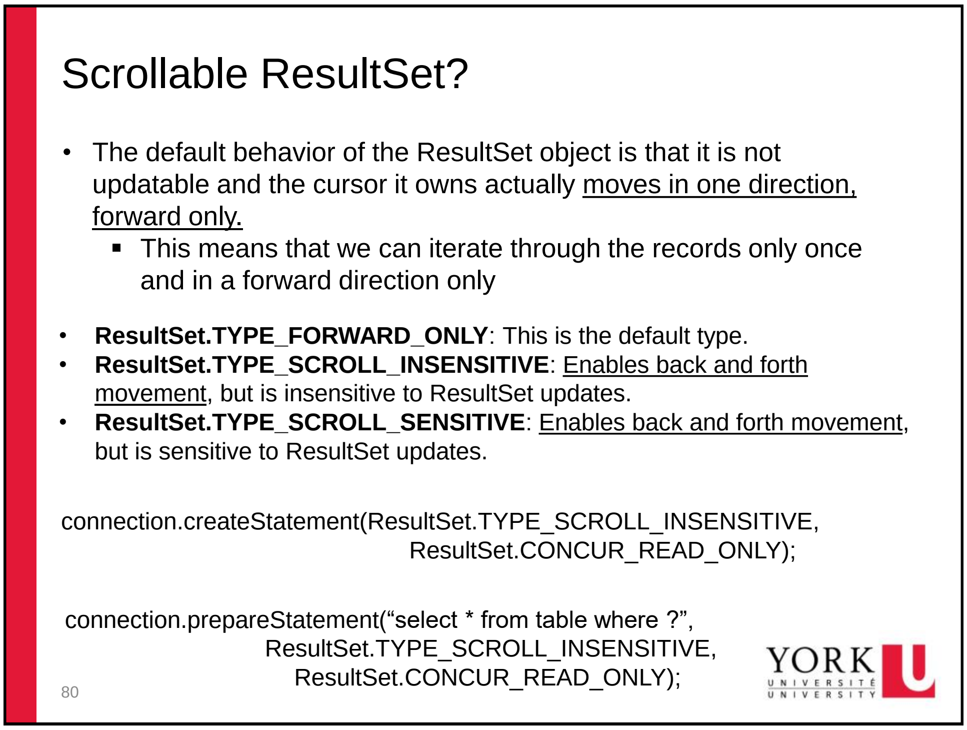


Scrollable ResultSet?

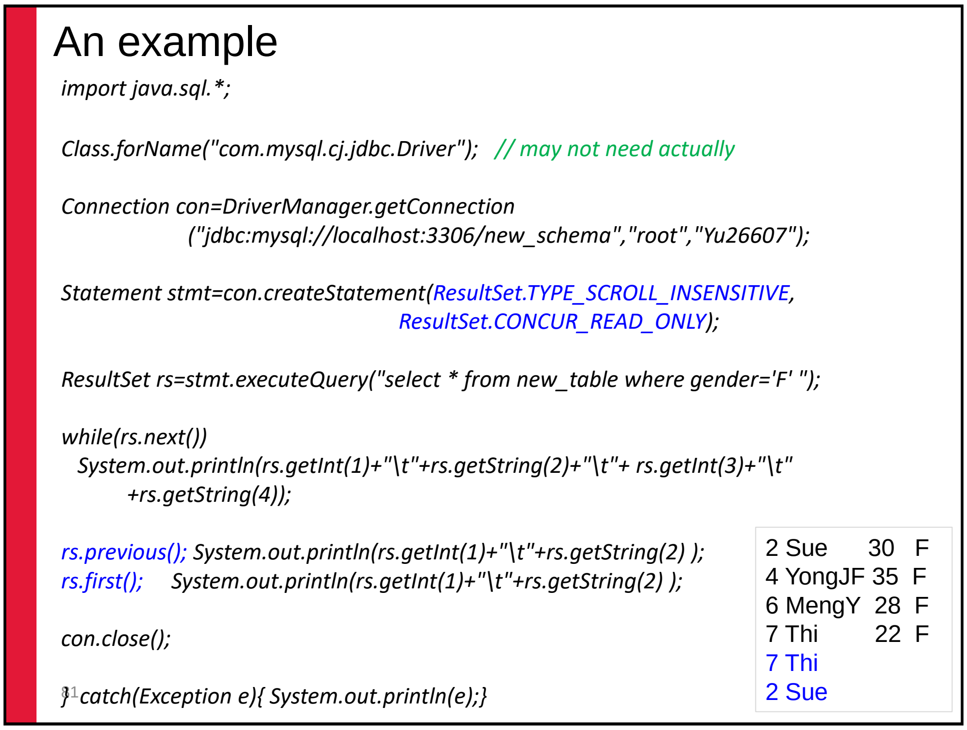
- The default behavior of the ResultSet object is that it is not updatable and the cursor it owns actually moves in one direction, forward only.
  - This means that we can iterate through the records only once and in a forward direction only

- **ResultSet.TYPE_FORWARD_ONLY**: This is the default type.
- **ResultSet.TYPE_SCROLL_INSENSITIVE**: Enables back and forth movement, but is insensitive to ResultSet updates.
- **ResultSet.TYPE_SCROLL_SENSITIVE**: Enables back and forth movement, but is sensitive to ResultSet updates.

```
connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                                   ResultSet.CONCUR_READ_ONLY);


connection.prepareStatement("select * from table where ?",
                  ResultSet.TYPE_SCROLL_INSENSITIVE,
                    ResultSet.CONCUR_READ_ONLY);
```

80

80



An example

```
import java.sql.*;

Class.forName("com.mysql.cj.jdbc.Driver");   // may not need actually

Connection con=DriverManager.getConnection
              ("jdbc:mysql://localhost:3306/new_schema","root","Yu26607");

Statement stmt=con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                                   ResultSet.CONCUR_READ_ONLY);

ResultSet rs=stmt.executeQuery("select * from new_table where gender='F' ");

while(rs.next())
  System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+ rs.getInt(3)+"\t"
        +rs.getString(4));

rs.previous(); System.out.println(rs.getInt(1)+"\t"+rs.getString(2) );
rs.first();    System.out.println(rs.getInt(1)+"\t"+rs.getString(2) );

con.close();

}catch(Exception e){ System.out.println(e);}
```

```
2 Sue     30  F
4 YongJF 35  F
6 MengY  28  F
7 Thi       22  F
7 Thi
2 Sue
```

81

81

An example

```
import java.sql.*;

Class.forName("com.mysql.cj.jdbc.Driver");   // may not need actually

Connection con=DriverManager.getConnection
                  ("jdbc:mysql://localhost:3306/new_schema","root","Yu26607");

Statement stmt=con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                                                 ResultSet.CONCUR_READ_ONLY);

ResultSet rs=stmt.executeQuery("select * from new_table where gender='F' ");

while(rs.next())
  System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+ rs.getInt(3)+"\t"
         +rs.getString(4));

rs.first();
while(rs.next())
  System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+ rs.getInt(3)+"\t"
         +rs.getString(4));

con.close();
```

```
2 Sue     30  F
4 YongJF 35  F
6 MengY  28  F
7 Thi      22  F
4 YongJF 35  F
6 MengY  28  F
7 Thi      22
```

82

Metadata – data about data

- Two kinds of meta data in JDBC
  - **Database Metadata**: To look up information about the database
  - **ResultSet Metadata**: To get the structure of data that is returned

- Metadata from DB
  Connection con = …
  *DatabaseMetaData d = con.**getMetaData**();*

  provide schema information describing its tables etc,

- Metadata from ResultSet
  ResultSet rs = stmt.executeQuery();
  *ResultSetMetaData rd = rs.**getMetaData**();*

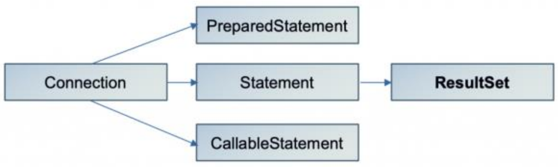
  provide information about the result, Column label, number of columns

83

JDBC Interface classes

RECALL

- Java.sql package
  - Driver
  - DriverManager
  - Connection
  - Statement
  - PreparedStatement
  - CallableStatement
  - ResultSet
  - **ResultSetMetaData**
  - **DatabaseMetatData**



- **ResultSet:** These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.
- **ResultSetMetaData** This interface is used to get the information about the result set such as, number of columns, name of the column, data type of the column, schema of the result set, table name, etc
  - It provides methods such as **getColumnCount**(), **getColumnName**(),
- **DatabaseMetaData** This interface is used to get the information about the database schema such as username, table name

