Chương IX

Kết nối các cơ sở dữ liệu với JDBC và lập trình trên mạng

Chương cuối giới thiệu:

- ✓ Tổng quan về kết nối CSDL với ODBC và JDBC;
- ✓ Xây dựng ứng dụng dựa trên JDBC;
- ✓ Phương pháp lập trình trên mạng và đa luồng;
- ✓ Các lớp *Sockets* của *Java* và các ứng dụng.

9.1 Giới thiệu tổng quan về ODBC và JDBC

Xu thế phát triển hiện nay của công nghệ phần mềm là xây dựng những ứng dụng tích hợp, dựa trên những cơ sở dữ liệu (CSDL) khác nhau được thiết lập ở nhiều nơi khác nhau trên mạng. ODBC (*Open DataBase Connectivity*) của Microsoft và JDBC (*Java DataBase Connectivity*) của Sun là những công cụ phổ dụng hỗ trợ để phát triển những hệ thống như thế.

ODBC

ODBC là một trong những giao diện CSDL được sự dụng phổ biến nhất hiện nay trên các máy PC. ODBC cung cấp các chức năng của một ngôn ngữ lập trình để tương tác với các CSDL như: bổ sung, cập nhật, xoá những dữ liệu không cần thiết và nhận được những thông tin chi tiết về các bảng, các chỉ số hay khung nhìn của các phần dữ liệu trong một hệ CSDL.

Một ứng dụng ODBC thường được tổ chức theo năm tầng: tầng ứng dụng, giao diện ODBC, tầng quản lý các bộ điều khiển, bộ điều khiển và tầng nguồn dữ liệu.

- Tầng ứng dụng cung cấp GUI và các chức năng xử lý nghiệp vụ được viết bằng những ngôn ngữ lập trình như Java, Visual Basic, hay C++. Chương trình ứng dụng của bạn có thể sử dụng những chức năng ODBC trong giao diện ODBC để tương tác với các CSDL.
- Tầng quản lý các bộ điều khiển (*Driver Manager*) là một bộ phận của Microsoft ODBC. Nó làm nhiệm vụ quản lý các bộ điều khiển có mặt trong hệ thống để nạp hay chọn được những bộ điều khiển thích hợp và cung cấp cho những bộ điều khiển đó những thông tin cần thiết. Bởi vì một chương trình ứng dụng có thể kết nối với nhiều hơn một CSDL, nên *Driver Manager* phải đảm bảo rằng chính hệ QTCSDL (Quản Trị CSDL) của hệ CSDL tương ứng sẽ nhận được tất cả các lời gọi hàm hướng tới nó để xử lý và các nguồn dữ liệu được chuyển đúng tới chương trình ứng dụng theo yêu cầu.
- Tầng bộ điều khiển (*Driver*) là thành phần cho biết cụ thể về các CSDL cần kết nối. Đó là những bộ điều khiển được gán cho những hệ CSDL tương ứng như:

Access Driver, SQL Server Driver, và *Oracle Driver,* v.v. Giao diện ODBC có tập các hàm như các lệnh SQL, quản lý sự kết nối, các thông tin về CSDL, v.v. Đối với những CSDL ở trên mạng cục bộ hay trên Internet, tầng Driver còn đảm nhiệm cả chức năng xử lý sự trao đổi thông tin trên mạng.

- Trong ngữ cảnh của ODBC, nguồn dữ liệu (*Data Source*) có thể là hệ CSDL nhỏ, đơn giản của MS Access hay những kho dữ liệu (Data Warehouse) lớn có hàng chục *gigabyte.*

JDBC

JDBC là giao diện để kết nối với CSDL, gồm một tập các lớp đối tượng hỗ trợ để xử lý CSDL quan hệ và để tương tác với các nguồn dữ liệu khác nhau. ODBC là giao diện của ngôn ngữ *C,* và rất khó chuyển đổi tương ứng sang được Java, còn JDBC là giao diện của Java và hoàn toàn thống nhất với những thành phần khác của hệ thống Java.

JDBC API (*Application Programming Interface*) được JavaSoft phát triển và là một phần của tất cả các cài đặt ứng dụng của JVM. JDBC API cung cấp các chức năng cơ bản để truy nhập (thường ở mức trung gian) tới hầu hết các hệ CSDL thông qua SQL (*Structural Query Language*). Chúng ta có thể truy nhập và tìm kiếm mọi thông tin, kể cả dữ liệu về âm thanh, hình ảnh động từ nhiều nguồn khác nhau trên mạng bằng cách sử dụng JDBC API để nạp chúng vào các đối tượng của Java và sau đó xử lý các thông tin đó. Bạn có thể nạp JDBC từ địa chỉ: http://splash.javaSoft.com/jdbc.

Kiến trúc của JDBC

Mỗi CSDL đều có một API riêng và muốn tương tác với CSDL đó thì phải biết được kiến trúc của chúng. Muốn sử dụng được các thông tin, dữ liệu từ nhiều CSDL trên mạng, chúng ta phải sử dụng những giao diện lập trình ứng dụng (API) để kết nối và truy vấn vào các CSDL.

