CNT 4714 – Project Four – Spring 2024

Title: "Project Four: Developing A Three-Tier Distributed Web-Based Application"
Points: 100 points (bonus problem potentially adds 15 points – see page 30.)
Due Date: **Tuesday April 23, 2024 by 11:59 pm (WebCourses time)**

Objectives: To incorporate many of the techniques you've learned 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, named `project4` (creation/population script available on Webcourses under Project 4), 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 MySQL DBMS 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 key values. 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 consist of an HTML landing page which is used to authenticate end users. The authentication of users is handled via a servlet in the webapp that validates the user entered credentials with those in another database named credentialsDB maintained on the MySQL DB server. (The credentialsDB is created and populated using the `project4UserCredentialsScript.sql`.). This servlet is running as a system-level application with root user privileges (more later). The credentialsDB contains a single table named usercredentials of usernames and their associated passwords. Validation consists of matching both the user entered name and password against the values stored in the usercredentials table. If the user entered credentials do not match an entry (row) in the usercredentials table, the user will be denied access to the system. If a match is found, the authenticated the user will be automatically redirected to one of four different JSP pages. One which handles root-level user clients and one which handles non-root-level clients, that allow the users 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 consist of dedicated data entry forms for entering new records into the four tables in the database. This third page will only be used by special data entry users who do not directly enter SQL commands into the interface, but only enter data into specific tables in the database via parameterized commands. Finally, a fourth JSP page will be reserved for accountant-level users, who, similar to data-entry users, do not enter SQL commands directly, but rather execute remote stored procedures (RPCs) that reside on the database server.

The front-ends of all four user applications 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 forms and issue the SQL command in the background. The data-entry level front end will have two buttons on each form, one for entering the data and one for clearing data and results. Finally, the fourth front-end, for accountant-level users, will consist of a selection of "reports" that can be run based on their selection. The accountant-level user will simply select an option from a list of possible "reports" and effect of this will be to execute a RPC on the database server and return the results of the report to the accountant-level front-end page. The account-level application will use the CallableStatement interface. More details on this interface and remote procedure calls will be covered in detail in the Q&A sessions as well as in additional documentation that supports this project.

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 four different properties files, one for the root-level users, one for the client-level users (both the same as project 3 except for the different database and we'll only use client1 for this project – I renamed my client1 to just client for project 4), one for the data entry-level users, and one for the accountant-level users.. The data-entry level user account is a new user that will need to be created, as you did with the clients and accountant users in project 3. The accountant-level user is the same user as in project 3, but will have new permissions assigned. Once all of your user accounts are created, then run the `permissionScriptProject4.sql` to assign the various users their correct privileges. This will only need to be done one time.

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, J4, 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 28 for a bonus problem that implements a modified version of this business rule.)** The business logic of the second tier will reside in the servlets 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. Many additional details on the business logic will be covered in the Q&A sessions.

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. This is because client-level users do not have updating privileges on the `project4` database and our business logic can only be potentially triggered by updating operations.

The data entry-level servlet will provide the user four templates (forms) for the each of the tables in the `project4` database. The correct updating command will be executed by mid-tier level servlets issuing whichever updating commands are appropriate based on which form was submitted by the data-entry user. The updating commands are executed by extracting the parameters from the form and issuing a prepared statement update to the correct database table. You may want to refer to the JDBC notes from Module 3 and Project 3, to refresh your memory of how the preparedStatement() interface differs from the normal Statement() interface.
The accountant-level servlet will provide the user with a set of options, where each option is a "report" (an underlying SQL query) that will return the report details to the accountant-level front-end page. Each element in the list of reports that can be generated references a stored procedure that resides on the MySQL server under the database `project4`. Executing remote stored procedures requires using the CallableStatement interface. A supplemental document will be available that illustrates how the stored procedures are created and referenced. The MySQL Workbench tool will be very useful in easily creating your stored procedures.

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 – Spring 2024 – Project Four
    Assignment title:  A Three-Tier Distributed Web-Based Application
    Date:  April 23, 2024
*/
```

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

Input Specification: The credentialsDB is created by the `project4UserCredentials.sql script`. The suppliers/part/jobs/shipments database (named project4) that is created/populated by the script `project4dbscript.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 four sets of commands scripts hat you will execute against this database included in the:

1. `project4rootcommands.sql`,
2. `project4clientcommands.sql`,
3. `project4dataentrycommands.sql`, and
4. `project4accountantcommands.txt`

All are available on WebCourses under Project 4. As with Project 3 your client-level user (only have client1 or just client for this project) will have only `select` privileges on the `project4` database. The data entry-level user will have only `select`, `insert` and `update` privileges on the `project4` database. The accountant-level user will have only `execute` privileges on the `project4` database. Also, as with Project 3, your front-end cannot execute an entire script at one time. You'll need to execute the commands in each 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. Your four different user accounts should be named: root, client1, dataentryuser, and theaccountant.

Project initiation sequence (must do these steps in the shown order).

1. Run the `project4dbscript.sql` script to create the project 4 database
2. Create new `dataentryuser` using the MySQL Workbench…see project 3 additional document on creating user accounts in MySQL.
3. Run the `project4UserCredentialsScript.sql` to set the user credentials for the client, dataentryuser, and the accountant on the project4 database.
4. Run the `permissionsScriptProject4.sql` script to set the permissions for the client, dataentryuser, and accountant-level users.
5. Create properties files for all four end-user types.
6. Create properties file for the system-app level user. This is the internal app used to validate user credentials, so this properties files contains root-level properties to the credentailsDB.

Output Specification: All output is generated by the servlets and should appear in the user's browser as text/html output 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 `project4dbscript.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, just as you did with Project 3, run all commands in sequence from the `project4rootcommands.sql` script file (total of 20 different commands), followed immediately by all commands in sequence from the `project4clientcommands.sql` script file (total of 4 different commands), followed immediately by all commands in sequence from the `project4dataentrycommands.sql` script file (total of 8 different commands), followed immediately by all commands in sequence from the `project4accountantcommands.sql` script file (total of 5 different commands).

Deliverables:

(1) You should submit your entire `Project-4` webapp folder (zipped) from Tomcat for this project. 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 Tuesday April 23, 2024**.

(2) **NOTE:** If you keep copies of your .java files in the webapp the way that I do, then your `.java` files will be included in the zip folder from (1) above. If you do not have your `.java` files inside the webapp as I do, then include another folder that contains all the `.java` files for your webapp and place this in the top level of the Project-4 webapp. Let the TAs know the location of your .java files.

(3) The following 20 screen shots from the `project4rootcommands.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

(4) The following 4 screenshots from the `project4clientcommands.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) The following 8 screenshots from the `project4dataentrycommands.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
   e. Command 5
   f. Command 6
   g. Command 7
   h. Command 8

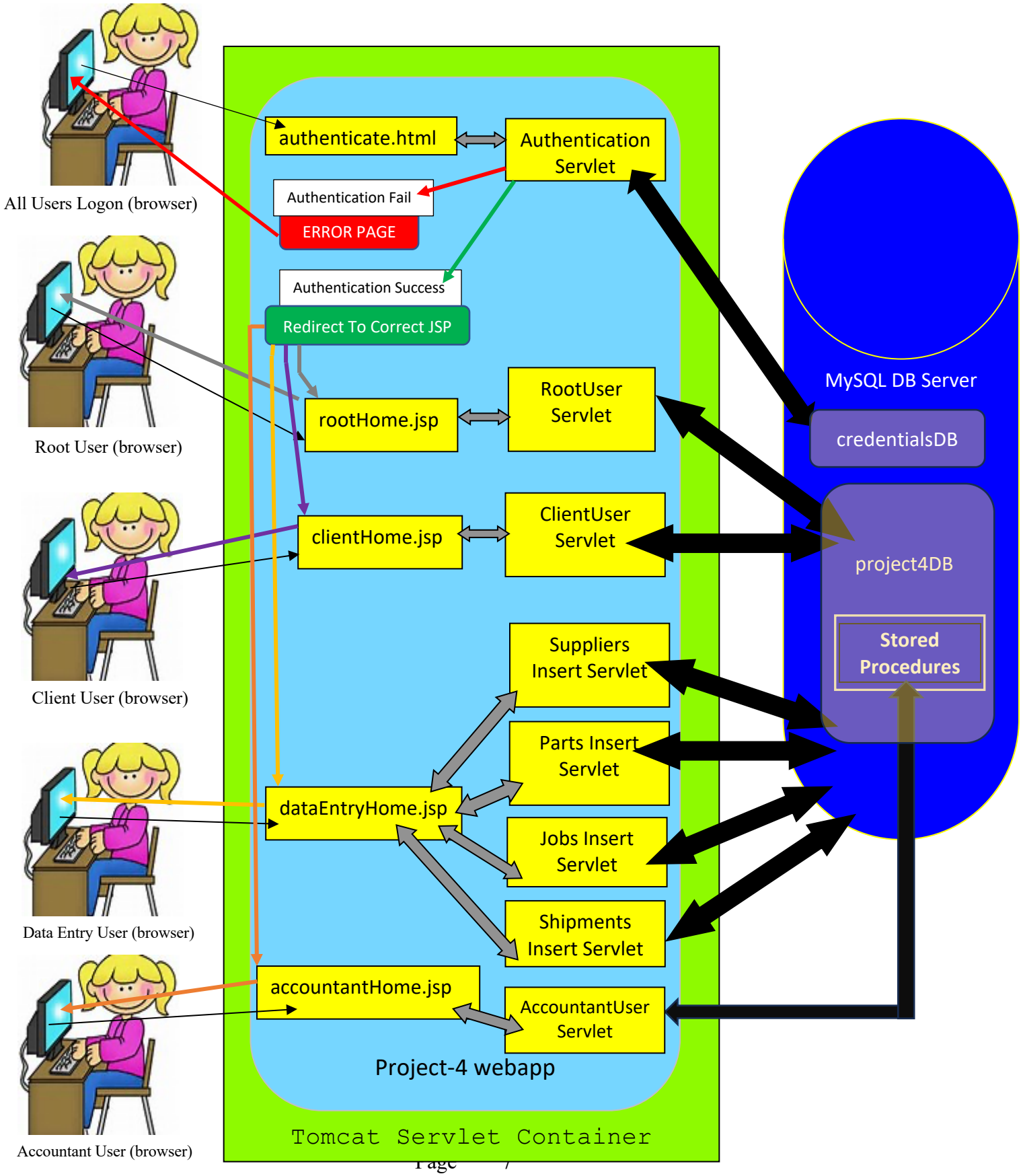
(6) The following 5 screenshots from the `project4accountantcommands.sql` script file must be submitted as part of the deliverables for this project. (You can include the screenshots in the top-level of your webapps folder if you'd like, just be sure to include a note that you've done so).
   a. Command 1
   b. Command 2
   c. Command 3
   d. Command 4
   e. Command 5

Additional Information:
Be very careful when setting up the directory structures required for the web applications running under your server (Tomcat 10.1.18 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 information for select parts of this project will also be made available.

