CNT 4714 – Project Three – Summer 2022

Title: "Project Three: Developing A Three-Tier Distributed Web-Based Application"
Points: 100 points (bonus problem potentially adds 15 points – see page 14.)
Due Date: Thursday August 4, 2022 by 11:59 pm (WebCourses time)

Objectives: To incorporate many of the techniques you've learned so far this semester into a distributed three-tier web-based application which uses servlets and JSP technology running on a Tomcat container/server to access and maintain a persistent MySQL database using JDBC.

Description: In this assignment you will utilize a suppliers/parts/jobs/shipments database (creation/population script available on Webcourses under Project 3) as the back-end database. Front-end access to this database by end users will occur through a single page displayed in the client's web browser. The schema of the backend database consists of four tables with the following schemas for each table:

suppliers (snum, sname, status, city) //information about suppliers
parts (pnum, pname, color, weight, city) //information about parts
jobs (jnum, jname, numworkers, city) //information about jobs
shipments (snum, pnum, jnum, quantity) //suppliers ship parts to jobs in specific quantities

The database will enforce referential integrity via foreign key constraints. The primary key for the shipments table is a composite key consisting of three foreign keys (the primary keys in the suppliers, parts, and jobs tables). Referential integrity means that a shipment record cannot exist unless it links back (via referential integrity) to existing entities on all foreign keys. Thus, a shipment record cannot exist unless the referenced snum, pnum, and jnum already exist in their respective tables.

The first-tier (user-level front-end) of your web-application will be three different JSP pages, one of which handles root-level user clients, another which handles non-root-level clients, to enter arbitrary SQL commands into a window (i.e. a form) and submit them to a server application for processing. The third JSP page will be a dedicated data entry form for entering new shipment records into the shipments table.

The front-ends of all three applications (**and only the front-end**) will utilize JSP technology. The front-ends for the root-level and client-level users, will provide the user a simple form in which they will enter a SQL command (any DML, DDL, or DCL command could theoretically be entered by the user, however we will restrict to queries, insert, update, replace, and delete commands). These two front-ends will provide only three buttons for the user, an "Execute Command" button that will cause the execution of the SQL command currently in the input window, a "Reset Form" button that simply clears any content currently in the form input area, and a "Clear Results" button that will erase the currently displayed data (user optional). The third front-end will be utilized only by naïve data-entry users by filling in a form. The data-entry users will not enter SQL commands to accomplish their tasks. Rather, their web-application will use the preparedStatement interface and extract the parameters from their form and issue the SQL command in the background.

The front-ends will run on any web-based browser that you would like to use. The applications will connect to the backend database via properties files dependent on which front-end page is utilized. This connection must be handled using properties read from a properties file. You will have three different properties files, one for the root-level users, one for the client-level users (all the same as project 2 except for the different database), and one for the data entry-level users.

The second-tier servlets, are in charge of handling the SQL command interface for the users. The root-level user app (and the data entry level app – see below), will also implement the server-side business/application logic. This logic will increment by 5, the status of a supplier anytime that supplier is involved in the insertion/update of a shipment record in which the quantity is greater than or equal to 100. Note that any update of quantity >= 100 will affect any supplier involved in a shipment with a quantity >= 100. The example screen shots illustrate this case. An insert of a shipment tuple (S5, P6, J7, 400) will cause the status of every supplier who has a shipment with a quantity of 100 or greater to be increased by 5. In other words, even if a supplier's shipment is not directly affected by the update, their status will be affected if they have any shipment with quantity >= 100. **(See page 14 for a bonus problem that implements a modified version of this business rule.)** The business logic of the second tier will reside in the servlet on the Tomcat web-application server (server-side application). This means that the business logic is not to be implemented in the DBMS via a trigger.

The client-level servlet will handle the SQL command interface, just as the root-level servlet does, however, due to the restrictions on the client-level privileges, no business-logic will be implemented in this application.

The data entry-level servlet will provide the user a template (form) for the shipments table data to be entered and will execute the necessary updating command by extracting the parameters from the form and issuing a prepared statement update to the database. You may want to refer to the JDBC notes from Module 3 to refresh your memory of how the preparedStatement() interface differs from the normal Statement() interface.

The third-tier (back-end) is the persistent MySQL database described above and is under control of the MySQL DBMS server. You will create and maintain this database via the creation/population script. See the important note below concerning when/how to re-run this script for your final submission.

References:
Notes: Lecture Notes for MySQL installation and use. Documentation for MySQL available at: http://www.mysql.com. More information on JDBC can be found at: http://www.oracle.com/technetwork/java/javase/jdbc/index.html . More information on Tomcat can be found at http://tomcat.apache.org. Lecture Notes for Servlets. Lecture Notes for JSPs.

Restrictions:
Your source file shall begin with comments containing the following information:

```
/*  Name:
    Course: CNT 4714 – Summer 2022 – Project Three
    Assignment title:  A Three-Tier Distributed Web-Based Application
    Date:  August 4, 2022
*/
```

Special Note: Due to end of semester time constraints this will be a hard deadline.

Input Specification: The suppliers/part/jobs/shipments database (named `project3`) that is created/populated by the script `project3dbscript.sql`, is the back-end to this application. All other input comes from the front-end user submitted to the application server based servlet entered as either queries or updates to this database. There are three sets of commands that you are to execute against this database included in the `project3rootcommands.sql`, `project3clientcommands.sql`, and `project3dataentrycommands.sql` available on WebCourses under Project 3. As with Project 2, your client-level user will have only `select` privileges on the `project3` database. The data entry-level user will have only `select, insert` and `update` privileges on the `project3` database. Also, as with Project 2, your front-end cannot execute the entire script at one time. You'll need to execute the commands in this script one at a time in your application (copy and paste!). You can run the scripts in the MySQL Workbench if you'd like to compare/see the result sets for each user command.

Output Specification: All output is generated by the servlets and should appear in the user's browser as a text/html page presented to the user. All MySQL-side errors should be caught and reported to the user via the interface. **IMPORTANT:** Be sure to re-run the `project3dbscript.sql` database creation/population script before you begin creating your screen shots for submission. By doing so you will ensure that the database is in its initial state so that all update operations will produce the values we are expecting to see in your result outputs. Then, as with Project 2, run all commands in sequence from the `project3rootcommands.sql` script file (total of 20 different commands), followed immediately by all commands in sequence from the `project3clientcommands.sql` script file (total of 4 different commands), followed immediately by all commands in sequence from the `project3dataentrycommands.sql` script file.

Deliverables:

(1) You should submit your entire `Project3` webapp folder from Tomcat for this program. If you submit the entire folder, then all of the files necessary to execute your web application will be included with the directory structure intact. Submit this via WebCourses no later than **11:59pm Sunday August xx, 2022**.

(2) The following 20 screen shots from the `project3rootcommands.sql` script file must be submitted as part of the deliverables for this project. (You can include the screenshots in the top-level of your webapps folder if you'd like, just be sure to include a note that you've done so.)

a. Command 1
b. Command 2A
c. Command 2B
d. Command 2C
e. Command 3A
f. Command 3B
g. Command 3C
h. Command 3D
i. Command 3E
j. Command 4
k. Command 5A
l. Command 5B
m. Command 5C
n. Command 5D
o. Command 5E

p. Command 6
q. Command 7
r. Command 8
s. Command 9
t. Command 10

(3) The following 4 screenshots from the `project3clientcommands.sql` script file must be submitted as part of the deliverables for this project. (You can include the screenshots in the top-level of your webapps folder if you'd like, just be sure to include a note that you've done so.)
   a. Command 1
   b. Command 2
   c. Command 3
   d. Command 4

(4) The following 4 screenshots from the `project3dataentrycommands.sql` script file must be submitted as part of the deliverables for this project. (You can include the screenshots in the top-level of your webapps folder if you'd like, just be sure to include a note that you've done so.)
   a. Command 1
   b. Command 2
   c. Command 3
   d. Command 4