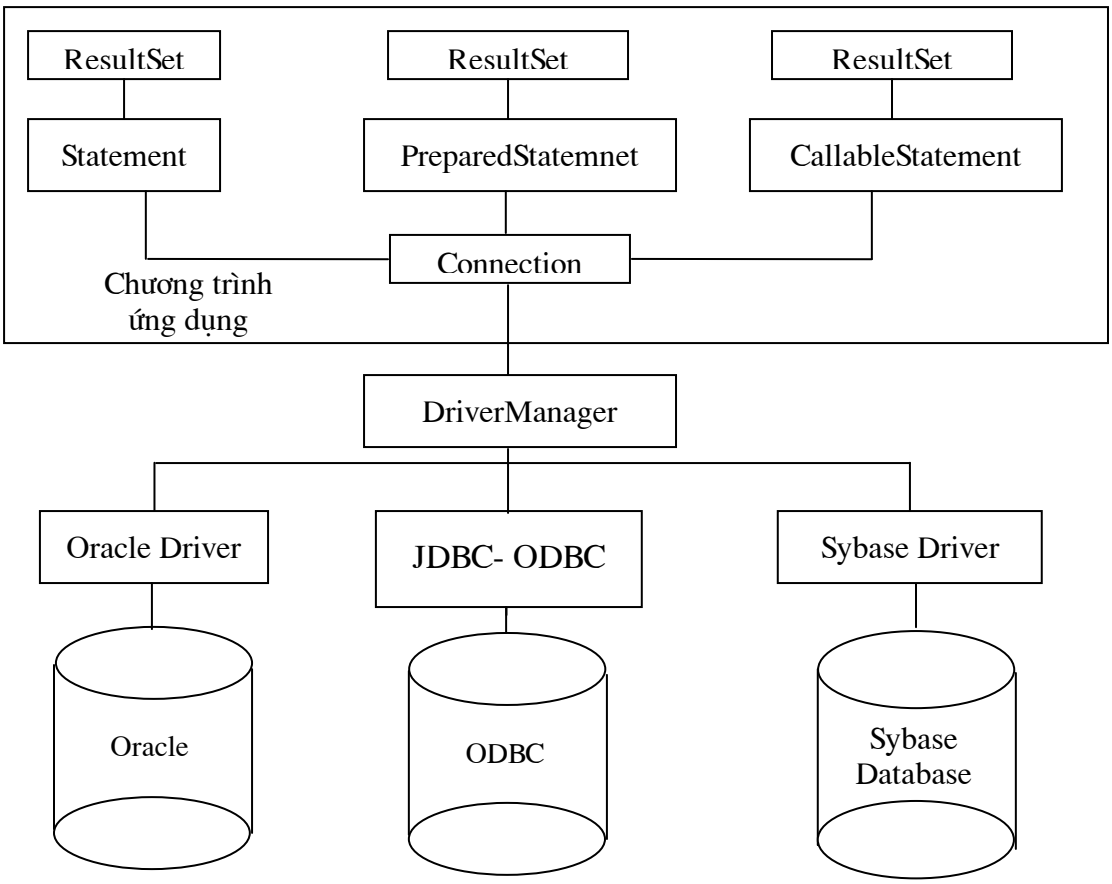
Tương tự như ODBC, JDBC của Sun nhằm tạo ra những giao diện trung gian giữa CSDL và Java. Với JDBC API, chúng ta có thể thực hiện được những chức năng cơ bản đối với CSDL:

- Thực hiện các truy vấn vào một CSDL,
- Xử lý kết quả từ truy vấn,
- Xác định các thông tin về cấu hình hệ thống.

Các lớp trong JDBC API được sử dụng để thực hiện những chức năng trên được tổ chức theo kiến trúc như trong hình H9-1.

JDBC định nghĩa các đối tượng API và các hàm để tương tác với những CSDL đã được chỉ định. Chương trình ứng dụng Java kết nối các CSDL có thể được tổ chức như sau:

- Trước tiên là tạo ra đối tượng kết nối vào một CSDL,
- Tạo ra đối tượng để xử lý các câu lệnh,
- Truyền tham số cho các lệnh SQL trong các hệ quản trị CSDL (DBMS) và các đối tượng xử lý các câu lệnh đó,
- Tìm kiếm các thông tin là các kết quả truy vấn.



Hình H9-1 Kiến trúc của JDBC

Các lớp JDBC được tổ chức trong gói *java.sql.* Các lớp JDBC và các chương trình ứng dụng *Applet hay ứng dụng độc lập* của Java có thể:

- Để ở các máy khách,
- Hoặc cũng có thể nạp xuống từ mạng.

Nhưng tốt nhất là chúng ta để các lớp JDBC ở máy khách, còn các DBMS (hệ quản trị CSDL) và các dữ liệu nguồn có thể đặt ở máy phục vụ ở đâu đó trên mạng.

Một chương trình muốn kết nối các CSDL thông qua JDBC thì phải sử dụng bộ điều khiển dữ liệu gốc theo giao diện tương ứng với từng hệ DBMS (hình H9-1). Đối với cầu nối JDBC - ODBC trong Windows thì nên sử dụng JDBCODBC.DLL. Chương trình ứng dụng của Java có thể trao đổi với máy phục vụ, hay các ứng dụng khác thông qua các giao thức *RPC* hoặc *HTTP* [4].

