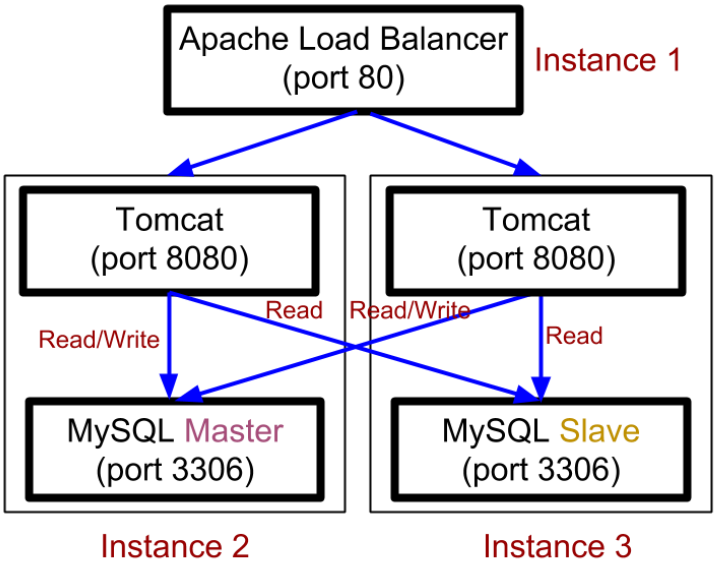
Click the links to access the files in Github

Refer to instance 1, 2, 3 as the following:



Task 1

- How did you use connection pooling?

We enable JDBC connection pooling for all Fabflix Servlet by add configuration in context.xml and web.xml. Besides, we also change corresponding code in all Servlet java files. Below are more details:

1. Copy the JDBC Driver's jar into $CATALINA_HOME/lib(lib folder of Tomcat Server)

2. Add configuration in context.xml: Add a resource (type: DataSource) to the context which includes database username, password, jdbc mysql connection url, and pooling configuration.

3. Add configuration in web.xml: Add a reference to the DataBase just registered in context.xml.

4. Change corresponding code in all servlets: Obtain an environment context -> look up the DataSource just registered and referenced in configuration file -> Use the DataSource object to create a connection to the database.

    - File name, line numbers as in Github
        - context.xml : line 13 - 16

(Path: cs122b-spring18-team-3/Fabflix-website/WebContent/META-INF/context.xml)

- web.xml : line 12 - 17

(Path: cs122b-spring18-team-3/Fabflix-website/WebContent/WEB-INF/web.xml)

- AndroidLoginServlet.java : line 62, 64, 69

(Path: cs122b-spring18-team-3/Fabflix-website/src/AndroidLoginServlet.java)

- CheckoutServlet.java : line 57, 59, 64

(Path: cs122b-spring18-team-3/Fabflix-website/src/CheckoutServlet.java)

- DashboardServlet.java : line 56, 58, 63

(Path: cs122b-spring18-team-3/Fabflix-website/src/DashboardServlet.java)

- DbServlet.java : line 72, 74, 79

(Path: cs122b-spring18-team-3/Fabflix-website/src/DbServlet.java)

- IndexServlet.java : line 34, 36, 41

(Path: cs122b-spring18-team-3/Fabflix-website/src/IndexServlet.java)

- LoginServlet.java : line 58, 60, 66

(Path: cs122b-spring18-team-3/Fabflix-website/src/LoginServlet.java)

- StarServlet.java : line 38, 40, 51

(Path: cs122b-spring18-team-3/Fabflix-website/src/StarServlet.java)

- autoCompleteServlet.java : line 44, 46

(Path: cs122b-spring18-team-3/Fabflix-website/src/autoCompleteServlet.java)

- Snapshots showing use in your code

context.xml:

```
13      <Resource name="jdbc/localDB" auth="Container" type="javax.sql.DataSource"
14                maxTotal="100" maxIdle="30" maxWaitMillis="10000" username="root"
15                password="mm941026" driverClassName="com.mysql.jdbc.ReplicationDriver"
16                url="jdbc:mysql:replication://172.31.18.8:3306,172.31.28.80:3306/moviedb?autoReconnect=true&amp;useSSL=false&amp;
17
18
```

```
r" type="javax.sql.DataSource"
tMillis="10000" username="root"
e="com.mysql.jdbc.ReplicationDriver"
31.18.8:3306,172.31.28.80:3306/moviedb?autoReconnect=true&amp;useSSL=false&amp;cachePrepStmts=true&amp;roundRobinLoadBalance=true"/>
```

web.xml:

```
    </welcome-file-list>
    <resource-ref>
      <description>MySQL DataSource example</description>
      <res-ref-name>jdbc/localDB</res-ref-name>
      <res-type>javax.sql.DataSource</res-type>
      <res-auth>Container</res-auth>
    </resource-ref>
```