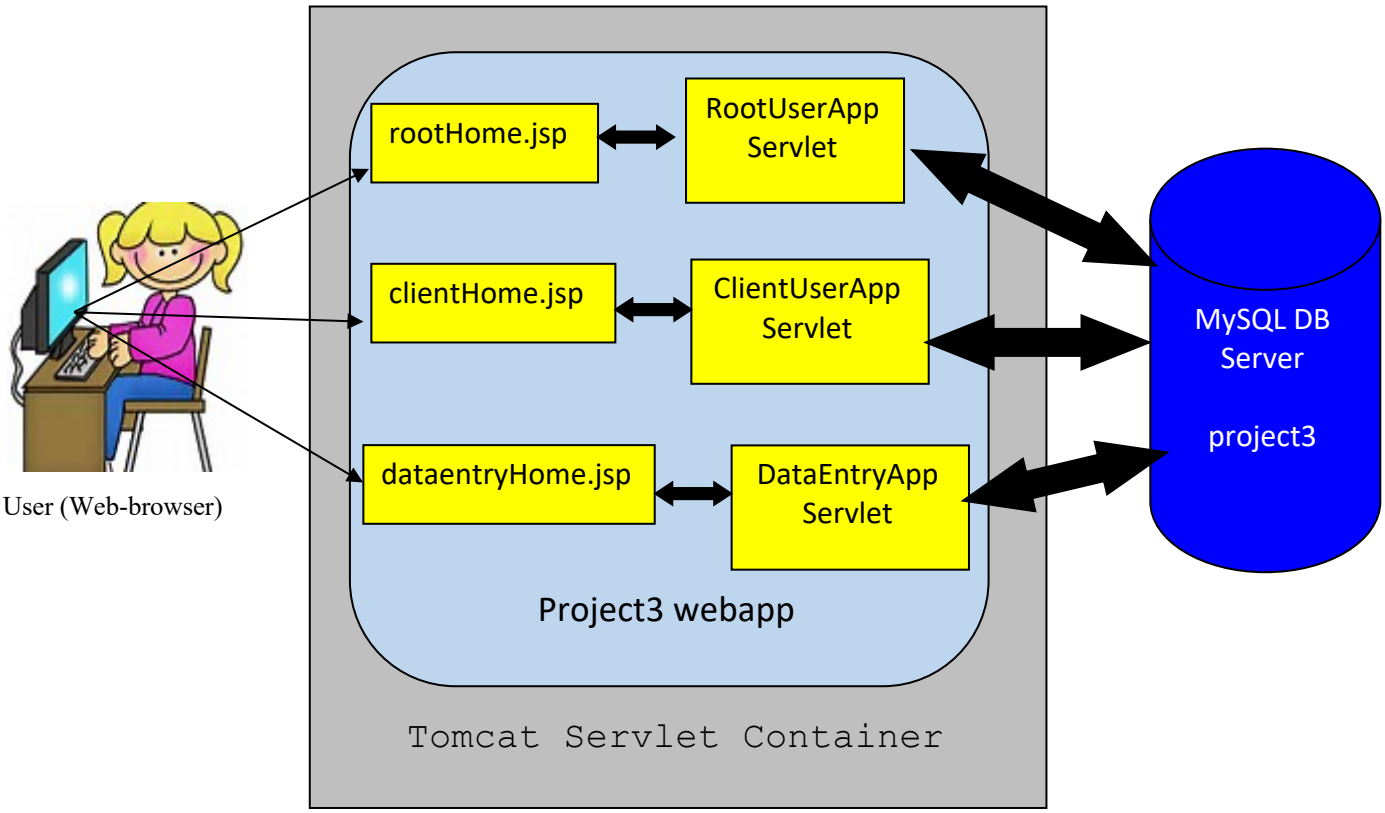
(5) One final screenshot, taken from the MySQL Workbench using a root-user connection, executing the command `select * from suppliers;` This will show the final status of all suppliers after the execution of all commands in the three user-level scripts.

Additional Information:
Be very careful when setting up the directory structures required for the web applications running under your server (Tomcat 10.0.22 or later). See the course notes on servlets for the exact directory structure that must be developed. Be sure that your development IDE and the JVM running under Tomcat are of the same vintage.

Attend/watch Q&A sessions for more information and project details. Additional videos for select parts of this project will also be made available.

Important: Please name your webapp: **Project3.** Let the TAs know if you are doing the bonus problem by attaching a note to your WebCourses submission.

Schematic Overview of Project Components:



Suggested project development approach

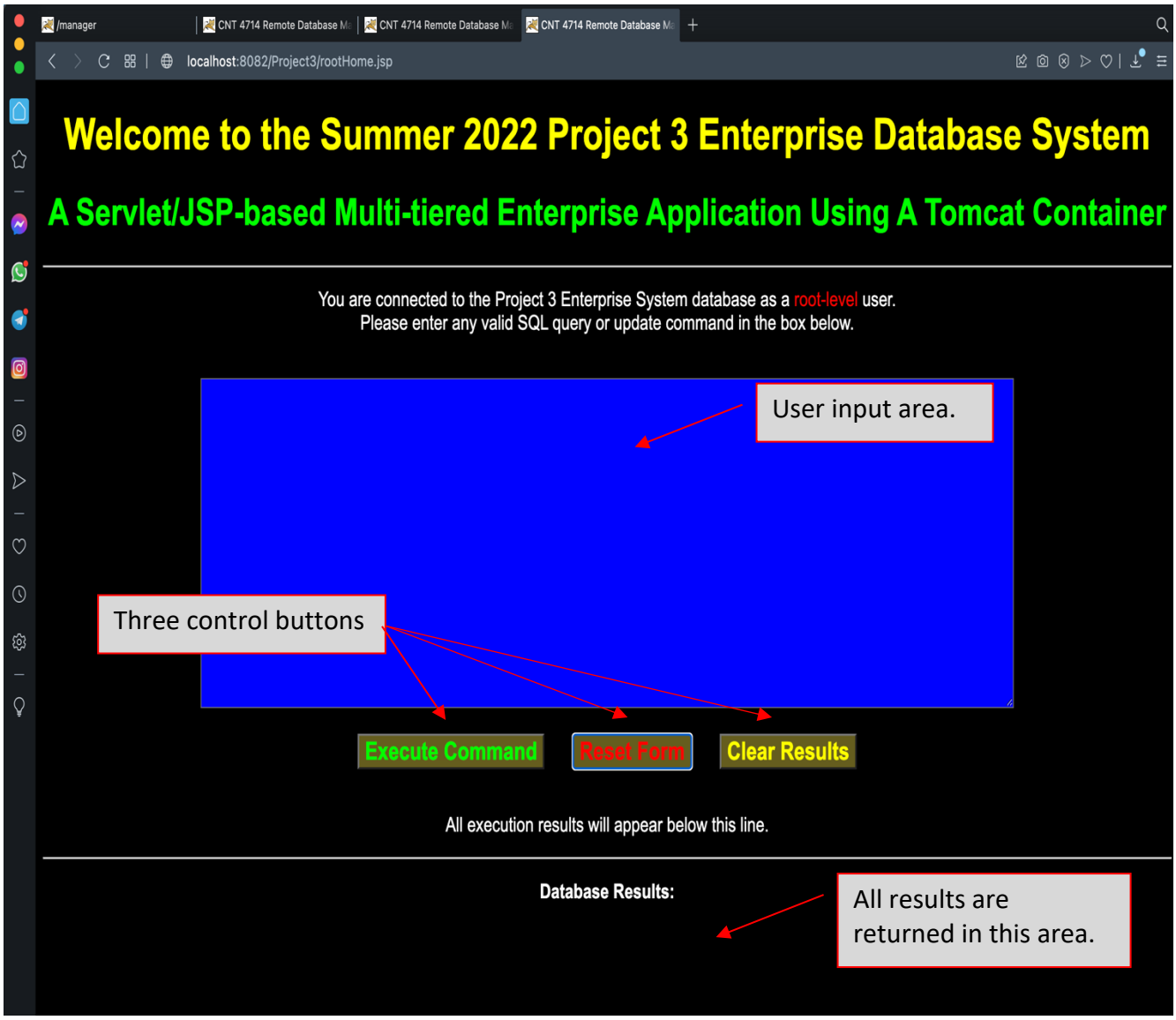
1. Install Tomcat and run some of the examples from the notes to ensure that Tomcat is installed and configured properly before beginning any steps on the project itself.
2. Develop front-end `.html` files `rootHome.html, clientHome.html` and `dataentryHome.html`. These will be later converted to `.jsp` files (see 7 below). This can be done in any editing environment of your choice. Do not specify a specific action for your form submission at this point (use a null string for the action).
3. Construct basic `Project3` webapp framework inside Tomcat `webapp` folder. Deploy files from step 1 above and test/refine in browser of your choice.
4. Construct initial `web.xml` file in `Project3/WEB-INF`. Additional refinement may be needed later.
5. Create properties files for the root-level users, client-level users, and data entry-level users. These will be placed in the `lib` folder of the `Project3` web app (i.e. `Project3/WEB-INF/lib`)..
6. Begin development of the servlets. Basic operation of the root-level servlet and the client-level servlet are the same, with only a small difference (more later). So develop the client-level servlet first and copy and paste with modifications for the root-level servlet later. As we will discuss in the Q&A sessions, the initial servlet (for testing) should do nothing more than simply return "Hi".
7. Load servlet test files into Tomcat `Project3` webapp in correct location and perform initial integration testing of complete package.
8. Further develop all servlet code to complete the basic functionality of the servlets. This includes modification of the front-end interfaces to become `.jsp` files so that all results are

returned to a single page via a targeted location and not require either a complete browser page refresh or the user to employ the browser "back" button. These techniques will be explained later in the JSP notes and also Q&A sessions.

