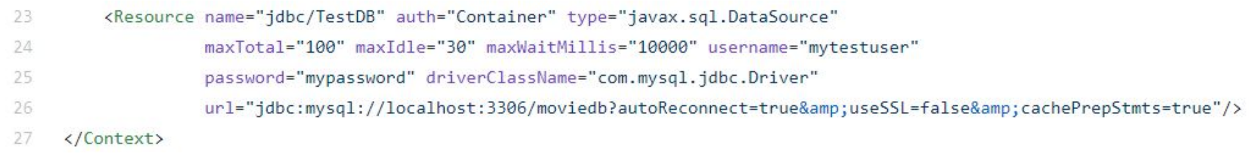
## Task 1

- How did you use connection pooling?

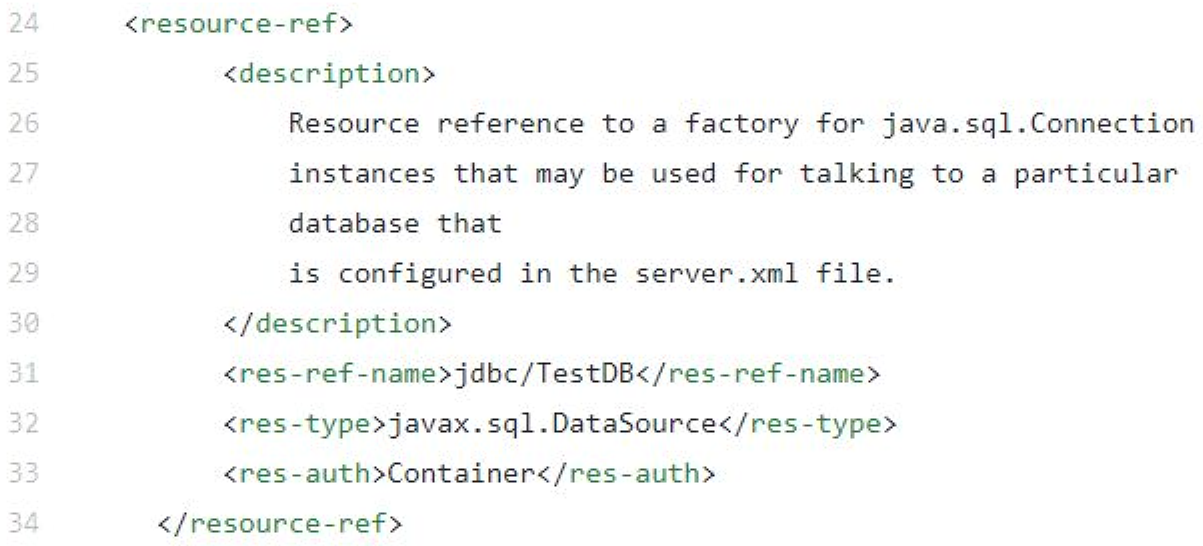
In our context.xml, we added another Resource tag and defined another DataSource that had a maxTotal of 100, a maxIdle of 30, and a maxWaitMillis of 10000. We named this new DataSource TestDB. In our web.xml, we added a new resource-ref tag of TestDB so our program could refer to this new DataSource. Then, in all of the servlets that opened a database connection, we did a lookup to TestDB instead of moviedb.

- File name, line numbers as in Github
  MovieServlet.java : 162-172
  SingleMovieServlet.java : 46-53
  SingleStarServlet.java : 50 - 58