AndroidLoginServlet.java:

```
        try {
            // the following few lines are for connection pooling
            // Obtain our environment naming context

            Context initCtx = new InitialContext();

            Context envCtx = (Context) initCtx.lookup("java:comp/env");
            if (envCtx == null)
                System.out.println("envCtx is NULL");

            // Look up our data source
            DataSource ds = (DataSource) envCtx.lookup("jdbc/localDB");

            if (ds == null)
```

CheckoutServlet.java:

```
50                  // Output stream to STDOUT
51                  PrintWriter out = response.getWriter();
52
53                  try {
54              // the following few lines are for connection pooling
55              // Obtain our environment naming context
56
57              Context initCtx = new InitialContext();
58
59              Context envCtx = (Context) initCtx.lookup("java:comp/env");
60              if (envCtx == null)
61                  out.println("envCtx is NULL");
62
63              // Look up our data source
64              DataSource ds = (DataSource) envCtx.lookup("jdbc/localDB");
65
66              if (ds == null)
67                  System.out.println("ds is null.");
68
69              Connection dbcon = ds.getConnection();
70              dbcon.setReadOnly(true);
71              if (dbcon == null)
72                  System.out.println("dbcon is null.");
```

DashboardServlet.java:

```
48
49                  // Output stream to STDOUT
50                  PrintWriter out = response.getWriter();
51                  JsonArray jsonArray = new JsonArray();
52                  try {
53              // the following few lines are for connection pooling
54              // Obtain our environment naming context
55
56              Context initCtx = new InitialContext();
57
58              Context envCtx = (Context) initCtx.lookup("java:comp/env");
59              if (envCtx == null)
60                  out.println("envCtx is NULL");
61
62              // Look up our data source
63              DataSource ds = (DataSource) envCtx.lookup("jdbc/localDB");
64
65              // the following commented lines are direct connections without pooling
66              //Class.forName("org.gjt.mm.mysql.Driver");
67              //Class.forName("com.mysql.jdbc.Driver").newInstance();
68              //Connection dbcon = DriverManager.getConnection(loginUrl, loginUser, loginPasswd);
69
70              if (ds == null)
71                  System.out.println("ds is null.");
```

DbServlet.java:

```
66
67                  // Output stream to STDOUT
68                  PrintWriter out = response.getWriter();
69
70                  try {
71
72              Context initCtx = new InitialContext();
73
74              Context envCtx = (Context) initCtx.lookup("java:comp/env");
75              if (envCtx == null)
76                  out.println("envCtx is NULL");
77
78              // Look up our data source
79              DataSource ds = (DataSource) envCtx.lookup("jdbc/localDB");
80
81              if (ds == null)
82                  out.println("ds is null.");
83
84              Connection dbcon = ds.getConnection();
85              if (dbcon == null)
```

IndexServlet.java:

```
27                  // Output stream to STDOUT
28                  PrintWriter out = response.getWriter();
29
30                  try {
31              // the following few lines are for connection pooling
32              // Obtain our environment naming context
33
34              Context initCtx = new InitialContext();
35
36              Context envCtx = (Context) initCtx.lookup("java:comp/env");
37              if (envCtx == null)
38                  out.println("envCtx is NULL");
39
40              // Look up our data source
41              DataSource ds = (DataSource) envCtx.lookup("jdbc/localDB");
42
43              if (ds == null)
44                  System.out.println("ds is null.");
45
46              Connection dbcon = ds.getConnection();
```

LoginServlet.java:

```
52          /  in the real project, you should talk to the database to verify username/password
53          */
54          try {
55              // the following few lines are for connection pooling
56              // Obtain our environment naming context
57
58              Context initCtx = new InitialContext();
59
60              Context envCtx = (Context) initCtx.lookup("java:comp/env");
61              if (envCtx == null)
62                  out.println("envCtx is NULL");
63
64              // Look up our data source
65              System.out.println("Start to look up database");
66              DataSource ds = (DataSource) envCtx.lookup("jdbc/localDB");
67              System.out.println("Lookup successfully");
68
69              if (ds == null)
```

StarServlet.java:

```
31                  // Output stream to STDOUT
32                  PrintWriter out = response.getWriter();
33
34                  try {
35              // the following few lines are for connection pooling
36              // Obtain our environment naming context
37
38              Context initCtx = new InitialContext();
39
40              Context envCtx = (Context) initCtx.lookup("java:comp/env");
41              if (envCtx == null)
42                  out.println("envCtx is NULL");
43
44              // Look up our data source
45              DataSource ds = (DataSource) envCtx.lookup("jdbc/localDB");
46
47              if (ds == null)
48                  System.out.println("ds is null.");
49
50              Connection dbcon = ds.getConnection();
```

autoCompleteServlet.java:

```
34
35                      // return the empty json array if query is null or empty
36                      if (titleQuery == null || titleQuery.trim().isEmpty()) {
37                              response.getWriter().write(jsonArray.toString());
38                              return;
39                      }
40
41              // the following few lines are for connection pooling
42              // Obtain our environment naming context
43
44              Context initCtx = new InitialContext();
45
46              Context envCtx = (Context) initCtx.lookup("java:comp/env");
47              if (envCtx == null)
48                  System.out.println("envCtx is NULL");
49
50              // Look up our data source
51              DataSource ds = (DataSource) envCtx.lookup("jdbc/localDB");
52
53
54              if (ds == null)
55                  System.out.println("ds is null.");
56
57              Connection dbcon = ds.getConnection();
58              dbcon.setReadOnly(true);
59
```

- How did you use Prepared Statements?