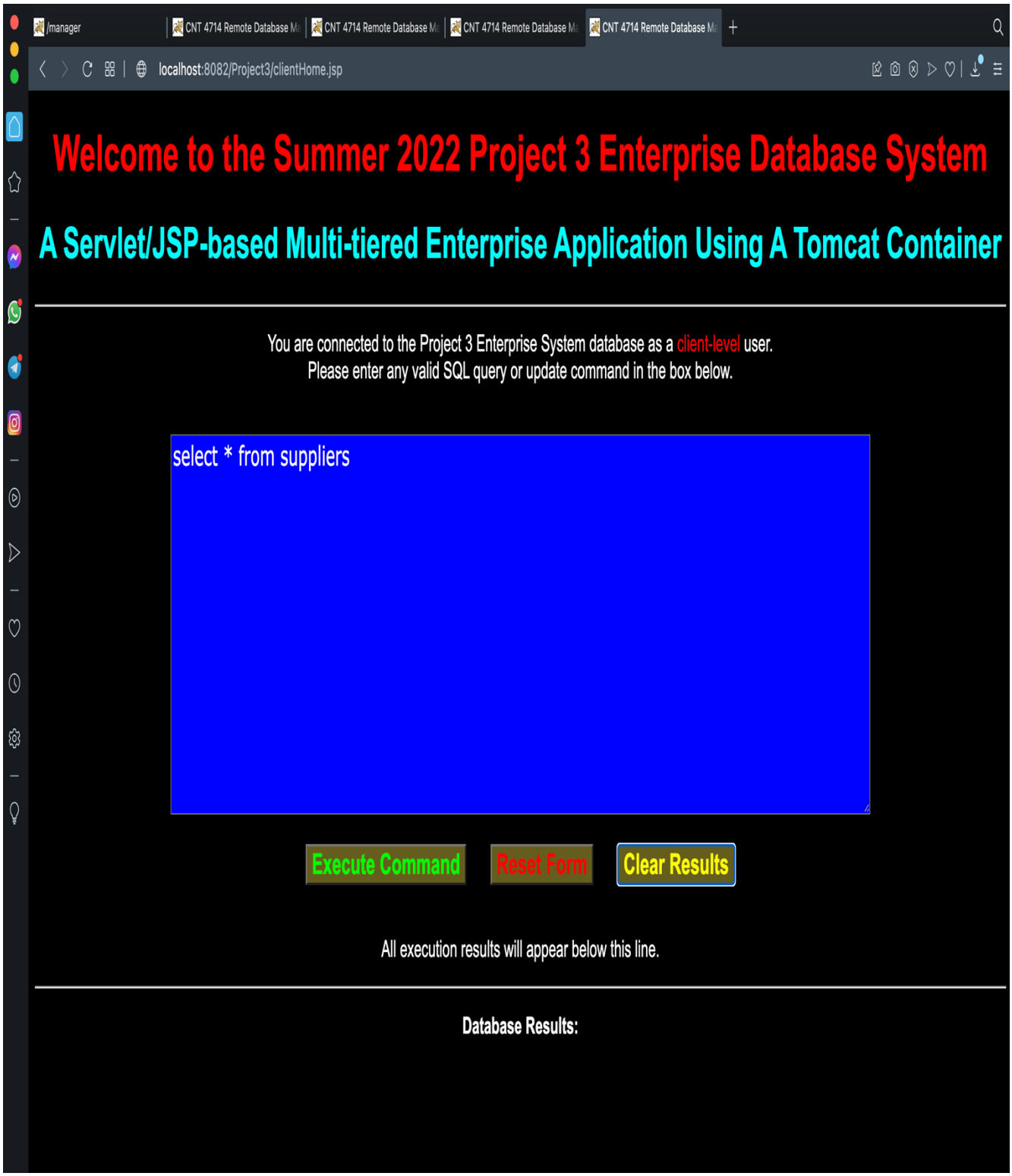
9. Add business logic to the rool-level servlet – develop non-bonus version first.
10. Optional: implement the bonus-version of the business logic.
11. Add business logic to the data entry-level servlet.
12. Recreate the `project3` database.
13. Run through the three user-level command scripts and generate screenshots from your application running these commands.

Some screen shots illustrating the application.

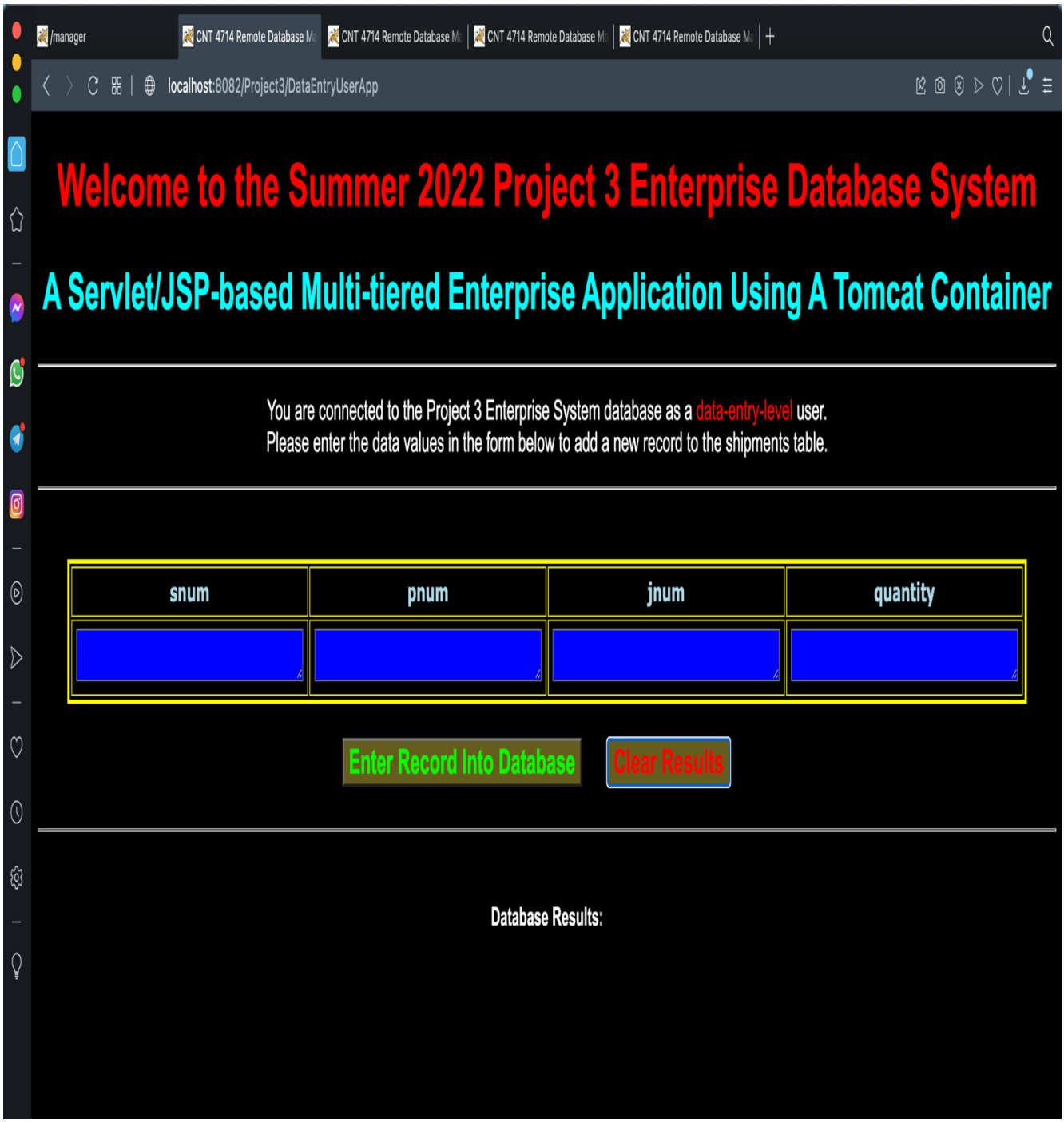
Main root-level user initial JSP page (initial configuration):



Main client-level user initial JSP page (initial configuration):



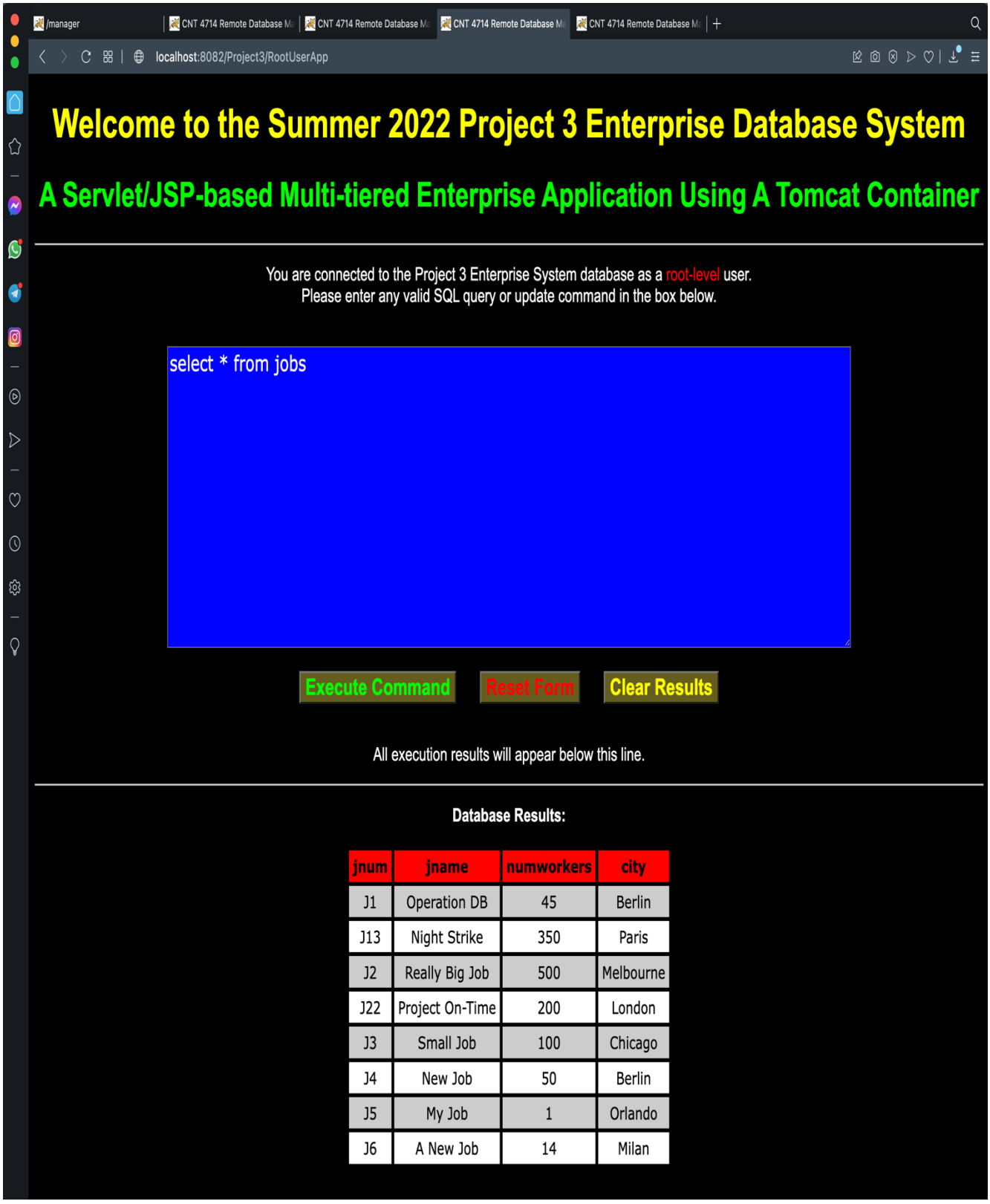
The data entry-level user interface.



