Introduction to Web Engineering SENG2050/6050

JDBC

Overview

- Java DataBase Connectivity – JDBC
  - Overview
  - Connecting to a Database
  - Executing Queries
  - Embedding SQL Queries
  - Prepared Statements
  - Result Sets

Java DataBase Connectivity (JDBC)

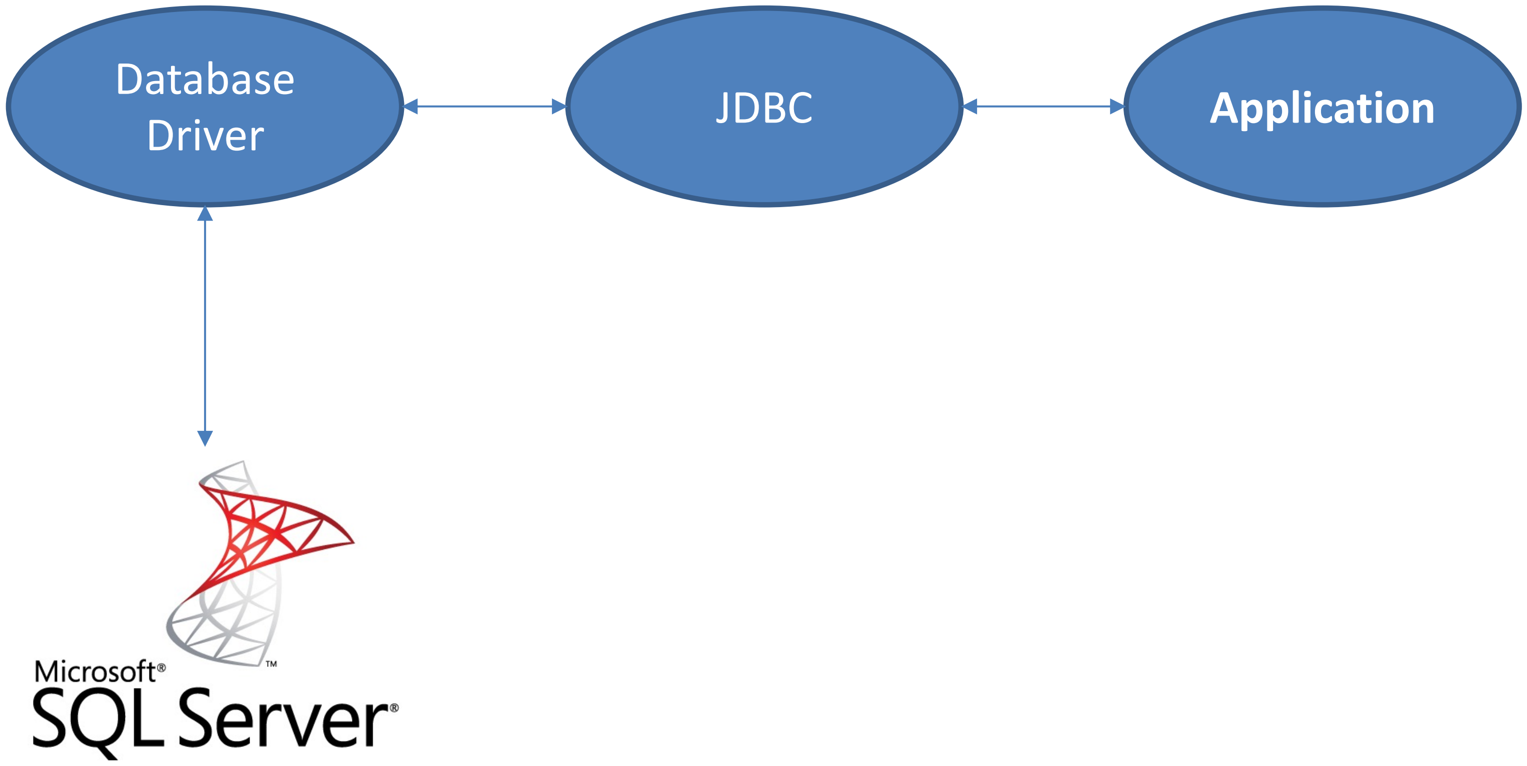
- Provides mechanisms to:
  - Configure a database
  - Connect to a database
  - Construct and execute queries
  - Process results
  - Disconnect from the database
- Can interact with any SQL database
  - As long as it has a JDBC driver