We use Prepared Statement for all queries with different parameters in the file: Root of the repository >> Fabflix-website >> src >> DbServlet.java. We put a placeholder ? in the Prepared Statement, so the Prepared Statement will just be compiled once. Then it can be repeatedly used by setting different parameters in the Prepared Statement. As a result, it improves the performance. When we combine connection pooling with Prepared Statement(associated with one connection, we set the JDBC MySQL URL(in context.xml file) to:

url="jdbc:mysql:replication://172.31.18.8:3306,172.31.28.80:3306/moviedb?autoReconnect=true&useSSL=false&cachePrepStmts=true&roundRobinLoadBalance=true"/>

This url is added with property: cachePrepStmts=true, when we use the Prepared Statement with connection pooling.

Example of use:

1. First import java.sql.PreparedStatement
2. Initiate a PreparedStatement instance by SQL Connection instance and a query: PreparedStatement ps = connection.prepareStatement(query);
3. Set different by parameter to the Prepared Statement by calling setInt(), setString(), and more.
4. Execute the Prepared Statement:

ResultSet rs = preparedStatement.executeQuery();

- File name, line numbers as in Github
  In DbServlet.java , the line number starts from line 92 ~ 385
  - Line 93
  - Line 100, 101
  - Line 115, 116, 117, 120
  - Line 126
  - Line 151, 152, 153
  - Line 157
  - Line 167
  - Line 184
  - Line 189, 190, 191, 192
  - Line 198, 199, 200, 201
  - Line 218, 219, 220, 221, 223, 224, 225, 228, 229
  - Line 236, 237, 238
  - Line 264, 265, 266
  - Line 270, 271, 272
  - Line 276, 277
  - Line 288
  - Line 318, 319
  - Line 323
  - Line 345, 346
  - Line 350
  - Line 371, 372
  - Line 383, 384

