CS157A: Introduction to Database Management Systems

JDBC
(**J**ava **D**ata**B**ase **C**onnectivity)
Suneuy Kim

JDBC Setting Up

1. Install Java
2. Install Database - MySQL
3. Install JDBC driver
   - JDBC Driver for MySQL (MySQL Connector/J) http://www.mysql.com/products/connector/
   - You need to add the path to the driver in your classpath.

```
    e.g.) C:\Program Files\MySQL\Connector J 5.1.25\mysql-connector-java-5.1.25-bin.jar;
```

4. JDBC Tutorial: http://docs.oracle.com/javase/tutorial/jdbc/
5. JDBC API: java.sql and javax.sql package

Eclipse

[Q] java.lang.ClassNotFoundException: com.mysql.jdbc.Driver in Eclipse

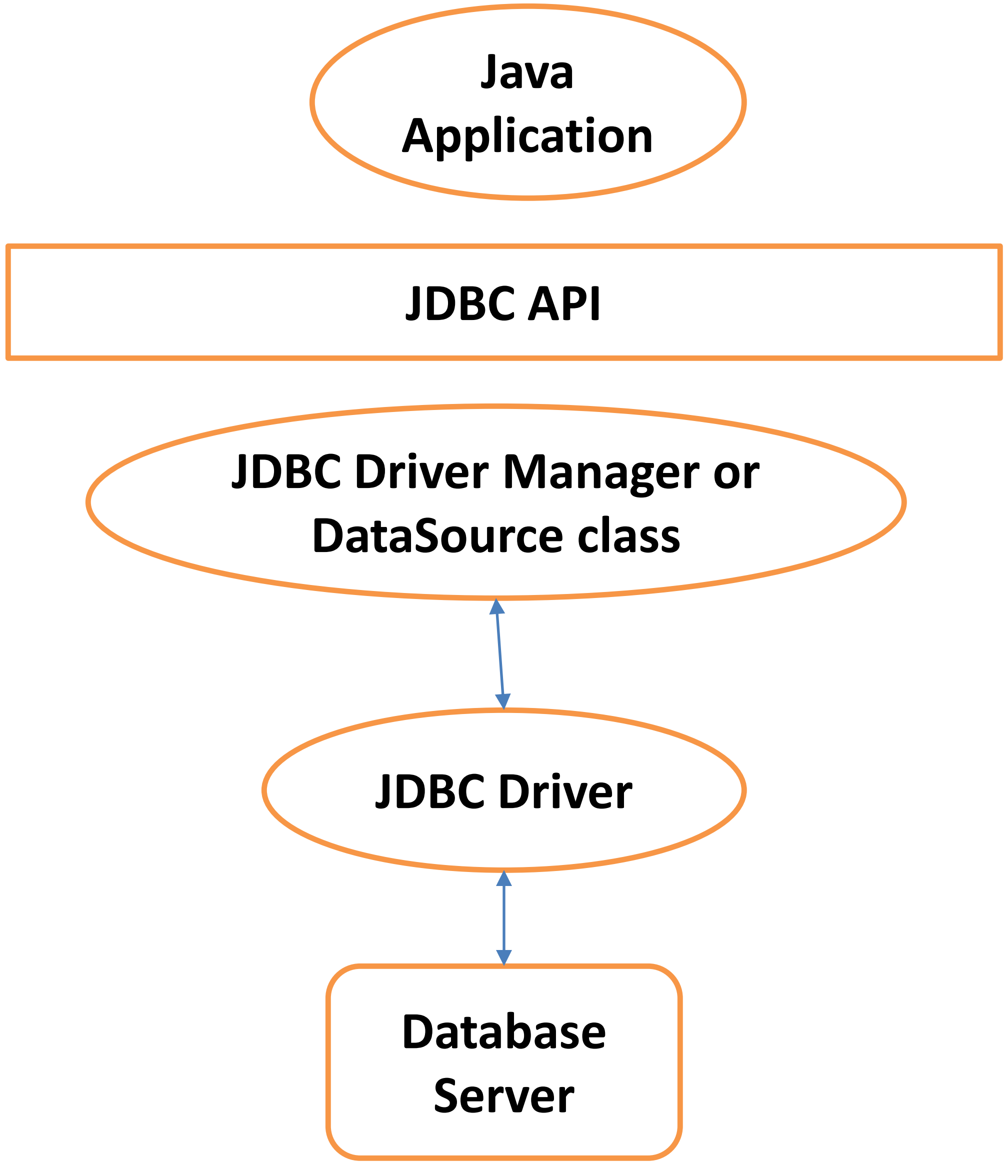
[A]

- Right Click the project -- > build path -- > configure build path
- In Libraries Tab press Add External Jar and Select your jar (e.g. mysql-connector-java-5.1.35-bin.jar)

JDBC

- http://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/
- JDBC API : The industry standard for database-independent connectivity between Java and a SQL database
  - Establish a connection with a database or access any tabular data source
  - Send SQL statements
  - Process the results

JDBC Architecture



Establishing a Connection

Typically, a JDBC application connects to a target data source using one of two classes:

- DriverManager and DataSource
- DataSource is preferred over DriverManager

1) Allows underlying data source to be transparent to your application. With a DataSource you need to know only the JNDI name.
2) Supports connection pool. Connections are managed by the application server (e.g. Apache Tomcat, Oracle WebLogic Server) and can be fetched while at runtime.
3) Helpful for enterprise applications

Connect to MySQL with JDBC Driver using DriverManager

```
// Driver class is automatically loaded
// since JDBC 4.0
Class.forName("com.mysql.jdbc.Driver");

Connection connection  =
            DriverManager.getConnection
("jdbc:mysql://localhost:3306/library",
"root", "default$");
```

Note: For driver manager you need to know all the details (host, port, username, password, driver class) to connect to DB an to get connections.

