

## Task 1

- How did you use connection pooling?

In our context.xml, we added another Resource tag and defined another DataSource that had a maxTotal of 100, a maxIdle of 30, and a maxWaitMillis of 10000. We named this new DataSource TestDB. In our web.xml, we added a new resource-ref tag of TestDB so our program could refer to this new DataSource. Then, in all of the servlets that opened a database connection, we did a lookup to TestDB instead of moviedb.

- File name, line numbers as in Github  
MovieServlet.java : 162-172  
SingleMovieServlet.java : 46-53  
SingleStarServlet.java : 50 - 58
- Snapshots showing use in your code

```
23     <Resource name="jdbc/TestDB" auth="Container" type="javax.sql.DataSource"
24             maxTotal="100" maxIdle="30" maxWaitMillis="10000" username="mytestuser"
25             password="mypassword" driverClassName="com.mysql.jdbc.Driver"
26             url="jdbc:mysql://localhost:3306/moviedb?autoReconnect=true&useSSL=false&cachePrepStmts=true"/>
27 </Context>

24     <resource-ref>
25         <description>
26             Resource reference to a factory for java.sql.Connection
27             instances that may be used for talking to a particular
28             database that
29             is configured in the server.xml file.
30         </description>
31         <res-ref-name>jdbc/TestDB</res-ref-name>
32         <res-type>javax.sql.DataSource</res-type>
33         <res-auth>Container</res-auth>
34     </resource-ref>
```

```

162     Context envCtx = (Context) initCtx.lookup("java:comp/env");
163     if (envCtx == null)
164         response.getWriter().println("envCtx is NULL");
165
166     // Look up our data source
167     DataSource ds = (DataSource) envCtx.lookup("jdbc/TestDB");
168         //create a connection type to the database
169
170         //start TJ Timer
171     long startTjTime = System.nanoTime();
172     Connection dbcon = ds.getConnection();
173     ---
174
175     Context initCtx = new InitialContext();
176
177     Context envCtx = (Context) initCtx.lookup("java:comp/env");
178     if (envCtx == null)
179         response.getWriter().println("envCtx is NULL");
180
181     // Look up our data source
182     DataSource ds = (DataSource) envCtx.lookup("jdbc/TestDB");
183     ..
184
185     try {
186         Context initCtx = new InitialContext();
187
188     Context envCtx = (Context) initCtx.lookup("java:comp/env");
189     if (envCtx == null)
190         response.getWriter().println("envCtx is NULL");
191
192     // Look up our data source
193     DataSource ds = (DataSource) envCtx.lookup("jdbc/TestDB");
194
195     Context initCtx = new InitialContext();
196
197     Context envCtx = (Context) initCtx.lookup("java:comp/env");
198     if (envCtx == null)
199         out.println("envCtx is NULL");
200
201     // Look up our data source
202     DataSource ds = (DataSource) envCtx.lookup("jdbc/TestDB");

```

- How did you use Prepared Statements?

In our MovieServlet.java, we replaced the query parameters with a question mark (?) and added each query parameter to an ArrayList. We did this because we used multiple if/else statements in constructing our query. Right before executing the query, we declared a Prepared Statement.

We looped through the ArrayList and called `setString` on each query parameter to inject it into the Prepared Statement.

