Group 90
Name: Zuomin Ren, 95288742
Name: Rui Cao, 13482991

Task 1

a.How did you use connection pooling?

We use connection pooling through three separated steps.

1. Add the "context.xml" in path: project5\WebContent\META-INF\context.xml. In this file, we define the JDNI/JDBC resource with the proper connection pooling attributes.

2. Modify the "web.xml" in path: project5\WebContent\WEB-INF\web.xml so that the application can use the resource. For detail, we added the resource ref tag to describe the resource that we want to use.

3. Add a new connection function called "make_connection" in path: project5\src\Eofilm\dbFunctions.java, which will enable connection pooling.

b.File name, line numbers as in Github

Corresponding to these three steps mentioned above, the file name and line numbers are as follow:

1. **File name:** context.xml
   Path: project5\WebContent\META-INF\context.xml
   Line numbers: All lines in context.xml
2. **File name:** web.xml
   Path: project5\WebContent\WEB-INF\web.xml
   Line numbers: line 13 to 18
3. **File name:** dbFunctions.java
   Path: project5\src\Eofilm\dbFunctions.java
   Line numbers: line 36 to 42

c.Snapshots

According to these three modifications, the corresponding snapshots are as follow:

1. context.xml: all lines

```
<Context docBase="/Users/zuomin/mycs122b-projects/cs122b-winter18-team-90/project5" path="/project5"
reloadable="true" source="org.eclipse.jst.jee.server:project5">

        <Resource name="jdbc/MovieDB"
        auth="Container"
        type="javax.sql.DataSource"
        username="root"
        password="123456"
        driverClassName="com.mysql.jdbc.Driver"
         maxActive="15"
         maxTotal="100" maxIdle="30" maxWaitMillis="10000"
         url="jdbc:mysql://localhost:3306/moviedb?autoReconnect=true&amp;useSSL=false&amp;cachePrepStmts=true" />
        />

</Context>
```

2. web.xml: line 13 to 18

```
13      <resource-ref>
14         <description>Pooling</description>
15         <res-ref-name>jdbc/MovieDB</res-ref-name>
16         <res-type>javax.sql.DataSource</res-type>
17         <res-auth>Container</res-auth>
18     </resource-ref>
```

3. dbFounctions.java: line 36 to 42

```
36      public void make_connection(String path, String user_name, String pass) throws Exception
37      {
38          Context initContext = new InitialContext();
39          Context envContext = (Context)initContext.lookup("java:comp/env");
40          DataSource ds = (DataSource)envContext.lookup("jdbc/MovieDB");
41          connection = ds.getConnection();
42      }
```

d.How did you use Prepared Statements?

There is a function called “search_movies” which is used to search movies in the database in dbfunction.java. Inside this function, we change “statement” method to “preparestatement” method by using the “preparestatement” function contained in java.sql library. We also use “preparestatement” method in other functions which need to connect to database and execute search operation.

e.File name, line numbers as in Github

File name: dbFunctions.java
Path: project5\src\Eofilm\dbFunctions.java
Line numbers: line 127

f.Snapshots

```
117     public LinkedHashMap<String, movie> search_movies(searchparameters curSearch) throws Exception {
118         StringBuilder query = new StringBuilder("SELECT DISTINCT movies.id,title,year,director "
119                 + "FROM stars "
120                 + "INNER JOIN stars_in_movies ON stars.id = stars_in_movies.starId "
121                 + "LEFT OUTER JOIN movies ON movies.id = stars_in_movies.movieId "
122                 + "LEFT OUTER JOIN genres_in_movies ON genres_in_movies.movieId = movies.id WHERE ");
123
124         build_query(query, curSearch);
125         LinkedHashMap<String, movie> movie_list = new LinkedHashMap<String, movie>();
126
127         PreparedStatement ps = connection.prepareStatement(query.toString());
128         add_ps_parameters(ps, curSearch);
129         ResultSet rs = ps.executeQuery();
130         populate_list(movie_list, rs);
131         rs.close();
132         ps.close();
133         return movie_list;
134     }
```

Task 2

a.Address of AWS and Google instances

Aws

Instance1: 54.193.89.217(public IP)

Instance2(master): 54.153.38.147(public IP)

Instance3(slave): 54.215.191.25(public IP)

Google

Instance: 35.231.27.40:80/project5/login (you have to go to login page at first)

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

Yes, the website get opened both on Google's 80 port and AWS' 8080 port.

c.How connection pooling works with two backend SQL?

For connection pooling to work with load balancing, each of the instances need to have their own set of connection pools for operation. The code and file name likes what mentioned in task1.

Task 3

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

~/cs122b-winter18-team-90/project5/jmeter/ localhost_access_log.2018-03-16

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

~/cs122b-winter18-team-90/project5/jmeter/jmeter_report

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

~/cs122b-winter18-team-90/project5/jmeter/ScripttoParseLogs

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

~/cs122b-winter18-team-90/project5/project5.war
~/cs122b-winter18-team-90/project5/README