Source: DriverMangerTester.java

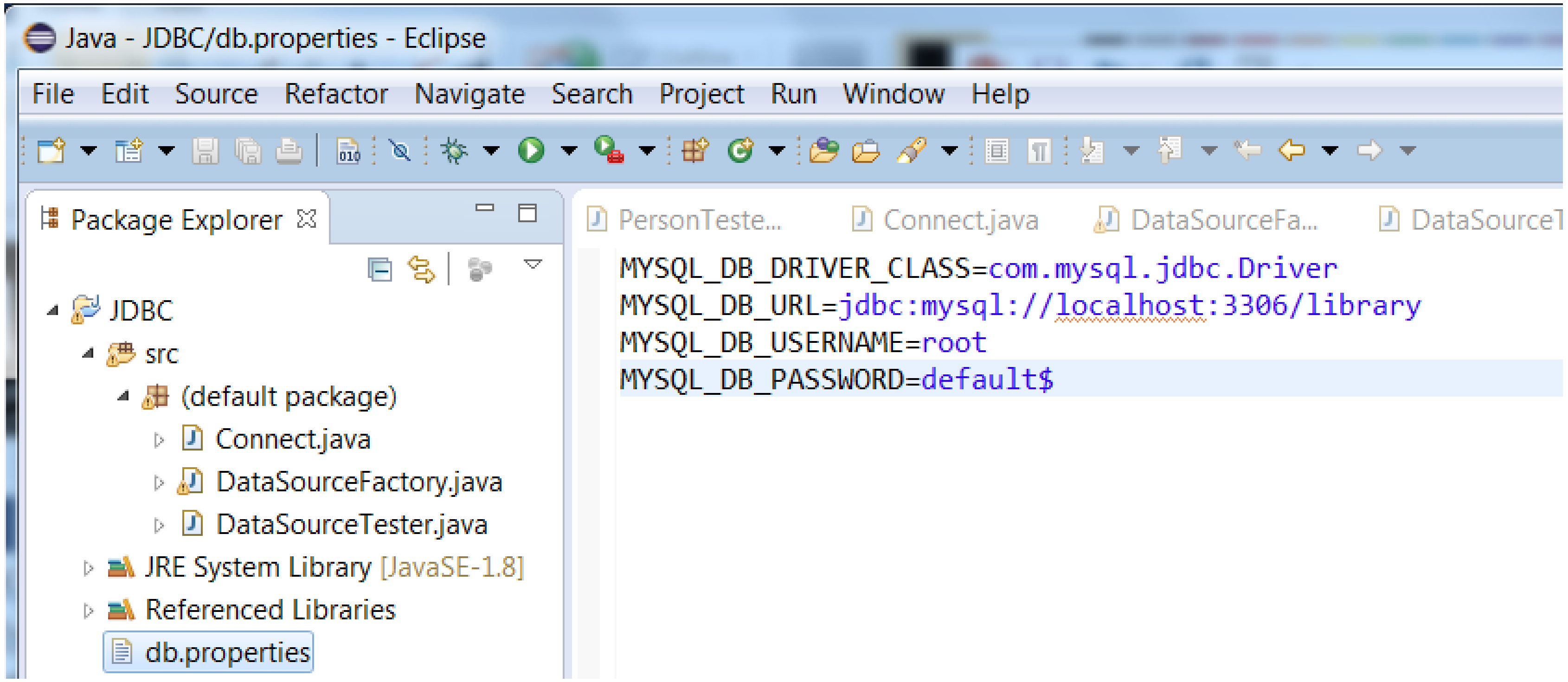
Use of `DriverManager`

- For driver manager you need to know all the details (host, port, username, password, driver class) to connect to DB an to get connections.

Use of `Data Source`

```
import javax.sql.DataSource;
import com.mysql.jdbc.jdbc2.optional.MysqlDataSource;
 . . .
Properties props = new Properties();
FileInputStream fis =
          new FileInputStream("db.properties");
props.load(fis);
DataSource mysqlDS =
          new MysqlDataSource();
mysqlDS.setURL(props.getProperty("MYSQL_DB_URL"));
mysqlDS.setUser(props.getProperty("MYSQL_DB_USERNAME"));
mysqlDS.setPassword(props.getProperty("MYSQL_DB_PASSWORD
"));
Connection  con = ds.getConnection();
```

