CS122B
Project5 Report

Team 106 Members:

Name: Chenghao Zhang
Student ID: 32026824
Name: Rohan Rajeev
Student ID: 38249388

Task 1

1.How did you use connection pooling?

A connection pool is a cache of database connections maintained so that the connections can be reused when future requests to the database are required. Creating a new connection every time on a request is a very expensive process and involves delay which becomes significant when the server has to handle lots of request. In connection pooling, after a connection is created, it is placed in the pool and it is used again so that a new connection does not have to be established. If all the connections are being used, a new connection is made and is added to the pool.

To setup the connection pooling, we edit the context.xml file as shown below. Resources to connect to the database are mentioned. For connection pooling, we account parameters like total connections, maximum idle connections, timeout etc. While defining the datasource, we set connection instances to TestDB and use this resource to enable connection pooling.

Context.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<Context>

    <!-- Defines a Data Source Connecting to localhost moviedb-->
    <Resource name="jdbc/moviedb"
              auth="Container"
              driverClassName="com.mysql.jdbc.Driver"
              type="javax.sql.DataSource"
              username="mytestuser"
              password="Zch700805!"
              url="jdbc:mysql://127.0.0.1:3306/moviedb"/>
'

    <Resource name="jdbc/TestDB" auth="Container" type="javax.sql.DataSource"
              maxTotal="100" maxIdle="30" maxWaitMillis="10000" username="mytestuser"
              password="Zch700805!" driverClassName="com.mysql.jdbc.Driver"
              url="jdbc:mysql://localhost:3306/moviedb?autoReconnect=true&amp;useSSL=false"/>

    <Resource name="jdbc/WriteDB" auth="Container" type="javax.sql.DataSource"
              maxTotal="100" maxIdle="30" maxWaitMillis="10000" username="mytestuser"
              password="Zch700805!" driverClassName="com.mysql.jdbc.Driver"
              url="jdbc:mysql://172.31.40.125:3306/moviedb?autoReconnect=true&amp;useSSL=false"/>
</Context>
```

StarsServlet.java

```
149             Context initCtx = new InitialContext();
150
151             Context envCtx = (Context) initCtx.lookup("java:comp/env");
152             DataSource ds = (DataSource) envCtx.lookup("jdbc/TestDB");
153             // Get a connection from dataSource
154             Connection dbcon = ds.getConnection();
```

LoginServlet.java

```
58              Context initCtx = new InitialContext();
59
60              Context envCtx = (Context) initCtx.lookup("java:comp/env");
61              DataSource ds = (DataSource) envCtx.lookup("jdbc/TestDB");
62
63              String email="";
64              // Get a connection from dataSource
65              Connection dbcon = ds.getConnection();
```

Singlemovie.java

```
44              Context initCtx = new InitialContext();
45
46              Context envCtx = (Context) initCtx.lookup("java:comp/env");
47              DataSource ds = (DataSource) envCtx.lookup("jdbc/TestDB");
48              // Get a connection from dataSource
49              Connection dbcon = ds.getConnection();
```

SingleStarServlet.java

```
44                  Context initCtx = new InitialContext();
45
46                  Context envCtx = (Context) initCtx.lookup("java:comp/env");
47                  DataSource ds = (DataSource) envCtx.lookup("jdbc/TestDB");
48                  // Get a connection from dataSource
49                  Connection dbcon = ds.getConnection();
50
```

Similarly, we have used connection pooling in all the servlets that establishes connection with the database. We have two connection pooling, one is connected to the localhost database, which is used for read request. And the other one is connected to master database, which is used for write request only(Task 2.4).

2.How did you use Prepared Statements?

- Our Web Application uses prepared statements in all servlets to query the database. To define a prepared statement we create a PreparedStatement object.
- This object can take parameters and supply it with different values each time we execute it. Finally, to execute a PreparedStatement object, we call an execute statement - executeQuery.

- Prepared statements are pre-compiled and work much much faster compared to normal statements. It helps in prevention of SQL injection attacks.
- Also, the prepare statement can be used in a loop, it is dynamic. In that case, it makes our code more efficient and faster.

File name, line numbers, Snapshots as in Github

LoginServlet.java

