**JDBC**

I**mportant Packages**:

javax.sql

java.sql

**Key classes :**  **Development Process:**

DriverManager 1. Get a connection to database

Connection 2. Create a Statement object

Statement 3. Execute SQL query

ResultSet 4. Process Result Set

DataSource

* 1. Get a connection to a database
* jdbc:<driver protocol>:<driver connection details>

ex. jdbc:mysql://localhost:3306/demodb

**import java.sql.\***

**…**

**String dbUrl = “jdbc:mysql://localhost:3306/demo”;**

**String user = “student”;**

**String password = “student”;**

**Connection myConn = DriverManager.getConnection(dbUrl,user,password);**

2. Create a Statement object

The Statement object depends on connection

- It will be used later to execute SQL query

**Statement myStmt = myConn.createStatement;**

3. Execute SQL query

**ResultSet myRs = myStmt.executeQuery(‘select \* from employees’);**

**4.** Process Result Set

* Result Set is initially places before first row
* Method: ResultSet.next()

- moves forward one row

- returns true if there are more rows to process

* Looping through a result set:
* Collection of methods for reading data

ResultSet myRs = myStmt.executeQeury(“select \* from employees”);

**while(myRs.next()){**

**// read data from each row**

**}**

* getXXX(columnName)
* getXXX(columnIndex) one-based

…

ResultSet myRs = myStamt.executeQuery(“select \* from employess”);

**while(myRs.next()){**

**System.out.println(myRs.getString(“last\_name”));**

**System.out.println(myRs.getString(“first\_name”));**

**}**

**Database Setup**

Before run any java code must follow two steps:

* Step1: install MySQL
* Step2: Create New User: student

**Step 1: Install MySQL**

1. Goto: <http://dev.mysql.com/downloads>
2. Click on: MySQL Community Server
3. Download version for your Operating system like mac or win(choose web installer or offline - I chose web)
4. no need to sigh up - no thanks, just start my download
5. in installer - choose developer default - next
6. choose MySql Workbench - Execute
7. default selection - execution - download the default required software - Next
8. In Type and Network - remain default settings -Next
9. In Accounts and Roles - make MySQL Root password - 12 char -strong - Next
10. In Windows Service - remain default - Next
11. In Apply Server Configuration - Execute
12. Finish- check - next - execute - installation complete - finish
13. Start mysql workbench for check - give root password - access to default tables and schemas
14. right click on table - Select Rows -Limit 1000 - Congratulation you have installed successfully!!

**Step 2: Create New Database User**

1. Open mysql workbench - double click on root user - type root password
2. should see Schemas tab - goto Management/Administration tab
3. click on Users and Privileges - click on Add Account
4. Fill details - Login Name: student; Limit to Hosts Matching: localhost; Password: student; Confirm Password: student;
5. click on the Administrative Roles tab - make sure DBA is selected - Apply
6. now we have new user created student
7. to verify - close Local instance 3306 near home icon
8. click on MySQL Connection + sign - fill the details - Connection Name: student; Username: student - Test Connection - enter the password student
9. if succeed - click on OK. - Congratulation you successfully added new user student!!!!

Download source Code:

<http://www.luv2code.com/downloads/udemy-jdbc/jdbc-source-code-v2.zip>

**Setting Up Your Development Environment**

* **System Requirements -** need to have MySQL Database installed

- SQL GUI tool - MySQL Workbench

* **Loading sample database table -**

- open mysql workbench - goto File -> Open SQL Script… -> downloaded code :sql folder->table-setup.sql - select all - run -> refresh database schema - see schema name : demo

* **installing JDBC Driver -**

- Download jdbc driver: google - download mysql jdbc driver -> goto first link -> select platform independent - > download zip - no thanks, just start my download -> extract zip - mysql-connector-java-8.0.18,jar

* **Testing Database Connection**

- Setup eclipse project : open eclipse - > create java project: jdbc-test -> paste mysql-connector-java-8.0.18,jar to root folder - > add jar in build path -> paste class JdbcTest.java - Run - See the message “Database connection successful!”

**Getting rid of MySQL SSL Warning Message**

**JDBC and MySQL: Getting rid of the dreaded message**

*WARN: Establishing SSL connection without server’s identity verification is not recommended*

---

When connecting to a MySQL database, you may encounter this scary warning message.

WARN: Establishing SSL connection without server’s identity verification is not recommended

Your app will continue to work fine … it is just the MySQL database yelling at you.

**Solution**

To get rid of the warning message, append **?useSSL=false** to the end of your database connection string.

For example,

Replace – jdbc:mysql://localhost:3306/demo

With – jdbc:mysql://localhost:3306/demo**?useSSL=false**

Note that I appended **?useSSL=false** to the end.