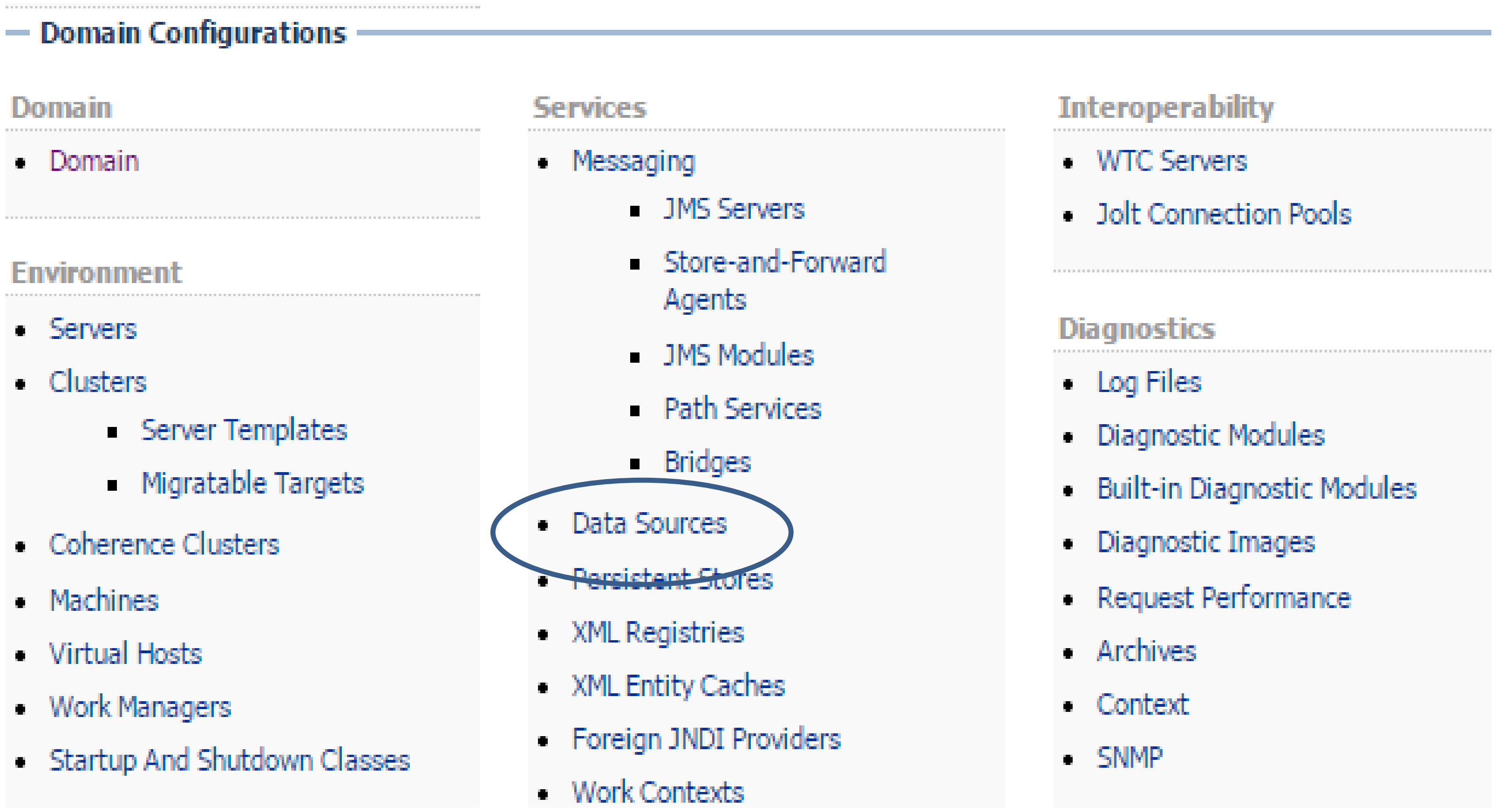
Source: DataSourceTester.java



```
MYSQL_DB_DRIVER_CLASS=com.mysql.jdbc.Driver
MYSQL_DB_URL=jdbc:mysql://localhost:3306/library
MYSQL_DB_USERNAME=root
MYSQL_DB_PASSWORD=default$
```

Externalizing properties in a properties file doesn't change anything about the fact that you need to know them. With an enterprise application, the Data Source is configured in the application server such as Tomcat or WebLogicServer using these properties and an application needs to know the JNDI name of the data source only.

Use of `Data Source` `(Oracle WebLogic Server)`



Domain Configurations

Domain

- Domain

Environment

- Servers
- Clusters
  - Server Templates
  - Migratable Targets
- Coherence Clusters
- Machines
- Virtual Hosts
- Work Managers
- Startup And Shutdown Classes

Services

- Messaging
  - JMS Servers
  - Store-and-Forward Agents
  - JMS Modules
  - Path Services
  - Bridges
- Data Sources
- Persistent Stores
- XML Registries
- XML Entity Caches
- Foreign JNDI Providers
- Work Contexts

