Chapter 35 Advanced Java Database Programming

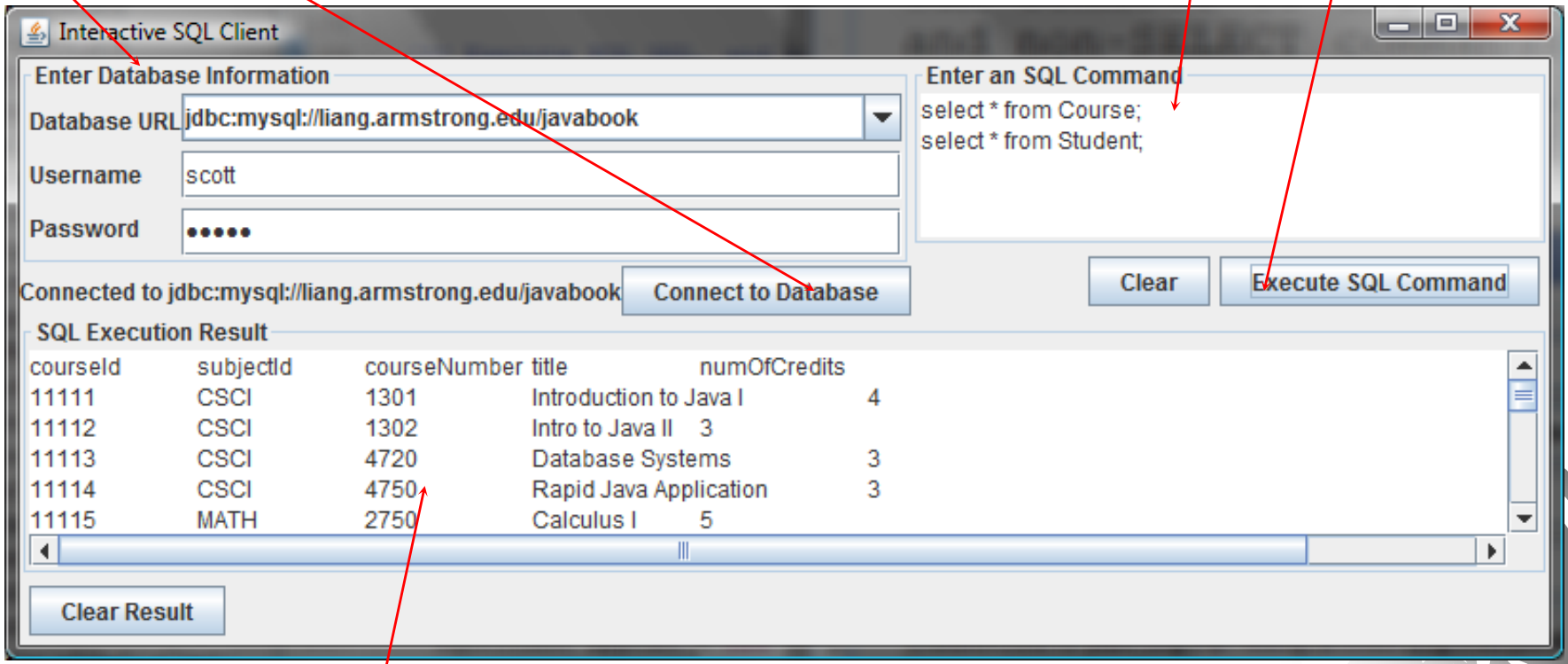
Objectives

- To create a universal SQL client for accessing local or remote database (§35.2).
- To execute SQL statements in a batch mode (§35.3).
- To process updatable and scrollable result sets (§35.4).
- To simplify Java database programming using RowSet (§35.5).
- To store and retrieve images in JDBC (§35.6).

Example:
Creating an Interactive SQL Client

Connect to any JDBC data source.

Entering and executing SQL commands interactively



The execution result is displayed for the SELECT queries, and the execution status is displayed for the non-SELECT commands.

SQLClient

Run

Batch Updates

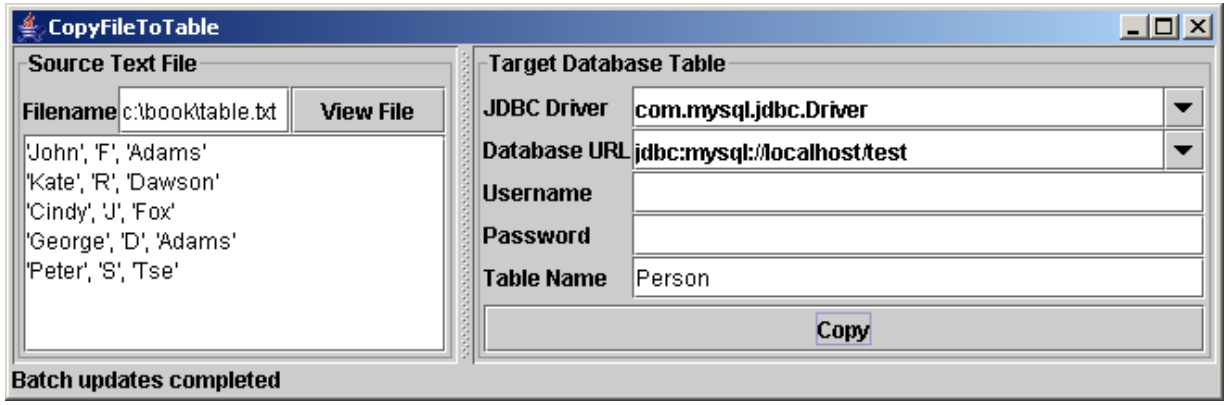
To improve performance, JDBC 2 introduced the batch update for processing nonselect SQL commands. A batch update consists of a sequence of nonselect SQL commands. These commands are collected in a batch and submitted to the database all together.