Important: Please name your webapp: **Project-4** Let the TAs know if you are doing the bonus problem by attaching a note to your WebCourses submission. Be sure to let the TAs know where your .java files are located and also where all of the screenshots are located.

Schematic Overview of Project Components:



Suggested project development approach

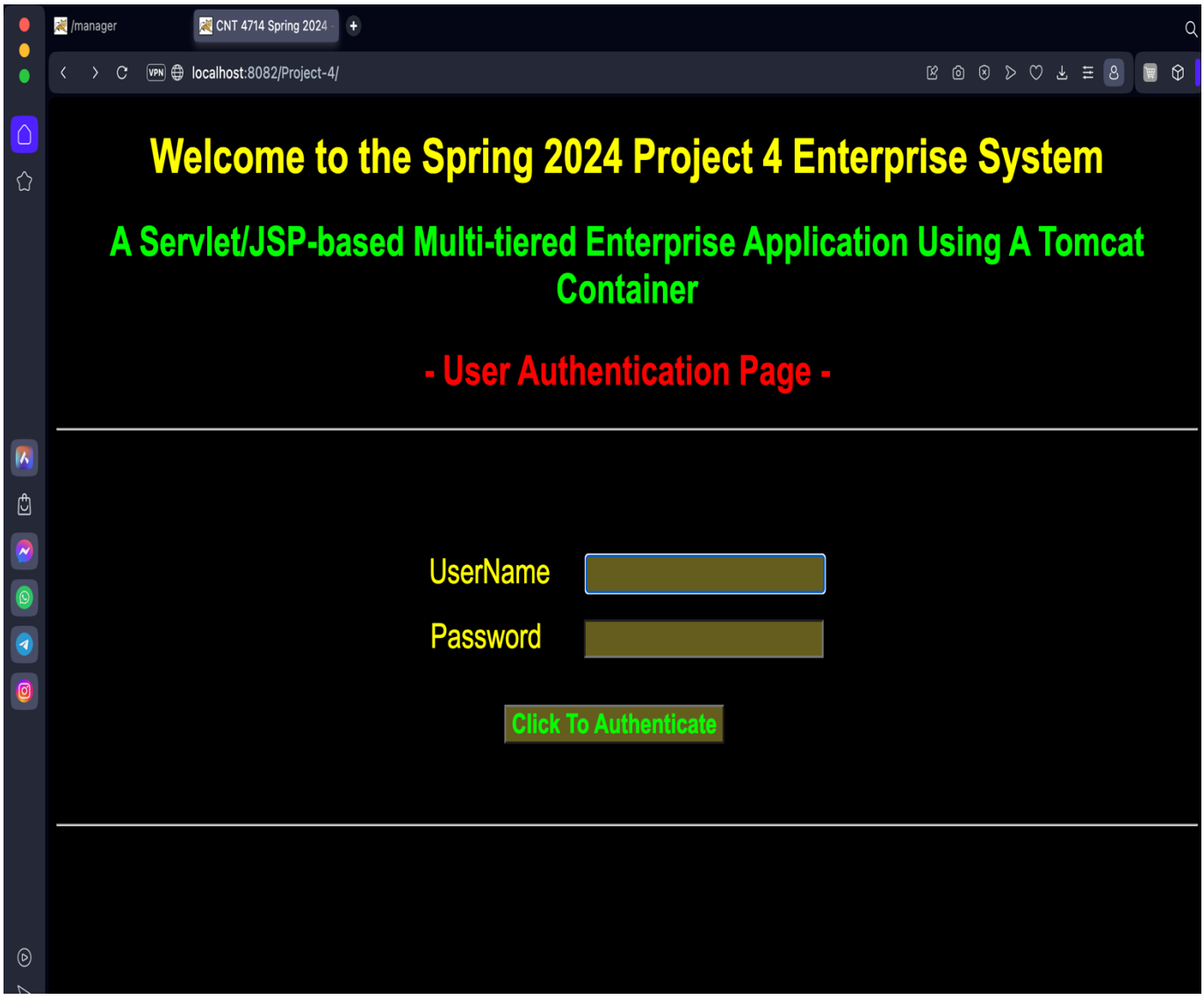
1. Install Tomcat and run some of the examples from the notes to ensure that Tomcat is installed and configured properly before beginning any steps on the project itself. Do not attempt to start the actual project unless you know that (1) Tomcat is properly configured and running, (2) that your IDE Java and your Tomcat JVM are of the same vintage, (3) you have successfully run at least one known working example.
2. **IMPORTANT: Keep back-up copies of all files within your webapp in a location NOT inside Tomcat. Maintain these backup copies religiously.**
3. Under the Input Specification section above (page 4), run all of the database creation/population scripts, then create the necessary user accounts, and finally, run the permission assignment script.
4. Develop the authentication HTML front-end and it's associated error page.
5. Develop front-end `.html` files `rootHome.html, clientHome.html, dataentryHome.html`, and `accountantHome.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). I will demonstrate this in Q&A sessions.
6. Construct basic `Project-4` webapp framework inside Tomcat `webapps` folder.
7. Construct initial `web.xml` file in `Project-4/WEB-INF`. Additional refinement will be needed later.
8. Deploy files from step 1 above and test/refine in browser of your choice.
9. Create properties files for the root-level users, client-level users, data entry-level users, accountant-level users, and a system-level user. These will be placed in the `lib` folder of the `Project-4` web app (i.e. `Project-4/WEB-INF/lib`)..
10. Begin development of the servlets. Basic operation of the root-level servlet and the client-level servlet are the same (more later on the actual differences of these servlets). 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".
11. You will ultimately have servlets supporting the root-users (1), client-users (1), data-entry users (4), and accountant-level users (1). There is also a servlet that support the `authentication.html` page.
12. Load servlet test files into Tomcat `Project-4` webapp in correct location and perform initial integration testing of complete package.
13. **IMPORTANT: Keep back-up copies of all files within your webapp in a location NOT inside Tomcat. Maintain these backup copies religiously.**
14. 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.
15. Add business logic to the rool-level servlet – develop non-bonus version first.
16. Optional: implement the bonus-version of the business logic.
17. Add business logic to the data entry-level servlet. This will apply only to the servlet handling inserts to the shipments table.
18. Recreate the `project4` database.

19. Run through the four user-level command scripts and generate screenshots from your application running these commands.
20. You are done! Congratulations! You have developed a three-tier distributed wed-based application running in a Tomcat container utilizing a MySQL backend database server.

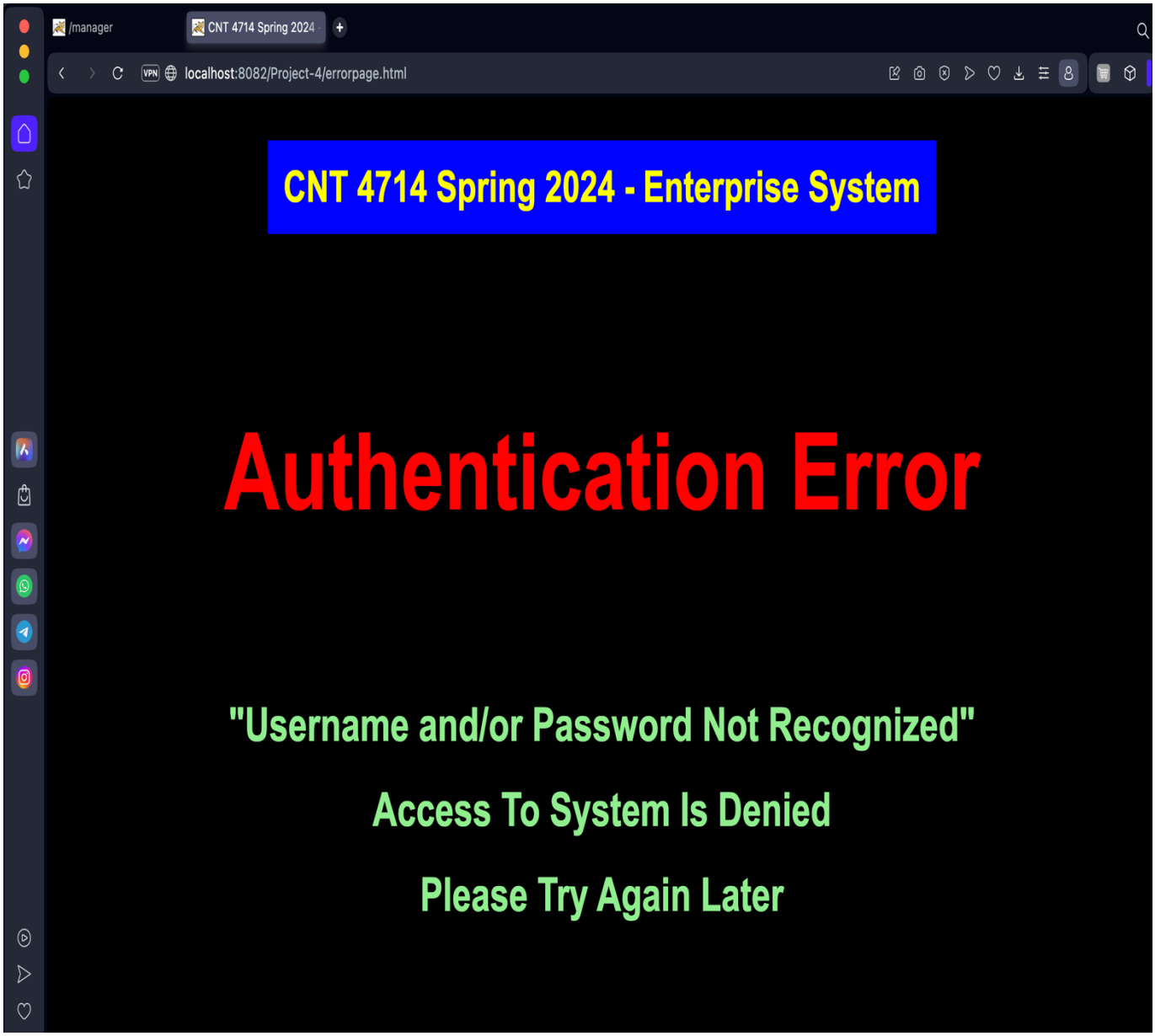
Some screen shots illustrating the application.

