James Roebuck

IT351–1703A: Advanced Java Programming

CTU

August 9, 2017

Prof. Anthony Lowe

Contents

[Unit 1: Client Server Application 3](#_Toc490047795)

[GUI Prototype 3](#_Toc490047796)

[Class Diagram 3](#_Toc490047797)

[Sequence Diagram 4](#_Toc490047798)

[Pseudocode 5](#_Toc490047799)

[Successful Program Execution 7](#_Toc490047800)

[Unit 2 8](#_Toc490047801)

[Class Diagram 8](#_Toc490047802)

[Pseudocode Logic 8](#_Toc490047803)

[Server Class 8](#_Toc490047804)

[ThreadingServerHelper run() method 9](#_Toc490047805)

[Updated ServerHelper Class 9](#_Toc490047806)

[Multiple Client Connections to Server – Screen Captures 10](#_Toc490047807)

[Unit 3 – Database-Driven Desktop Applications 13](#_Toc490047808)

[MySQL Setup and NetBeans Connection to Database 13](#_Toc490047809)

[Setup and Configuration 13](#_Toc490047810)

[Integration into NetBeans Application 17](#_Toc490047811)

[GUI Prototype 19](#_Toc490047812)

[Input Requirement Exceptions for Product and Customer Create/Update 20](#_Toc490047813)

[Display Database 20](#_Toc490047814)

[Create Database Record 21](#_Toc490047815)

[Update Database Record 23](#_Toc490047816)

[Delete Functions 26](#_Toc490047817)

[Unit 4 – Database-Driven Web Application 28](#_Toc490047818)

[System Architecture 28](#_Toc490047819)

[Setup 28](#_Toc490047820)

[Deployment and Use of Web Application 32](#_Toc490047821)

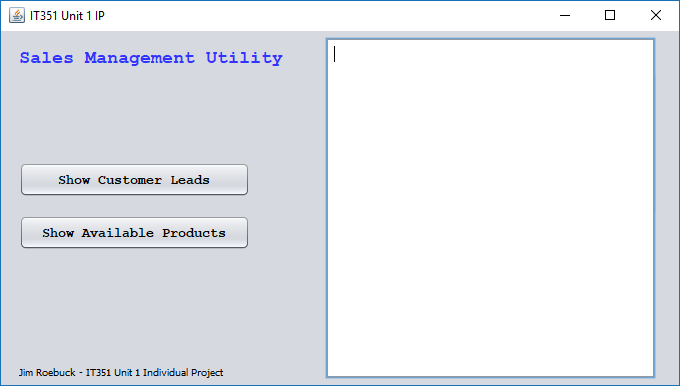
[Unit 5 – User Manual and Testing 42](#_Toc490047822)

[User Manual and Installation Guide 42](#_Toc490047823)

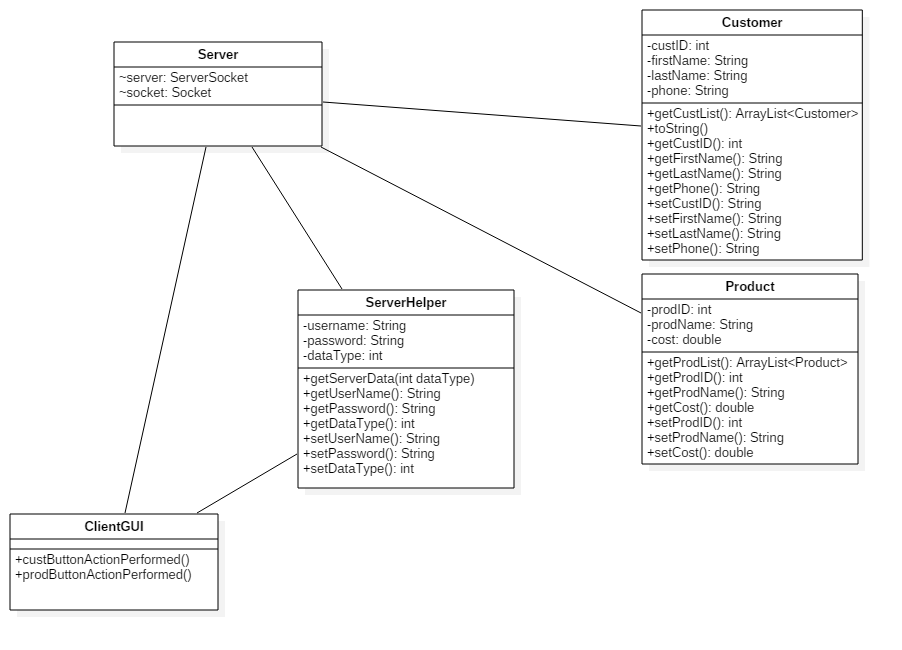
[Test Plan 54](#_Toc490047824)

# Unit 1: Client Server Application

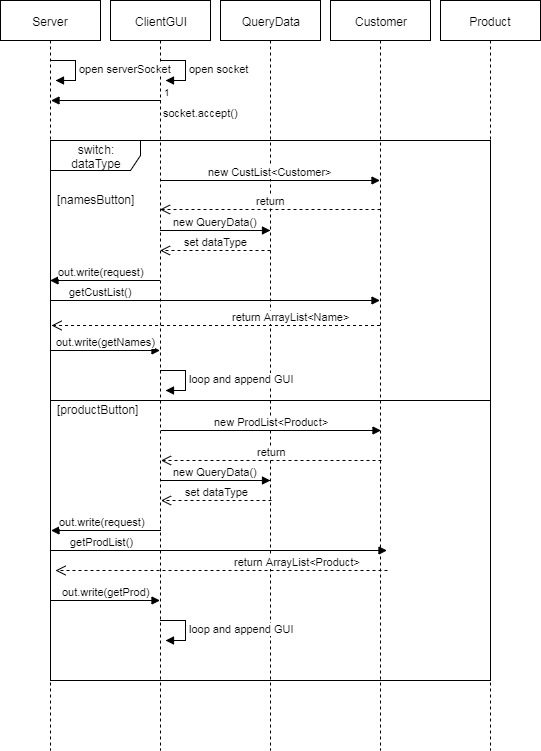
## GUI Prototype



## Class Diagram



## Sequence Diagram

**

## Pseudocode

**Server class**

The server class will exist only as a mechanism for connecting to a client, receiving queries, and returning data. A server socket connection is created and assigned a port. The server then loops until exited. . Data streams are opened, and awaits client requests. It then takes in a QueryData request, and if a switch condition is met, the object is written to an ArrayList. The buffer is flushed, and the server reiterates the loop.

*Server main method*

CREATE ServerSocket object

LOOP until exit

Upon Request

CREATE socket object

ACCEPT server connection

CREATE ObjectInputStream

CREATE ObjectOutputStream

CREATE QueryData object

READ in query

SWITCH datatype

case 1 Customer

WRITE getCust() objects to ArrayList to OutputStream

case 2 Product

WRITE getProd() objects to ArrayList to OutputStream

**ClientGUI class**

The ClientGUI will mainly call on other classes and will consist mainly of action events of the GUI components. For the first week, only two operations are taking place, the display of customer and product information to the GUI. When the buttons of the interface are pressed, methods are called to create and open a socket connection to the server, create data streams, and request objects to List that is sent back to the Customer or Product action events. These Lists are then looped through, with each object being appended to the JTextArea.

*namesButtonActionPerformed*

CREATE Socket

CONNECT to Server

CREATE new customer ArrayList

CAST getDataRequest(DataRequest.NAMES) to ArrayList

CLEAR PREVIOUS DATA IN JTextArea

LOOP through Names ArrayList

APPEND objects to JTextAREA

*prodButtonActionPerformed*

CREATE Socket

CONNECT to Server

CREATE new products ArrayList

CAST getDataRequest(DataRequest.PRODUCTS) to ArrayList

CLEAR PREVIOUS DATA IN JTextArea

LOOP through Product ArrayList

APPEND objects to JTextAREA

**QueryData class**

Following the video lectures this week, I’m going to try and create the component that both object types being requested from the server can utilize. I think I want to move it from the GUI client class, however, and for now it’s moved to the QueryData class for server interactions. Some of the pseudocode is redundant from the previous pseudocode, as I’m following the pattern of only establishing the socket connection when data is being queried.

*getDataFromServer(datatype: int)*

CREATE Socket object and connect to “localhost”

CREATE ObjectOuputStream object

CREATE ObjectInputStream object

CREATE new QueryData object

SET object dataType

WRITE QueryData object to output stream

Flush buffer

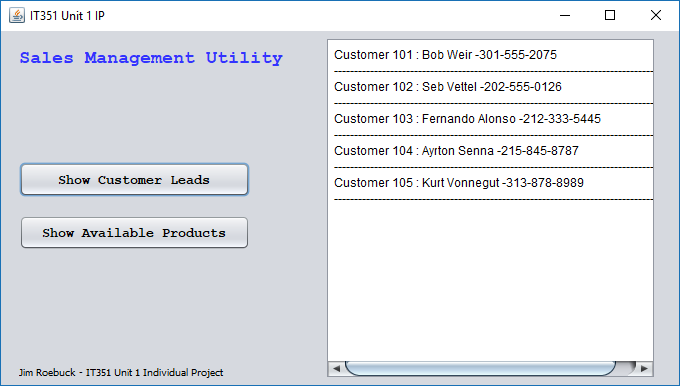
CREATE LIST

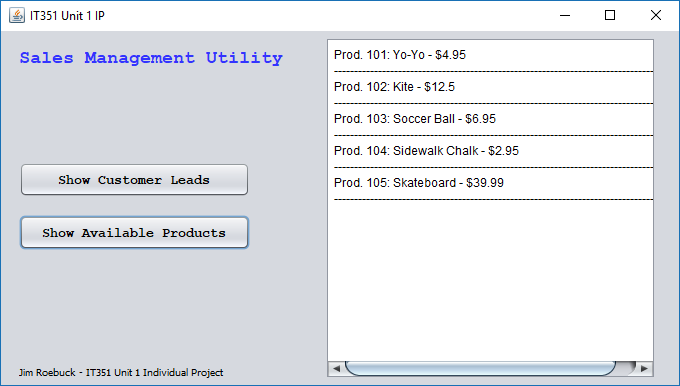
ADD returned input stream to list

CLOSE socket connection

RETURN List

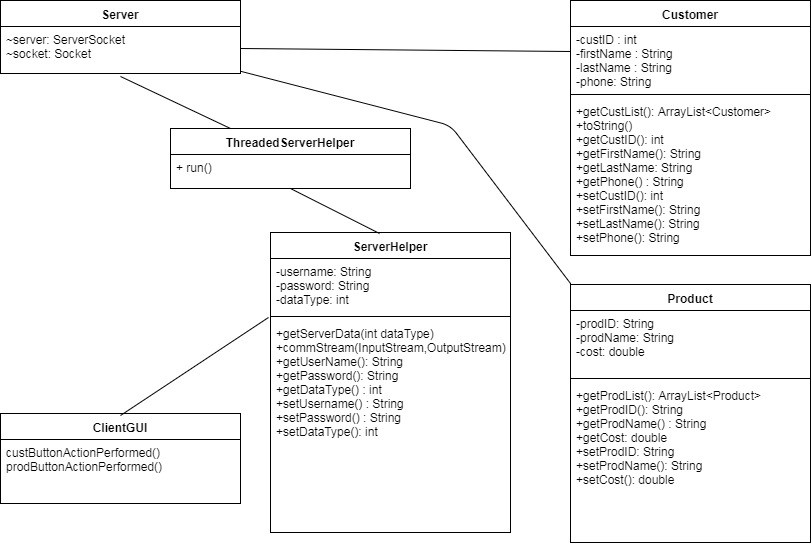
## Successful Program Execution





# Unit 2

## Class Diagram



## Pseudocode Logic

### Server Class

CREATE ServerSocket object

Loop Until Exit {

ACCEPT Socket connection

CONSTRUCT ThreadingServerHelper object

Set constructor input to socket ObjectInputStream

Set constructor output to socket ObjectOutputStream

CONSTRUCT new thread object

Set thread to ThreadingServerHelper object

Start thread }

### ThreadingServerHelper run() method

Construct ServerHelper object

Call ServerHelper.commStream()

Set commStream() to ThreadingServerHelper input and output stream

### Updated ServerHelper Class

**commStream(InputStream inStream, OutputStream outStream)**

Construct socket object

Set to server and socket number

Construct ObjectInputStream object and set to method arg inStream

Construct ObjectOutputStream object and set to method arg outStream

Construct ServerHelper object

Cast ServerHelper object to object input stream

Conditional switch from ServerHelper.getDataType()

Case ServerHelper.NAMES:

Write Customer.getCustList() to outStream

Flush

Break

Case ServerHelper.PLACES

Write Product.getProdList() to outStream

Flush

Break

**getServerData(int dataType)**

Construct ObjectOutputStream object

Construct ObjectInputStream object

Construct new ServerHelper object

Set object datatype to method arg

Write ServerHelper object to OutputStream

Flush buffer

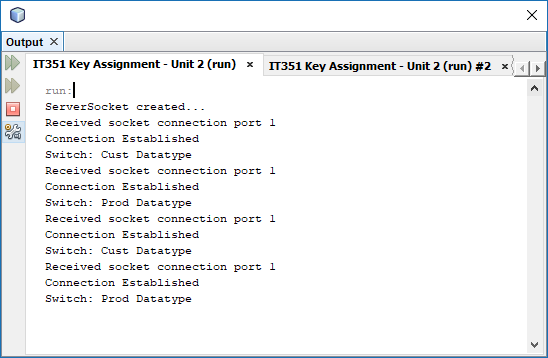
Create new list

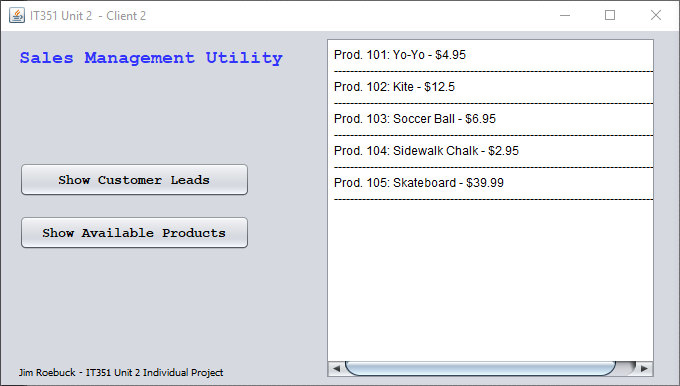
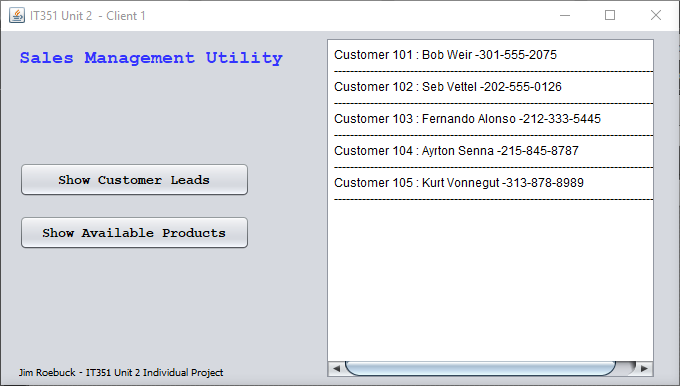
Add returned input stream to list

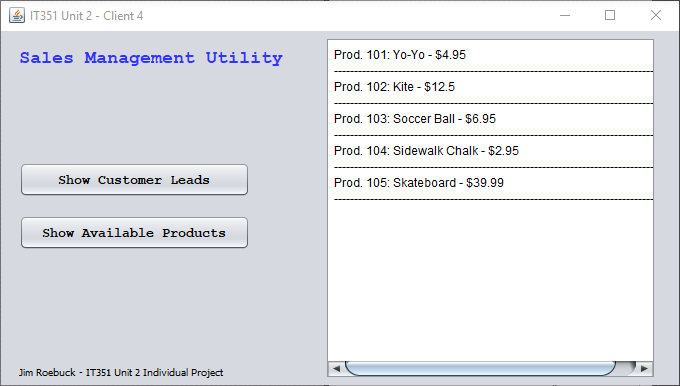
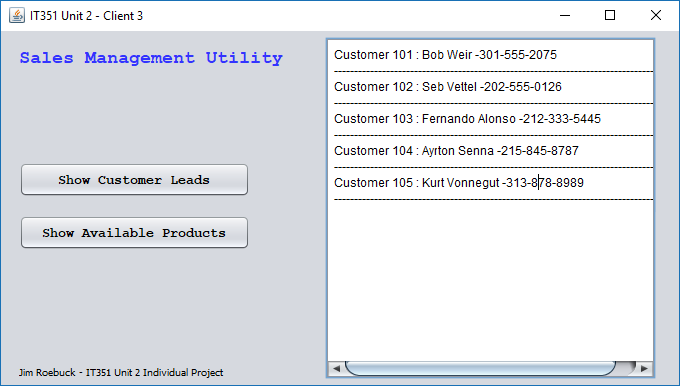
Close socket connection

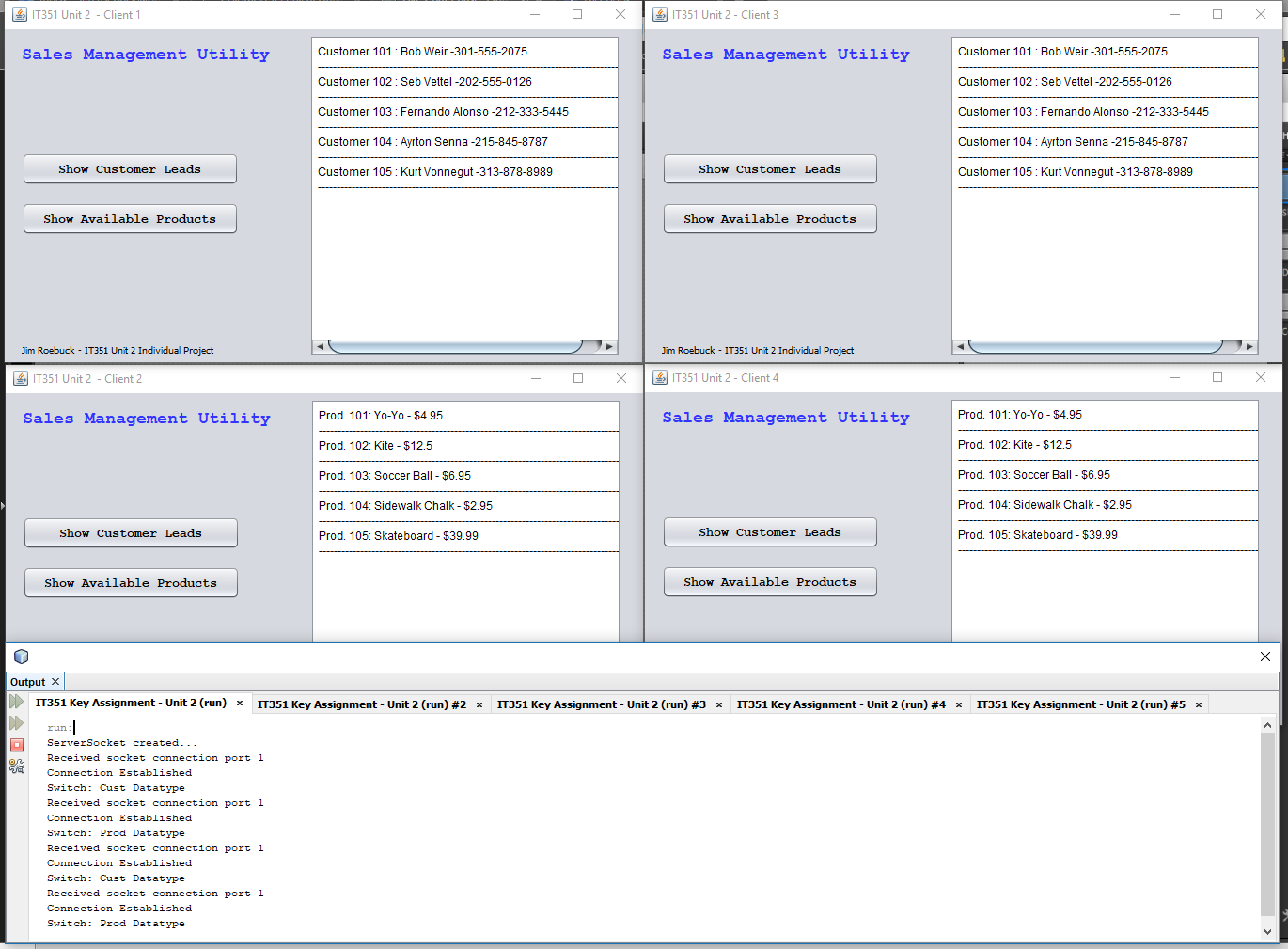
Return list

## Multiple Client Connections to Server – Screen Captures









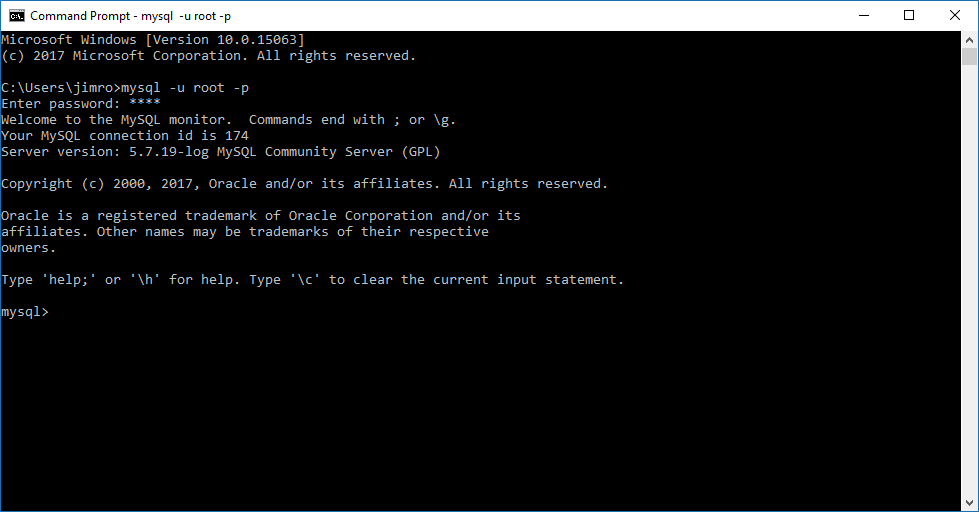
# Unit 3 – Database-Driven Desktop Applications

