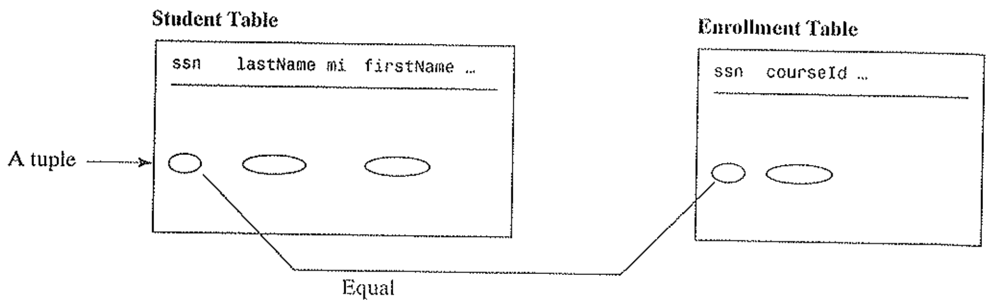
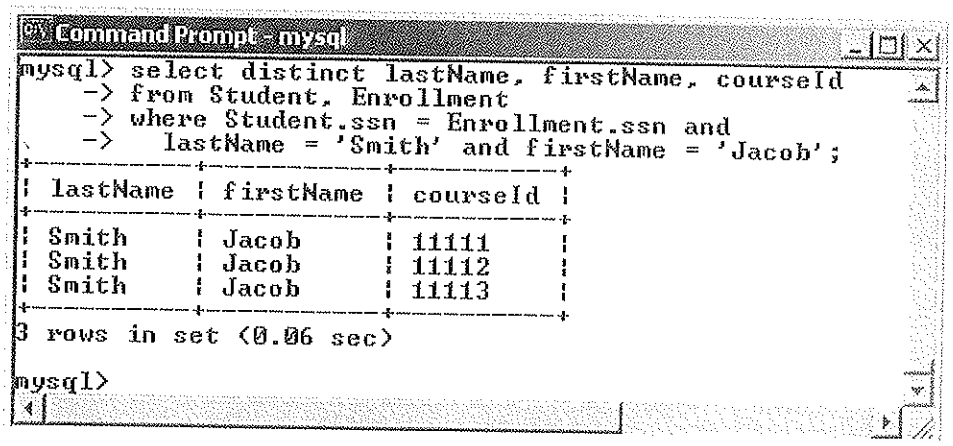


FIGURE 25.16 `Student` *and* `Enrollment` *are joined on* `ssn`.

The tables `Student` and `Enrollment` are listed in the `from` clause. The query examines every pair of rows, each made of one item from `Student` and another from `Enrollment`, and selects the pairs that satisfy the condition in the `where` clause. The rows in `Student` have the last name, Smith, and the first name, Jacob, and both rows from `Student` and `Enrollment` have the same `ssn` values. For each pair selected, `lastName` and `firstName` from `Student` and `courseId` from `Enrollment` are used to produce the result, as shown in Figure 25.17. `Student` and `Enrollment` have the same attribute `ssn`. To distinguish them in a query, use `Student.ssn` and `Enrollment.ssn`.



```
mysql> select distinct lastName, firstName, courseId
    -> from Student, Enrollment
    -> where Student.ssn = Enrollment.ssn and
    ->   lastName = 'Smith' and firstName = 'Jacob';
+----------+-----------+----------+
| lastName | firstName | courseId |
+----------+-----------+----------+
| Smith    | Jacob     | 11111    |
| Smith    | Jacob     | 11112    |
| Smith    | Jacob     | 11113    |
+----------+-----------+----------+
3 rows in set (0.06 sec)

mysql>
```

FIGURE 25.17 *Query 7 demonstrates queries involving multiple tables.*

25.4 JDBC

The Java API for developing Java database applications is called *JDBC*. JDBC is the trademarked name of a Java API that supports Java programs that access relational databases. JDBC is not an acronym, but it is often thought to stand for Java Database Connectivity.

JDBC provides Java programmers with a uniform interface for accessing and manipulating a wide range of relational databases. Using the JDBC API, applications written in the Java programming language can execute SQL statements, retrieve results, present data in a user-friendly interface, and propagate changes back to the database. The JDBC API can also be used to interact with multiple data sources in a distributed, heterogeneous environment.