Authentication Page – The Main Initial Landing Page

To run this webapp all users begin on this landing page (this is authentication.html).

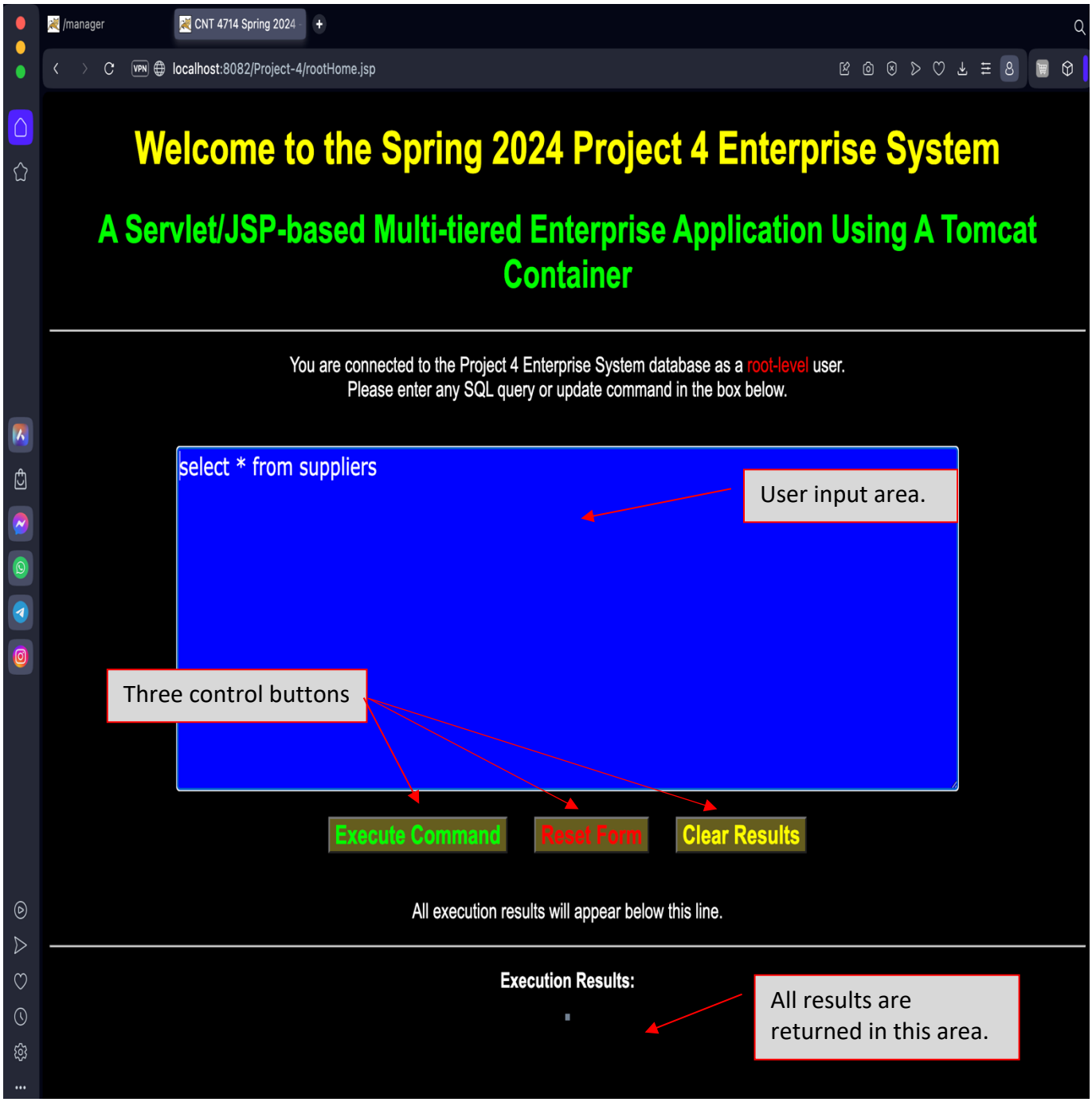


If the user enters incorrect login credentials that do not match any found in the `credentials.csv` file. The errorpage.html is displayed.



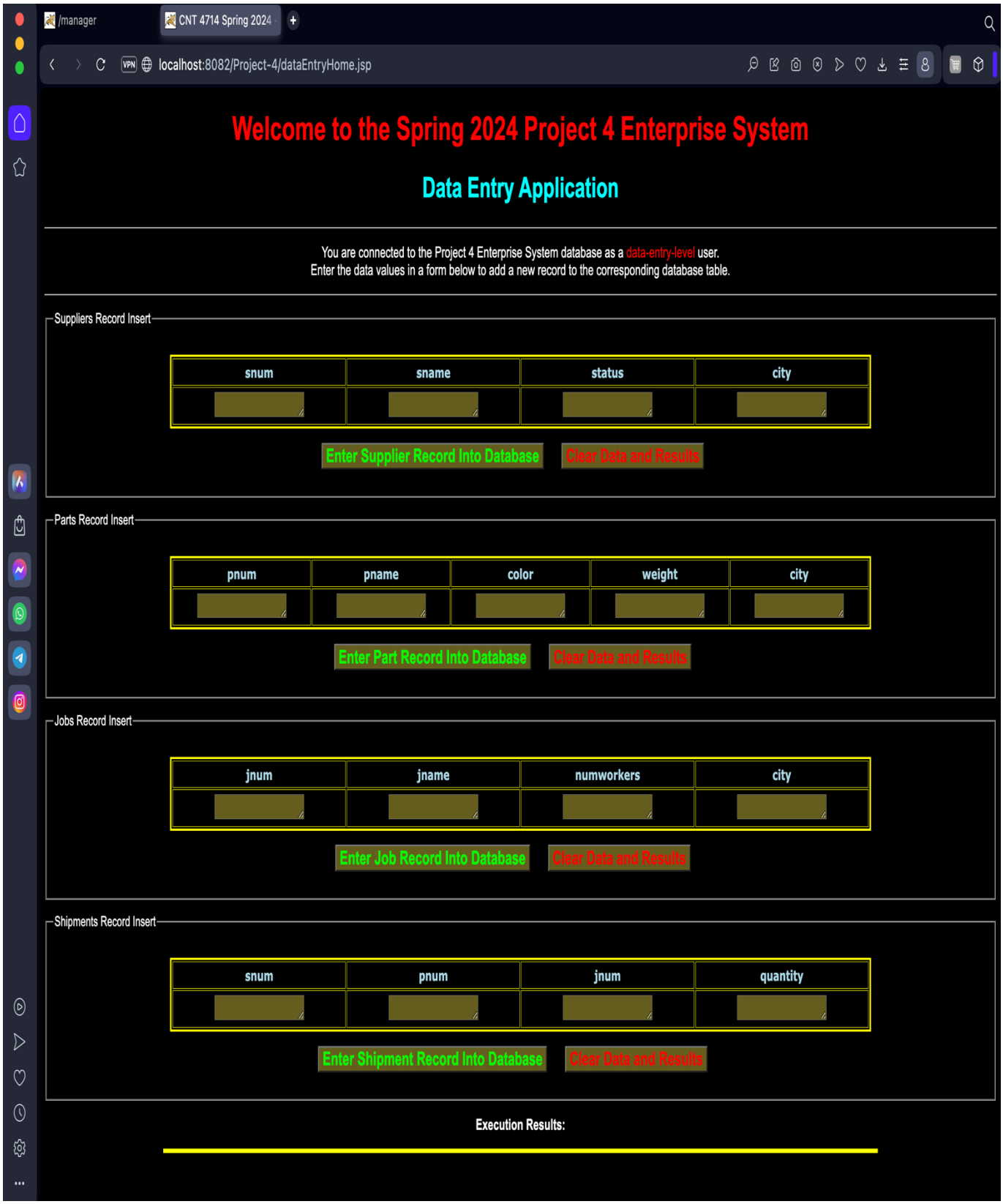
If the user's login credentials are validated (they match with an entry in the usercredentials table in the credentialsDB), the user will be automatically redirected to a landing page which corresponds to the webapp which they are authorized to access. These landing pages, for each category of user are shown below.