That will get rid of the pesky message … whew!

**Insert Data**

String dbUrl = “jdbc:mysql://localhost:3306/demo”;

String user = “student”;

String pass = “student”;

Connection myConn = DriverManager.getConnection(dbUrl, user, pass):

Statement myStmt = myConn.createStatment();

**int rowAffected = myStmt.executeUpdate(**

**“insert into employees ”+**

**“(last\_name, first\_name, email, department, salary) ”+**

**“values ”+**

**“(‘Khare’, ‘Raj’, ‘1.raj.khare@gmail.com’, ‘IT’, ‘52000.00’)”);**

**Note:** we can use executeUpdate() for - insert , update, delete

**Update Data**

String dbUrl = “jdbc:mysql://localhost:3306/demo”;

String user = “student”;

String pass = “student”;

Connection myConn = DriverManager.getConnection(dbUrl, user, pass):

Statement myStmt = myConn.createStatment();

**int rowAffected = myStmt.executeUpdate(**

**“update employees ”+**

**“set email=‘raj.khare.jave@gmail.com’ ”+**

**“where last\_name=‘Khare’ and first\_name=‘Raj’”);**

**Delete Data**

String dbUrl = “jdbc:mysql://localhost:3306/demo”;

String user = “student”;

String pass = “student”;

Connection myConn = DriverManager.getConnection(dbUrl, user, pass):

Statement myStmt = myConn.createStatment();

**int rowAffected = myStmt.executeUpdate(**

**“delete from employees ”+**

**“where last\_name=‘Khare’ and first\_name=‘Raj’”);**

**Prepared Statements**

* **What are Prepared Statements**
* **Create a Prepared Statements**
* **Setting Parameter Values**
* **Execute a Prepared Statement**
* **Reusing a Prepared Statement**

**What are Prepared Statements**:

* A Prepared Statement is precompiled SQL statement
* Prepared Statement provides the following benefits

- Makes it easier to set SQL parameter values

- Prevent against SQL injection attack

- May improve application performance

* + SQL statement is precompiled

**Using Prepared Statements**

* **Instead of hard coding your SQL values**

**select \* from employees**

**where salary > 8000 and department=‘Legal’**

* **Set parameter placeholders**

**- Use a question mark for placeholder: ?**

**select \* from employees**

**where salary > ? and department=?**

**Create the prepared statement:**

**PreparedStatement myStmt =**

**myConn.prepareStatement(“select \* from employees”+**

**” where salary > ? and department=?”);**

**myStmt.setDouble(1, 8000);**

**myStmt.setString(2, “Legal”);**

**//now execute the query**

**ResultSet myRs = myStmt.executeQuery();**

* **Can also use prepared statement for**

**- Insert, update and deletes**

**PreparedStatement myStmt =**

**myConn.prepareStatement(“delete from employees” +**

**” where salary > ? and department=?”);**

**//set params**

**myStmt.setDouble(1, 8000);**

**myStmt.setString(2, “Legal”);**

**//execute statement**

**int rowAffected = myStmt.executeUpdate();**

**Stored Procedures**

* **What are Stored Procedures**
* **Using Callable Statements**
* **Call Stored Procedures that takes parameters**

**- IN parameters**

**- INOUT parameters**

**- OUT parameters**

**- Return a result set**

**What are Stored Procedures:**

* **A group of SQL statements that perform a particular task**
* **Normally created by the DBA**
* **Created in a SQL language supported by the native database**

**(**Each database has its own language, for e.g MySql - Stored procedure language, Oracle - PL/SQL and Microsoft SQL Server - Transact-SQL **)**

* **Can have the combination of input, output, and input/output parameters**

**Using Callable Statements:**

* **To call stored procedure from Java**

**- The JDBC API provides the** CallableStatement

**CallableStatement myCall =**

**myConn.prepareCall(“{call some\_stored\_proc()}”);**

**. . .**

**myCall.execute();**

**Parameters**

* **The JDBC API supports different parameters**

**- IN (default)**

**- INOUT**

**- OUT**

* **The stored procedures can also return result sets**
* **Following code example will show how to register parameter types and values**