```
82              String query = "select password  " +
83                      "from customers " +
84                      "where email=?  " ;
85              PreparedStatement statement = dbcon.prepareStatement(query);
86              statement.setString(1, username);
87             // statement.setString(2, password);
```

Mainservlet.java

```
108             String query1="select * from movies where MATCH (title) AGAINST (? IN BOOLEAN MODE) " +
109                     "or ed(title,?)<=1 " +
110                     "limit 10";
111             PreparedStatement statement1 = dbcon.prepareStatement(query1);
112             statement1.setString(1, ll);
113             statement1.setString(2, movie_name);
```

Payservlet.java

```
60              String query1="select count(id) " +
61                      "from creditcards " +
62                      "where id=?  and firstName=? and lastName=? and expiration=? ";
63              PreparedStatement statement1 = dbcon.prepareStatement(query1);
64              statement1.setString(1, cardnumber);
65              statement1.setString(2, firstname);
66              statement1.setString(3, lastname);
67              statement1.setString(4, expiration);
```

```
96              String query2 ="select id " +
97                      "from customers " +
98                      "where email=? ";
99
100             PreparedStatement statement2 = dbcon.prepareStatement(query2);
101             statement2.setString(1, usern);
```

StarServlet.java

```
157             String query =
158                     "select distinct a.id, a.title, a.year, a.director, " +
159                             "GROUP_CONCAT(distinct a.genre_name) as genre_name, a.rating,  " +
160                             "GROUP_CONCAT(distinct s.name order by s.id) as star_name, " +
161                             " GROUP_CONCAT(distinct s.id) as star_id " +
162                             "from " +
163                             "(select distinct m.id, m.title, m.year, m.director, " +
164                             " GROUP_CONCAT(distinct g.name) as genre_name, r.rating " +
165                             "from movies as m, ratings as r, genres as g, genres_in_movies as y " +
166                             "where m.id=y.movieId and y.genreId=g.id and r.movieId=m.id " +
167                             "and (MATCH (m.title) AGAINST (? IN BOOLEAN MODE)"+x+" or ed(m.title,?)<=2 ) " +
168                             "and m.year like ? " +
169                             "and m.director like ? " +
170                             "and g.name like ? and m.title like ?  " +
171                             "group by m.id " +
172                             ") as a,  " +
173                             " " +
174                             "stars as s, stars_in_movies as x " +
175                             "where a.id=x.movieId and x.starId=s.id and s.name like ?  " +
176                             "group by a.id  " +
177                             "order by  " +
178                             sort  +
179                             " limit ?, ? " ;
180
```

```
203             PreparedStatement statement = dbcon.prepareStatement(query);
204             statement.setString(1, id_fix);
205             statement.setString(2,id);
206             statement.setString(3,year_fix);
207             statement.setString(4,director_fix);
208             statement.setString(5,genres_fix);
209             statement.setString(6,letters_fix);
210
211             statement.setString(7,star_fix);
212
213             statement.setInt(8,offset);
214             statement.setInt(9,numberfix);
215
216             ResultSet rs = statement.executeQuery();
```

SingleMovie.java

```
52              String query =
53                      "select distinct a.id, a.title, a.year, a.director, " +
54                              "GROUP_CONCAT(distinct a.genre_name) as genre_name, a.rating,  " +
55                              "GROUP_CONCAT(distinct s.name order by s.id) as star_name,  " +
56                              "GROUP_CONCAT(distinct s.id) as star_id " +
57                              "from " +
58                              "(select distinct m.id, m.title, m.year, m.director, " +
59                              " GROUP_CONCAT(distinct g.name) as genre_name, r.rating " +
60                              "from movies as m, ratings as r, genres as g, genres_in_movies as y " +
61                              "where m.id=y.movieId and y.genreId=g.id and r.movieId=m.id " +
62                              "group by m.id " +
63                              "order by r.rating desc " +
64                              ") as a,  " +
65                              " " +
66                              "stars as s, stars_in_movies as x " +
67                              "where a.id=x.movieId and x.starId=s.id and x.movieId=?  " +
68                              "group by a.id " +
69                              "order by a.rating desc ";
70
71
72              // Declare our statement
73              PreparedStatement statement = dbcon.prepareStatement(query);
74
75              // Set the parameter represented by "?" in the query to the id we get from url,
76              // num 1 indicates the first "?" in the query
77              statement.setString(1, id);
```

SingleStar.java