## MySQL Setup and NetBeans Connection to Database

## Setup and Configuration

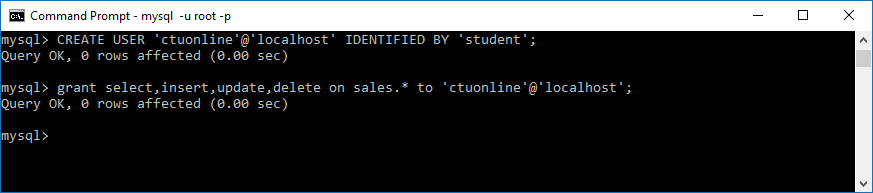
Everything about this week’s assignment is brand new and extremely exciting for me. As I progressed through the videos and integrated my program with the MySQL database structures, I quickly began to feel like I’m getting into increasingly powerful and exciting areas I’ve yet to explore in Java. Though there’s some complexity in installing and configuring the server and command structure, I found setting up the environment to be pretty intuitive. The only issue I ran into was that I didn’t initially set the windows services for the SQL server to initialize upon Windows startup, which created connection issues for me when I was attempting to reconnect later. However, I was able to reconfigure this service in MySQL workbench to automatically start when Windows boots, and the issue is resolved

Connected to MySQL using root:



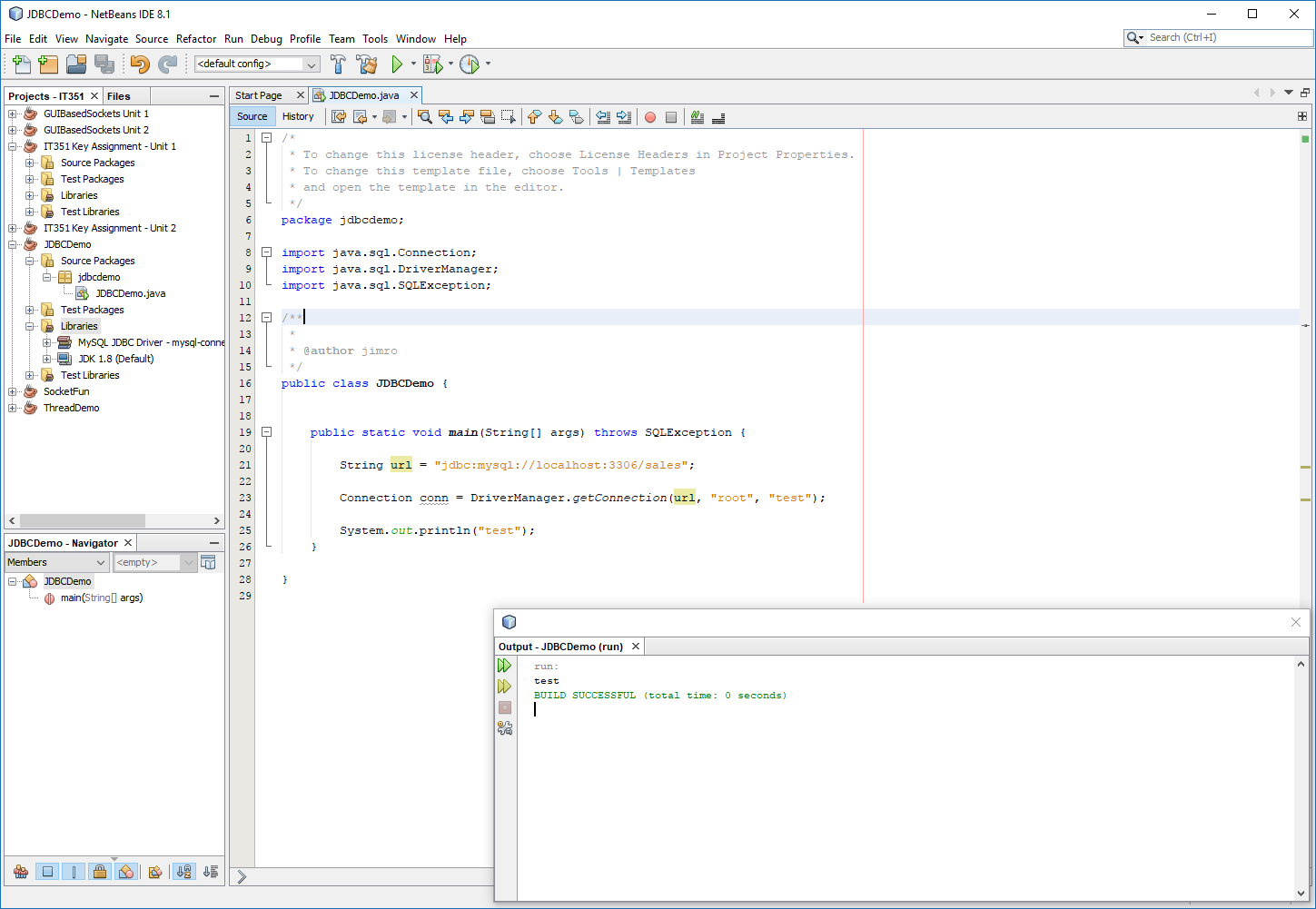
Next, we are instructed to create a user login which will be used in the Java program to connect to the SQL databases using the statement:

mysql> CREATE USER 'ctuonline'@'localhost' IDENTIFIED BY 'student';

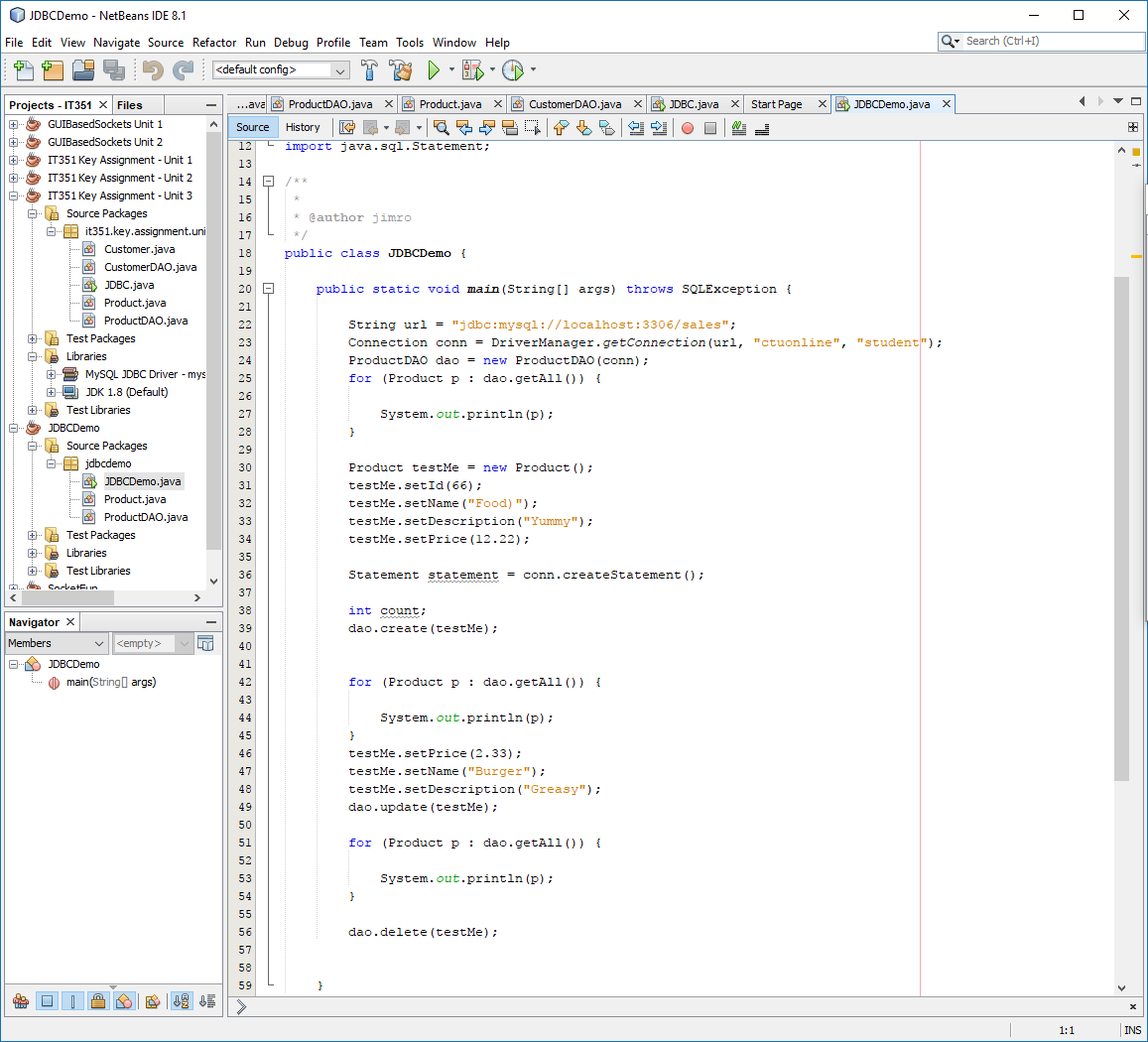


Now that the user is created, the Java Database Connection (JDBC) program created in NetBeans can connect to the SQL database. First, the JDBC connection URL points the application to the correct database. The URL used, jbdc:mysql://localhost:3306/sales, identifies the JDBC protocol, the MySQL sub-protocol to identify which database structure is being connected, the database host as well as connecting port for the MySQL connection, as well as the database name to access (tutorialspoint, 2017). Then, DriverManager interface passes the username and password to the database server, which is wrapped in a Java.sql.Connection which provides the database connectivity.

NetBeans Connected to MySQL database

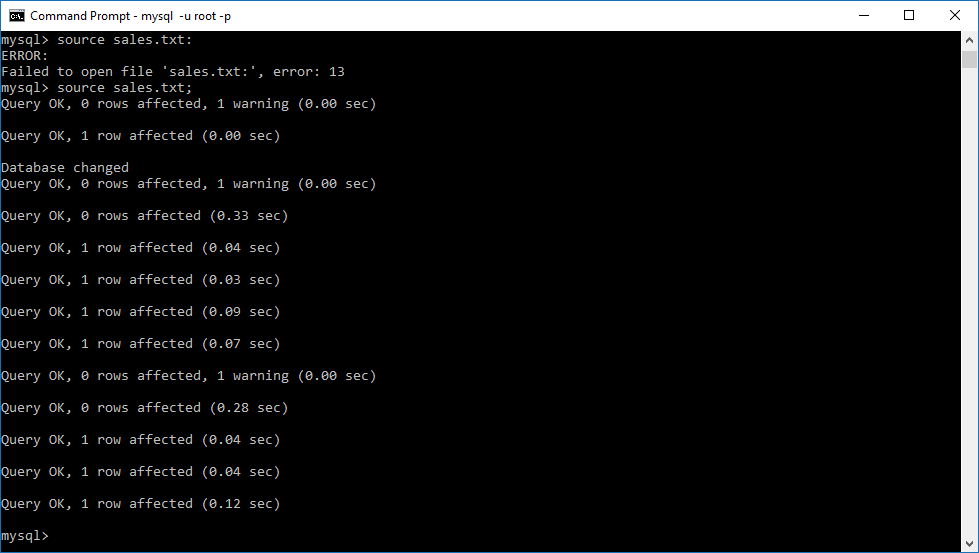


NetBeans Reconfigured to use ctuonline/student login to MySQL

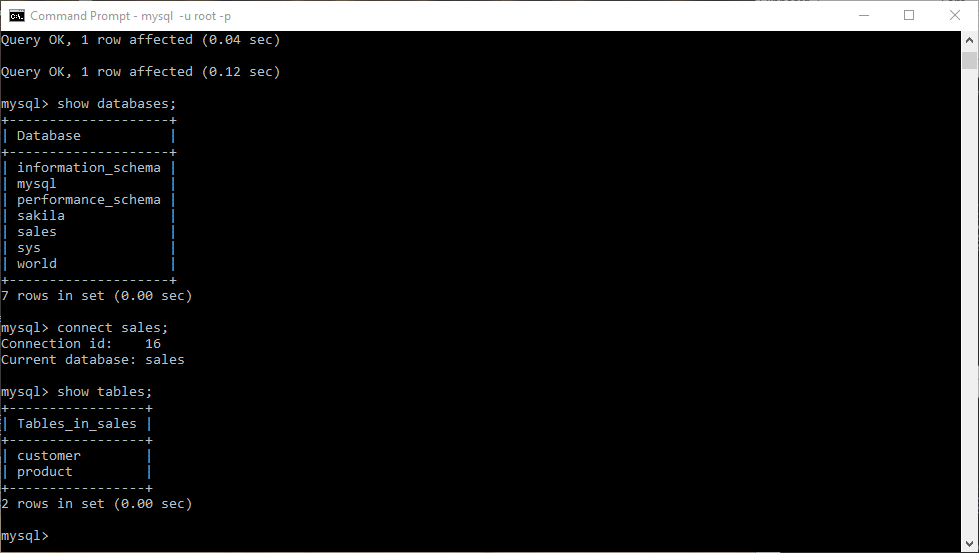


At this point, the JDBC connection is established, and the MySQL database is up and running. Next, the provided database schema file needs to be imported. Though this can be done within the SQL Workbench application, I chose to utilize the SQL commands, as outside of the single course I’ve now taken this is otherwise new to me.

Importing Sales Database

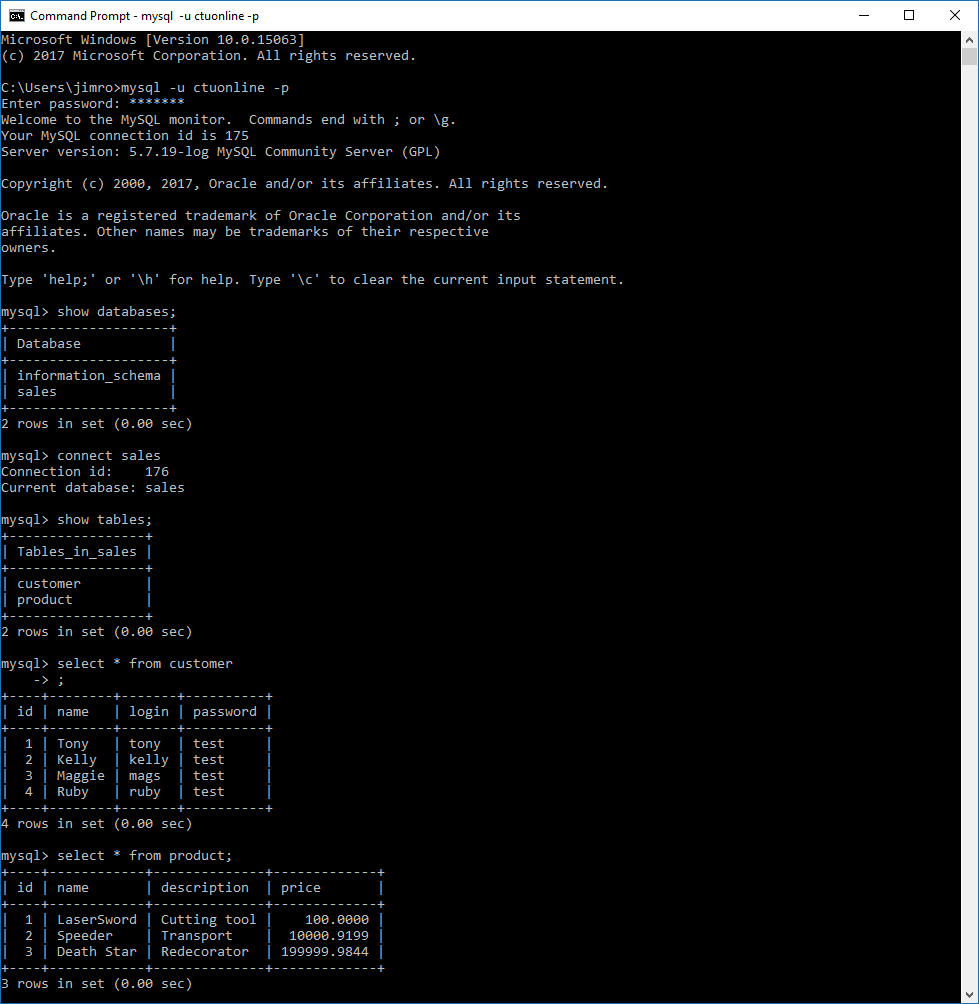


Displaying Imported Sales Database Customer and Product Tables



Once the database is imported into MySQL, all of the tabular data can be parsed and manipulated through SQL queries and statements. Additionally, as the JDBC is established in NetBeans, calls from the Java application are can also return or manipulate data in the database tables. To illustrate the tables have been successfully imported into MySQL, the following show the customer and product table contents.

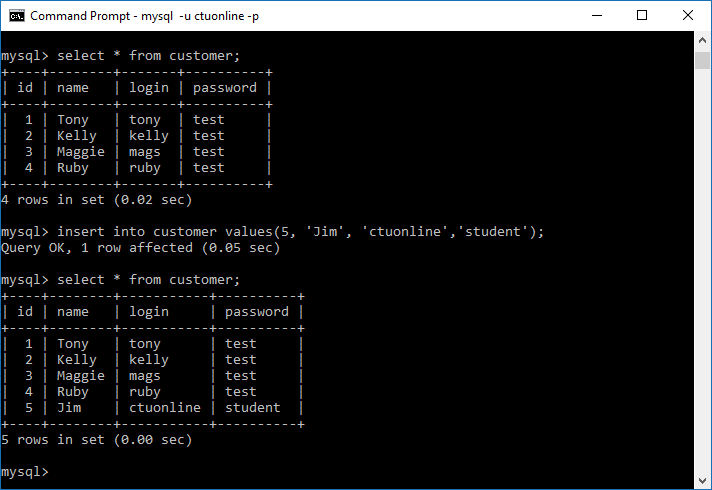
Customer and Product Tables Queried



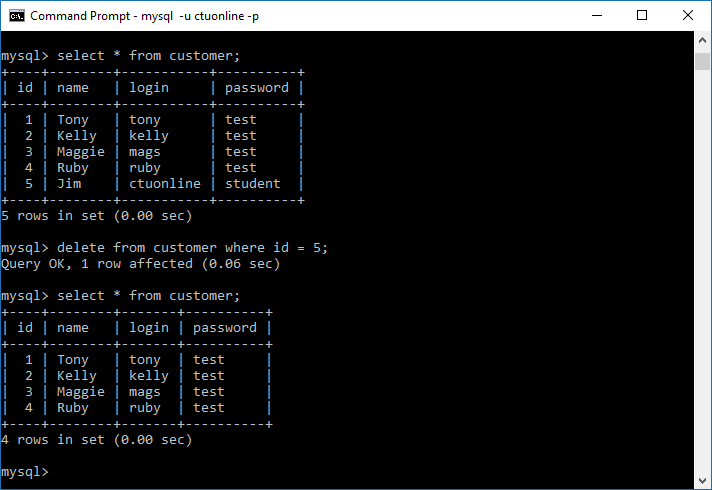
## Integration into NetBeans Application

Once the connection between the database and NetBeans is established, much of the logic follows the progression from the previous course, which managed handling data as it passes between objects. In order to increase the versatility and ability to manage database queries and updates, Data Access Object components are built to manage the product and customer tables and objects. Building these powerful, versatile components really helped me understand the depth of how this hiding this dirty work from the rest of the application helps to isolate the front and back end of the program. I found this part of the work to be extremely rewarding, and see how focus on setting up this foundation would allow the design of the GUI to seamlessly integrate into the components of the JBDC and DAO structures.

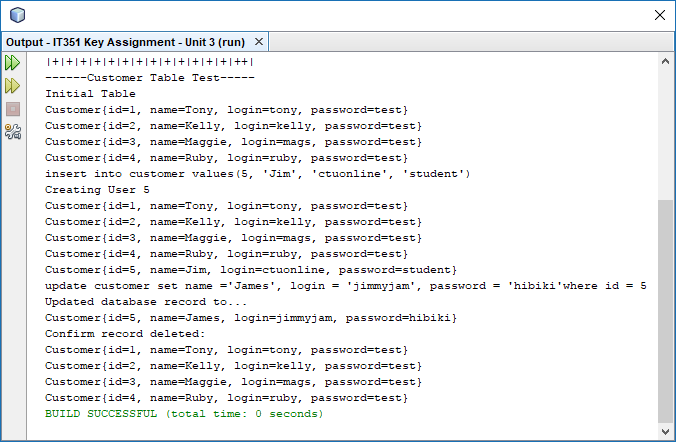
Following the logic set forth in the instructional videos on the product table, I created the CustomerDAO to handle the communication between the JDBC class and MySQL. One thing that helped build the components was to create the query first on the command line, and then using this syntax in Netbeans, while abstracting out the data into data accessors. Once I had my logic and syntax down, I was able to build create(), update(), and delete() methods to handle data retrieval and manipulation. Additionally, a method is built to loop through the Customer objects, retrieve their data, and add them to an ArrayList.