The relationships between Java programs, JDBC API, JDBC drivers, and relational databases are shown in Figure 25.18. The JDBC API is a set of Java interfaces and classes used to write Java

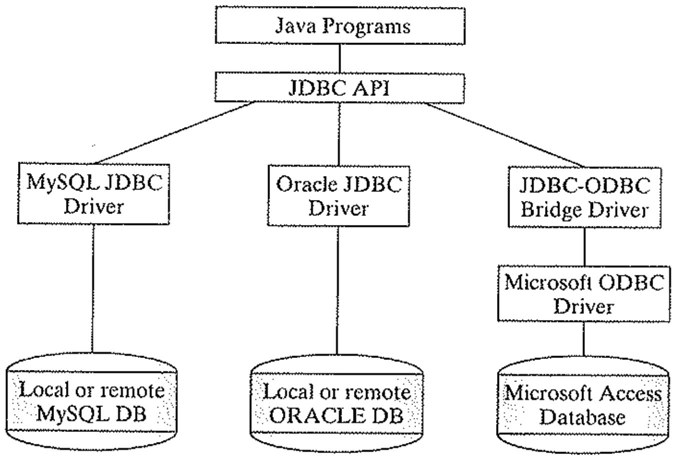


FIGURE 25.18 *Java programs access and manipulate databases through JDBC drivers.*

programs for accessing and manipulating relational databases. Since a JDBC driver serves as the interface to facilitate communications between JDBC and a proprietary database, JDBC drivers are database-specific. You need MySQL JDBC drivers to access the MySQL database, and Oracle JDBC drivers to access the Oracle database. Even with the same vendor, the drivers may be different for different versions of a database. For instance, the JDBC driver for Oracle 9 is different from the one for Oracle 8. A JDBC-ODBC bridge driver is included in JDK to support Java programs that access databases through ODBC drivers. An ODBC driver is preinstalled on Microsoft Windows 98, NT, 2000, and XP. You can use the JDBC-ODBC driver to access Microsoft Access database.

25.4.1 Developing Database Applications Using JDBC

The JDBC API is a Java application program interface to generic SQL databases that enables Java developers to develop DBMS-independent Java applications using a uniform interface.

The JDBC API consists of classes and interfaces for establishing connections with databases, sending SQL statements to databases, processing the results of the SQL statements, and obtaining database metadata. Four key interfaces are needed to develop any database application using Java: `Driver`, `Connection`, `Statement`, and `ResultSet`. These interfaces define a framework for generic SQL database access. The JDBC driver vendors provide implementation for them. The relationship of these interfaces is shown in Figure 25.19. A JDBC application loads an appropriate driver

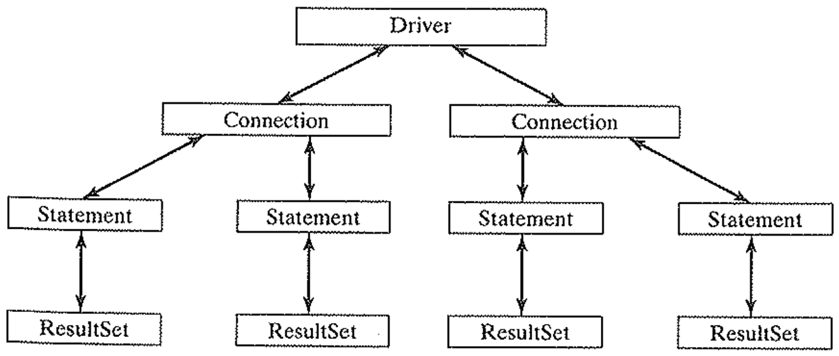


FIGURE 25.19 *JDBC classes enable Java programs to connect to the database, send SQL statements, and process results.*

using the `Driver` interface, connects to the database using the `Connection` interface, creates and executes SQL statements using the `Statement` interface, and processes the result using the `ResultSet` interface if the statements return results. Note that some statements, such as SQL data definition statements and SQL data modification statements, do not return results.

The JDBC interfaces and classes are the building blocks in the development of Java database programs. A typical Java program takes the steps outlined below to access the database.

1. **Loading drivers.** An appropriate driver must be loaded using the statement shown below before connecting to a database:

```
Class.forName("JDBCDriverClass");
```

A driver is a concrete class that implements the `java.sql.Driver` interface. The drivers for Access, MySQL, and Oracle are listed in Table 25.3.