JDBC

- Provides an Application Programming Interface (API) that allows Java programs to connect to a database
- JDBC helps to make it possible to write a single Java application or component that can access a range of different Relational DataBase Management Systems (RDBMS)
  - Create 1 codebase independent of the underlying database

Microsoft® SQL Server™ ® MySQL® PostgreSQL

JDBC



JDBC

- Can be used anywhere in your Java code:
  - Java Beans
  - Java Applications
  - Java Servlets
  - JSP (avoid this, remember our separation of concerns)
- import java.sql.*;

JDBC

- The RDBMS is another server (separate from Tomcat)
  - Even when it’s on the same computer
  - We need to connect to the database
    - Typically a time consuming operation
- Requires:
  - A database driver
    - This helps JDBC and our code to be reusable regardless of the database server used
      - Not 100% guaranteed, your queries might still be database specific, try to avoid this is possible
  - The URL of the database
  - A user ID and password (registered with the database)

JDBC

- In a standard Java application we may need to manage every aspect of the database communication:
  - Connections
  - Authentication
  - The database driver
  - The database url

JDBC – Tomcat

- In a web application hosted by a web server or servlet container (Tomcat) we only need to give Tomcat:
  1. The details of our database
     - apache-tomcat/conf/context.xml
       - For all webapps to use
       - Useful to allow the database connectivity to change depending on the server
     - apache-tomcat/webapps/[APPNAME]/META-INF/context.xml
       - Only for [APPNAME] webapp to use
       - Useful if only [APPNAME] should access the database
       - Useful when you submit me an assignment so I don't have to edit my server's config
  2. The driver
     - MS SQL – https://docs.microsoft.com/en-us/sql/connect/jdbc/microsoft-jdbc-driver-for-sql-server
     - Place "mssql-jdbc-8.4.1.jreX" in:
       - apache-tomcat/lib/
         - » For all webapps to use
       - apache-tomcat/webapps/[APPNAME]/WEB-INF/lib/
         - » For [APPNAME] to use
  3. Some code to run
- Tomcat can manage the rest

X is the Java version
(Use closest to installed)

JDBC – Tomcat

- Allowing Tomcat to manage our database communication is important because:
  - Creating connections is resource intensive and time consuming
    - Tomcat can create a connection pool, a collection of connections ready to be used.
    - We simply request a connection, use it, then we can tell Tomcat we are finished with it
    - Tomcat manages the connection lifecycle
  - Databases can be moved/changed
    - Testing purposes
    - Server migration
    - Main database server may go down
  - Redirecting our application to a different database is as simple as telling Tomcat where the new one is
    - No need to change our working application code
  - Authentication details may change
    - Shouldn’t have to change application code to cater for this
    - Instead we can tell Tomcat the change.

JDBC – Tomcat

- apache-tomcat/conf/context.xml   or
- apache-tomcat/webapps/[APPNAME]/META-INF/context.xml

```
<Context>
 ...
 ...
 <Resource
    name="SENG2050_2015"
    auth="Container"
    type="javax.sql.DataSource"
    maxActive="100"
    maxIdle="30"
    maxWait="10000"
    username="username"
    password="password"
    driverClassName="com.microsoft.sqlserver.jdbc.SQLServerDriver"
    url="jdbc:sqlserver://localhost[\INSTANCENAME]:1433;databaseName=seng2050_lab06"/>
</Context>
```

Existing context contents

The name you will use to refer to this resource

Max number of active connections at any 1 time

Max number of unused connections to keep in the pool

How long to wait for a connection before throwing an exception

Your database login credentials

Database URL

MSSQL Instance Name (See setup videos)

Database name

JDBC – Tomcat

- There are more settings you can change
- Some of the ones I showed on the previous slide don’t need to be set
  - They have default values
- See more here:
  - https://people.apache.org/~fhanik/jdbc-pool/jdbc-pool.html

