**Task 1**

* Connection pooling is a mechanism to create and maintain a collection of JDBC connection objects. The primary objective of maintaining the pool of connection object is to leverage re-usability. A new connection object is created only when there are no connection objects available to reuse. This technique can improve overall performance of the application.
* Copied “mysql-connector-java-5.1.46.jar” to our Tomcat/lib directory on our AWS instance/production system.
* Changed the code block in \META-INF\context.xml

\WEB-INF\web.xml

All the connections in all the Servlets

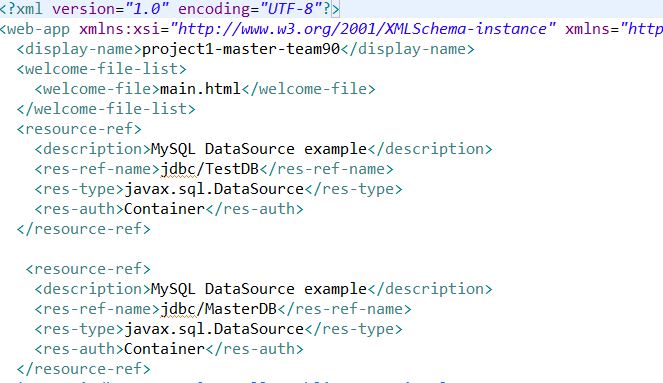
to enable connection pooling. Based on the code sections provided in Tomcat Pooling example.

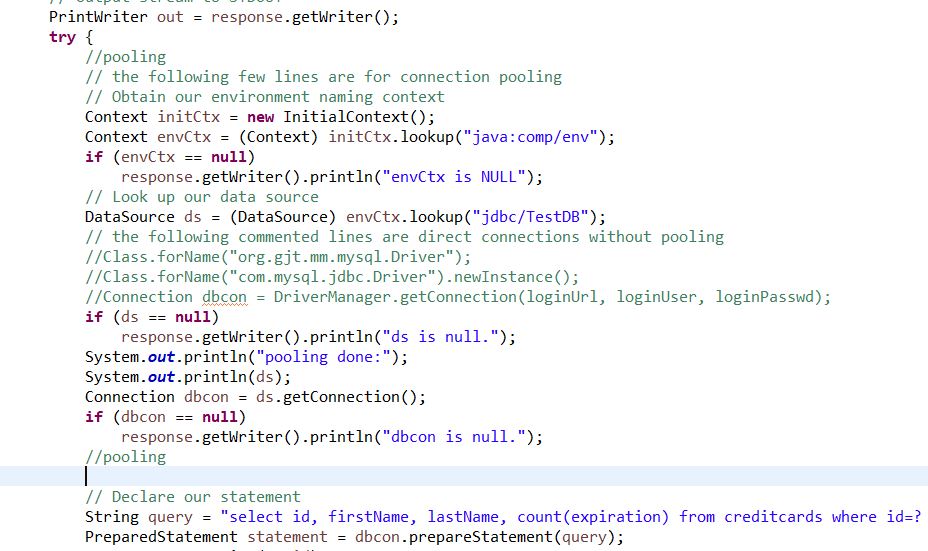
* Changed all the JDBC statements to Prepared Statements.
* File name, line numbers as in Github