TABLE 25.3 JDBC Drivers

Database	*Driver Class*	*Source*
Access	`sun.jdbc.odbc.JdbcOdbcDriver`	Already in JDK
MySQL	`com.mysql.jdbc.Driver`	Companion Website
Oracle	`oracle.jdbc.driver.OracleDriver`	Companion Website

The JDBC-ODBC driver for Access is bundled in JDK. The MySQL JDBC driver is contained in mysqljdbc.jar and Oracle JDBC driver is contained in classes12.jar. Both files are in the source code directory on the Companion Website. To use the MySQL and Oracle drivers, you have to add mysqljdbc.jar and classes12.jar in the classpath using the following DOS command in Windows:

```
classpath=%classpath%;c:\book\mysqljdbc.jar;c:\book\classes12.jar
```

If your program accesses several different databases, all their respective drivers must be loaded.

2. **Establishing connections.** To connect to a database, use the static method `getConnection(databaseURL)` in the `DriverManager` class, as follows:

```
Connection connection = DriverManager.getConnection(databaseURL);
```

where `databaseURL` is the unique identifier of the database on the Internet. Table 25.4 lists the URLs for the MySQL, Oracle, and Access databases.

TABLE 25.4 JDBC URLs

Database	*URL Pattern*
Access	`jdbc:odbc:dataSource`
MySQL	`jdbc:mysql://hostname/dbname`
Oracle	`jdbc:oracle:thin:@hostname:port#:oracleDBSID`

For an ODBC data source, the `databaseURL` is `jdbc:odbc:dataSource`. An ODBC data source can be created using the ODBC Data Source Administrator on Windows. See Supplement M, "Tutorial for Microsoft Access," on how to create an ODBC data source for an Access database. Suppose a data source named ExampleMDBDataSource has been created for an Access database. The following statement creates a `Connection` object:

connect Access DB

```
Connection connection = DriverManager.getConnection
  ("jdbc:odbc:ExampleMDBDataSource");
```

The databaseURL for a MySQL database specifies the host name and database name to locate a database. For example, the following statement creates a Connection object for the local MySQL database test:

connect MySQL DB

```
Connection connection = DriverManager.getConnection
  ("jdbc:mysql://localhost/test");
```

Recall that by default MySQL contains two databases named *mysql* and *test*. You can create a custom database using the MySQL SQL command create database *databasename*.

The databaseURL for an Oracle database specifies the *host name*, the *port#* where the database listens for incoming connection requests, and the *oracleDBSID* database name to locate a database. For example, the following statement creates a Connection object for the Oracle database on liang.armstrong.edu with *user name* scott and password tiger:

connect Oracle DB

```
Connection connection = DriverManager.getConnection
  ("jdbc:oracle:thin:@liang.armstrong.edu:1521:ora9i",
   "scott", "tiger");
```

3. **Creating statements.** If a Connection object can be envisioned as a cable linking your program to a database, an object of Statement or its subclass can be viewed as a cart that delivers SQL statements for execution by the database and brings the result back to the program. Once a Connection object is created, you can create statements for executing SQL statements as follows:

```
Statement statement = connection.createStatement();
```

4. **Executing statements.** An SQL DDL or update statement can be executed using executeUpdate(String sql), and an SQL query statement can be executed using executeQuery(String sql). The result of the query is returned in ResultSet. For example, the following code executes the SQL statement *create table Temp (col1 char(5), col2 char(5))*:

```
statement.executeUpdate
  ("create table Temp (col1 char(5), col2 char(5))");
```

The next code executes the SQL query *select firstName, mi, lastName from Student where lastName = 'Smith'*:

```
// Select the columns from the Student table
ResultSet resultSet = statement.executeQuery
  ("select firstName, mi, lastName from Student where lastName "
    + " = 'Smith'");
```

5. **Processing ResultSet.** The ResultSet maintains a table whose current row can be retrieved. The initial row position is null. You can use the next method to move to the next row and the various get methods to retrieve values from a current row. For example, the code given below displays all the results from the preceding SQL query:

```
// Iterate through the result and print the student names
while (resultSet.next())
  System.out.println(rset.getString(1) + " " + rset.getString(2)
    + ". " + rset.getString(3));
```

The getString(1), getString(2), and getString(3) methods retrieve the column values for firstName, mi, and lastName, respectively. Alternatively, you can use getString. ("firstName"), getString("mi"), and getString("lastName") to retrieve the same three column values. The first execution of the next() method sets the current row to the first row in the result set, and subsequent invocations of the next() method set the current row to the second row, third row, and so on, to the last row.