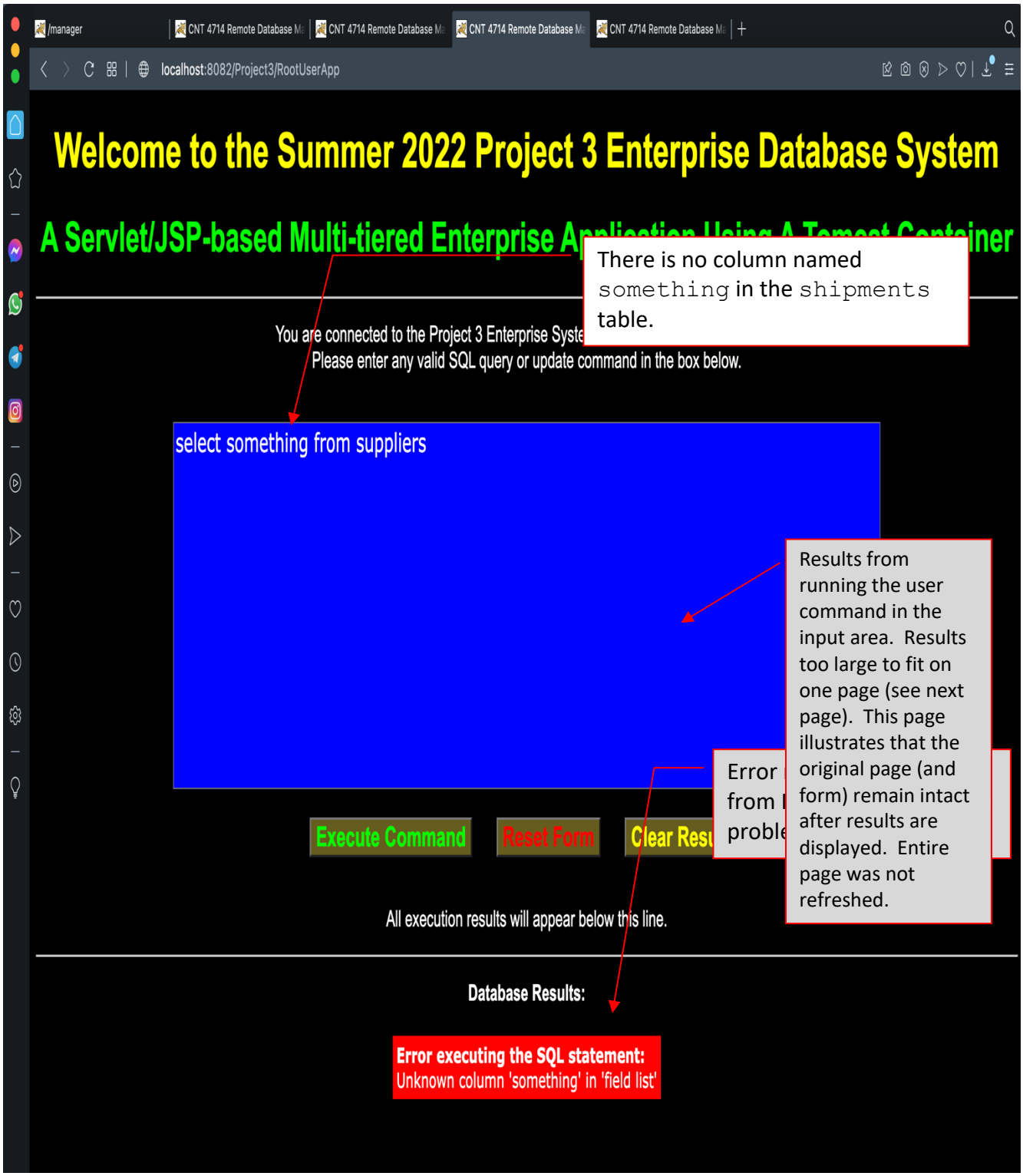
Root-level User Examples

The following several screenshots illustrate operations from the rool-level user interface.

User simply clicks the “Execute Command” button and the SQL command in the form is executed:



User makes a mistake entering an SQL command:



Inserts and updates may cause changes to the supplier status field (business logic is triggered) as shown below:

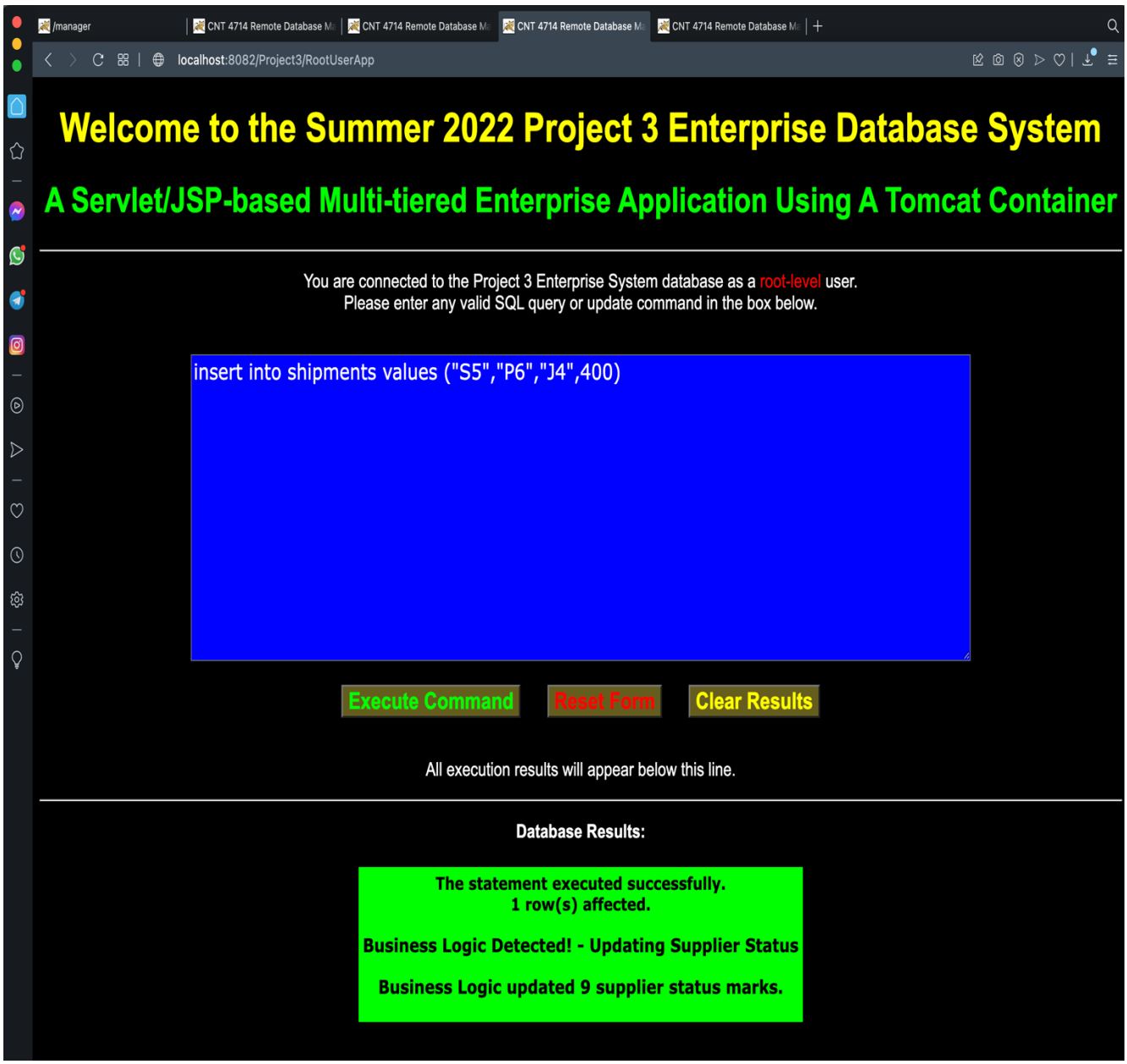
Current state of the suppliers table (i.e., select * from suppliers) (only partial screen shown to allow viewing entire results table):
Note the current status of supplier number S5. (Also note status of S1, S12, S17, S21, S22, S3, S44, and S6.)