**IN Parameters**

* **Our DBA created a stored procedure**

**- Developed for MySQL**

**PROCEDURE ‘increase\_salaries\_for\_department’(**

**IN the\_department VARCHAR(64), IN increase\_amount DECIMAL(10,2))**

**BEGIN**

**Update employess SET salary= salary + increase\_amount**

**WHERE department=the\_department;**

**END**

* **Increase salary for everyone in department** 😀

**- First param is department name**

**- Second param is increase amount**

**IN Parameters - Java Coding**

* **Prepare the callable statement**

**CallableStatement myCall =**

**myConn.prepareCall(“{call increase\_salaries\_for\_department(?, ?)}”);**

**myStmt.setString(1, ”Engineering”);**

**myStmt.setDouble(2, 10000);**

**// now execute the statement**

**myStmt.execute();**

**INOUT Parameters**

* **Our DBA Created the stored procedure**

- **Developed for MySQL**

**PROCEDURE ‘greet\_the\_department’(INOUT department VARCHAR(64))**

**BEGIN**

**SET department = concat(‘Hello to the awesome’, department, ‘team!’ );**

**END**

* **Verify simple INOUT example**

**- Param is the department name**

**- Update the param with**

* **Hello to the awesome <department> team!**

**INOUT Parameters - Java Coding**

* **Prepare the callable statement**

**myStmt.registerOutParameter(1, Types.VARCHAR)//use this for INOUT**

**myStmt.setString(1, ”Engineering”);**

**// Call stored procedure**

**myStmt.execute();**

**//Get the value from INOUT parameter**

**String theResult = myStmt.getString(1);**

**System.out.println(“The Result: ”+ theResult);**

**CallableStatement myCall =**

**myConn.prepareCall(“{call greet\_the\_department(?)}”);**

**Output: Hello to the awesome Engineering team!**

**OUT Parameters**

* **Our DBA Created the stored procedure**

- **Developed for MySQL**

**PROCEDURE `get\_count\_for\_department` (IN the\_department VARCHAR(64), OUT the\_count INT)**

**BEGIN**

**SELECT COUNT(\*)INTO the\_count FROM employess**

**WHERE department=the\_department;**

**END**

* **Verify simple OUT example**

**- First param is the department name**

**- Second param is the output for the count**

**OUT Parameters - Java Coding**

* **Prepare the callable statement**

**myStmt.setString(1, ”Engineering”);**

**myStmt.registerOutParameter(2, Types.INTEGER)**

**// Call stored procedure**

**myStmt.execute();**

**//Get the value from OUT parameter**

**int theCount = myStmt.getInt(2);**

**System.out.println(“The Count: ”+ theCount);**

**CallableStatement myCall =**

**myConn.prepareCall(**

**“{call get\_count\_for\_department (?,?)}”);**

**Stored Procedure – Return Result Set**

* **Our DBA Created the stored procedure**

- **Developed for MySQL**

**PROCEDURE `get\_employees\_for\_department`(**

**IN the\_department VARCHAR(64))**

**BEGIN**

**SELECT \* FROM employess**

**WHERE department=the\_department;**

**END**

* **Stored procedure returns a result set / cursor**

**- First param is the department name**

**Result Set - Java Coding**

**Prepare the callable**

**CallableStatement myCall =**

**myConn.prepareCall(**

**“{call get\_employees\_for\_department (?)}”);**

**myStmt.setString(1, ”Engineering”);**

**// Call stored procedure**

**myStmt.execute();**

**//Get the result set**

**myRs = myStmt.getResultSet();**

**// Display the result set**

**...**

**Transactions**

* **What are the Transactions?**
* **Developing Transactions**

**What are the Transactions?**

* **A transaction is a unit of work**
* **One or more SQL statement executed together**

**- Either all of the statements are executed - Commit**

**- Or none of the statements are executed – Rollback**

Start Transaction

SQL INSERT

SQL UPDATE

SQL DELETE

SQL …

OK TO SAVE?

**Yes**

**Commit**

**No**

**Rollback**

**Database**

**JDBC Transactions**

* **By default the database connection is auto-commit**

**- Need to explicitly turn off auto-commit**

**myConn.setAutoCommit(false);**

* **Developers controls commit or rollback**

**myConn.commit();**

**// or**

**myConn.rollback();**

**For e.g:**

**// start transaction**

**myConn.setAutoCommit(false);**

**// perform multiple SQL statements(insert, update, delete)**

**//. . .**

**boolean ok = askUserIfOkToSave();**

**if(ok){**

**// store in database**

**myConn.commit();**

**} else{**

**// discard**

**myConn.rollback();**

**}**

**Get Metadata Instance**

* **Retrieve the metadata instance**

**DatabaseMetaData databaseMetaData = myConn.getMetaData();**

* **DatabaseMetaData methods**

**- getDatabaseProductName()**

**-** **getDatabaseProductVersion();**

**-** **getDriverName();**

**-** **etc…**

* See JavaDoc for details

- Google: **jdbc databasemetadata**

**Database Schema**

* **Use DatabaseMetaData to get db schema**
* **List of tables**
* **List of column names**

**Get ResultSet Metadata**

* **Retrieve the metadata instance**

**ResultSetMetaData myRsMetaData = myRs.getMetaData();**

* **Sample of ResultSetMetaData methods**

|  |  |  |
| --- | --- | --- |
| **getColumnName()** | **getColumnType()** | **getColumnTypeName()** |
| **getPrecision()** | **getScale()** | **getColumnCount()** |
| **isAutoIncrement()** | **isNullable()** | **isCurrencty()** |
| **. . .** | **. . .** | **. . .** |

* See JavaDoc for details
* Google: **javadoc** **jdbc resultset metadata**

**What is a BLOB?**

* A **BLOB**(binary large object) is a collection of binary data stored as a single entity in a database.
* BLOBs are typically documents , images, audio, or other binary objects.
* Note: Database supports for BLOBs in not universal.

