Chapter 8

Hybrid languages: JDBC

- Mixing SQL and general purpose programming languages
  - The need
  - ResultSet: The main point of confluence
- JDBC
- Efficiency issues in JDBC
- Support for metadata

8.1. Java Database Connectivity (JDBC)

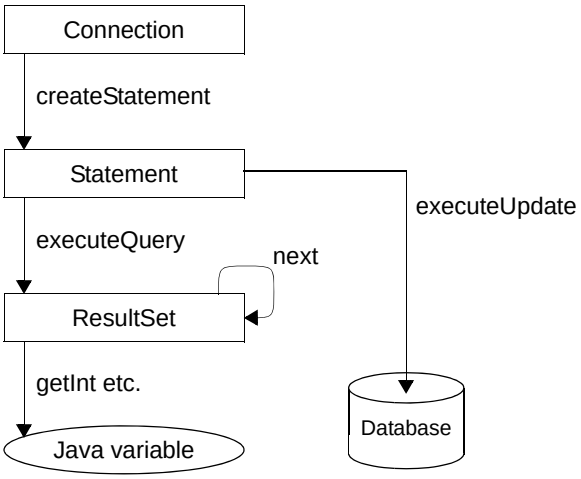
- Also visit our web page on JDBC for WHERE IS IT
  - A tutorial that is a detailed version of this slide presentation
  - A demo that mainly consists of JDBCDemo.java program
  - The demo program is explained in detail here and in the tutorial
  - Note that JDBC + a simple driver are included in JDK, already installed by you in Project 0.
- In these slides we discuss
  - JDBC in some detail
  - Give a detailed example drawn from JDBCDemo.java
  - How to write efficient JDBC programs
- JDBC is a hybrid of SQL and general purpose programming language Java
  - Java acts as a host language for issuing SQL commands
  - It is similar to ODBC, open database connectivity
  - There are other hybrids, e.g. Oracles's SQL/PL
  - JDBC is object-oriented and it has grown more systematically
- Sun Microsystem's web page *JDBC Data Access API*"
  - at http://java.sun.com/products/jdbc/
  - This is the definitive JDBC reference

8.2. JDBC = A driver + java.sql package

- One needs a driver to connect to a database
  - The drivers are specific to a database platform
  - Drivers are typically manufactured by database or third party vendors
  - In our demo we use a driver supplied by Sun that is bundled with JDK
- JDBC essentially consists of java.sql and javax.sql packages
- They are included in the JDK that you installed in Project 0
- We mainly concentrate on java.sql package
- JDBC is huge, we concentrate on the core part

8.3. java.sql package

- java.sql consists of interfaces and classes
  - The diagram below shows classes and interfaces as rectangles
  - Directed arrows shows transition from one object type to another
  - Labels show functions



- Connection objects allow a connection through a driver
- Statement object can hold SQL statement to be executed
- An update statement can be executed to update the database
  - In "executeUpdate" the term "update" is drawn from English
  - It refires REFERS to execution of insert, delete and update of SQL
- A query is executed to retrieve a ResultSet object
  - A resultset can be scanned one tuple at a time
  - Attribute values can be brought into ordinary java variables

8.4. ResultSet

- The concept of a ResultSet is the core of all hybrid languages
- A ResultSet is used as an iterator
  - A resultset is opened automatically when a query is executed:
    rs1 = stmt1.executeQuery ("... an SQL query ...");
  - rs1.next() returns a boolean; it returns TRUE if there is a next tuple
    the function next() also brings the next tuple in rs1
  - rs1.close() closes rs1
- A resultset allows tuple-at-a-time scan
- Contents of a tuple can be brought into java variables
- Complex decisions can be made and actions can be taken
- This is what ~~that~~ brings the power of a general purpose programming language at our disposal
- This is far more powerful than what can be done in SQL
- Another term for the resultset in hybrid languages is cursor

8.5. An example in 7 slides: Steps in a JDBC program

- Example: Consider Emp(Name, DName, Salary) relation
  - 1. We want to create a report of employees in Toys department
  - 2. We want to transfer two employees from Toys to other departments
- The JDBC program will consist of the following steps
  - Step 0. Import the java.sql package.
  - Step 1. Load and register the driver.
  - Step 2. Establish the connection to the database
  - Step 3. Create a statement that is a query on employees in Toys
  - Step 4. Execute the statement to create a resultset
  - Step 5. Process the resultset to print a report
  - Step 6. Close the statement.
- For executing updates we need following steps
  - Step 3. For the existing connection, create a PreparedStatement
    A prepared statement is compiled once executed many times
  - Step 4. Execute the prepared statement twice, once for each update
  - Step 7. Commit transactions on the connection, close the connection
- We show these steps in detail in the following slides

8.6. An example in 7 slides: import, load and register

- Step 0. Import the java.sql package

```
// Step 0. Import Java.sql package
import java.sql.*;
```