Testing logic to implement in Customer DAO which allowed me to introduce proper syntax into my component:  


Testing delete syntax to plug into my Customer DAO

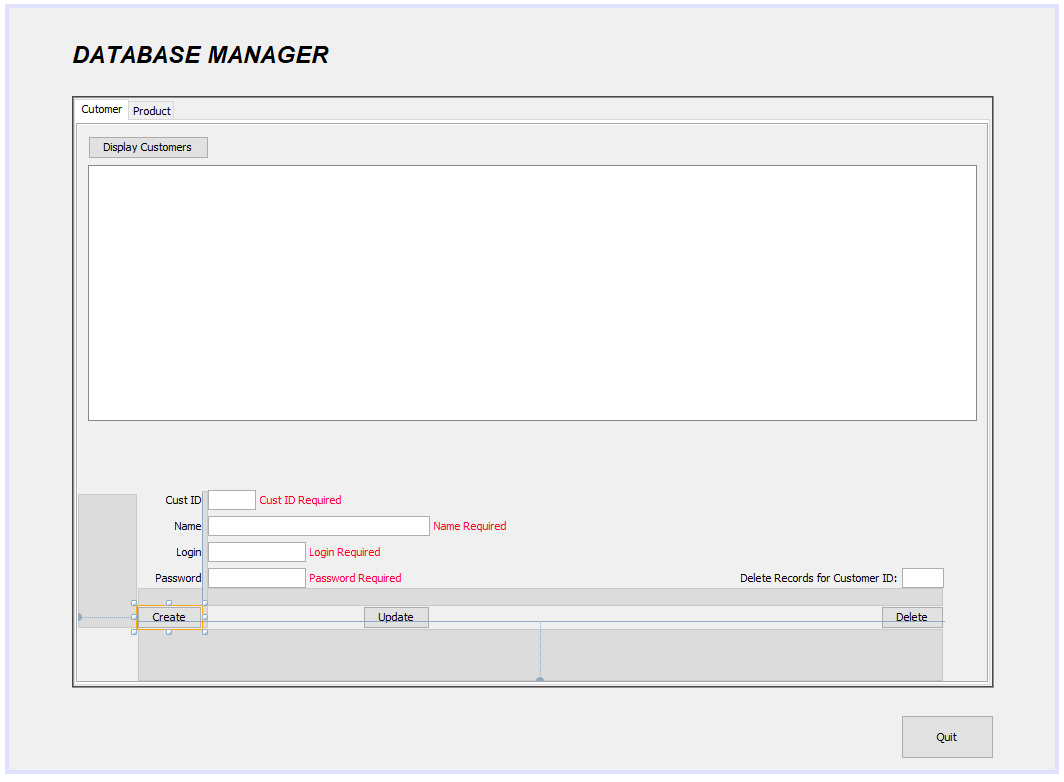


Output of JDBC main(), which uses CustomerDAO class create(), update(), getAll(), and delete() methods

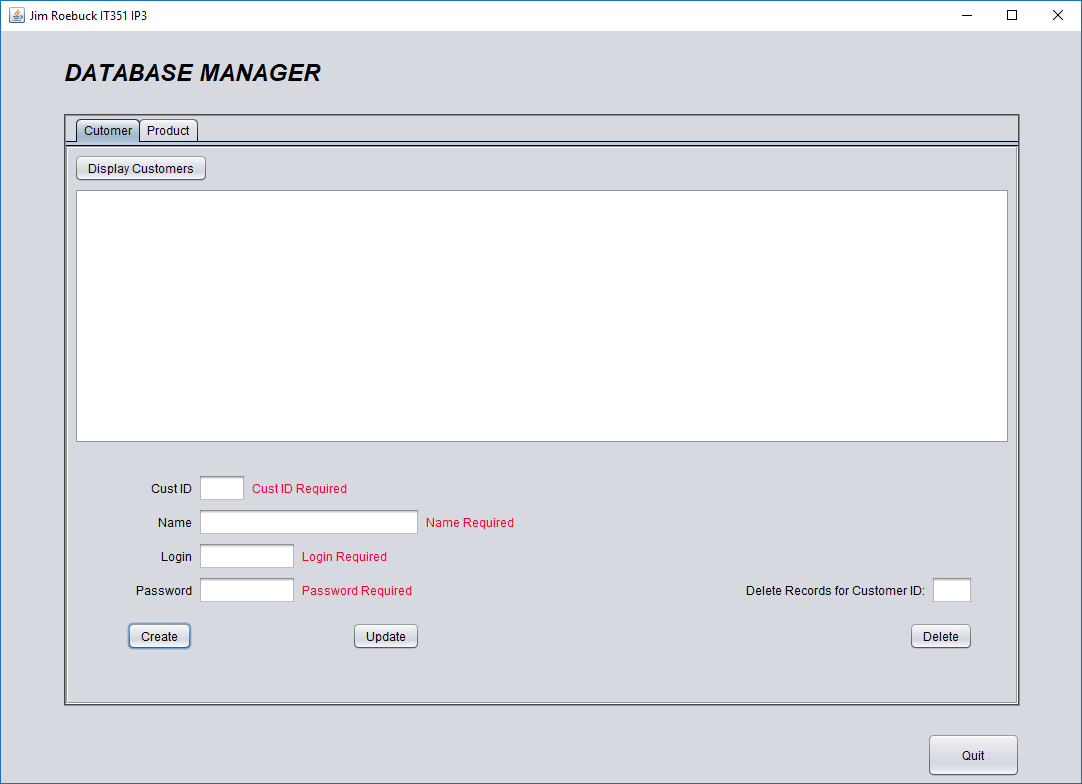


Though this assignment and preparation took more time than I expected it would at the onset of the week, I’m incredibly happy to feel excited and prepared to tackle the Individual Project this week. Having completed the first stages of the assignment, I can access the customer and product tables, and query, create, update, and delete the data through my application. The end result of this project is that my database is connected to my Java program, and using SQL commands through DAO components, I have a logical program structure that should allow me the ability to be successful in building a GUI that handles these functions in an intuitive, visually-appealing design.

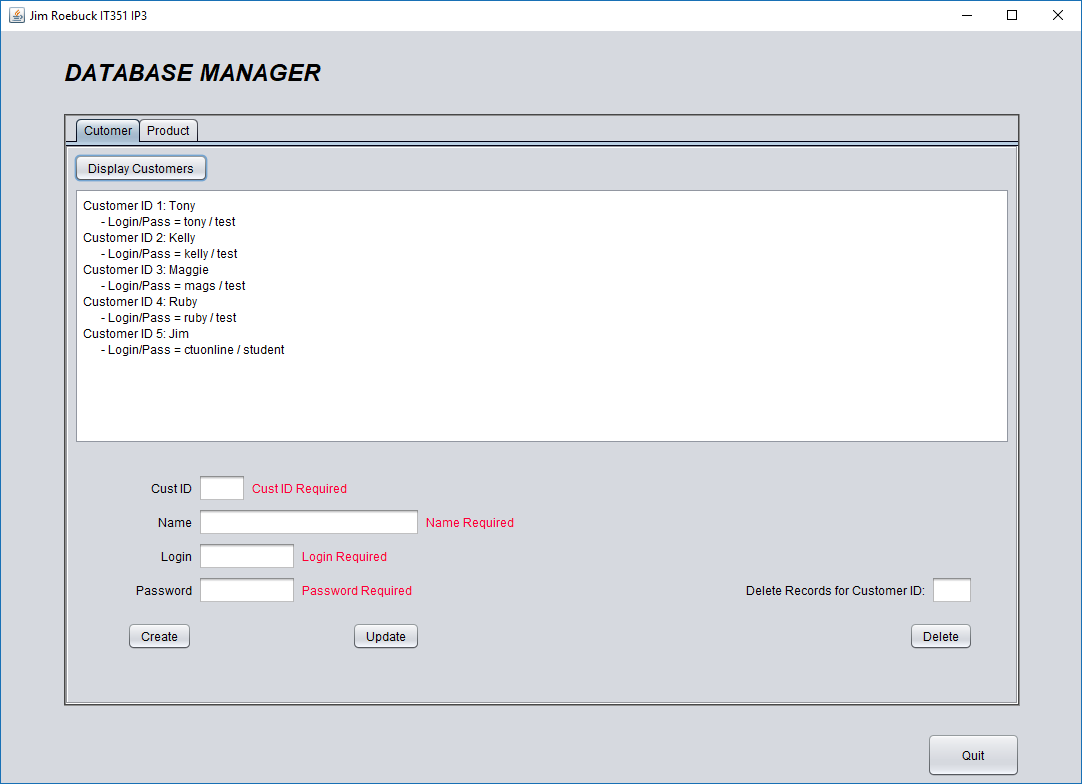
### GUI Prototype

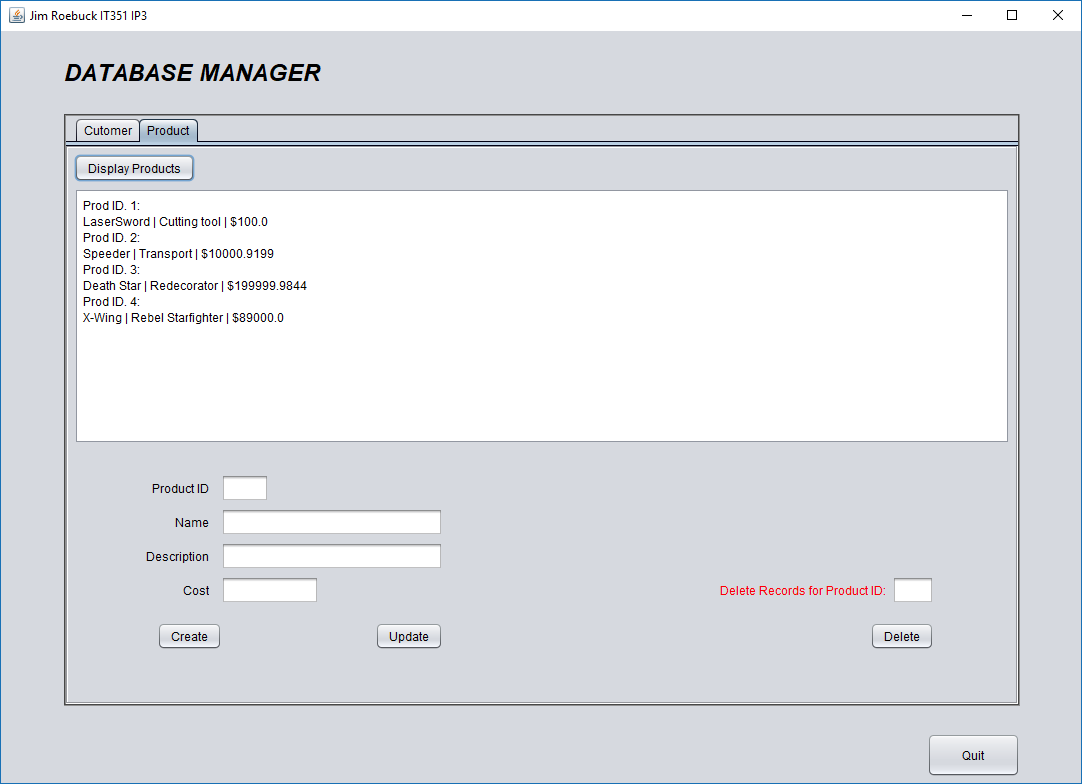


### Input Requirement Exceptions for Product and Customer Create/Update



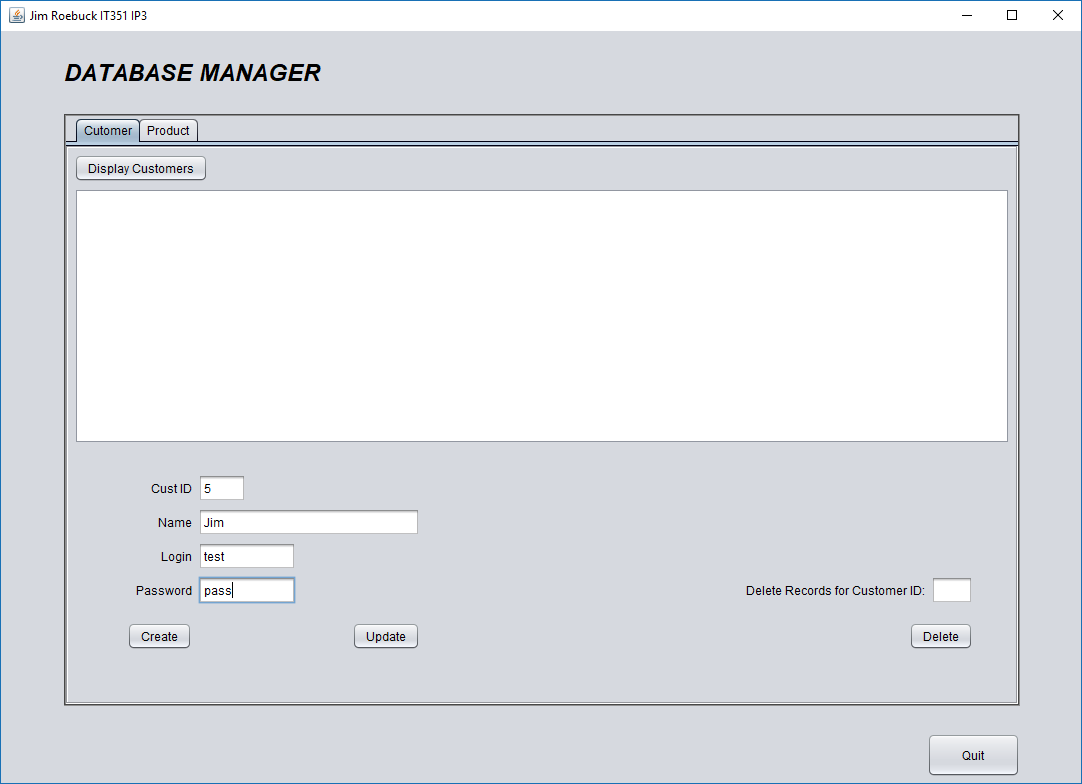
### Display Database



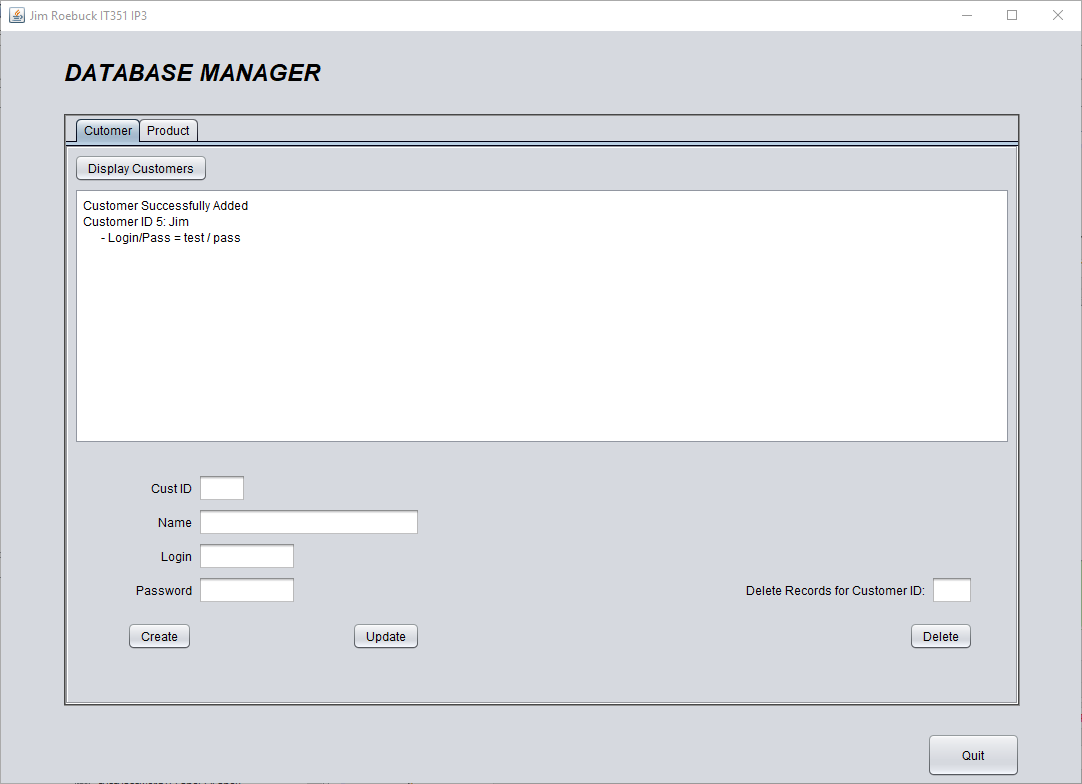


### Create Database Record

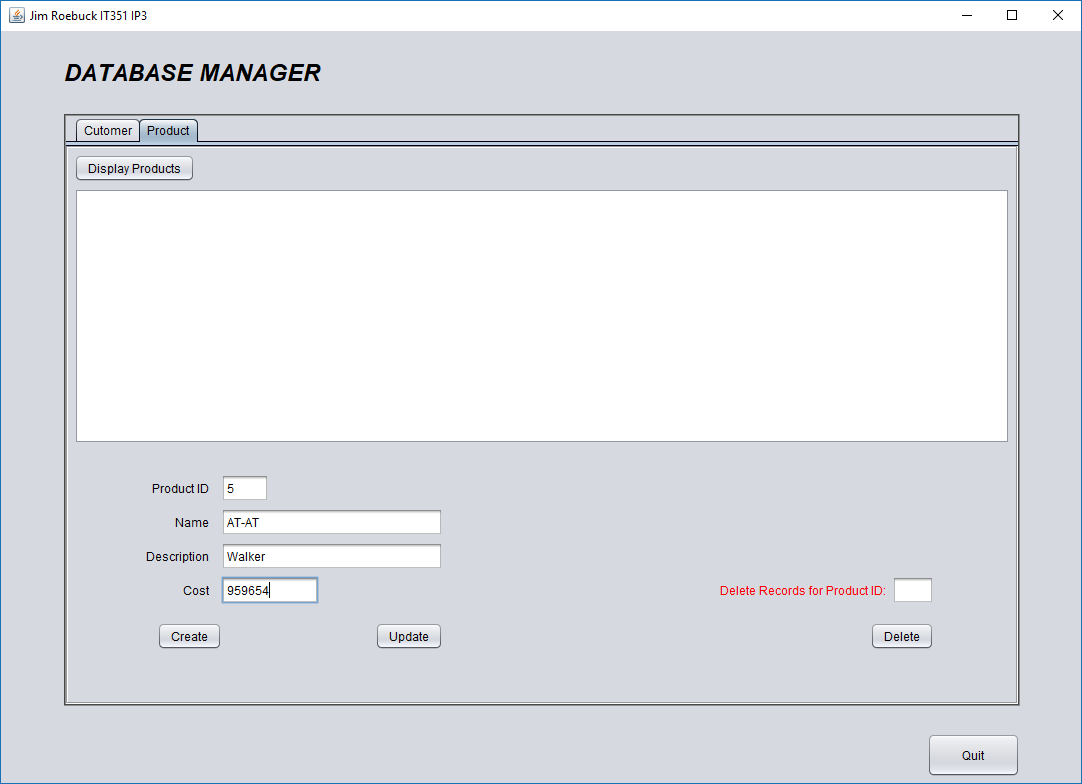
#### Customer

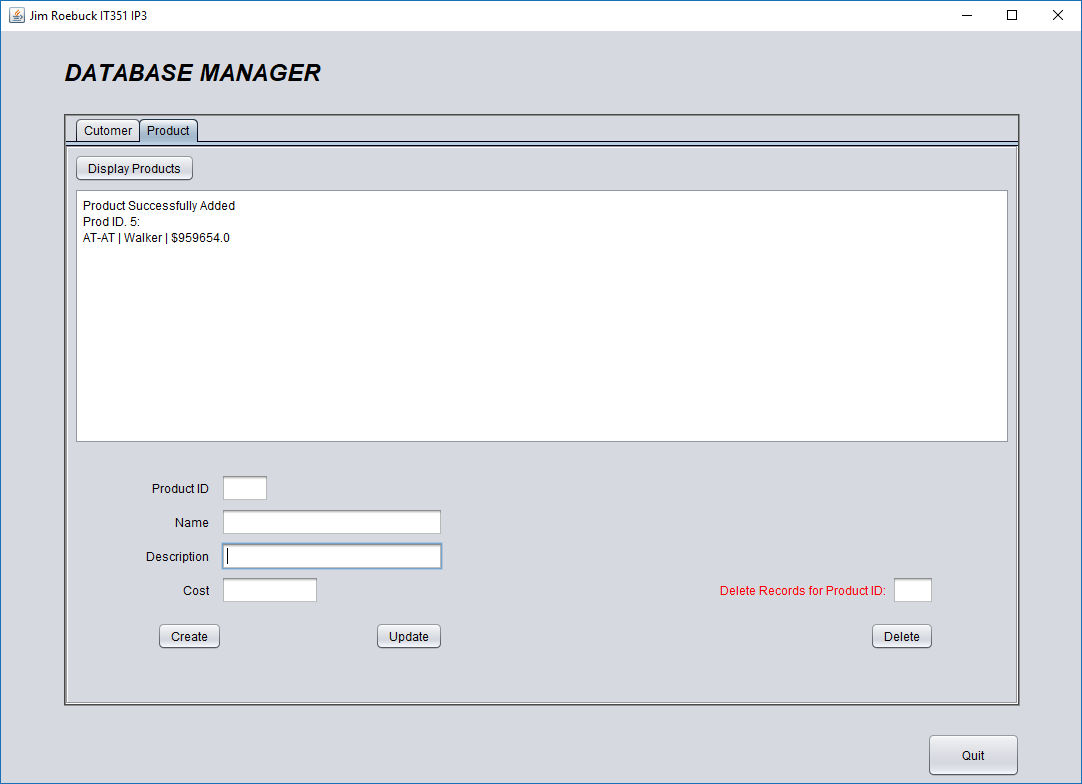


After create is clicked, input text fields are cleared and a confirmation prints to the jTextArea:



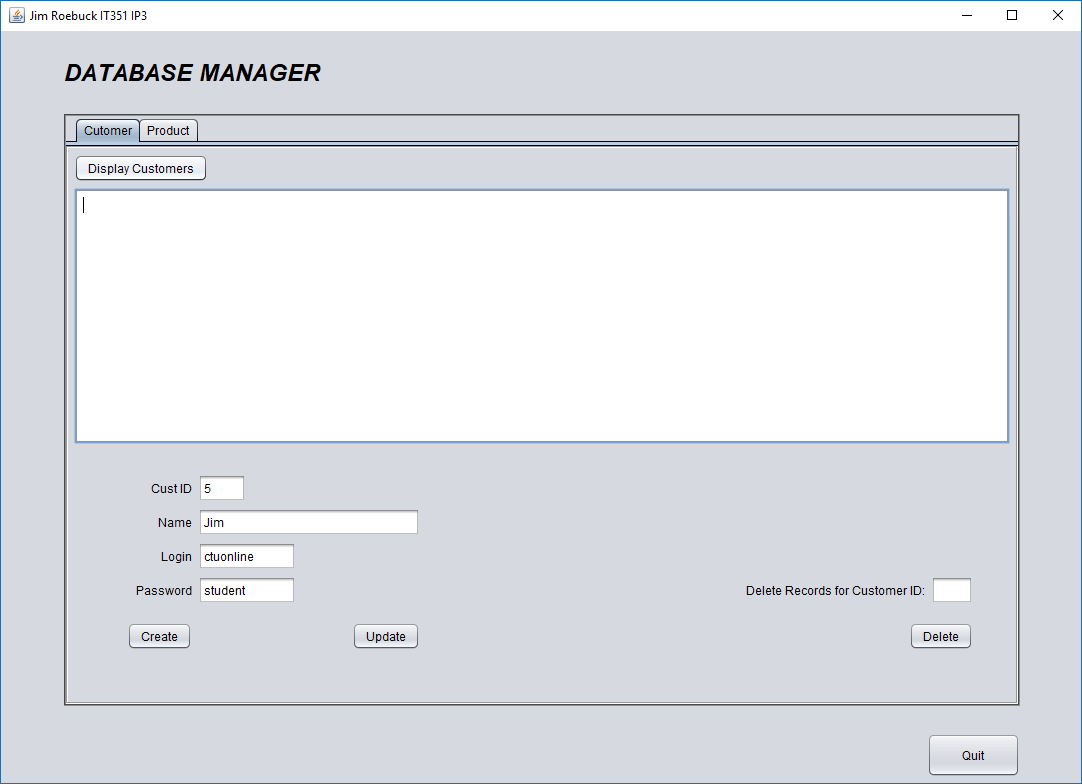
#### Create Product

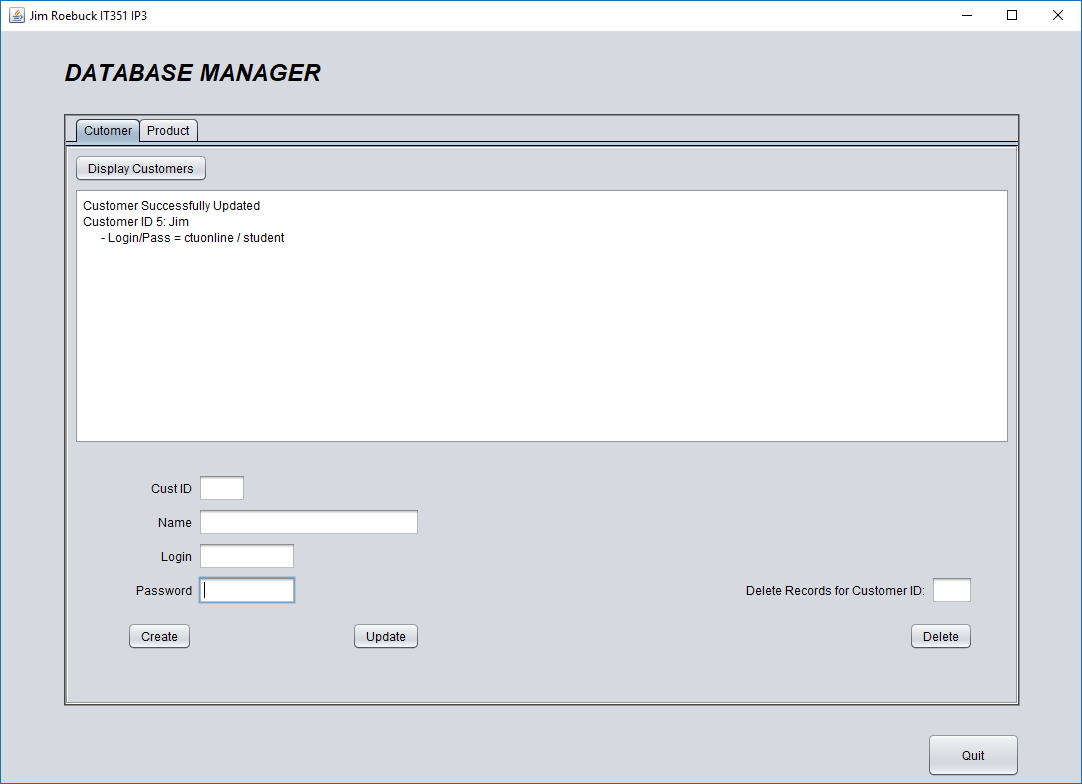




### Update Database Record

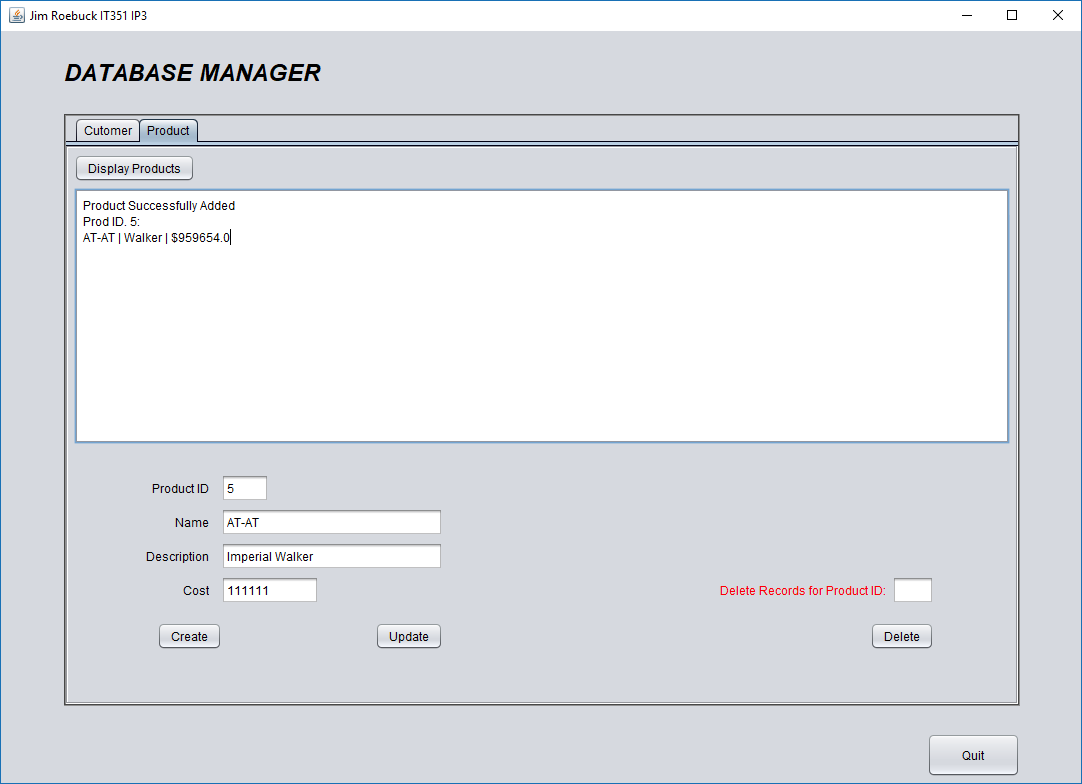
#### Customer





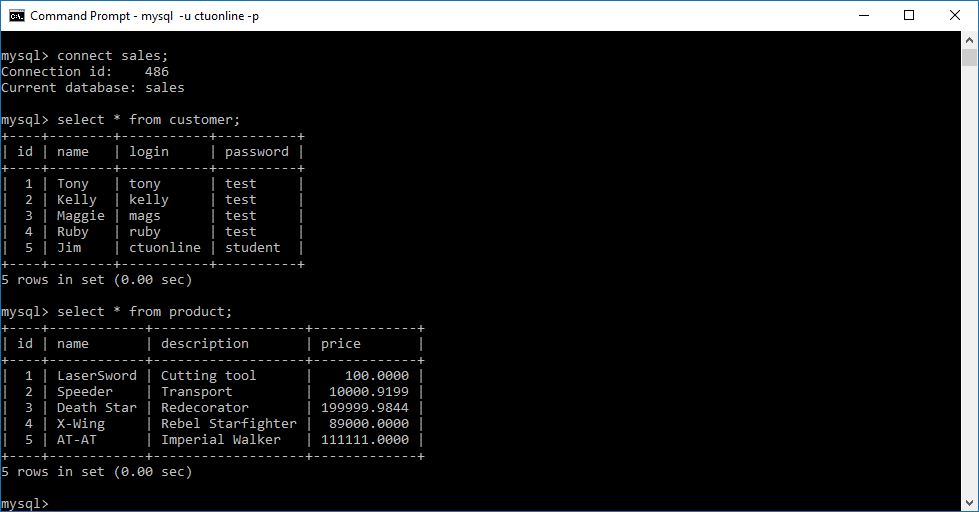
#### Update Product

#### 

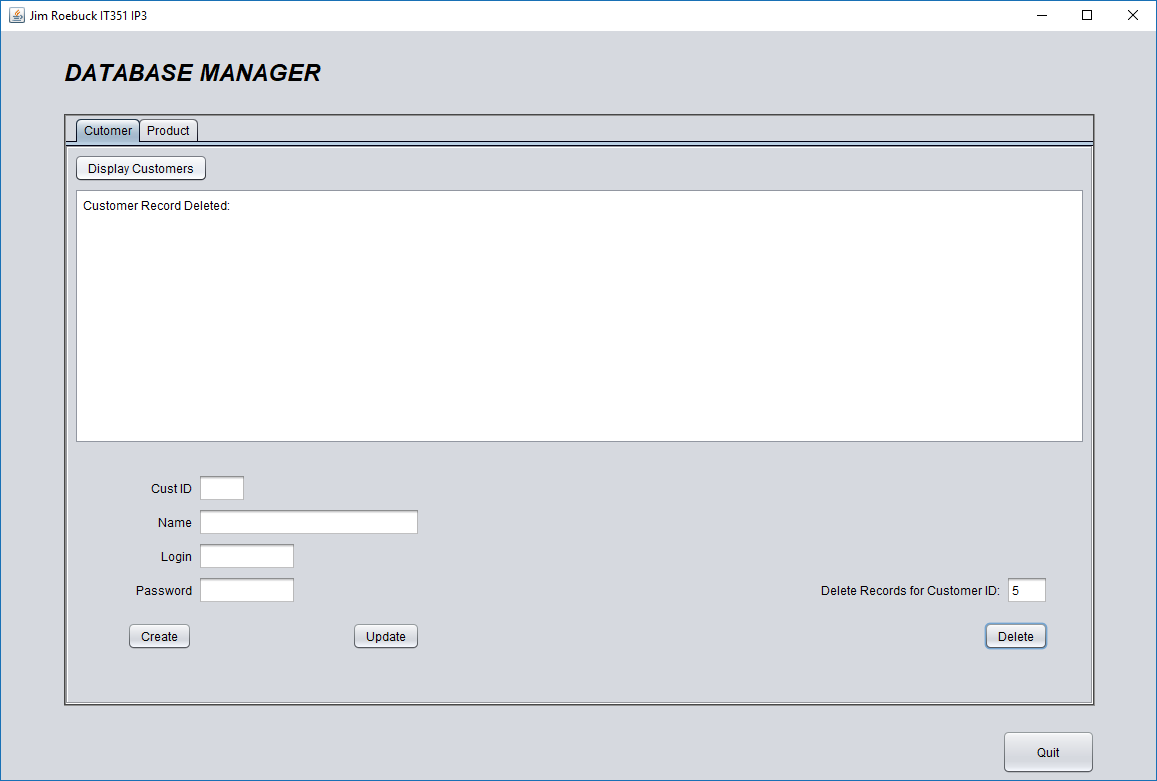


### Delete Functions

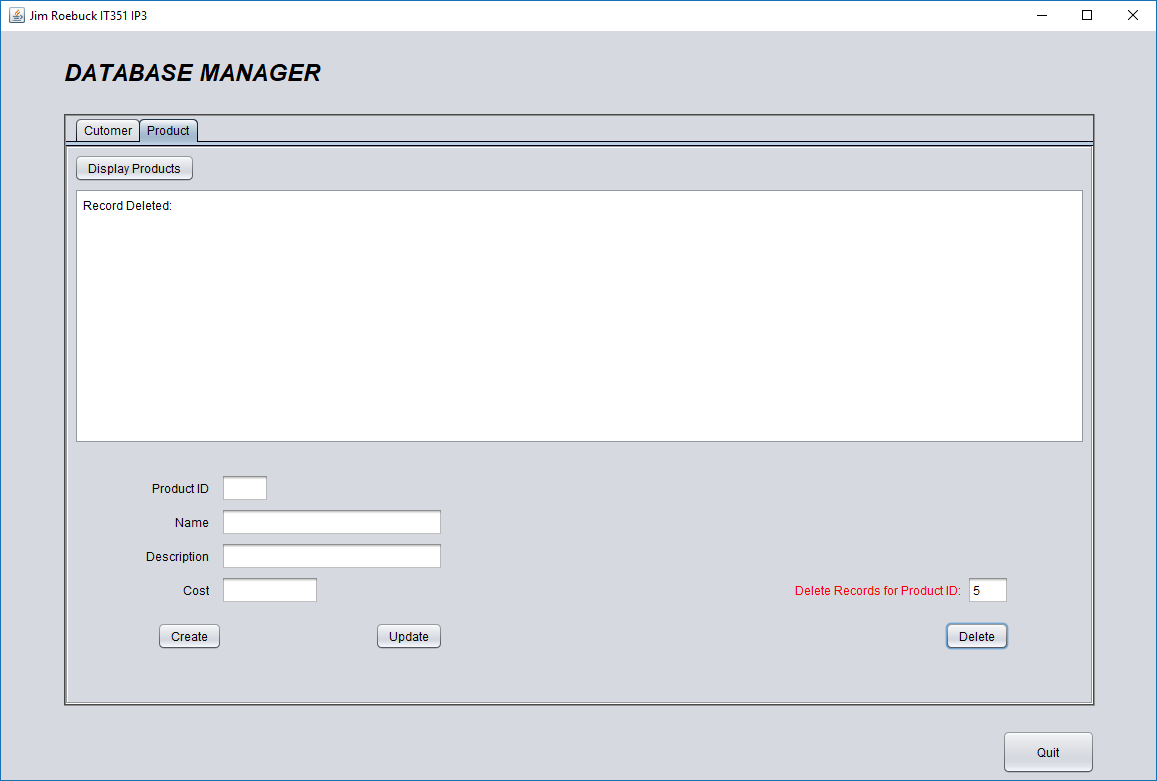
#### Databases After Table Create/Update



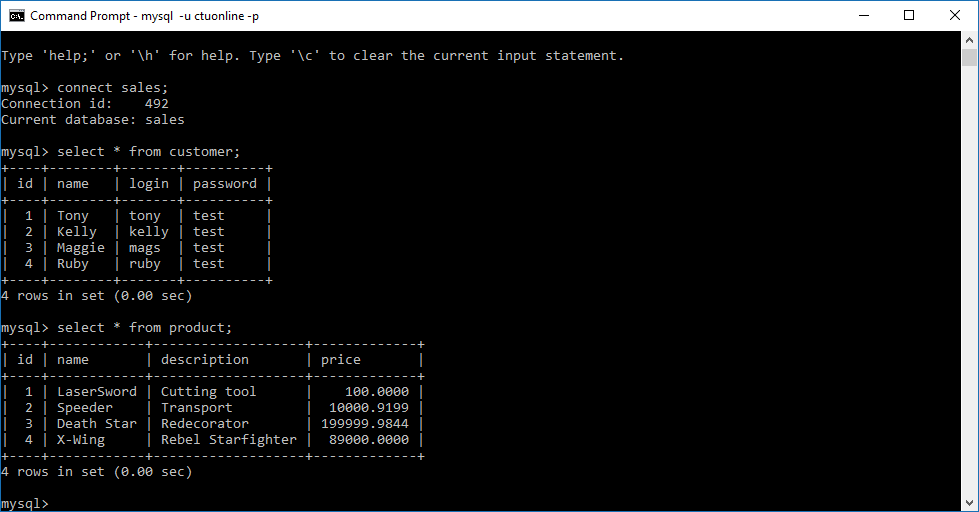
#### Delete Customer



#### Delete Product



#### Databases after table records deleted



# Unit 4 – Database-Driven Web Application

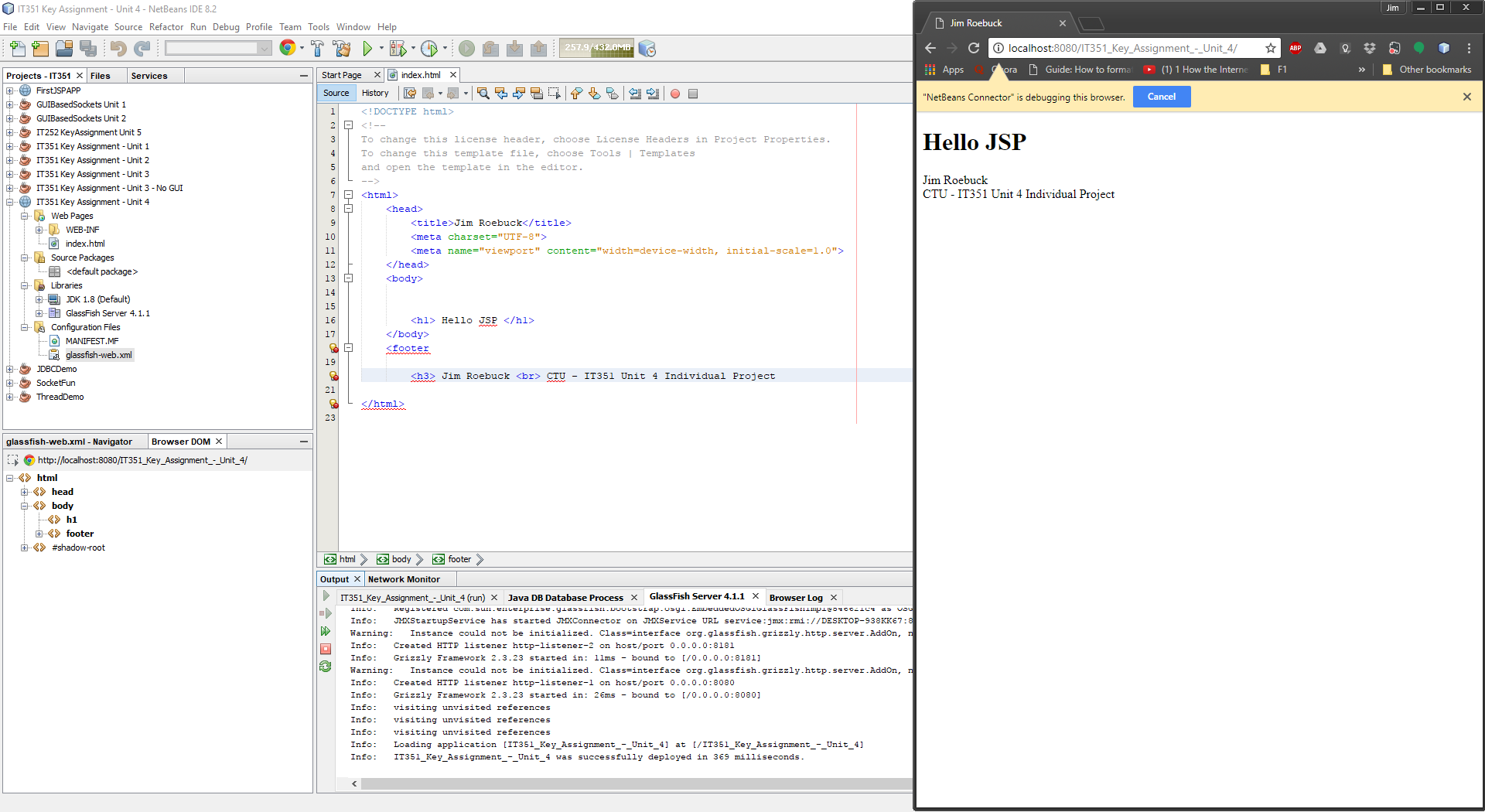
## System Architecture

The next evolution of the database management project will remove the standalone GUI and replace it with a web application. The same MySQL database will be utilized, however create, read, update and delete (CRUD) functionality will be accessed via a web browser. To set up this connectivity, the program will run as a web application using Java Web Components, including Java Server Pages with Java Server Faces to build an accessible interface. In order for the program to run successfully, the three components that must communicate are the:

* Glassfish Server – Provides platform as a web server for communication between client using web browser and database
* MySQL Database – sales database, accessed using the JDBC protocol, allows client to communicate and implement CRUD expressions on database tables.
* Web Application – Utilizing JSF and JSP, Java code is incorporated into HTML code to run application via web browser.

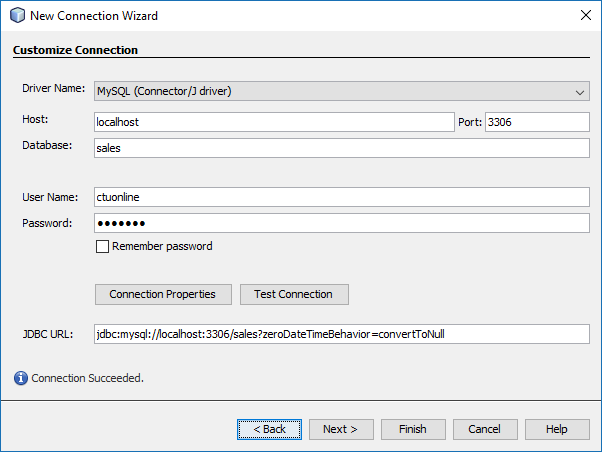
## Setup

First, a new web application is built and set up to utilize a GlassFish server, which will connect to the SQL database and allow web browsers to connect to its port connection. A default index.html is created, which shows the browser connected to the localhost at port 8080. This will provide the foundation of the web application, but the database schema, as well as the web application interface, need to be incorporated into the application.



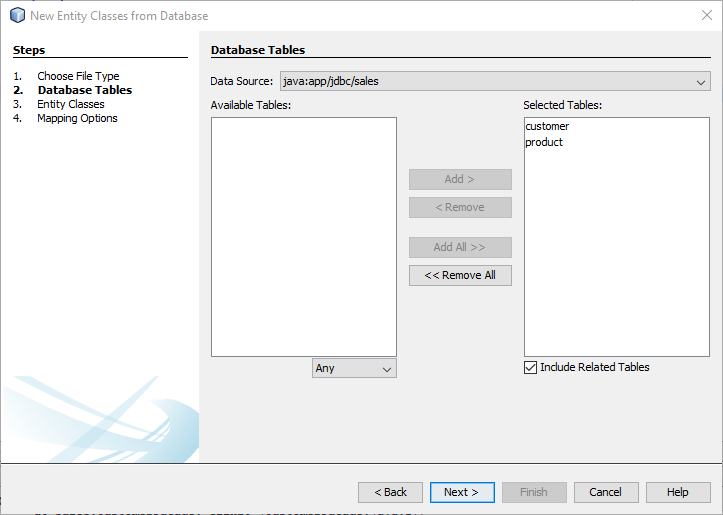
Web Application created with GlassFish server connectivity.Index.html connected to localhost port 8080.

Next, the program must add an entity class which connects to the persistent MySQL database. The application will connect to the MySQL database using the Java Database Connectivity protocol, and needs to access the sales database. Once the jdbc/sales connection is established, the same ctuonline/student username and password from previous weeks will still be able to connect to the database. The following screenshot illustrates the successful test of the connection from the application to the SQL database.