JDBC - ODBC Bridge

JavaSoft cung cấp bộ điều khiển JDBC để truy nhập vào các nguồn dữ liệu dựa vào ODBC, tức là tạo ra cầu nối giữa chúng.

Cầu nối JDBC - ODBC được tổ chức thành lớp JdbcOdbc và thành thư viện ngoại để truy nhập theo bộ điều khiển ODBC (thư viện động JDBCODBC.DLL).

Với cầu nối JDBC - ODBC thì JDBC có ưu điểm nổi trội là có khả năng truy nhập tới hầu như tất cả các CSDL phổ biến, tương tự như các bộ điều khiển của ODBC.

9.2 Chương trình ứng dụng JDBC

Để xử lý được dữ liệu từ một CSDL, chương trình Java phải thực hiện lần lượt theo các bước sau:

1. Trước tiên là gọi hàm *getConnection()* để nhận được đối tượng của lớp *Connection;*
2. Tạo ra một đối tượng của lớp *Statement;*
3. Chuẩn bị một đối tượng để xử lý lệnh của SQL và truy vấn vào dữ liệu theo yêu cầu; Câu lệnh SQL có thể thực hiện trực tiếp thông qua đối tượng của *Statement* hoặc có thể được biên dịch dịch thông qua đối tượng của *PreparedStatement* hay gọi một thủ tục để lưu lại thông qua *CallableStatement.*
4. Khi hàm *executeQuery()* được thực hiện, thì kết quả được cho lại là đối tượng của lớp *ResualtSet* bao gồm các dòng dữ liệu và có thể sử dụng hàm *next()* để xác định các dữ liệu theo yêu cầu.

Sau đây chúng ta xét một vài chương trình tương đối đơn giản để truy vấn, cập nhật vào một CSDL.

Trước tiên, chúng ta hãy tạo ra một CSDL theo mô hình quan hệ, trong đó, mỗi mục dữ liệu được xem như một dòng tin về đối tượng được lưu trữ. Mỗi dòng sẽ chứa một số cột, được gọi là trường dữ liệu. Đặc điểm chính của mô hình quan hệ là dữ liệu được lưu trữ phải là thuần nhất, nghĩa là số trường của tất cả dòng dữ liệu phải bằng nhau. Một số các dòng và các trường dữ liệu được tổ chức thành bảng quan hệ (bảng dữ liệu) mô tả về một thực thể hay một mối quan hệ nào đó của hệ thống ứng dụng. Nhiều bảng dữ liệu như thế sẽ tạo thành CSDL.

Khi các CSDL quan hệ trở nên phổ biến, các chuyên gia CSDL mong muốn có một ngôn ngữ CSDL vạn năng để thao tác trên các dữ liệu và SQL chính là ngôn ngữ đáp ứng được các yêu đó. SQL có những cấu trúc lệnh để tạo lập, cập nhật, xoá bỏ các bảng, dòng hay các trường dữ liệu trong hệ thống. Ngoài ra SQL còn hỗ trợ để quản lý quyền truy nhập tới các phần tử dữ liệu, quản lý người sử dụng, hay thực hiện sao chép dữ liệu, v.v. SQL có thể sử dụng kết hợp với những ngôn ngữ lập trình như Java, C++, v.v., để xử lý dữ liệu và tương tác với các hệ QTCSDL.

Giả sử chúng ta có một CSDL bao gồm những bảng dữ liệu về sinh viên và các môn học của sinh viên. Ví dụ: Bảng *Student* gồm ba trường *StudentNo, FName,* và *LName,* trong đó *StudentNo* là *khoá.*

StudentNo	FName	LName
1	Vũ Văn	An
2	Trần Anh	Tuấn
3	Lê Hoa	Lư
4	Nguyễn Công	Tuyền

Bảng B9.1 Bảng dữ liệu *Student*

Bảng *Subject* có hai trường *SubjectNo* và *SubjectName, SubjectNo* là khoá.

SubjectNo	SubjectName
M1	Lập trình Java
M2	Cơ sở dữ liệu
M3	Toán rời rạc
M4	Phân tích, thiết kế hệ thống

Bảng B9.2 Bảng dữ liệu *Subject*

Tất nhiên các bảng dữ liệu trong một hệ CSDL phải được liên kết với nhau thông qua các *khoá ngoại* . Để bảng *Student* liên kết được với bảng *Subject,* chúng ta phải tạo thêm bảng mới, ví dụ: bảng *Student_Subject* có ba trường: *StudentNo, SubjectNo* và *Marks* (điểm thi môn học của sinh viên) xác định mối quan hệ giữa hai bảng dữ liệu trên và lưu điểm thi từng môn của sinh viên.

StudentNo	SubjectNo	Marks
1	M2	6
2	M3	7
3	M1	5
4	M4	8

Bảng B9.3 Bảng dữ liệu *Student_Subject*

Chúng ta có thể sử dụng MS Access để tạo ra các bảng trên của CSDL *SVDB.*

Như vậy, trước khi viết chương trình JDBC, chúng ta cần phải biết được cấu trúc của nguồn dữ liệu. Chính hàm *getConnection()* sẽ cho lại tên miền của nguồn dữ liệu (DSN), ID của người sử dụng và mật khẩu (password) để truy nhập vào nguồn dữ liệu cần kết nối.

Ví dụ 9.1 Truy vấn vào CSDL.