84

```
Connection con=DriverManager.getConnection
                ("jdbc:mysql://localhost:3306/new_schema","root","Yu26607");

DatabaseMetaData dbmd = con.getMetaData();
System.out.println("Driver Name" + dbmd.getDriveName());
System.out.println("Driver Version" + dbmd.getDriveVersion());
System.out.println("User Name" + dbmd.getUserName());
System.out.println("Data base Name" + dbmd.getDatabaseName());
System.out.println("Data base version" + dbmd.getDatabaseVersion());

Staetemetn stmt = con.createStatemetn();
ResultSet rs = stmt.executreQuery("select * from new_table where gender='F' "));

ResultSetMetaData md = rs.getMetaData();

// get number of columns
int nCols = md.getColumnCount();
// print column names
for(int i=1; i <= nCols; ++i)
   System.out.print( md.getColumnName(i) + "\t");

 // output resultset
while ( rs.next() )
{    for(int i=1;  i <= nCols; ++i)
        System.out.print( rs.getString(i) + "\t");
     System.out.println( );
}
```

```
Driver Name: MySQL Connector/J
Driver Version: mysql-connector-java-8.0.27 ..
UserName: root@localhost
Database Product name: MySQL
Database Product Version: 8.0.27

#columns: 4
ID    name     age    gender
2     Sue       30     F
4     YongJF   35     F
6     MengY   28     F
7      Thi       22     F
```

85

JDBC

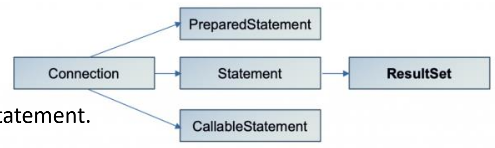
- JDBC Introduction
  - Basics
  - Scrollable ResultSet
  - Metadata
- **Improve: PreparedStatement**
- Web app
  - Improve: Data source
- SQLite
- DAO Design pattern
- SQL injection

86

86

Improve: using PreparedStatement

Types of Statement available



- **Statement**

  This represents a simple sql/mysql statement.

  Statement stmt = con.createStatement();

- **PreparedStatement**

  This represent precompiled sql/my sql statement which allows improved performance. It allows to execute the query multiple times and we can set the values according to our need.

  *PreparedStatement psmt = con.prepareStatement("select * from S where ? ");*

- **CallableStatement**

  This allows the access of stored procedures that are stored on the database

  CallableStatement csmt = con.prepareCall();

87

87

Improve: using PreparedStatement

- In database management systems (DBMS), a **prepared statement**, **parameterized statement**, or **parameterized query** is a feature where the database pre-compiles SQL code and stores the results, separating it from data.

- Benefits of prepared statements are:
  - efficiency, because they can be used repeatedly without re-compiling
    - Pre-compilation and DB-side caching of the SQL statement leads to overall faster execution and the ability to reuse the same SQL statement in batches.
  - security, by reducing or eliminating SQL injection attacks
    - Automatic prevention of SQL injection attacks by built in escaping of quotes and other special characters

Statement vs prepared statement

Most relational databases handles a JDBC / SQL query in four steps:

1. **Parse the incoming SQL query**
2. **Compile the SQL query**
3. **Plan/optimize the data acquisition path**
4. **Execute the optimized query / acquire and return data**

- A **Statement** will always proceed through the four steps above for each SQL query sent to the database.
- A **Prepared Statement** pre-executes steps (1) - (3) in the execution process above. When creating a Prepared Statement (with SQL), parsing/precompiling and some pre-optimization is performed immediately. The effect is to lessen the load on the database engine at execution time.

A common workflow for prepared statements is:

- **Prepare**: The application creates the statement template and sends it to the DBMS. Certain values are left unspecified, called parameters, placeholders or bind variables (labelled "?" below):

  INSERT INTO products (name, price) VALUES (?, ?);
- **Compile**: The DBMS compiles (parses, optimizes and translates) the statement template, and stores the result without executing it.
- **Execute**: The application supplies (or binds) values for the parameters of the statement template, and the DBMS executes the statement (without recompiling). The application may request the DBMS to execute the statement many times with different values.

Prepared Statement

- It has three main uses
  - Create parameterized statements such that data for parameters can be dynamically substituted
  - Precompiling SQL statements to avoid repeated parsing/compiling of the same SQL statement – efficient for repeated executions
  - Create statements where data values may not be character strings
- If parameters for the query are not set the driver returns an SQL Exception
- Only the no parameters versions of *executeUpdate*() and *executeQuery*() are allowed with prepared statements.

Example

```
// Creating a prepared Statement
String sqlString = "UPDATE authors SET lastname = ? Authid = ?";
PreparedStatement ps = connection.prepareStatement(sqlString);
ps.setString(1, “Allamaraju”);     // Sets first placeholder to Allamaraju
ps.setInt(2, 212);                // Sets second placeholder to 212
ps.executeUpdate();                // Executes the update
```

YORK UNIVERSITÉ UNIVERSITY

An example

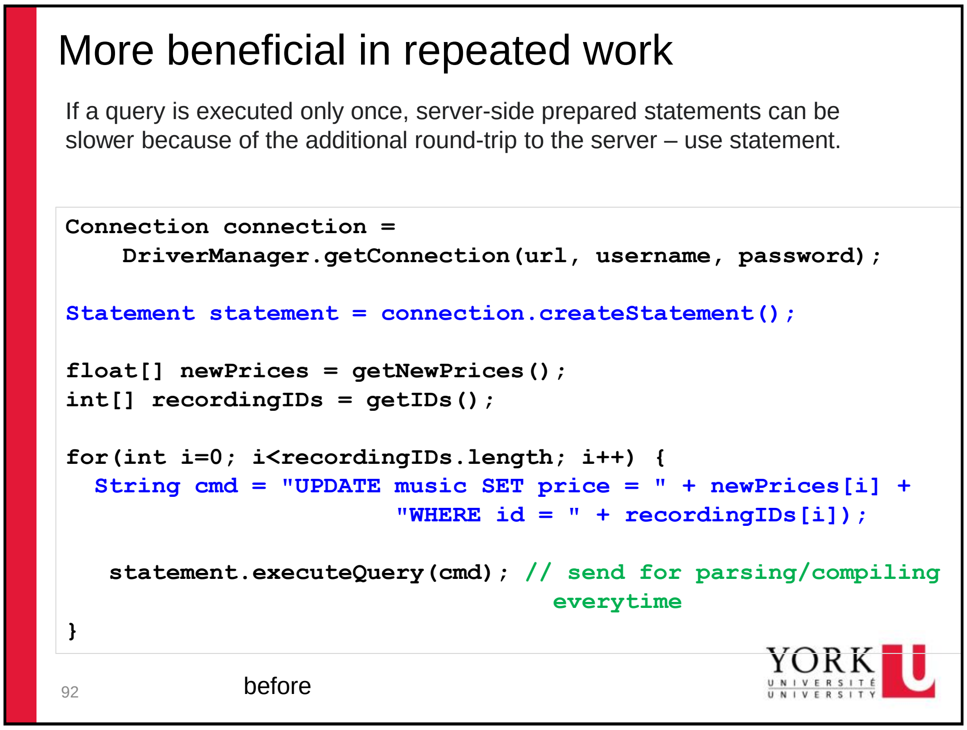
```
Class.forName("com.mysql.cj.jdbc.Driver"); // may not need actually

Connection con=DriverManager.getConnection
              ("jdbc:mysql://localhost:3306/new_schema","root","Yu26607");
```

~~Statement stmt=con.createStatement();~~
~~ResultSet rs=stmt.executeQuery("select * from new_table where gender='F' ");~~

```
PreparedStement psmt = con.prepareStatement("select * from new_table where gender=? ");
                     // send for parsing/pre-compiling (only once)

psmt.setString (1, "F");
ResultSet rs = psmt.executeQuery(); // will not parse/compile again

while(rs.next())
  System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+ rs.getInt(3)+"\t" + rs.getString(4) );

con.close();
```

```
2 Sue      30  F
4 YongJF 35  F
6 MengY  28  F
7 Thi        22  F
```