Database Results:

snum	sname	status	city
S1	Michael Schumacher	1	Berlin
S10	David Coulthard	2	London
S11	Bernard Hinault	7	Paris
S12	Eddy Merckx	1	Brussels
S13	Candice Swanepoel	3	Cape Town
S14	Adriana Lima	4	Sao Paulo
S15	Jennifer Lawrence	6	Owensboro
S16	Fernando Alonso	4	Madrid
S17	Rubens Barichello	3	Sao Paulo
S18	Tom Boonen	2	Brussels
S19	Johan Messeuw	1	Eekloo
S2	Juan Pablo Montoya	4	Interlagos
S20	Danilo Rossi	2	Milan
S21	Lizzie Armistead	1	Hempstead
S22	Jan Ullrich	5	Bonn
S3	Dietrich Thurau	1	Berlin
S32	Bernd Schnieder	2	Berlin
S33	Rolf Aldag	3	Berlin
S4	Mark Webber	5	Melbourne
S44	Beryl Burton	4	London
S5	Jenson Button	4	London
S56	Marianne Vos	8	Zandvoort
S6	Nicola Gianniberti	2	Milan
S7	Christian Albers	3	Orlando
S8	Giancarlo Fisichella	3	Milan
S9	Kimi Rikonnen	2	Helsinki

Results from running the query "select * from suppliers" – to be used to illustrate an update operation explained on pages 12-14. Notice that the supplier S5's status is currently 4.

User issues the following insert command:



Alert message when an update to the quantity field in the shipments table has caused an update of a supplier's status in the supplier table. Note that the application will use this alert message any time the business logic is tested even if it did not trigger any updates. This means that this message would appear with different values even if no rows are updated (more examples below).

After executing update command (the previous insert), the user re-runs select * from suppliers. Note that S5's status has been increased by 5, but so too has S1, S12, S17, S21, S22, S3, S44, and S6. Allowing the previous insert command to affect only supplier S5's status is handled by the bonus version of this project (see below).

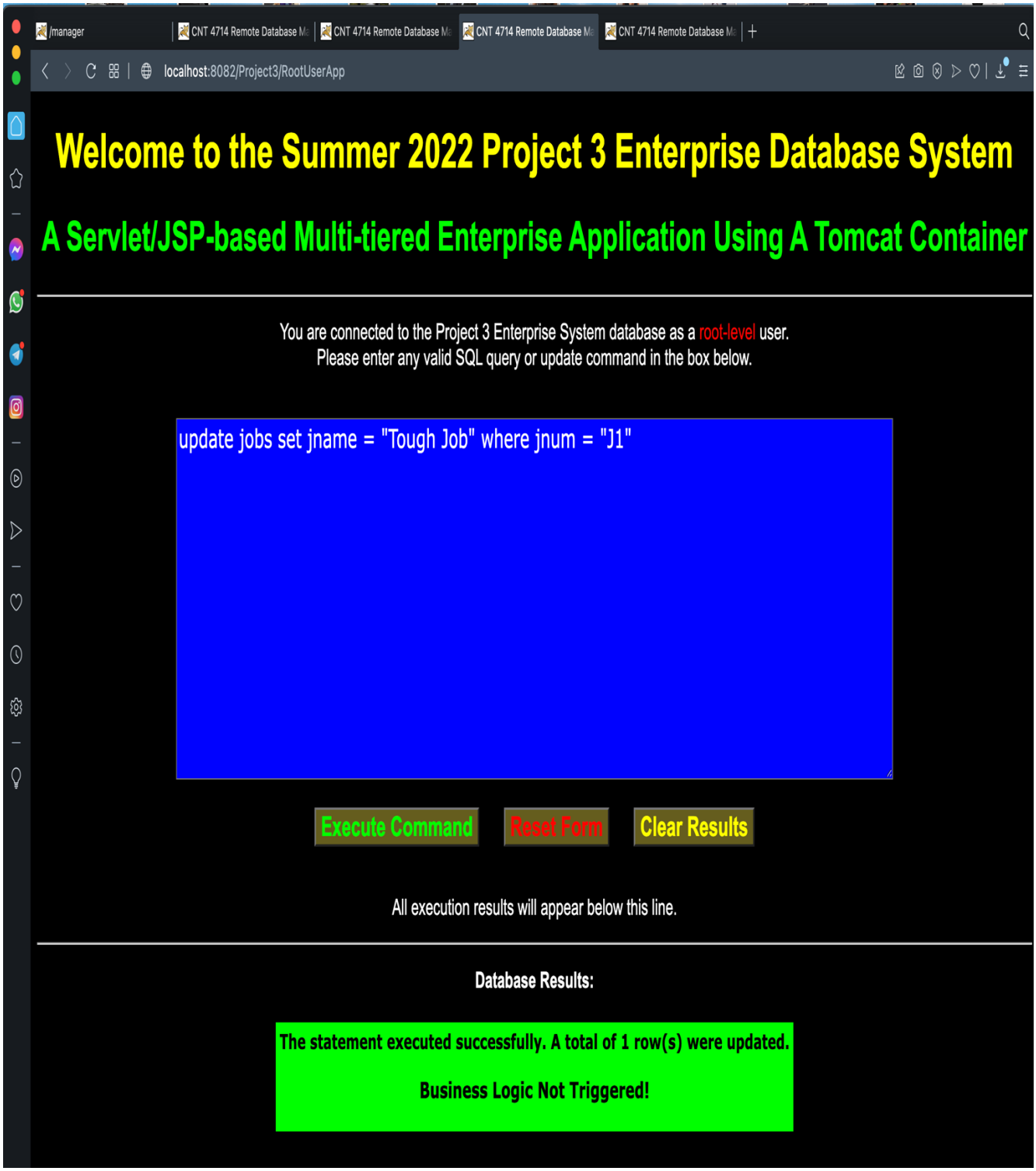
Database Results:

snum	sname	status	city
S1	Michael Schumacher	6	Berlin
S10	David Coulthard	2	London
S11	Bernard Hinault	7	Paris
S12	Eddy Merckx	6	Brussels
S13	Candice Swanepoel	3	Cape Town
S14	Adriana Lima	4	Sao Paulo
S15	Jennifer Lawrence	6	Owensboro
S16	Fernando Alonso	4	Madrid
S17	Rubens Barichello	8	Sao Paulo
S18	Tom Boonen	2	Brussels
S19	Johan Messeuw	1	Eekloo
S2	Juan Pablo Montoya	4	Interlagos
S20	Danilo Rossi	2	Milan
S21	Lizzie Armistead	6	Hempstead
S22	Jan Ullrich	10	Bonn
S3	Dietrich Thurau	6	Berlin
S32	Bernd Schnieder	2	Berlin
S33	Rolf Aldag	3	Berlin
S4	Mark Webber	5	Melbourne
S44	Beryl Burton	9	London
S5	Jenson Button	9	London
S56	Marianne Vos	8	Zandvoort
S6	Nicola Gianniberti	7	Milan
S7	Christian Albers	3	Orlando
S8	Giancarlo Fisichella	3	Milan
S9	Kimi Rikonnen	2	Helsinki

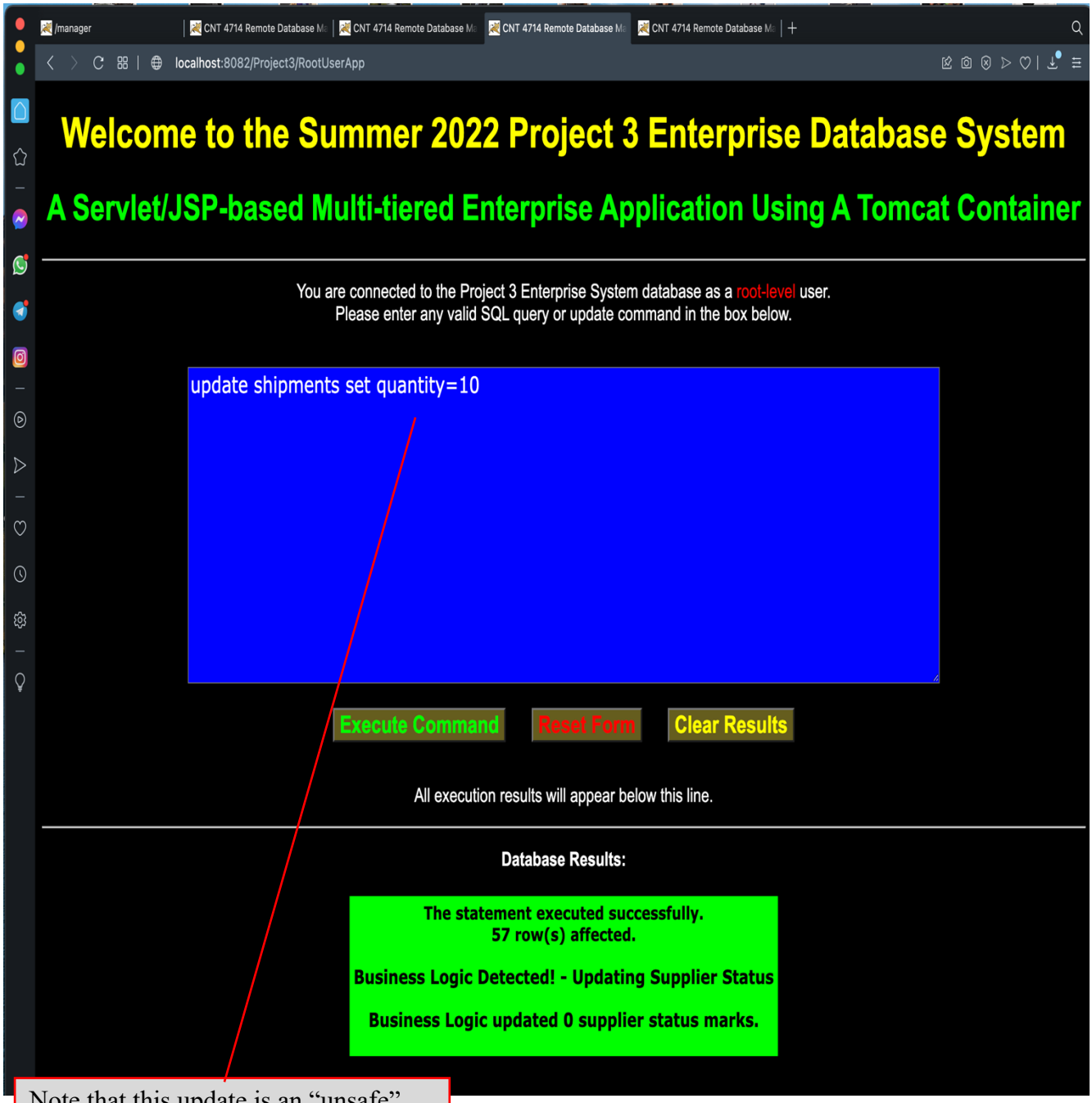
Notice on page 11 (in the original suppliers table) that supplier S5 had a status of 4. After this update, the business logic has increased supplier S5's status by 5, so it is now 9.