Trong ví dụ này chúng ta muốn liệt kê tất cả họ (FName) và tên sinh viên (LName) từ bảng *Student* trong CSDL *StudentDB* bằng lệnh SELECT của SQL.

// Một ví dụ đơn giản sử dụng JDBC để đọc dữ liệu từ CSDL: JDBCSample.java.

```
import java.sql.*;
public class JDBCSample{
     public static void main(String[] args){
          try{// Nạp Driver JdbcOdbcDriver
              Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
             }catch (ClassNotFoundException e){
             System.out.println("Khong nap duoc lop Driver");
               return;
               }
```

```
        try{// Mọi truy nhập vào CSDL đều thực hiện trong khối try - catch
            // Xác định tên của CSDL, tên người sử dụng và mật khẩu
            Connection con =
                DriverManager.getConnection("jdbc:odbc:StudentDB","","");
        // Thiết lập và thực hiện các lệnh của SQL
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT FName,
                                LName FROM SINHVIEN");
        // Hiển thị kết quả sau khi đã sử dụng các lệnh của SQL để truy vấn
        while(rs.next()){
              System.out.print(rs.getString("FName")+ " ");
              System.out.println(rs.getString("LName"));
              }
        // Đóng lại các nguồn dữ liệu đã được mở
        rs.close();
        stmt.close();
        con.close();
      }catch(SQLException se){
              System.out.println("Loi o SQL: " + se.getMessage());
              se.printStackTrace(System.out);
        }
    }
}
```

Đây là chương trình Java đơn giản sử dụng JDBC API. Chương trình này minh hoạ các bước cơ sở, cần thiết để truy nhập vào các bảng dữ liệu và liệt kê một số trường dữ liệu từ các bảng dữ liệu đó.

Lớp DriverManager

Trước khi sử dụng một bộ điều khiển (*Driver*) thì nó phải được đăng ký với *DriverManager.* Điều này thực hiện được bằng cách sử dụng hàm *forName()* của lớp *Class:*

```
try{
     Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
     Class.forName("com.oracle.jdbc.OracleDriver");
 }catch (ClassNotFoundException e){
     // Xử lý ngoại lệ khi không nạp được
     }
```

Bộ điều khiển JDBC sử dụng lớp JDBC URL (Uniform Resource Locator) để xác định và kết nối với một CSDL. Dạng tổng quát của nó là:

jdbc:driver:databaseName

Lớp Connection

Đối tượng của lớp *java.sql.Connection* có thể sử dụng hàm *DriverManager.getConnection()* để thiết lập một kết nối:

```
Connection con =
```

```
DriverManager.getConnection("url","userName","password");
```

Trong đó, `url` là đối tượng của JDBC URL xác định tên của CSDL cần truy nhập, `userName` tên của người sử dụng và mật khẩu `password`. Đối với những CSDL khi mở không yêu cầu khai báo tên người sử dụng và mật khẩu thì các tham biến đó có thể là rỗng.

```
Connection con = DriverManager.getConnection("url","","");
```

Sau khi đã thiết lập được đối tượng *con* để kết nối, chúng ta có thể sử dụng các câu lệnh của SQL để truy vấn vào CSDL. Trong JDBC có ba lớp xử lý các câu lệnh đó.

Statement

Đại diện cho các lệnh của SQL.

PreparedStatement

Đại diện cho các lệnh được tiền xử lý của SQL nhằm tăng hiệu quả xử lý.

CallableStatement

Cho phép các chương trình sử dụng JDBC để truy nhập vào bên trong một CSDL.

Lớp Statement

Trong ví dụ trên, chúng ta sử dụng đối tượng của lớp *Statement* để tạo ra khả năng xử lý các câu lệnh của SQL.

```
Statement stmt = con.createStatement();
```

Sau đó có thể sử dụng câu lệnh *SELECT* của SQL để truy vấn và thao tác trên những dữ liệu nhận được.

Lớp ResultSet

```
ResultSet rs = stmt.executeQuery("SELECT * FROM STUDENT");
```

Chọn ra tất cả các dòng của bảng *STUDENT* và gán cho đối tượng *rs* của *ResultSet.* Chúng ta có thể xem *ResultSet* như là đối tượng chứa những kết quả truy vấn vào CSDL theo câu lệnh *SELECT* của SQL.

Để đọc giá trị của các cột (trường dữ liệu) có thể sử dụng hàm dạng *getXXX(),* trong đó *XXX* được thay thế tương ứng bằng tên các lớp, như *getString(), getByte(), getInt(), v.v.* tuỳ thuộc vào các kiểu của các trường dữ liệu. Hàm *next()* cho phép đọc dòng dữ liệu tiếp theo.

```
while(rs.next()){
     System.out.println(rs.getString("TEN_SV"));
     }
```

Lưu ý: Giá trị trả lại của hàm *getXXX(args)* là dữ liệu của trường có tên là *args* của các dòng dữ liệu đã được chọn ra. Ngoài ra cũng cần phân biệt các kiểu của

Java với các kiểu dữ liệu của SQL. Bảng B9.4 mô tả các kiểu dữ liệu tương ứng của Java, SQL và các hàm *getXXX().*