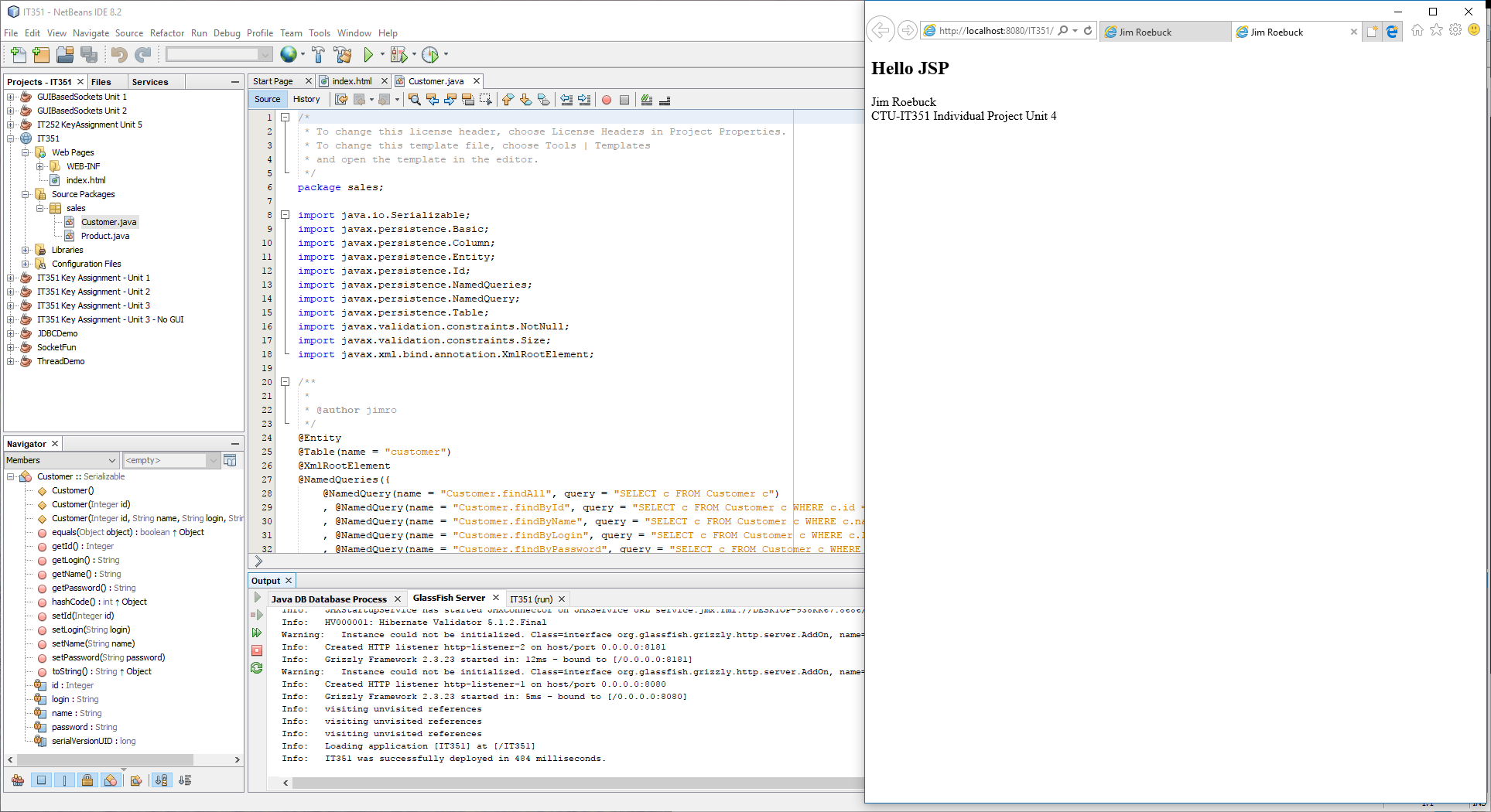


Database Persistence Entity Class connection created with a successful test connection to server. This allows the JSP connection to the MySQL sales database, which will permit web-app interactivity with the local database.

Once the connection has been made to the database, the customer and product tables are imported into our application. Once this data source is connected to the application, classes for both customer and product objects are added to the program. These classes contain the same attributes as the objects in previous weeks, as the table schema was set up to use the same names. This consistency shows in these classes, where attributes, as well as getter() and setter() methods are created for object manipulation. Though this ties the data source into the program, as the screen shot shows below, the program still runs index.html to the web browser, as the Java Server Face (JSF) page hasn’t been incorporated into the program.

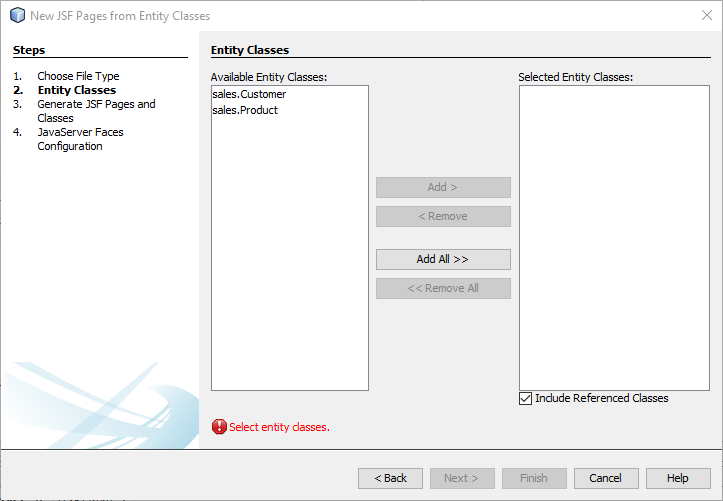
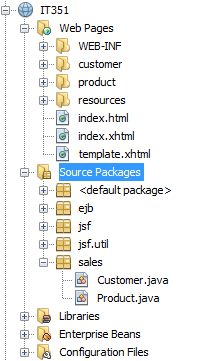


Successfully connected to MySQL database, adding customer and product tables to my app.



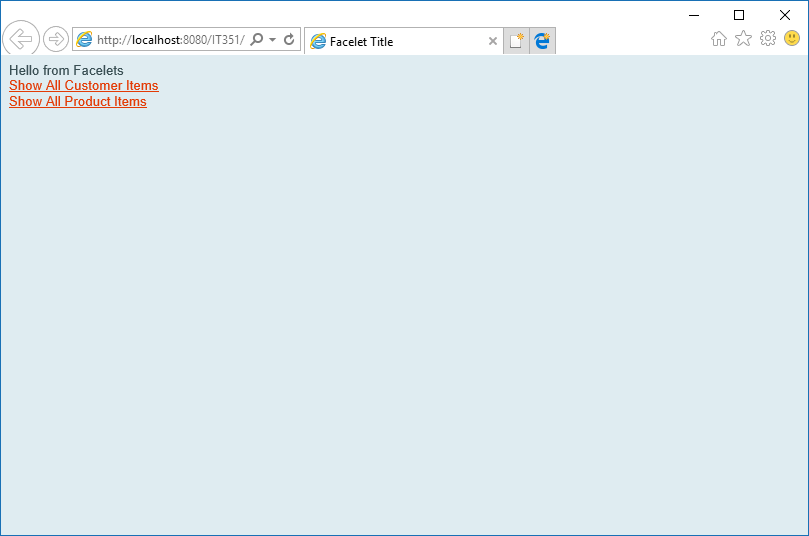
Customer and Product tables imported as classes into my project

Once data structure is implemented, the java server pages can be created based off of the database customer and product tables. Utilizing the imbedded JSF framework, much of the interface and application structure are handled for us through the framework itself. When the new JSF page is created from the entity class just established, the database tables are automatically located. These tables, as well as the object classes derived from them, are added into our program. Once configured, the web application will launch after the GlassFish server is connected.

Adding JSF entity classes

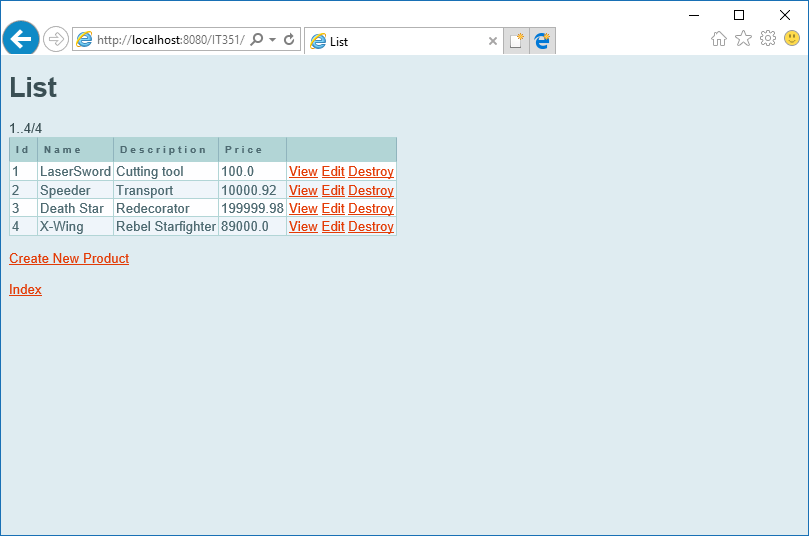
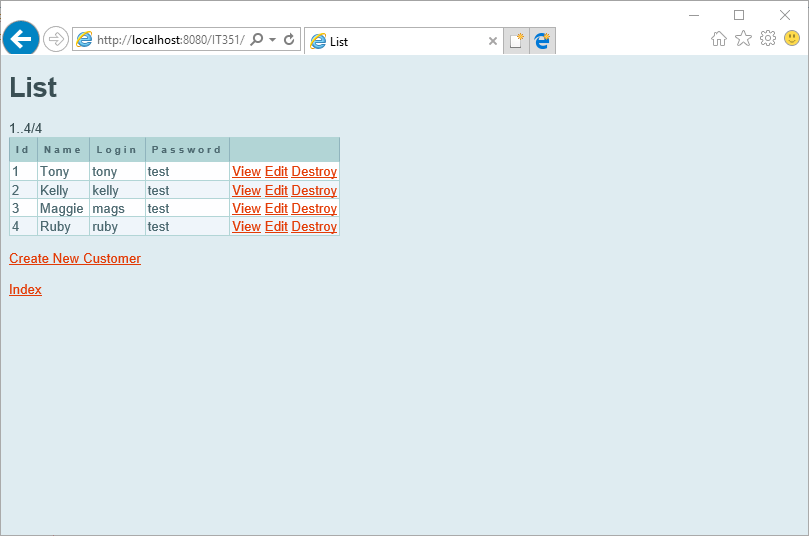
After the JSF page is finished, running the program no longer brings up the simple index.html page. Instead, the browser points to the server connection at the application <http://localhost:8080/IT351/>. With the server connected and the project running, this link will launch the program and allow for database management.



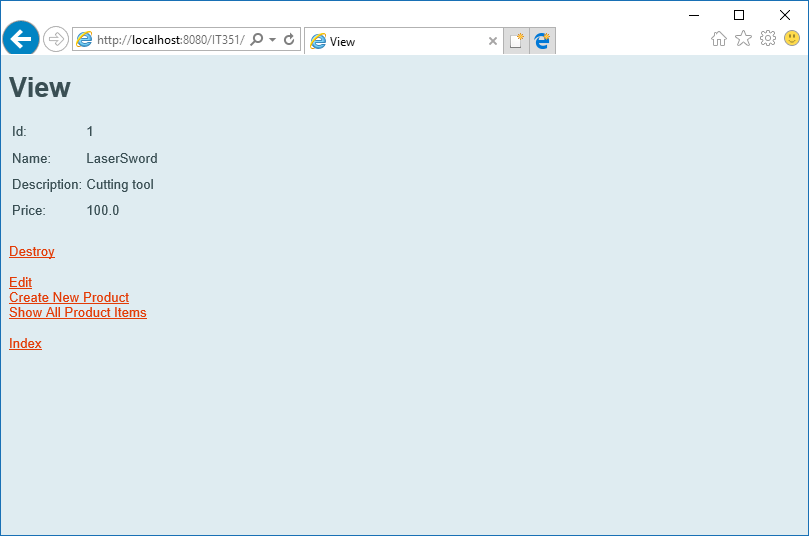
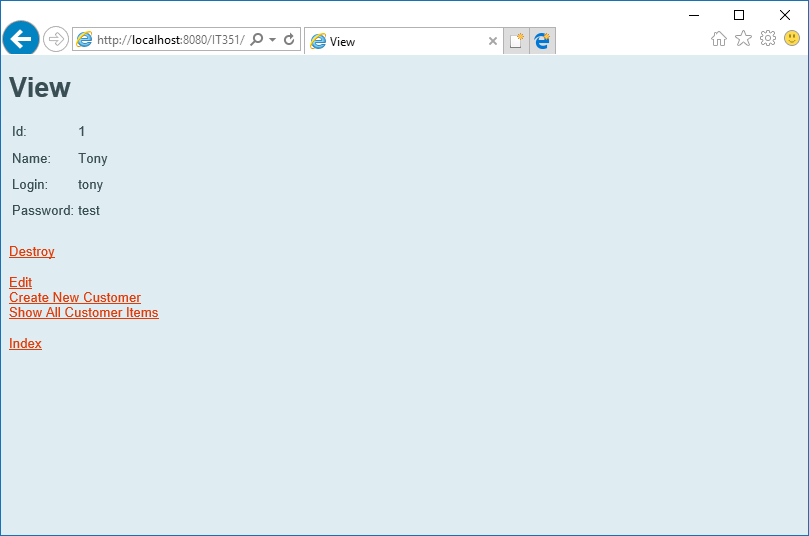
Successful deployment of web application

## Deployment and Use of Web Application

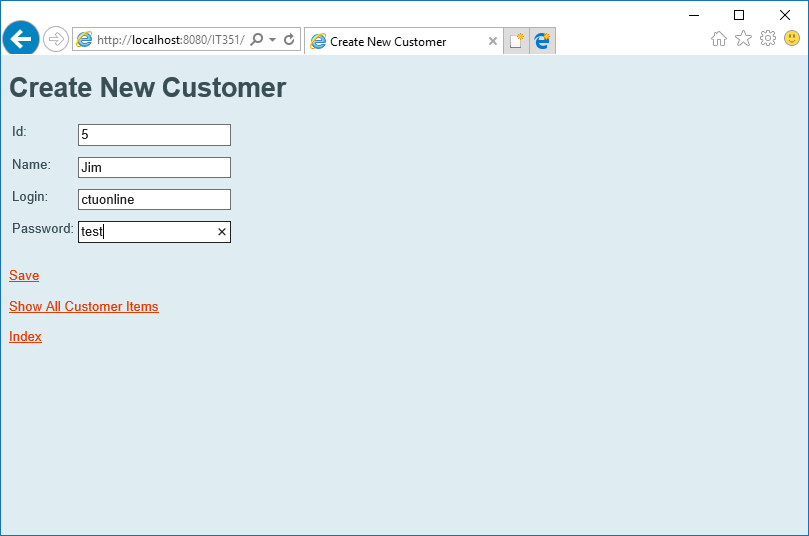
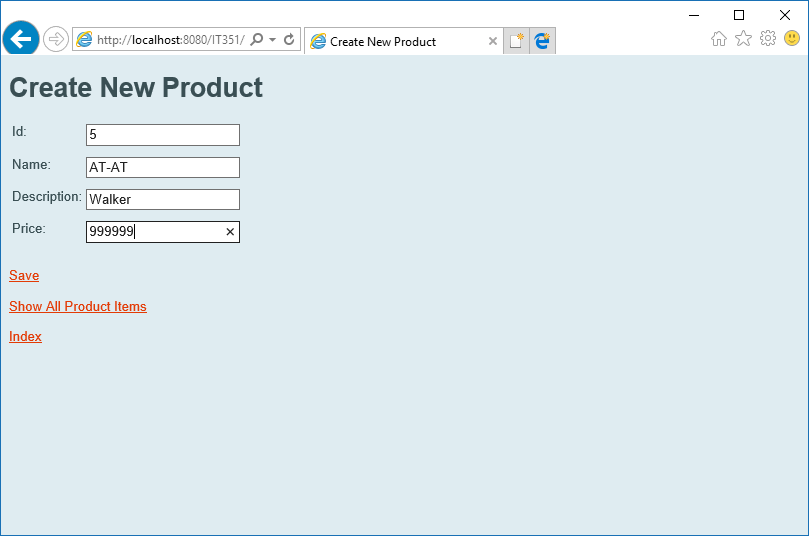
From here, the same CRUD functions we’ve been using to view and manipulate our database are available. Users can view tables, create new records, or update and delete existing records. Anything done in the application directly effects the MySQL database, as the following screenshots illustrate. A full user guide walking through the steps of running and accessing the application are included in another file in this folder.



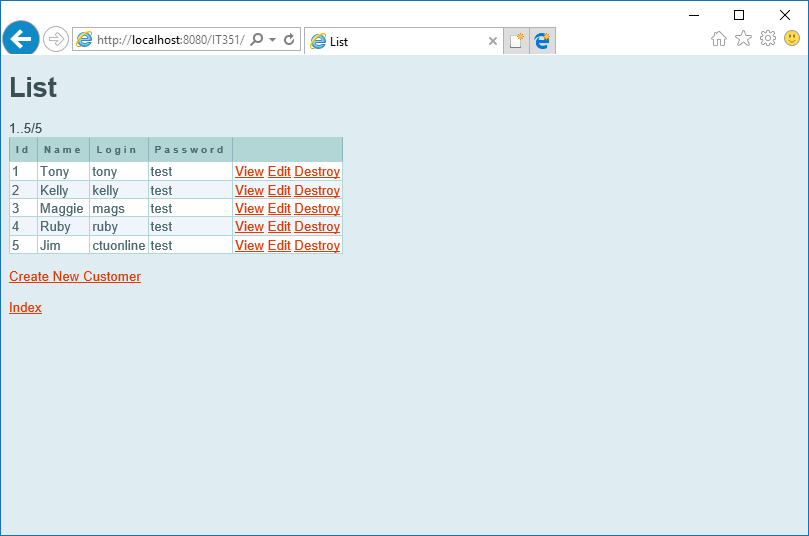
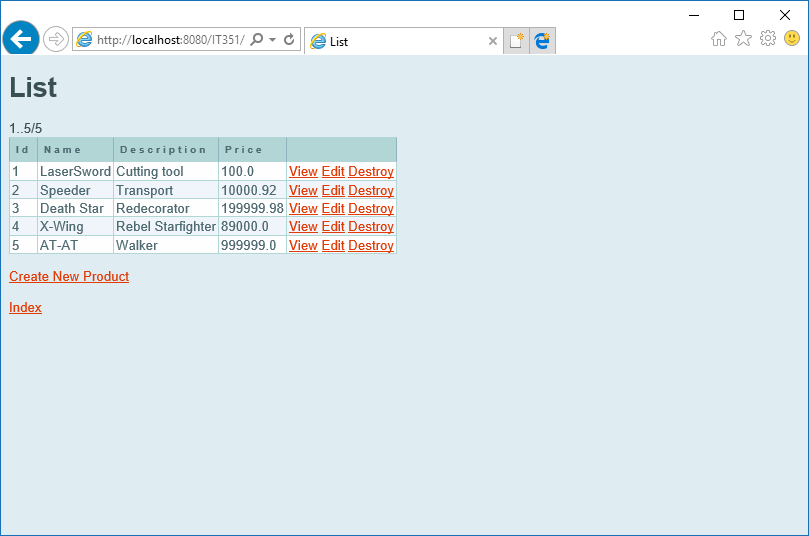
Show all Customer/Product object



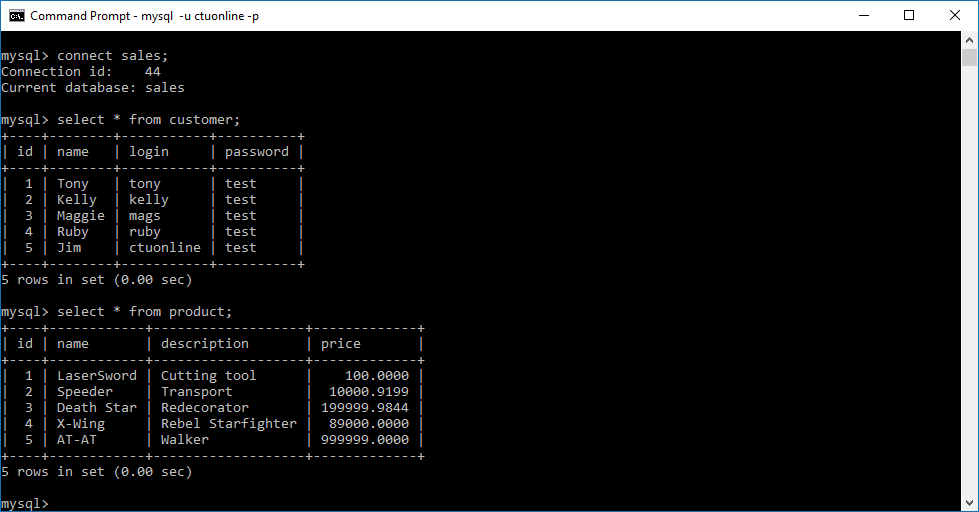
View specific customer/product object

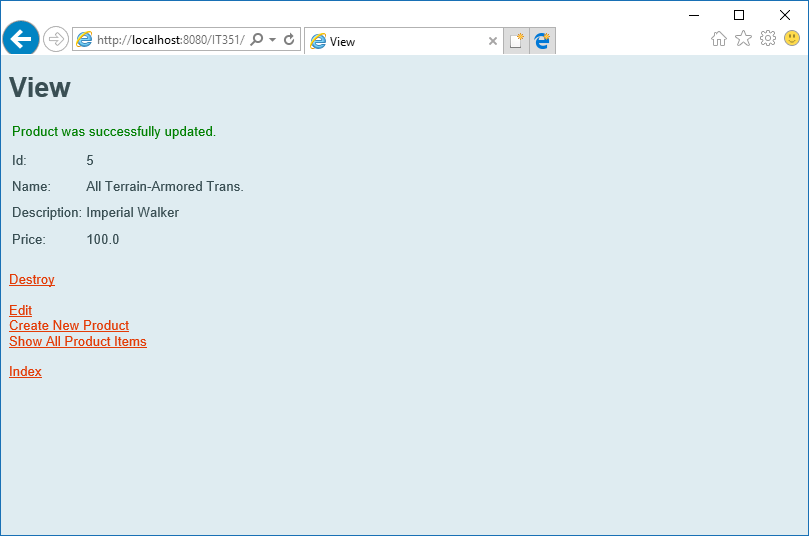
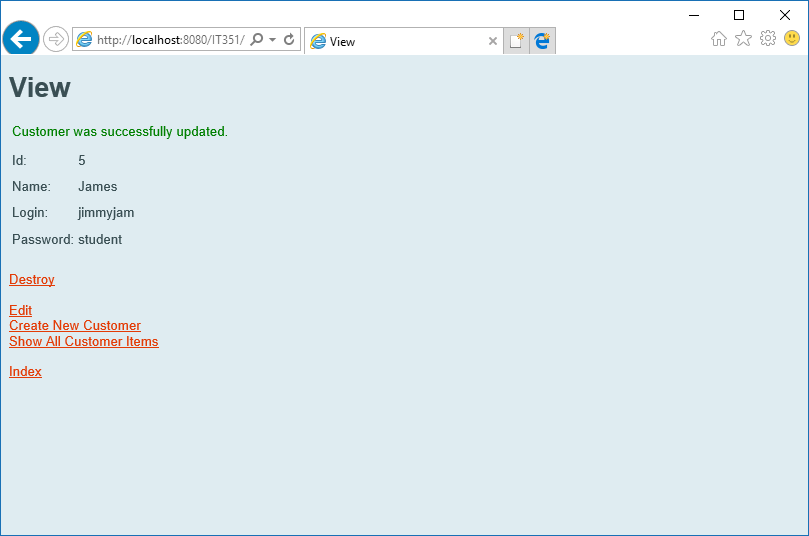
Create new customer/product objects