- File name, line numbers as in Github  
src/MovieServlet.java : 67 - 175
- Snapshots showing use in your code

```

67         if(mode.equals("browse")) {
68             String genre_id = request.getParameter("id");
69             if(genre_id != null) { //means that there is an id
70                 base_query += "group by movies.id, movies.title, movies.year, movies.director, r.rating\n" +
71                     "having find_in_set(?, genre_id)\n";
72                 query_items.add(genre_id);
73             }
74             else { //means that they have selected browsing by letter
75                 String first_letter = request.getParameter("search");
76                 //first_letter
77                 first_letter = first_letter + '%';
78                 base_query += "where movies.title like ?\n" +
79                     "group by movies.id, movies.title, movies.year, movies.director, r.rating\n";
80                 query_items.add(first_letter);
81             }
82         }
83     else if(mode.equals("search")) {
84         String search_query = "";
85
86         String search_title = request.getParameter("title");
87         String search_director = request.getParameter("director");
88         String search_year = request.getParameter("year");
89         String search_star = request.getParameter("star");
90
91
92         boolean searchTitleExist = search_title != null && !search_title.isEmpty();
93         boolean searchDirectorExist = search_director != null && !search_director.isEmpty();
94         boolean searchYearExist = search_year != null && !search_year.isEmpty();
95
96
97         boolean searchStarExist = search_star != null && !search_star.isEmpty();
98
99         ArrayList<String> queryList = new ArrayList<String>();
100
101         if(searchTitleExist || searchDirectorExist || searchYearExist || searchStarExist) {
102             search_query += "where ";
103
104             if(searchTitleExist) {
105                 String[] words = search_title.split(" ");
106                 String base_string = "";
107                 for(int i = 0; i < words.length; i++) {
108                     base_string += "+" + words[i] + "*";
109                 }
110
111                 String title_search_query = "match(movies.title) against(? in boolean mode)\n";
112                 queryList.add(title_search_query);
113                 query_items.add(base_string);
114             }
115             if(searchDirectorExist) {
116                 //search_director
117                 search_director = '%' + search_director + '%';
118                 queryList.add("movies.director like ?\n");
119                 query_items.add(search_director);
120             }
121             if(searchYearExist) {
122                 //search_year
123                 search_year = '%' + search_year + '%';
124                 queryList.add("movies.year like ?\n");
125                 query_items.add(search_year);
126             }
127             if(searchStarExist) {
128                 //search_star
129                 search_star = '%' + search_star + '%';
130                 queryList.add("s.name like ?\n");
131                 query_items.add(search_star);
132             }
133         }
134     }

```

```

136         search_query += "and " + queryList.get(i);
137     }
138 }
139
140     base_query += search_query + "group by movies.id, movies.title, movies.year, movies.director, r.rating\n";
141
142 }
143
144     String[] determine_sort = order.split(" ");
145     base_query += "order by " + determine_sort[0];
146     if(determine_sort.length > 1) {
147         if(determine_sort[1].equals("Highest") || determine_sort[1].equals("Z-0")) {
148             base_query += " desc\n";
149         }
150         else if(determine_sort[1].equals("Lowest") || determine_sort[1].equals("0-Z")) {
151             base_query += " asc \n";
152         }
153     }
154     else {
155         base_query += " desc \n";
156     }
157     //limit and page
158     base_query += " limit " + itemLimit + " offset " + page;
159
160     Context initCtx = new InitialContext();
161
162     Context envCtx = (Context) initCtx.lookup("java:comp/env");
163     if (envCtx == null)
164         response.getWriter().println("envCtx is NULL");
165
166     // Look up our data source
167     DataSource ds = (DataSource) envCtx.lookup("jdbc/TestDB");
168     //create a connection type to the database
169
170     //start TJ Timer
171     long startTjTime = System.nanoTime();
172     Connection dbcon = ds.getConnection();
173
174
175     PreparedStatement statement = dbcon.prepareStatement(base_query);

```

## **Task 2**

- Address of AWS and Google instances
- AWS
  - Port 80: <http://52.14.37.54/project1/>
  - Port 8080: <http://52.14.37.54:8080/project1/>
- Google
  - Port 80: <http://35.235.92.4/project1/>
  - Port 8080: <http://35.235.92.4:8080/project1/>
- Have you verified that they are accessible? Does Fablix site get opened both on Google's 80 port and AWS' 8080 port?
- Yes, they are both accessible.

Yes, both Google's 80 port and AWS' 8080 port are accessible.

- Explain how connection pooling works with two backend SQL (in your code)?

Our original instance will send requests to either the master instance or the slave instance. In our apache configuration file, we added a load balancer member for both the master and slave instances. This way, each request will be redirected to either the master or slave instance. We added a Proxy tag to the configuration file, with BalancerMembers inside. We also added a ProxyPass inside the body of the VirtualHost tag.