- Snapshots showing use in your code (from DbServlet)
  - Highlighted yellow parts are the prepared statement

```
91
92              dbcon.setReadOnly(true);
93                      PreparedStatement preparedStatementCount;
94                      String queryCount = "";
95
96                      if (!genreQuery.equals("")) {
97                              queryCount = "select count(*) as total from " +
98                                              "movies, genres, genres_in_movies where movies.id=genres_in_movies.movieId "
99                                              + "and genres_in_movies.genreId=genres.id and genres.name=?;";
100                             preparedStatementCount = dbcon.prepareStatement(queryCount);
101                             preparedStatementCount.setString(1, genreQuery);
102                     }
103                     else if (!starQuery.equals("")) {
104                             queryCount = "select count(*) as total " +
105                     "from stars " +
106                     "inner join stars_in_movies " +
107                     "on stars.id=stars_in_movies.starId " +
108                     "inner join movies " +
109                     "on stars_in_movies.movieId=movies.id " +
110                     "where lower(name) like ? AND " +
111                     "lower(title) like ? AND " +
112                     "lower(director) like ?" +
113                     (yearQuery.equals("") ? ";" : (" AND year=?;")); // string format args
114
115                             preparedStatementCount = dbcon.prepareStatement(queryCount);
116                             preparedStatementCount.setString(1, starQuery.toLowerCase() + "%");
117                             preparedStatementCount.setString(2, titleQuery.toLowerCase() + "%");
118                             preparedStatementCount.setString(3, directorQuery.toLowerCase() + "%");
119                             if (!yearQuery.equals("")) {
120                                     preparedStatementCount.setInt(4, Integer.parseInt(yearQuery));
121                             }
122
123                     }
124                     else if (titleQuery.equals("") && directorQuery.equals("") && yearQuery.equals("")){
125                             queryCount = "select count(*) as total from movies;";
126                             preparedStatementCount = dbcon.prepareStatement(queryCount);
127                     }
```

```
125                             queryCount = "select count(*) as total from movies;";
126                             preparedStatementCount = dbcon.prepareStatement(queryCount);
127                         }
128                         else {
129 //                              int titleLength = titleQuery.length();
130 //                              int threshold = (int) Math.floor(titleLength * 0.4);
131 
132                             queryCount = "select count(*) as total from movies " +
133                                                     "where MATCH(title) against (? IN BOOLEAN MODE) AND " +
134                                                     "lower(director) like ?" +
135                                                     (yearQuery.equals("") ? ";" : (" AND year=?;"));
136 //                              fuzzy search query
137 //                              queryCount = "select count(*) as total from movies " +
138 //                                      "where (MATCH(title) against (? IN BOOLEAN MODE) or edth(lower(title), ?, ?))
139 //                                      "lower(director) like ?" +
140 //                                      (yearQuery.equals("") ? ";" : (" AND year=?;"));
141 
142                             String titleMatchPattern = "";
143                             for (String token : tokenArray) {
144                                     titleMatchPattern += "+" + token.trim() + "* ";
145                             }
146                             titleMatchPattern = titleMatchPattern.trim();
147 
148                             // For test
149                             System.out.println("title match pattern: " + titleMatchPattern);
150 
151                             preparedStatementCount = dbcon.prepareStatement(queryCount);
152                             preparedStatementCount.setString(1, titleMatchPattern);
153                             preparedStatementCount.setString(2, directorQuery.toLowerCase() + "%");
154 
155 
156                             if (!yearQuery.equals("")) {
157                                     preparedStatementCount.setInt(3, Integer.parseInt(yearQuery));
158                             }
159                         }
160 
```

```
160
161                     // For test
162                     System.out.println("queryCount: " + queryCount);
163
164
165                     long jdbcStartTime1 = System.nanoTime();
166                     //#########################################################################
167                     ResultSet rsCount = preparedStatementCount.executeQuery();
168                     //#########################################################################
169                     long jdbcEndTime1 = System.nanoTime();
170                     jdbcSum += jdbcEndTime1 - jdbcStartTime1;
171
172
173                     int counter = 0;
174                     while (rsCount.next()) {
175                             counter = rsCount.getInt("total");
176                     }
177
178                     if (!idQuery.equals(""))
179                             counter = 1;
180
181                     System.out.println("Counter: " + counter);
182
183                     String query = "";
184                     PreparedStatement preparedStatement;
185
186                     if (!idQuery.equals("")) {
187                             query = "select movies.id as movieId, title, year, director, rating from movies left join ratin
188                                     "where movies.id=? order by rating desc limit ?, ?;";
189                             preparedStatement = dbcon.prepareStatement(query);
190                             preparedStatement.setString(1, idQuery);
191                             preparedStatement.setInt(2, pageNumber);
192                             preparedStatement.setInt(3, movieNumber);
193                     }
194                     else if (!genreQuery.equals("")){
195                             query = String.format("select movies.id as movieId, title, year, director, rating from genres,
196                                     "where movies.id=genres_in_movies.movieId " +
197                                     "and genres_in_movies.genreId=genres.id and genres.name=? order by %s limit ?,
198                             preparedStatement = dbcon.prepareStatement(query);
199                             preparedStatement.setString(1, genreQuery);
200                             preparedStatement.setInt(2, pageNumber);
201                             preparedStatement.setInt(3, movieNumber);
202                     }
203                     else if (!starQuery.equals("")) {
```

```
203                     else if (!starQuery.equals("")) {
204                             query = String.format("select movies.id as movieId, title, year, director, rating " +
205                     "from stars " +
206                     "inner join stars_in_movies " +
207                     "on stars.id=stars_in_movies.starId " +
208                     "inner join movies " +
209                     "on stars_in_movies.movieId=movies.id " +
210                     "left join ratings " +
211                     "on movies.id=ratings.movieId " +
212                     "where lower(name) like ? AND " +
213                     "lower(title) like ? AND " +
214                     "lower(director) like ?" +
215                     (yearQuery.equals("") ? "" : (" AND year=?" )) +
216                     " order by %s limit ?, ?;", sortBy);
217     //                (("'" + starQuery + "%'"), ("'" + titleQuery +"%'"), ("'" + directorQuery+ "%'"), sortBy); // string f
218                             preparedStatement = dbcon.prepareStatement(query);
219                             preparedStatement.setString(1, starQuery.toLowerCase() + "%");
220                             preparedStatement.setString(2, titleQuery.toLowerCase() + "%");
221                             preparedStatement.setString(3, directorQuery.toLowerCase() + "%");
222                             if (!yearQuery.equals("")) {
223                                     preparedStatement.setInt(4, Integer.parseInt(yearQuery));
224                                     preparedStatement.setInt(5, pageNumber);
225                                     preparedStatement.setInt(6, movieNumber);
226                             }
227                             else {
228                                     preparedStatement.setInt(4, pageNumber);
229                                     preparedStatement.setInt(5, movieNumber);
230                             }
231
232                     }
233                     else if (titleQuery.equals("") && directorQuery.equals("") && yearQuery.equals("")) {
234                             query = String.format("select movies.id as movieId, title, year, director, rating from movies l
235                                             "order by %s limit ?, ?;", sortBy);
236                             preparedStatement = dbcon.prepareStatement(query);
237                             preparedStatement.setInt(1, pageNumber);
238                             preparedStatement.setInt(2, movieNumber);
239                     }
240                     else {
241                             int titleLength = titleQuery.length();
242                             int threshold = (int) Math.floor(titleLength * 0.4);
243
```