```
52                          String query = "SELECT  s.name, s.birthYear, starId, GROUP_CONCAT(distinct movieId) as movieId, " +
53                                          "GROUP_CONCAT(distinct title order by movieId) as title " +
54                                          "from stars as s, stars_in_movies as sim, movies as m  " +
55                                          "where sim.starId = s.id and m.id=sim.movieId and s.id=? " +
56                                          "group by s.id";
57
58                          // Declare our statement
59                          PreparedStatement statement = dbcon.prepareStatement(query);
60
61                          // Set the parameter represented by "?" in the query to the id we get from url,
62                          // num 1 indicates the first "?" in the query
63                          statement.setString(1, id);
```

DashBoard.java

```
66                  String query3 = "INSERT INTO stars VALUES(?,?,?);";
67                  PreparedStatement statement3 = dbcon.prepareStatement(query3);
68                  statement3.setString(1, starid);
69                  statement3.setString(2, starname);
```

Task 2

1.Address of AWS and Google instances

Public IP URL address:

AWS 1: http://18.222.122.188/project1-api-example/login.html
AWS 2: http://18.219.234.138:8080/project1-api-example/login.html
AWS 3: http://18.222.70.121:8080/project1-api-example/login.html
Google: http://34.73.179.138/project1-api-example/login.html

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

Yes, both got opened.

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

- So I create two connection poolings in the Context.xml file.
  One is connected to the localhost database(AWS local), called TestDB. This pooling connects to slaves database in slaves instance and connects to master in the master database. So this connection will be used for **read request.**
  The second connection connects to the **master's database.** It is called WriteDB. So this connection will be used as **write request only.**

```
14
15      <Resource name="jdbc/TestDB" auth="Container" type="javax.sql.DataSource"
16                maxTotal="100" maxIdle="30" maxWaitMillis="10000" username="mytestuser"
17                password="Zch700805!" driverClassName="com.mysql.jdbc.Driver"
18                url="jdbc:mysql://localhost:3306/moviedb?autoReconnect=true&amp;useSSL=false"/>
19
20      <Resource name="jdbc/WriteDB" auth="Container" type="javax.sql.DataSource"
21                maxTotal="100" maxIdle="30" maxWaitMillis="10000" username="mytestuser"
22                password="Zch700805!" driverClassName="com.mysql.jdbc.Driver"
23                url="jdbc:mysql://172.31.40.125:3306/moviedb?autoReconnect=true&amp;useSSL=false"/>
```

- There are only two write operations in this project. The first one is insert sales records into sales table in **Payservlet.java**, the second is that insert movies and stars to relevant tables in **DashBoard.java**. In these two request, I would use WriteDB connection in pooling connection, so whatever need to be written will go to master database because it is using master's private ip address. Since the slave database point to the master database log file, the slave database will also do some changes.

  Payservlet.java:

```
55          Context envCtx = (Context) initCtx.lookup( name: "java:comp/env");
56          DataSource ds = (DataSource) envCtx.lookup( name: "jdbc/WriteDB");
57          // Get a connection from dataSource
58          Connection dbcon = ds.getConnection();
```

DashBoard.java:

```
135         Context initCtx = new InitialContext();
136         Context envCtx = (Context) initCtx.lookup( name: "java:comp/env");
137         if (envCtx == null)
138             out.println("envCtx is NULL");
139         // Look up our data source
140         DataSource ds = (DataSource) envCtx.lookup( name: "jdbc/WriteDB");
141         Connection dbcon = ds.getConnection();
```

- All other request is read operation, I will just use TestDB in connection pooling. In this case, the read request could go to either master or slave database because it is using localhost.
  All other reading request sevelet:

```
88          Context initCtx = new InitialContext();
89
90          Context envCtx = (Context) initCtx.lookup( name: "java:comp/env");
91          DataSource ds = (DataSource) envCtx.lookup( name: "jdbc/TestDB");
92          // Get a connection from dataSource
93          Connection dbcon = ds.getConnection();
```

-

4.How read/write requests were routed?

- In the Instance 1, I create a apache2 balancer on instance 1 and google instance. I use a balancer like this to redirect all the request sent to instance1(using private ip aws address), making sure all requests sent to two instance equally. I also use sticky session to make sure the validation of session.

