G52APR

Applications Programming

Java DataBase Connectivity – JDBC

What is JDBC?

- "An API that lets you access virtually any tabular data source from the Java programming language"
- "… access virtually any data source, from relational databases to spreadsheets and flat files."
- We'll focus on accessing Oracle type databases
- See http://docs.oracle.com/javase/tutorial/jdbc/

Tabular data source

Table 1: Employees

Employee_Number	First_Name	Last_Name	Date_of_Birth	Car_Number
10001	Axel	Washington	28-Aug-43	5
10083	Arvid	Sharma	24-Nov-54	null
10120	Jonas	Ginsberg	01-Jan-69	null
10005	Florence	Wojokowski	04-Jul-71	12

Relational Database

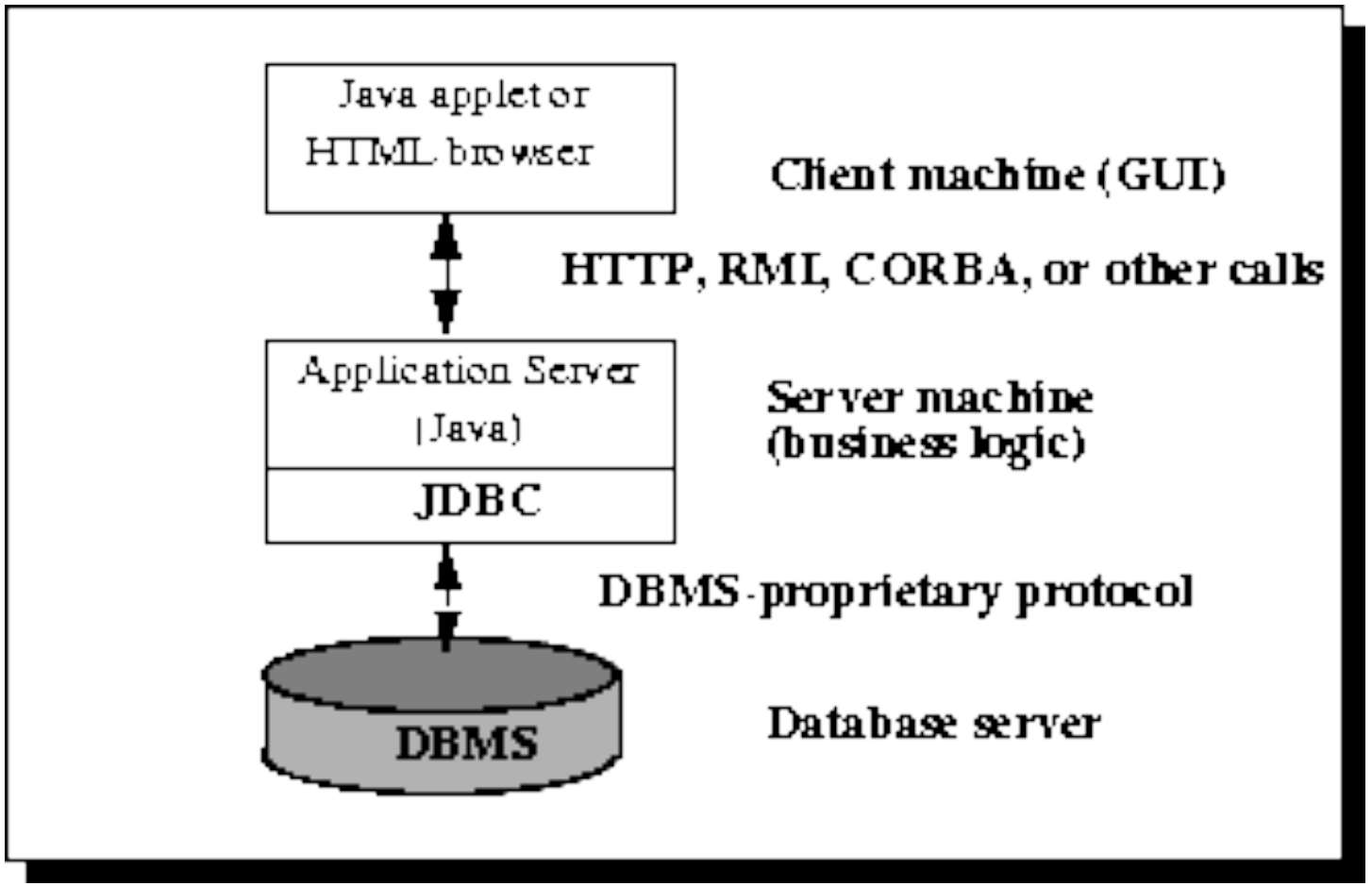
Employee_Number	First_Name	Last_Name	Date_of_Birth	Car_Number
10001	Axel	Washington	28-Aug-43	5
10083	Arvid	Sharma	24-Nov-54	null
10120	Jonas	Ginsberg	01-Jan-69	null
10005	Florence	Wojokowski	04-Jul-71	12