- File name, line numbers as in Github
  - WEB-CONTENT/META-INF/Context.xml : 6 - 26
  - WEB-CONTENT/WEB-INF/Web.xml : 12 - 24
- Snapshots

```

5      <!-- Defines a Data Source Connecting to localhost moviedb-->
6      <Resource name="jdbc/moviedb"
7          auth="Container"
8          driverClassName="com.mysql.jdbc.Driver"
9          type="javax.sql.DataSource"
10         username="mytestuser"
11         password="mypassword"
12         url="jdbc:mysql://localhost:3306/moviedb"/>
13
14     <Resource name="jdbc/moviedb-write"
15         auth="Container"
16         driverClassName="com.mysql.jdbc.Driver"
17         type="javax.sql.DataSource"
18         username="mytestuser"
19         password="mypassword"
20         url="jdbc:mysql://172.31.32.117:3306/moviedb"/>
21
22
23     <Resource name="jdbc/TestDB" auth="Container" type="javax.sql.DataSource"
24         maxTotal="100" maxIdle="30" maxWaitMillis="10000" username="mytestuser"
25         password="mypassword" driverClassName="com.mysql.jdbc.Driver"
26         url="jdbc:mysql://localhost:3306/moviedb?autoReconnect=true&useSSL=false&cachePrepStmts=true"/>
27
28 </resource-ref>
29
30 <description>MySQL DataSource example</description>
31 <res-ref-name>jdbc/moviedb</res-ref-name>
32 <res-type>javax.sql.DataSource</res-type>
33 <res-auth>Container</res-auth>
34 </resource-ref>
35
36 <resource-ref>
37     <description>MySQL DataSource example</description>
38     <res-ref-name>jdbc/moviedb-write</res-ref-name>
39     <res-type>javax.sql.DataSource</res-type>
40     <res-auth>Container</res-auth>
41 </resource-ref>
42
43 <resource-ref>
44     <description>
45         Resource reference to a factory for java.sql.Connection
46         instances that may be used for talking to a particular
47         database that
48         is configured in the server.xml file.
49     </description>
50     <res-ref-name>jdbc/TestDB</res-ref-name>
51     <res-type>javax.sql.DataSource</res-type>
52     <res-auth>Container</res-auth>
53 </resource-ref>

```



- How read/write requests were routed?
  - In the context XML, we added a new DataSource named moviedb-write to make all write requests go to the master instance. Then, in the servlets that do a write to the database (AddMovieServlet.java, AddStarServlet.java and CheckoutServlet.java), we made the database connection refer to moviedb-write instead of moviedb.
  - File name, line numbers as in Github  
 WEB-CONTENT/WEB-INF/Web.xml : 18 - 23  
 WEB-CONTENT/META-INF/Context.xml : 14 - 20  
 src/AddMovieServlet.java : 66 - 73  
 src/AddStarServlet.java : 53 - 61  
 src/CheckoutServlet.java : 59 - 67
  - Snapshots

```

18 <resource-ref>
19   <description>MySQL DataSource example</description>
20   <res-ref-name>jdbc/moviedb-write</res-ref-name>
21   <res-type>javax.sql.DataSource</res-type>
22   <res-auth>Container</res-auth>
23 </resource-ref>

14 <Resource name="jdbc/moviedb-write"
15     auth="Container"
16     driverClassName="com.mysql.jdbc.Driver"
17     type="javax.sql.DataSource"
18     username="mytestuser"
19     password="mypassword"
20     url="jdbc:mysql://172.31.32.117:3306/moviedb"/>

```



```

53         try {
54             Context initCtx = new InitialContext();
55
56             Context envCtx = (Context) initCtx.lookup("java:comp/env");
57             if (envCtx == null)
58                 response.getWriter().println("envCtx is NULL");
59
60             // Look up our data source
61             DataSource ds = (DataSource) envCtx.lookup("jdbc/moviedb-write");
62
63             Context initCtx = new InitialContext();
64
65             Context envCtx = (Context) initCtx.lookup("java:comp/env");
66             if (envCtx == null)
67                 response.getWriter().println("envCtx is NULL");
68
69             // Look up our data source
70             DataSource ds = (DataSource) envCtx.lookup("jdbc/moviedb-write");
71
72             try {
73                 Context initCtx = new InitialContext();
74
75                 Context envCtx = (Context) initCtx.lookup("java:comp/env");
76                 if (envCtx == null)
77                     response.getWriter().println("envCtx is NULL");
78
79                 // Look up our data source
80                 DataSource ds = (DataSource) envCtx.lookup("jdbc/moviedb-write");

```

### **Task 3**

- Have you uploaded the log files to Github? Where is it located?

Yes, The log files will be in the landing page of the github repository in a folder called "JMeter-Logs", and the directory is cs122b-winter19-team-31/JMeter-Logs.

- Have you uploaded the HTML file (with all sections including analysis, written up) to Github? Where is it located?

Yes, the HTML file along with analysis is included inside the root directory. The directory for this information is [cs122b-winter19-team-31/JMeter-report].

- Have you uploaded the script to Github? Where is it located?

Yes, The script is located within the project1 folder, the directory is  
cs122b-winter19-team-31/

- Have you uploaded the WAR file and README to Github? Where is it located?

Yes, WAR file and README file will be in the initial github repo page :  
cs122b-winter19-team-31/