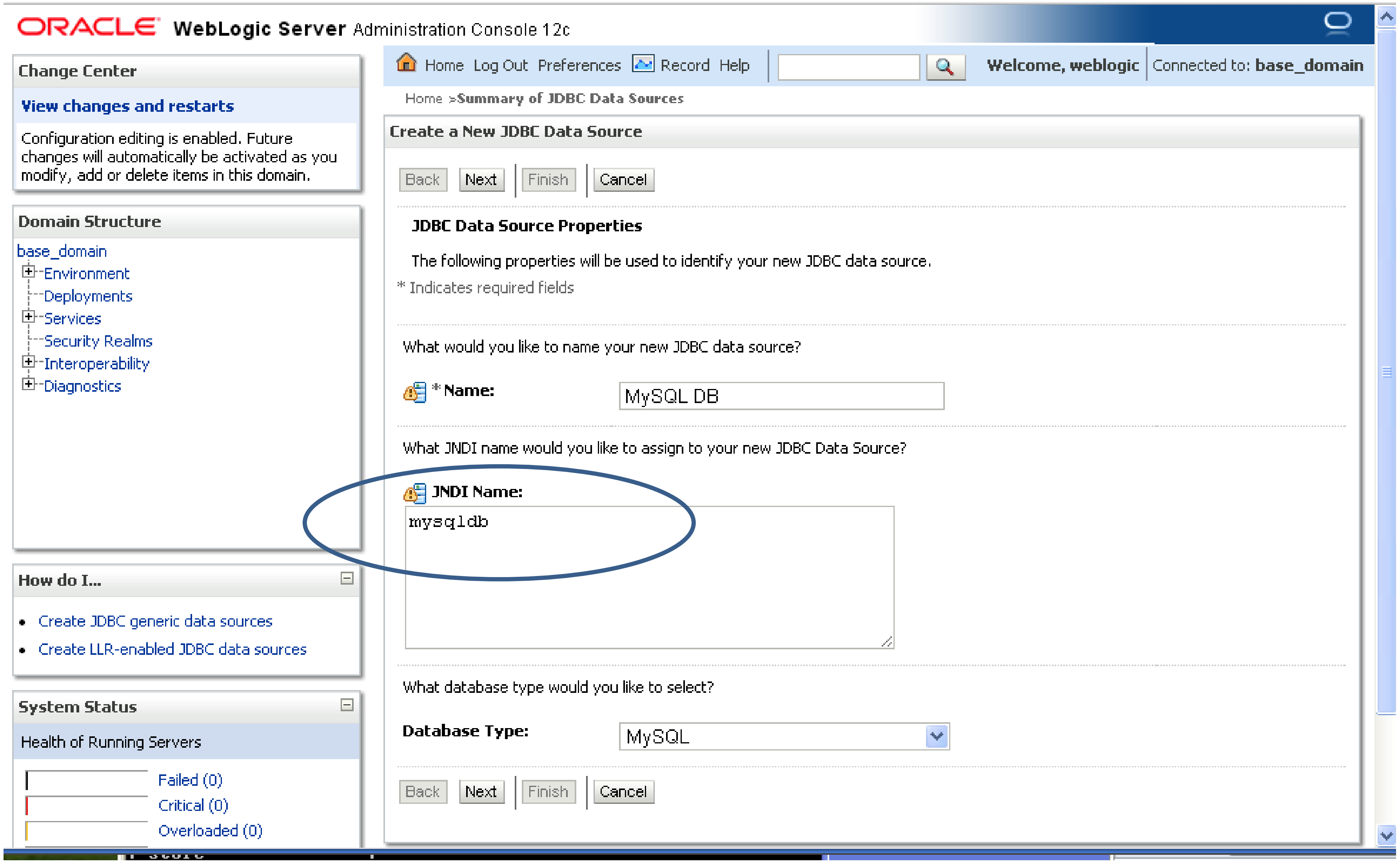
Interoperability

- WTC Servers
- Jolt Connection Pools

Diagnostics

- Log Files
- Diagnostic Modules
- Built-in Diagnostic Modules
- Diagnostic Images
- Request Performance
- Archives
- Context
- SNMP

Out of Scope



ORACLE WebLogic Server Administration Console 12c

Home Log Out Preferences Record Help

Welcome, weblogic | Connected to: base_domain

Home >Summary of JDBC Data Sources

Change Center

View changes and restarts

Configuration editing is enabled. Future changes will automatically be activated as you modify, add or delete items in this domain.

Domain Structure

- base_domain
  - Environment
  - Deployments
  - Services
  - Security Realms
  - Interoperability
  - Diagnostics

How do I...

- Create JDBC generic data sources
- Create LLR-enabled JDBC data sources

System Status

Health of Running Servers

- Failed (0)
- Critical (0)
- Overloaded (0)

Create a New JDBC Data Source

Back Next Finish Cancel

JDBC Data Source Properties

The following properties will be used to identify your new JDBC data source.

* Indicates required fields

What would you like to name your new JDBC data source?

*** Name:** MySQL DB

What JNDI name would you like to assign to your new JDBC Data Source?

JNDI Name:

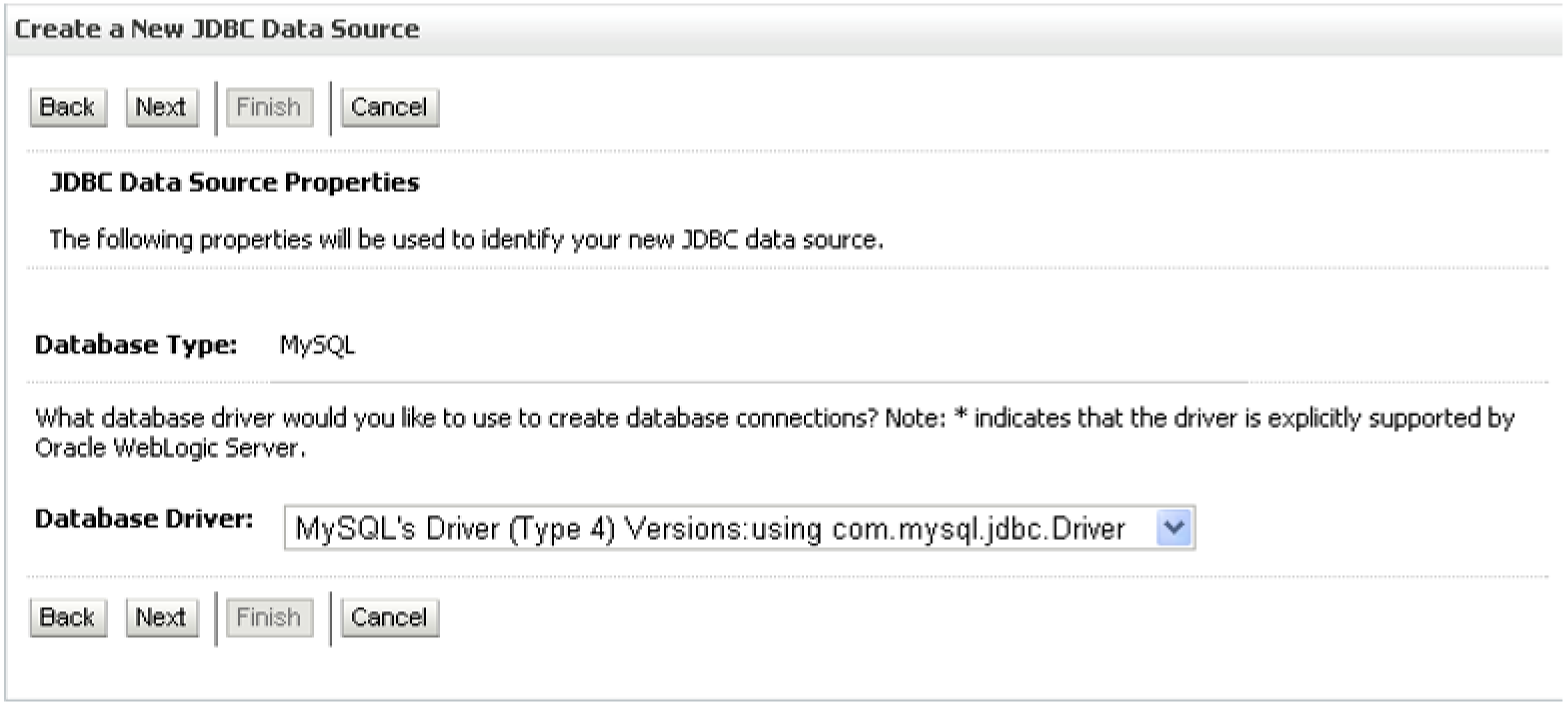
```
mysqldb
```

What database type would you like to select?