Notice too, that suppliers S1, S12, S17, S21, S22, S3, S44, and S6) also had their status increased by 5, since they already recorded with a shipment in which the quantity was >= 100 when the insert command was issued, even though the issued command did not affect them directly.. See bonus problem below for a "fix".

Example of an update command that does not trigger the business logic.



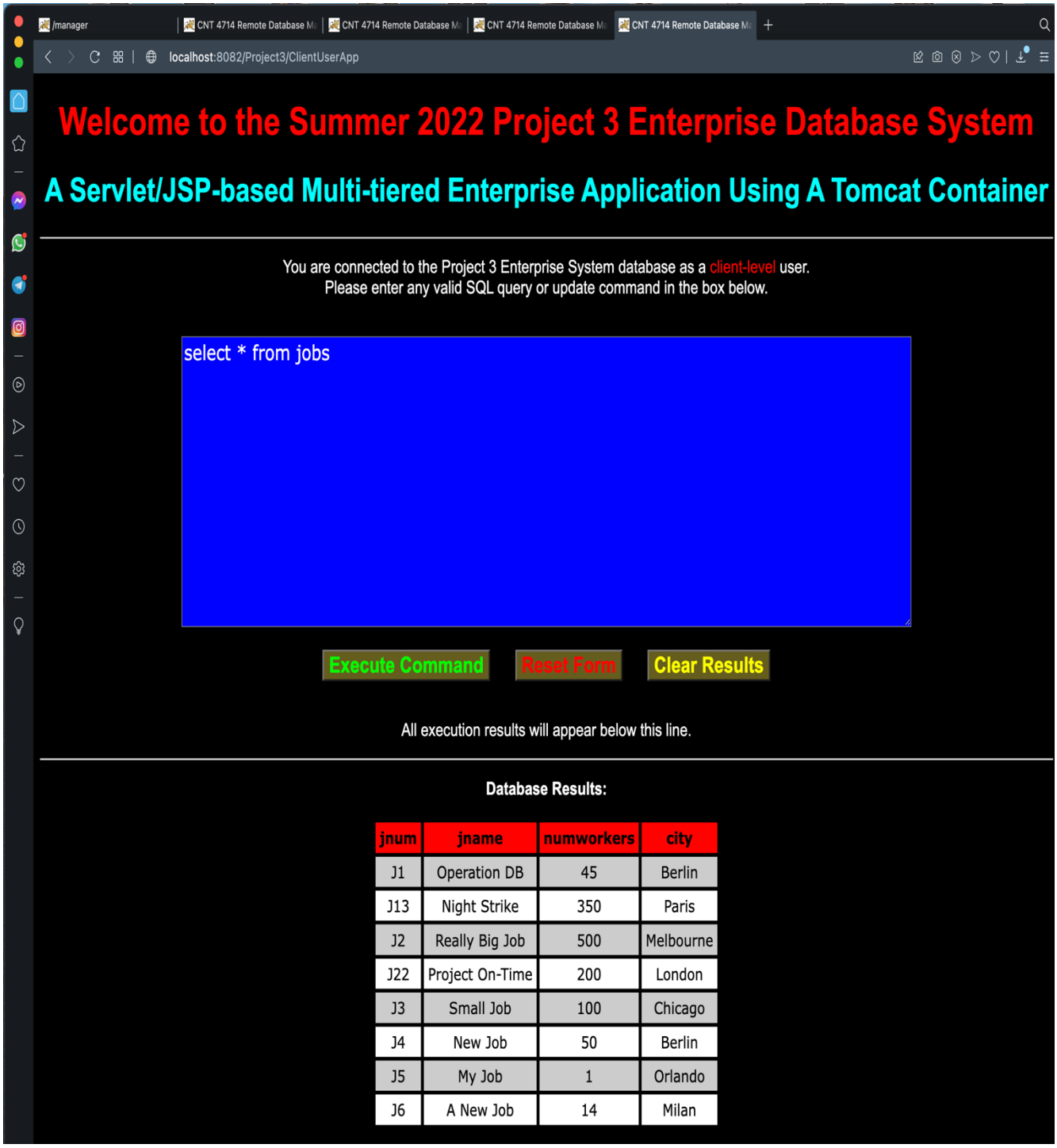
Example of an update command that triggers the business logic but results in no changes to any supplier's status.



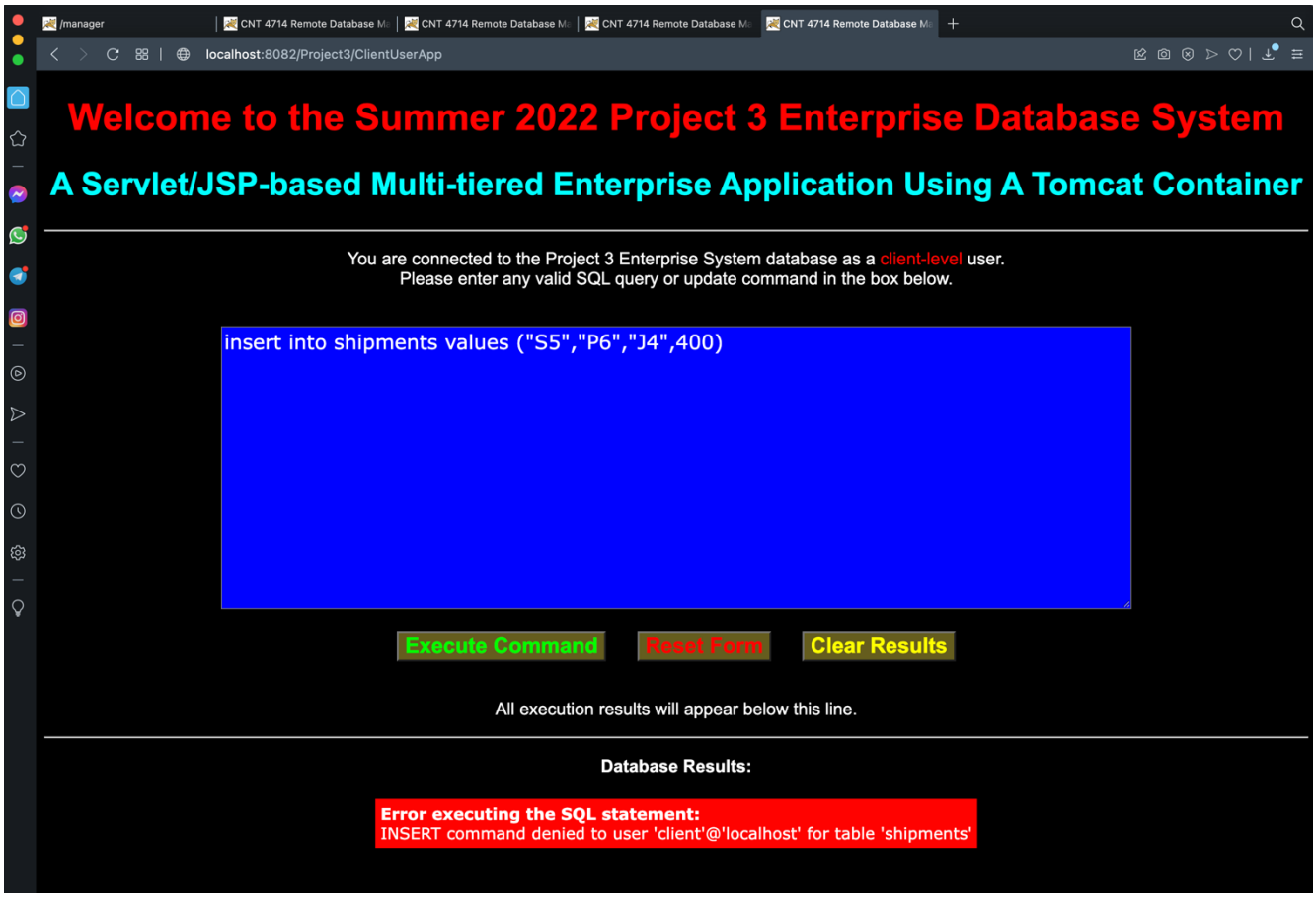
Note that this update is an "unsafe" update since there is no limiting clause and every row in the table will be updated. In this case all 57 rows in the table will now have a quantity of 10.