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

// Add SQL commands to the batch
statement.addBatch("create table T (C1 integer, C2 varchar(15))");
statement.addBatch("insert into T values (100, 'Smith')");
statement.addBatch("insert into T values (200, 'Jones')");

// Execute the batch
int count[] = statement.executeBatch();
```

The executeBatch() method returns an array of counts, each of which counts the number of the rows affected by the SQL command. The first count returns 0 because it is a DDL command. The rest of the commands return 1 because only one row is affected.

Example: Copying Text Files to Table

Write a program that gets data from a text file and copies the data to a table. The text file consists of the lines, each of which corresponds to a row in the table. The fields in a row are separated by commas. The string values in a row are enclosed in single quotes. You can view the text file by clicking the View File button and copy the text to the table by clicking the Copy button. The table must already be defined in the database.



CopyFileToTable Run

Scrollable and Updateable Result Set

The result sets used in the preceding examples are read sequentially. A result set maintains a cursor pointing to its current row of data. Initially the cursor is positioned before the first row. The next() method moves the cursor forward to the next row. This is known as *sequential forward reading*. It is the only way of processing the rows in a result set that is supported by JDBC 1.

With JDBC 2, you can scroll the rows both forward and backward and move the cursor to a desired location using the first, last, next, previous, absolute, or relative method. Additionally, you can insert, delete, or update a row in the result set and have the changes automatically reflected in the database.

Creating Scrollable Statements

To obtain a scrollable or updateable result set, you must first create a statement with an appropriate type and concurrency mode. For a static statement, use

```
Statement statement = connection.createStatement
  (int resultSetType, int resultSetConcurrency);
```

TYPE_FORWARD_ONLY
TYPE_SCROLL_INSENSITIVE
TYPE_SCROLL_SENSITIVE

For a prepared statement, use

```
PreparedStatement statement = connection.prepareStatement
  (String sql, int resultSetType, int resultSetConcurrency);
```

CONCUR_READ_ONLY
CONCUR_UPDATABLE

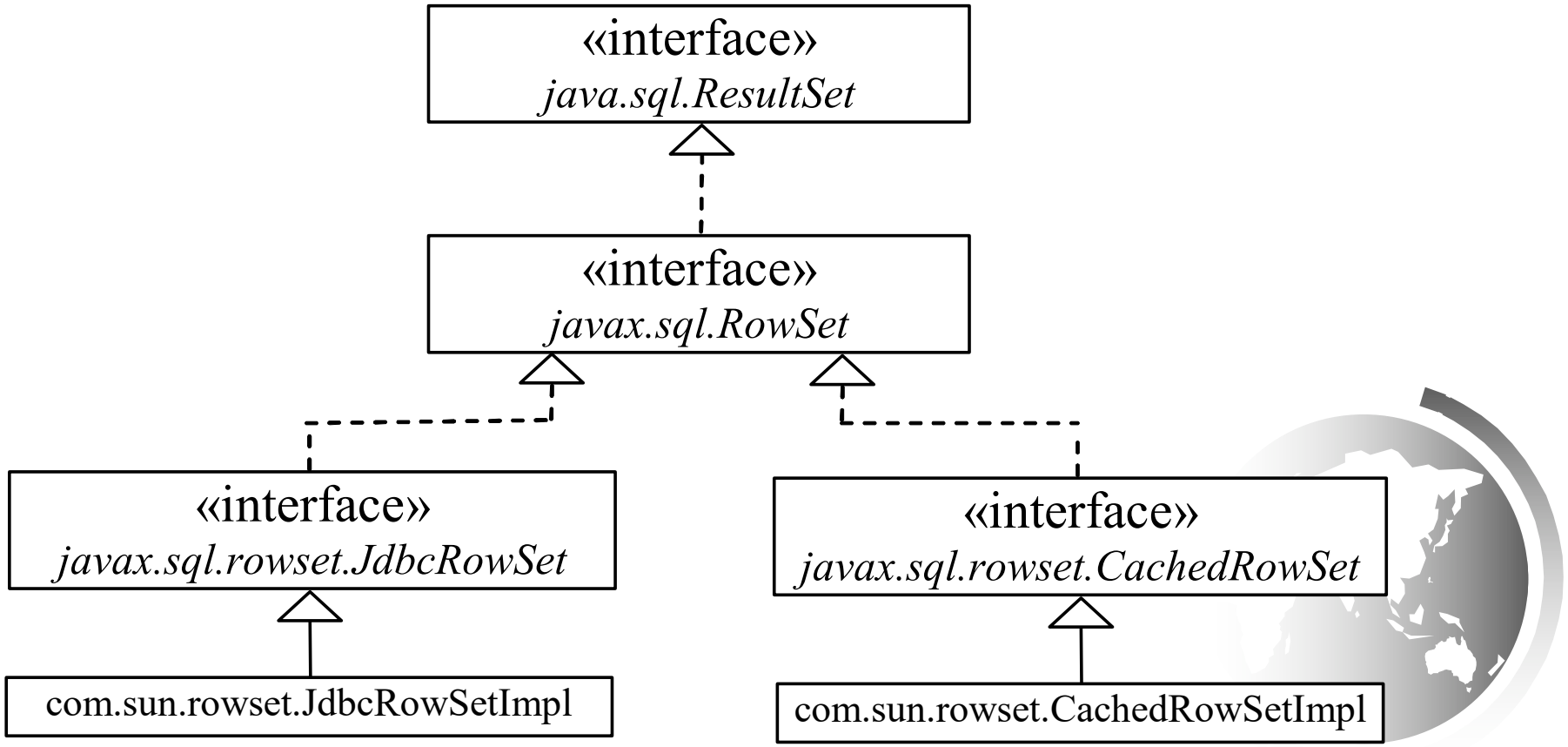
The resulting set is scrollable