Database Type: MySQL

Back Next Finish Cancel

Out of Scope



JDBC Data Source Properties

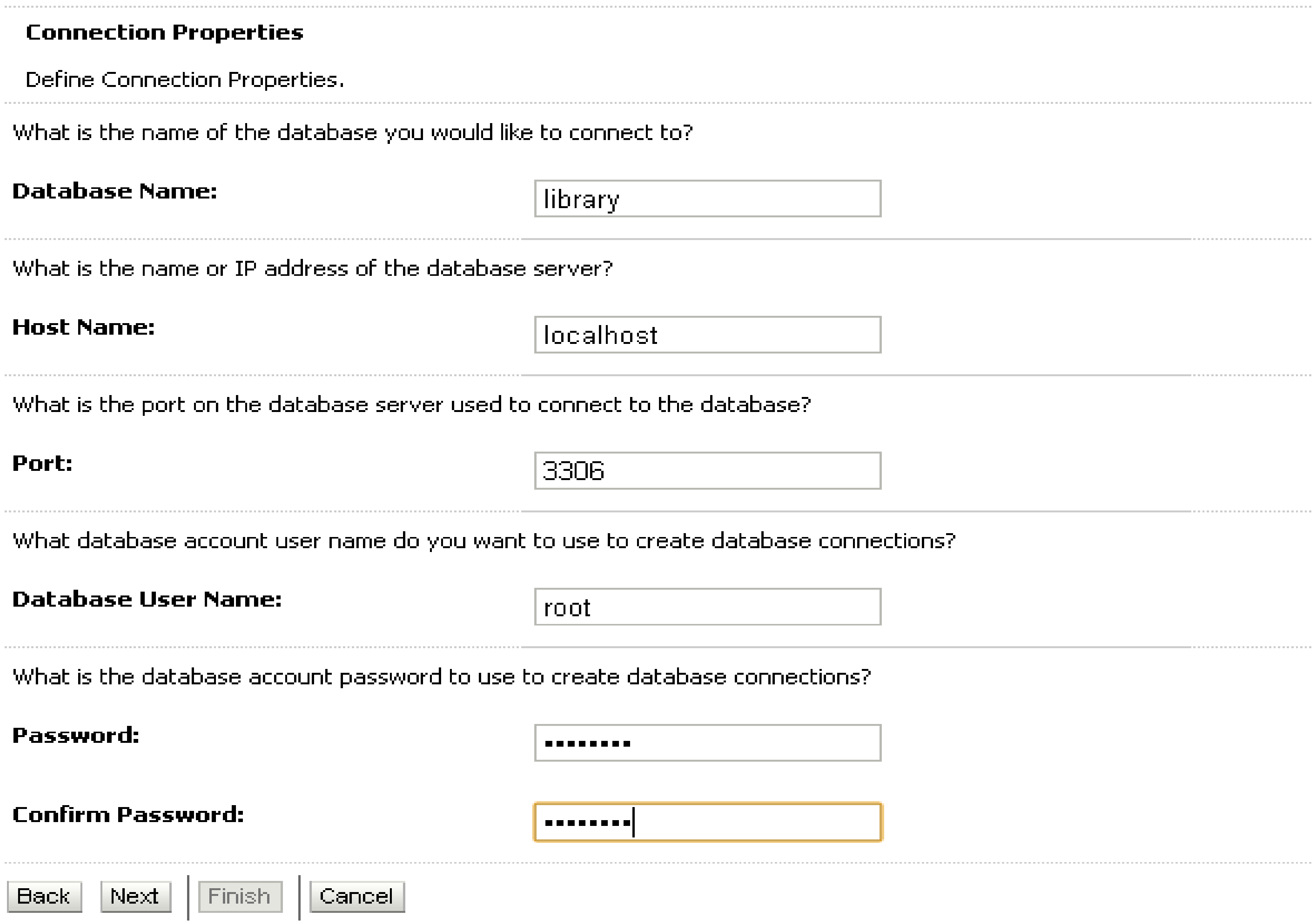
The following properties will be used to identify your new JDBC data source.

Database Type: MySQL

What database driver would you like to use to create database connections? Note: * indicates that the driver is explicitly supported by Oracle WebLogic Server.

Database Driver: MySQL's Driver (Type 4) Versions:using com.mysql.jdbc.Driver

Out of Scope



Connection Properties

Define Connection Properties.

What is the name of the database you would like to connect to?

Database Name:	library

What is the name or IP address of the database server?

Host Name:	localhost

What is the port on the database server used to connect to the database?

Port:	3306

What database account user name do you want to use to create database connections?

Database User Name:	root

What is the database account password to use to create database connections?

Password:	••••••••
Confirm Password:	••••••••

Out of Scope

```
import javax.naming.*;
import javax.sql.DataSource;
public class DatabaseConnection
{ private static DataSource dataSource = null;
  private static Context context = null;
  public static DataSource getDataSource() throws Exception
 { if (dataSource != null) return dataSource;
   if (context == null) context = new InitialContext();

    dataSource = (DataSource) context.lookup("mysqldb");
    return dataSource;
  }
}
. . .
Connection conn =
DatabaseConnection.getDataSource().getConnection();
```

Out of Scope

My examples use the DriverManager class instead of the DataSource because it is easier to use and the examples do not require the features of the DataSource class.

Processing SQL Statements with JDBC

- `Statement`

To submit the SQL statements to the database.

- `ResultSet`

Holds results of SQL statements. It acts as an iterator to allow you to iterate over its data.