JDBC – How to use

1. Import libraries
2. Load the datasource
3. Get a connection
4. Execute Queries
5. Cleanup

JDBC – How to use

1. Imports:
   – import javax.sql.*;
   – import java.sql.*;
   – import javax.naming.InitialContext;
2. Load the datasource:
   – DataSource datasource = (DataSource)new InitialContext().lookup("java:/comp/env/SENG2050_2015");
3. Get a connection:
   – Connection connection = dataSource.getConnection();

JDBC – How to use

4. Execute Queries:
   - String query = “**SELECT * FROM item**”;
     Statement s = connection.createStatement();
     ResultSet rs = s.executeQuery(query);
     while(rs.next()){//process the results of the query}
   - String update = "**INSERT INTO item(name, price, description) VALUES (?, ?, ?)**";
     PreparedStatement s = connection.prepareStatement(update);
     s.setString(1, “someValue”);
     s.setDouble(2, 1.23);
     s.setString(3, “someOtherValue”);
     s.executeUpdate();
5. Cleanup:
   – s.close();
   – rs.close();
   – connection.close();
   – This tells the connection pool that we are done with the connection.

JDBC – How to use

- Note: If it touches the database it can probably throw an exception
  - java.sql.SQLException
- Make sure you deal with these in some way
  - If your method can't deal with the exception add **throws SQLException** to the method's signature
  - If it can deal with it, surround the code in a **try-catch** block

JDBC – How to use – Step 4

- To query the database use **executeQuery**, supplying a string that uses:
  - SELECT
- i.e.

```
Statement s = connection.createStatement();
ResultSet rs = s.executeQuery("SELECT * FROM item");
```

- Then you can iterate over the results:

```
while(rs.next()){
    //process the results of the query
    System.out.println(rs.getString("name"));
}
```

The name of the column.

JDBC – How to use – Step 4

- When you have run a query that successfully returns results you will receive a ResultSet object
  - This is an encapsulation of the result rows you would get if you issued your command from the command line
  - Maintains a cursor that points to the current row
- You will need to extract your data from this ResultSet
  - Use .get***Type***(int index) to get the value at the given index (starting from 1 not 0) of the current row
    - ***Types*** are Int, Long, BigDecimal, Double, String, UnicodeStream, BinaryStream, TimeStamp, Time, Object
    - i.e.:
      - rs.getString(1);
      - rs.getInt(2);
      - rs.getTime(1);
    - This converts the result depending on which method you call
  - Use .get***Type***(String columnName) to get the value with the given columnName, by casting it to the given Type

  - Use .next() to move to the cursor the next row in the ResultSet

JDBC – Example ResultSet

SELECT * FROM user;

user
id : varchar (PK)
name : varchar

Initially the cursor is not pointing at the first row

id	name
1	Ross
2	John
3	Joe
4	Mary
5	Bec
6	Alice
7	Bob
8	Charlie
9	Yuqing

Calling rs.next() moves the cursor to the first row

Calling rs.getString(2) returns “Ross”

