Team: **1** Authors: Kelly McKeown, David Hopkins Project: **5**

==========

Task 1

- **How did you use connection pooling?**

  Connection pooling was configured in /META-INF/context.xml, and implemented in /src/fabflix/core/Core.java, as shown below.

  - **File name, line numbers as in Github**
    - /META-INF/context.xml
      - line(s): 6-15, 18-27

  - **Snapshots showing use in your code**

/META-INF/context.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <Context>
4
5      <!-- Defines a Data Source Connecting to localhost moviedb-->
6      <Resource name="jdbc/moviedb"
7                auth="Container"
8                driverClassName="com.mysql.cj.jdbc.Driver"
9                type="javax.sql.DataSource"
10               maxTotal="100"
11               maxIdle="30"
12               maxWaitMillis="10000"
13               username="mytestuser"
14               password="mypassword"
15               url="jdbc:mysql://localhost:3306/moviedb?autoReconnect=true&amp;useSSL=false&amp;cachePrepStmts=true"/>
16
17
18     <Resource name="jdbc/wmoviedb"
19               auth="Container"
20               driverClassName="com.mysql.cj.jdbc.Driver"
21               type="javax.sql.DataSource"
22               maxTotal="100"
23               maxIdle="30"
24               maxWaitMillis="10000"
25               username="mytestuser"
26               password="mypassword"
27               url="jdbc:mysql://172.31.13.116:3306/moviedb?autoReconnect=true&amp;useSSL=false"/>
28 </Context>
```

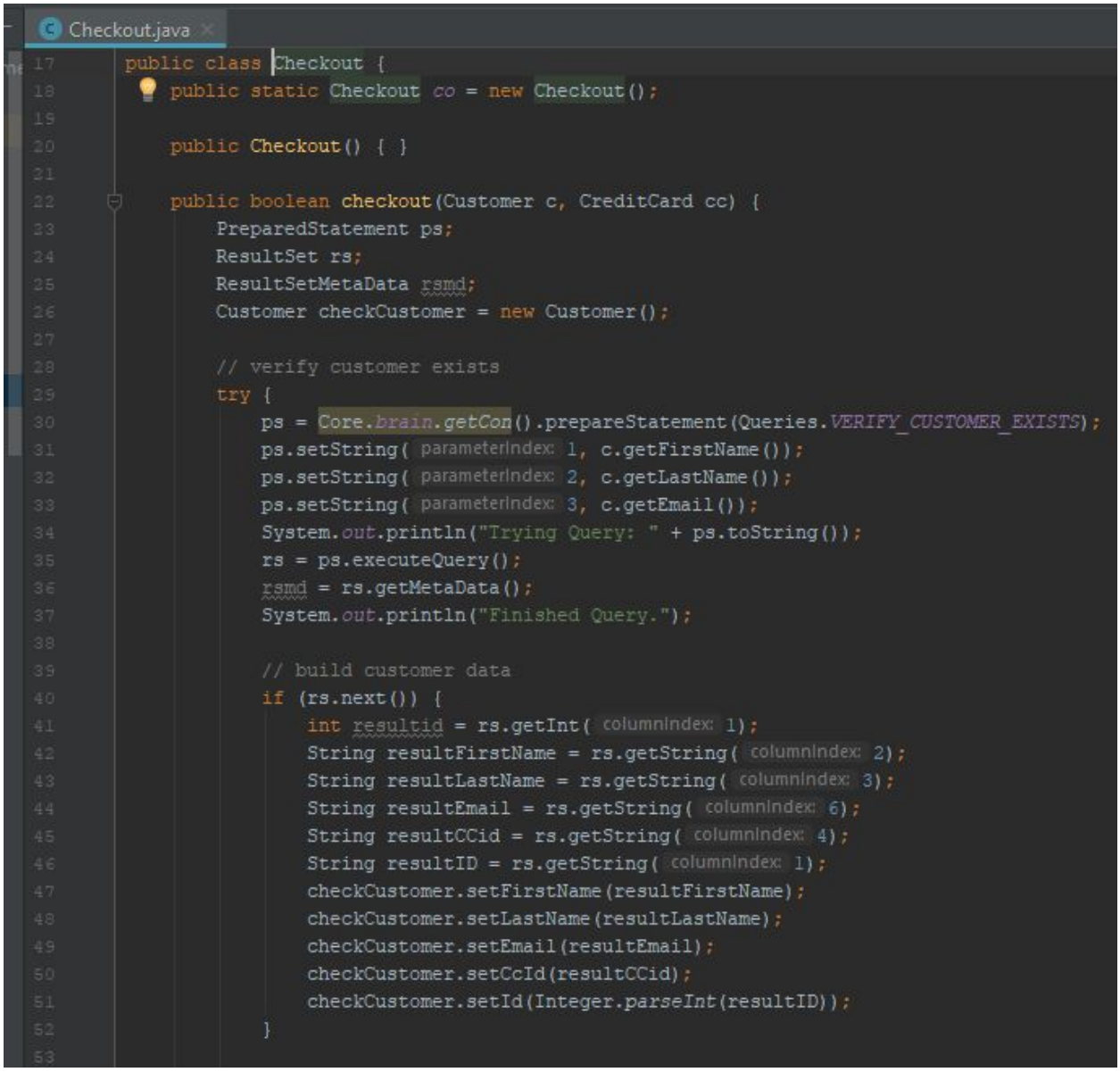
- **How did you use Prepared Statements?**

  In our backend implementation, PreparedStatements are used in every instance where the database must be queried for data. We have java classes listening on specific paths

for incoming JSON requests. When such a request is received, these classes will then call upon other classes which contain PreparedStatements, and will prepare the queries depending on any parameters provided in the request.