- `SQLException`

Handles any errors that occur in a database application.

Creating JDBC Application

- Import the packages

```
e.g.) import java.sql.*
```

- Register the JDBC driver (automatically done since JDBC 4.0)

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

- Open a connection .

```
Connection conn = DriverManager.getConnection()
```

- Create a Statement

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

- Execute a query .

```
statementObject.execute(sq);
```

- Extract data from result set .

Appropriate `ResultSet.getXXX()` method

- Clean up the environment

```
conn.close();
```

Source: JDBCExample.java

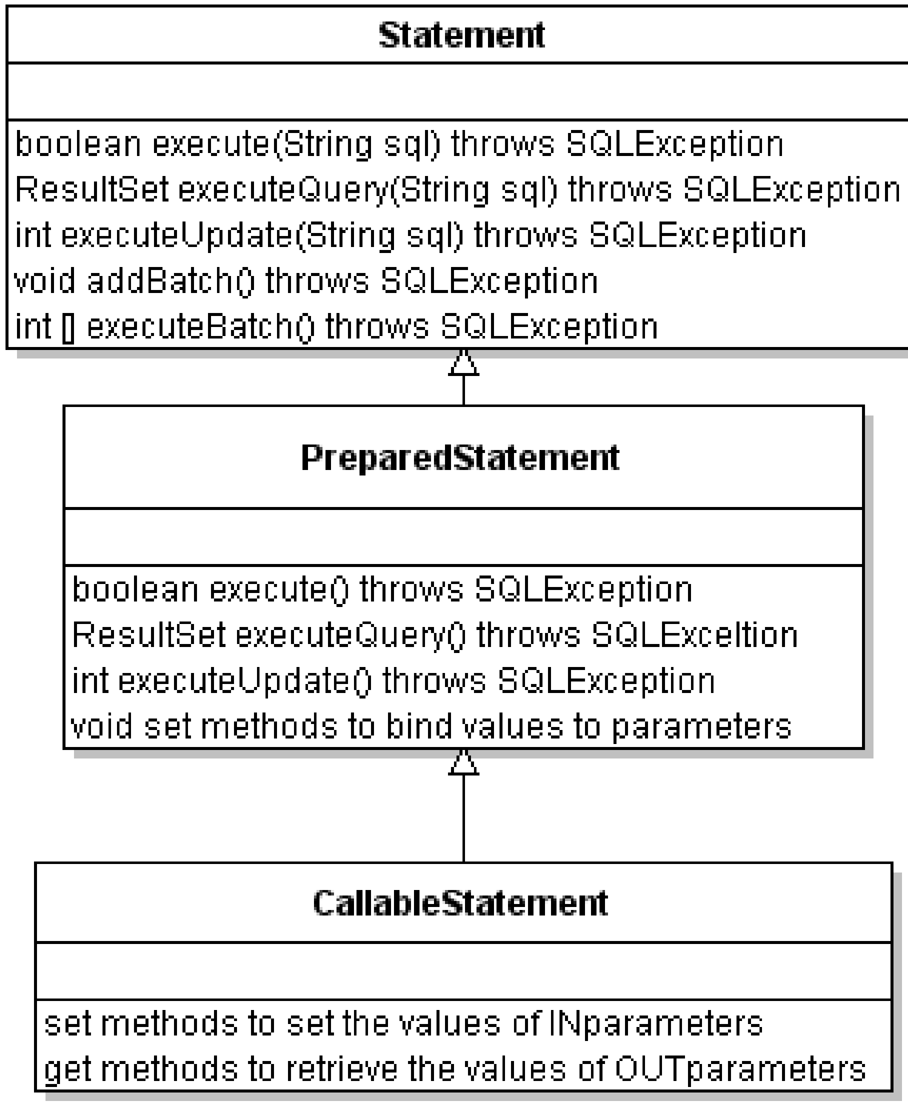
Example Source Codes

- DriverManagerTester.java
- DataSourceTester.java
- JDBCExample.java : Use this as a template to create your JDBC applications.

Statements

- Statement: Used to implement simple SQL statements with no parameters.
- PreparedStatement: Used for precompiling SQL statements that might contain input parameters.
- CallableStatement: Used to execute stored procedures that may contain both input and output parameters.

JDBC Statements


Statement
boolean execute(String sql) throws SQLException
ResultSet executeQuery(String sql) throws SQLException
int executeUpdate(String sql) throws SQLException
void addBatch() throws SQLException
int [] executeBatch() throws SQLException
PreparedStatement
boolean execute() throws SQLException
ResultSet executeQuery() throws SQLExceltion
int executeUpdate() throws SQLException
void set methods to bind values to parameters
CallableStatement
set methods to set the values of INparameters
get methods to retrieve the values of OUTparameters

Statement

- To execute a simple SQL statement with no parameters.

```
Statement stmt = null;
try { stmt = conn.createStatement(); [redacted] }
catch (SQLException e) { … }
finally { stmt.close(); }
```

- Create, insert, update or delete statement: `statementObj.executeUpdate(sql);`
- Select query that returns one ResultSet: `statementObj.executeQuery(sql);`
- Query that might returns multiple ResultSets:
  `statementObj.execute(sql); statementObject.getResultSet();`

Create/Drop Database

- JDBC - Create Database

```
String sql =
     "CREATE DATABASE STUDENTS";
```

- JDBC - Drop Database

```
String sql =
     "DROP DATABASE STUDENTS";
```

Create/Drop Tables

- JDBC - Create Tables

```
String sql = "CREATE TABLE REGISTRATION "
           + "(id INTEGER not NULL, "
           + " first VARCHAR(255), "
           + " last VARCHAR(255), "
           + " age INTEGER, "
           + " PRIMARY KEY ( id ))";
```

- JDBC - Drop Tables

```
String sql = "DROP TABLE REGISTRATION ";
```

Modification

- JDBC - Insert Records

```
String sql="INSERT INTO Registration "
+ "VALUES (100, 'Zara', 'Ali', 18)";
```

