Database Access with JDBC

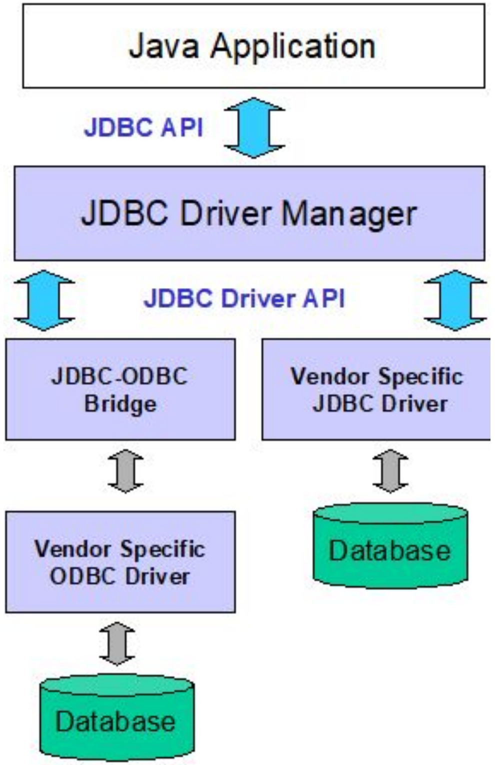
Slides: http://www.moreservlets.com

JDBC Introduction

- **JDBC provides a standard library for accessing relational databases**
  - API standardizes
    - Way to establish connection to database
    - Approach to initiating queries
    - Method to create stored (parameterized) queries
    - The data structure of query result (table)
      - Determining the number of columns
      - Looking up metadata, etc.
  - API does not standardize SQL syntax
    - JDBC is not embedded SQL
  - JDBC classes are in the java.sql package
- **Note: JDBC is not officially an acronym; unofficially, “Java DataBase Connectivity” is commonly used**

JDBC Drivers

- **JDBC consists of two parts:**
  - JDBC API, a purely Java-based API
  - JDBC Driver Manager, which communicates with vendor-specific drivers that perform the real communication with the database.
    - Point: translation to vendor format is performed on the client
      - No changes needed to server
      - Driver (translator) needed on client



Seven Basic Steps in Using JDBC

1. Load the driver
2. Define the Connection URL
3. Establish the Connection
4. Create a Statement object
5. Execute a query
6. Process the results
7. Close the connection

JDBC: Details of Process

1. **Load the driver**

```
Class.forName("com.mysql.jdbc.Driver");
```

2. **Define the Connection URL**

```
// localhost
String host = "jdbc:mysql://127.0.0.1/cars_db";
```

3. **Establish the Connection**

```
 String username = "jay_debesee";
 String password = "secret";
 Connection connection =
    DriverManager.getConnection(host,
                                username,
                                password);
```

JDBC: Details of Process (Continued)

4. **Create a Statement**

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

5. **Execute a Query**

```
String query =
  "SELECT col1, col2, col3 FROM sometable";
 ResultSet resultSet =
   statement.executeQuery(query);
```

– To modify the database, use `executeUpdate`, supplying a string that uses `UPDATE`, `INSERT`, or `DELETE`

– Use `setQueryTimeout` to specify a maximum delay to wait for results

JDBC: Details of Process (Continued)

6. **Process the Result**

```
while(resultSet.next()) {
  System.out.println(resultSet.getString(1) + " " +
                     resultSet.getString(2) + " " +
                     resultSet.getString(3));
}
```