YORK UNIVERSITÉ UNIVERSITY



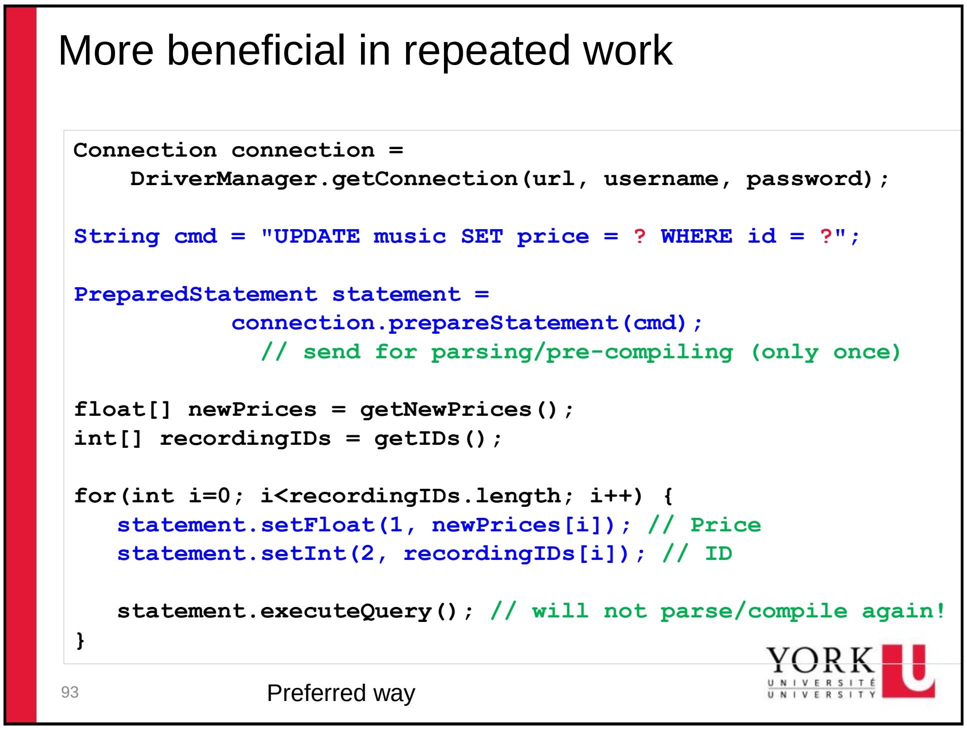
More beneficial in repeated work

If a query is executed only once, server-side prepared statements can be slower because of the additional round-trip to the server – use statement.

```
Connection connection =
    DriverManager.getConnection(url, username, password);

Statement statement = connection.createStatement();

float[] newPrices = getNewPrices();
int[] recordingIDs = getIDs();

for(int i=0; i<recordingIDs.length; i++) {
  String cmd = "UPDATE music SET price = " + newPrices[i] +
                     "WHERE id = " + recordingIDs[i]);

   statement.executeQuery(cmd); // send for parsing/compiling
                                           everytime
}
```

92 before

92



More beneficial in repeated work

```
Connection connection =
    DriverManager.getConnection(url, username, password);

String cmd = "UPDATE music SET price = ? WHERE id = ?";

PreparedStatement statement =
           connection.prepareStatement(cmd);
             // send for parsing/pre-compiling (only once)

float[] newPrices = getNewPrices();
int[] recordingIDs = getIDs();

for(int i=0; i<recordingIDs.length; i++) {
   statement.setFloat(1, newPrices[i]); // Price
   statement.setInt(2, recordingIDs[i]); // ID

   statement.executeQuery(); // will not parse/compile again!
}
```

93 Preferred way

93

Prepared Statement

- If you are going to execute similar SQL statements multiple times, using parameterized (or “prepared”) statements can be more efficient than executing a raw query each time. The idea is to create a parameterized statement in a standard form that is sent to the database for compilation before actually being used.
  - Only parse/compile once. Later statement will be executed without compiling
- However, performance is not the only advantage of a prepared statement. Security is another advantage. We recommend that you always use a prepared statement or stored procedure to update database values when accepting input from a user through an HTML form.
- Prevent SQL Injection attack.

94

94

Most relational databases handles a JDBC / SQL query in four steps:

1. **Parse the incoming SQL query**
2. **Compile the SQL query**
3. **Plan/optimize the data acquisition path**
4. **Execute the optimized query / acquire and return data**

- A **Statement** will always proceed through the four steps above for each SQL query sent to the database.
- A **Prepared Statement** When creating a Prepared Statement (with SQL), 1-3 parsing/precompiling and some pre-optimization is performed immediately.

Statement	PreparedStatement
It is base interface.	It extends statement interface.
You can not pass parameters at runtime.	You can pass parameters at runtime.
Used for DDL statement, CREATE, ALTER, DROP statements.	Used for the queries which are to be executed multiple times.
Performance is relatively low.	Performance is better than Statement.
Used to execute normal SQL queries.	Used to execute dynamic SQL queries.
We can not use statement for reading/writing binary data.	We can use Preparedstatement for reading/writing binary data.
No binary protocol is used for communication.	Binary protocol is used for communication.

95

95

JDBC

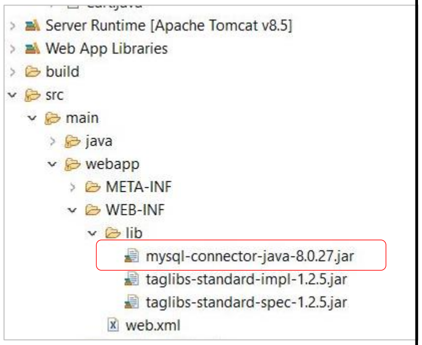
- JDBC Introduction
  - Basics
  - Scrollable ResultSet
  - Metadata
- Improve: PreparedStatement
- **Web app**
  - Improve: Data source
- SQLite
- DAO Design pattern
- SQL injection

96

96

Java application and web application

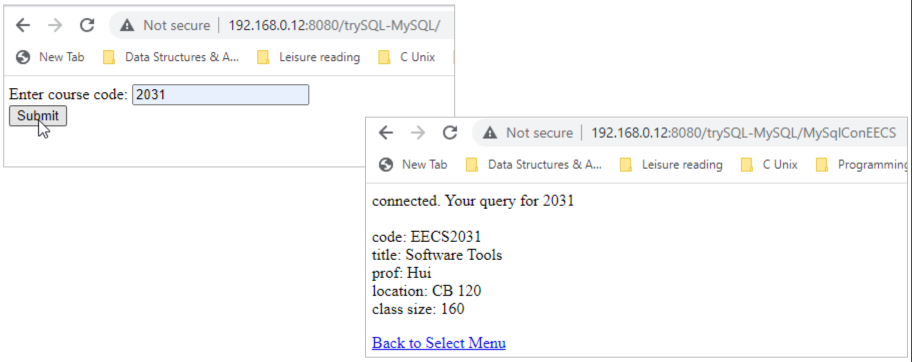
- Application:
  - Add jar file to build path
- Web app:
  - Add jar file to project web lib
    - may need *forName*
  - Add to Tomcat classpath run configuration
    - Apply to all web applications



```
> Server Runtime [Apache Tomcat v8.5]
> Web App Libraries
> build
v src
  v main
    > java
    v webapp
      > META-INF
      v WEB-INF
        v lib
            mysql-connector-java-8.0.27.jar
            taglibs-standard-impl-1.2.5.jar
            taglibs-standard-spec-1.2.5.jar
          web.xml
```

97

97


Not secure | 192.168.0.12:8080/trySQL-MySQL/
New Tab
Data Structures & A...
Leisure reading
C Unix
Submit
Not secure | 192.168.0.12:8080/trySQL-MySQL/MySqlConEECS
Programming
Back to Select Menu

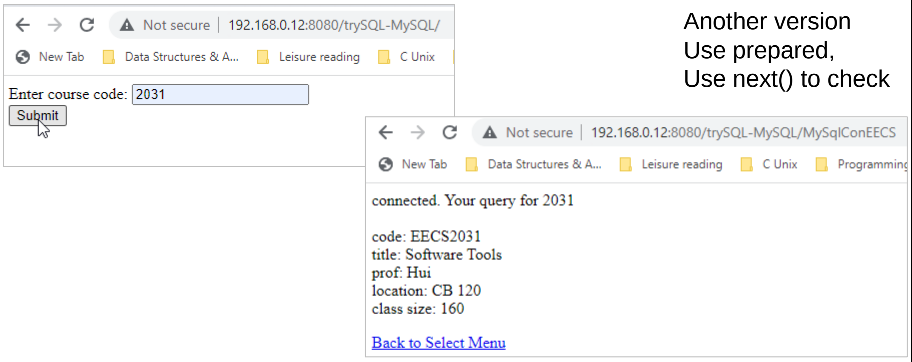
Enter course code: 2031

```
connected. Your query for 2031

code: EECS2031
title: Software Tools
prof: Hui
location: CB 120
class size: 160
```