- JDBC - Update Records

```
String sql = "UPDATE Registration " +
"SET age = 30 WHERE id in (100, 101)";
```

- JDBC - Delete Records

```
String sql = "DELETE FROM Registration "
+ "WHERE id = 101";
```

Select-From-Where clause

```
String sql =
  "SELECT id, first, last, age
   FROM Registration";
String sql =
 "SELECT id, first, last, age
  FROM Registration" +
" WHERE id >= 101 ";
```

Like/Order by

JDBC - Like Clause

```
sql = "SELECT id, first, last, age
FROM Registration" + " WHERE first
LIKE '%za%' ";
```

JDBC –Order by Clause

```
String sql = "SELECT id, first, last,
age FROM Registration" + " ORDER BY
first ASC";
```

Example: JDBC Statement

- JDBCStatementExample.java

JDBC PreparedStatement

- This extended statement gives you the flexibility of supplying arguments dynamically.
- All parameters in JDBC are represented by the parameter symbol **?**.
- Each parameter marker is referred to by its ordinal position, starting at 1.
- The `setXXX()` methods bind values to the parameters, where represents the Java data type of the value

JDBC PreparedStatement

```
PreparedStatement pstmt = null;
try { String SQL =
   "Update Employees SET age = ? WHERE id = ?";
    pstmt = conn.prepareStatement(SQL);

}
catch (SQLException e) {…}
finally { pstmt.close(); }
```

```
pstmt.setInt(1, 35);
pstmt.setInt(2, 111);
preparedStatementObj.executeUpdate();
Note: execute()/executeQuery()as needed.
```

JDBC PreparedStatement: Example

JDBCPreparedStatementExample.java

JDBC CallableStatement: Steps

1. Prepare the callable statement by using Connection.prepareCall().
2. Register the output parameters (if any exist)
3. Set the input parameters (if any exist)
4. Execute the CallableStatement, and retrieve any result sets or output parameters.

StoredProcedure with IN Parameter

```
DROP PROCEDURE IF EXISTS getFacultyByName;
DELIMITER //
CREATE PROCEDURE getFacultyByName(IN
facultyName VARCHAR(50))
BEGIN
SELECT *
FROM Faculty
WHERE name=facultyName;
END//
DELIMITER ;
```

To call a stored procedure with IN parameter

```
CallableStatement cstmt = null;
try { String sql = "{call getFacultyByName(?)}";
      cstmt = conn.prepareCall(sql);
      cstmt.setString(1,”James Sonnier”);
      cstmt.execute();
      . . . }
catch (SQLException e) { . . . }
finally { . . .}
```

Stored Procedure with OUT parameter

```
DROP PROCEDURE IF EXISTS countByAge;
DELIMITER //
CREATE PROCEDURE countByAge(IN retirementAge INT,
OUT total INT)
BEGIN
SELECT count(*)
INTO total
FROM Faculty
WHERE retirementAge < age;
END//
DELIMITER ;
---------------------------------------------------
CALL countByAge(50, @result);
SELECT @result;
```

To call a stored procedure with OUT parameter

```
cs = conn.prepareCall("{CALL countByAge(?, ?)}");
cs.setInt(1,50);
cs.registerOutParameter(2, Types.INTEGER);
System.out.println(cs.getInt(2)); // index based
System.out.println(cs.getInt("total")); // name based
```

Stored Procedure returning a scalar value

```
DROP PROCEDURE IF EXISTS countByAge2;
DELIMITER //
CREATE PROCEDURE countByAge2(IN retirementAge
INT)
BEGIN
SELECT count(*)
FROM Faculty
WHERE retirementAge < age;
END//
DELIMITER ;
--------------------------------------------
CALL countByAge(50);
```

To call a stored procedure returning a scalar value

```
boolean hasResult = cs.execute();
if (hasResult)
{
  rs = cs.getResultSet();
  System.out.println(rs.getInt(1));
}
```

To get multiple ResultSets from a stored procedure

```
CallableStatement cs = …
cs.execute();
ResultSet rs = cs.getResultSet();
// Process the first ResultSet

cs.getMoreResults();
rs = cs.getResultSet();
// Process the second ResultSet
```

JDBC CallableStatement: Example

- JDBCCallableStatementExample.java
- MultipleResultSets.java

Batch Update (from JDBCStatementExample.java)

```
conn.setAutoCommit(false);

statement = conn.createStatement();
statement.addBatch("INSERT INTO Students " +
"VALUES (495, 'Robert Cliff', 22)");
statement.addBatch("INSERT INTO Students " +
"VALUES (333, 'Toni Smith', 27)");
statement.addBatch("INSERT INTO Students " +
"VALUES (555, 'Robert E.Laskey', 25)");

int [] updateCounts = statement.executeBatch();
conn.commit();
```

`int [] updateCounts = statement.executeBatch();`

- Each executed statement returns a update count indicating how many rows are affected by this statement.
- In previous example, the executeBatch returns an array containing three 1s.
- Use executeBatch if a statement return a update count
  - insert, update and delete: n >= 0
  - create and drop: 0
- Otherwise, executeBatch can’t be used (e.g. select)

ResultSet

- Represents a table of data returned by executing query statement.
- A ResultSet maintains a cursor.
- Initially the cursor is positioned before the first row.
- The navigational methods move the cursor in ResultSet. It returns false if there is no rows in the ResultSet. `while(rs.next()){ }`

ResultSet

```
try
{
Statement statement =
conn.createStatement(
      ResultSet.TYPE_FORWARD_ONLY,
      ResultSet.CONCUR_READ_ONLY);
ResultSet rs =
stmt.executeQuery("SELECT * from User");

}
catch(SQLException ex) { .... }
finally { .... }
```

ResultSet Type

- `ResultSet.TYPE_FORWARD_ONLY`
- `ResultSet.TYPE_SCROLL_INSENSITIVE`
- `ResultSet.TYPE_SCROLL_SENSITIVE`

Notes:

- `FORWARD_ONLY vs. SCROLL`
- `INSENSITIVE vs. SENSITIVE`: the result set is (in) sensitive to changes made after the result set was created.