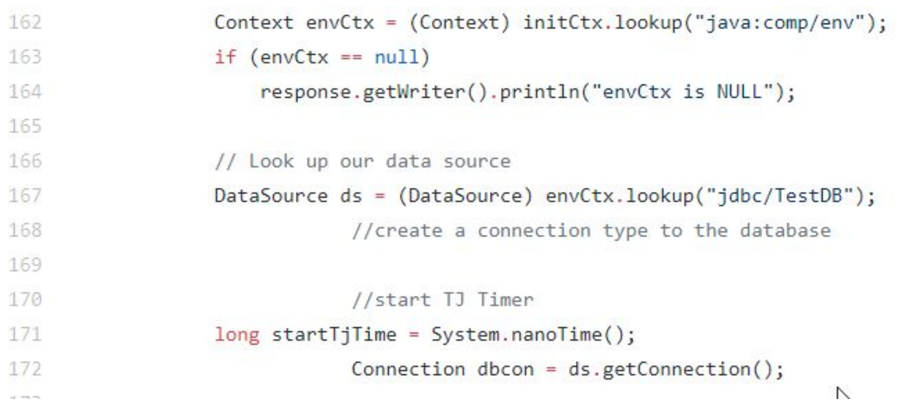
- Snapshots showing use in your code



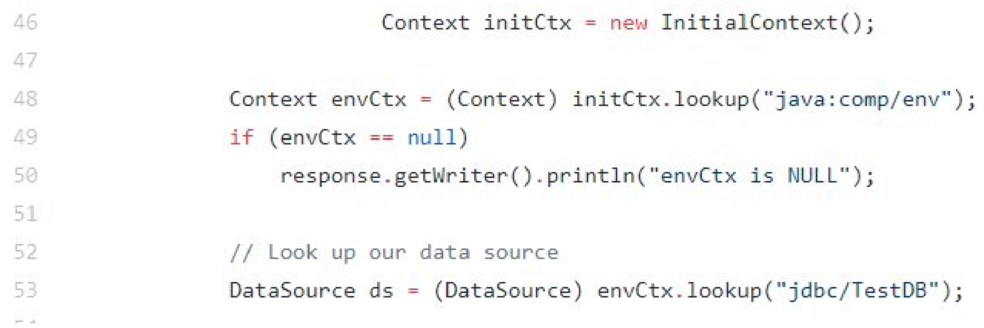
```
    <Resource name="jdbc/TestDB" auth="Container" type="javax.sql.DataSource"
              maxTotal="100" maxIdle="30" maxWaitMillis="10000" username="mytestuser"
              password="mypassword" driverClassName="com.mysql.jdbc.Driver"
              url="jdbc:mysql://localhost:3306/moviedb?autoReconnect=true&amp;useSSL=false&amp;cachePrepStmts=true"/>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



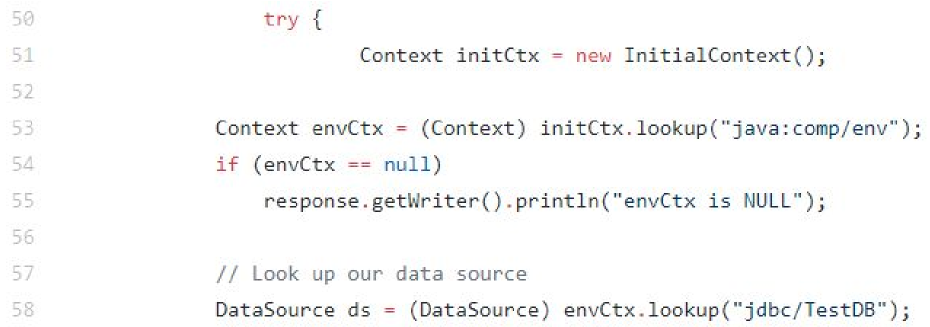
```
    <resource-ref>
          <description>
              Resource reference to a factory for java.sql.Connection
              instances that may be used for talking to a particular
              database that
              is configured in the server.xml file.
          </description>
         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



