Task 1:

The File Path: cs122b-winter19-team-29/Fabflix_Web_Project/src/SearchServletAPI.java
The Line numbers for pooled connection: 51 - 69
The How:

- First, we added the database connections into the context.xml and web.xml files in both master and slave AWS instances
- Next, we start with creating the pooled connection by creating a new Context object and binding it to the JDBC driver.
- Finally, we establish the connection to the database 'moviedb' by looking it up in the context of the pooled connection under PooledDB, and assuming everything is fine and not null up to this point, we have connected this instance of the pooled connection context to work with PreparedStatements to query the 'moviedb' database.

```
51                  Context initCtx = new InitialContext();
52
53              Context envCtx = (Context) initCtx.lookup("java:comp/env");
54              jdbcCurrentTime += System.nanoTime() - jdbcStartTime;
55              if (envCtx == null)
56                  out.println("envCtx is NULL @ SearchServlet");
57
58              // Look up our data source
59              jdbcStartTime = System.nanoTime();
60              DataSource ds = (DataSource) envCtx.lookup("jdbc/PooledDB");
61              jdbcCurrentTime = System.nanoTime() - jdbcStartTime;
62
63              if (ds == null)
64                  out.println("ds is null @ SearchServlet.");
65              jdbcStartTime = System.nanoTime();
66                  Connection dbcon = ds.getConnection();
67                  jdbcCurrentTime = System.nanoTime() - jdbcStartTime;
68                  if (dbcon == null)
69                  out.println("dbcon is null @ SearchServlet.");
70
```

- Below are screenshots of PreparedStatements being used for searching in Fabflix:

```
                // Query
                query = "SELECT movies.id, movies.title, movies.year, movies.director,\n"
                        + "GROUP_CONCAT(DISTINCT genres.name ORDER BY genres.name SEPARATOR',') as 'genreList',\n"
                        + "GROUP_CONCAT(DISTINCT genres.id ORDER BY genres.name SEPARATOR',') as 'genreIdList',\n"
                        + "GROUP_CONCAT(DISTINCT stars.name ORDER BY stars.name SEPARATOR',') as 'starList',\n"
                        + "GROUP_CONCAT(DISTINCT stars.id ORDER BY stars.name SEPARATOR',') as 'starIdList', rating\n"
                        + "FROM movies, ratings, genres, genres_in_movies, stars, stars_in_movies\n"
                        + "WHERE movies.id = ratings.movieId AND movies.id = genres_in_movies.movieId AND genres.id = genre
                        + "AND movies.title LIKE ? \n"                                  // 1 fieldsQueryTitle
                        + "AND CAST(movies.year AS CHAR(4)) LIKE ? \n"  // 2 fieldsQueryYear
                        + "AND movies.director LIKE ? \n"                               // 3 fieldsQueryDirector
                        + "GROUP BY movies.id\n"
                        + "HAVING starList LIKE ? \n";                  // 4 fieldsQueryStar
                if (sort.equalsIgnoreCase("title"))
                        query += "ORDER BY title ";
                else
                        query += "ORDER BY rating ";
                if (order.equalsIgnoreCase("ASC"))
                        query += "ASC\n";
                else
                        query += "DESC\n";
                query += "LIMIT ? OFFSET ?;\n";                                         // 7 results, 8 (Integer.parseInt(p

        }


        // Prepare the statement before populating the ?'s based on search type
        PreparedStatement stmt = dbcon.prepareStatement(query);

        // Fill in the ? parameters based on search type
        if (search.equals("genre"))
        {
                // 1 id, 2 sort, 3 order, 4 results, 5 (Integer.parseInt(page) * Integer.parseInt(results))
                stmt.setString(1, id);
                stmt.setInt(2, Integer.parseInt(results));
                stmt.setInt(3, Integer.parseInt(page) * Integer.parseInt(results));
        }
        else if (search.equals("title"))
        {
                // 1 id, 2 sort, 3 order, 4 results, 5 (Integer.parseInt(page) * Integer.parseInt(results))
                stmt.setString(1, id + "%");
                stmt.setInt(2, Integer.parseInt(results));
                stmt.setInt(3, Integer.parseInt(page) * Integer.parseInt(results));
        }
        else if (search.equals("fts"))
        {
                System.out.println("title=" + id);
                stmt.setString(1, id);
                stmt.setInt(2,  Integer.parseInt(results));
                stmt.setInt(3,  Integer.parseInt(page) * Integer.parseInt(results));
        }
        else
        {
                // 1 fieldsQuery, 2 test, 3 sort, 4 order, 5 results, 6 (Integer.parseInt(page) * Integer.parseInt(results)
                stmt.setString(1, "%" + fieldsQueryTitle + "%");
                stmt.setString(2, "%" + fieldsQueryYear + "%");
                stmt.setString(3, "%" + fieldsQueryDirector + "%");
                stmt.setString(4, "%" + fieldsQueryStar + "%");
                stmt.setInt(5, Integer.parseInt(results));
                stmt.setInt(6, Integer.parseInt(page) * Integer.parseInt(results));
        }
```

Note: The remainder of the task 1 document will show a few more file paths for other servlets utilizing the pooled connection code and PreparedStatements.