- **File name, line numbers as in Github**
  - /src/fabflix/core/Checkout.java
    - line(s): 30
  - /src/fabflix/core/Core.java
    - line(s): 164, 182, 225, 268, 295, 320, 351, 380
  - /src/fabflix/core/LoginVerifyUtils.java
    - line(s): 48, 62
  - /src/fabflix/core/MovieList.java
    - line(s): 44
  - /src/fabflix/core/AndroidFulltextSearch.java
    - line(s): 30
  - /src/fabflix/core/UpdateDB.java
    - line(s): 139
  - /src/fabflix/core/Autocomplete.java
    - line(s): 31
  - /src/webpages/DashboardPage.java
    - line(s): 111
  - /src/webpages/InsertStarDashboard.java
    - line(s): 103
  - /src/webpages/RetrieveMetadataDashboard.java
    - line(s): 85

- **Snapshots showing use in your code**


Checkout.java
public class Checkout {
public static Checkout co = new Checkout();
public Checkout() { }
public boolean checkout(Customer c, CreditCard cc) {
PreparedStatement ps;
ResultSet rs;
ResultSetMetaData rsmd;
Customer checkCustomer = new Customer();
// verify customer exists
try {
ps = Core.brain.getCon().prepareStatement(Queries.VERIFY_CUSTOMER_EXISTS);
ps.setString(parameterIndex: 1, c.getFirstName());
ps.setString(parameterIndex: 2, c.getLastName());
ps.setString(parameterIndex: 3, c.getEmail());
System.out.println("Trying Query: " + ps.toString());
rs = ps.executeQuery();
rsmd = rs.getMetaData();
System.out.println("Finished Query.");
// build customer data
if (rs.next()) {
int resultid = rs.getInt(columnIndex: 1);
String resultFirstName = rs.getString(columnIndex: 2);
String resultLastName = rs.getString(columnIndex: 3);
String resultEmail = rs.getString(columnIndex: 6);
String resultCCid = rs.getString(columnIndex: 4);
String resultID = rs.getString(columnIndex: 1);
checkCustomer.setFirstName(resultFirstName);
checkCustomer.setLastName(resultLastName);
checkCustomer.setEmail(resultEmail);
checkCustomer.setCcId(resultCCid);
checkCustomer.setId(Integer.parseInt(resultID));
}

Core.java