Listing 25.1 is a complete example that demonstrates connecting to a database, executing a simple query, and processing the query result with JDBC. The program connects to a local MySQL database and displays the students whose last name is Smith.

LISTING 25.1 SimpleJDBC.java

load driver

connect database

create statement

execute statement

get result

close connection

```
 1 import java.sql.*;
 2
 3 public class SimpleJdbc {
 4   public static void main(String[] args)
 5       throws SQLException, ClassNotFoundException {
 6     // Load the JDBC driver
 7     Class.forName("com.mysql.jdbc.Driver");
 8     System.out.println("Driver loaded");
 9
10     // Establish a connection
11     Connection connection = DriverManager.getConnection
12       ("jdbc:mysql://localhost/test");
13     System.out.println("Database connected");
14
15     // Create a statement
16     Statement statement = connection.createStatement();
17
18     // Execute a statement
19     ResultSet resultSet = statement.executeQuery
20       ("select firstName, mi, lastName from Student where lastName "
21         + " = 'Smith'");
22
23     // Iterate through the result and print the student names
24     while (resultSet.next())
25       System.out.println(resultSet.getString(1) + "\t" +
26         resultSet.getString(2) + "\t" + resultSet.getString(3));
27
28     // Close the connection
29     connection.close();
30   }
31 }
```

The statement in Line 7 loads a JDBC driver for MySQL, and the statement in Lines 11–12 connects to a local MySQL database. You may change them to connect to an Access or Oracle database. The last statement (Line 29) closes the connection and releases resource related to the connection.

NOTE
Do not use a semicolon (;) to end the Oracle SQL command in a Java program. The semicolon does not work with the Oracle JDBC drivers. It does work, however, with the other drivers used in the book.

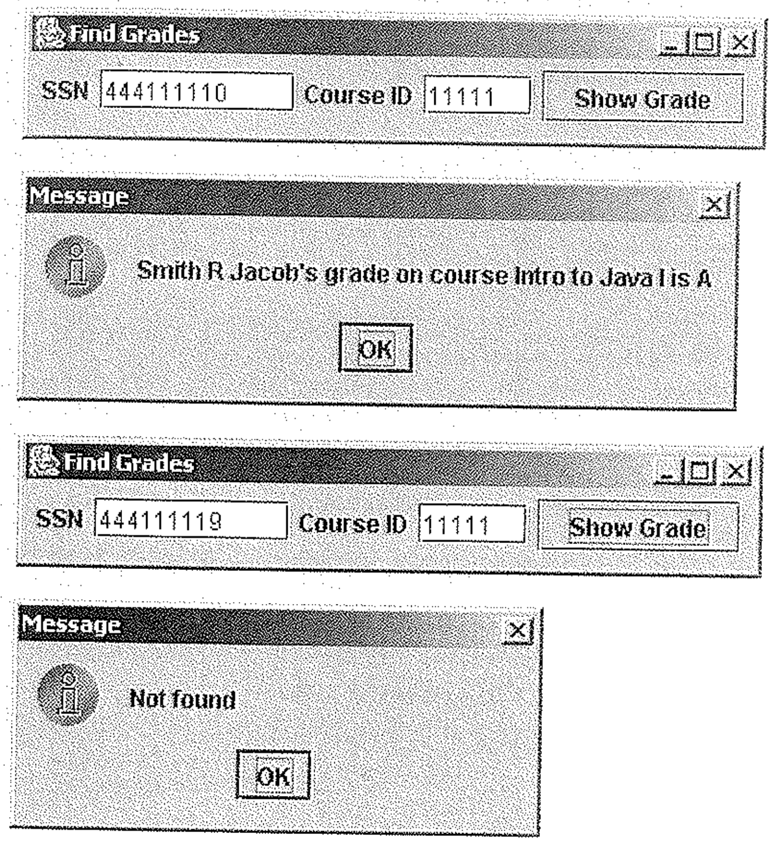
NOTE
The `Connection` interface handles transactions and specifies how they are processed. By default, a new connection is in auto-commit mode, and all its SQL statements are executed and committed as individual transactions. The commit occurs when the statement completes or the next execute occurs, whichever comes first. In the case of statements returning a result set, the statement completes when the last row of the result set has been retrieved or the result set has been closed. If a single statement returns multiple results, the commit occurs when all the results have been retrieved. You can use the `setAutoCommit(false)` method to disable auto-commit, so that all SQL statements are grouped into one transaction that is terminated by a call to either the `commit()` or the `rollback()` method. The `rollback()` method undoes all changes made by the transaction.

