Team 71:
Members:

Jason Bugallon, jbugallo, 85806059
Kulraj Dhaliwal, kulrajd, 18833491

Project 5 - Report

Task 1

- How did you use connection pooling?

Connection pooling was enabled by changing the datasource that was used in the code to connect to the mySQL database from moviedb (no connection pooling) to localdb (connection pooling). The connection pooling datasource (localdb) was defined in the project's context.xml file found in the META-INF folder and was also referenced into the web.xml file in the WEB-INF folder. Snapshots of the datasource defined inside the context.xml and the reference to it defined inside the web.xml can be viewed below.

- File name, line numbers as in Github
  - **Reading (found in project1/src directory)**
    - **MovieListServlet, lines 78-93**
    - **BrowseServlet, lines 41-56**
    - **SingleMovieServlet, lines 39-54**
    - **SingleStarServlet, lines 39-54**
    - **CartServlet, lines 53-68**
  - **Meta-Data (found in project1/WebContent/**
    - **META-INF/context.xml, line 15-19**
    - **WEB-INF/web.xml, lines 29-39**

- Snapshots showing use in your code **(This is only showing a few out of many usages of connection pooling in the codebase)**
- **All code in these snapshots are found in this directory: project1/src**
- **All snapshots can be found in an organized manner inside the snapshots/connection-pooling-usage/ directory**

MovieListServlet.java

```
78                  // Start connection pooling
79              Context initCtx = new InitialContext();
80
81              Context envCtx = (Context) initCtx.lookup("java:comp/env");
82              if (envCtx == null)
83                  out.println("envCtx is NULL");
84
85              // Look up our data source
86              DataSource ds = (DataSource) envCtx.lookup("jdbc/localdb");
87
88              if (ds == null)
89                  out.println("ds is null.");
90
91              Connection dbc = ds.getConnection();
92              if (dbc == null)
93                  out.println("dbcon is null.");
```

-

BrowseServlet.java

```
            // Start connection pooling
Context initCtx = new InitialContext();

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

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

if (ds == null)
    out.println("ds is null.");

Connection dbcon = ds.getConnection();
if (dbcon == null)
    out.println("dbcon is null.");
```

SingleMovieServlet.java

```
                // Start connection pooling
  Context initCtx = new InitialContext();

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

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

  if (ds == null)
      out.println("ds is null.");

  Connection dbcon = ds.getConnection();
  if (dbcon == null)
      out.println("dbcon is null.");
```

SingleStarServlet.java

```
39                      // Start connection pooling
40              Context initCtx = new InitialContext();
41
42              Context envCtx = (Context) initCtx.lookup("java:comp/env");
43              if (envCtx == null)
44                  out.println("envCtx is NULL");
45
46              // Look up our data source
47              DataSource ds = (DataSource) envCtx.lookup("jdbc/localdb");
48
49              if (ds == null)
50                  out.println("ds is null.");
51
52              Connection dbcon = ds.getConnection();
53              if (dbcon == null)
54                  out.println("dbcon is null.");
```

CartServlet.java

```
53                      // Start connection pooling
54              Context initCtx = new InitialContext();
55
56              Context envCtx = (Context) initCtx.lookup("java:comp/env");
57              if (envCtx == null)
58                  out.println("envCtx is NULL");
59
60              // Look up our data source
61              DataSource ds = (DataSource) envCtx.lookup("jdbc/localdb");
62
63              if (ds == null)
64                  out.println("ds is null.");
65
66              Connection dbc = ds.getConnection();
67              if (dbc == null)
68                  out.println("dbcon is null.");
```

Set-up / Meta-data

context.xml

```
15      <Resource name="jdbc/localdb" auth="Container" type="javax.sql.DataSource"
16                maxTotal="100" maxIdle="30" maxWaitMillis="10000" username="Jason"
17                password="Jimmy123$" driverClassName="com.mysql.jdbc.Driver"
18                url="jdbc:mysql://localhost:3306/moviedb?autoReconnect=true&amp;useSSL=false&amp;cachePrepStmts=true"/>
19
```

web.xml

```
29    <resource-ref>
30        <description>
31            Resource reference to a factory for java.sql.Connection
32            instances that may be used for talking to a particular
33            database that
34            is configured in the server.xml file.
35        </description>
36        <res-ref-name>jdbc/localdb</res-ref-name>
37        <res-type>javax.sql.DataSource</res-type>
38        <res-auth>Container</res-auth>
39    </resource-ref>
```

- How did you use Prepared Statements?