```
158
159 @    private String query_for_password(String username) {
160          System.out.println("query_for_password(" + username + ")");
161          ResultSetMetaData rsmd;
162          try {
163              PreparedStatement ps = con.prepareStatement(Queries.LOGIN_QUERY);
164              ps.setString( parameterIndex: 1, username);
165              System.out.println("TRYING QUERY: " + ps.toString());
166              ResultSet rs = ps.executeQuery();
167              System.out.println("FINISHED QUERY!");
168              rsmd = rs.getMetaData();
169
170              if (rs.next()) {
171                  return rs.getString( columnIndex: 1);
172              }
173          } catch (SQLException e) {
174              e.printStackTrace();
175          }
176          return null;
177      }
178
179 @    private String query_for_employeePassword(String username) {
180          ResultSetMetaData rsmd;
181          try {
182              PreparedStatement ps = con.prepareStatement(Queries.EMPLOYEE_LOGIN_QUERY);
183              ps.setString( parameterIndex: 1, username);
184              ResultSet rs = ps.executeQuery();
185              rsmd = rs.getMetaData();
186
187              if (rs.next()) {
188                  return rs.getString( columnIndex: 1);
189              }
190          } catch (SQLException e) {
191              e.printStackTrace();
192          }
193          return null;
194      }
195
```

LoginVerifyUtils.java

```
        }
    }
    private static String query_for_password(String username) {
        System.out.println("query for password for username: "+username);
        ResultSetMetaData rsmd;
        try {
            PreparedStatement ps = Core.brain.getCon().prepareStatement(Queries.LOGIN_QUERY);
            ps.setString( parameterIndex: 1, username);
            ResultSet rs = ps.executeQuery();
            rsmd = rs.getMetaData();

            if (rs.next()) {
                return rs.getString( columnIndex: 1);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return null;
    }
    public static  boolean customerEmailExists(String email){
        ResultSetMetaData rsmd;
        try {
            PreparedStatement ps = Core.brain.getCon().prepareStatement(Queries.CUSTOMER_EXISTS_QUERY);
            ps.setString( parameterIndex: 1, email);
            ResultSet rs = ps.executeQuery();
            rsmd = rs.getMetaData();

            if (rs.next()) {
                // return rs.getBoolean("email");
                //    System.out.println(rs.getString("email"));
                //    if(!rs.getString(email).equals("null")){
                return true;
                //  }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
```

```
    public boolean employeeEmailExists(String email) {
        ResultSetMetaData rsmd;
        try {
            PreparedStatement ps = con.prepareStatement(Queries.EMPLOYEE_EXISTS_QUERY);
            ps.setString( parameterIndex: 1, email);
            ResultSet rs = ps.executeQuery();
            rsmd = rs.getMetaData();

            if (rs.next()) {
                // return rs.getBoolean("email");
                //    System.out.println(rs.getString("email"));
                //    if(!rs.getString(email).equals("null")){
                return true;
                //  }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return false;
    }
```

Core.java

```
286         /* -------------------- QUERY FUNCTIONS -------------------- */
287         public MovieModel getSingleMovieData(String movieID, MovieInfoOptions options) {
288             System.out.println("getSingleMovieData(" + movieID + ")");
289             PreparedStatement ps;
290             ResultSet rs;
291             ResultSetMetaData rsmd;
292             Movie m;
293
294             try {
295                 ps = Core.getCon().prepareStatement(Queries.GET_SINGLE_MOVIE_DATA_ALL);
296                 ps.setString( parameterIndex: 1, movieID);
297                 System.out.println("  Trying Query: " + ps.toString());
298                 rs = ps.executeQuery();
299                 rsmd = rs.getMetaData();
300                 System.out.println("  Finished Query.");
301                 if (rs.next()) {
302                     m = getMovieFromResultSet(rs, options);
303                     return new MovieModel(m);
304                 }
305             } catch (SQLException e) {
306                 e.printStackTrace();
307             }
308             return null;
309         }
310
```

Core.java