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
                          throws ServletException, IOException {

     String code = request.getParameter("code");
     Class.forName("com.mysql.cj.jdbc.Driver");
     Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/..);

     Statement stmt = con.createStatement();
     ResultSet rs = stmt.executeQuery("select * from Courses where code =" + code);

     if (! rs.isBeforeFirst())  out.println("No such a course");
     else{
        rs.next();
        out.println("code: EECS"+res.getInt(1) + "<br> title"…
```

99

99


Not secure | 192.168.0.12:8080/trySQL-MySQL/
New Tab
Data Structures & A...
Leisure reading
C Unix
Submit
Not secure | 192.168.0.12:8080/trySQL-MySQL/MySqlConEECS
Programming
Back to Select Menu

Enter course code: 2031

```
connected. Your query for 2031

code: EECS2031
title: Software Tools
prof: Hui
location: CB 120
class size: 160
```

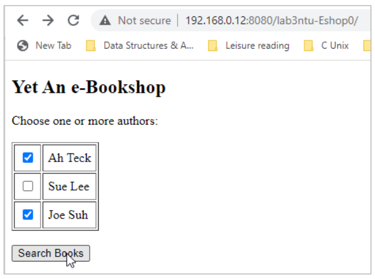
Another version
Use prepared,
Use next() to check

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
                          throws ServletException, IOException {

     String code = request.getParameter("code");
     Class.forName("com.mysql.cj.jdbc.Driver");
     Connection con= DriverManager.getConnection("jdbc:mysql://localhost:3306/..);

     PreparedStatement stmt = con.PreparedStatement(select * from Courses where
                                                                   code = ?");

     stmt.setString(1, code);
     ResultSet rs = stmt.executeQuery();

     if (! rs.next())  out.println("No such a course");
     else{   out.println("code: EECS"+res.getInt(1) + "<br> title"…
```

100

100

Related to our work



Not secure | 192.168.0.12:8080/lab3ntu-Eshop0/

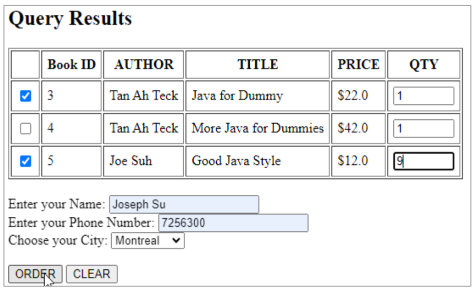
New Tab | Data Structures & A... | Leisure reading | C Unix

Yet An e-Bookshop

Choose one or more authors:

- [✓] Ah Teck
- [ ] Sue Lee
- [✓] Joe Suh

Search Books



Query Results

	Book ID	AUTHOR	TITLE	PRICE	QTY
✓	3	Tan Ah Teck	Java for Dummy	$22.0	1
	4	Tan Ah Teck	More Java for Dummies	$42.0	1
✓	5	Joe Suh	Good Java Style	$12.0	9

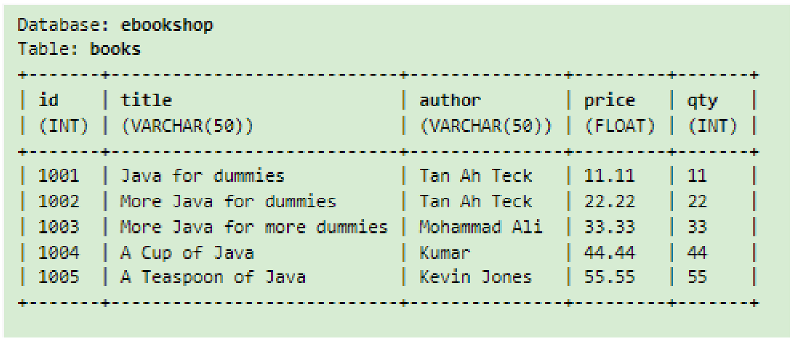
Enter your Name: Joseph Su
Enter your Phone Number: 7256300
Choose your City: Montreal

ORDER CLEAR

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
                         throws ServletException, IOException {

        String authors[] = request.getParameterValues("author");
        Table t = new Table();
        ArrayList<Book> resu = t.search(authors);

        /* for (Book e : resu) {
          printWritter…..(e.id);
           printWritter…..(e.author );
           …
        }  */

        request.setAttribute("searchedBooks", resu);
        String target = "bookResultView.jsp";
        request.getRequestDispatcher(target).forward(request, response);
```

101

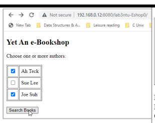
101



```
Database: ebookshop
Table: books
+--------+---------------------------+---------------+---------+-------+
| id     | title                     | author        | price   | qty   |
| (INT)  | (VARCHAR(50))             | (VARCHAR(50)) | (FLOAT) | (INT) |
+--------+---------------------------+---------------+---------+-------+
| 1001   | Java for dummies          | Tan Ah Teck   | 11.11   | 11    |
| 1002   | More Java for dummies     | Tan Ah Teck   | 22.22   | 22    |
| 1003   | More Java for more dummies| Mohammad Ali  | 33.33   | 33    |
| 1004   | A Cup of Java             | Kumar         | 44.44   | 44    |
| 1005   | A Teaspoon of Java        | Kevin Jones   | 55.55   | 55    |
+--------+---------------------------+---------------+---------+-------+
```

102

102

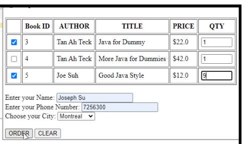


Yet An e-Bookshop

Choose one or more authors:

✓	Ah Teck
☐	Sue Lee
✓	Joe Suh

Search Books



	Book ID	AUTHOR	TITLE	PRICE	QTY
✓	3	Tan Ah Teck	Java for Dummy	$22.0	1
☐	4	Tan Ah Teck	More Java for Dummies	$42.0	1
✓	5	Joe Suh	Good Java Style	$12.0	9

Enter your Name: Joseph Su
Enter your Phone Number: 7256300
Choose your City: Montreal

ORDER CLEAR

```
protected void doGet(HttpServletRequest request, HttpServletResponse response){

    String authors[] = request.getParameterValues("author");
    // get list of books from database

    String query = "select * from books where author = '"+authors[0]+"'" ;
    for(int i=1; i<authors.length; i++)
          query += " OR author = '" + authors[i]+ "'";
    query += ";" ;
    Class.forName("com.mysql.cj.jdbc.Driver");
    Connection con= DriverManager.getConnection("jdbc:mysql://localhost:3306/..);
    Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery(query);