  We used Prepared Statements when we were querying for something the user requested. The request would be sent to the Servlet using GET and instead of making a query directly include the user input, we put a placeholder "?" where their input would go. We then made a prepared statement and set the placeholder "?" to the user input.

  - File name, line numbers as in Github
    - **MovieListServlet.java Lines:**
      - **197-222 (quick search (full text search against against movie title)), 97-106, 118-127, 289-300 (some advanced search prepared statements)**
    - **-**

  - Snapshots showing use in your code (All this code can be found in the project1/src directory)
  - **All snapshots can be found in an organized manner inside the snapshots/prepared-statements-usage/ directory**

    - **Quick search (full text search against movie title) prepared statement**

```
query = "SELECT m.id, title, year, director, rating\r\n" +
        "FROM movies m, ratings r\r\n" +
        "WHERE m.id = r.movieId AND MATCH(m.title) AGAINST(? in BOOLEAN MODE)\r\n" +
        "ORDER BY SORT1 SORT2\r\n" +
        "LIMIT LIMITP\r\n" +
        "OFFSET OFFSETP";
query = processQuery(query, params);
statement = dbc.prepareStatement(query);

String[] splited = params.get("title")[0].split("\\s+");
if (splited.length == 1)
    {
        if (splited[0].length() < 3) {statement.setString(1, splited[0] + "*");}
        else {
        statement.setString(1, params.get("title")[0]);
        }
    }
else
{
    String searchable = "+" + splited[0] + "*";
    for (int i = 1; i < splited.length; i++)
    {
        searchable += "+" + splited[i] + "*";
    }
    statement.setString(1, searchable);
}
```

Advanced search prepared statements:

```
// Genre List Query
String genresQuery =
        "SELECT name\r\n" +
        "FROM movies m, genres g, genres_in_movies gm\r\n" +
        "WHERE m.id = gm.movieId AND g.id = gm.genreId AND m.id = ?";

PreparedStatement genresStatement = dbc.prepareStatement(genresQuery);

genresStatement.setString(1, movie_id);
ResultSet genresResult = genresStatement.executeQuery();


// Stars List Query
String starsQuery =
        "SELECT s.id, s.name\r\n" +
        "FROM movies m, stars_in_movies sm, stars s \r\n" +
        "WHERE m.id = sm.movieId AND s.id = sm.starId AND m.id = ?";

PreparedStatement starsStatement = dbc.prepareStatement(starsQuery);

starsStatement.setString(1, movie_id);
ResultSet starsResult = starsStatement.executeQuery();

if(params.containsKey("title") && params.containsKey("year"))
{
    query = "SELECT m.id, title, year, director, rating\r\n" +
            "FROM movies m, ratings r\r\n" +
            "WHERE m.id = r.movieId AND m.title LIKE ? AND m.year = ?\r\n" +
            "ORDER BY SORT1 SORT2\r\n" +
            "LIMIT LIMITP\r\n" +
            "OFFSET OFFSETP";
    query = processQuery(query, params);
    statement = dbc.prepareStatement(query);
    statement.setString(1, "%"+params.get("title")[0]+"%");
    statement.setString(2, params.get("year")[0]);
}
```

Task 2

- Address of AWS and Google instances
  (NOTE**: some functionality are unavailable without login such as checkout and a few buttons breaking. The employee dashboard can't be accessed without logging in with an employee account)**

  Google instance (Load Balancer):
  35.236.17.45/project1/login.html (with login)
  35.236.17.45/project1/ (without login)
  Instance 1 (Load Balancer):
  fabflix.fun/project1/login.html or 13.52.129.176/project1/login.html (with login)
  fabflix.fun/project1/ or 13.52.129.176/project1/ (without login)
  Instance 1 (Stand-alone):
  fabflix.fun:8080/project1/login.html or 13.52.129.176:8080/project1/login.html (with login)
  fabflix.fun:8080/project1/ or 13.52.129.176:8080/project1/ (without login)
  Instance 2 (Master):

13.57.252.139:8080/project1/login.html (with login)
13.57.252.139:8080/project1/ (without login)

Instance 3 (Slave):

13.57.253.17:8080/project1/login.html (with login)
13.57.253.17:8080/project1/ (without login)

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

Yes, we have verified that they are accessible from Google's 80 port, all AWS instances' 8080 ports and the AWS instance 1's (load balancer) 80 port.