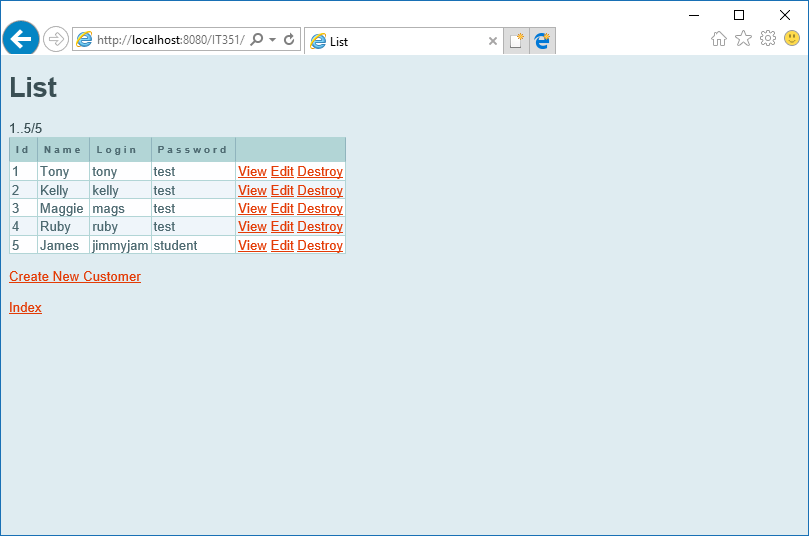
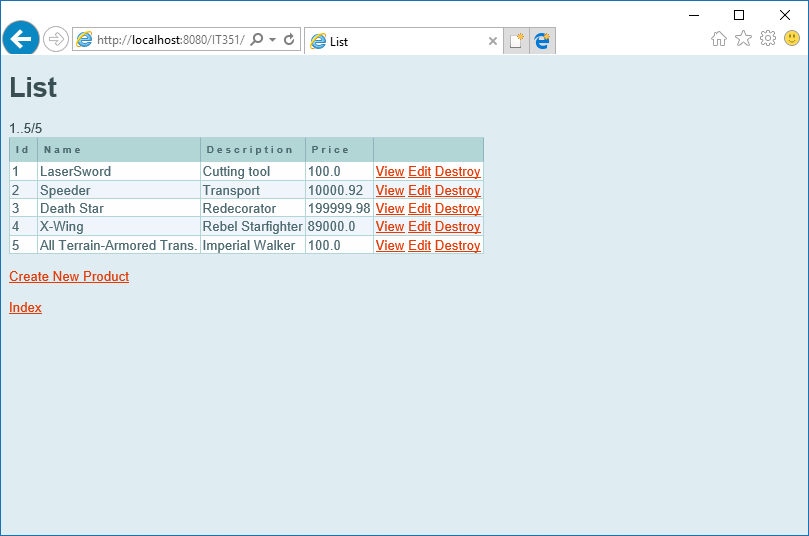
View Customer/Product Lists after entries added

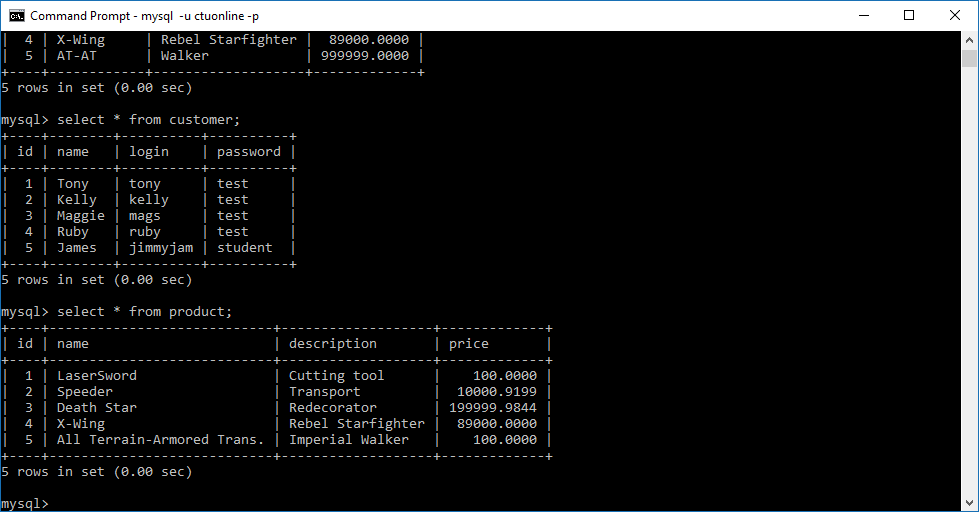


MySQL command-line view of tables after successfully added

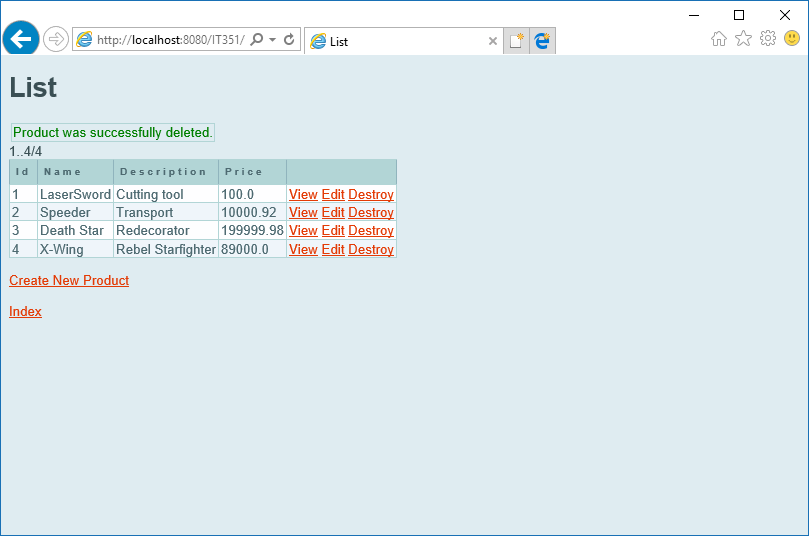
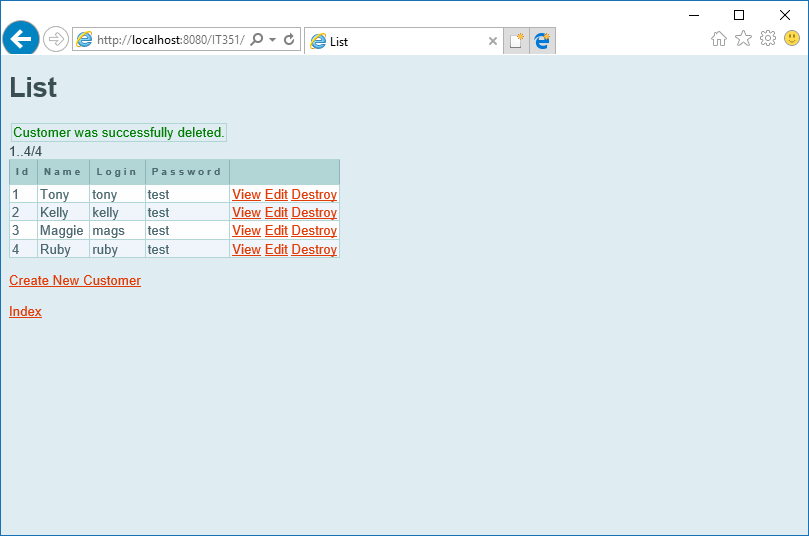


Updated customer/product records

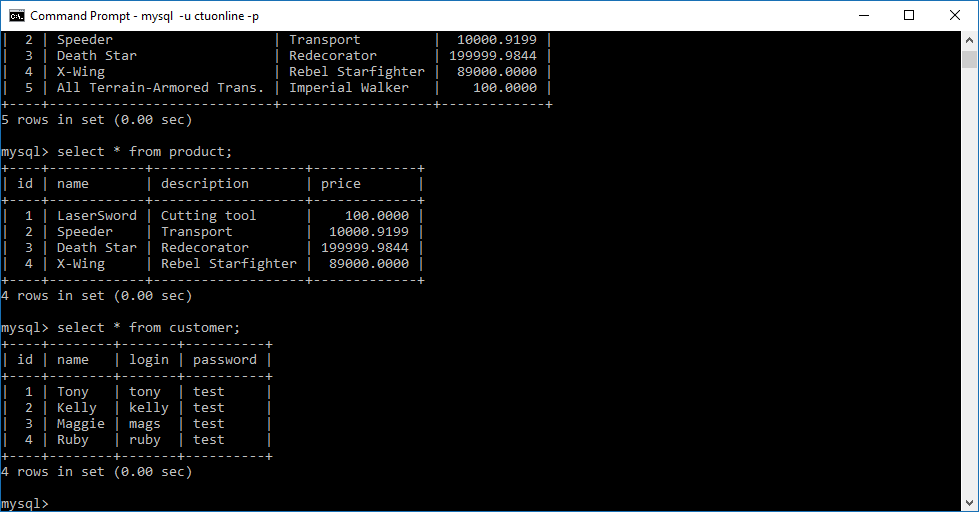
 



Database displayed to web app and command-line after records updated



Items successfully deleted from database



Database after items deleted

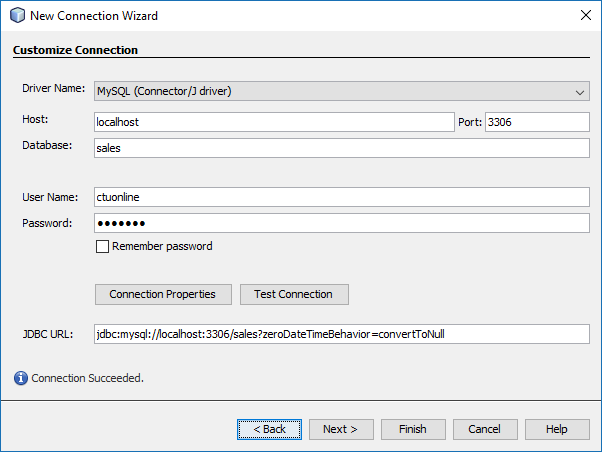
# Unit 5 – User Manual and Testing

## User Manual and Installation Guide

Setup

To begin, please import the attached project folder(IT351) into NetBeans. A full walkthrough of the implementation of the application can be found in the Key Assignment document found in this compressed folder. The program is setup to continue with the naming conventions and database parameters used in previous weeks, using the following configuration for MySQL connectivity. Though these should import along with the program, the application will connect to MySQL with the following values.

* Hostname – localhost
* Port – 3306
* Username – ctuonline
* Password – student



Loading MySQL and GlassFish Server

Execution of the web application assumes the MySQL database has been populated with the sales database, which includes the customer and product tables. If these aren’t loaded, the following scripts will populate the sales database and create the foundation for the program (Lowe, 2017).

#### SQL Script – User Setup

CREATE USER 'ctuonline'@'localhost' IDENTIFIED BY 'student';

grant select, insert, update, delete on sales.\* to 'ctuonline'@'localhost';

#### SQL Script – Sales Database Setup

DROP DATABASE IF EXISTS sales;

CREATE DATABASE sales;

USE sales;

DROP TABLE IF EXISTS `customer`;

CREATE TABLE `customer` (

`id` int(10) NOT NULL,

`name` varchar(30) NOT NULL,

`login` varchar(10) NOT NULL,

`password` varchar(20) NOT NULL,

PRIMARY KEY(id)

);

insert into customer values (1, 'Tony', 'tony', 'test');

insert into customer values (2, 'Kelly', 'kelly', 'test');

insert into customer values (3, 'Maggie', 'mags', 'test');

insert into customer values (4, 'Ruby', 'ruby', 'test');

DROP TABLE IF EXISTS `product`;

CREATE TABLE `product` (

`id` int(10) NOT NULL,

`name` varchar(30) NOT NULL,

`description` varchar(50) NOT NULL,

`price` FLOAT(10,4) NOT NULL,

PRIMARY KEY(id)

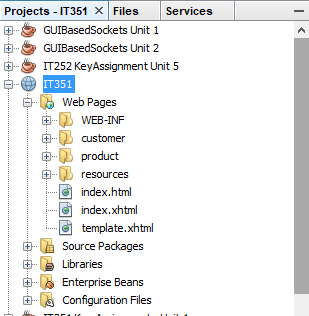
);

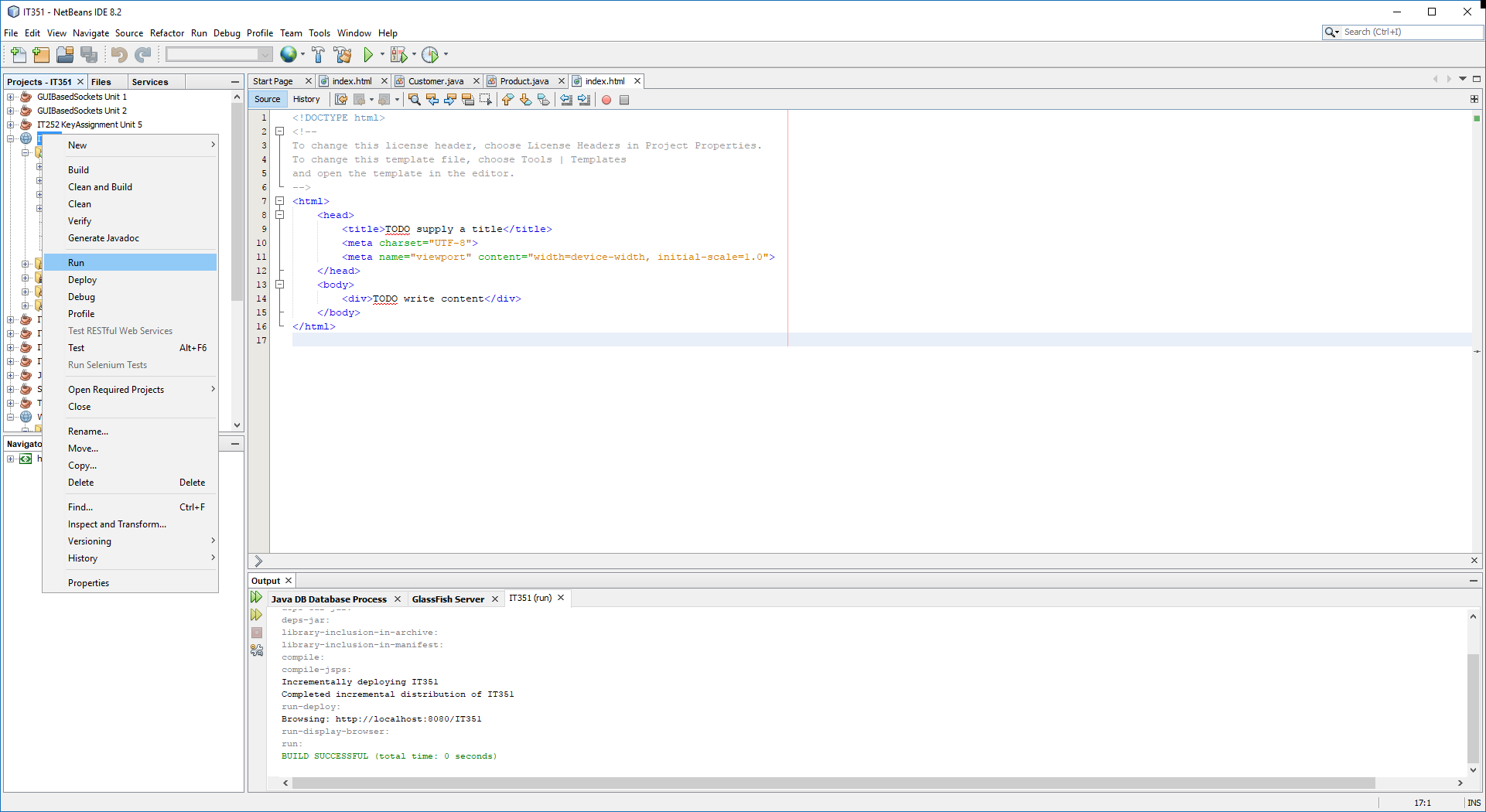
insert into product values (1, 'Lightsaber', 'Cutting tool', 100.99);

insert into product values (2, 'Speeder', 'Transport', 10000.92);

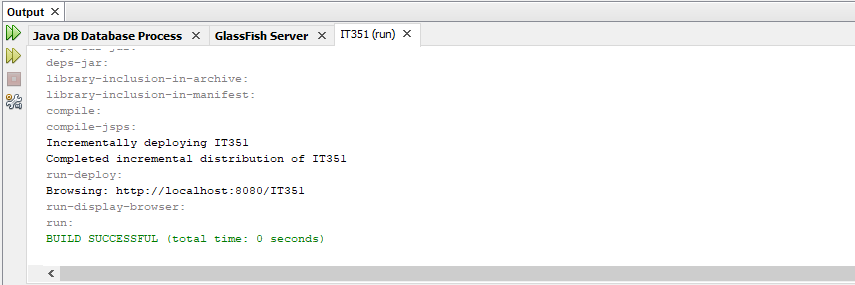
insert into product values (3, 'Death Star', 'Redeco

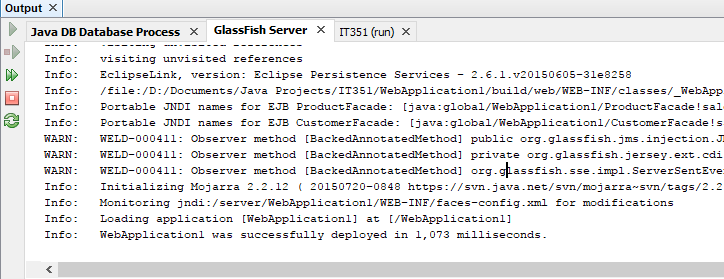
With the MySQL connected and populated with the sales database schema, run the IT351 project to start and run the GlassFish server. Right click on the IT351 project in the project explorer, and click “run”, as illustrated below.





In the output console, you should see that the IT351 project successfully ran. Additionally, the GlassFish Server tab should post that the application was successfully deployed





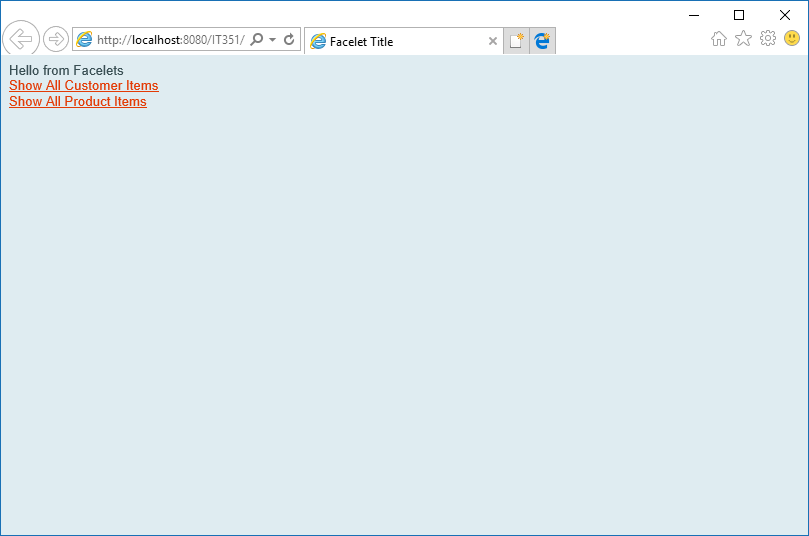
Deployment and Use of Web Application

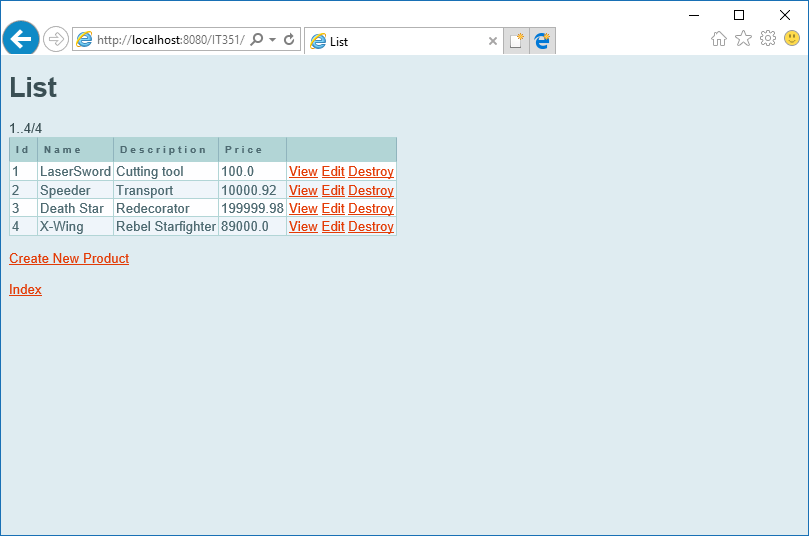
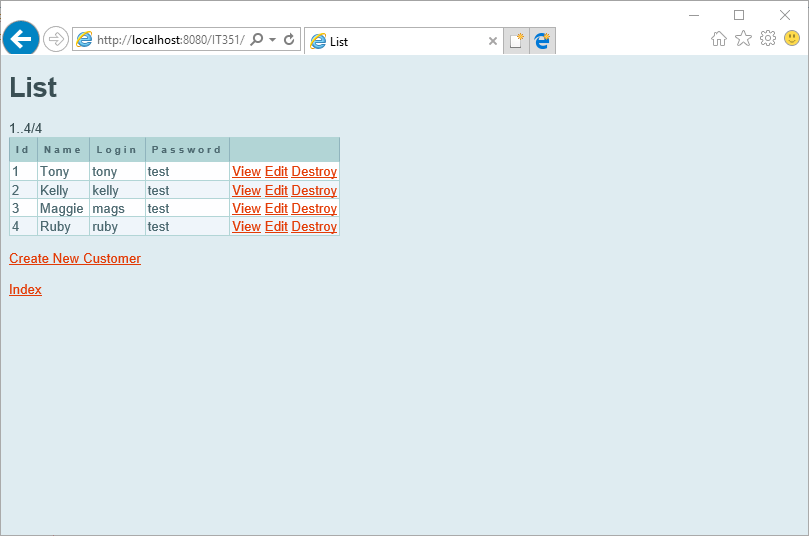
With the GlassFish Server running and connected to a deployed SQL database, the web application will provide create, read, edit, and delete (CRUD) functions on the sales database. The web application should load automatically when the IT351 project is run to start the server. However, if the window is closed, or the application needs to be accessed while the server is already up, the following URL can be entered into your web browser:

<http://localhost:8080/IT351/>

Application Functionality

The application loads with the option to view either Customer or Product tables. By clicking either of the two links, a list containing all table elements will load, as illustrated below:



Successful first deployment of Web Application

Show list of all items in Customer/Product tables

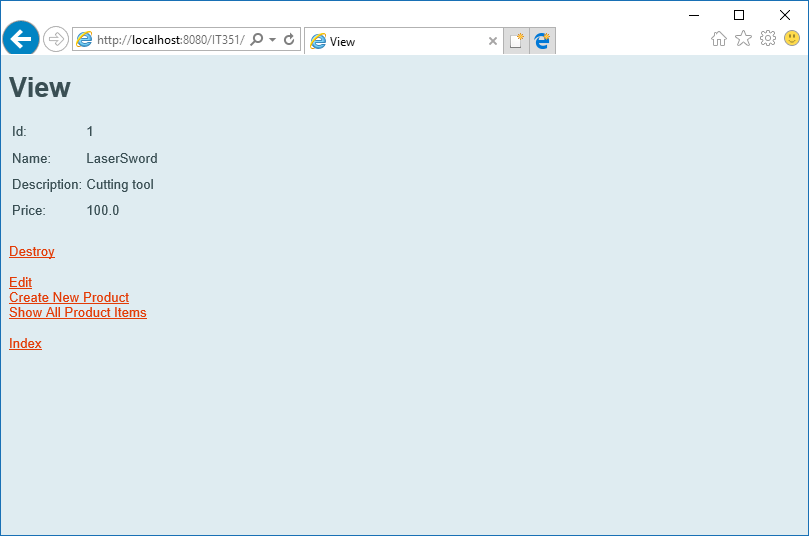
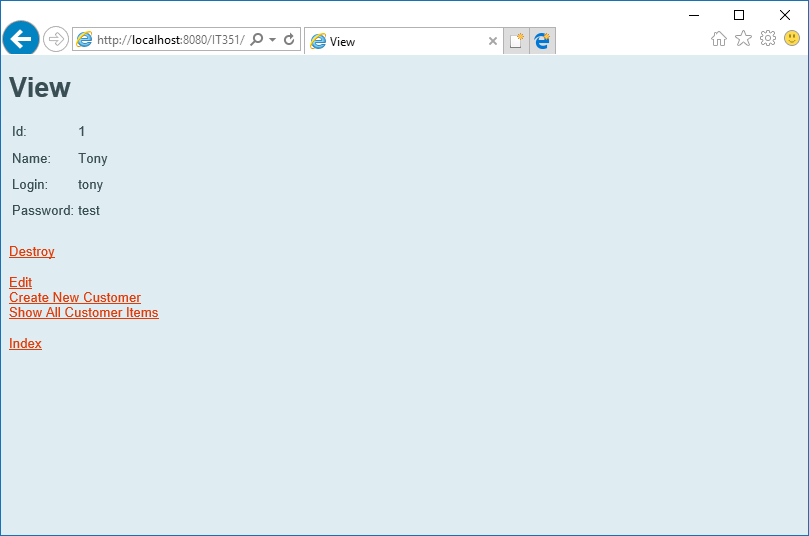
As depicted, once the table records are displayed they have several options displayed. Links on the webpage a user to perform the following functions:

* *Index*:Return to the main menu, allowing user to choose other tables in database.
* *Create New*: Add an additional object to table.
* *View*: Review details of specific object.
* *Edit*: Change values of an existing record.
* *Destroy*: Delete record from database.

CRUD Functions

View –

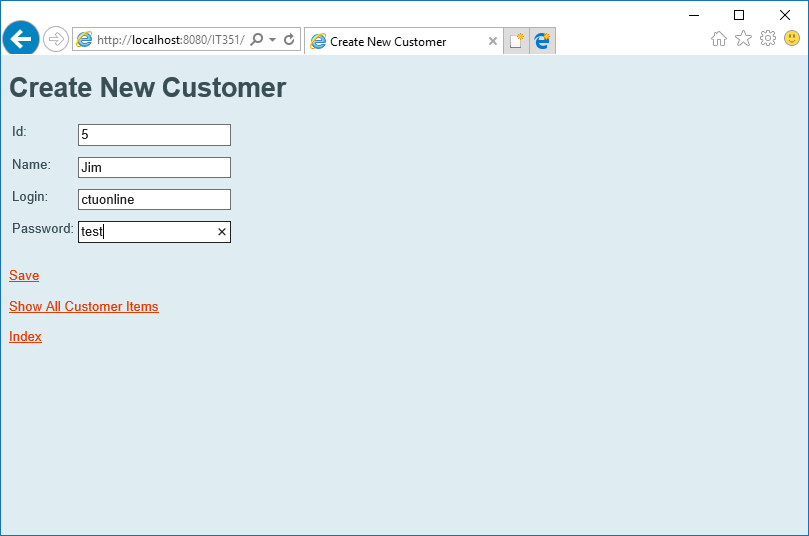
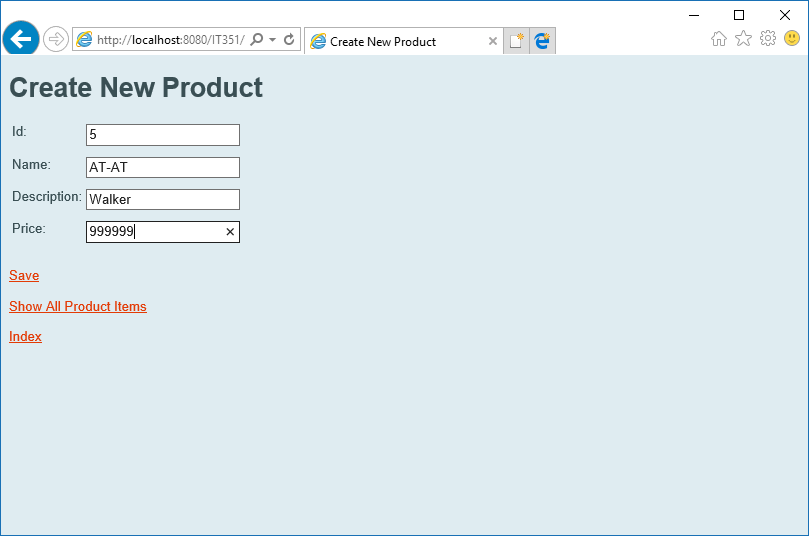
By clicking view, details of the table record are displayed, along with additional options to manage the data



View specific customer/product object

Create New –

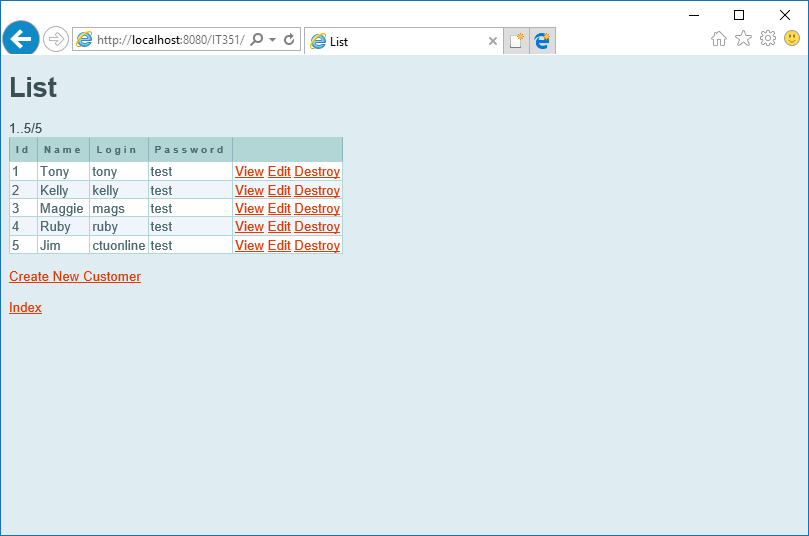
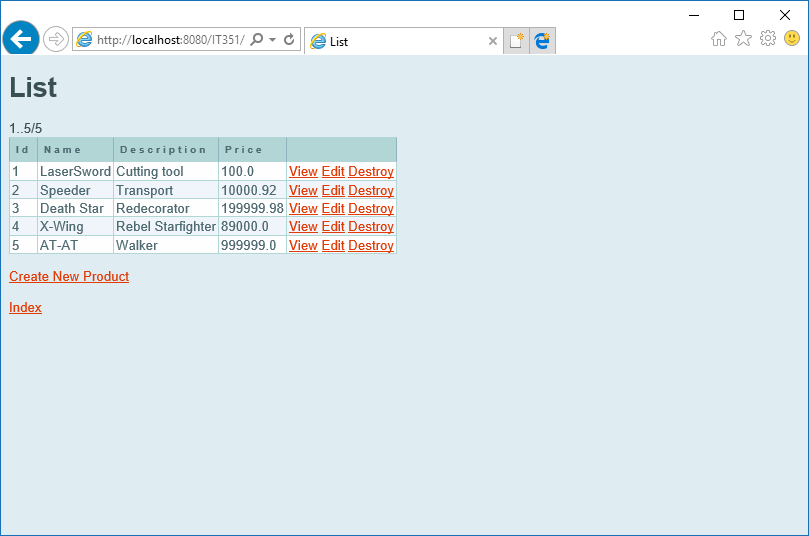
Clicking the option to create a new record displays a page with text fields for the object value input. Fill out all of the text fields for the item, then click save to add the record to the database

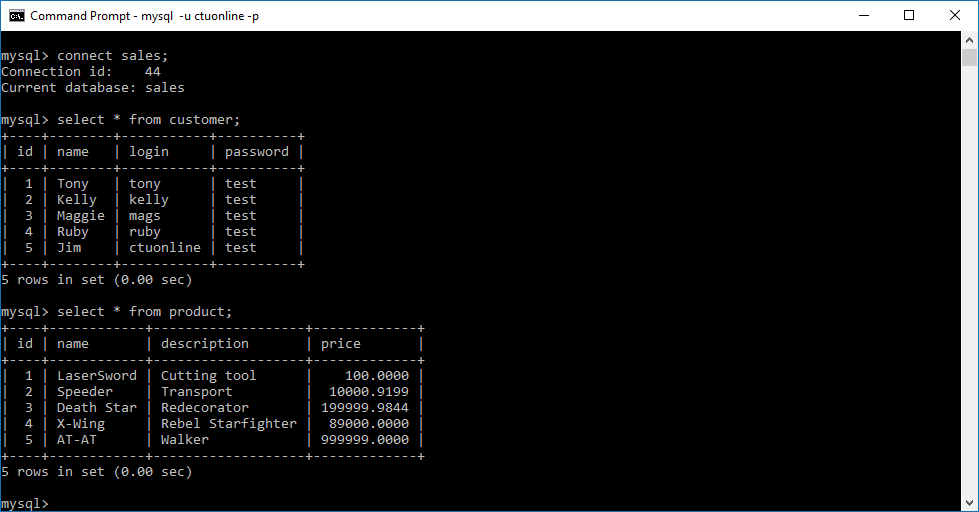
Create new customer/product objects

Create New – Output

With the record saved, the list displays to the web browser with the added item. With the item added, we can test and verify the data was correctly added by accessing the database directly through MySQL.

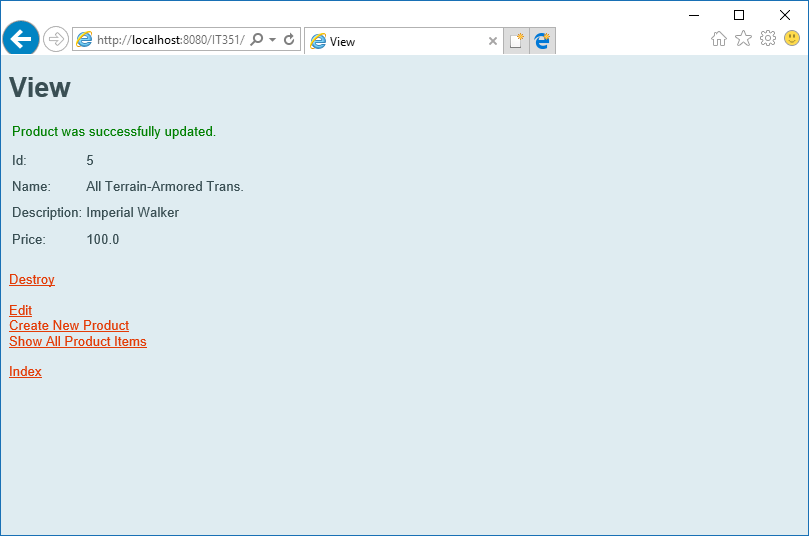
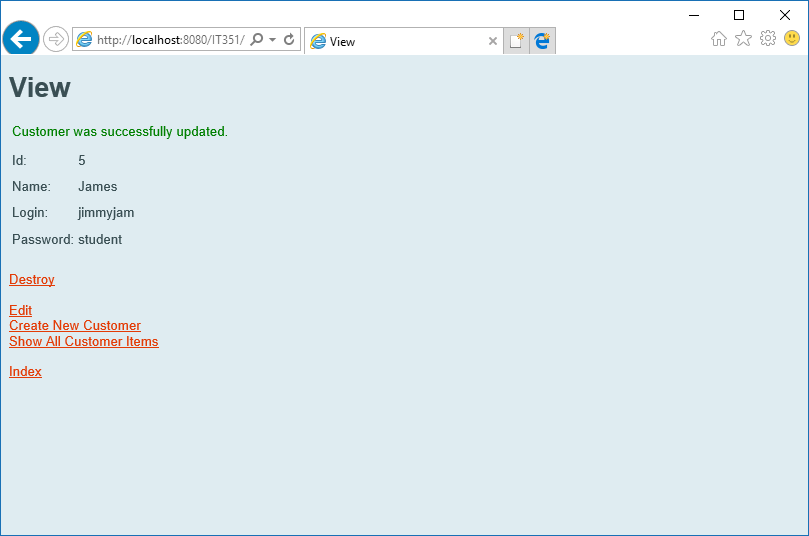
View Customer/Product Lists after entries added



MySQL command-line view of tables after successfully added

Update

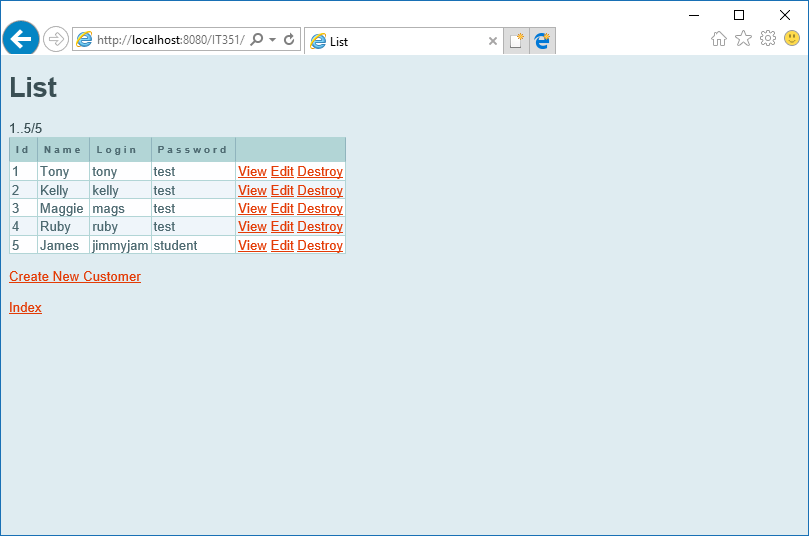
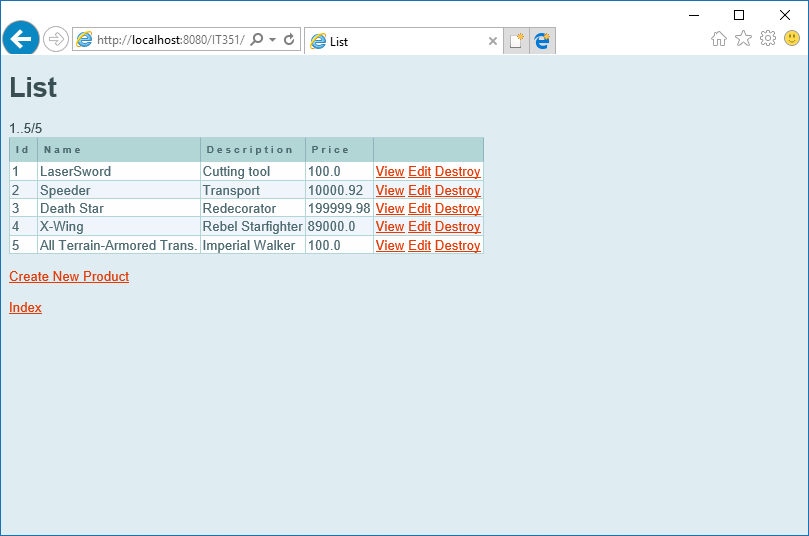
The same process is utilized to update an existing database record. With the customer or product table displayed, click “edit” next to the item needing modification. Fill out all text fields with the updated information, then click “save”. The following output will display to the web browser.

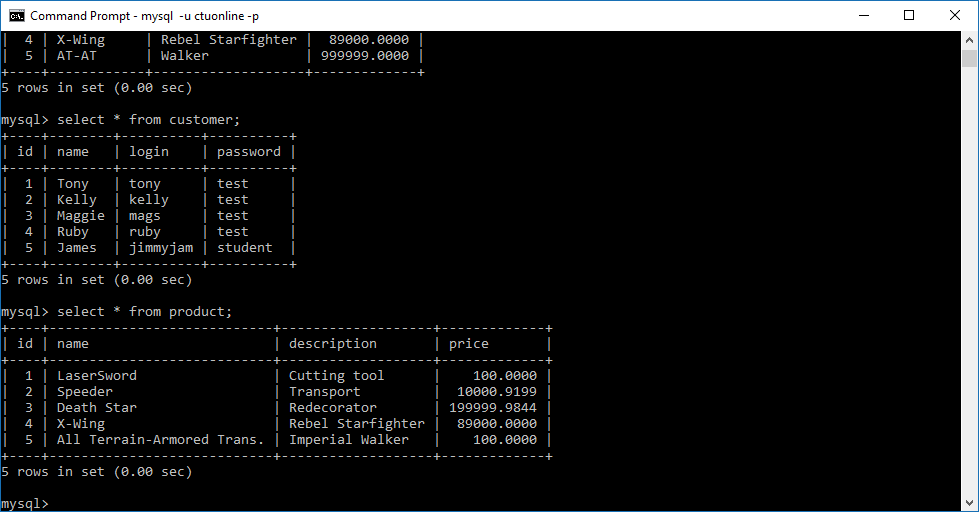


Updated customer/product records

Update- Output

With the data updated, both the web browser and command-line SQL connection will demonstrate the updated records. Click “show all” to review updated table in the web browser, or use the command select \* from *tablename;* to query the table records.