The File Path:
cs122b-winter19-team-29/Fabflix_Web_Project/src/SingleGenreServletAPI.java
The Line numbers for pooled connection: 41 - 57
More screenshots:

```
41                  try
42                  {
43                  Context initCtx = new InitialContext();
44
45              Context envCtx = (Context) initCtx.lookup("java:comp/env");
46              if (envCtx == null)
47                  out.println("envCtx is NULL @ SingleGenreServlet");
48
49              // Look up our data source
50              DataSource ds = (DataSource) envCtx.lookup("jdbc/PooledDB");
51
52              if (ds == null)
53                  out.println("ds is null @ SingleGenreServlet.");
54
55                  Connection dbcon = ds.getConnection();
56                  if (dbcon == null)
57                  out.println("dbcon is null @ SingleGenreServlet.");
58
59                          // Create the SQL query
60                          String query = "SELECT genres.id, genres.name, movies.id, movies.title \n" +
61                                         "FROM genres, genres_in_movies, movies \n" +
62                                         "WHERE genres.id = ?\n" +
63                                         "AND genres.id = genres_in_movies.genreId \n" +
64                                         "AND movies.id = genres_in_movies.movieId;";
65                                  // 1 genreId
66
67                          PreparedStatement statement = dbcon.prepareStatement(query);
68
69                          statement.setString(1, genreId);
70
71                          ResultSet rs = statement.executeQuery();
```

The File Path:
cs122b-winter19-team-29/Fabflix_Web_Project/src/SingleMovieServletAPI.java
The Line numbers for pooled connection: 44 - 58
More screenshots:

```
44                  Context initCtx = new InitialContext();
45
46              Context envCtx = (Context) initCtx.lookup("java:comp/env");
47              if (envCtx == null)
48                  out.println("envCtx is NULL @ SingleMovieServlet");
49
50              // Look up our data source
51              DataSource ds = (DataSource) envCtx.lookup("jdbc/PooledDB");
52
53              if (ds == null)
54                  out.println("ds is null @ SingleMovieServlet.");
55
56                  Connection dbcon = ds.getConnection();
57                  if (dbcon == null)
58                  out.println("dbcon is null @ SingleMovieServlet.");
59
60                          // Construct a query with parameter represented by "?"
61                          String query = "SELECT movies.id, title, year, director,\n"
62                                  + "GROUP_CONCAT(DISTINCT genres.name ORDER BY genres.name SEPARATOR',') as 'genreList',\n"
63                                  + "Group_CONCAT(DISTINCT genres.id ORDER BY genres.name SEPARATOR',') as 'genreIdList',\n"
64                                  + "GROUP_CONCAT(DISTINCT stars.name ORDER BY stars.name SEPARATOR',') as 'starList',\n"
65                                  + "GROUP_CONCAT(DISTINCT stars.id ORDER BY stars.name SEPARATOR',') as 'starIdList'\n"
66                                  + "FROM movies, genres, genres_in_movies, stars, stars_in_movies\n"
67                                  + "WHERE movies.id = ?\n"
68                                  + "AND movies.id = genres_in_movies.movieId "
69                                  + "AND genres.id = genreId "
70                                  + "AND movies.id = stars_in_movies.movieId "
71                                  + "AND starId = stars.id";
72                                  // 1 id
73
74                          // Declare our statement
75                          PreparedStatement statement = dbcon.prepareStatement(query);
76                          //Statement statement = dbcon.createStatement();
77
78                          // Set the parameter represented by "?" in the query to the id we get from url,
79                          // num 1 indicates the first "?" in the query
80                          statement.setString(1, id);
81
82                          // Perform the query
83                          //ResultSet rs = statement.executeQuery();
84                          ResultSet rs = statement.executeQuery();
```

The File Path:
cs122b-winter19-team-29/Fabflix_Web_Project/src/CheckoutServletAPI.java
The Line numbers for pooled connection: 56 - 70
More screenshots:

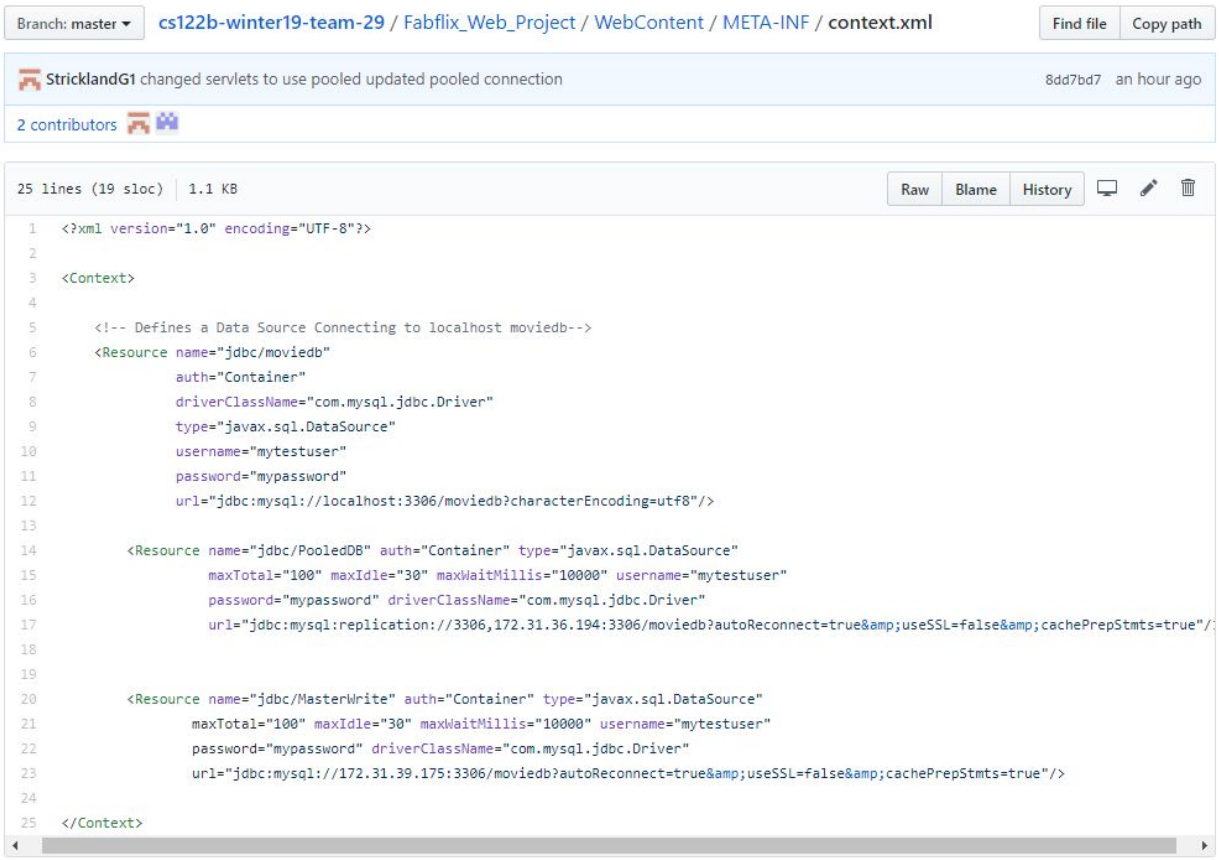
```
56                  Context initCtx = new InitialContext();
57
58              Context envCtx = (Context) initCtx.lookup("java:comp/env");
59              if (envCtx == null)
60                  out.println("envCtx is NULL @ CheckoutServlet");
61
62              // Look up our data source
63              DataSource ds = (DataSource) envCtx.lookup("jdbc/PooledDB");
64
65              if (ds == null)
66                  out.println("ds is null @ CheckoutServlet.");
67
68                  Connection dbcon = ds.getConnection();
69                  if (dbcon == null)
70                  out.println("dbcon is null @ CheckoutServlet.");
71
72                  String query = "SELECT customers.id\n" +
73                                  "FROM creditcards, customers\n" +
74                                  "WHERE customers.email = ?\n" +                                          // 1
75                                  "AND creditcards.id = ?\n" +                                             // 2
76                                  "AND customers.ccId = creditcards.id\n" +
77                                  "AND creditcards.firstName = ?\n" +                                 // 3
78                                  "AND creditcards.lastName = ?\n" +                                       // 4
79                                  "AND creditcards.firstName = customers.firstName\n" +
80                                  "AND creditcards.lastName = customers.lastName\n" +
81                                  "AND expiration = ?;";                                                                  // 5
82
83                  PreparedStatement statement = dbcon.prepareStatement(query);
84
85                  statement.setString(1, username);
86                  statement.setString(2, ccId);
87                  statement.setString(3, ccFname);
88                  statement.setString(4, ccLname);
89                  statement.setString(5, ccExp);
90
91                  ResultSet rs = statement.executeQuery();
92
```

```
107                 // Loop through every item in the cart, and enter each movie sold as a sales entry for that customer
108                 for (Item item : tempCart)
109                 {
110                         System.out.println(customerId + " " + item.getMovieId() + " " + item.getMovieTitle() + " " + date + " x" + item.get
111
112                     String salesQuery = "INSERT INTO sales (customerId, movieId, saleDate, quantity)"
113                                         + "VALUES (?, ?, ?, ?)";
114
115                         PreparedStatement salesStatement = dbcon.prepareStatement(salesQuery);
116
117                         salesStatement.setInt(1, Integer.parseInt(customerId));
118                         salesStatement.setString(2, item.getMovieId());
119                         salesStatement.setDate(3, java.sql.Date.valueOf(date));
120                         salesStatement.setInt(4, item.getQuantity());
121
122                         salesStatement.executeUpdate();
123
124                         // Don't forget to close the statement to complete the query transaction!
125                         salesStatement.close();
126                 }
```