```
343         public Movie[] getAllMoviesWithStar(String starID) {
344             System.out.println("getAllMoviesWithStar(" + starID + ")");
345             PreparedStatement ps;
346             ResultSet rs;
347             ResultSetMetaData rsmd;
348             ArrayList<Movie> movieList;
349
350             try {
351                 ps = Core.getCon().prepareStatement(Queries.GET_ALL_MOVIES_WITH_STAR);
352                 ps.setString( parameterIndex: 1, starID);
353                 System.out.println("Trying Query: " + ps.toString());
354                 rs = ps.executeQuery();
355                 rsmd = rs.getMetaData();
356                 System.out.println("Query Finished!");
357
358                 if (rs.next()) {
359                     System.out.println("Building list from RS");
360                     movieList = buildListFromResultSet(rs, new MovieInfoOptions());
361                     System.out.println("Returning Movie[] of all movies with star");
362                     return buildMovieArrayFromList(movieList);
363                 }
364             } catch (SQLException e) {
365                 e.printStackTrace();
366             }
367             System.out.println("RETURNING NULL!");
368             return null;
369         }
370
```

Core.java

```
373     public ShoppingCart addMovieToShoppingCart(String username, String movieID, int Qty) {
374         PreparedStatement ps;
375         ResultSet rs;
376         ResultSetMetaData rsmd;
377         ShoppingCart sc = getShoppingCart(username);
378
379         try {
380             ps = getCon().prepareStatement(Queries.GET_SINGLE_MOVIE_DATA_ALL);
381             ps.setString( parameterIndex: 1, movieID);
382             rs = ps.executeQuery();
383             rsmd = rs.getMetaData();
384
385             if (rs.next()) {
386                 Movie m = getMovieFromResultSet(rs, new MovieInfoOptions());
387                 for (int i = 0; i < Qty; i++)
388                     sc.addMovie(m);
389             }
390             return sc;
391         } catch (SQLException e) {
392             e.printStackTrace();
393         }
394         return sc;
395     }
396
```

MovieList.java

```
25  public final class MovieList {
26      public static MovieList ml = new MovieList();
27
28      private MovieList() { }
29
30      public MovieModel[] buildMovieList(SearchParameters sp) {
31          System.out.println("=============================================================================================================");
32          System.out.println("buildMovieList()");
33          System.out.println("-------------------------------------------------------------------------------------------------------------");
34          ArrayList<Movie> movies;
35          MovieModel[] movieArray;
36          String query = QueryBuilder.qb.buildQuery(sp);
37          PreparedStatement ps;
38          ResultSet rs;
39          ResultSetMetaData rsmd;
40          StatTimerLogger stl = new StatTimerLogger( fileName: "TJ_timings.txt");
41
42          try {
43              System.out.println("  QUERY SHELL IS: " + query);
44              ps = Core.getCon().prepareStatement(query);
45              // set parameters for ps
46              System.out.println("  Setting parameters for query...");
47              for (int indexOfParam = 0, indexOfQuery = 1; indexOfParam < sp.getParams().length - 2; ++indexOfParam, ++indexOfQuery) {
48  //                System.out.println("    Setting parameter: " + sp.getParams()[indexOfParam] + " to index " + indexOfQuery);
49                  ps.setString(indexOfQuery, sp.getParams()[indexOfParam]);
50              }
51              ps.setInt( parameterIndex: sp.getParams().length - 1, sp.getLimitInt());
52              ps.setInt(sp.getParams().length, sp.getOffsetInt());
53  //            System.out.println("  DONE!");
54  //            sleep(3000);
55              System.out.println("  Trying Query: " + ps.toString());
56              stl.setQuery(ps.toString());
57              stl.startTiming();
58              rs = ps.executeQuery();
59              stl.stopTiming();
60              rsmd = rs.getMetaData();
61              System.out.println("  Finished Query!");
62              movies = Core.brain.buildListFromResultSet(rs, new MovieInfoOptions());
63              movieArray = Core.brain.buildMovieModelArrayFromList(movies);
64              return movieArray;
65          } catch (SQLException e) {
66              e.printStackTrace();
67  //        } catch (InterruptedException e) {
68  //            e.printStackTrace();
69          }
70          return null;
71      }
72  }
```

MovieList › buildMovieList()

AndroidFulltextSearch.java

```java
package fabflix.core;

import ...

