Apache tomcat runs natively using a **thread pool**, meaning that the server relieves itself from directly managing its threads. Instead of allocating new threads; whenever it needs a thread it asks for it from the pool, and when it is done, the thread is returned to the pool. The thread pool can now be used to implement sophisticated thread management techniques, such as:

1. Keeping threads "open" and reusing them over and over again. This saves the trouble associated with creating and destroying threads continuously.
   * Usually the administrator can instruct the pool not to keep too many idle threads, freeing them if needed.
2. Setting an upper bound on the number of threads used concurrently. This prevents the resources allocation problem associated with unlimited thread allocation.
   * If the container maxed out to the threads upper limit, and a new request arrives, the new request will have to wait for some other (previous) request to finish and free the thread used to service it.

This default setup allows our server to handle a large number of concurrent requests, improving its performance at very little cost. In the case of our implementation: every connection from a user is handled by the thread pool, whereas the connections to the SQL server is handled by the JDBC which queue’s each command to the database, implementing them in the order in which they are received.

Information cited from <http://www.jajakarta.org/tomcat/tomcat3.2-4.0/tomcat-3.2.3/doc/uguide/tomcat_ug.html>