Root-level user interface (a JSP page - initial configuration shown):



Client-level user interface (a JSP page - initial configuration shown):
(Note that exept for the color scheme in the headers and the fact that a client-level user is specified, this page is essentially identical to that of the root user interface.)

The data entry-level user interface (a JSP page – initial configuration shown).

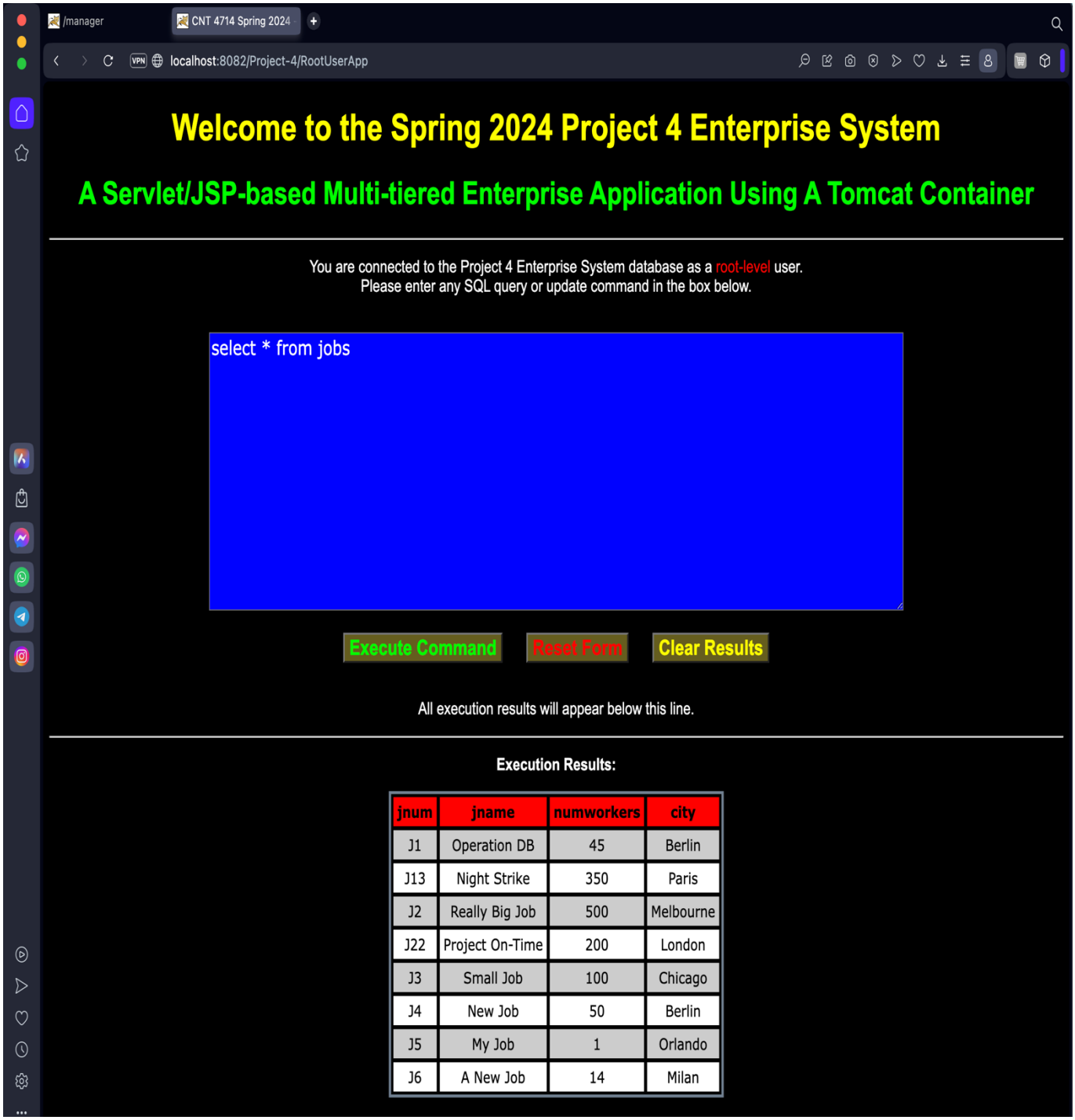


The accountant-level user interface (a JSP page – initial configuration shown).

Root-level User Examples

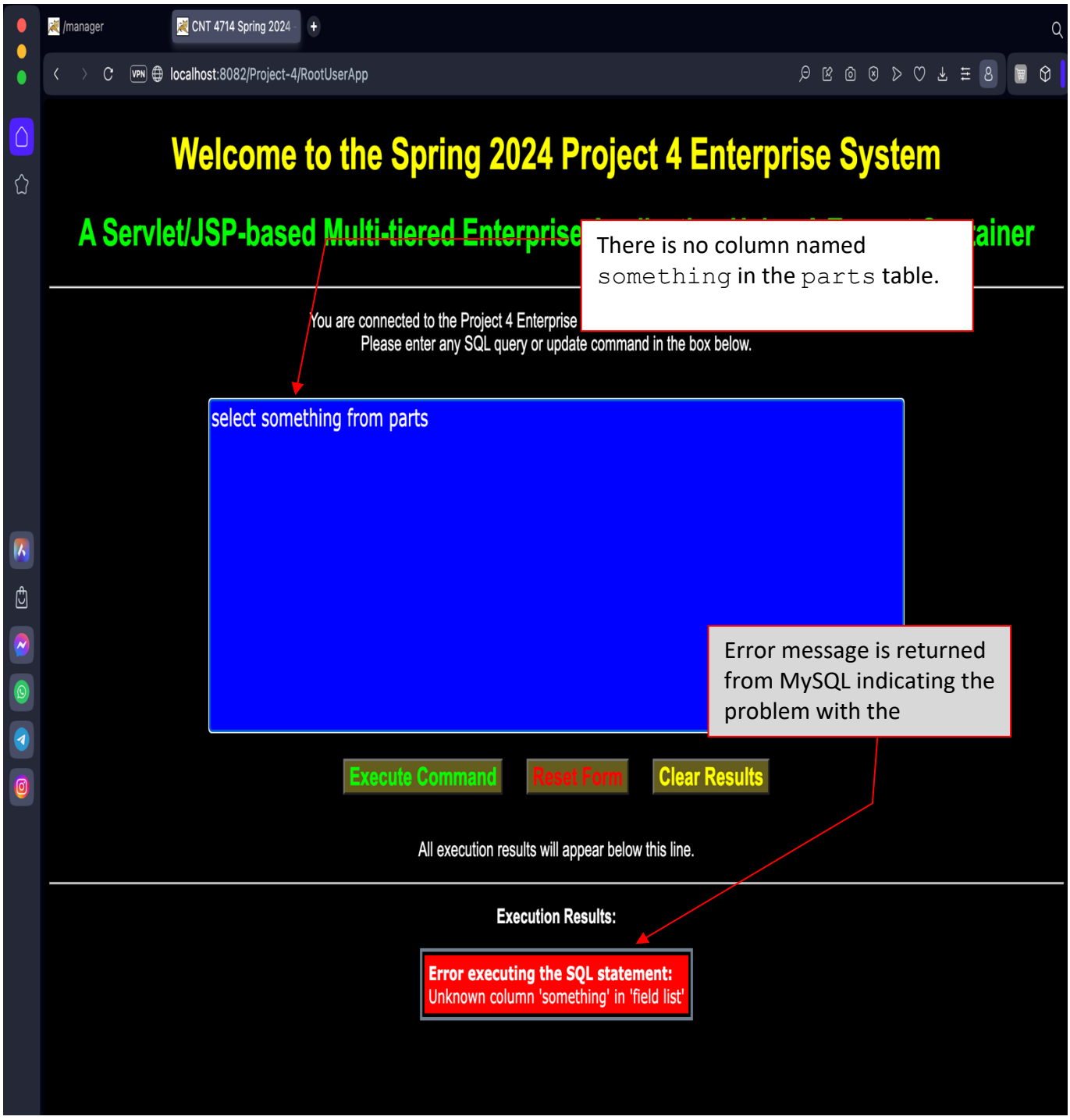
The following several screenshots illustrate operations from the root-level user interface.

After entering an SQL command, the user simply clicks the “Execute Command” button and the SQL command in the form is executed:



jnum	jname	numworkers	city
J1	Operation DB	45	Berlin
J13	Night Strike	350	Paris
J2	Really Big Job	500	Melbourne
J22	Project On-Time	200	London
J3	Small Job	100	Chicago
J4	New Job	50	Berlin
J5	My Job	1	Orlando
J6	A New Job	14	Milan

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):
Note the current status of supplier number S5. (Also note status of S1, S12, S17, S21, S22, S3, S44, and S6.)