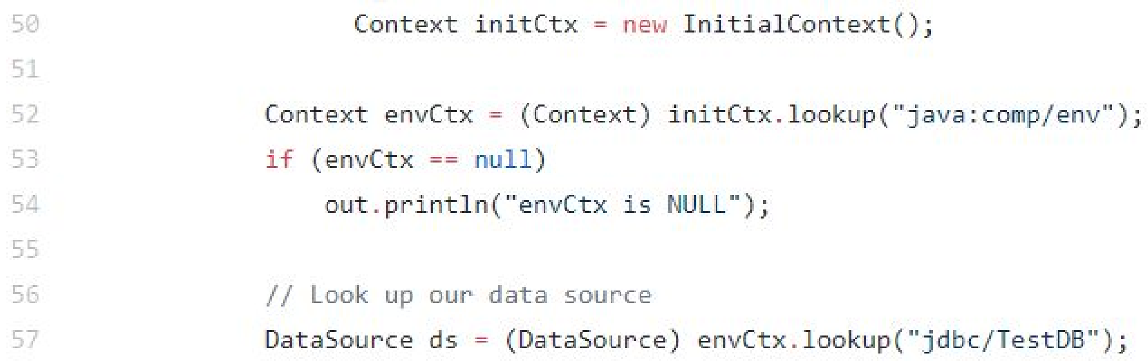
```
Context envCtx = (Context) initCtx.lookup("java:comp/env");
if (envCtx == null)
    response.getWriter().println("envCtx is NULL");

// Look up our data source
DataSource ds = (DataSource) envCtx.lookup("jdbc/TestDB");
            //create a connection type to the database

            //start TJ Timer
long startTjTime = System.nanoTime();
            Connection dbcon = ds.getConnection();
```



```
                Context initCtx = new InitialContext();

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

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



```
    try {
            Context initCtx = new InitialContext();

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

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



```
            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/TestDB");
```

- How did you use Prepared Statements?

In our MovieServlet.java, we replaced the query parameters with a question mark (?) and added each query parameter to an ArrayList. We did this because we used multiple if/else statements in constructing our query. Right before executing the query, we declared a Prepared Statement.

We looped through the ArrayList and called setString on each query parameter to inject it into the Prepared Statement.

- File name, line numbers as in Github
  src/MovieServlet.java : 67 - 175

- Snapshots showing use in your code

```java
if(mode.equals("browse")) {
        String genre_id = request.getParameter("id");
        if(genre_id != null) {//means that there is an id
                base_query += "group by movies.id, movies.title, movies.year, movies.director, r.rating\n" +
                                          "having find_in_set(?, genre_id)\n";
                query_items.add(genre_id);
                }
        else {//means that they have selected browsing by letter
                String first_letter = request.getParameter("search");
                //first_letter
                first_letter = first_letter + '%';
                base_query += "where movies.title like ?\n" +
                                          "group by movies.id, movies.title, movies.year, movies.director, r.rating\n
                query_items.add(first_letter);
        }
}
else if(mode.equals("search")) {
        String search_query = "";

        String search_title = request.getParameter("title");
        String search_director = request.getParameter("director");
        String search_year = request.getParameter("year");
        String search_star = request.getParameter("star");


        boolean searchTitleExist = search_title != null && !search_title.isEmpty();
        boolean searchDirectorExist = search_director != null && !search_director.isEmpty();
        boolean searchYearExist = search_year != null && !search_year.isEmpty();
        boolean searchStarExist = search_star != null && !search_star.isEmpty();

        ArrayList<String> queryList = new ArrayList<String>();

        if(searchTitleExist || searchDirectorExist || searchYearExist || searchStarExist) {
                search_query += "where ";

                if(searchTitleExist) {
                        String[] words = search_title.split(" ");
                        String base_string = "";
                        for(int i = 0; i < words.length; i++) {
                                base_string += "+" + words[i] + "*";
                        }

                        String title_search_query = "match(movies.title) against(? in boolean mode)\n";
                        queryList.add(title_search_query);
                        query_items.add(base_string);
                }
                if(searchDirectorExist) {
                        //search_director
                        search_director = '%' + search_director + '%';
                        queryList.add("movies.director like ?\n");
                        query_items.add(search_director);
                }
                if(searchYearExist) {
                        //search_year
                        search_year = '%' + search_year + '%';
                        queryList.add("movies.year like ?\n");
                        query_items.add(search_year);
                }
                if(searchStarExist) {
                        //search_star
                        search_star = '%' + search_star + '%';
                        queryList.add("s.name like ?\n");
                        query_items.add(search_star);
                }
```

```
                                        search_query += "and " + queryList.get(i);
                                }
                        }

                        base_query += search_query + "group by movies.id, movies.title, movies.year, movies.director, r.rating\n";

                }

                String[] determine_sort = order.split(" ");
                base_query += "order by " + determine_sort[0];
                if(determine_sort.length > 1) {
                        if(determine_sort[1].equals("Highest") || determine_sort[1].equals("Z-0")) {
                                base_query += " desc\n";
                        }
                        else if(determine_sort[1].equals("Lowest") || determine_sort[1].equals("0-Z")) {
                                base_query += " asc \n";
                        }
                }
                else {
                        base_query += " desc \n";
                }
                //limit and page
                base_query += " limit " + itemLimit + " offset " + page;

                Context initCtx = new InitialContext();

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

        // Look up our data source
        DataSource ds = (DataSource) envCtx.lookup("jdbc/TestDB");
                //create a connection type to the database

                //start TJ Timer
        long startTjTime = System.nanoTime();
                Connection dbcon = ds.getConnection();


                PreparedStatement statement = dbcon.prepareStatement(base_query);
```