Car_Number	Make	Model	Year
5	Honda	Civic DX	2006
12	Toyota	Corolla	2009

Relational Database

- A relational database presents information in tables with rows and columns.
- A distinguishing feature of relational databases is that it is possible to get data from more than one table in what is called a join.
- A Relational Database Management System (RDBMS) handles the way data is stored, maintained, and retrieved.
- Structural Query Language (SQL) is a language designed to be used with relational databases.

General Architecture



Basic steps to use a database

1. Establish a connection
2. Create JDBC Statements
3. Execute Statements
4. GET ResultSet
5. Close the connection

1. Establish a connection

- **import java.sql.*;**
- **Load the vendor specific driver**

```
Class.forName("oracle.jdbc.driver.OracleDriver");
    // Dynamically loads a driver class, for Oracle database
```

- **Make the connection**

```
Connection con =
DriverManager.getConnection("jdbc:mysql://
  localhost:1527/Employees", username, passwd);
// Establishes connection to database by obtaining a
Connection object
```

Database address

- The address of the database is:

  jdbc:mysql://localhost:1527/Employees

  The first part, **jdbc:derby://localhost**, is the database type and server that you're using. The 1527 is the port number. The database name is **Employees**. This can all go in a String variable:

```
String host = "jdbc:mysql://localhost:1527/Employees";
```

  Two more strings can be added for the username and password:

```
String uName = "Your_Username_Here";
String uPass= " Your_Password_Here ";
Connection con=DriverManager.getConnection(host,
    uName, uPass);
```

Connection interface

- Statement **createStatement**()
  Creates a Statement object for sending SQL statements to the database.
- void **close**()
  Releases this Connection object's database and JDBC resources immediately instead of waiting for them to be automatically released.
- void **commit**()
  Makes all changes made since the previous commit/rollback permanent and releases any database locks currently held by this Connection object.
- void **rollback**()
  Undoes all changes made in the current transaction and releases any database locks currently held by this Connection object.
- void **setAutoCommit**(boolean autoCommit)
  Sets this connection's auto-commit mode to the given state.
- Statement **createStatement**(int resultSetType, int resultSetConcurrency)
  Creates a Statement object that will generate ResultSet objects with the given type and concurrency.
- ……

2. Create JDBC statement(s)

- Statement stmt = con.createStatement() ;

// Creates a Statement object for sending SQL statements to the database

Statement interface

- ResultSet **executeQuery**(String sql)
  Executes the given SQL statement, which returns a single ResultSet object.
- int **executeUpdate**(String sql)
  Executes the given SQL statement, which may be an INSERT, UPDATE, or DELETE statement or an SQL statement that returns nothing, such as an SQL DDL statement.
- void **close**()
  Releases this Statement object's database and JDBC resources immediately instead of waiting for this to happen when it is automatically closed.
- ……

3. Executing SQL Statements

- String queryThing = " SELECT * FROM Employees";
  stmt.executeQuery(queryThing);