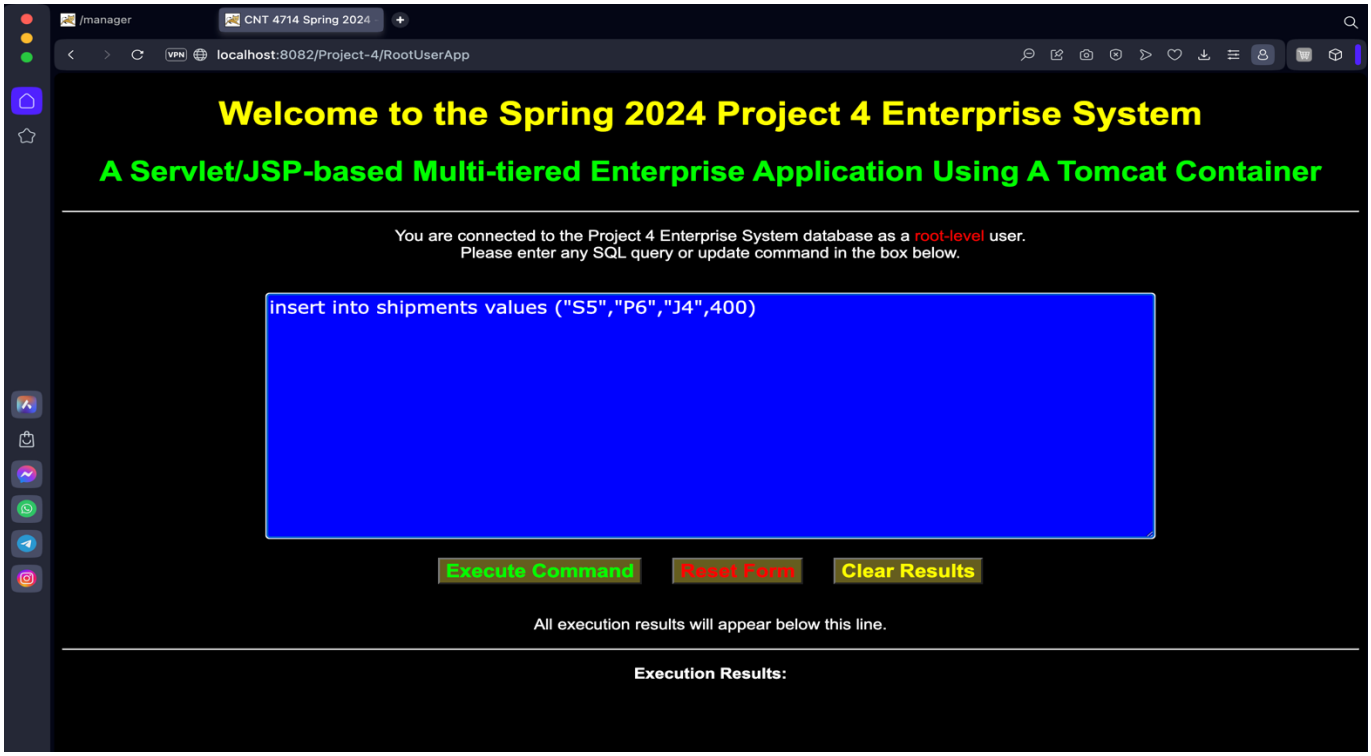
Database Results:

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

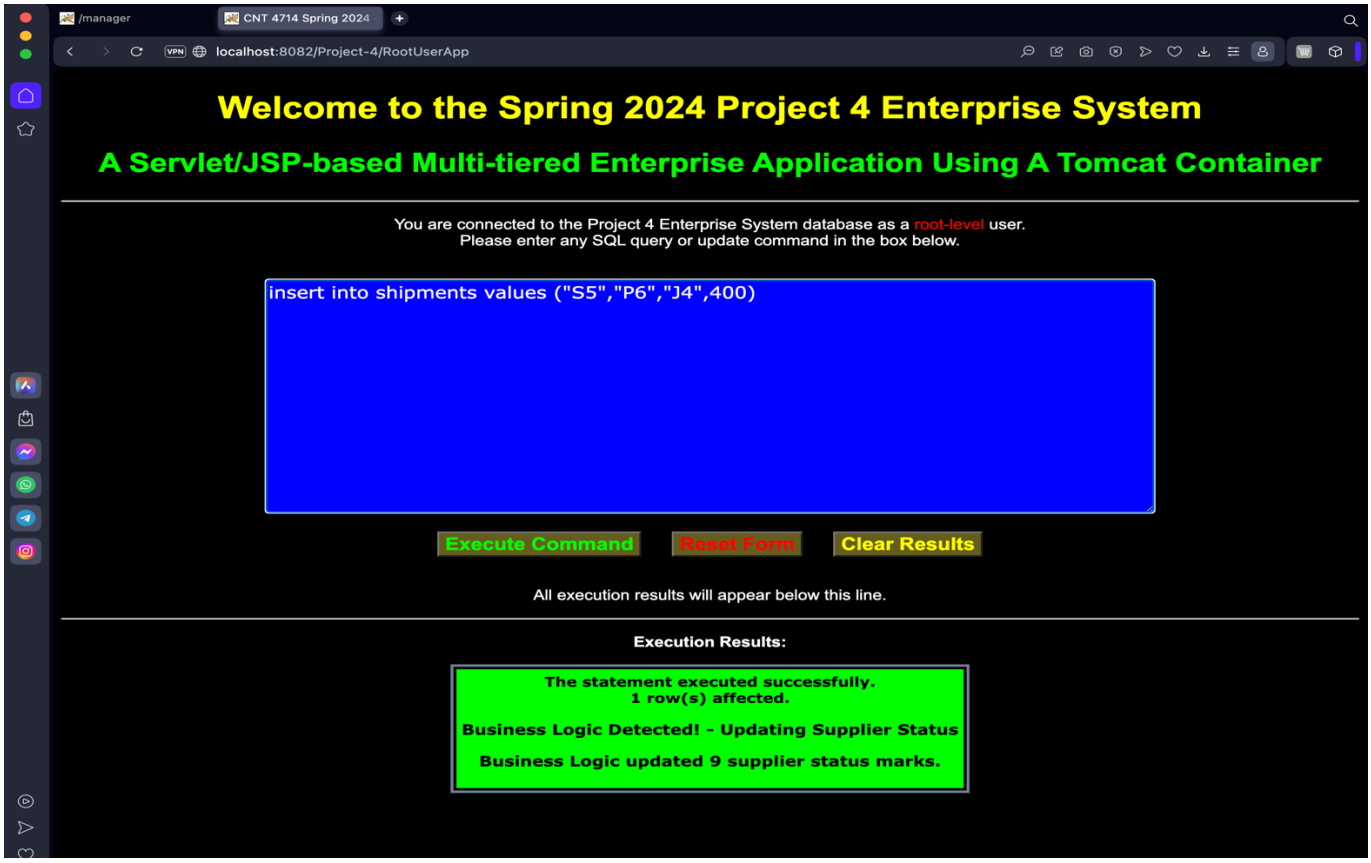
Results from running the user command `select * from suppliers` in the input area.

Results from running the query "select * from suppliers" – to be used to illustrate an update operation explained on pages 16-17. Notice that the supplier S5's status is currently 4.

Root 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). We will discuss the business logic in much greater detail in the Q&A sessions.

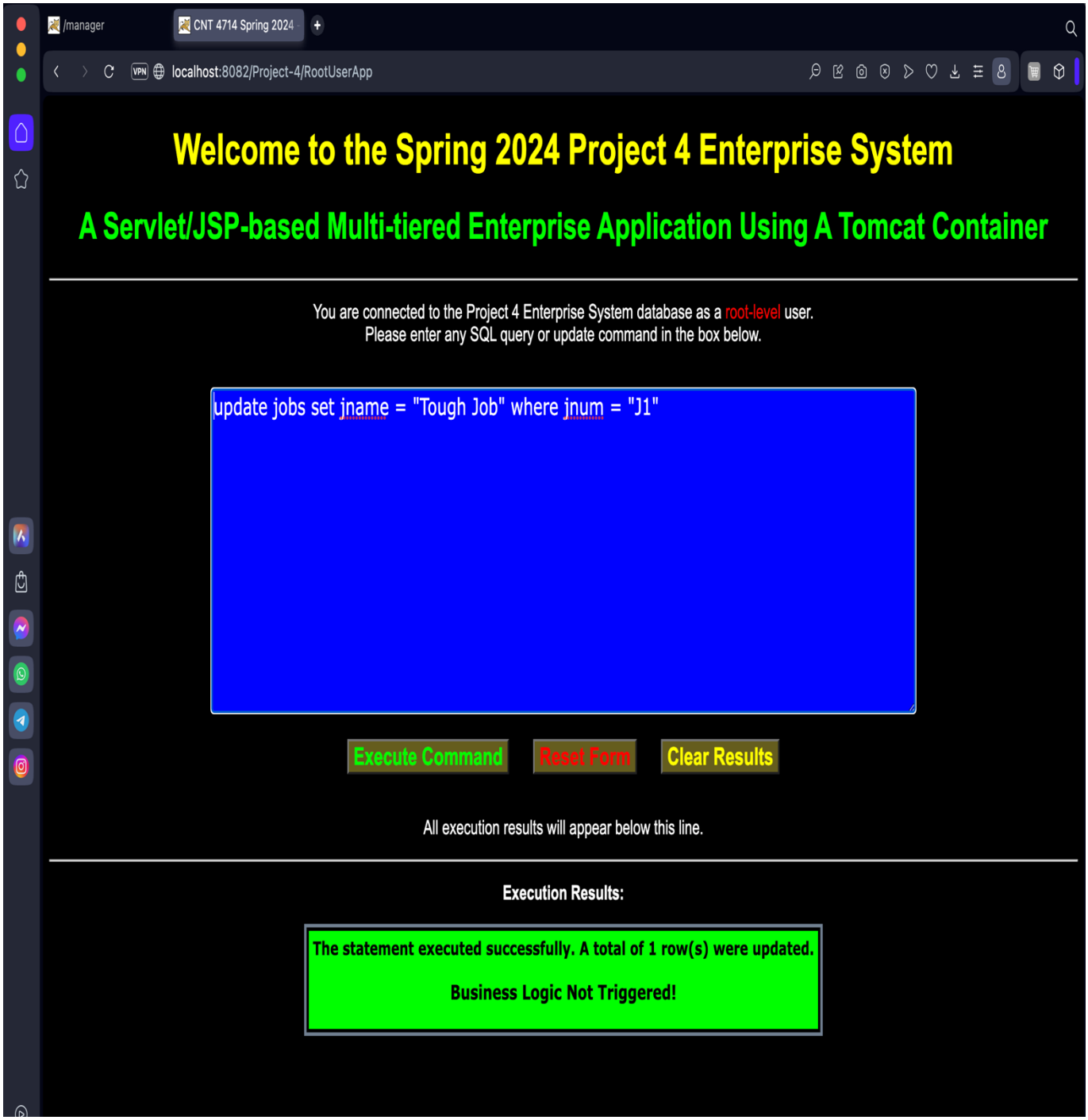
Database Results:

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

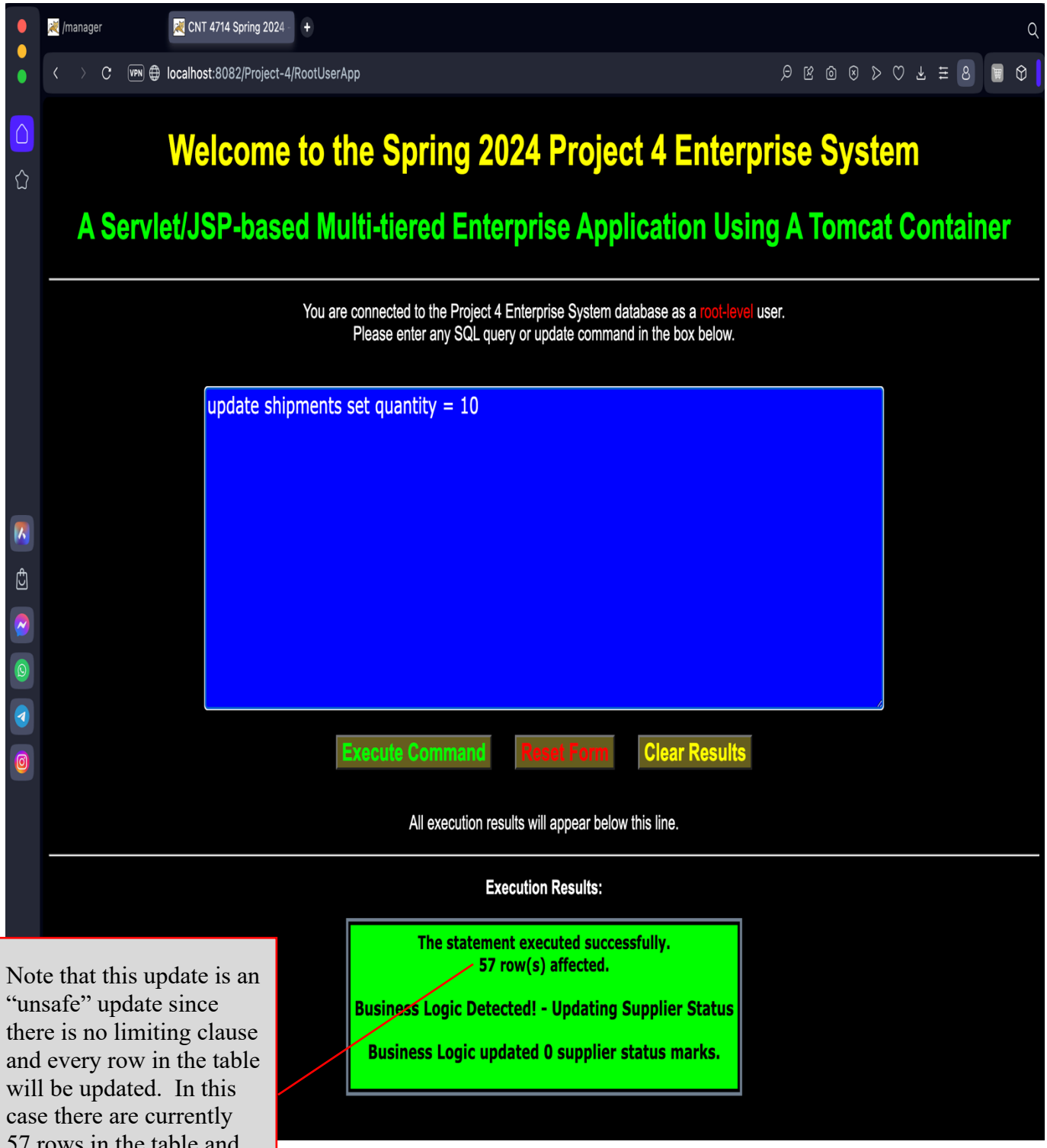
Notice on page 15 (in the original suppliers table) that supplier S5 had a status of 4. After this update, the business logic has increased supplier S5's status by 5, so it is now 9.

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.



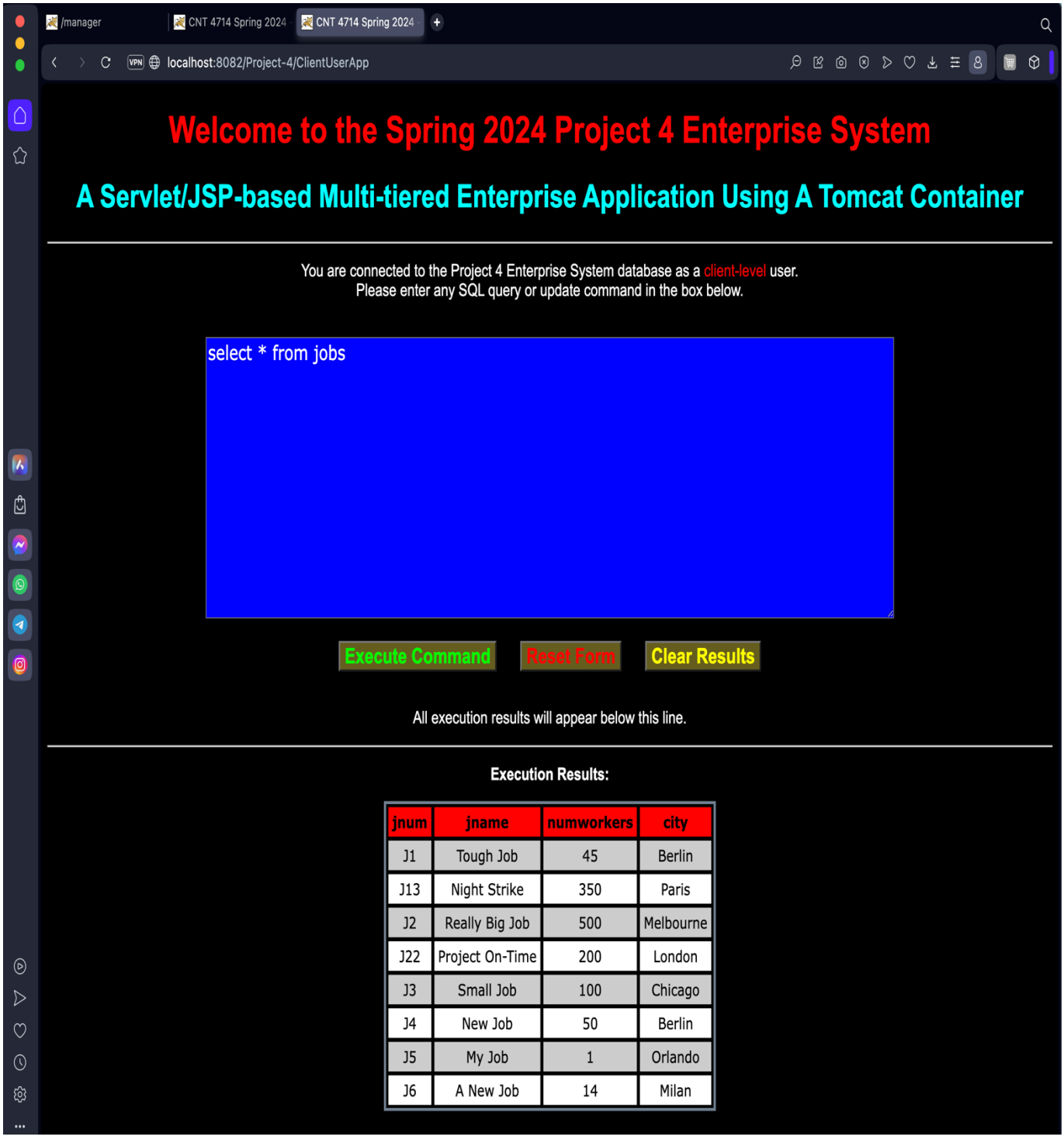
Example of an update command that triggers the business logic but results in no changes to any supplier's status.



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 there are currently 57 rows in the table and all 57 rows will now have a quantity of 10.