     ArrayList<Book> resu = new ArrayList<>();
     while (rs.next()) {
          int id = rs.getInt("id");
          String title = rs.getString("title");
          String author = rs.getString("author");
          float price = rs.getFloat("price");
          Book b = new Book (id,  author, title, price);   // "populate bean"
          resu.add(b);
     }
     request.setAttribute("searchedBooks", resu);
     String target = "bookResultView.jsp";
     request.getRequestDispatcher(target).forward(request, response);
```

Improve?: DAO

103

103

JDBC

- JDBC Introduction
  - Basics
  - Scrollable ResultSet
  - Metadata

- Improve: PreparedStatement

- Web app
  - **Improve: Data source**

- SQLite

- DAO Design pattern

- SQL injection

104

104

With Driver manager

- Application code database details (e.g., url, user, passwd). Need to change code if database changes
  - database not transparent to applications
- Each time a client attempts to access a backend service, it requires OS resources to create, maintain, and close connections to the datastore. This creates a large amount of overhead causing database performance to deteriorate.
  - Does not support connection pooling

- Connecting to a backend service is an expensive operation, as it consists of the following steps:
  - Open a connection to the database using the database driver.
  - Open a TCP socket for CRUD operations
  - Authenticate users
  - Perform CRUD operations over the socket.
  - Close the connection.
  - Close the socket.

105

105

With Driver manager

- For simple operations at small scale, the steps involved in opening and closing a connection are okay. As application scales up, however, the constant opening and closing of connections becomes more expensive and can begin to impact application's performance.

- Often, it makes sense to find a way of keeping connections open and passing them from operation to operation as they're needed, rather than opening and closing a brand-new connection for each operation.

- Connection pooling: Instead of opening and closing connections for every request, connection pooling uses a cache of database connections that can be reused when future requests to the database are required.

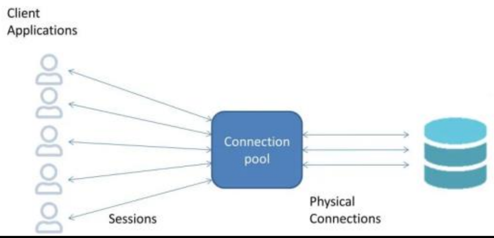
106

106

Connection pooling

- Connection pooling is a technique used in database management systems to manage a pool of database connections that can be reused for multiple requests instead of creating a new connection for each request. This helps to reduce the overhead associated with establishing a new connection, which can be time-consuming and resource-intensive.

- When a client requests a connection to the database, the connection pool checks if there are any available connections in the pool. If there is an available connection, it's returned to the client. If there are no available connections, the pool creates a new connection and returns it to the client. Once the client is done with the connection, it's returned back to the pool for reuse.



107

107

JDBC Data Source

- The JDBC date source interface is an alternative to **DriverManager** class and conventional JDBC url.
- All the database information is maintained externally to the application, present in the Naming service and retrieved using the JNDI API. The Data Source object contains the connection information which will make the actual connection and execute the JDBC commands.

- Details of Database is transparent to the application
- Support connection pooling

- Javax.sql adds functionality for enterprise applications
  - DataSources
  - JNDI
  - Connection Pooling
  - Rowsets
  - Distributed Transactions

108

108

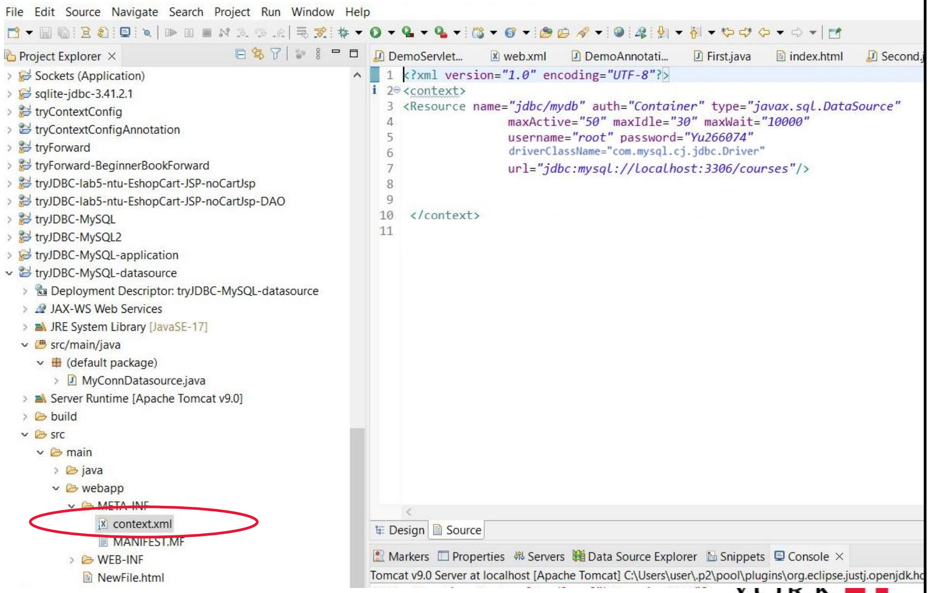
- Syntax:

```
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.sql.DataSource;
```

context ctx = new InitialContext();
DataSource ds = (DataSource) ctx.lookup(…)
Connection con = ds.getConnection("username", "passwd")

Recall RMI

110


File Edit Source Navigate Search Project Run Window Help
Project Explorer
DemoServlet… web.xml DemoAnnotati… First.java index.html Second.j
Design Source
Markers Properties Servers Data Source Explorer Snippets Console

- Sockets (Application)
- sqlite-jdbc-3.41.2.1
- tryContextConfig
- tryContextConfigAnnotation
- tryForward
- tryForward-BeginnerBookForward
- tryJDBC-lab5-ntu-EshopCart-JSP-noCartJsp
- tryJDBC-lab5-ntu-EshopCart-JSP-noCartJsp-DAO
- tryJDBC-MySQL
- tryJDBC-MySQL2
- tryJDBC-MySQL-application
- tryJDBC-MySQL-datasource
  - Deployment Descriptor: tryJDBC-MySQL-datasource
  - JAX-WS Web Services
  - JRE System Library [JavaSE-17]
  - src/main/java
    - (default package)
      - MyConnDatasource.java
  - Server Runtime [Apache Tomcat v9.0]
  - build
  - src
    - main
      - java
      - webapp
        - META-INF
          - context.xml
          - MANIFEST.MF
        - WEB-INF
        - NewFile.html

```
<?xml version="1.0" encoding="UTF-8"?>
<context>
<Resource name="jdbc/mydb" auth="Container" type="javax.sql.DataSource"
                maxActive="50" maxIdle="30" maxWait="10000"
                username="root" password="Yu266074"
                driverClassName="com.mysql.cj.jdbc.Driver"
                url="jdbc:mysql://localhost:3306/courses"/>


 </context>
```

Tomcat v9.0 Server at localhost [Apache Tomcat] C:\Users\user\.p2\pool\plugins\org.eclipse.justj.openjdk.ho

111

111

Context.xml under webapp/META-INF

```
<?xml version="1.0" encoding="UTF-8"?>
<context>
 <Resource name="jdbc/mydb" auth="Container" type="javax.sql.DataSource"
          maxActive="50" maxIdle="30" maxWait="10000"
          username="root" password="Yu266074"
          driverClassName="com.mysql.cj.jdbc.Driver"
          url="jdbc:mysql://localhost:3306/courses"    />
</context>
```

```
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.sql.DataSource;

void doGet(HttpServletRequest request, HttpServletResponse response)  {
  try{
    Context ctx = new InitialContext();
    DataSource dSource = (DataSource)
                             ctx.lookup("java:comp/env/jdbc/mydb");

    Connection con= dSource.getConnection();

    String courseCode = request.getParameter("code");
    Statement stmt=con.createStatement();
    ResultSet rs=stmt.executeQuery("select * from new_table where code
```

112

112

JDBC

- JDBC Introduction
  - Basics
  - Scrollable ResultSet
  - Metadata
- Improve: PreparedStatement
- Web app
  - Improve: Data source
- **Serverless RDMS: SQLite**
- DAO Design pattern
- SQL injection

113

113

JDBC with SQLite

- Add jar file   sqlite-jdbc-xxxx.jar

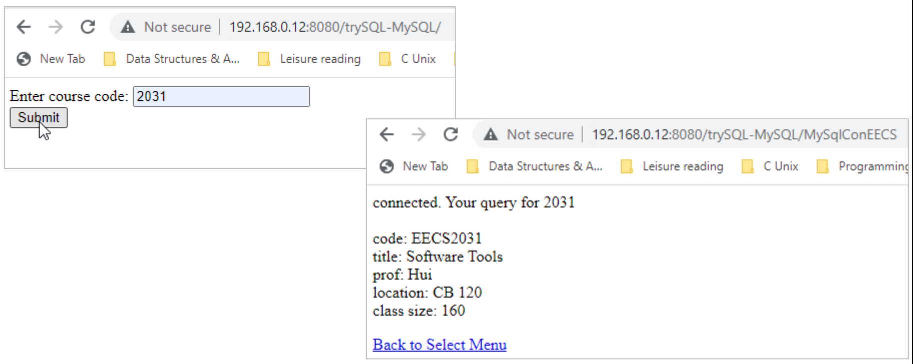
- Driver

```
Class.forName("org.sqlite.JDBC");
Connection con = driverManager.getConnection
                           ("jdbc:sqlite:path_of_db_file");
```

- Other same as MySql.

114

114


Not secure | 192.168.0.12:8080/trySQL-MySQL/

Enter course code: 2031

Submit

Not secure | 192.168.0.12:8080/trySQL-MySQL/MySqlConEECS

connected. Your query for 2031

code: EECS2031
title: Software Tools
prof: Hui
location: CB 120
class size: 160

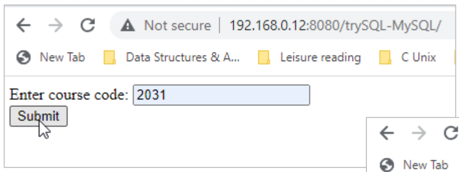
Back to Select Menu

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
                          throws ServletException, IOException {

     String code = request.getParameter("code");
     Class.forName("org.sqlite.JDBC");
     Connection con= driverManager.getConnection("jdbc:sqlite:C:\\users\\
                                                   Desktop\\Course.db");

     Statement stmt=con.createStatement();
     ResultSet rs=stmt.executeQuery("select * from Courses where code =" + code);

     if (! rs.isBefreFirst())  out.println("No such a course");
     else{
        rs.next();
```