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

When working with two backend SQL servers, there will be two resources in my context.xml and these two are referenced in my web.xml. These two resources will be two sql connection pools, the localdb and the masterdb. The localdb is the local SQL database, so it can either be the slave DB or the master DB depending on what instance (instance 2 or instance 3) the load balancer sent the user to. The masterdb is the SQL database located in instance 3 (master SQL DB). The localdb will take care of servlets that only do read requests while the masterdb will take care of servlets that has anything to do with writing (insert, altering, etc.) to the SQL server, this means that servlets that does both reading and writing will be handled by the masterdb. The specific lines of the context.xml and web.xml can be found below as well as screenshots of them inside the codebase. For specific usage in the servlet (reading/writing routing), these are further explained in the next part where it specifically talks about their usage in the code and also snapshots of usage and locations in the codebase.

  - File name, line numbers as in Github
    - **Set up / Meta-data**
      - **project1/WebContent/META-INF/context.xml, lines 14-24**
      - **project1/WebContent/WEB-INF/web.xml, lines 18-39**

  - Snapshots **(Can also be found in the snapshots/connection-pooling-usage directory)**
    - **Set-up / meta-data**

context.xml

```
14        <!-- Defines a Data Source Connecting to localhost (can be either slave or master instance based on the local instance)
15    <Resource name="jdbc/localdb" auth="Container" type="javax.sql.DataSource"
16              maxTotal="100" maxIdle="30" maxWaitMillis="10000" username="Jason"
17              password="Jimmy123$" driverClassName="com.mysql.jdbc.Driver"
18              url="jdbc:mysql://localhost:3306/moviedb?autoReconnect=true&amp;useSSL=false&amp;cachePrepStmts=true"/>
19
20        <!-- Defines a Data Source Connecting to master instance (instance 2) moviedb (with connection pooling)-->
21    <Resource name="jdbc/masterdb" auth="Container" type="javax.sql.DataSource"
22              maxTotal="100" maxIdle="30" maxWaitMillis="10000" username="Jason"
23              password="Jimmy123$" driverClassName="com.mysql.jdbc.Driver"
24              url="jdbc:mysql://13.57.252.139:3306/moviedb?autoReconnect=true&amp;useSSL=false&amp;cachePrepStmts=true"/>
```

web.xml

```
<resource-ref>
    <description>
        Resource reference to a factory for java.sql.Connection
        instances that may be used for talking to a particular
        database that
        is configured in the server.xml file.
    </description>
    <res-ref-name>jdbc/masterdb</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
</resource-ref>
<resource-ref>
    <description>
        Resource reference to a factory for java.sql.Connection
        instances that may be used for talking to a particular
        database that
        is configured in the server.xml file.
    </description>
    <res-ref-name>jdbc/localdb</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
</resource-ref>
```

- How read/write requests were routed?

  The activities where we wrote to the SQL server was when we added to the sales table after a successful checkout from a user, when an employee added a new star, and when an employee added a new movie. So in total, there were three servlets that handled writing and explicitly accessed the masterdb. Every other servlet did read requests to the SQL server so they all used the localdb. The specific servlets with line numbers and snapshots of the code where the connection pooling is set up can be found below.

  - File name, line numbers as in Github
    - **Writing (found in project1/src/ directory):**
      - **ConfirmationServlet.java, lines 62-77**
      - **AddMovieServlet.java, lines 47-62**
      - **AddStarServlet.java, lines 44-59**
    - **Reading (found in project1/src/ directory) - not all only a couple as there are too many to show:**
      - **MovieListServlet, lines 78-93**
      - **BrowseServlet, lines 41-56**
      - **SingleMovieServlet, lines 39-54**
      - **SingleStarServlet, lines 39-54**
      - **CartServlet, lines 53-68**