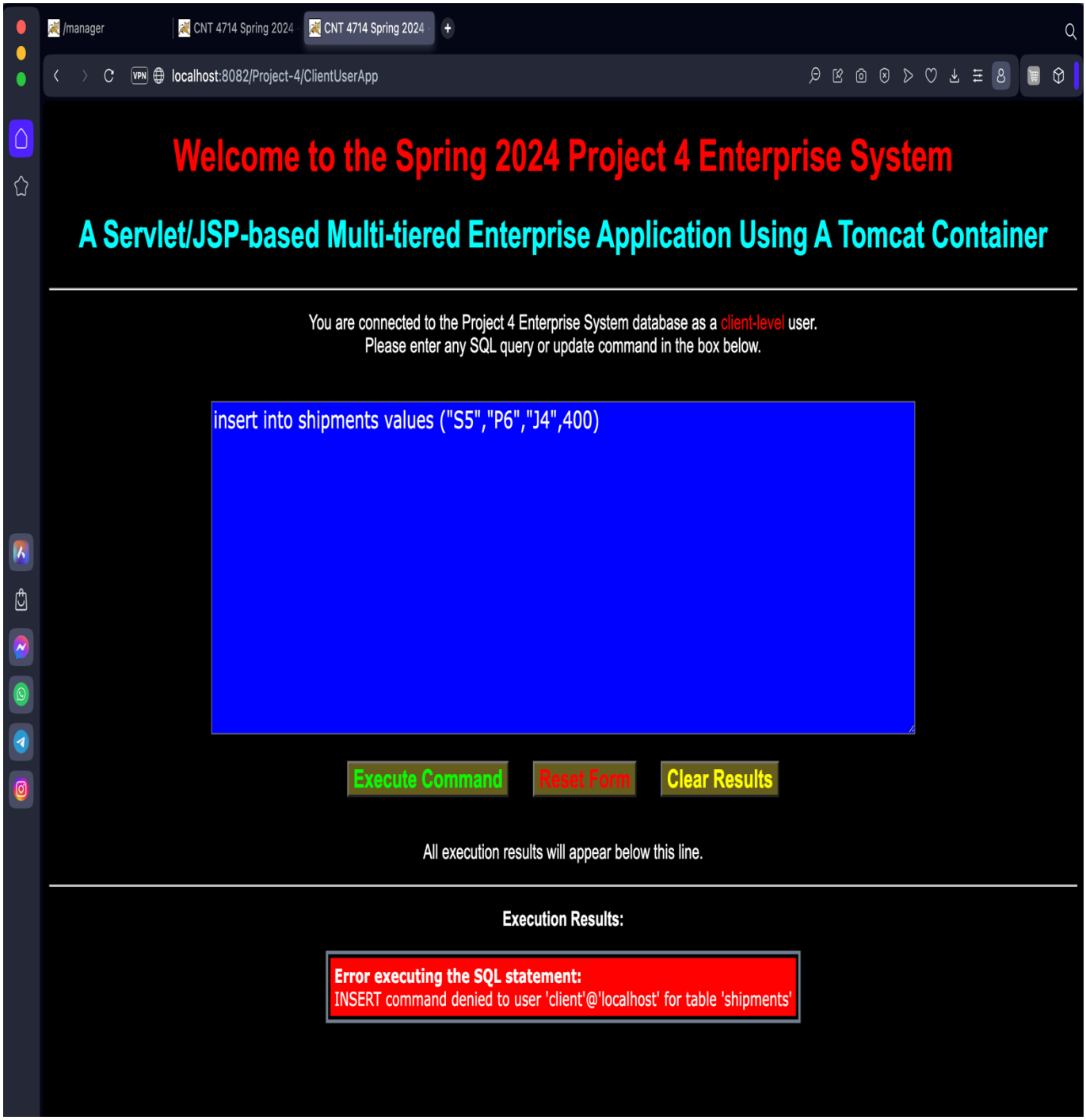
Client-level User Examples

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 15).



jnum	jname	numworkers	city
J1	Tough Job	45	Berlin
J13	Night Strike	350	Paris
J2	Really Big Job	500	Melbourne
J22	Project On-Time	200	London
J3	Small Job	100	Chicago
J4	New Job	50	Berlin
J5	My Job	1	Orlando
J6	A New Job	14	Milan

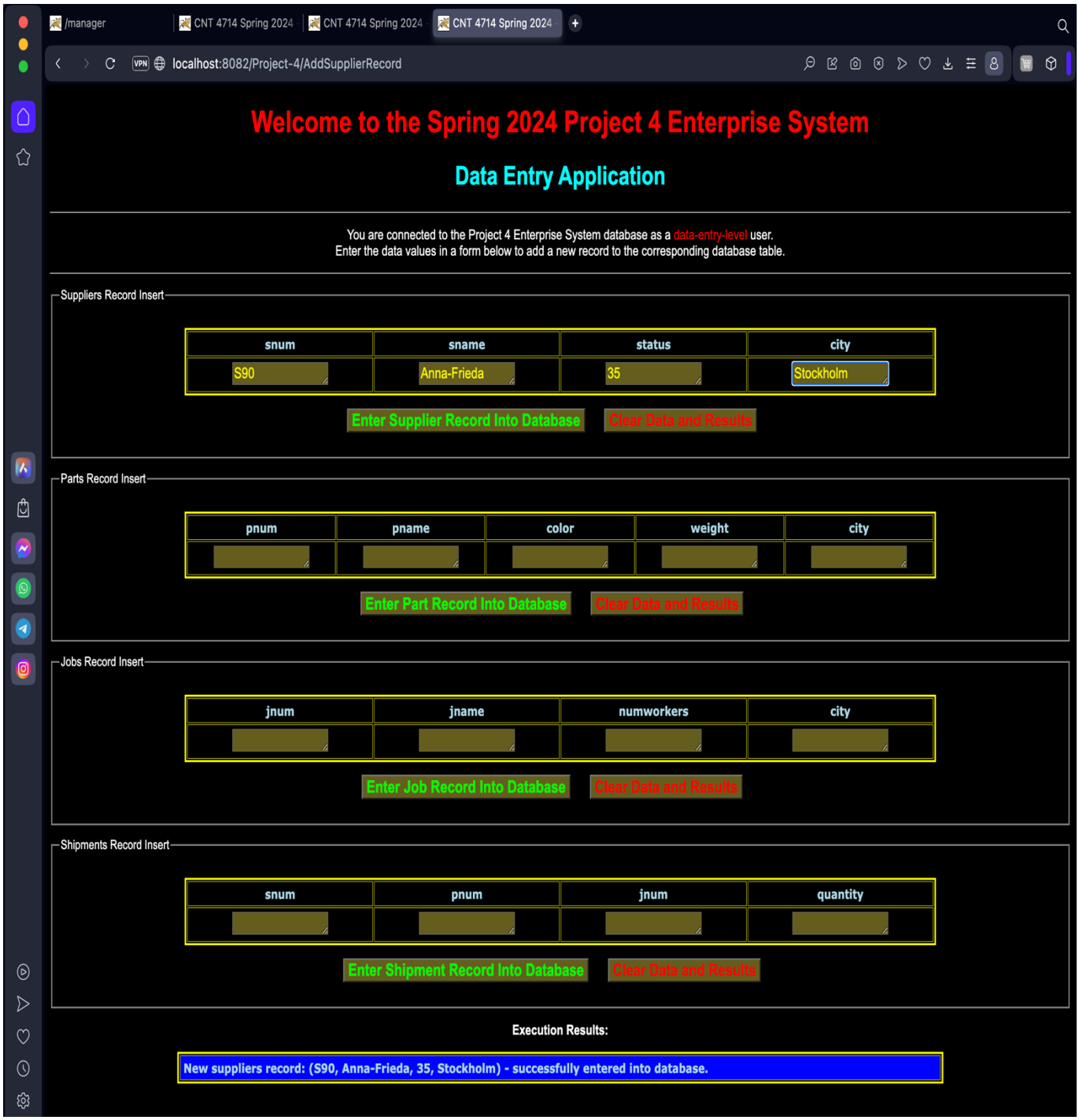
A client-level user issues a command for which they do not have privilege to execute.



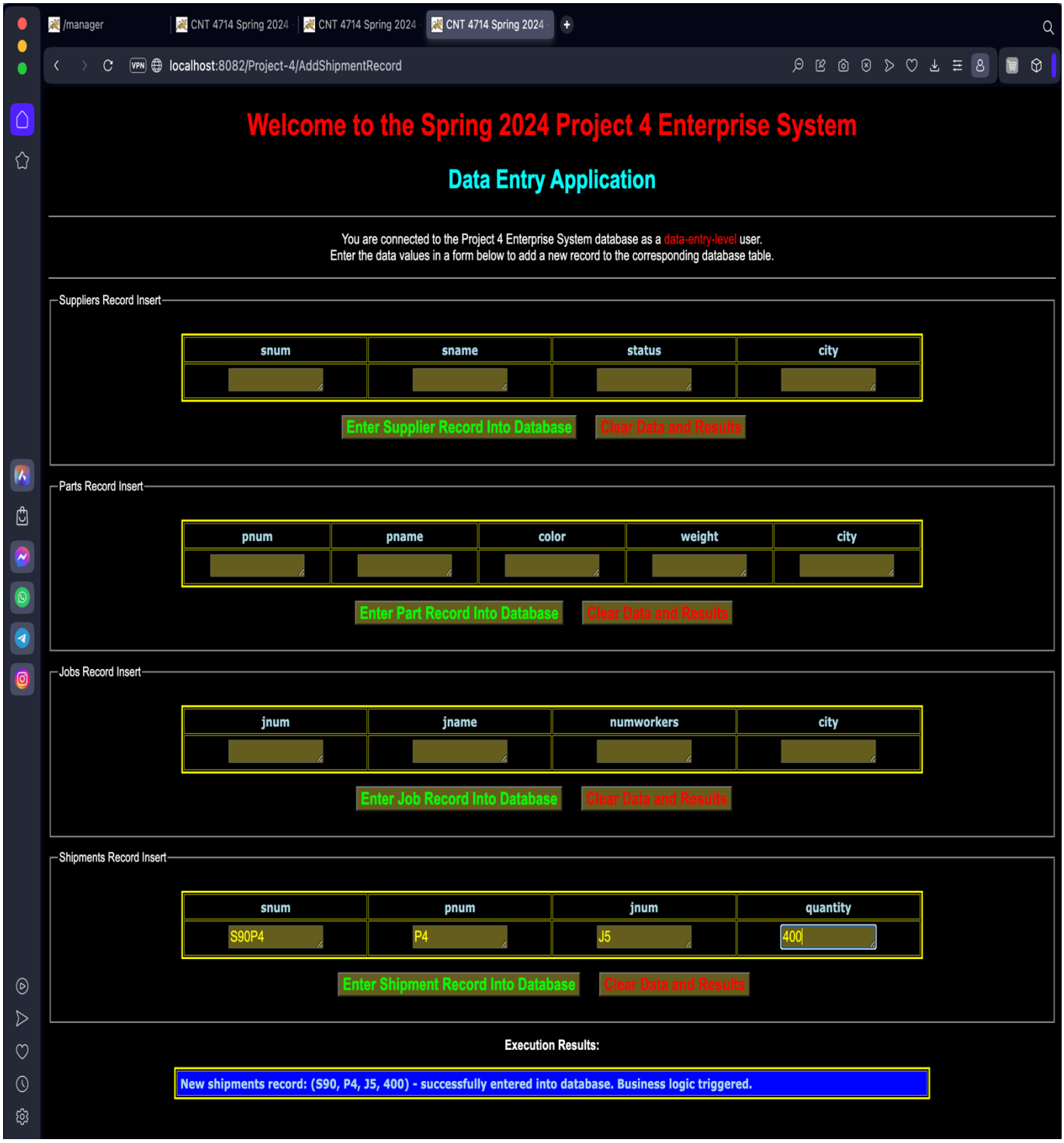
Data Entry-level User Examples

The data entry-level user interface provides a template (form) for the user to enter data values for new suppliers, parts, jobs, or shipments records. Although the page contains four separate forms, only one form can be submitted at a time. Note: the database does not specify any default values for any attributes in any of the table in the `project4` database. Therefore, the user must supply all values to be used by the PreparedStatement interface in each form.