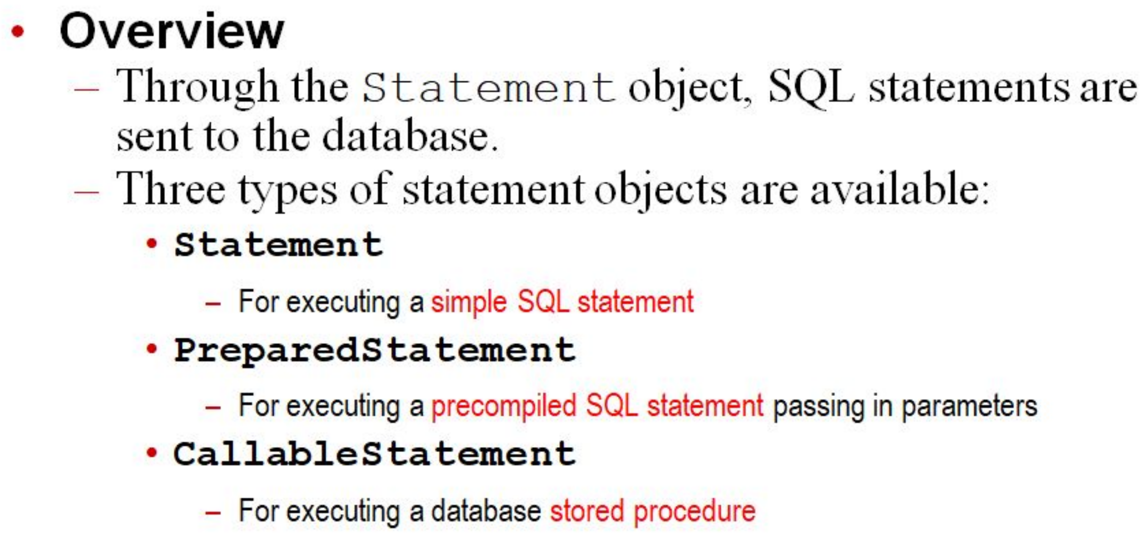
   - First column has index 1, not 0
   - `ResultSet` provides various `getXxx` methods that take a column index *or column name* and returns the data
   - You can also access result meta data (column names, etc.)

7. **Close the Connection**

```
 connection.close();
```

   - Since opening a connection is expensive, postpone this step if additional database operations are expected

Using Statement



- Overview
  - Through the `Statement` object, SQL statements are sent to the database.
  - Three types of statement objects are available:
    - **`Statement`**
      - For executing a simple SQL statement
    - **`PreparedStatement`**
      - For executing a precompiled SQL statement passing in parameters
    - **`CallableStatement`**
      - For executing a database stored procedure

Useful Statement Methods

- **executeQuery**
  - Executes the SQL query and returns the data in a table (ResultSet)
  - The resulting table may be empty but never null

```
ResultSet results =
  statement.executeQuery("SELECT a, b FROM table");
```

- **executeUpdate**
  - Used to execute for INSERT, UPDATE, or DELETE SQL statements
  - The return is the number of rows that were affected in the database
  - Supports Data Definition Language (DDL) statements CREATE TABLE, DROP TABLE and ALTER TABLE

```
int rows =
  statement.executeUpdate("DELETE FROM EMPLOYEES" +
                          "WHERE STATUS=0");
```

Useful Statement Methods (Continued)

- **execute**
  - Generic method for executing stored procedures and prepared statements
  - Rarely used (for multiple return result sets)
  - The statement execution may or may not return a ResultSet (use statement.getResultSet). If the return value is true, two or more result sets were produced
- **getMaxRows/setMaxRows**
  - Determines the maximum number of rows a `ResultSet` may contain
  - Unless explicitly set, the number of rows is unlimited (return value of 0)
- **getQueryTimeout/setQueryTimeout**
  - Specifies the amount of a time a driver will wait for a STATEMENT to complete before throwing a `SQLException`

Prepared Statements - Precompiled Queries

- Idea
  - If you are going to execute similar SQL statements multiple times, using "prepared" (parameterized) statements can be more efficient
  - Create a statement in standard form that is sent to the database for compilation before actually being used
  - Each time you use it, you simply replace some of the marked parameters using the `setXxx` methods
- As PreparedStatement inherits from Statement the corresponding execute methods have no parameters
  - execute()
  - executeQuery()
  - executeUpdate()

Prepared Statement, Example

```
Connection connection =
  DriverManager.getConnection(url, user,
  password);
PreparedStatement statement =
  connection.prepareStatement("UPDATE employees "+
                              "SET salary = ? " +
                              "WHERE id = ?");
int[] newSalaries = getSalaries();
int[] employeeIDs = getIDs();
for(int i=0; i<employeeIDs.length; i++) {
  statement.setInt(1, newSalaries[i]);
  statement.setInt(2, employeeIDs[i]);
  statement.executeUpdate();
}
```

That's All Folks