  - Snapshots **(Just a couple not all as there are too many)**

    - **Writing**

ConfirmationServlet.java

```
62                  // Start connection pooling
63              Context initCtx = new InitialContext();
64
65              Context envCtx = (Context) initCtx.lookup("java:comp/env");
66              if (envCtx == null)
67                  out.println("envCtx is NULL");
68
69              // Look up our data source
70              DataSource ds = (DataSource) envCtx.lookup("jdbc/masterdb");
71
72              if (ds == null)
73                  out.println("ds is null.");
74
75              Connection dbc = ds.getConnection();
76              if (dbc == null)
77                  out.println("dbcon is null.");
```

AddMovieServlet.java

```
47                  // Start connection pooling
48              Context initCtx = new InitialContext();
49
50              Context envCtx = (Context) initCtx.lookup("java:comp/env");
51              if (envCtx == null)
52                  out.println("envCtx is NULL");
53
54              // Look up our data source
55              DataSource ds = (DataSource) envCtx.lookup("jdbc/masterdb");
56
57              if (ds == null)
58                  out.println("ds is null.");
59
60              Connection dbcon = ds.getConnection();
61              if (dbcon == null)
62                  out.println("dbcon is null.");
```

AddStarServlet.java

```
                    // Start connection pooling
            Context initCtx = new InitialContext();

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

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

            if (ds == null)
                out.println("ds is null.");

            Connection dbcon = ds.getConnection();
            if (dbcon == null)
                out.println("dbcon is null.");
```

- **Reading (Just a couple)**

MovieListServlet.java

```
            // Start connection pooling
        Context initCtx = new InitialContext();

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

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

        if (ds == null)
            out.println("ds is null.");

        Connection dbc = ds.getConnection();
        if (dbc == null)
            out.println("dbcon is null.");
```

-

BrowseServlet.java

```
            // Start connection pooling
        Context initCtx = new InitialContext();

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

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

        if (ds == null)
            out.println("ds is null.");

        Connection dbcon = ds.getConnection();
        if (dbcon == null)
            out.println("dbcon is null.");
```

SingleMovieServlet.java

```
39                  // Start connection pooling
40              Context initCtx = new InitialContext();
41
42              Context envCtx = (Context) initCtx.lookup("java:comp/env");
43              if (envCtx == null)
44                  out.println("envCtx is NULL");
45
46              // Look up our data source
47              DataSource ds = (DataSource) envCtx.lookup("jdbc/localdb");
48
49              if (ds == null)
50                  out.println("ds is null.");
51
52              Connection dbcon = ds.getConnection();
53              if (dbcon == null)
54                  out.println("dbcon is null.");
```

SingleStarServlet.java

```
39                  // Start connection pooling
40              Context initCtx = new InitialContext();
41
42              Context envCtx = (Context) initCtx.lookup("java:comp/env");
43              if (envCtx == null)
44                  out.println("envCtx is NULL");
45
46              // Look up our data source
47              DataSource ds = (DataSource) envCtx.lookup("jdbc/localdb");
48
49              if (ds == null)
50                  out.println("ds is null.");
51
52              Connection dbcon = ds.getConnection();
53              if (dbcon == null)
54                  out.println("dbcon is null.");
```

CartServlet.java

```
53                      // Start connection pooling
54              Context initCtx = new InitialContext();
55
56              Context envCtx = (Context) initCtx.lookup("java:comp/env");
57              if (envCtx == null)
58                  out.println("envCtx is NULL");
59
60              // Look up our data source
61              DataSource ds = (DataSource) envCtx.lookup("jdbc/localdb");
62
63              if (ds == null)
64                  out.println("ds is null.");
65
66              Connection dbc = ds.getConnection();
67              if (dbc == null)
68                  out.println("dbcon is null.");
```

Task 3

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

  Yes, they can be found in under the directory jmeter/logs. There are two folders found in this directory, scaled-version and single-instance. The log files for the scaled-version and single-instance can be found in each of these directories respectively. In each line of the log file, the first entry is an entry for the ts and the second entry is tj, they are separated by commas (ts,tj).

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

  Yes, it can be found in the jmeter/ directory, it is named jmeter_report.html.

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

  Yes, it can be found in the root of the repository, it is named getAveragesScript.py. This script was created and tested with Python 3.7.2. To use it you must call it in the command line and supply it a single log file.

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

  Yes, they are both located in the root of the github repository. The WAR file is named project1.war. If you are to test this, the war file has to remain as this name because some of our hyperlinks in our web app hard codes "project1" into its urls.