Data entry user enters a new suppliers table record.



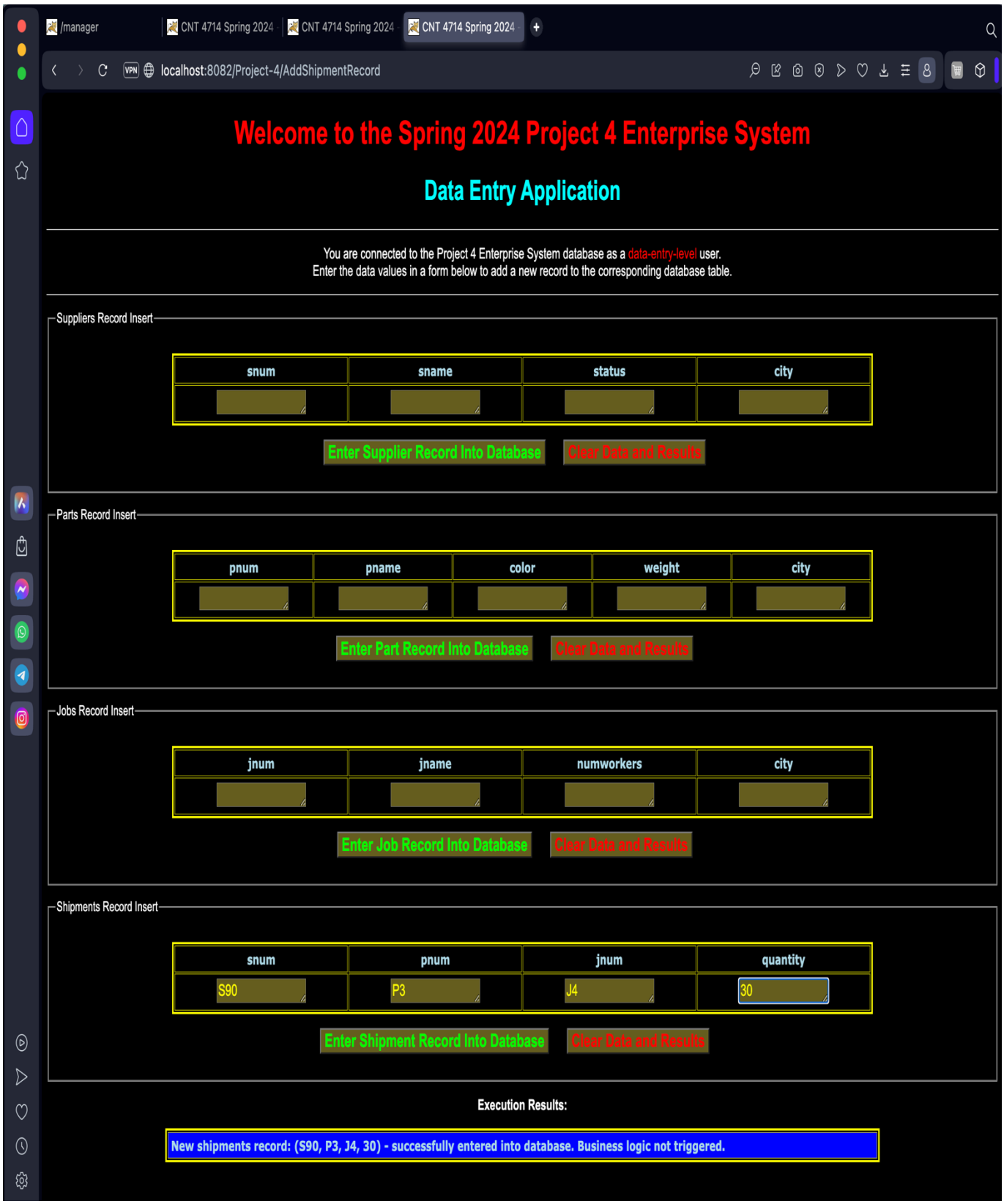
In this example, the quantity value will trigger the business logic and update supplier S5's status by 5.



	S8	Giancarlo Fisichella	3	Milan
	S9	Kimi Rikonnen	2	Helsinki
	S90	Anna-Frieda	40	Stockholm

Workbench view of the suppliers table after dataentryuser update shown above.

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.

Accountant-level User Examples

The accountant-level interface provides a list of "reports" that can be run against the project4 database. These reports are generated by invoking remote procedure calls to execute stored procedures residing on the MySQL DB server. The remote procedures are invoked using the CallableStatement interface. As with the other interfaces in this web application, the results of the remote procedure execution will be returned to the bottom of the original page.

Accountant user select first option.

Accountant user select fourth option

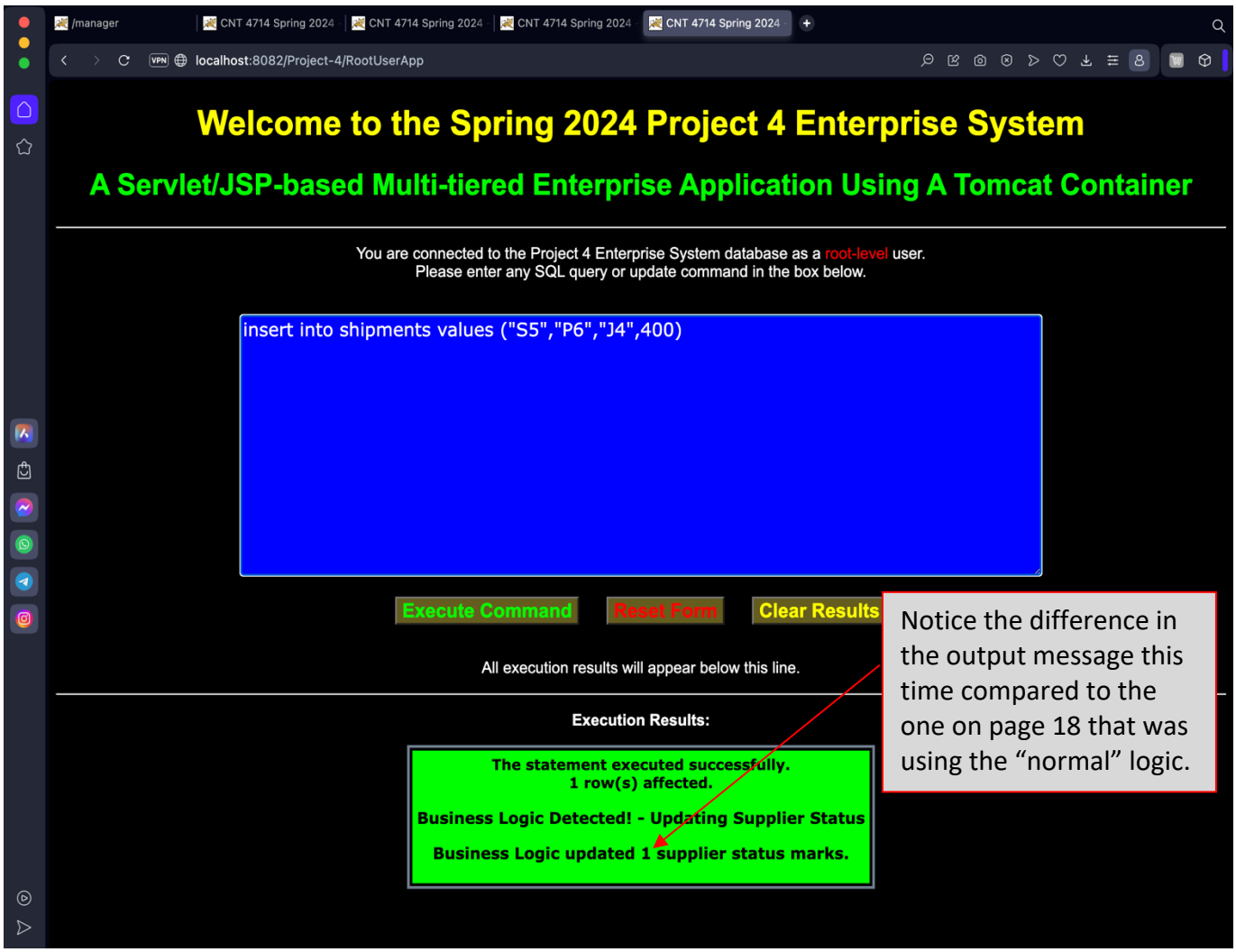
BONUS PROBLEM: 15 points

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 18), we now get the correct effect for our update command. See next page.



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 19.

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.