public class AndroidFulltextSearch {
    public static AndroidFulltextSearch  ml = new AndroidFulltextSearch ();

   public AndroidFulltextSearch() { }

    public MovieModel[] buildMovieList(String title, int offset) {
        System.out.println("================================================================================================");
        System.out.println("AndroidFulltextSearch()");
        System.out.println("------------------------------------------------------------------------------------------------");
        ArrayList<Movie> movies;
        MovieModel[] movieArray;
        String query =  "SELECT m.id, m.title, m.year, m.director," +
                " m.backdrop_path, m.budget, m.overview, m.poster_path, m.revenue, r.rating, " +
                "r.numVotes, GROUP_CONCAT(DISTINCT g.id SEPARATOR ';') AS genreIDs, GROUP_CONCAT(DISTINCT g.name SEPARATOR ';') " +
                "AS genres, GROUP_CONCAT(DISTINCT CONCAT(s.id,'=', s.name) SEPARATOR ';') " +
                "AS stars FROM movies m, ratings r, genres_in_movies gm, genres g, stars s, stars_in_movies sm" +
                " WHERE MATCH (m.title) AGAINST( ? IN BOOLEAN MODE) AND  g.id = gm.genreId AND gm.movieId = m.id AND r.movieId= m.id AND" +
                " s.id = sm.starId AND sm.movieId = m.id    GROUP BY m.id  ORDER BY m.title  ASC  LIMIT 10 OFFSET ?;";

        PreparedStatement ps;
        ResultSet rs;
        ResultSetMetaData rsmd;
        String tokenList= buildMatchingString(title);


        try {
            System.out.println("  QUERY SHELL IS: " + query);
            ps = Core.getCon().prepareStatement(query);
            // set parameters for ps
            System.out.println("  Setting parameters for query...");

//              System.out.println("    Setting parameter: " + sp.getParams()[indexOfParam] + " to index " + indexOfQuery);
            ps.setString( parameterIndex: 1,tokenList);
            ps.setInt( parameterIndex: 2,offset);
```

AndroidFulltextSearch › buildMovieList()

Autocomplete.java

```java
public final class Autocomplete {
    public static Autocomplete ac = new Autocomplete();

    private Autocomplete() {

    }

    public MovieModel[] buildSearchResults(String searchString) {
        System.out.println("buildSearchResults(" + searchString + ")");
        String ss = buildMatchString(searchString);
        String query = "SELECT id, title FROM movies WHERE MATCH (title) AGAINST (\'" + ss + "\' IN BOOLEAN MODE) LIMIT 10;";
        System.out.println("QUERY = " + query);

        ArrayList<Movie> movies;
        MovieModel[] movieArray;
        PreparedStatement ps;
        ResultSet rs;
        ResultSetMetaData rsmd;

         try {
            ps = Core.getCon().prepareStatement(query);
            System.out.println("  Trying Query: " + ps.toString());
            rs = ps.executeQuery();
            rsmd = rs.getMetaData();
            System.out.println("  Finished Query!");
            MovieInfoOptions mo = new MovieInfoOptions(
                    getID: true,  getTitle: true,  getYear: false,
                    getDirector: false,  getBackdropPath: false,  getBudget: false,
                    getOverview: false,  getPosterPath: false,  getRevenue: false,
                    getRating: false,  getNumVotes: false,  getGenres: false,
                    getStars: false);
            movies = Core.brain.buildListFromResultSet(rs, mo);
            movieArray = Core.brain.buildMovieModelArrayFromList(movies);
            return movieArray;
        } catch (SQLException e) {
            e.printStackTrace();
            return null;
        }
```

Autocomplete › buildSearchResults()

DashboardPage.java

```
                try {

                    query = Queries.ADD_MOVIE_QUERY;
                    System.out.println("  QUERY SHELL IS: " + query);
                    ps = Core.getCon().prepareStatement(query);
                System.out.println("GOT PARAMETERS! Building SearchParameters object...");
                    ps.setString( parameterIndex: 1,title);
                System.out.println("GOT PARAMETERS! Building SearchParameters object...");
                    ps.setInt( parameterIndex: 2,year);
                System.out.println("GOT PARAMETERS! Building SearchParameters object...");
                    ps.setString( parameterIndex: 3,director);
                System.out.println("GOT PARAMETERS! Building SearchParameters object...");
                    ps.setString( parameterIndex: 4,genre);
                System.out.println("GOT PARAMETERS! Building SearchParameters object...");
                    ps.setString( parameterIndex: 5,mStar);