- Step 1. Load and register a JDBC driver

```
// Step 1: Load the driver
try {
   // Load the driver (registers itself)
   Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
}
catch (Exception E) {
   System.err.println ("Unable to load driver.");
   E.printStackTrace ();
}
```

  - Driver "sun.jdbc.odbc.JdbcOdbcDriver" is bundled with JDK
  - The DriverRegister method does not appear in the code.
  - The driver calls the DriverManager.registerDriver() to register itself
  - There are four types of drivers, not considered here

8.7. An example in 7 slides: establishing a connection

- Step 2. Establishing a connection

```
try {
   // Step 2. Connect to the database
   Connection conn1; // An object of type connection
   String dbUrl = "jdbc:odbc:DemoDataSource";
   String user = "";
   String password = "";
   conn1 = DriverManager.getConnection (dbUrl, user, password);
   System.out.println ("*** Connected to the database ***");
   . . .
} // End of try
catch (SQLException E) {
   System.out.println ("SQLException: " + E.getMessage());
   System.out.println ("SQLState: " + E.getSQLState());
   System.out.println ("VendorError: " + E.getErrorCode());
} // End of catch
```

- The code for connection embedded in a try-and-catch page
  This allows exceptions to be handled by exception handler
- Alternate syntax allows prompting for username and password
- Connections are complicated and fragile objects
  A connection may not succeed in the first attempt
  A connection may ~~disconnect~~END? unexpectedly
- Therefore, parameters such as “MaxReconnects” are allowed
- Parameters have reasonable default values associated with them
- For noisy connections higher MaxReconnects may be provided
- These details are very interesting, but not considered in the course
- From a JDBC program, connections can be made to multiple databases, spreadsheets, text files, even residing on diverse platforms

8.8. An example in 7 slides: create and execute statement

- Step 3. Create a Statement

```
// Step 3A. Create stmt1 object AND ASSOCIATE IT WITH conn1
Statement stmt1 = conn1.createStatement ();
```

A STATEMENT HAS TO BE ASSOCIATED WITH A CONNECTION

- Step 4. Execute the Statement

```
// Step 4A. Execute a query, receive result in a result set
ResultSet rs1 = stmt1.executeQuery ("select e.Name, e.Salary" + " " +
                                    "from Emp e" + " " +
                                    "where e.DName = 'Toys' ");
```

- Note a blank (" " +) is forced between two lines
- The resultset object is created to hold the result of the query

8.9. An example in 7 slides: process result set

- Step 5. Process the result set
  - The concept of a resultset has already been discussed

```
// Step 5A. Process the result set
. . .
int count = 0;
double totalPayroll = 0.0;
String jName; // To store value of Name attribute
double jSalary; // To store value of Salary attribute
while(rs1.next()) {
   // Access and print contents of one tuple
   jName = rs1.getString (1); // Value accessed by its position
   jSalary = rs1.getInt ("Salary"); // Access by attribute Name
   System.out.println (jName + " " + jSalary);
   count = count + 1;
   totalPayroll = totalPayroll + jSalary;
}
. . .
```

  - get functions are used to extract attribute values in java variables
    For an integer attribute getInt( ) is used, similar for other types
  - Attribute values can be referred to by ~~its~~THEIR position or name
    For example, see rs1.getString(1) and rs1.get~~String~~INT("Salary")
  - Attribute values are deposited in variables of the host language (Java) in order to make decisions based upon their values.
  - In the sample code jSalary and jName are such variables.
  - Just like an ordinary java program, a count and running total of salaries are calculated in the loop.
  - Contents of the tuple are also printed in the loop
  - On exit from the loop the count and grand total of salaries are printed

8.10. An example in 7 slides: close and commit

- Step 6. Close the statement

```
// Step 6A. Close statement
stmt1.close ();
```

- Step 7. Close the connection
  - It is appropriate to close a connection when it is no longer needed
  - commit() function should also be used in case a batch of updates is made through a connection
  - If something goes wrong with the connection no changes will be made to the database
  - If everything goes well, all transactions will be applied to the database
  - This may lead to the following code

```
// Step 7. Commit transactions on conn1 and close it
conn1.commit();
conn1.close ();
```

  - But, we choose to make the two updates using the same connection
  - Therefore, the above code appears much later in JDBCDemo.java

8.11. An example in 7 slides: prepared statement

- Using a prepared statement
  - We use a PreparedStatement
  - Such statement is compiled once, executed many times

```
// Step 3B. Create a Prepared Statement object
PreparedStatement stmt2 =
   conn1.prepareStatement ("update Emp" + " " +
                           "set DName=? , Salary=?" + " " +
                           "where Name = ? " );
// Step 4B(i) Execute stmt2: Move Hari to Shoes with a salary of $65,000:
stmt2.setString(1,"Shoes");
stmt2.setInt(2,65000);
stmt2.setString(3,"Hari");
stmt2.executeUpdate();

// Step 4B(ii) Execute stmt2: Move Leu to Credit with a salary of $75,000
stmt2.setString(1,"Credit");
stmt2.setDouble(2,75000.0);
stmt2.setString(3,"Leu");
stmt2.executeUpdate();
```