Calling rs.next() moves the cursor to the next row

Calling rs.getInt(“id”) returns 2

Calling rs.absolute(5) moves the cursor to the 5th row

JDBC – How to use – step 4

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

JDBC – How to use

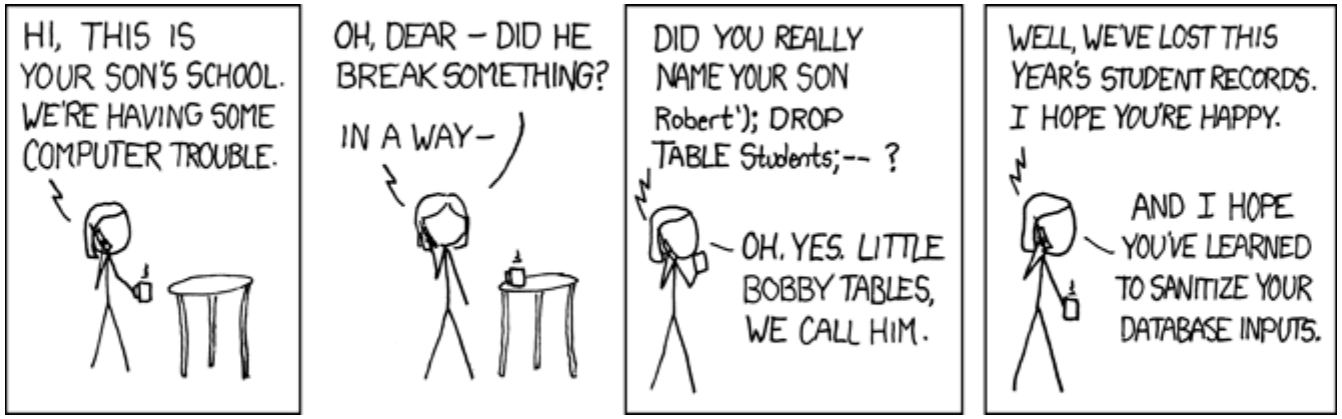
item
id : varchar (PK)
name : varchar
description : varchar
price : decimal

- Use prepared statements when creating dynamic SQL
  - SQL that contains parameter/variable values
  - i.e. instead of:
    - s.executeQuery(“SELECT * FROM item WHERE name = ‘”+item.name+”’”);
  - Use:
    - s = connection.prepareStatement(“SELECT * FROM item WHERE name = ?”);
    - Then use:
      - s.setString(1, item.name);
  - Why?

JDBC – Why you should use PreparedStatement

user
id : varchar (PK)
name : varchar

- Query: SELECT * FROM user WHERE id=1;
- Purpose: Get all the data from the user table associated to a user with the id of 1
- How **NOT** to do it
  - String id = request.getParameter("id");
    s.executeQuery("SELECT * FROM user WHERE id =" + id);
- Why: what if id is actually "1; DROP TABLE user;--"
  - The database will actually see 2 commands:
    1. SELECT * FROM user WHERE id =1;
    2. DROP TABLE user;--
  - Is that what you wanted?