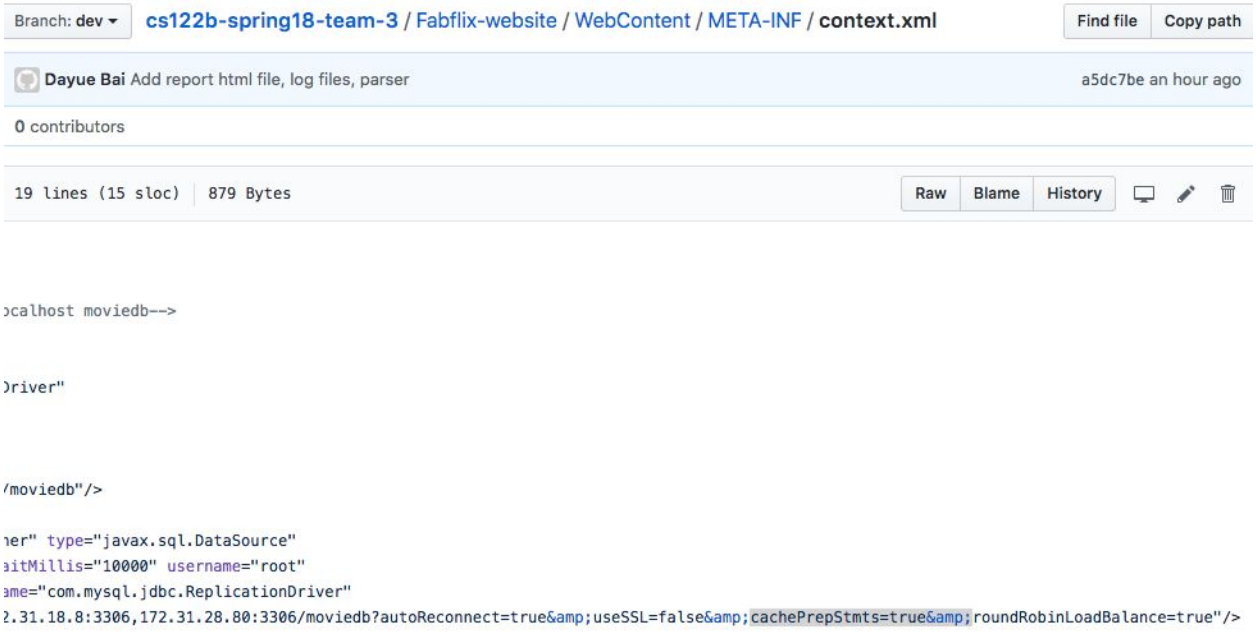
```
243
244                         query = String.format("select movies.id as movieId, title, year, director, rating from movies 
245                                 "where MATCH(title) against (? IN BOOLEAN MODE) AND " +
246                                 "lower(director) like ?" +
247                                 (yearQuery.equals("") ? "" : (" AND year=?" )) + " order by %s limit ?, ?;", s
248
249 //                          fuzzy search query
250 //                          query = String.format("select movies.id as movieId, title, year, director, rating from movies 
251 //                                  "where (MATCH(title) against (? IN BOOLEAN MODE) or edth(lower(title), ?, ?)) 
252 //                                  "lower(director) like ?" +
253 //                                  (yearQuery.equals("") ? "" : (" AND year=?" )) + " order by %s limit ?, ?;", s
254
255                         String titleMatchPattern = "";
256                         for (String token : tokenArray) {
257                                 titleMatchPattern += "+" + token.trim() + "* ";
258                         }
259                         titleMatchPattern = titleMatchPattern.trim();
260
261                         // For test
262                         System.out.println("title match pattern: " + titleMatchPattern);
263
264                         preparedStatement = dbcon.prepareStatement(query);
265                         preparedStatement.setString(1, titleMatchPattern);
266                         preparedStatement.setString(2, directorQuery.toLowerCase() + "%");
267
268
269                         if (!yearQuery.equals("")) {
270                                 preparedStatement.setInt(3, Integer.parseInt(yearQuery));
271                                 preparedStatement.setInt(4, pageNumber);
272                                 preparedStatement.setInt(5, movieNumber);
273
274                         }
275                         else {
276                                 preparedStatement.setInt(3, pageNumber);
277                                 preparedStatement.setInt(4, movieNumber);
278                         }
279                 }
280
281                 // For test
282                 System.out.println("Acutal sent query: " + query);
283
```

```
282                     System.out.println("Acutal sent query: " + query);
283
284
285                     // Perform the query
286                     long jdbcStartTime2 = System.nanoTime();
287                     //*****************************************************
288                     ResultSet rs = preparedStatement.executeQuery();
289                     //*****************************************************
290                     long jdbcEndTime2 = System.nanoTime();
291                     jdbcSum += jdbcEndTime2 - jdbcStartTime2;
292
293                     JsonArray jsonArray = new JsonArray();
294                     // Iterate through each row of rs
295
```

```
314                         jsonObject.addProperty("movieRating", movieRating);
315
316                 String query_star = "select starId, name from stars, stars_in_movies where stars.id=stars_in_movies.sta
317                         "stars_in_movies.movieId=?;";
318                 PreparedStatement preparedStatementStar = dbcon.prepareStatement(query_star);
319                 preparedStatementStar.setString(1, movieId);
320
321                 long jdbcStartTime3 = System.nanoTime();
322                 //*********************************************************
323                 ResultSet resultSetStar = preparedStatementStar.executeQuery();
324                 //*********************************************************
325                 long jdbcEndTime3 = System.nanoTime();
326                 jdbcSum += jdbcEndTime3 - jdbcStartTime3;
327
328                 while (resultSetStar.next()) {
329                         String starName = resultSetStar.getString("name");
330                         String starId = resultSetStar.getString("starId");
331                         if (! resultSetStar.isLast()) {
332                                 listofIds += starId +",";
333                                 listofStars += starName + ",";
334                         }
```

```
342
343                     String query_genre = "select name from genres, genres_in_movies where genres.id=genres_in_movies.genre
344                                     "genres_in_movies.movieId=?;";
345                     PreparedStatement preparedStatementGenre = dbcon.prepareStatement(query_genre);
346                     preparedStatementGenre.setString(1, movieId);
347
348                     long jdbcStartTime4 = System.nanoTime();
349                     //*****************************************************************
350                     ResultSet resultSetGenre = preparedStatementGenre.executeQuery();
351                     //*****************************************************************
352                     long jdbcEndTime4 = System.nanoTime();
353                     jdbcSum += jdbcEndTime4 - jdbcStartTime4;
354
355                     while (resultSetGenre.next()) {
356                             String genreName = resultSetGenre.getString("name");
357                             if (! resultSetGenre.isLast())
358                                     listofGenres += genreName + ",";
359                             else
360                                     listofGenres += genreName;
361                     }
362                     jsonObject.addProperty("listofGenres", listofGenres);
363
364                             jsonArray.add(jsonObject);
365
366                             if (rs.isLast())
367                                     ((JsonObject) jsonArray.get(0)).addProperty("totalFound", counter);
368
369                             resultSetStar.close();
370                             resultSetGenre.close();
371                             preparedStatementStar.close();
372                             preparedStatementGenre.close();
373                     }
374
375             // write JSON string to output
376             out.write(jsonArray.toString());
377             // set response status to 200 (OK)
378             response.setStatus(200);
379
380                     // Close open resources
381                     rsCount.close();
382                     rs.close();
383                     preparedStatementCount.close();
384                     preparedStatement.close();
385                     dbcon.close();
386                 } catch (Exception e) {
387                     // write error message JSON object to output
388                     JsonObject jsonObject = new JsonObject();
389                     jsonObject.addProperty("errorMessage", e.getMessage());
```