115

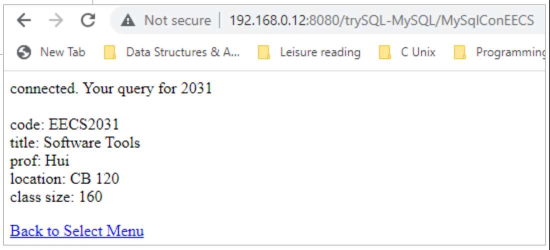
115

Another version
Use prepared,
Use next to check



Not secure | 192.168.0.12:8080/trySQL-MySQL/

Enter course code: 2031
Submit



Not secure | 192.168.0.12:8080/trySQL-MySQL/MySqlConEECS

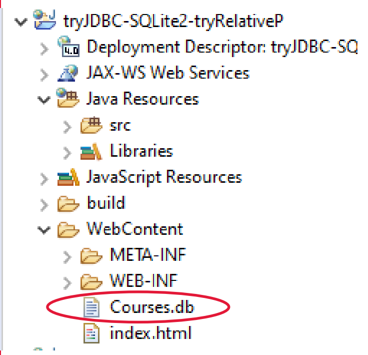
```
connected. Your query for 2031

code: EECS2031
title: Software Tools
prof: Hui
location: CB 120
class size: 160

Back to Select Menu
```

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
                           throws ServletException, IOException {

     String code = request.getParameter("code");
     Class.forName("org.sqlite.JDBC");
     Connection con= driverManager.getConnection("jdbc:sqlite:C:\\users\\
                                                        Desktop\\Course.db");


     PreparedStatement stmt =con.PreparedStatement(select * from Courses where
                                                                  code = ?");

     stmt.setString(1, code);
     ResultSet rs=stmt.executeQuery();
```

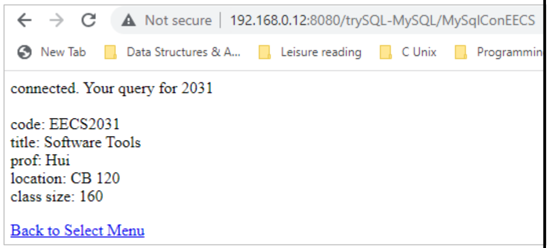
116

116

db file in project
More portable



```
tryJDBC-SQLite2-tryRelativeP
  Deployment Descriptor: tryJDBC-SQ
  JAX-WS Web Services
  Java Resources
    src
    Libraries
  JavaScript Resources
  build
  WebContent
    META-INF
    WEB-INF
    Courses.db
    index.html
```



Not secure | 192.168.0.12:8080/trySQL-MySQL/MySqlConEECS

```
connected. Your query for 2031

code: EECS2031
title: Software Tools
prof: Hui
location: CB 120
class size: 160

Back to Select Menu
```

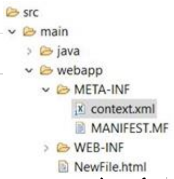
```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
                           throws ServletException, IOException {

     String code = request.getParameter("code");
     Class.forName("org.sqlite.JDBC");
     String path = this.getServletContext().getRealPath("/Courses.db") ;
     Connection con=DriverManager.getConnection("jdbc:sqlite:" + path);

     PreparedStatement stmt =con.PreparedStatement(select * from Courses where
                                                                  code = ?");

     stmt.setString(1, code);
     ResultSet rs=stmt.executeQuery();

     if (! rs.next())  out.println("No such a course");
```

117

117

Data source
Context.xml under webapp/META-INF

```
<?xml version="1.0" encoding="UTF-8"?>
<context>
 <Resource name="jdbc/mydb" auth="Container" type="javax.sql.DataSource"
          maxActive="50" maxIdle="30" maxWait="10000"
           driverClassName="org.sqlite.jdbc"
          url="jdbc:mysql:C: :\\users\\Desktop\\Course.db"/>
</context>
```



```
src
  main
    java
    webapp
      META-INF
        context.xml
        MANIFEST.MF
      WEB-INF
      NewFile.html
```

```
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.sql.DataSource;

void doGet(HttpServletRequest request, HttpServletResponse response)  {
  try{
    Context ctx = new InitialContext();
    DataSource dSource = (DataSource)
                                   ctx.lookup("java:comp/env/jdbc/mydb");

    Connection con= dSource.getConnection();

    String courseCode = request.getParameter("code");
    Statement stmt=con.createStatement();
    ResultSet rs=stmt.executeQuery("select * from new_table where code
= '
```

118

JDBC

- JDBC Introduction
  - Basics
  - Scrollable ResultSet
  - Metadata
- Improve: PreparedStatement
- Web app
  - Improve: Data source
- Serverless RDMS: SQLite
- **DAO Design pattern**
- SQL injection

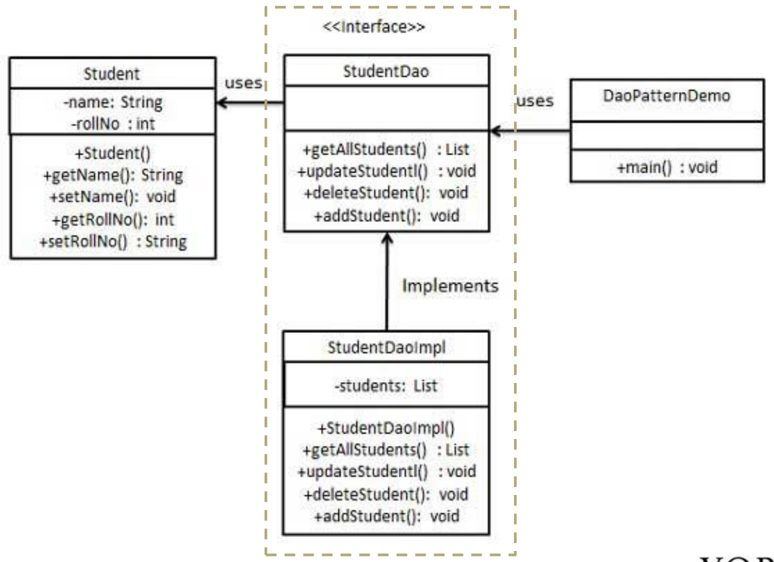
120

120

Data Access Object (DAO) design pattern

Data Access Object Pattern or **DAO** pattern is used to separate low level data accessing API or operations from high level business logics/services. Following are the participants in Data Access Object Pattern.

- **Data Access Object Interface** - This interface defines the standard operations to be performed on a model object(s).
- **Data Access Object concrete class** - This class implements above interface. This class is responsible to get data from a data source which can be database / xml or any other storage mechanism.
- **Model Object or Value Object** - This object is simple POJO containing get/set methods to store data retrieved using DAO class.



Somewhere show db once implemented

```
public class Student {
   private String name;
   private int rollNo;

   Student(String name, int rollNo){
      this.name = name;
      this.rollNo = rollNo;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public int getRollNo() {
      return rollNo;
   }

   public void setRollNo(int rollNo)
      this.rollNo = rollNo;
   }
}
```

```
Create Data Access Object Interface.
StudentDao.java

import java.util.List;

