentioned. One particular model, called the **relational model,** has become an industry standard. In this model, the database is viewed as a set of tables with one row for each entry. For instance, the data from a simple employee database would be viewed as follows:

Name	IDNumber	Pay Rate	Department
John	45678	7.50	Accounting
Mark	56789	8.75	Accounting
Paul	67891	9.35	Marketing
Matt	78912	10.50	Accounting
Gabe	89123	6.35	Marketing
Joel	91234	10.50	Development
Naomi	98765	7.15	Development
Jamie	12345	9.15	Development
April	23456	8.75	Accounting
Jodi	34567	10.50	Service

In this set of data, called a **table,** the top row specifies the contents of each column of data. In this case we have the employee's name, ID number, pay rate, and department. The table has one row, called a **record,** for each employee, and each record has one entry for each column, called a **field.** We will call this table the `Employee` table. Note that there is no mention here of the files that contain the data, or in what format the data is represented. The database system takes care of these details so the programmer doesn't have to.

One popular query language for the relational model is called **SQL** (for "Structured Query Language"). SQL provides commands to add data for tables, retrieve data from tables, and modify data in tables. For example, consider the following command:

```
SELECT *
FROM Employee
WHERE Rate = 10.5;
```

This command will retrieve, or *select*, all the records (specified by the `"*"`) from the `Employee` table that have a pay rate of $10.50. The resulting records are as follows:

Name	IDNumber	Pay Rate	Department
Matthew	78912	10.50	Accounting
Joel	91234	10.50	Development
Jodi	34567	10.50	Service

Note that the result is itself a table. This elegant feature allows the output of one SQL command to be used as the input to another.

AN EXAMPLE: THE SELECT COMMAND

To gain a better understanding of the relationship between file manipulation and database management, we will build a simple relational database system with a scaled-down implementation of the select command. This implementation will go in a library so we'll be able to see the differences between the file manipulation used to implement the select command (in the library), and the use of the command itself (in a driver program).

The select command assumes a relational view of the database, so we must implement one. Recall that the relational data model views a database as a table or set of tables where the structure of each table can be specified by indicating the fields it will contain, and the type of data stored in each field. This structure is called a **schema.** The schema for the `Employee` table is as follows[1]:

```
Name       string
ID         int
Rate       double
Department string
```

This schema says that `Employee` will contain a record for each employee and that each record will contain the employee's name (a `string`), the employee's ID number (an `int`), his or her pay rate (a

1. Note that a full database system would maintain a considerably more complex schema than this, but this will serve our current needs.

`double`), and a department name (a `string`). We will store this schema in a schema file named `Employee.schema`.

The actual data for `Employee` will be stored in a separate data file named `Employee.data`. The contents of this file for the `Employee` table are as follows:

```
John     45678   7.50  Accounting
Mark     56789   8.75  Accounting
Paul     67891   9.35  Marketing
Matt     78912  10.50  Accounting
Gabe     89123   6.35  Marketing
Joel     91234  10.50  Development
Naomi    98765   7.15  Development
Jamie    12345   9.15  Development
April    23456   8.75  Accounting
Jodi     34567  10.50  Service
```

Notice how the columns in `Employee.data` match the rows in `Employee.schema`. The first column is the name (a `string`), second column is the ID number (an `int`), and so forth.

The representation of the `Employee` table, therefore, consists of two files: `Employee.schema` and `Employee.data`. We can now implement a `select()` function which makes use of these files. The function will have four parameters:

- the name of a table to operate on (e.g., `Employee`);
- the name of a field on which to select (e.g., `Rate`);
- the relational operator to use (we'll assume == for now);
- the field value to check (e.g., `10.5`).

It will return all the records in the given table whose specified field is equal to the specified value. So, for example, the C++ function call equivalent to the select command given would be:

```
select("Employee", "RATE", "=", "10.50");
```

The following program is a simple driver for this function that prompts the user of the system for the name of a table, a field name, a relational operator, and an appropriate value for that field. The input format is very similar to that of the SQL select command shown. It then calls the `select()` function.

POP 9-1 A Driver Program to Test the Select Function.

```
...
int main()
{
  string
    tableName,          // table, not its file name
    projectedFieldName, // the field being projected
    selectFieldName,    // field that you want to restrict
    selectOperator,     // rel-operator for the restriction
    selectFieldValue;   // value the field should have

  // Get the specifics of the select command from the user
  // (using a format similar to the SQL select statement.
  cout << "\nWelcome to SelectBase, a simple database system"
          "\n\nPlease complete the following query: "
          "\n\nSELECT *";
  cin >> projectedFieldName;
  cout << "\nFROM ";
  cin >> tableName;
  cout << "WHERE ";
  cin >> selectFieldName >> selectOperator
      >> selectFieldValue;
  // Execute the Select command.
  select(tableName, selectFieldName, selectOperator, selectFieldValue);
}
```

Sample run:

```
Welcome to SelectBase, a simple database system

Please complete the following query:

SELECT *
FROM Employee
WHERE Rate = 10.5

Matt     78912  10.50  Accounting
Joel     91234  10.50  Development
Jodi     34567  10.50  Service
```

This driver function illustrates the level of database programming. There are no file manipulation commands here, no `for` loops, no I/O streams, and the programmer doesn't know anything about the `.schema` and `.data` files or their format. All of these things are maintained by the implementation of `select()`, and thus do not concern the database programmer.