- Snapshots showing use in context.xml: (used when the Prepared Statement is used with connection pooling)
- Set cachePrepStmts property to true



Task 2

- Address of AWS and Google instances

  Google Cloud Platform public IP: 104.198.4..5
  AWS Instance 1 public IP: 18.188.249.178
  AWS Instance 2 public IP: 13.59.209.92
  AWS Instance 3 public IP: 18.222.83.5

- Have you verified that they are accessible? Does Fablix site get opened both on Google's 80 port and AWS' 8080 port?

  We have verified they are accessible. The Fabflix website get opened on both the Google' 80 port and the AWS's 8080 port.

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

  We enable connection pooling to work with backend SQL by doing the following procedures.

- Create two databases respectively on master and slave instance. Ensure data is synchronized.

- Install an apache2 server on AWS Instance. This server serves as an load balancer which allocates requests to both AWS Instance 2 and Instance 3. We add some configuration to 000-default.conf file. Add proxy and rules for our Fabflix website.
- Enable sticky session configuration for apache 2 server to make the session persists over requests of the same client.
- For connection pooling, we only add one resource tag to context.xml, because we use JDBC Replication Driver to control read and write operations (more details will be covered in next section: How read/write are routed). We register databases on both master and slave instance in the JDBC Replication Driver. In this way, after load balancer allocate request to the backend Tomcat server, write requests will only be sent to database of master instance, while read requests can be sent to one of the database on backend instances.