public interface StudentDao {
   public List<Student> getAllStudents();
   public Student getStudent(int rollNo);
   public void addStudent(Student s)
   public void updateStudent(Student stud);
   public void deleteStudent(int rollNo);
}
```

123

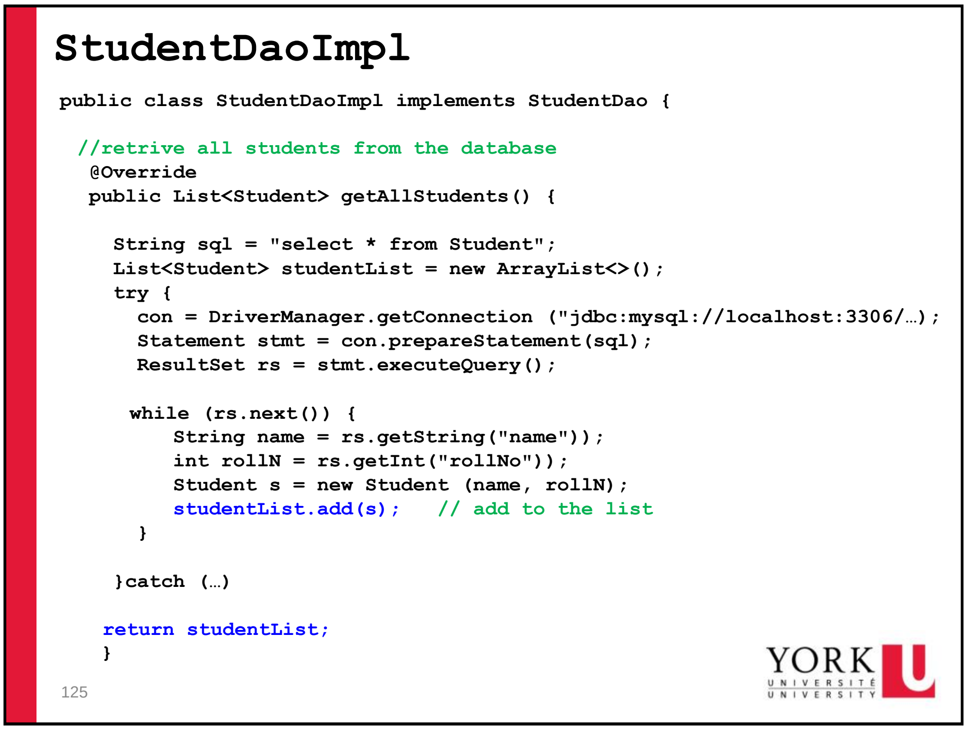
123

StudentDaoImpl

```
public class StudentDaoImpl implements StudentDao {

  public StudentDaoImpl(){
      initialize database connect etc…
  }

  //retrive a student from the database based on rollid
  @Override
  public Student getStudent(int rollNo) {

    String sql = "select * from Student p where p.person_id = ?";
    Student s = null;
    try {
      con = DriverManager.getConnection ("jdbc:mysql://localhost:3306/…);
      Statement stmt = con.prepareStatement(sql);
      stmt.setInt(1, rollNo);
      ResultSet rs = stmt.executeQuery();

     if (rs.next()) {
         String name = rs.getString("name"));
         int rollN = rs.getInt("rollNo"));
         Student s = new Student (name, rollN);  // populate bean
      }
    }catch (…)

     return s;
  }
```

124

124

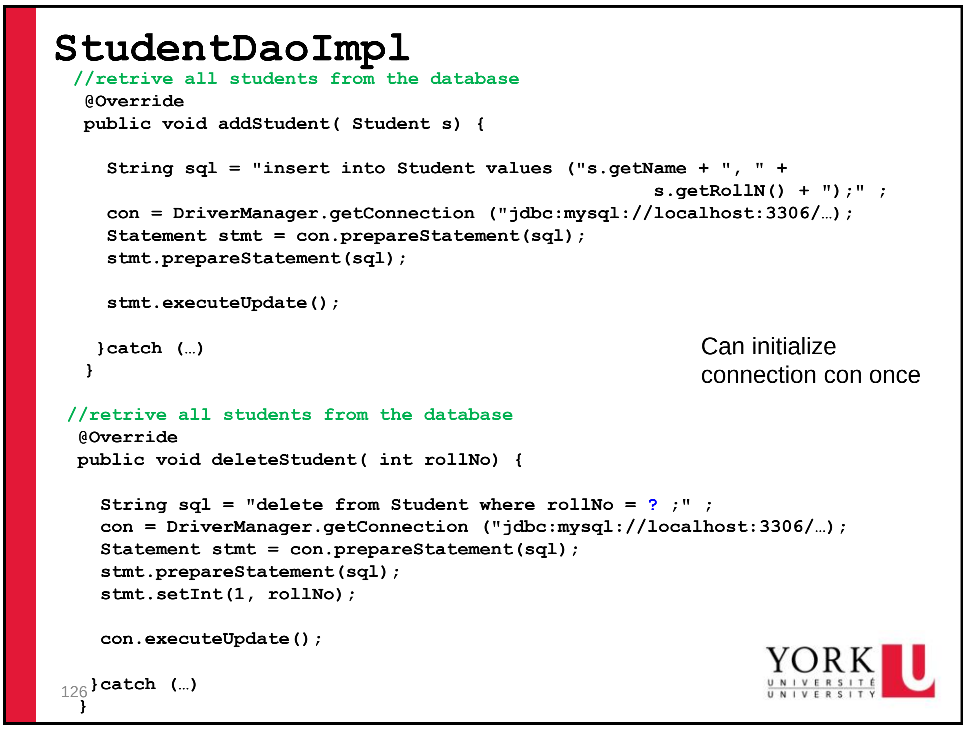


StudentDaoImpl

```
public class StudentDaoImpl implements StudentDao {

  //retrive all students from the database
   @Override
   public List<Student> getAllStudents() {

     String sql = "select * from Student";
     List<Student> studentList = new ArrayList<>();
     try {
       con = DriverManager.getConnection ("jdbc:mysql://localhost:3306/…);
       Statement stmt = con.prepareStatement(sql);
       ResultSet rs = stmt.executeQuery();

      while (rs.next()) {
          String name = rs.getString("name"));
          int rollN = rs.getInt("rollNo"));
          Student s = new Student (name, rollN);
          studentList.add(s);   // add to the list
       }

     }catch (…)

    return studentList;
    }
```

125

125



StudentDaoImpl

```
  //retrive all students from the database
   @Override
   public void addStudent( Student s) {

     String sql = "insert into Student values ("s.getName + ", " +
                                              s.getRollN() + ");" ;
     con = DriverManager.getConnection ("jdbc:mysql://localhost:3306/…);
     Statement stmt = con.prepareStatement(sql);
     stmt.prepareStatement(sql);

     stmt.executeUpdate();

    }catch (…)
   }

  //retrive all students from the database
   @Override
   public void deleteStudent( int rollNo) {

     String sql = "delete from Student where rollNo = ? ;" ;
     con = DriverManager.getConnection ("jdbc:mysql://localhost:3306/…);
     Statement stmt = con.prepareStatement(sql);
     stmt.prepareStatement(sql);
     stmt.setInt(1, rollNo);

     con.executeUpdate();

    }catch (…)
   }
```

Can initialize connection con once

126

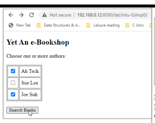
126

DAOPatternDemo
Servelt doGet()

```
      StudentDao studentDao = new StudentDaoImpl();  // ”program by interface”

      //print all students
      for (Student student : studentDao.getAllStudents()) {
         printWriter.println("Student: [RollNo : " + student.getRollNo() +
                  ", Name : " + student.getName() + " ]");
      }


      String rollNo = Integer.parseInt( request.getParameter("id") );
      Student student = studentDao.getStudent(rollNo);
      printWriter.println("Student: [RollNo : " + student.getRollNo() +
                  ", Name : " + student.getName() + " ]");


   }
}
```

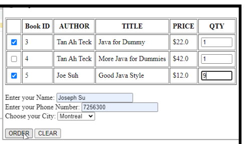
127

127


Yet An e-Bookshop

Choose one or more authors:

✓	Ah Teck
☐	Sue Lee
✓	Joe Suh



	Book ID	AUTHOR	TITLE	PRICE	QTY
✓	3	Tan Ah Teck	Java for Dummy	$22.0	1
☐	4	Tan Ah Teck	More Java for Dummies	$42.0	1
✓	5	Joe Suh	Good Java Style	$12.0	9

Enter your Name: Joseph Su
Enter your Phone Number: 7256300
Choose your City: Montreal

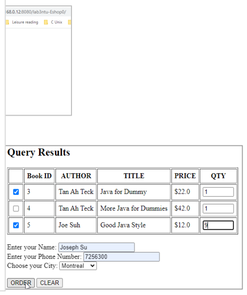
```
protected void doGet(HttpServletRequest request, HttpServletResponse response){

    String authors[] = request.getParameterValues("author");
    // get list of books from database

    String query = "select * from books where author = '"+authors[0]+"'" ;
    for(int i=1; i<authors.length; i++)
          query += " OR author = '" + authors[i]+"'";
    query += ";" ;
    Class.forName("com.mysql.cj.jdbc.Driver");
    Connection con= DriverManager.getConnection("jdbc:mysql://localhost:3306/..);
    Statement stmt=con.createStatement();
    ResultSet rs=stmt.executeQuery(query);

     ArrayList<Book> resu = new ArrayList<>();
     while (rs.next()) {
          int id = rs.getInt("id");
          String title = rs.getString("title");
          String author = rs.getNString("author");
          float price = rs.getFloat("price");
          Book b = new Book (id,  author, title, price);
          resu.add(b);
     }
     request.setAttribute("searchedBooks", resu);
     String target = "bookResultView.jsp";
     request.getRequestDispatcher(target).forward(request, response);
```

Mix with business logic

129

129

```
Create Data Access Object Interface.