Context.xml



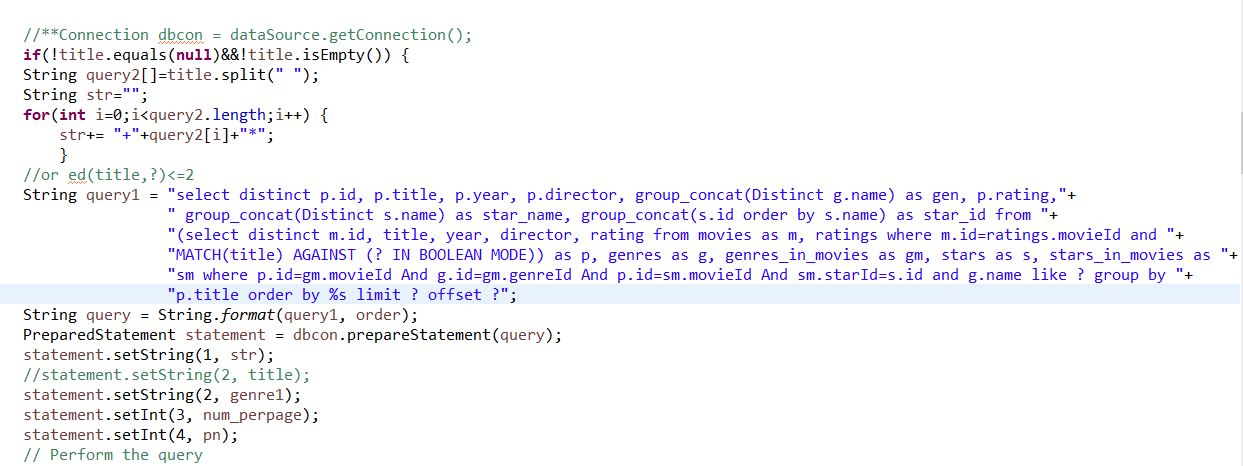
Web.xml





* We had already changed all our query statements to prepared statements in the earlier projects. All those queries had a place holder in them, so prepared statement was helpful, as the statements got precompiled and the performance of the search improved.
* File name, line numbers as in Github

Snapshots of Prepared Statements used in Search functionality in our project.



Snapshots of Prepared Statements used in DashboardLogin servlets:



Snapshots of Prepared Statements used in Dashboard servlets:





**Task 2**

* AWS main Instance: <http://13.57.32.177:8080>
* AWS Master Instance: <http://54.193.70.160:8080>
* AWS Slave Instance: <http://13.57.222.241:8080>

Yes, Fabflix site opens on both AWS’ 8080 port.

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

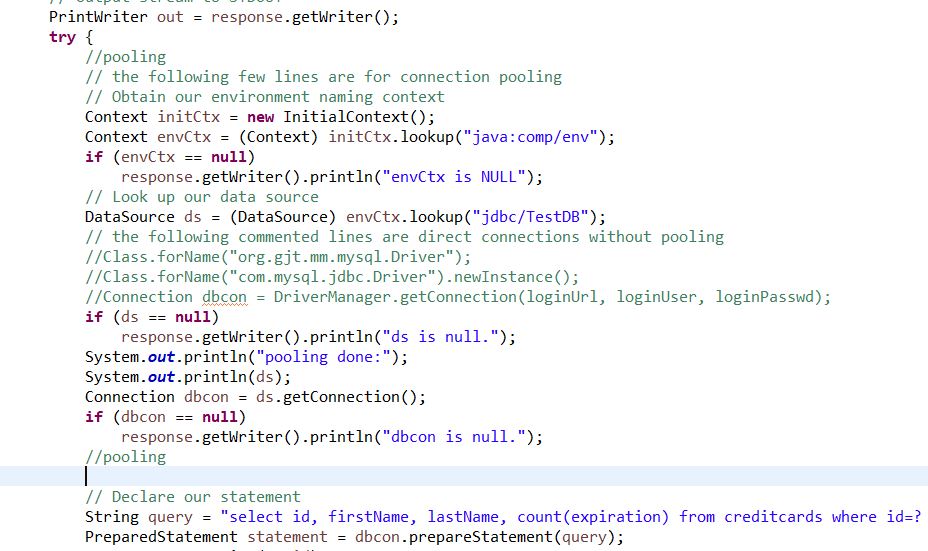
In connection pooling, when DataSource uses connection pooling, the lookup return a connection from the pool of available connection objects. If there is no available connection, the lookup creates a new connection. The application establishes a connection to the database and accesses data in the usual way. So with two backend SQL (master, slave) connected in the connection pooling.

We changed our main instance to a load balancer. A load balancer is a device that distributes network or application traffic across a cluster of servers. Load balancing improves responsiveness and increases availability of applications.

So wherever we are reading from the database we have used the load balancer (localhost/13.57.32.177). When one application server becomes unavailable or too much loaded, the load balancer directs all new application requests to other available servers in the pool.