                    System.out.println("  Trying Query: " + ps.toString());
                    rs = ps.executeQuery();
                    System.out.println(" getting string");
                if (rs.next()) {
                    success= rs.getBoolean( columnIndex: 1);
                }


                    if(success){
                        response="{\"success\":\"1\"}";
                    }else{
                       response= "{\"success\":\"0\"}";
                    }
                    rsmd = rs.getMetaData();
                    System.out.println("  Finished Query!");
              //  }
```

RetrieveMetadataDashboard.java

```
            PreparedStatement ps;
            ResultSet rs;
            ResultSetMetaData rsmd;
            String query =Queries.METADATA_QUERY;
            ps = Core.getCon().prepareStatement(query);


                System.out.println("  Trying Query: " + ps.toString());
                rs = ps.executeQuery();


                rsmd = rs.getMetaData();
                System.out.println("  Finished Query!");
                System.out.println("  Retrieving Data from Result Set!");
                boolean hasMore=rs.next();
                System.out.println("hasMore:"+hasMore);

                    if (hasMore) {
                        do {
//                          metadataJsonObject+=
//                                  "{\"table\":"+rs.getString("TABLE_NAME")
//                                   +",\"column\":\""+rs.getString("COLUMN_NAME")
//                                   +"\", \"data_type\":\""+rs.getString("DATA_TYPE")+"\"}";

                            md.add(new MetadataModel(rs.getString( columnLabel: "TABLE_NAME"),rs.getString( columnLabel: "COLUMN_NAME"),rs.getString( columnLabel: "DATA_TYPE")));

                            hasMore=rs.next();
                        //    System.out.println(metadataJsonObject);
                        //    System.out.println("hasMore:"+hasMore);
                            if(hasMore){
                        //        metadataJsonObject+=",";
                            }
```

RetrieveMetadataDashboard › getDashBoard()

Task 2

- **Address of AWS and Google instances**
  - Google: http://35.227.84.246/project5/login.html
  - AWS Instance 1: http://18.220.196.86/project5/login.html
  - AWS Instance 2(master): http://18.220.217.218:8080/project5/login.html
  - AWS Instance 3(slave): http://18.217.157.152:8080/project5/login.html

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

  Verified that both are accessible. When accessed, they both redirect to the appropriate 8080 instance.

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

  /META-INF/context.xml defines two database resources--one for read requests, and one for write requests. The resource for read requests are directed to the localhost MySQL instance, so that in the case of the either master/slave the read can be processed locally. However, the other resource explicitly defines the IP of the master MySQL instance, which will handle the read requests. On the backend, two connections are created to reflect these data sources.

  - File name, line numbers as in Github
    - /META-INF/context.xml
      - line(s): 6-15, 18-27

  - Snapshots

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <Context>
4
5      <!-- Defines a Data Source Connecting to localhost moviedb-->
6      <Resource name="jdbc/moviedb"
7                auth="Container"
8                driverClassName="com.mysql.cj.jdbc.Driver"
9                type="javax.sql.DataSource"
10               maxTotal="100"
11               maxIdle="30"
12               maxWaitMillis="10000"
13               username="mytestuser"
14               password="mypassword"
15               url="jdbc:mysql://localhost:3306/moviedb?autoReconnect=true&amp;useSSL=false&amp;cachePrepStmts=true"/>
16
17
18     <Resource name="jdbc/wmoviedb"
19               auth="Container"
20               driverClassName="com.mysql.cj.jdbc.Driver"
21               type="javax.sql.DataSource"
22               maxTotal="100"
23               maxIdle="30"
24               maxWaitMillis="10000"
25               username="mytestuser"
26               password="mypassword"
27               url="jdbc:mysql://172.31.13.116:3306/moviedb?autoReconnect=true&amp;useSSL=false"/>
28 </Context>
```