```
ResultSet resultSet = statement.executeQuery(query);
```

ScrollUpdateResultSet Run

RowSet: JdbcRowSet and CachedRowSet

JDBC 2 introduced a new RowSet interface that can be used to simplify database programming. The RowSet interface extends java.sql.ResultSet with additional capabilities that allow a RowSet instance to be configured to connect to a JDBC url, username, password, set a SQL command, execute the command, and retrieve the execution result.



SQL BLOB and CLOB Types

BLOB

Database can store not only numbers and strings, but also images. SQL3 introduced a new data type BLOB (*B*inary *L*arge *OB*ject) for storing binary data, which can be used to store images.

CLOB

Another new SQL3 type is CLOB (*C*haracter *L*arge *OB*ject) for storing a large text in the character format. JDBC 2 introduced the interfaces java.sql.Blob and java.sql.Clob to support mapping for these new SQL types. JBDC 2 also added new methods, such as getBlob, setBinaryStream, getClob, setBlob, and setClob, in the interfaces ResultSet and PreparedStatement to access SQL BLOB, and CLOB values.

To store an image into a cell in a table, the corresponding column for the cell must be of the BLOB type. For example, the following SQL statement creates a table whose type for the flag column is BLOB.

```
create table Country(name varchar(30), flag blob,
  description varchar(255));
```

Storing and Retrieving Images in JDBC

To insert a record with images to a table, define a prepared statement like this one:

```
PreparedStatement pstmt = connection.prepareStatement(
  "insert into Country values(?, ?, ?)");
```

Images are usually stored in files. You may first get an instance of InputStream for an image file and then use the setBinaryStream method to associate the input stream with a cell in the table, as follows:

Store image

```
// Store image to the table cell
File file = new File(imageFilenames[i]);
InputStream inputImage = new FileInputStream(file);
pstmt.setBinaryStream(2, inputImage, (int)(file.length()));
```

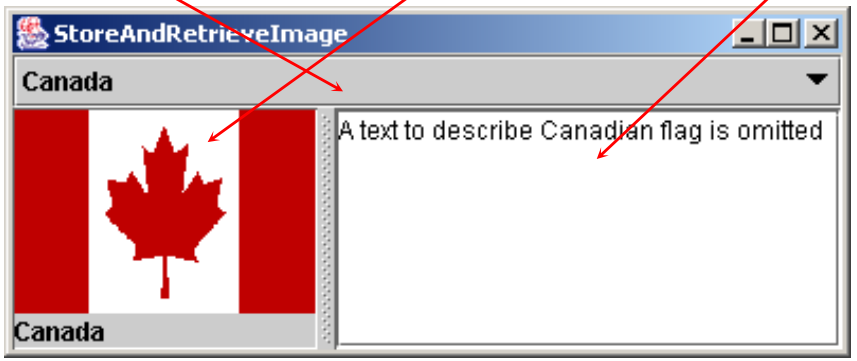
Retrieve image

To retrieve an image from a table, use the getBlob method, as shown below:

```
// Store image to the table cell
Blob blob = rs.getBlob(1);
ImageIcon imageIcon = new ImageIcon(
  blob.getBytes(1, (int)blob.length()));
```

Example: Scrolling and Updating Table

In this example, you will create a table, populate it with data, including images, and retrieve and display images. The table is named Country. Each record in the table consists of three fields: name, flag, and description. Flag is an image field. The program first creates the table and stores data to it. Then the program retrieves the country names from the table and adds them to a combo box. When the user selects a name from the combo box, the country's flag and description are displayed.



StoreAndRetrieveImage Run