BookDao.java

import java.util.List;

public interface BookDao {
   public List<Book> getBooks(Strong authors[]);
   public Book getBookByID(int id);
   public void addBook(Book b)
   public void updateBook(int id, …);
   public void deleteBook(int id);
}
```



Query Results

	Book ID	AUTHOR	TITLE	PRICE	QTY
✓	3	Tan Ah Teck	Java for Dummy	$22.0	1
☐	4	Tan Ah Teck	More Java for Dummies	$42.0	1
✓	5	Joe Suh	Good Java Style	$12.0	9

Enter your Name: Joseph Su
Enter your Phone Number: 7256300
Choose your City: Montreal

ORDER CLEAR

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
                         throws ServletException, IOException {

     String authors[] = request.getParameterValues("author");
     // get list of books from database
     BookDao bd = new BookDaoImp();
     ArrayList<Book> resu = bd.getBooks (authors);


     request.setAttribute("searchedBooks", resu);
     String target = "bookResultView.jsp";
     request.getRequestDispatcher(target).forward(request, response);
```

DAO

130

130

JDBC

- JDBC Introduction
  - Basics
  - Scrollable ResultSet
  - Metadata

- Improve: PreparedStatement

- Web app
  - Improve: Data source

- Serverless rdms: SQLite

- DAO Design pattern

- **SQL injection**

133

133

WEB Security: SQL injection

SQL injection is a technique to maliciously exploit applications that use client-supplied data in SQL statements.

Attackers trick the SQL engine into executing unintended commands by supplying specially crafted string input, thereby gaining unauthorized access to a database to view or manipulate restricted data.

SQL injection techniques all exploit a single vulnerability in the application: Incorrectly validated or non-validated string literals are concatenated into a dynamically built SQL statement and interpreted as code by the SQL

134

134

SQL Injection attacks (or SQLi) alter SQL queries, injecting malicious code by exploiting application vulnerabilities.

Successful SQLi attacks allow attackers to modify database information, access sensitive data, execute admin tasks on the database, and recover files from the system. In some cases attackers can issue commands to the underlying database operating system.

135

135

example1

SQL injection usually occurs when you ask a user for input, like their username/userid, and instead of a name/id, the user gives you an SQL statement that you will **unknowingly** run on your database.

UserId: ____

```
// as int

txtUserId = request.getParameter("UserId");
sql = "SELECT * FROM Users WHERE UserId = " + txtUserId;
stmt.executeQuery (sql);
```

The problem here is that the SQL statement uses concatenation to combine data. The attacker can provide a string like this instead of the pass variable:

UserId: 105 OR 1=1

```
SELECT * FROM Users WHERE UserId = 105 OR 1=1;   // or 4=4  '1'='1'
```

Because 1=1 is a condition that always evaluates to true, the entire WHERE statement will be true, regardless of the username or password provided.
137 The WHERE statement will return all the users in the table

137

example2

SQL injection usually occurs when you ask a user for input, like their username/userid, and instead of a name/id, the user gives you an SQL statement that you will **unknowingly** run on your database.

UserId: ____

```
// as string

txtUserId = request.getParameter("UserId");
sql = "SELECT * FROM Users WHERE UserId = '" + txtUserId + "'";
stmt.executeQuery (sql);
```

UserId: 105 OR 1=1

UserId: abc' OR '5=5

```
SELECT id FROM users WHERE username=‘abc' OR '5=5'
```

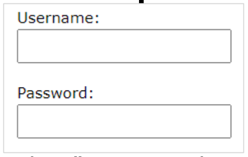
UserId: abc' OR '5'='5

138 SELECT id FROM users WHERE username=‘abc' OR '5'='5'

YORK

138

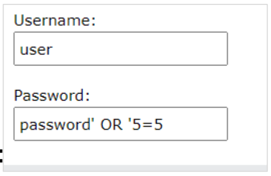
example3



sql = "SELECT id FROM users WHERE username='" + user + "' AND password='" + pass + "'"

The attacker can provide a string like this instead of the pass variable:

password' OR '5=5



The resulting SQL query will be run against the database:

SELECT id FROM users WHERE username='user' AND pass='password' OR '5=5'

Because 5=5 is a condition that always evaluates to true, the entire WHERE statement will be true, regardless of the username or password provided.

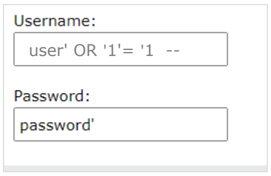
The WHERE statement will return the first ID from the users table, which

140

140

or using comments to even block the rest of the query (there are three types of SQL comments[15]). All three lines have a space at the end:

' OR '1'='1' --
' OR '1'='1' {
' OR '1'='1' /*



user' OR '1'= '1 --

The resulting SQL query will be run against the database:

SELECT id FROM users WHERE username='user' or '1=1' -- AND pass='password'

141

141

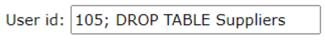
example4

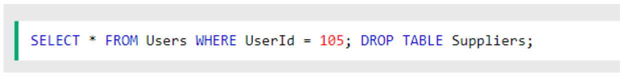
SQL injection usually occurs when you ask a user for input, like their username/userid, and instead of a name/id, the user gives you an SQL statement that you will **unknowingly** run on your database.

UserId:

```
// as int

txtUserId = request.getParameter("UserId");
sql = "SELECT * FROM Users WHERE UserId = " + txtUserId;
stmt.executeQuery (sql);
```

The problem here is that the SQL statement uses concatenation to combine data. The attacker can provide a string like this instead of the pass variable:


User id: 105; DROP TABLE Suppliers



```
SELECT * FROM Users WHERE UserId = 105; DROP TABLE Suppliers;
```

Most databases support batched SQL statement.
A batch of SQL statements is a group of two or more SQL statements, separated by semicolons.

142

142

Mitigations

- **Web application firewalls**
- **Use stored procedure**
- **Escape input**
- **Pattern check**
  Integer, float or boolean, string parameters can be checked if their value is valid representation for the given type. Strings that must follow some strict pattern (date, UUID, alphanumericcan be checked if they match this pattern.

- **Database permissions**
  Limiting the permissions on the database login used by the web application to only what is needed may help reduce the effectiveness of any SQL injection attacks that exploit any bugs in the web application.

- **Parameterized statements Prepared Statements**
  With most development platforms, parameterized statements that work with parameters can be used, instead of embedding user input in the statement. A placeholder can only store a value of the given type and not an arbitrary SQL fragment. Hence the SQL injection would simply be treated as a strange (and probably invalid) parameter value.

enter the userID of tom' or '1'='1, the parameterized query would not be vulnerable and would instead look for a username which literally matched the entire string tom' or '1'=

143

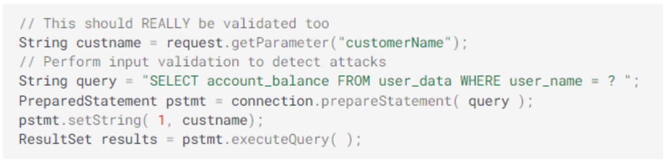
143

Mitigations

“The most important advantage of prepared statements is that they help prevent SQL injection attacks. SQL injection is a technique to maliciously exploit applications that use client-supplied data in SQL statements. Attackers trick the SQL engine into executing unintended commands by supplying specially crafted string input, thereby gaining unauthorized access to a database to view or manipulate restricted data.

Prepared statements always treat client-supplied data as content of a parameter and never as a part of an SQL statement.

- **Parameterized statements Prepared Statements**



```
// This should REALLY be validated too
String custname = request.getParameter("customerName");
// Perform input validation to detect attacks
String query = "SELECT account_balance FROM user_data WHERE user_name = ? ";
PreparedStatement pstmt = connection.prepareStatement( query );
pstmt.setString( 1, custname);
ResultSet results = pstmt.executeQuery( );
```

enter the userName of tom' or '1'='1, the parameterized query would not be vulnerable and would instead look for a username which literally matched the entire string tom' or '1'=