EXAMPLE 25.1 ACCESSING A DATABASE FROM A JAVA APPLET

Problem

This example demonstrates connecting to a database from a Java applet. The applet lets the user enter the SSN and the course ID to find a student's grade, as shown in Figure 25.20.

EXAMPLE 25.1 (CONTINUED)


Find Grades

SSN 444111110 Course ID 11111 Show Grade

Message

Smith R Jacob's grade on course Intro to Java I is A

OK

Find Grades

SSN 444111119 Course ID 11111 Show Grade

Message

Not found

OK

FIGURE 25.20 *A Java applet can access the database on the server.*

Solution

Using the JDBC-ODBC bridge driver, your program cannot run as an applet from a Web browser because the ODBC driver contains non-Java native code. The JDBC drivers for MySQL and Oracle are written in Java and can run from the JVM in a Web browser. The code in Listing 25.2 uses the MySQL database on the host liang.armstrong.edu:

LISTING 25.2 FindGrade.java

```java
import javax.swing.*;
import java.sql.*;
import java.awt.*;
import java.awt.event.*;

public class FindGrade extends JApplet {
  private JTextField jtfSSN = new JTextField(9);
  private JTextField jtfCourseId = new JTextField(5);
  private JButton jbtShowGrade = new JButton("Show Grade");

  // Statement for executing queries
  private Statement stmt;

  /** Initialize the applet */
  public void init() {
    // Initialize database connection and create a Statement object
    initializeDB();

```

EXAMPLE 25.1 (CONTINUED)

button listener

load MySQL driver
Oracle driver commented

connect database

connect to Oracle commented

create statement

execute statement

show result

```
    jbtShowGrade.addActionListener(
      new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        jbtShowGrade_actionPerformed(e);
      }
    });

    JPanel jPanel1 = new JPanel();
    jPanel1.add(new JLabel("SSN"));
    jPanel1.add(jtfSSN);
    jPanel1.add(new JLabel("Course ID"));
    jPanel1.add(jtfCourseId);
    jPanel1.add(jbtShowGrade);

    this.getContentPane().add(jPanel1, BorderLayout.NORTH);
  }

  private void initializeDB() {
    try {
      // Load the JDBC driver
      Class.forName("com.mysql.jdbc.Driver");
//        Class.forName("oracle.jdbc.driver.OracleDriver");
      System.out.println("Driver loaded");

      // Establish a connection
      Connection connection = DriverManager.getConnection
        ("jdbc:mysql://liang.armstrong.edu/test");
//    ("jdbc:oracle:thin:@liang.armstrong.edu:1521:ora9i",
//      "scott", "tiger");
      System.out.println("Database connected");

      // Create a statement
      stmt = connection.createStatement();
    }
    catch (Exception ex) {
      ex.printStackTrace();
    }
  }

  private void jbtShowGrade_actionPerformed(ActionEvent e) {
    String ssn = jtfSSN.getText();
    String courseId = jtfCourseId.getText();
    try {
      String queryString = "select firstName, mi, " +
        "lastName, title, grade from Student, Enrollment, Course " +
        "where Student.ssn = '" + ssn + "' and Enrollment.courseId "
        + "= '" + courseId +
        "' and Enrollment.courseId = Course.courseId " +
        " and Enrollment.ssn = Student.ssn";

      ResultSet rset = stmt.executeQuery(queryString);

      if (rset.next()) {
        String lastName = rset.getString(1);
        String mi = rset.getString(2);
        String firstName = rset.getString(3);
        String title = rset.getString(4);
        String grade = rset.getString(5);

        // Display result in a dialog box
        JOptionPane.showMessageDialog(null, firstName + " " + mi +
          " " + lastName + "'s grade on course " + title + " is " +
          grade);
      } else {
        // Display result in a dialog box
        JOptionPane.showMessageDialog(null, "Not found");
      }
    }
```