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Source: xkcd.com

JDBC – Why you should use PreparedStatement

user
id : varchar (PK)
name : varchar

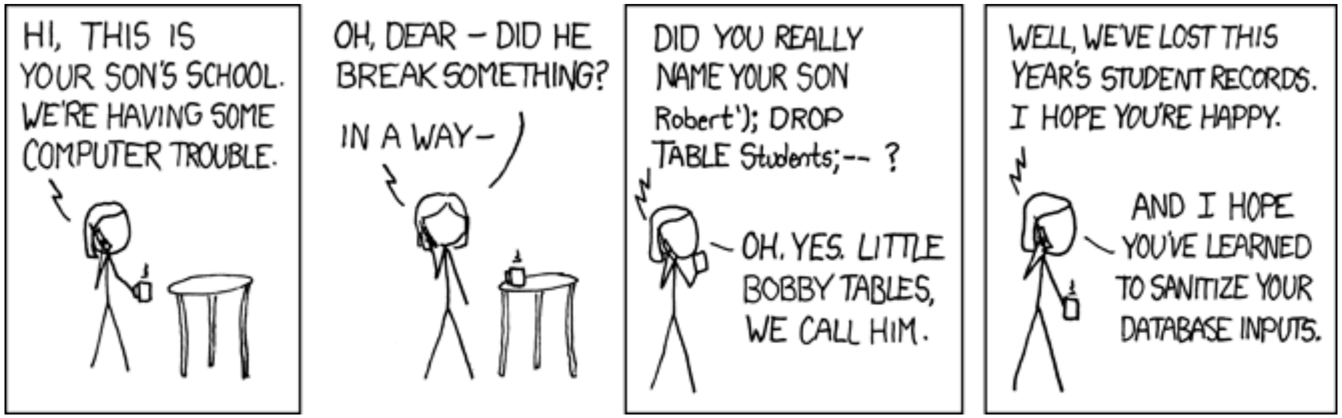
- Query: SELECT * FROM user WHERE id=1;
- Purpose: Get all the data from the user table associated to a user with the id of 1
- How to do it:

  -
    ```
    PreparedStatement s = connection.prepareStatement(“SELECT * FROM user WHERE id=?”);
    s.setString(1, id);
            // the first parameter is the index of the ? In the query string (starting from 1)
    s.executeUpdate();
    ```
- Why:
  - The database server will then recognise that id is some data that the query will use, not a command or a part of the query.
  - Under the hood:
    - prepareStatement() instructs MySQL to create a stored procedure that takes a parameter (marked with the ?).
    - setString() sets the values of the parameter.
    - executeUpdate() runs the stored procedure with the value you set.
  - Inputting a command will result in MySQL treating it as a string and not returning any results (unless there exists an id that matches the command), or an SQLException if the parameter is of the wrong type.

JDBC – Why you should use PreparedStatement

bankAccount
id : varchar (PK)
name : varchar
bal : decimal (FK)
userId : varchar (FK)

- Query: SELECT name, balance FROM bankAccount WHERE userId=1;
- Purpose: Get the account name and balance from the bankAccount table associated to a user with the userId of 1
- How **NOT** to do it
  - String userId = request.getParameter("userId");
    s.executeQuery("SELECT name, bal FROM bankAccount WHERE userId ="+ userId);
- Why: what if userId is actually "1 OR 1=1;--"
  - The database will actually see:
    - SELECT name, bal FROM bankAccount WHERE userId = 1 OR 1=1;--
  - Is that what you wanted?


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

JDBC – Why you should use PreparedStatement

bankAccount
id : varchar (PK)
name : varchar
bal : decimal (FK)
userId : varchar (FK)

- Query: SELECT name, balance FROM bankAccount WHERE userId=1;
- Purpose: Get the account name and balance from the bankAccount table associated to a user with the userId of 1
- How to do it
  - PreparedStatement s = connection.prepareStatement("SELECT name, bal FROM bankAccount WHERE userId =?");
    s.setString(1,userId);
    s.executeUpdate();
  - Why:
    - The database server will then recognise that userId is some data that the query will use, not a command or a part of the query.
    - Under the hood, prepareStatement() instructs MySQL to create a stored procedure that takes a parameter (marked with the ?). setString() sets the values of the parameter. executeUpdate() runs the stored procedure with the value you set.
    - Inputting a command will result in MySQL treating it as a string and not returning any results (unless there exists an id that matches the command), or an SQLException if the parameter is of the wrong type.

JDBC – How to use

- When you finish processing the results of a query, you should…
  - Call s.close(); – close the Statement object
  - Call rs.close(); - close the ResultSet object
  - If you need to keep the results for future calculations, then store them as attributes of a JavaBean first
- When you finish processing all queries for one "transaction", you should…
  - Call connection.close();

Summary

- Databases can help our web applications store and manage data
  - Making our applications more “dynamic”
- We use JDBC to connect our applications (web or traditional) to a database server
- We can then:
  - Send commands to the database server
    - To store or read data in a database
  - Process the results of these commands
- We should be careful not to make our applications vulnerable to SQL Injection
  - Never trust user input
  - Always validate user input (both on the client and on the server)
  - Always user PreparedStatements to run queries that require user input
  - Never create “dynamic queries” through string concatenation

Online Resources

- Oracle JDBC tutorial:
  - https://docs.oracle.com/javase/tutorial/jdbc/basics/
- JavaDoc
  - http://docs.oracle.com/javase/7/docs/api/java/sql/package-summary.html
- MSSQL database driver
  - https://docs.microsoft.com/en-us/sql/connect/jdbc/download-microsoft-jdbc-driver-for-sql-server?view=sql-server-ver15
- Microsoft SQL Server database software
  - See lab videos for setup at home
- Tomcat JDBC connection Pool
  - https://people.apache.org/~fhanik/jdbc-pool/jdbc-pool.html
  - https://tomcat.apache.org/tomcat-8.0-doc/jndi-datasource-examples-howto.html#Preventing_database_connection_pool_leaks
- XKCD comic
  - https://xkcd.com/327/
- Info on SQL injection and other issues (take a look at these)
  - https://www.owasp.org/index.php/Top_10_2013-A1-Injection
  - http://www.smashingmagazine.com/2011/01/keeping-web-users-safe-by-sanitizing-input-data/