Client-level User Examples

A client-level user issues a command for which they have privileges (note that this is the same result as the one shown for a root-level user on page 9).



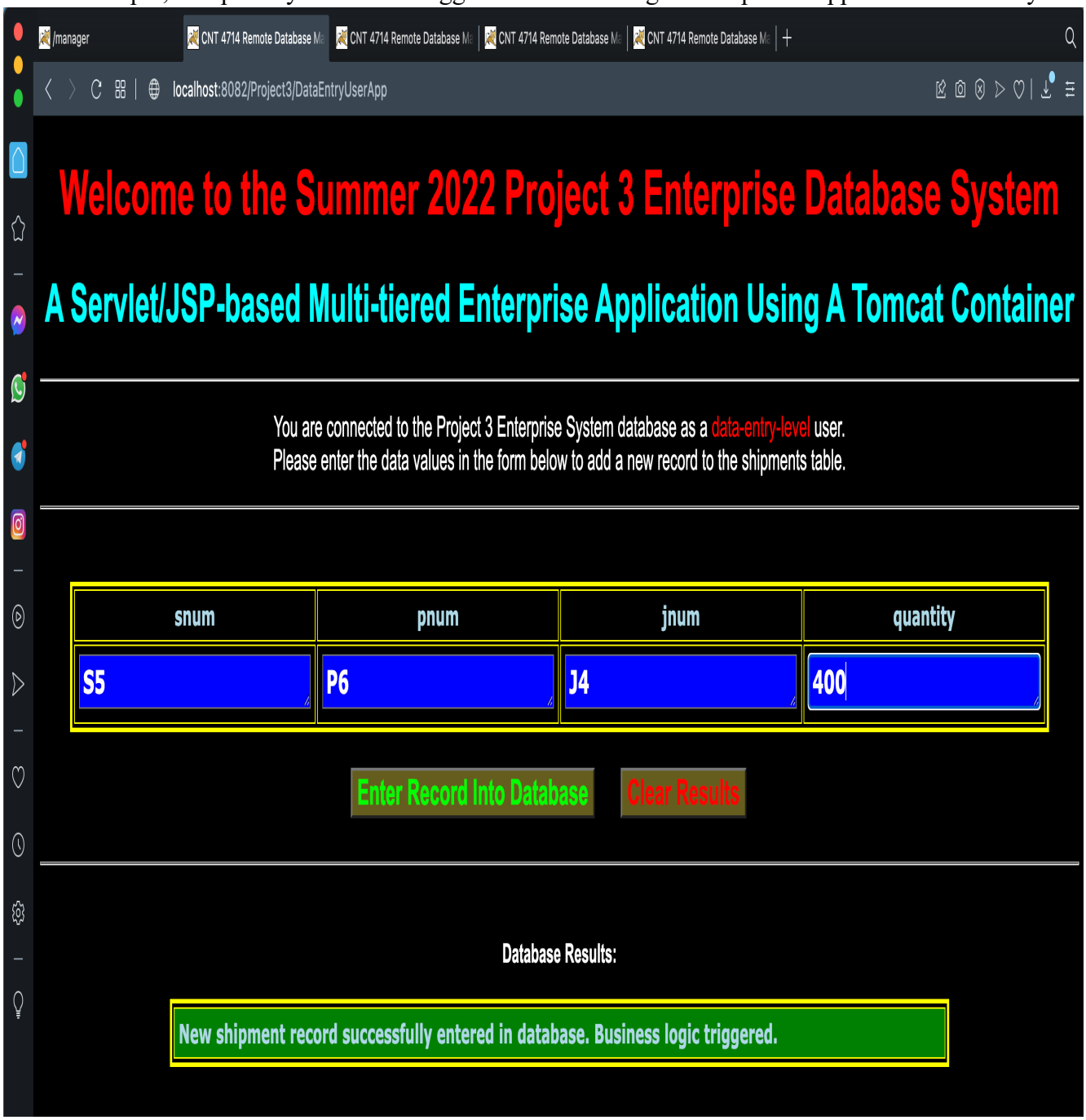
A client-level user issues a command for which they do not have privilege to execute.



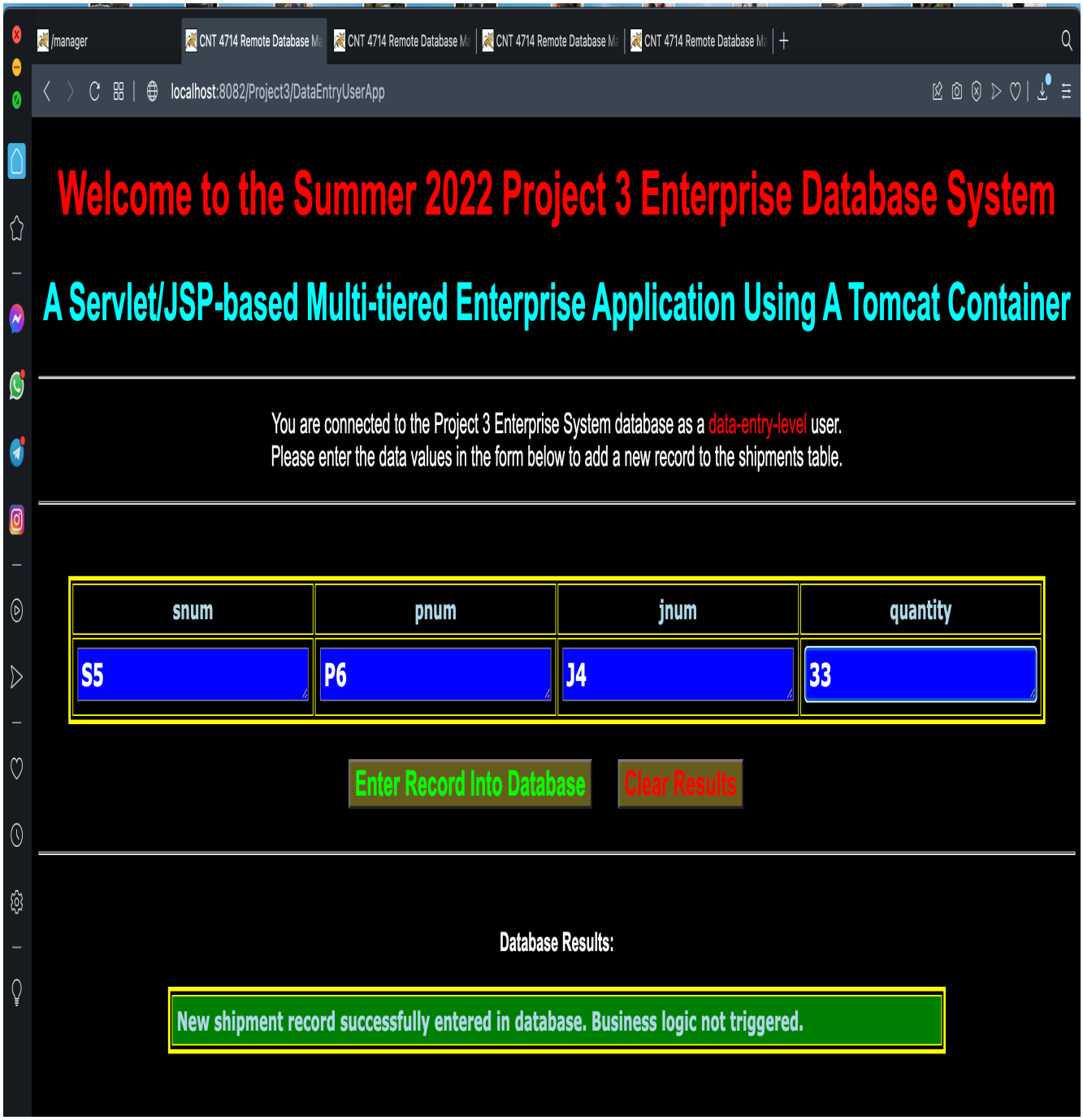
Data Entry-level User Examples

The data entry-level user interface provides only a template (form) for the user to enter data values for new shipment records. Note: the database does not specify any default values for any attributes in the shipments table. Therefore, the user must supply all four values to be used by the PreparedStatement interface.