Kiểu của SQL	Kiểu của Java	Hàm getXXX()
CHAR	String	getString()
VARCHAR	String	getString()
LONGVARCHAR	String	getString()
NUMBERIC	java.math.BigDecimal	getBigDecimal()
DECIMAL	java.math.BigDecimal	getBigDecimal()
BIT	Boolean (boolean)	getBoolean()
TINYINT	Integer (byte)	getByte()
SMALLINT	Integer (short)	getShort()
INTEGER	Integer (int)	getInt()
BIGINT	Long (long)	getLong()
REAL	Float (float)	getFloat()
FLOAT	Double (double)	getDouble()
DOUBLE	Double (double)	getDouble()
BINARY	byte[]	getBytes()
VARBINARY	byte[]	getBytes()
LONGVARBINARY	byte[]	getBytes()
DATE	java.sql.Date	getDate()
TIME	java.sql.Time	getTime()
TIMESTAMP	java.sql.Timestamp	getTimestamp()

Bảng B9.4 Các kiểu dữ liệu tương ứng của Java, SQL và các hàm *getXXX()*

Lớp *Statement* còn có hàm *executeUpdate()* để cập nhật dữ liệu và cho lại kết quả là số nguyên chỉ ra số dòng của CSDL đã được kết nối, hàm *execute()* được sử dụng để thực hiện các lệnh của SQL.

Ví dụ 9.2 Đọc, hiển thị và cập nhật dữ liệu trong CSDL: StudentDB.

```
import java.sql.*;
public class UpdateStudentName{
  int i,NoOfColumns;
  String StNo, StFName, StLName;
  String uid, pw;
  public void ListStudent() throws SQLException {
```

// Khởi tạo và nạp JDBC-ODBC driver.

```
 try{
   Class.forName("jdbc.odbc.JdbcOdbcDriver");
   }catch(ClassNotFoundException e){
      System.out.println("Unable to load Driver Class");
      return;
   }
```

// Thiết lập đối tượng kết nối

```
 Connection ExlCon=DriverManager.getConnection("jdbc:odbc:StudentDB",
                                   uid="admin",pw="sa");
```

// Tạo lập đối tượng của Statment

```
 Statement ExlStmt=ExlCon.createStatement();
```

// Thực hiện lệnh truy vấn SQL thông qua lệnh *SELECT*

```
 ResultSet Exlrs=ExlStmt.executeQuery(
        "SELECT StudentNo, FName, LName FROM Student");
```

// Thực hiện đọc từng dòng từ kết quả truy vấn và hiển thị lên màn hình

```
 System.out.println("Student Number  First Name  LastName");
 while (Exlrs.next()){
```

// Đọc từng trường dữ liệu và gán cho các biến của Java

```
 StNo= Exlrs.getString(1);            // Lấy kết quả của trường số 1
 StFName=Exlrs.getString(2);    // Lấy kết quả của trường số 2
 StLName=Exlrs.getString(3);    // Lấy kết quả của trường số 3
 System.out.println(StNo + " " + StFName + " " + StLName);
   }
}
```

// Thay họ (*StFName*) và tên (*StLName*) sinh viên có số hiệu là *STNo:* cập nhật lại

```
public void UpdateName (String StFName,
        String StLName,String StNo) throws SQLException
 {
   int RetValue;
```

// Khởi tạo và nạp JDBC-ODBC driver.

```
   try{
   Class.forName("jdbc.odbc.JdbcOdbcDriver");
   }catch(ClassNotFoundException e){
        System.out.println("Unable to load Driver Class");
        return;
        }
```

// Thiết lập đối tượng kết nối

```
   Connection ExlCon=DriverManager.getConnection("jdbc:odbc:StudentDB",
                              uid="admin",pw="sa");
```

// Tạo lập đối tượng của Statment

```
   Statement ExlStmt=ExlCon.createStatement();
```

// Tạo ra một xâu gồm các tham số cho lệnh executeUpdate()

```
   String SQLBuffer = "UPDATE Studends SET FirstName = " + StFName +
          ",LastName = " + StLName + " WHERE StudentsNo = "+StNo;
```

```
      // Thực hiện cập nhật dữ liệu trong SQL thông qua executeUpdate()
      RetValue= ExlStmt.executeUpdate(SQLBuffer);
      System.out.println("Updated " +RetValue+ "rows in the Database.");
      }

  public static void main(String[] args){
    UpdateStudentName sv = new UpdateStudentName();
    try{
      // Liệt kê tất cả họ, tên sinh viên ở bảng Student như đã được xây dựng ở trên
      sv.ListStudent();
      // Cập nhật: Thay sinh viên có mã số 2 có họ Vu Thuy và tên là Lan
      sv.UpdateName ("Vu Thuy ","Lan","2");
    }catch(SQLException se){
        System.out.println("SQL Error" + se);
          return;
        }
  }
}
```

Lớp DatabaseMetaData

Muốn xử lý tốt các dữ liệu của một CSDL thì chúng ta phải biết được những thông tin chung về cấu trúc của CSDL đó như: hệ QTCSDL, tên của các bảng dữ liệu, tên gọi của các trường dữ liệu, v.v .