- String insertThing = "Insert into Thing values“ +
  "(123456789,abc,100)";
  stmt.executeUpdate(insertThing);

4. Get ResultSet

String queryThing = “SELECT * FROM Employees";

ResultSet rs = Stmt.executeQuery(queryThing);

```
while (rs.next()) {
    int ssn = rs.getInt("Employee_Number");
    String first_name = rs.getString(“First_name");
    String last_name = rs.getString(“Last_name");
    int marks = rs.getInt(“Car_Number");
}
```

5. Close connection

- stmt.close();
  // close statement
- con.close();
  // close connection

Summary

- Three interfaces and three methods.
- DriverManager.getConnection(host, uName, uPass);
  - > Connection interface
- Connection.createStatement()
  - > Statement interface
- Statement.executeQuery(queryThing)
  - > ResultSet interface

An Example

```
public static void viewTable(Connection con, String dbName) throws SQLException {
    Statement stmt = null;
    String query = "SELECT Employee_Number, First_Name, Last_Name, Date_of_Birth, Car_Number
    FROM " + dbName";
    try {
        stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery(query);
        while (rs.next()) {
            int EmployeeNumber = rs.getInt(" Employee_Number");
            String FirstName = rs.getString(" First_Name");
            String LastName = rs.getString(" First_Name");
            String DateOfBirth = rs.getString(" Date_of_Birth");
            int CarNumber = rs.getInt("Car_Number");
            System.out.println(EmployeeNumber + "\t" + FirstName + "\t" + LastName  + "\t" +
                CarNumber);
        }
    } catch (SQLException e ) {
        JDBCTutorialUtilities.printSQLException(e);
    } finally {
        stmt.close();
    }
}
```

SQL commands

- *String query = "SELECT Employee_Number, Date_of_Birth, Car_Number FROM Employees WHERE Car_Number IS NOT NULL";*

- Data Manipulation Language (DML)
  - SELECT
  - INSERT
  - DELETE
  - …
- Data Definition Language (DDL)
  - CREATE TABLE
  - ALTER TABLE
  - …

SQL commands

- ResultSet rs = stmt.executeQuery(“**SELECT * FROM table_name**”)

// select all the records from a table

- ResultSet rs = stmt.executeQuery(“ **SELECT * FROM table_name WHERE column_name IS NOT NULL**”);

// select all the records from a table

- …(“**SELECT col_blob FROM mysql_all_table**”);

// select column reference from all tables

http://download.oracle.com/javase/1.4.2/docs/api/java/sql/package-summary.html

NOTE: SQL is not case sensitive
A BLOB is a reference to data in a database.

Transactions and JDBC

- JDBC allows SQL statements to be grouped together into a single transaction.
- “Sequence of operations performed as a single logical unit of work”.
  - Atomic: all the work in the transaction is treated as a single unit. Either it is all performed or none of it is.
  - Consistent: a completed transaction leaves the database in a consistent internal state.
  - Isolations: the transaction sees the database in a consistent state. If two transactions try to update the same table, one will go first and then the other will follow.
  - Durability means that the results of the transaction are permanently stored in the system.

Transactions and JDBC

- Transaction control is performed by the Connection object, default mode is auto-commit, i.e., each SQL statement is treated as a transaction.
- We can turn off the auto-commit mode with con.setAutoCommit(false);
- And turn it back on with con.setAutoCommit(true);
- Once auto-commit is off, no SQL statement will be committed until an explicit commit is invoked con.commit();
- At this point all changes done by the SQL statements will be made permanent in the database.

Using Transactions

- Step 1: turn off autocommit:
  - conn.setAutoCommit(false);
- Step 2: create and execute statements like normal
- Step 3: commit or rollback
  - if all succeeded:
    - conn.commit();
  - else, if one or more failed:
    - conn.rollback();
- Step 4: turn autocommit back on
  - conn.setAutoCommit(true);

Handling Errors with Exceptions

- Programs should recover and leave the database in a consistent state.
- If a statement in the try { …} block throws an exception or warning, it can be caught in one of the corresponding catch statements.
- E.g., you could rollback your transaction in a catch { …} block or close database connection and free database related resources in finally {…} block.