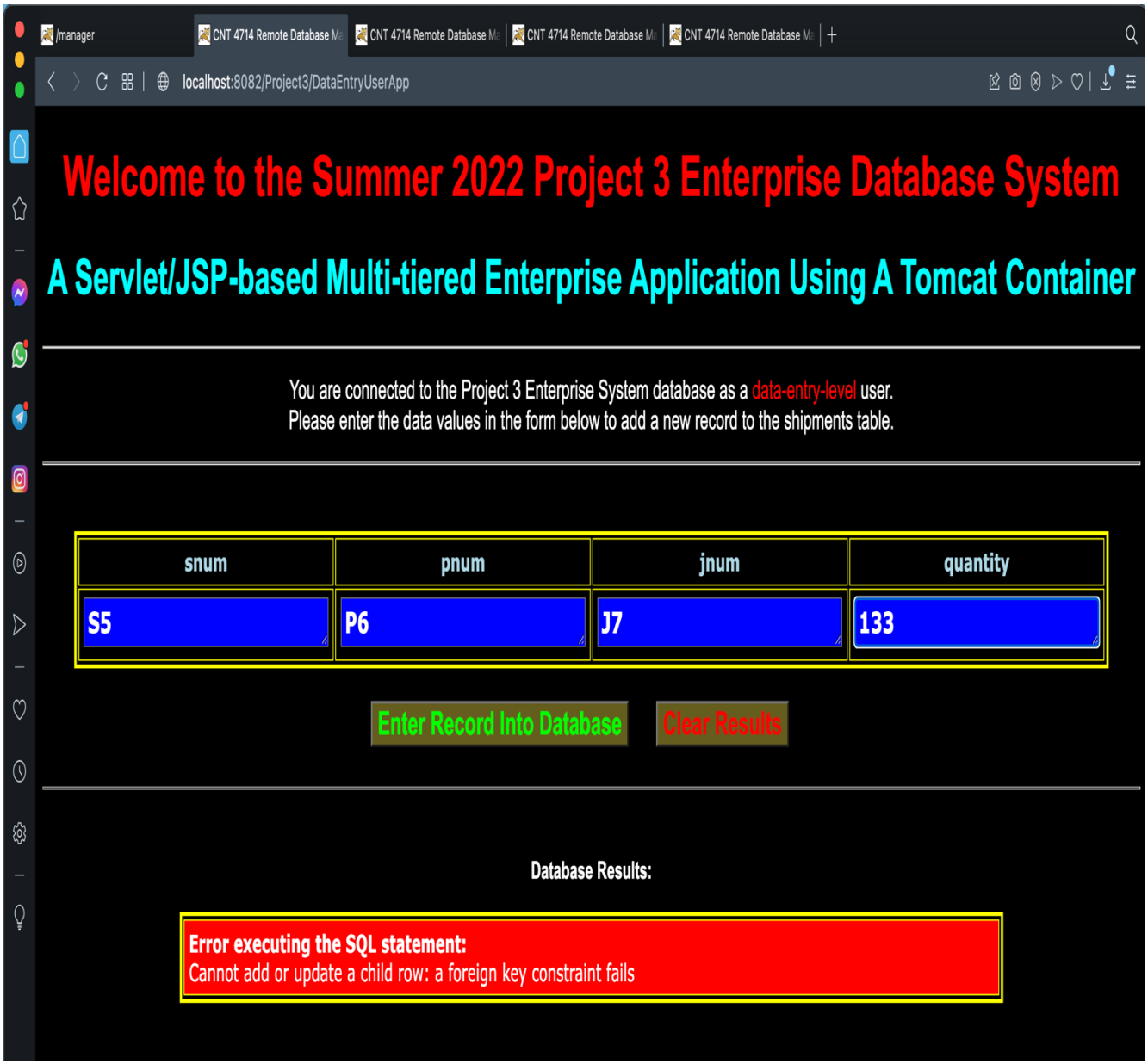
In this example, the quantity value will trigger the business logic and update supplier S5's status by 5.



In this example, the business logic is not triggered as the quantity value is less than 100.



In this example, the attempt to insert the record violates referential integrity and is not allowed.



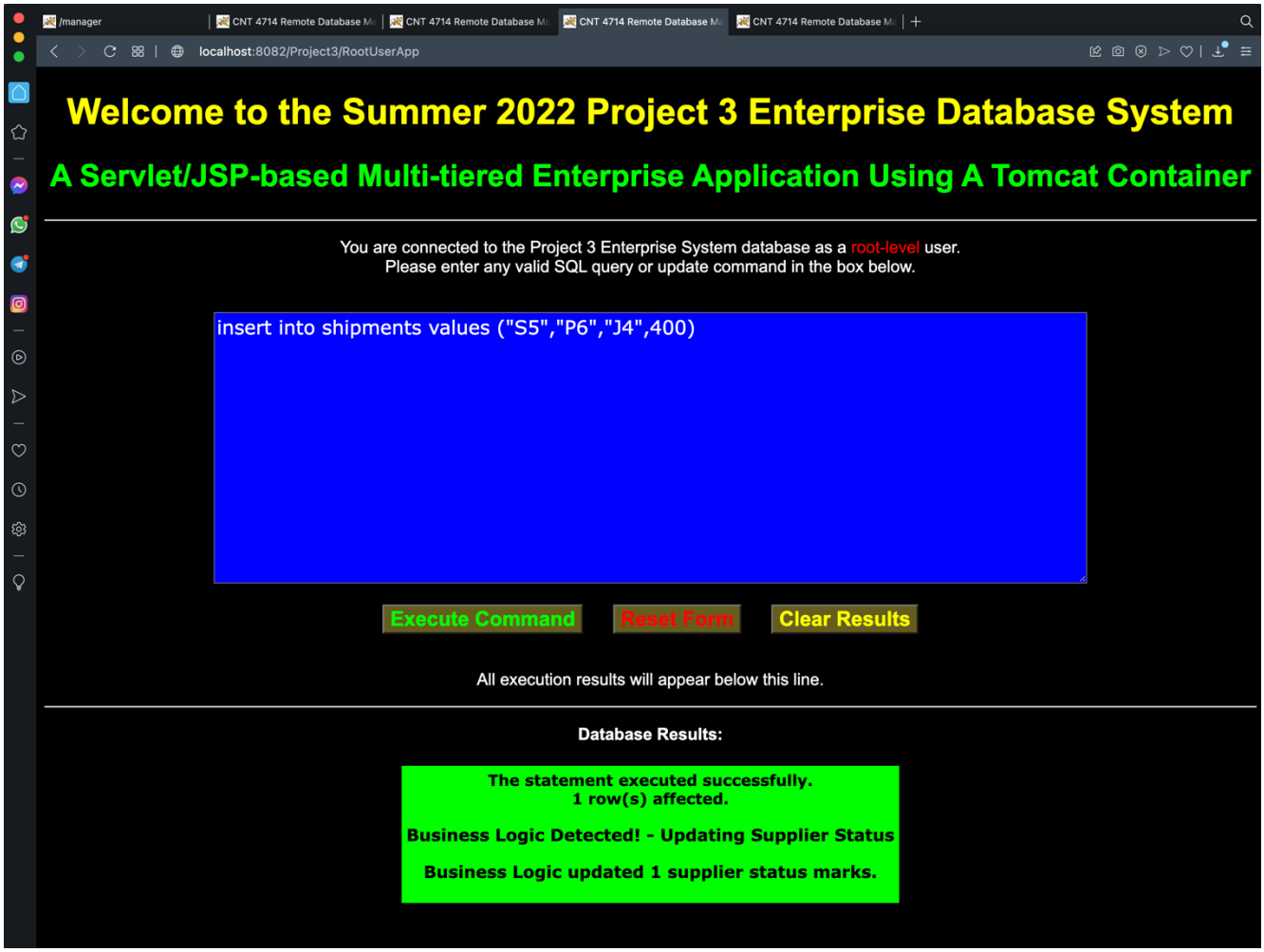
BONUS PROBLEM: 15 points

Instead of allowing any update/insert of a quantity >= 100 to affect any supplier with a shipment involving a quantity >= 100, adjust the business logic portion of your application so that an insert/update of a quantity greater than 100, causes a change to the status of only those suppliers directly affected by the update. For example, using the case shown above, when inserting the row (S5, P6, J4, 400) into the shipments table, only the status of supplier S5 should be increased by 5 (see screen shot below). However, an update such as: UPDATE shipments SET quantity = quantity + 50 WHERE pnum = "P3", would increase by 5 the status of every supplier who ships part P3 in a quantity >= 100 after the update has been issued.

NOTE: If you elect to do the bonus problem, submit only this version of your application. Do not also submit the non-bonus problem version. Let the TAs know if you've elected to do the bonus problem or not.

I will provide many hints for the bonus problem during the Q&A sessions for this project.

With the correct business logic (the bonus version) in place, issue the original insert command as above (on page 12), we now get the correct effect for our update command.



Database Results:

snum	sname	status	city
S1	Michael Schumacher	1	Berlin
S10	David Coulthard	2	London
S11	Bernard Hinault	7	Paris
S12	Eddy Merckx	1	Brussels
S13	Candice Swanepoel	3	Cape Town
S14	Adriana Lima	4	Sao Paulo
S15	Jennifer Lawrence	6	Owensboro
S16	Fernando Alonso	4	Madrid
S17	Rubens Barichello	3	Sao Paulo
S18	Tom Boonen	2	Brussels
S19	Johan Messeuw	1	Eekloo
S2	Juan Pablo Montoya	4	Interlagos
S20	Danilo Rossi	2	Milan
S21	Lizzie Armistead	1	Hempstead
S22	Jan Ullrich	5	Bonn
S3	Dietrich Thurau	1	Berlin
S32	Bernd Schnieder	2	Berlin
S33	Rolf Aldag	3	Berlin
S4	Mark Webber	5	Melbourne
S44	Beryl Burton	4	London
S5	Jenson Button	9	London
S56	Marianne Vos	8	Zandvoort
S6	Nicola Gianniberti	2	Milan
S7	Christian Albers	3	Orlando
S8	Giancarlo Fisichella	3	Milan
S9	Kimi Rikonnen	2	Helsinki

Notice that this time, with the improved business logic that only the supplier directly affected by the insert has had their status updated, all other supplier status values remain unchanged. Compare with table on page 11.

No changes to S1, S12, S17, S21, S22, S3, S44, or S6 this time.

Only supplier S5 had a change of status due to the insertion of the row (S5, P6, J7, 400) as they were the only supplier affected by this update.