Để biết được những thông tin chung về cấu trúc của một hệ CSDL, chúng ta có thể sử dụng giao diện *java.sql.DatabaseMetaData* thông qua hàm *getMetaData().*

```
DatabaseMetaData dbmeta = con.getMetaData();
```

trong đó, *con* là đối tượng kết nối đã được tạo ra bởi lớp *Connection.*

Lớp `DatabaseMetaData` cung cấp một số hàm được nạp chồng để xác định được những thông tin về cấu hình của một CSDL. Một số hàm cho lại đối tượng của String (*getURL()*), một số trả lại giá trị logic (*nullsAreSortedHigh()*) hay trả lại giá trị nguyên như hàm *getMaxConnection()*). Những hàm khác cho lại kết quả là các đối tượng của *ResultSet* như: *getColumns(), getTableType(), getPrivileges(), v.v.*

Ví dụ 9.3 Chương trình xác định những thông tin chung về cấu trúc của CSDL.

```
import java.sql.*;
import java.util.StringTokenizer;
public class DBViewer {
     final static String jdbcURL = "jdbc:odbc:StudentDB";
     final static String jdbcDriver = "sun.jdbc:odbc:JdbcOdbcDriver";
     public static void main(String[] args) {
       System.out.println("---Database Viewer ---");
       try {
```

```
// Đăng ký bộ điều khiển JdbcOdbcDriver
Class.forName(jdbcDriver);
// Kết nối với CSDL StudentDB
Connection con =
  DriverManager.getConnection(jdbcURL, "", "");
// Tạo ra dbmd để nhận các thông tin về cấu trúc của CSDL đã kết nối
DatabaseMetaData dbmd = con.getMetaData();
// In ra những thông tin về cấu trúc của CSDL đã kết nối
System.out.println("Drive Name: " + dbmd.getDriverName());
System.out.println("Database Product: " +
                dbmd.getDatabaseProductName());
System.out.println("SQL Keywords Supported:" );
// Tạo ra đối tượng st để nhận các từ khoá của SQL
StringTokenizer st =
     new StringTokenizer(dbmd.getSQLKeywords(),",");
while(st.hasMoreTokens())
     // Hiển thị các từ khoá của SQL
     System.out.println(""+st.nextToken());
     // Đối tượng allTables chứa tất cả các bảng trong CSDL
     String[]tableTypes = {"TABLE"};
     ResultSet allTables =
            dbmd.getTables(null,null,null,tableTypes);
     while (allTables.next())  {
          // Đọc và in ra tên, kiểu của từng bảng
          String table_name =
             allTables.getString("TABLE_NAME");
          System.out.println("Table Name:" + table_name);
          System.out.println("Table Type:" +
             allTables.getString("TABLE_TYPE"));
          System.out.println("Indexs: ");
          // Xác định tên của chỉ số (Index) và tên các trường dữ liệu
          ResultSet indexList = dbmd.getIndexInfo(null,null,
               table_name,false,false);
          while (indexList.next()) {
            // In ra tên của chỉ số (Index) và ten các trường dữ liệu
            System.out.println("Index Name:" +
              indexList.getString("INDEX_NAME"));
            System.out.println("Column Name:" +
               indexList.getString("COLUMN_NAME"));
          }
          indexList.close();
     }
     allTables.close();
     con.close();
   }catch (ClassNotFoundException ce) {
```

```
            System.out.println("Unable to load database driver class");
          }catch (SQLException se) {
            System.out.println("SQL Exception: " + se.getMessage());
          }
      } }
```

Giả sử *StudentDB* là CSDL được tạo ra bởi Microsoft Access thì chương trình trên thực hiện sẽ cho kết quả dạng:

```
--- Dabase Viewer ---
Driver Name: JDBC-ODBC Bridge
Database Product: Access
SQL Keywords Supported:
ALPHANUMERIC
BYTE
...
Table Name: Student
Table Type:  TABLE
Indexs:
  Index Name: PrimaryKey
  Column Name: StudentNo
  ...
```

Lớp ResultSetMetaData

Giao diện *ResultSetMetaData* cung cấp các thông tin về cấu trúc cụ thể của *ResultSet*, bao gồm cả số cột, tên và giá trị của chúng. Ví dụ sau là một chương trình hiển thị các kiểu và giá trị của từng trường của một bảng dữ liệu.

Ví dụ 9.3 Chương trình hiển thị một bảng dữ liệu.

```
import java.sql.*;
import java.util.StringTokenizer;
public class TableViewer {
     final static String jdbcURL = "jdbc:odbc:StudentDB";
     final static String jdbcDriver =
                          "sun.jdbc:odbc:JdbcOdbcDriver";
     final static String table = "STUDENT";
     public static void main(java.lang.String[]args) {
       System.out.println("---Table Viewer ---");
       try {
          Class.forName(jdbcDriver);
          Connection con =
               DriverManager.getConnection(jdbcURL, "", "");
          Statement stmt = con.createStatement();
          // Đọc ra cả bảng Student và đưa vào đối tượng rs
          ResultSet rs = stmt.executeQuery("SELECT * FROM " + table);
          // Đọc ra các thông tin về rs
          ResultSetMetaData rsmd = rs.getMetaData();
          // Xác định số cột của rsmd
          int colCount = rsmd.getColumnCount();
```

```
            for(int col = 1; col <= colCount; col++)
            {
                  // In ra tên và kiểu của từng trường dữ liệu trong rsmd
                  System.out.print(rsmd.getColumnLabel(col));
                  System.out.print(" (" + rsmd.getColumnTypeName(col) + ")");
                  if(col < colCount)
                     System.out.print(", ");
                  }
            System.out.println();

            while(rs.next()){
                  // In ra dòng dữ liệu trong rsmd
                  for(int col = 1; col <= colCount; col++)
                        {
                        System.out.print(rs.getString(col));
                        if(col < colCount)
                           System.out.print("  ");
                  }
                  System.out.println();
            }
            rs.close();
            stmt.close();
            con.close();
            }
            catch (ClassNotFoundException e) {
                  System.out.println("Unable to load database driver class");
               }
               catch (SQLException se) {
                     System.out.println("SQL Exception: " + se.getMessage());
               }
         }
   }
```