To see inside the implementation of `select()`, consider the code segment shown in the following program. This is where we place the nested loop that reads each field (as specified in `Employee.schema`) for each record (as contained in `Employee.data`). To make the implementa-

tion more general, the order and types of the fields are specified by the contents of the `.schema` file, rather than being hard-coded. This way, `select()` can manipulate many different tables, so long as they have consistent schema and data files.

POP 9-2 Nested for Loops for the `select()` Function.

```
for (;;)  // for each record in the data file
  {
    outputLine = "";    // add to this one field at a time.
    selectLine = false; // Assume the record is not selected.

    for (;;) // for each field in the schema file
      {
        // Get next field specification from the schema file.
        schemaFile >> currentFieldName >> currentFieldType;
        if ( schemaFile.eof() ) break;

        dataFile >> currentFieldValue;
        if ( dataFile.eof() )
        {
            schemaFile.close();
            dataFile.close();
            return;
        }

        // See if this record satisfies the selection criteria
        if (recordSelected(selectFieldName, currentFieldName,
                           selectFieldValue, currentFieldValue,
                           currentFieldType))
             selectLine = true;

        // Add the new field to the output string.
        outputline += currentFieldValue + "\t ";
      }

    // Output the current record if it has been selected.
    if (selectLine)
      cout << outputLine << endl;

    // Prepare to do a second pass through the schema file.
    schemaFile.clear();              // reset flags
    schemaFile.seekg(0, ios::beg);   // reset get position
  }
```

Exercises

1. Because the driver program discussed in this section reads relational tables based on a schema, you can perform many useful database operations without modifying the code. Try some of the following:
   a) Add a field to the database for the employee's last name. Remember that you'll have to modify both the schema and the data file in a consistent manner.
   b) Create a new table for a college bookstore that maintains information on books (e.g., the title, author, ISBN number, and price).
   c) Create a new table for your class, including fields for student names, ID numbers, and grades for the various projects you've done.
2. Modify the program in this section to add column headers to the table output. The headers should be based on the names of the fields given in the schema file.
3. The implementation of `select()` given in this section always uses the `==` operator. Modify the program to implement the other relational operators as well (i.e., `<`, `>`, `<=`, `>=`, and `!=`).
4. Modify the program from exercise 3 to write the results of the select command to a file rather than writing them to `cout`. The program should prompt the user for an output file name. It should then write the results to this file, or to `cout` if no filename is given. Now, use this program to select the following records:
   a) Find the employees in `Employee` with a pay rate equal to $10.50 and a Department of "Accounting".
   b) Find the employees in `Employee` with a pay rate greater than $8.00 and less than $10.00.
5. The relational model also supports a command that returns a specified column (or columns) from a table rather than returning the full record. This command, called **project,** is implemented in SQL by allowing the programmer to replace the `"*"` in the `SELECT` clause of the select command with a field name (or names).

   Modify the program in this section to support this. You should allow the user to enter either a `"*"` in the `SELECT` clause, or a specific field name. For example, if the user enters `*` in the `SELECT` clause, the full records will be returned:

```
Welcome to SelectBase, a simple database system

Please complete the following query:

SELECT *
FROM Employee
WHERE Department = Development

Joel   91234  10.50  Development
Naomi  98765   7.15  Development
Jamie  12345   9.15  Development
```

   If, on the other hand, the user enters a field name in the `SELECT` clause, only the value of that field will be returned:

```
Welcome to SelectBase, a simple database system

Please complete the following query:

SELECT Name
FROM Employee
WHERE Department = Development

Joel
Naomi
Jamie
```

FURTHER READING

The field of database systems is active, both in research and in applications. If you are interested in reading additional material, consider going to the following sources:

- Ullman and Widom's text *A First Course in Database Systems*, Prentice Hall, 1997—This is a good, current overview of the field of database systems. It covers the relational model, and also deals with newer object-oriented approaches to database modeling.
- E. F. Codd's original paper, "A Relational Model of Data for Large Shared Data Banks," *Communications of the ACM*, 1970, 13(6), pages 377–387—This is a classic paper on the relational database model.
- For information on some current database systems, try visiting the Oracle or Sybase corporation web sites: `www.oracle.com; www.sybase.com`.