**Create BLOB Column**

* **When creating a table in MySQL**

**- Add a column with BLOB datatype**

**CREATE TABLE ‘employees’(**

**‘id’ int(11) NOT NULL AUTO INCREMENT,**

**‘last\_name’ varchar(64)DEFAULT NULL,**

**‘first\_name’ varchar(64) DEFAULT NULL,**

**‘email’ varchar(64) DEFAULT NULL,**

**‘department’ varchar(64) DEFAULT NULL,**

**‘salary’ DECIMAL(10,2) DEFAULT NULL,**

**‘resume’ BLOB,**

**PRIMARY KEY (‘id’)**

**)**

**Writing BLOBs**

* **Add a resume for an employee**

**- Read local PDF file: sample\_resume.pdf**

**- Update database with the binary data**

//prepare statement

**String sql = “update employees set resume=?”**

**+ “ where email=’raj.khare@foo.com’”;**

**PreparedStatment myStmt = myConn.prepareStatment(sql);**

//set parameter for resume file name

**File theFile = new File(“sample\_resume.pdf”);**

**FileInputStream input = new FileInputStream(theFile);**

**myStmt.setBinaryStream(1, input);**

//Execute statement

**myStmt.executeUpdate();**

/

**Reading BLOBs**

* **Read BLOB from DB and write to local file**

**Statement myStmt = myConn.createStatement();**

**String sql = “select resume from employees ”**

**+ “ where email=’raj.khare@foo.com’”;**

//Execute query

**myRs = myStmt.executeQuery(sql);**

//set up a handle to output file

**File theFile = new File(“resume\_from\_db.pdf”);**

**FileOutputStream output = new FileOutputStream(theFile);**

//read BLOB and store in output file

**If(myRs.next())**

**InputStream input = myRs.getBinaryStream(“resume”);**

**byte[] buffer = new byte[1024];**

**while(input.read(buffer) > 0){**

**output.write(buffer);**

**}**

/

**What is a CLOB?**

* A **CLOB**(character large object) is a collection of character data stored as a single entity in a database.
* CLOBs are typically used to store large text document (plain text or XML).
* Note: Database supports for CLOBs in not universal.

**Create CLOB Column**

* **When creating a table in MySQL**

**- Add a column with LONGTEXT datatype**

**- Hold a maximum length of 4GB of characters**

**CREATE TABLE ‘employees’(**

**‘id’ int(11) NOT NULL AUTO INCREMENT,**

**‘last\_name’ varchar(64)DEFAULT NULL,**

**‘first\_name’ varchar(64) DEFAULT NULL,**

**‘email’ varchar(64) DEFAULT NULL,**

**‘department’ varchar(64) DEFAULT NULL,**

**‘salary’ DECIMAL(10,2) DEFAULT NULL,**

**‘resume’ LONGTEXT,**

**PRIMARY KEY (‘id’)**

**)**

**Writing CLOBs**

* **Add a resume for an employee**

**- Read local text file: sample\_resume.text**

**- Update database with the text data**

//prepare statement

**String sql = “update employees set resume=?”**

**+ “ where email=’raj.khare@foo.com’”;**

**PreparedStatment myStmt = myConn.prepareStatment(sql);**

//set parameter for resume file name

**File theFile = new File(“sample\_resume.txt”);**

**FileInputStream input = new FileInputStream(theFile);**

**myStmt.setCharacterStream(1, input);**

//Execute statement

**myStmt.executeUpdate();**

/

**Reading CLOBs**

* **Read CLOB from DB and write to local file**

**Statement myStmt = myConn.createStatement();**

**String sql = “select resume from employees ”**

**+ “ where email=’raj.khare@foo.com’”;**

//Execute query

**myRs = myStmt.executeQuery(sql);**

//set up a handle to output file

**File theFile = new File(“resume\_from\_db.txt”);**

**FileWriter output = new FilwWriter(theFile);**

//read BLOB and store in output file

**If(myRs.next())**

**Reader input = myRs.getCharacterStream(“resume”);**

**int theChar;**

**while((theChar = input.read())> 0){**

**output.write(theChar)**

**}**

**}**

/