Interface ResultSet

- A table of data representing a database result set, which is usually generated by executing a statement that queries the database.
- A ResultSet object 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 to the next row
  - returns false when there are no more so it can be used in a while loop to iterate through the result set.
  - Not updatable and has a cursor that moves forward only.

More ResultSet details

- The ResultSet interface provides getter methods (getBoolean, getInt, and so on) for retrieving column values from the current row.
- Values can be retrieved using either the index number of the column or the name of the column (In general, using the column index will be more efficient).
- Columns are numbered from 1. (For maximum portability, result set columns within each row should be read in left-to-right order, and each column should be read only once).
- For the getter methods, a JDBC driver attempts to convert the underlying data to the Java type specified in the getter method and returns a suitable Java value.

Other Functionality

- Create new tables
- Scrollable result sets
- Updateable result sets
- Cursor methods
- DB metadata

JDBC references

- JDBC Data Access API – JDBC Technology Homepage
  - http://java.sun.com/products/jdbc/index.html
- JDBC Database Access – The Java Tutorial
  - http://java.sun.com/docs/books/tutorial/jdbc/index.html
- JDBC Documentation
  - http://java.sun.com/j2se/1.4.2/docs/guide/jdbc/index.html
- java.sql package
  - http://java.sun.com/j2se/1.4.2/docs/api/java/sql/package-summary.html
- JDBC Technology Guide: Getting Started
  - http://java.sun.com/j2se/1.4.2/docs/guide/jdbc/getstart/GettingStartedTOC.fm.html
- JDBC API Tutorial and Reference (book)
  - http://java.sun.com/docs/books/jdbc/
- Oracle Java Tutorial
  - http://download.oracle.com/javase/tutorial/jdbc/basics/gettingstarted.html

JDBC at Nottingham

- Use the mysql database provided by the School.
- The TSG guide is at

https://support.cs.nott.ac.uk/help/docs/databases/mysql/jdbc.html.

Steps to follow are…

- You should log into the server and run: create_mysql
  - generates your username, password and database.
- MAKE A NOTE of the all the details
  - these cannot be recovered if you forget them!!!!!
- The host and port is:
  - mysql.cs.nott.ac.uk:3306

Working from Home

- port forward over ssh using: ssh -L 3306:mysql.cs.nott.ac.uk:3306 username@server.cs.nott.ac.uk
- *OR temporarily* use a local mysql database (eg Wamp or Mamp) BUT you MUST make it work with the Schools system ready for when we test it.

Example code

```
Import java.sql.*;

// create db outside of java: REMEMBER THE PASSWORD (you may already have one from database module)
// however you can create the table (if it doesn't exist) in java (and empty it if necessary?)

String url;
try {
    Connection conn;
    Statement stmt;
    Class.forName ("com.mysql.jdbc.Driver").newInstance ();
    url = "jdbc:mysql://mysql.cs.nott.ac.uk:3306/$YOUR_USER_NAME$”;
    conn = DriverManager.getConnection(url, "$YOUR_USER_NAME$", ”$YOUR_PASSWORD$");
    stmt = conn.createStatement();
    // create SQL statement here
    stmt.executeUpdate($SQL_STATEMENT$);
    //other executes also available
    conn.close();
}
catch (Exception e) {
    System.err.println("Got an exception! ");
    System.err.println(e.getMessage());
}
```

SET/SEM

G52APR: Application Programming

The University of Nottingham
UNITED KINGDOM · CHINA · MALAYSIA

Evaluate Survey Details for Students

07/11/2014 12:15

To take a survey for Application Programming please go to https://bluecastle.nottingham.ac.uk
and log in using your user name and password
You have from **11/11/2014 15:00** to **11/11/2014 17:00** to complete this survey