```
<Proxy "balancer://project1-api-example_balancer">
    BalancerMember "http://172.31.40.125:8080/project1-api-example" route=1
    BalancerMember "http://172.31.33.153:8080/project1-api-example" route=2
ProxySet stickysession=ROUTEID
</Proxy>
```

```
  ProxyPass /project1-api-example balancer://project1-api-example_balancer
  ProxyPassReverse /project1-api-example balancer://project1-api-example_balancer
```

Using public address for google instance balancer.

- So I create two connection poolings in the Context.xml file.
  One is connected to the localhost database, called TestDB. This pooling connects to slaves database in slaves instance and connects to master in the master database. So this connection will be used for **read request.**

The second connection connects to the **master's database.** It is called WriteDB. So this connection will be used as **write request only.**

Context.xml:

```
14
15      <Resource name="jdbc/TestDB" auth="Container" type="javax.sql.DataSource"
16                maxTotal="100" maxIdle="30" maxWaitMillis="10000" username="mytestuser"
17                password="Zch700805!" driverClassName="com.mysql.jdbc.Driver"
18                url="jdbc:mysql://localhost:3306/moviedb?autoReconnect=true&amp;useSSL=false"/>
19
20      <Resource name="jdbc/WriteDB" auth="Container" type="javax.sql.DataSource"
21                maxTotal="100" maxIdle="30" maxWaitMillis="10000" username="mytestuser"
22                password="Zch700805!" driverClassName="com.mysql.jdbc.Driver"
23                url="jdbc:mysql://172.31.40.125:3306/moviedb?autoReconnect=true&amp;useSSL=false"/>
```

- There are only two write operations in this project. The first one is insert sales records into sales table in Payservlet.java, the second is that insert movies and stars to relevant tables in DashBoard.java. In these two request, I would use WriteDB connection in pooling connection, so whatever need to be written will go to master database because it is using master's private ip address. Since the slave database point to the master database log file, the slave database will also do some changes.

Payservlet.java:

```
55          Context envCtx = (Context) initCtx.lookup( name: "java:comp/env");
56          DataSource ds = (DataSource) envCtx.lookup( name: "jdbc/WriteDB");
57          // Get a connection from dataSource
58          Connection dbcon = ds.getConnection();
```

DashBoard.java:

```
135         Context initCtx = new InitialContext();
136         Context envCtx = (Context) initCtx.lookup( name: "java:comp/env");
137         if (envCtx == null)
138             out.println("envCtx is NULL");
139         // Look up our data source
140         DataSource ds = (DataSource) envCtx.lookup( name: "jdbc/WriteDB");
141         Connection dbcon = ds.getConnection();
```

- All other request is read operation, I will just use TestDB in connection pooling. In this case, the read request could go to either master or slave database because it is using localhost.

All other reading request sevelet:

```
88          Context initCtx = new InitialContext();
89
90          Context envCtx = (Context) initCtx.lookup( name: "java:comp/env");
91          DataSource ds = (DataSource) envCtx.lookup( name: "jdbc/TestDB");
92          // Get a connection from dataSource
93          Connection dbcon = ds.getConnection();
```

-

- In this case, all write request are redirect to master database. All read requests are redirect to master and slaves' database equally by using load balancer in instance1.

Task 3

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

- Yes, it is located in the **logfile** folder in the root dictionary. The single_log corresponds to single instance test. The scale_log corresponds to scaled version of testing and in this case, both master and slaves have their own log files.

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

- Yes, it is located in the **html** folder in the root dictionary.

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

- Yes, the Script is called ParseLine.py and it is located at the root dictionary. Before you use it, make sure you use "python -m pip install regex" to download the regex library.
- Make sure the log file is named test.log and locate in the same directory as the script file.
- In the scaled version test, you need to run it for both master and slaves logs and do the average again.

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

- Yes, the readme file is located at the root dictionary and war file is called project1-api-example.war, it is the scaled version war file(with prepare statement, with connection pooling and http protocol) and it is located at both root and the /target/project1-api-example.war.