- **How read/write requests were routed?**

  When a PreparedStatement is to be executed, an appropriate connection that has already been defined in context.xml and created earlier in the application is selected (either the write or read). We use the two functions getCon() and getWriteCon() defined in /src/fabflix/core/Core.java to return these connections to other java classes needing to execute a PreparedStatement. The connections themselves are defined in /src/fabflix/core/Core.java.init();

  - **File name, line numbers as in Github**
    - **FOR WRITE CONNECTIONS ONLY! READ CONNECTION EXAMPLES ALREADY DEFINED EARLIER IN REPORT**
    - /src/fabflix/webpages/DashboardPage.java
      - line(s): 111
    - /src/fabflix/webpages/InsertStarDashboard.java
      - line(s): 102

  - Snapshots
  - /src/fabflix/webpages/InsertStarDashboard.java @line:104

```
98              try {
99
100                     query = Queries.INSERT_STAR_QUERY;
101                     System.out.println("  QUERY SHELL IS: " + query);
102                     ps = Core.getWriteCon().prepareStatement(query);
103                     ps.setString(1,starname);
```

  /src/fabflix/webpages/DashboardPage.java @line:111

```
108
109                     query = Queries.ADD_MOVIE_QUERY;
110                     System.out.println("  QUERY SHELL IS: " + query);
111                     ps = Core.getWriteCon().prepareStatement(query);
112                 System.out.println("GOT PARAMETERS! Building SearchParameters object...");
113                     ps.setString(1,title);
114                 System.out.println("GOT PARAMETERS! Building SearchParameters object...");
115                     ps.setInt(2,year);
```

  -

  - **Master**
  - Below the master bind address is opened to 0.0.0.0 and its server_id=1 and its log bin is uncommented.

ubuntu@ip-172-31-23-101: /etc/mysql/mysql.conf.d

```
basedir         = /usr
datadir         = /var/lib/mysql
tmpdir          = /tmp
lc-messages-dir = /usr/share/mysql
skip-external-locking
#
# Instead of skip-networking the default is now to listen only on
# localhost which is more compatible and is not less secure.
bind-address            = 0.0.0.0
#
# * Fine Tuning
#
```

-

```
#log-queries-not-using-indexes
#
# The following can be used as easy to replay backup logs or f
# note: if you are setting up a replication slave, see README.
#       other settings you may need to change.
server-id               = 1
log_bin                 = /var/log/mysql/mysql-bin.log
expire_logs_days        = 10
max_binlog_size   = 100M
#binlog_do_db           = include_database_name
```

- Both the Here the master status is shown. mysql-bin.000007 and position are both listened to by the slave

```
mysql> show master status;
+------------------+----------+--------------+------------------+-------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set |
+------------------+----------+--------------+------------------+-------------------+
| mysql-bin.000007 |      154 |              |                  |                   |
+------------------+----------+--------------+------------------+-------------------+
1 row in set (0.00 sec)

mysql>
```

-
- **Slave**
- The slaves bind-address it opened to 0.0.0.0 as well.

```
lc-messages-dir = /usr/share/mysql
skip-external-locking
#
# Instead of skip-networking the default is now to listen only on
# localhost which is more compatible and is not less secure.
bind-address            = 0.0.0.0
#
# * Fine Tuning
```

-
- Here the server-id=2 for the slave.

```
# The following can be used as easy to replay backup logs or for replic
# note: if you are setting up a replication slave, see README.Debian ab
#       other settings you may need to change.
server-id               = 2
#log_bin                        = /var/log/mysql/mysql-bin.log
expire_logs_days        = 10
max_binlog_size   = 100M
#binlog_do_db           = include_database_name
```

- Here it shows the slave status. It is currently listening to the master for any changes. This is a one way relationship. Changes made to the master will propagate to the slave, but changes made to the slave will not affect the

master. It can be seen that the bin file and position match the masters.

Task 3

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

  /project5/test_logs/

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

  /project5/test_logs/

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

  /project5/count_times.java

  Please note: the script was run once for each test case. It opens the files TJ_times.txt and TS_times.txt, the paths for which were different than they are now at the time of execution. If you run this script as is right now, it will not work because the paths are wrong. The TS and TJ times files for each test case are located in their corresponding folders in /project5/test_logs/.

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

  /project5/target/project5.war