Task 2:

We updated the Fabflix project's context.xml and web.xml to include the connection pooling resource for the local version, which is also applied to the master and slave versions when the project's .war file is deployed to the respective AWS instances. We pooled the master and slave connections under one "PooledDB" resource, and another for just strictly writing to the master instance under "MasterWrite":

cs122b-winter19-team-29/Fabflix_Web_Project/WebContent/META-INF/context.xml



Branch: master ▾ cs122b-winter19-team-29 / Fabflix_Web_Project / WebContent / META-INF / context.xml Find file Copy path

StricklandG1 changed servlets to use pooled updated pooled connection 8dd7bd7 an hour ago

2 contributors

25 lines (19 sloc) 1.1 KB Raw Blame History

```
<?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="mypassword"
              url="jdbc:mysql://localhost:3306/moviedb?characterEncoding=utf8"/>

        <Resource name="jdbc/PooledDB" auth="Container" type="javax.sql.DataSource"
                  maxTotal="100" maxIdle="30" maxWaitMillis="10000" username="mytestuser"
                  password="mypassword" driverClassName="com.mysql.jdbc.Driver"
                  url="jdbc:mysql:replication://3306,172.31.36.194:3306/moviedb?autoReconnect=true&amp;useSSL=false&amp;cachePrepStmts=true"/>


        <Resource name="jdbc/MasterWrite" auth="Container" type="javax.sql.DataSource"
                maxTotal="100" maxIdle="30" maxWaitMillis="10000" username="mytestuser"
                password="mypassword" driverClassName="com.mysql.jdbc.Driver"
                url="jdbc:mysql://172.31.39.175:3306/moviedb?autoReconnect=true&amp;useSSL=false&amp;cachePrepStmts=true"/>

</Context>
```

cs122b-winter19-team-29/Fabflix_Web_Project/WebContent/WEB-INF/web.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
  <display-name>CS_122B_Fablix_Project_API_Version</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
    <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/moviedb</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
        <res-auth>Container</res-auth>
    </resource-ref>
    <resource-ref>
        <description>pooled connection resource</description>
        <res-ref-name>jdbc/PooledDB</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
        <res-auth>Container</res-auth>
    </resource-ref>
    <resource-ref>
        <description>master MySQL connection resource for writing</description>
        <res-ref-name>jdbc/MasterWrite</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
        <res-auth>Container</res-auth>
    </resource-ref>
</web-app>
```

Note: For task 2, we noticed that we can access the project from GCP's public IP at port 80 using the public IP's of the master and slave AWS instances in the load balancer, but we need to manually type in a HTML page in the URL (i.e. /index.html).

Details regarding our AWS/GCP IP's and instances:

AWS Master instance:

- Public IP: 3.16.158.144
- Public DNS: ec2-3-16-158-144.us-east-2.compute.amazonaws.com

AWS Slave instance:

- Public IP: 13.59.184.185
- Public DNS: ec2-13-59-184-185.us-east-2.compute.amazonaws.com

GCP instance:

- Public IP: 35.236.87.69

Project URL:

- http:// (Pick any of the IP's above) :80/CS_122B_Fablix_Project_API_Version/index.html