Dịch và thực hiện chương trình trên chúng ta có được kết quả dạng:

```
- - - Table Viewer - - -
StudentNo (SHORT), FName (VARCHAR), LName (VARCHAR)
1  Vũ Văn An
2  Trần Anh Tuấn,
. . .
```

Những vấn đề sâu hơn về kết nối CSDL và ứng dụng, bạn có thể tham khảo ở tài liệu số [4].

9.3 Lập trình trên mạng

Để trao đổi được giữa các ứng dụng với nhau trên mạng, Java sử dụng *TCP/IP* để kết nối mạng thông qua kết nối các *socket (hốc)* để gửi và nhận các thông điệp (*message*). Đó chính là sự trao đổi điểm với điểm (*point - to - point*), trong đó một điểm cuối là ứng dụng của bạn.

Một *socket* đóng vai trò như là một đầu cuối của quá trình trao đổi trong một hệ thống hay giữa các hệ thống với nhau. Java cung cấp nhiều lớp để thực hiện các nhiệm vụ của *socket phục vụ* và *socket khách hàng.*

Tên lớp/ giao diện	Mô tả các chức năng
InetAddress	Biểu diễn địa chỉ của Internet
ServerSocket	Biểu diễn cho *Server Socket*
Socket	Biểu diễn cho *Client Socket*
DatagramSocket	Biểu diễn cho *Datagram Socket,* một dạng cài đặt của giao lễ kết nối
DatagramPacket	Biểu diễn cho *Datagram Packet,* mô tả chi tiết về nơi nhận và về dữ liệu được gửi đi
SocketImpl	Cài đặt cho *Server Socket* và *Socket*
SocketImplFactory	Một thể hiện cụ thể của *SocketImpl*

Bảng B9.5 Các lớp xử lý *socket trên mạng*

Socket của Java

Trong Java có lớp *Socket* ở gói *java.net* để thực hiện các kết nối theo TCP/IP. Ví dụ, để kết nối một *socket* với Web Server thực hiện qua cổng mặc định (*port* 80) ở địa chỉ: http://www.javasoft.com thì cần phải tạo ra một đối tượng của *Socket* như sau:

Socket soc = new Socket("www.javasoft.com",80);

Khi đó có thể xảy ra hai ngoại lệ:

1. Gặp ngoại lệ *UnknownHostException:* không xác định được máy chủ hay máy chủ khai báo không hợp lệ;
2. Gặp ngoại lệ *IOException:* nếu kết nối *socket* với máy chủ hợp lệ nhưng không thực hiện được.

Khi kết nối thành công thì việc tiếp theo là thực hiện trao đổi các thông điệp thông qua *InputStream* và *OutputStream.*

InputStream instream = soc.getInputStream();

OutputStream instream = soc.getOutputStream();

Ví dụ 9.4 Sử dụng lớp *Socket* để tìm thông tin (đọc các tệp *html*) từ *Web Server.* Giả sử chúng ta muốn truy nhập thông tin ở địa chỉ:

http://www.nus.edu.sg:80/NUSinfo/UG/ug.html

Lưu ý: Cú pháp của HTTP URL có dạng chính:

http://hostName[:portNumber]/directory/fileName

```
// WebRetriever.java
import java.io.*;
```

```
import java.net.*;
class WebRetriever {
     Socket soc;                    // Đối tượng để thực hiện kết nối socket
     OutputStream os;               // Đối tượng để gửi thông điệp
     InputStream is;                // Đối tượng để nhận thông điệp
     WebRetriever(String server, int port) throws IOException,
                                     UnknownHostException{
          soc = new Socket(server, port);
          os = soc.getOutputStream();
          is = soc.getInputStream();
          }
     void request(String path){// Gửi đi một thông điệp (yêu cầu)
          try{
               String msg = "GET " + path + "\n\n";
               os.write(msg.getBytes());
         }catch(IOException e){
               System.err.println("Error in HTTP request!");
               }
          }
     void getResponse(){        // Nhận một thông điệp (để trả lời)
          int c;
          try{
               while((c = is.read()) != -1)    // Đọc từng ký tự
                 System.out.print((char)c);    // In ra từng ký tự
               }catch(IOException e){
                 System.err.println("Error in reading from Web Server!");
               }
          }
     public void finalize(){    // Dọn dẹp (giải phóng các đối tượng)
          try{
               is.close();os.close();soc.close();
               }catch(IOException e){
                  System.err.println("Error in closing connection!");
               }
          }
     public static void main(String args[]){
          try{
               WebRetriever w = new WebRetriever("www.nus.edu.sg",80);
               w.request("/NUSinfo/UG/ug.html");
               w.getResponse();
               }catch(UnknownHostException h){
                    System.err.println("Hostname Unknown!");
               }catch(IOException e){
                    System.err.println("Error in connecting to Host!");
               }
          }
     }
```

Lớp ServerSocket

Trong ví dụ 9.4 chúng ta đã xét một chương trình thực hiện kết nối với máy phục vụ (*Server*). Tiếp theo chúng ta muốn xây dựng một chương trình để phục vụ khách hàng (*Client*). Chúng ta xét trường hợp đối xứng, trường hợp trao đổi hai chiều.