## **Task 2**

- Address of AWS and Google instances
- AWS
    - Port 80: http://52.14.37.54/project1/
    - Port 8080: http://52.14.37.54:8080/project1/
- Google
    - Port 80: http://35.235.92.4/project1/
    - Port 8080: http://35.235.92.4:8080/project1/
- Have you verified that they are accessible? Does Fablix site get opened both on Google's 80 port and AWS' 8080 port?
- Yes, they are both accessible.

Yes, both Google's 80 port and AWS' 8080 port are accessible.

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

  Our original instance will send requests to either the master instance or the slave instance. In our apache configuration file, we added a load balancer member for both the master and slave instances. This way, each request will be redirected to either the master or slave instance. We added a Proxy tag to the configuration file, with BalancerMembers inside. We also added a ProxyPass inside the body of the VirtualHost tag.
    - File name, line numbers as in Github
      WEB-CONTENT/META-INF/Context.xml : 6 - 26
      WEB-CONTENT/WEB-INF/Web.xml : 12 - 24
    - Snapshots

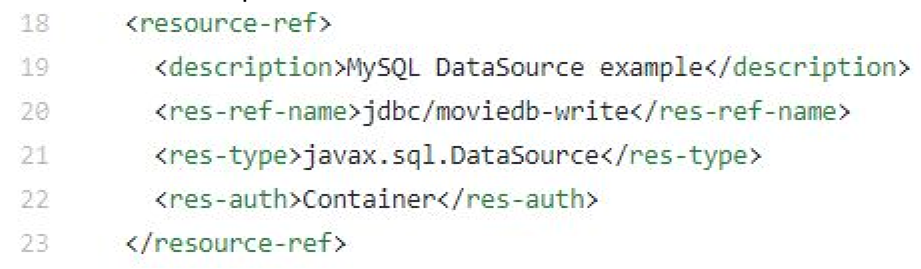
```xml
    <!-- Defines a Data Source Connecting to localhost moviedb-->
    <Resource name="jdbc/moviedb"
              auth="Container"
              driverClassName="com.mysql.jdbc.Driver"
              type="javax.sql.DataSource"
              username="mytestuser"
              password="mypassword"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

   <Resource name="jdbc/moviedb-write"
              auth="Container"
              driverClassName="com.mysql.jdbc.Driver"
              type="javax.sql.DataSource"
              username="mytestuser"
              password="mypassword"
              url="jdbc:mysql://172.31.32.117:3306/moviedb"/>


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

