The destructor method attempts to connect to the database. If the connection is successful, the method removes any preexisting `Dogs` table and creates a new one with the required fields. (Note: It would probably be better to rename the old one and create a new one.). If the old table can be removed and the new table created, then the method attempts to insert rows into the table. The SQL `INSERT` statement places the values from `$dog_name`, `$dog_weight`, `$dog_color`, and `$dog_breed` into a row in the table. The `foreach` loops retrieve each row from the associate array to be placed into the table. If any of the inserts are not successful, an exception is thrown. An example program is located under Chapter 6 on the book's web site.

Programming note—The Apache server must be properly configured and MySQL must be properly installed to run this (or a similar) database example. `$server` must be set to the URL, "localhost", or "127.0.0.1". `$db_username` must be set to the user ID name to access the database (`'root'` if a user ID has not been configured). `$db_password` must be set to the database password (or `''` if there is no password). `$database` must be set to the database name. There is a large varieties of ways to access and manipulate databases in the PHP language.

Do It

1. Download the example files for this section from the book's web site. Adjust the `deleteRecords` method to allow the ability to delete multiple records. However, also include a check to limit the amount of records that can be deleted. It would not be very secure to allow all records to be deleted. If an attempt is made to delete all records (or too many records), an exception should be raised. The exception should cause the calling program (eventually `dog_interface`) to write an error message to the main log file, e-mail the support personnel, and display the general message to the users (shown in Chapter 5). Adjust the `testdata` program to test the ability to delete multiple records and catch the exceptions.
2. Download the example files for this section from the book's web site. Adjust the `testdata` program to test all remaining scenarios that have not already been tested. These are related to inserting, updating (more than one), reading, and deleting records. Be sure to test improperly formatted information. Create a `try catch` block in the `testdata` program to capture any exceptions. You can use the `try catch` block from `dog_interface` in Chapter 5 as an example.

Backup and Recovery

There is always a possibility that something can go wrong when changes are made to stored information. While a well-developed application must filter and clean data before it is saved; it must also be prepared to handle the possibility that bad data may still flowed through and corrupt the information. In addition to intentional corruption, unforeseen problems (such as system crashes) may occur. An application must provide the ability to recover without the loss of data. This can be accomplished by logging change requests and backing up valid information. Recovery can be accomplished by using a valid backup and reapplying valid changes to the backup files to produce up-to-date information.

You can make just a few minor changes to the `dogdata` file (Example 6-1) to create a change log and to provide backup and recovery capability. First, you will create a main method (`processRecords`) that will interpret any data passed into the class. This function will simplify the recovery process by allowing the recovery program to pass all change log information into one method. This will also make dependency injection easier to accomplish.