    - File name, line numbers as in Github
        - context.xml : line 13 - 16

(Path: cs122b-spring18-team-3/Fabflix-website/WebContent/META-INF/context.xml)

        - 000-default.conf in apache2 server in instance 1

    - Snapshots

context.xml:

19 lines (15 sloc) | 879 Bytes — Raw | Blame | History

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <Context>
4      <!-- Defines a Data Source Connecting to localhost moviedb-->
5      <Resource name="jdbc/moviedb"
6                auth="Container"
7                driverClassName="com.mysql.jdbc.Driver"
8                type="javax.sql.DataSource"
9                username="root"
10               password="mm941026"
11               url="jdbc:mysql://localhost:3306/moviedb"/>
12
13     <Resource name="jdbc/localDB" auth="Container" type="javax.sql.DataSource"
14               maxTotal="100" maxIdle="30" maxWaitMillis="10000" username="root"
15               password="mm941026" driverClassName="com.mysql.jdbc.ReplicationDriver"
16               url="jdbc:mysql:replication://172.31.18.8:3306,172.31.28.80:3306/moviedb?autoReconnect=true&amp;useSSL=false&amp;
17
18
19 </Context>
```

```
r" type="javax.sql.DataSource"
tMillis="10000" username="root"
e="com.mysql.jdbc.ReplicationDriver"
31.18.8:3306,172.31.28.80:3306/moviedb?autoReconnect=true&amp;useSSL=false&amp;cachePrepStmts=true&amp;roundRobinLoadBalance=true"/>
```

000-default.conf in instance 1:

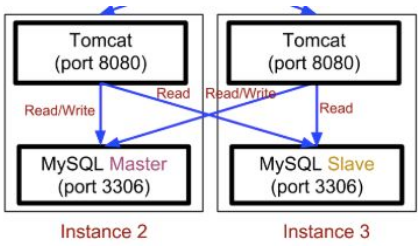
```
Header add Set-Cookie "ROUTEID=.%{BALANCER_WORKER_ROUTE}e; path=/" env=BALANCER_ROUTE_CHANGED
<Proxy "balancer://TomcatTest_balancer">
    BalancerMember "http://172.31.18.8:8080/TomcatTest/"
    BalancerMember "http://172.31.28.80:8080/TomcatTest/"
</Proxy>
#<Proxy "balancer://Session_balancer">
#       BalancerMember "http://172.31.18.8:8080/Session" route=1
#       BalancerMember "http://172.31.28.80:8080/Session" route=2
#       ProxySet stickysession=ROUTEID
#</Proxy>
<Proxy "balancer://Fabflix_balancer">
        BalancerMember "http://172.31.18.8:8080/Fabflix-website" route=1
        BalancerMember "http://172.31.28.80:8080/Fabflix-website" route=2
        ProxySet stickysession=ROUTEID
</Proxy>

<VirtualHost *:80>
        # Two new rules for Session example
        #ProxyPass /Session balancer://Session_balancer
        #ProxyPassReverse /Session balancer://Session_balancer

        # Two new rules for TomcatTest example
        ProxyPass /TomcatTest balancer://TomcatTest_balancer
        ProxyPassReverse /TomcatTest balancer://TomcatTest_balancer
        # Two new rules for Fabflix project
        ProxyPass /Fabflix-website balancer://Fabflix_balancer
        ProxyPassReverse /Fabflix-website balancer://Fabflix_balancer

        # The ServerName directive sets the request scheme, hostname and port that
```

- How read/write requests were routed?

  In context.xml, instead of adding two resource tags for databases on both master and slave instance, we just add one resource that use JDBC Replication Driver, in which we register master and slave database url and port. We set the Replication Driver connection url in this format: (set property: roundRobinLoadBalance = true)

jdbc:mysql:replication://[master host][:port],[slave host 1][:port][,[slave host 2][:port]]...[/[database]] »[?propertyName1=propertyValue1[&propertyName2=propertyValue2]...]

We enable round-robin scheme for the connection, in which read requests(such as select) can be sent to one of the backend database, if we call Connection.setReadOnly(true) in Java servlet. ReplicationDriver will ensure write requests(update, insert, delete,...) will only be sent to the database on master instance (AWS Instance 2), if we call Connection.setReadOnly(false) in Java servlet. In this way, we can realize the replication of data on two backend databases in a proper way. The route shown below can be realized.



- File name, line numbers as in Github

  DbServlet.java : line 92
  (Path: cs122b-spring18-team-3/Fabflix-website/src/DbServlet.java)

  CheckoutServlet.java : line 110, 115
  (Path: cs122b-spring18-team-3/Fabflix-website/src/CheckoutServlet.java)

  LoginServlet.java : line 79
  (Path: cs122b-spring18-team-3/Fabflix-website/src/LoginServlet.java)

  StarServlet.java : line 51
  (Path: cs122b-spring18-team-3/Fabflix-website/src/StarServlet.java)

  Context.xml : line 15, 16
  (Path: cs122b-spring18-team-3/Fabflix-website/WebContent/META-INF/context.xml)

- Snapshots

DbServlet:

```
85              if (dbcon == null)
86                  out.println("dbcon is null.");
87
88                          // Get a connection from dataSource
89  //                      Connection dbcon = dataSource.getConnection();
90
91
92              dbcon.setReadOnly(true);
93                          PreparedStatement preparedStatementCount;
94                          String queryCount = "";
95
96                          if (!genreQuery.equals("")) {
97                                  queryCount = "select count(*) as total from " +
98                                                  "movies, genres, genres_in_movies where movies.id=genres_in_movies.
99                                                  + "and genres_in_movies.genreId=genres.id and genres.name=?;";
100                                 preparedStatementCount = dbcon.prepareStatement(queryCount);
101                                 preparedStatementCount.setString(1, genreQuery);
102                         }
103                         else if (!starQuery.equals("")) {
```

CheckoutServlet.java

```
99                      PreparedStatement preparedUpdateStatement = dbcon.prepareStatement(updateQuery);
100                     preparedUpdateStatement.setInt(1, Integer.parseInt(customerId));
101
102                     for (String movieId : cartMap.keySet())
103                     {
104                             preparedUpdateStatement.setString(2, movieId);
105                             int amount = ((User) session.getAttribute("user")).getItemAmount(movieId);
106
107                             for (int counter = 0; counter < amount; ++counter) {
108
109                             // Update sales table
110                                     dbcon.setReadOnly(false);
111                             preparedUpdateStatement.executeUpdate();
112
113                             // Get last inserted sale ID
114                             Statement idQueryStatement = dbcon.createStatement();
115                             dbcon.setReadOnly(true);
116                             String idQuery = "select LAST_INSERT_ID() as id;";
117                             ResultSet rs = idQueryStatement.executeQuery(idQuery);
118
119                             while (rs.next()) {
120                                     int saleId = rs.getInt("id");
121
122                                     // Write the transaction to user purchase record, stored in User.java class
123                                     ((User) session.getAttribute("user")).writePurchaseRecord(saleId, movieId);
124
```

LoginServlet.java

```
72              Connection dbCon = ds.getConnection();
73              if (dbCon == null)
74                  System.out.println("dbcon is null.");
75
76              String username = "";
77              String password = "";
78              String query = "";
79              dbCon.setReadOnly(true);
80              PreparedStatement preparedStatement;
81
82              System.out.println("Starting to make prepareStatement");
83              // Generate a SQL query
84              if (request.getParameter("employee_email")==null && request.getParameter("e
85                      username = request.getParameter("email");
```

StarServlet.java

```
44              // Look up our data source
45              DataSource ds = (DataSource) envCtx.lookup("jdbc/localDB");
46
47              if (ds == null)
48                  System.out.println("ds is null.");
49
50              Connection dbcon = ds.getConnection();
51              dbcon.setReadOnly(true);
52              if (dbcon == null)
53                  System.out.println("dbcon is null.");
54
55                      // Construct query
56                      String query = "select id, name, birthYear from stars where id=?;";
57                      PreparedStatement preparedStatement = dbcon.prepareStatement(query);
58                      preparedStatement.setString(1, starId);
59
60                      // For test
```

context.xml:

```
6               auth="Container"
7               driverClassName="com.mysql.jdbc.Driver"
8               type="javax.sql.DataSource"
9               username="root"
10              password="mm941026"
11              url="jdbc:mysql://localhost:3306/moviedb"/>
12
13      <Resource name="jdbc/localDB" auth="Container" type="javax.sql.DataSource"
14              maxTotal="100" maxIdle="30" maxWaitMillis="10000" username="root"
15              password="mm941026" driverClassName="com.mysql.jdbc.ReplicationDriver"
16              url="jdbc:mysql:replication://172.31.18.8:3306,172.31.28.80:3306/moviedb?autc
17
18
19  </Context>
```

```
ue&amp;useSSL=false&amp;cachePrepStmts=true&amp;roundRobinLoadBalance=true"/>
```

Task 3

- Have you uploaded the log files to Github? Where is it located?
  Yes, we have uploaded the log files to Github.
  It is located at: /Fabflix-website/target/ . There are 9 log files in total and they are identified by the task shown in project 5 requirement. The names are followed the order shown in project requirement.

- Have you uploaded the HTML file (with all sections including analysis, written up) to Github? Where is it located?
  Yes, we have uploaded the HTML(with all sections including analysis and written up) to Github.
  It is located at: /Fabflix-website/Project 5-Task 3-Report/ . The name is jmeter_report.html.

- Have you uploaded the script to Github? Where is it located?
  Yes, we have uploaded the script to Github(at the root of the WAR file)
  Path of parser.py file: Root of repository >> Fabflix-website >> target >> parser.py

- Have you uploaded the WAR file and README to Github? Where is it located?
  Yes, we have uploaded the WAR file and README to Github.

  Path of Fabflix-website.war file:
  Root of repository >> Fabflix-website >> target >> Fabflix-website.war

  README.md file is located at the root of our Github repository