  - Note "?" are place holders in the prepared statement for actual values
  - These values are set before executing the statement every time

8.12. Efficiency and Logic in JDBC (3 slides)

- The efficiency in SQL is usually ~~differed~~DEFERRED to the system FIND POSSIBLY ANOTHER OCCURRENCE OF DIFFERED
  - This is a good practice
  - Database do a superb job at optimizing SQL queries
  - But general purpose languages such as Java are poor in this respect
  - Often JDBC programs are written carelessly;
    they deprive the database system of opportunities to do optimization.
  - This situation arises especially when simultaneous traversal of two or more relations is needed.
- Example: Consider two relations $r(A, B, C)$ and $s(B, D, E)$
  - Relations r and s occupy 200 and 400 pages, respectively
  - The relation r has 8,000 tuples. Of these, 40% satisfy A > 11.
  - The number of tuples in s do not matter in this example
  - Consider the following SQL-like query.

```
select x.A, x.C, y.D
from r x, s y
where x.B = y.B
   and x.A > 11
   and y.E = 55
   and f(A,B,C,D)
```

  - Here, f is a function that requires Java.

8.13. Efficiency and Logic in JDBC: Slide 2 of 3

- The following two JDBC solutions are possible
- First solution:
  - Execute the following query and scan r-tuples in a while loop in JDBC

```
select *
from r x
where x.A > 11
```

  - For each tuple in the result set of the above query, execute

```
select y.B, y.D
from s y
where y.E = 55
```

  - In a nested loop verify x.B is = y.B and execute f(A,B,C,D)
- Second solution
  - Execute the following query

```
select x.A, x.B, x.C, y.D
from r x, s y
where x.B = y.B
   and x.A > 11
   and y.E = 55
```

  - This query is same as in the chapter on optimization except x.B is now included in the output
  - This changes the size of output:

    Previously: $\lceil 8000 * 0.40 * 0.04 * 20 / 1000 \rceil = 3$ pages
    Now: $\lceil 8000 * 0.40 * 0.04 * (20 + 8) / 1000 \rceil = 4$ pages
  - The total cost becomes 603 page accesses instead of 602
  - In a single loop execute f(A,B,C,D)

8.14. Efficiency and Logic in JDBC: Slide 3 of 3

- Cost of first solution
  - Scan of r will require 200 page accesses. Let's assume that 40% of the tuples satisfy `A > 11`. Thus out of 8,000 tuples, 3,200 tuples will survive. Therefore, relation s will be scanned 3,200 times. Thus, the cost is 200 + 3200 * 400 = 1,280,200. With page accesses at 10 ms per page this adds up to about 3.6 hours.
  - One may argue that since no state-dependent variable is passed from outer query to the inner query, the compiler should be able to determine that the inner result set need to be computed only once. Let's suppose the inner resultset consists of 13 pages. Therefore, scanning the inner relation will require 3200 * 13 page accesses, needing about 7 minutes.
- Cost of second solution
  - As shown in lecture on optimization, under reasonable circumstances this requires 603 page accesses needing 6 seconds
- Significant gain in performance can be achieved by letting SQL rather than Java handle the nested loop
- One has to sacrifice some convenience
  - In second solution, two kinds of objects, interleaved together as a join of two tuples, arrive in a single stream.
  - By requiring the SQL query to sort the resultset, the stream can be made more logical.

8.15. When not to use Java arrays

- The concept of streaming data is fundamental in databases
  - By default such streams can be very large
  - Streams should be handled as a stream that is passing by
  - Temptations to store streaming data in arrays must be avoided
  - Senseless if the array will be processed in the same sequence as the original stream
- An instructive example is calculation of average salary
  - We only need to keep track of the running total and count
  - No need to store employee tuples in an array
    This will avoid wasting memory and slowing the application
  - The logic of an array-based solution can be applied directly to the streaming data
  - The reader should consider streaming of pairs of objects and consider carefully how to handle such a stream without wasting resources.
- A word of advice ...
  - Our projects use small databases to keep them manageable for you
  - But your code *must* reflect the mind-set that the databases are large
- When is an array appropriate?
  - Must give serious thought before using it.

8.16. Further remarks

- Alternative logic surrounding a result set
  - In DemoJDBC.java, the next() function is used as a certification for entering the body of a while loop.
  - An alternate logic is to execute next() once before entering the loop and once at the end of the body of the while loop:

```
rs1.next();
while (not eof) {
  do something with the tuple
  rs1.next()
}
```

- More recent versions of JDBC allow result sets to be traversed forward and backward and certain jumps
- JDBC allow query of metadata
  - One can connect to a database one knows nothing about
  - Copy the contents of the entire database!