Trong mô hình mạng của Java, các yêu cầu của khách hàng được xử lý bởi lớp *ServerSocket.*

Để kết nối được với máy phục vụ thì trạm phục vụ cần phải biết được hai thông tin: số hiệu của cổng kết nối và máy chủ (*Host*), còn *ServerSocket* thì cần phải biết số hiệu của máy khách.

Quá trình thực hiện theo mô hình *Server / Client* có thể mô tả khái quát như sau:

Quá trình phục vụ	Quá trình yêu cầu (khách hàng)
ServerSocket s = new ServerSocket (8080);	Quá trình thực hiện công việc riêng
// Cổng đã đăng ký chờ yêu cầu của khách	↓
Socket soc1 = s.accept();	
// Chờ khách sắp tới	
// Sự thực hiện của *Server* bị treo lại	*Socket soc2 = new Socket(server,7070)*
↓ Chờ . . .	
// Nhận được yêu cầu của khách hàng	
// Trạm *Server* hoạt động trở lại	

Hình H9-2 Quá trình thực hiện *Server/Client*

Hiển nhiên hai đối tượng *soc1, soc2* là đối ngẫu nhau: một đối tượng đọc những thông tin mà đối tượng kia gửi và ngược lại đối tượng này gửi và đối tượng kia nhận.

Ví dụ 9.5 Xây dựng lớp WebServer để phục vụ cho khách hàng.

```
import java.io.*;
import java.net.*;
import java.util.*;
class WebServer {
     Socket soc;
     OutputStream os;
     DataInputStream is;
     String resource;
     // Nhận các yêu cầu
     void getRequest(){
          try{
```

```
            String msg;
            while((msg = is.readLine()) != null){
                  // Nếu không có yêu cầu thì thoát khỏi chu trình while
                  if(msg.equals("")) break;
                  // Sử dụng lớp StringTokenizer để gói các xâu
                  StringTokenizer t= new StringTokenizer(msg);
                  // Lấy ra từ đầu của mỗi dòng
                  String token = t.nextToken();
                  // Nếu token là "GET" thì đọc token tiếp
                  if(token.equals("GET"))
                        resource = t.nextToken();
                  }
            }catch(IOException e){
              System.err.println("Error receiving Web request!");
            }
      }
void returnResponse(){
      int c;
      try{
        FileInputStream f = new FileInputStream("." + resource);
        while((c = is.read()) != -1)
            os.write(c);          // Ghi từng ký tự lên tệp
            f.close();
            }catch(IOException e){
              System.err.println("Error in reading from Web Server!");
            }
      }
public void finalize(){
      try{
            is.close();os.close();soc.close();
            }catch(IOException e){
              System.err.println("Error in closing connection!");
            }
      }
```

```
WebServer(Socket s) throws IOException{
     soc = s;
     os = soc.getOutputStream();
     is = new DataInputStream(soc.getInputStream());
     }

public static void main(String args[]){
     try{
          ServerSocket s = new ServerSocket(8080);
          WebServer w = new WebServer(s.accept());
          w.getRequest();
          w.returnResponse();
          }catch(IOException e){
               System.err.println("Error in Server!");
          }
     }
}
```

Muốn biết chi tiết và đầy đủ về lập trình trên mạng với các giao diện AWT API, bạn có thể tham khảo ở tài liệu [8].

Tài liệu tham khảo

[1] Barry Boone, *Java Essentials for C and C++ Programmers,* Addition-Wesley, 1996.

[2] Booch G., Rumbaugh J. and Jacoson I., *The Unified Software Development Process,* Addision - Wesley, 1998.

[3] Đoàn Văn Ban, *Phân tích, thiết kế và lập trình hướng đối tượng*, nhà XB Thống Kê, HN 1997.

[4] David Flanagan, Jim Farley, et. all, *Java Enterprise in a Nutshell,* O'Reilly & Associates, 1999, http://java.oreilly.com.

[5] Dennis Kafura, *Object-Oriented Software Design and Construction with Java,* Prentice Hall, 2000.

[6] James Gosling, Frank Yellin, *The Java™ Application Programming Interface, Volume I, II,* Addition-Wesley, 1996, or http://java.sun.com.

[7] Kalajdziski S., *The Unified Modeling Language Tutorial in 7 days,* http://odl-skopje.etf.ukim.edu.mk/uml-help/, 2001.

[8] Khalid A. Mughal and Rolf W. Rasmussen, *A Programmer's Guide to Java™ Certification,* Addition-Wesley, 1999, http://www.awl.com/cseng.

[9] Natarai Nagaratnam, et. all, *Java Networking and AWT API Superbible,* WaiteGroup Press, 1998.