* + File name, line numbers as in Github





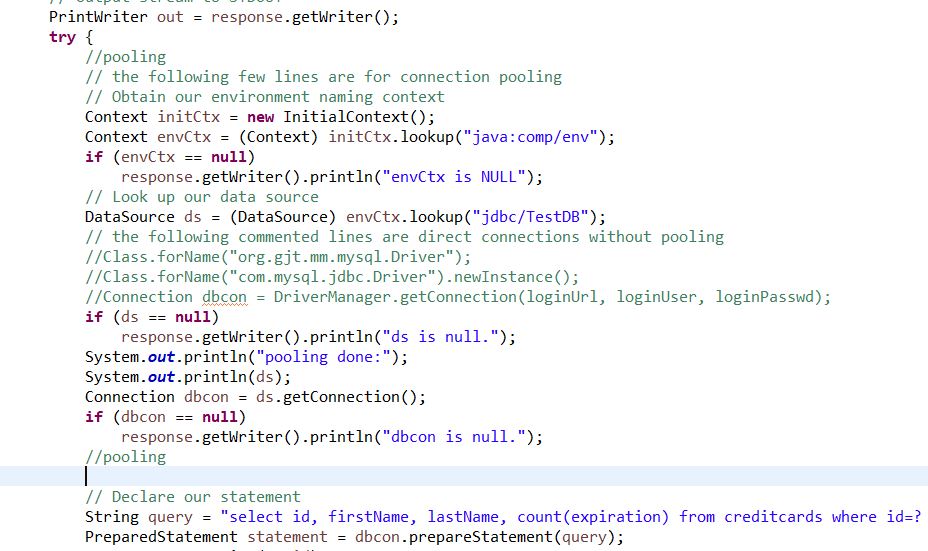


So, the load balancer then distributes the SQL requests to either of the two backend SQLs from the pool.

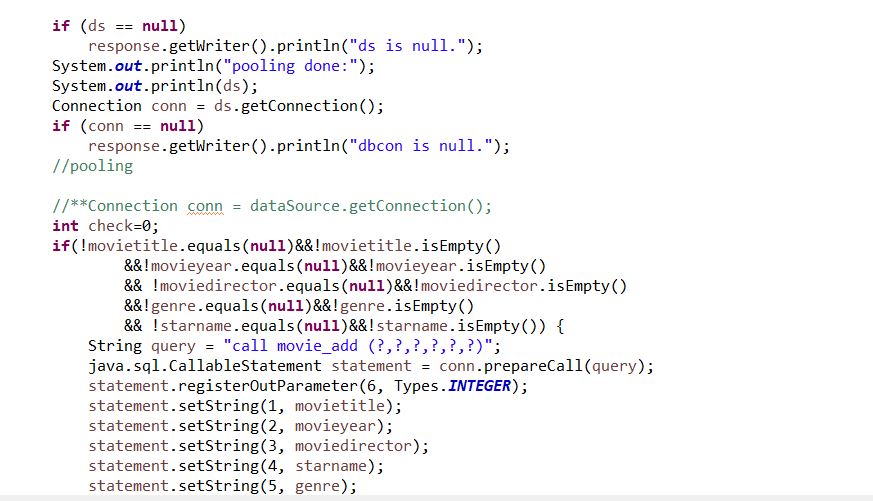
* How read/write requests were routed?
* So wherever we are reading from the database we have used the load balancer (localhost/13.57.32.177). When one application server becomes unavailable or too much loaded, the load balancer directs all new application requests to other available servers in the pool.
* But when we write to the database, like during adding sales details or stars/ movies addition to database, we use the master SQL only. The slave SQL will automatically replicate the whole Master SQL.
  + File name, line numbers as in Github



Connecting to Database for reading from database:



Connecting to Databases for writing to database:



**Task 3**

* Have you uploaded the log files to Github? Where is it located?
* Have you uploaded the HTML file (with all sections including analysis, written up) to Github? Where is it located?
* Have you uploaded the script to Github? Where is it located?
* Have you uploaded the WAR file and README to Github? Where is it located?