ResultSet type (MySQL)

`DatabaseMetaData.supportsResultSetType(int)`

returns true if the specified `ResultSet` type is supported and false otherwise.

```
DatabaseMetaData dmd = conn.getMetaData();

dmd.supportsResultSetType(ResultSet.TYPE_FORWARD_ONLY);
// false
dmd.supportsResultSetType(ResultSet.TYPE_SCROLL_INSENSIT
IVE); // true
dmd.supportsResultSetType(ResultSet.TYPE_SCROLL_SENSITIV
E); // false
```

Updatable ResultSet

This option indicates if the ResultSet is updatable or not.

- `ResultSet.CONCUR_READ_ONLY`
- `ResultSet.CONCUR_UPDATABLE:`

`DatabaseMetaData.supportsResultSetConcurrency(int, int)` returns true if the specified concurrency level is supported by the driver and false otherwise.

ResultSet Concurrency (MySQL)

```
DatabaseMetaData dmd = conn.getMetaData();

dmd.supportsResultSetConcurrency(ResultSet.T
YPE_SCROLL_INSENSITIVE,ResultSet.CONCUR_READ
_ONLY); // true

dmd.supportsResultSetConcurrency(ResultSet.T
YPE_SCROLL_INSENSITIVE,ResultSet.CONCUR_UPDA
TABLE); // true
```

ResultSet methods

A ResultSet object maintains a cursor that points to the current row in the result set.

- **Navigational methods:** used to move the cursor around.
- **Get methods:** used to view the data in the **columns** of the current row being pointed by the cursor.
- **Update methods:** used to update the data in the columns of the current row.

Navigation Methods

`beforeFirst() vs first()`

- `beforeFirst()` moves the cursor to the front of this ResultSet object, just before the first row. A subsequent next() call makes the first row the current row.

- `first()` moves the cursor to the first row in this ResultSet object. The first row becomes the current row. A subsequent next() makes the next row (second row) the current row.

Navigation Methods

`beforeFirst()` vs `first()`

Suppose rows A, B, C are in the ResultSet.

```
// rs.first()
while (rs.next())
{ // get and print the rows}
```

Without rs.first(): A, B, C

With rs.first(): B, C

Navigation Methods `next()` and `previous()`

- `next()`

- Moves the cursor forward one row from its current position.
- A ResultSet cursor is initially positioned before the first row;
- the first call to the method next makes the first row the current row;
- When a call to the next method returns false, the cursor is positioned after the last row. (No hasNext() unlike Java)

- `previous()`

- Moves the cursor to the previous row in this ResultSet object.
- When a call to the previous method returns false, the cursor is positioned before the first row. (No hasPrevious() unlike Java)

ResultSet: Update

(1) To update the current row.

```
   while (rs.next())
   {
    int id = rs.getInt("id");
    String name = rs.getString("name");
    int age = rs.getInt("age");
    System.out.println("ID:" + id + " 
Name:" + name + " Age:" + age);
    rs.updateInt("age", age * 10); // 
update the row in ResultSet
    rs.updateRow(); // update the row in 
the database
    }
```

ResultSet: Update

(2) To update a row at an absolute position:

```
rs.absolute(2); // moves the cursor
to the 2nd row of rs
rs.updateInt("id", 890);
rs.updateString("name", "Smith");
rs.updateInt("age", 43);
```

To cancel update

- `cancelRowUpdates()` cancels the updates made to the current row in this ResultSet object.

```
rs.updateInt("age", age * 10);
// Updating the ResultSet
rs.cancelRowUpdates();
rs.updateRow();
```

Note: Should be called before rs.updateRow() to be effective.

ResultSet:Insert

Use a staging tuple

```
rs.moveToInsertRow();
rs.updateInt("id", 890);
rs.updateString("name", "Smith");
rs.updateInt("age", 43);

rs.insertRow(); // into this ResultSet and
into the database
rs.beforeFirst(); //move the cursor to a
desired position.
```

ResultSet: Delete

```
rs.first(); // if the first row is
the target
rs.deleteRow(); //Deletes the current
row from this ResultSet and also from
the underlying database.
```

Example:ResultSet

- JDBCResultSet.java

SQLExceptions

A SQLException object contains

A description of the error	**SQLException.getMessage**
A SQLState code	SQLException.getSQLState
An error code	SQLException.getErrorCode (vender specific error code)
A cause	SQLException.getCause
A reference to any *chained* exceptions	SQLException.getNextException

Example: SQLException Handling

- ExceptionExample.java
- **Mapping MySQL Error Numbers to JDBC SQLState Codes**

http://dev.mysql.com/doc/connector-j/en/connector-j-reference-error-sqlstates.html

SQLStates

- SQL State (SQLSTATE) Error Codes are defined by the ISO/ANSI and Open Group (X/Open) SQL Standards.
- List of SQLStates (SQLStates.txt)

  A complete list of the SQLSTATE error codes can be found in the documentations of the ISO/ANSI and Open Group (X/Open) SQL Standards.

Some popular JDBC drivers

RDBMS	JDBC Driver Name
MySQL	Driver Name com.mysql.jdbc.Driver Database URL format: jdbc:mysql//hostname/databaseName
Oracle	Driver Name: oracle.jdbc.driver.OracleDriver Database URL format: jdbc:oracle:thin@hostname:portnumber:databaseName
DB2	Driver Name: COM.ibm.db2.jdbc.net.DB2Driver Database URL format: jdbc:db2:hostname:portnumber/databaseName
Access	Driver Name: com.jdbc.odbc.JdbcOdbcDriver Database URL format: jdbc:odbc:databaseName