```xml
<resource-ref>
  <description>MySQL DataSource example</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>MySQL DataSource example</description>
  <res-ref-name>jdbc/moviedb-write</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/TestDB</res-ref-name>
      <res-type>javax.sql.DataSource</res-type>
      <res-auth>Container</res-auth>
  </resource-ref>
```

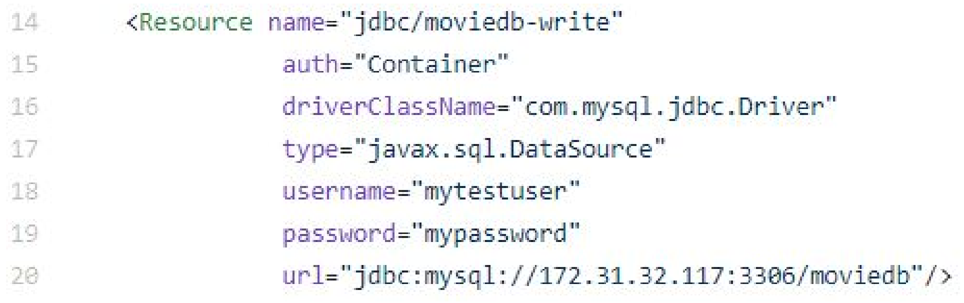
- How read/write requests were routed?
    - In the context XML, we added a new DataSource named moviedb-write to make all write requests go to the master instance. Then, in the servlets that do a write to the database (AddMovieServlet.java, AddStarServlet.java and CheckoutServlet.java), we made the database connection refer to moviedb-write instead of moviedb.

    - File name, line numbers as in Github
      WEB-CONTENT/WEB-INF/Web.xml : 18 - 23
      WEB-CONTENT/META-INF/Context.xml : 14 - 20
      src/AddMovieServlet.java : 66 - 73
      src/AddStarServlet.java : 53 - 61
      src/CheckoutServlet.java : 59 - 67

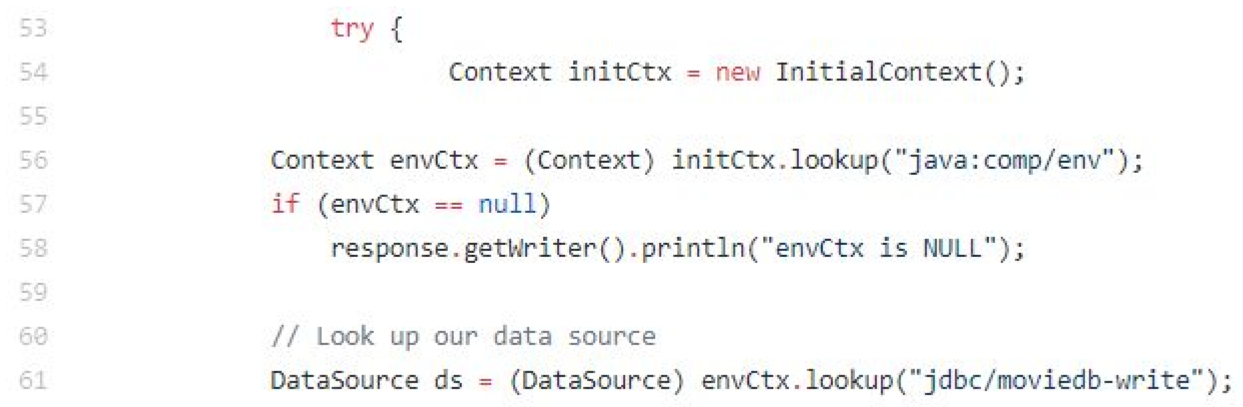
    - Snapshots



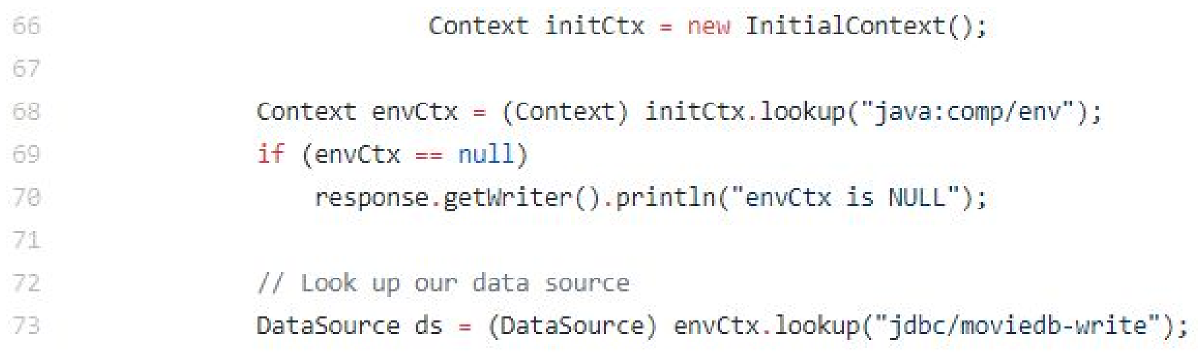
```
18    <resource-ref>
19      <description>MySQL DataSource example</description>
20      <res-ref-name>jdbc/moviedb-write</res-ref-name>
21      <res-type>javax.sql.DataSource</res-type>
22      <res-auth>Container</res-auth>
23    </resource-ref>
```



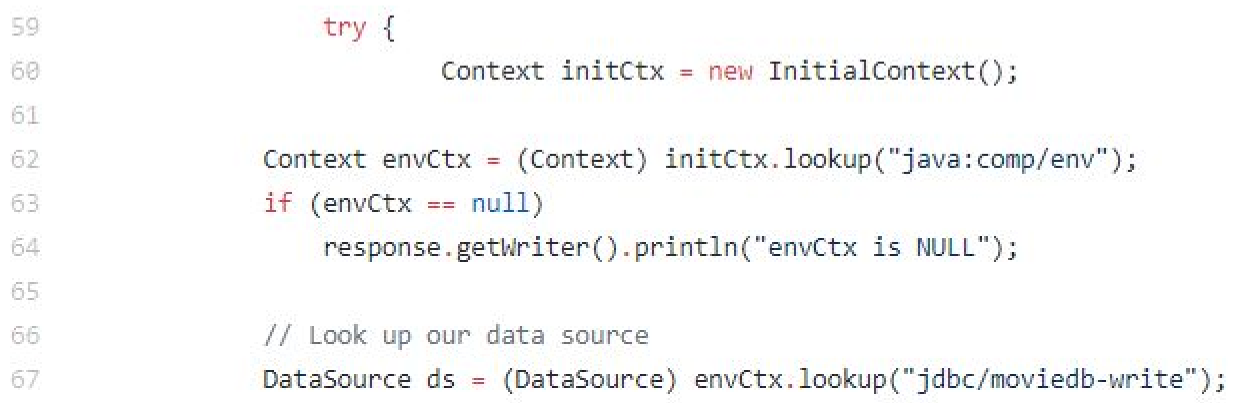
```
14    <Resource name="jdbc/moviedb-write"
15              auth="Container"
16              driverClassName="com.mysql.jdbc.Driver"
17              type="javax.sql.DataSource"
18              username="mytestuser"
19              password="mypassword"
20              url="jdbc:mysql://172.31.32.117:3306/moviedb"/>
```



```
            try {
                    Context initCtx = new InitialContext();

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

        // Look up our data source
        DataSource ds = (DataSource) envCtx.lookup("jdbc/moviedb-write");
```



```
                    Context initCtx = new InitialContext();

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

        // Look up our data source
        DataSource ds = (DataSource) envCtx.lookup("jdbc/moviedb-write");
```



```
            try {
                    Context initCtx = new InitialContext();

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

        // Look up our data source
        DataSource ds = (DataSource) envCtx.lookup("jdbc/moviedb-write");
```

**Task 3**

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

  Yes, The log files will be in the landing page of the github repository in a folder called "JMeter-Logs", and the directory is cs122b-winter19-team-31/JMeter-Logs.

- Have you uploaded the HTML file (with all sections including analysis, written up) to Github? Where is it located?
  Yes, the HTML file along with analysis is included inside the root directory. The directory for this information is [cs122b-winter19-team-31/JMeter-report].

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

  Yes, The script is located within the project1 folder, the directory is
  cs122b-winter19-team-31/

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

  Yes, WAR file and README file will be in the initial github repo page :
  cs122b-winter19-team-31/