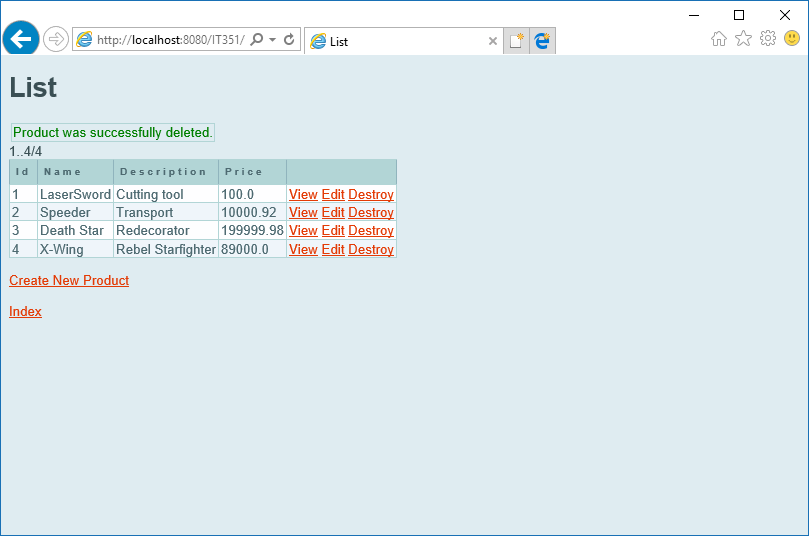
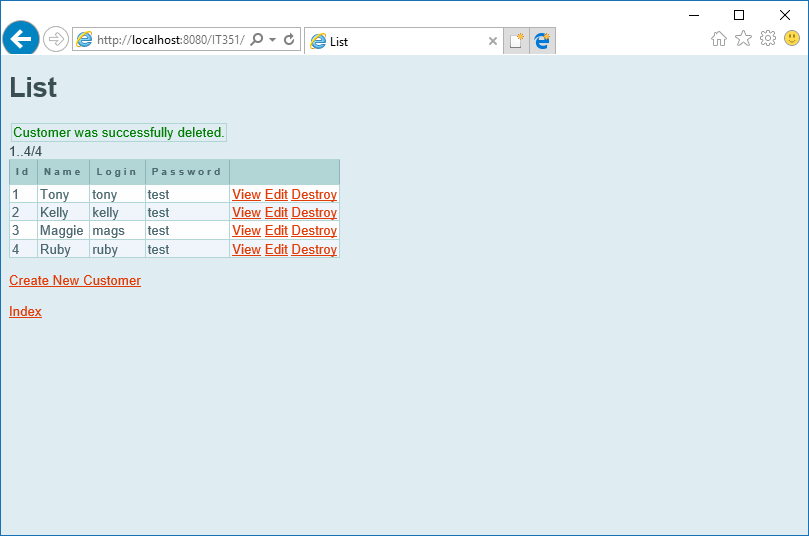
 



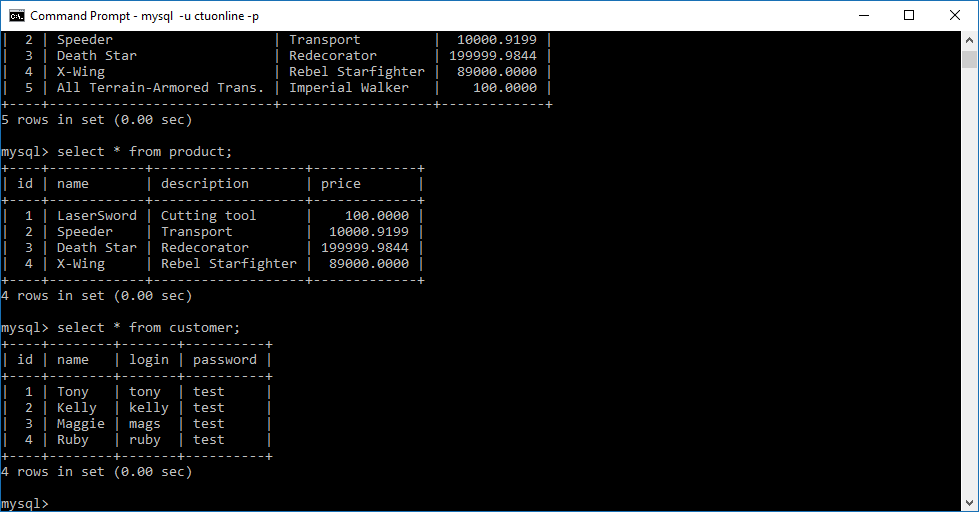
Database displayed to web app and command-line after records updated

Delete

Deleting records is a similarly easy process. With the table displayed, simply click “destroy” next to the item that you wish to remove. Verification of the record deletion is the same as previous steps, as illustrated below:



Items successfully deleted from database



Database after items deleted

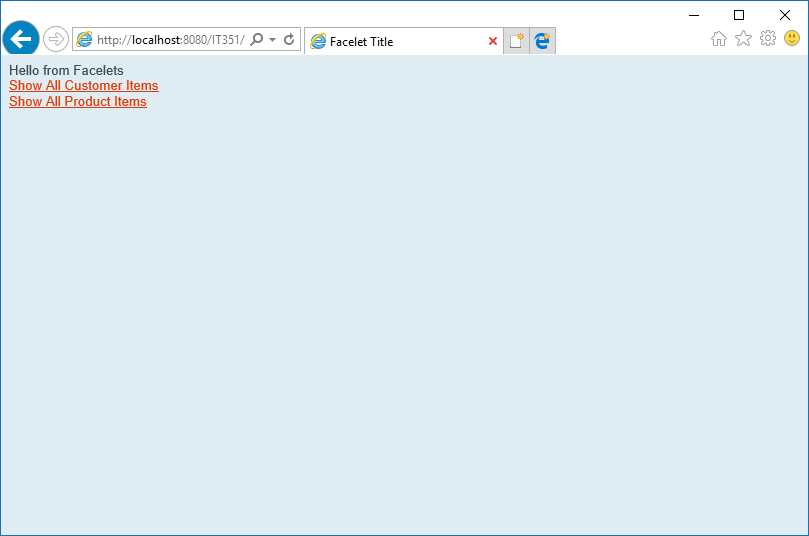
## Test Plan

## Display Customer Table (TC\_custRead)

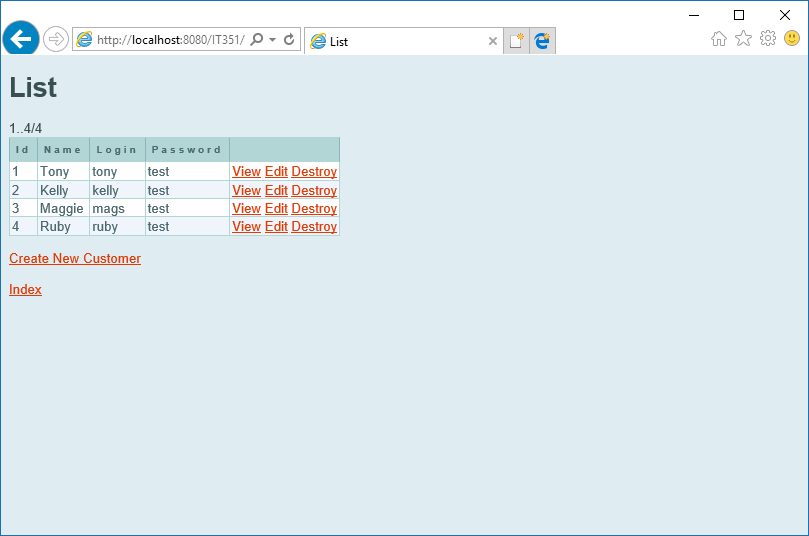
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Database Manager Test Script** | | | | | | | | | | | | | |
| Test Script ID: | | TC\_custRead | | |  | | | | | | | | |
| Test Description: | | Connect to web application and display customer table | | | | | | | | | | | |
| System: | | Preproduction/Training and Testing | | | Test Type: | | | Functional | | | | | |
| Traceability: | | Functionality | | | Priority: | | | Critical | | | | | |
| Expected Results: | | Sales database customer table will be displayed to user | | | | | | | | | | | |
| Pass/Fail Metric: | | Queried data will return in no more than 4 seconds | | | | | | | | | | | |
|  | | | | | | | | | | | | | |
|  | |  |  | |  | | |  |  | |  |  | |
|  | |  |  | |  | | |  |  | |  |  | |
|  | |  |  | |  | | |  |  | |  |  | |
|  | |  |  | |  | | |  |  | |  |  | |
| **Database Manager Test Case** | | | | | | | | | | | | |
| Test Script ID: | TC\_custRead | | | Test Case ID: | | TC\_001 | | | | | | |
| Test Description: | Connect to web application and display customer | | | | | | | | | | | |
| Test Driver ID: | Tester01 | | | System State: | | Preproduction/Training and Testing | | | | | | |
| Input Data Values: | mysql -u ctuonline -p student | | | | | | | | | | | |
| Test Environment: | User LAN-connected Windows desktop connected to localhost | | | | | | | | | | | |
| Req. Traceability: | Module:Functionality | | | | | | | | | | | |
| Test Case Step Instruction | | | | Expected Results | | | Actual Results | | | Pass/Fail | | |
| 1. Open web browser and click link to: <http://localhost:8080/IT351> | | | | Web app will load with option to display items | | | Web app loaded | | | Pass | | |
| 1. Click link to show all customer items | | | | List of customers will load to browser | | | Customers Displayed | | | Pass | | |
| 1. Click Index link to return to start page | | | | Browser will return to start page | | | Returned to Index | | | Pass | | |
|  | | | |  | | |  | | |  | | |

### 

### (TC\_custRead) Output



Step 1 – Connected to Web App

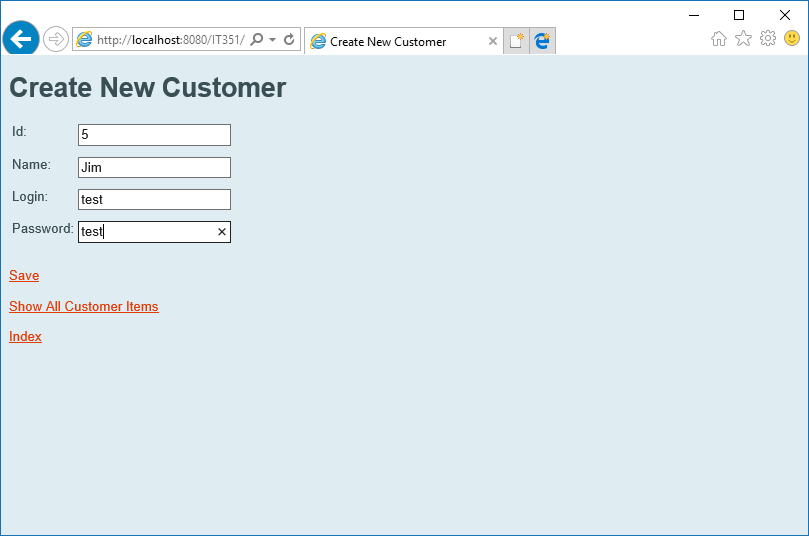


Step 2 – Display Customer Items

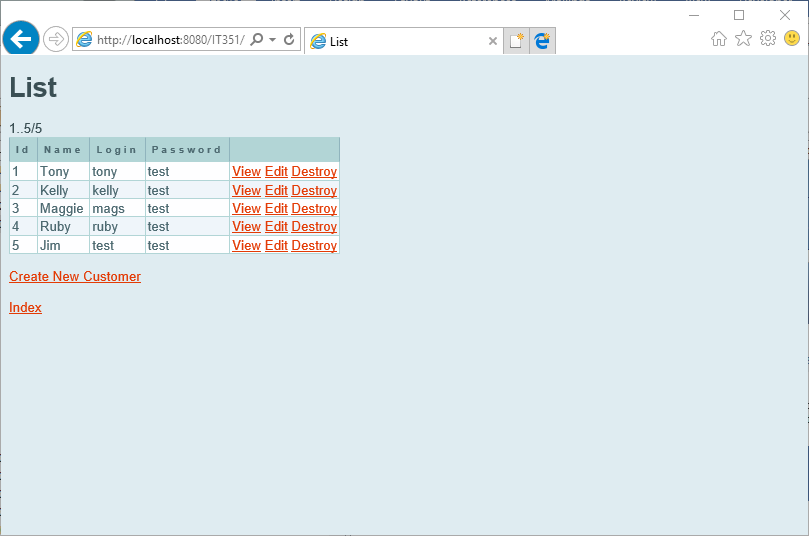
## Create Customer (TC\_custCreate)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Database Manager Test Script** | | | | | | | | | | | | | |
| Test Script ID: | | TC\_custCreate | | |  | | | | | | | | |
| Test Description: | | Create new customer table item | | | | | | | | | | | |
| System: | | Preproduction/Training and Testing | | | Test Type: | | | Functional | | | | | |
| Traceability: | | Module: Functionality | | | Priority: | | | Critical | | | | | |
| Expected Results: | | New customer will be created | | | | | | | | | | | |
| Pass/Fail Metric: | | New customer ID 5 will be displayed in web browser | | | | | | | | | | | |
|  | | | | | | | | | | | | | |
|  | |  |  | |  | | |  |  | |  |  | |
|  | |  |  | |  | | |  |  | |  |  | |
|  | |  |  | |  | | |  |  | |  |  | |
|  | |  |  | |  | | |  |  | |  |  | |
| **Database Manager Test Case** | | | | | | | | | | | | |
| Test Script ID: | TC\_custCreate | | | Test Case ID: | | TC\_002 | | | | | | |
| Test Description: | Create new customer record | | | | | | | | | | | |
| Test Driver ID: | Tester01 | | | System State: | | Preproduction/Training and Testing | | | | | | |
| Input Data Values: | mysql -u ctuonline -p student //  New customer: ID – 5 / Name – Jim / Login – test / Password - test | | | | | | | | | | | |
| Test Environment: | User LAN-connected Windows desktop connected to localhost | | | | | | | | | | | |
| Req. Traceability: | Module:Functionality | | | | | | | | | | | |
| Test Case Step Instruction | | | | Expected Results | | | Actual Results | | | Pass/Fail | | |
| 1. Open web browser and click link to: <http://localhost:8080/IT351> | | | | Web app will load with option to display items | | | Web app loaded | | | Pass | | |
| 1. Click link to show all customer items | | | | List of customers will load to browser | | | Customers Displayed | | | Pass | | |
| 1. Click “Create New Customer” link to add item | | | | Create New Customer webpage will be displayed | | | Page displayed | | | Pass | | |
| 1. Enter new customer values using data values provided above. When entered, click “Save” | | | | “Customer was successfully created” message displayed to page | | | Confirmation Displayed | | | Pass | | |
| 1. Click “Show All Customer Items” link to confirm customer is added | | | | List will contain 5 records, including exact data as provided in data values | | | Item accurately added and displayed to list | | | Pass | | |

### (TC\_custCreate) Output



Step 4 – New customer created



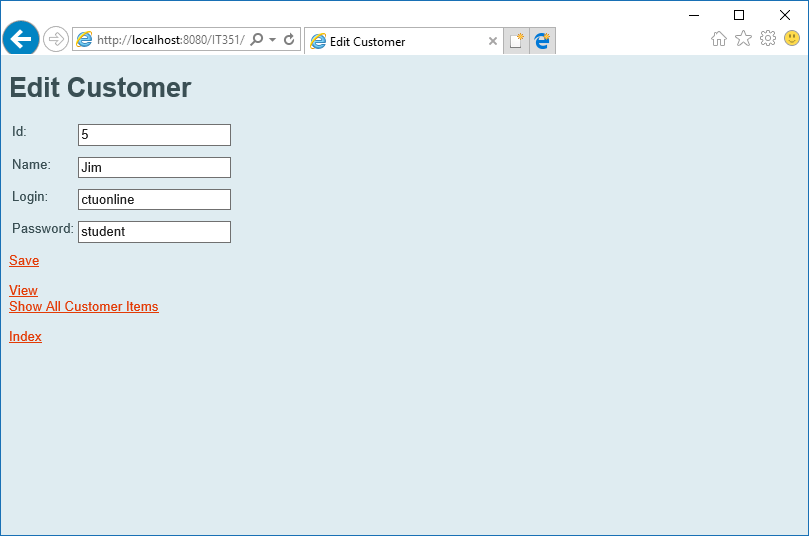
Step 5 – Updated customer list to show added item

## 

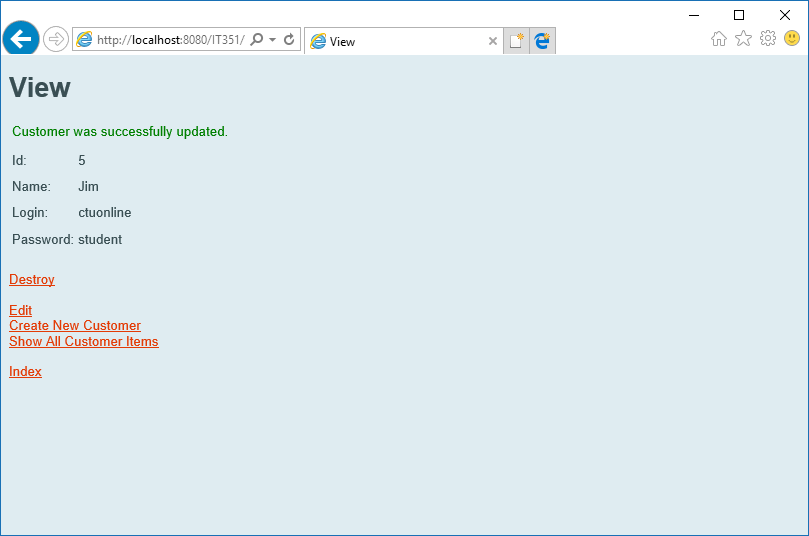
## Update Customer (TC\_custUpdate)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Database Manager Test Script** | | | | | | | | | | | | | |
| Test Script ID: | | TC\_custUpdate | | |  | | | | | | | | |
| Test Description: | | Update an existing customer item | | | | | | | | | | | |
| System: | | Preproduction/Training and Testing | | | Test Type: | | | Functional | | | | | |
| Traceability: | | Module: Functionality | | | Priority: | | | Critical | | | | | |
| Expected Results: | | Existing customer item will be updated and displayed to web page | | | | | | | | | | | |
| Pass/Fail Metric: | | Customer 5 will be updated Item will be displayed to web page | | | | | | | | | | | |
|  | | | | | | | | | | | | | |
|  | |  |  | |  | | |  |  | |  |  | |
|  | |  |  | |  | | |  |  | |  |  | |
|  | |  |  | |  | | |  |  | |  |  | |
|  | |  |  | |  | | |  |  | |  |  | |
| **Database Manager Test Case** | | | | | | | | | | | | |
| Test Script ID: | TC\_custUpdate | | | Test Case ID: | | TC\_003 | | | | | | |
| Test Description: | Update an existing customer item | | | | | | | | | | | |
| Test Driver ID: | Tester01 | | | System State: | | Preproduction/Training and Testing | | | | | | |
| Input Data Values: | mysql -u ctuonline -p student //  Update customer ID 5 – Jim / Login – ctuonline / Password - student | | | | | | | | | | | |
| Test Environment: | User LAN-connected Windows desktop connected to localhost | | | | | | | | | | | |
| Req. Traceability: | Module: Functionality | | | | | | | | | | | |
| Test Case Step Instruction | | | | Expected Results | | | Actual Results | | | Pass/Fail | | |
| 1. Open web browser and click link to: <http://localhost:8080/IT351> | | | | Web app will load with option to display items | | | Web app loaded | | | Pass | | |
| 1. Click link to show all customer items. | | | | List of customers will load to browser | | | Customers Displayed | | | Pass | | |
| 1. Click “Edit” link in row for customer ID 5, Name Jim. | | | | Edit Customer webpage will be displayed | | | Edit page displayed | | | Pass | | |
| 1. Update login and password using (login) “ctuonline” and (pass) “student” values and click “Save”. | | | | “Customer was successfully updated” message displayed to page | | | Confirmation Displayed with updated item values | | | Pass | | |

### (TC\_custUpdate) Output



Step 4 – Customer with updated login and password values

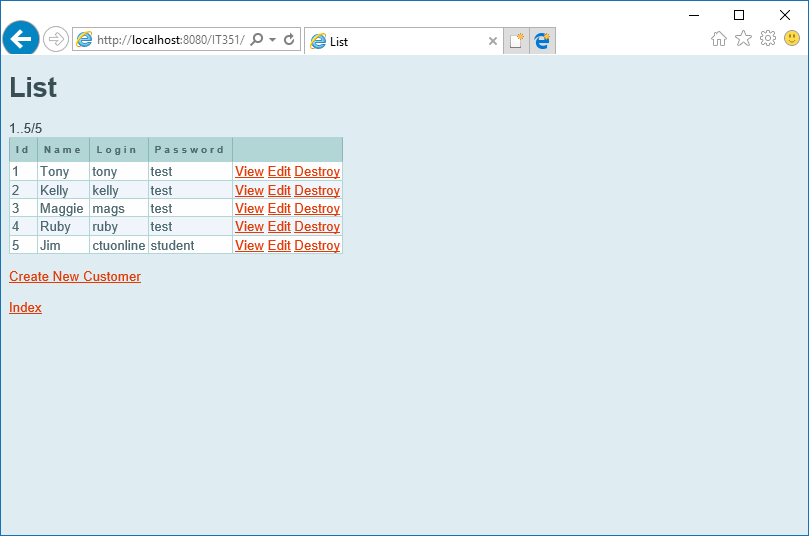


Verification with updated values after clicking “Save”

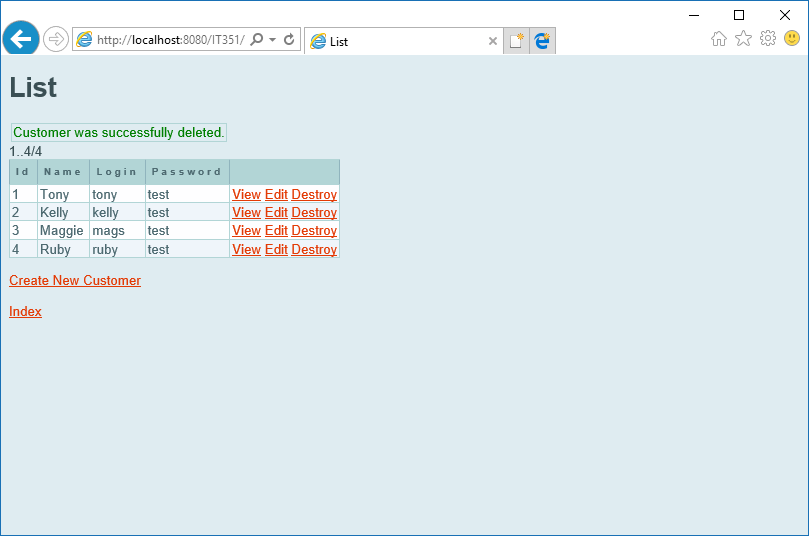
## Delete Customer (TC\_custDelete)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Database Manager Test Script** | | | | | | | | | | | | | |
| Test Script ID: | | TC\_custDelete | | |  | | | | | | | | |
| Test Description: | | Delete an existing customer item | | | | | | | | | | | |
| System: | | Preproduction/Training and Testing | | | Test Type: | | | Functional | | | | | |
| Traceability: | | Module: Functionality | | | Priority: | | | Critical | | | | | |
| Expected Results: | | Existing customer item will be deleted and verified by viewing list | | | | | | | | | | | |
| Pass/Fail Metric: | | Item 5 is successfully deleted | | | | | | | | | | | |
|  | | | | | | | | | | | | | |
|  | |  |  | |  | | |  |  | |  |  | |
|  | |  |  | |  | | |  |  | |  |  | |
|  | |  |  | |  | | |  |  | |  |  | |
|  | |  |  | |  | | |  |  | |  |  | |
| **Database Manager Test Case** | | | | | | | | | | | | |
| Test Script ID: | TC\_custDelete | | | Test Case ID: | | TC\_004 | | | | | | |
| Test Description: | Delete an existing customer item | | | | | | | | | | | |
| Test Driver ID: | Tester01 | | | System State: | | Preproduction/Training and Testing | | | | | | |
| Input Data Values: | mysql -u ctuonline -p student // | | | | | | | | | | | |
| Test Environment: | User LAN-connected Windows desktop connected to localhost | | | | | | | | | | | |
| Req. Traceability: | Module: Functionality | | | | | | | | | | | |
| Test Case Step Instruction | | | | Expected Results | | | Actual Results | | | Pass/Fail | | |
| 1. Open web browser and click link to: <http://localhost:8080/IT351> | | | | Web app will load with option to display items | | | Web app loaded | | | Pass | | |
| 1. Click link to show all customer items. | | | | List of customers will load to browser | | | Customers Displayed | | | Pass | | |
| 1. Click “Destroy” link in row for customer ID 5, Name Jim. | | | | List will update and item associated with ID 5 will be removed. Verification message displayed to webpage along with updated list to show item removal | | | Item removed and updated list displayed. “Customer was successfully deleted” confirmation. | | | Pass | | |

### (TC\_custDelete) Output



“Destroy” clicked for row 5 to remove item from database.



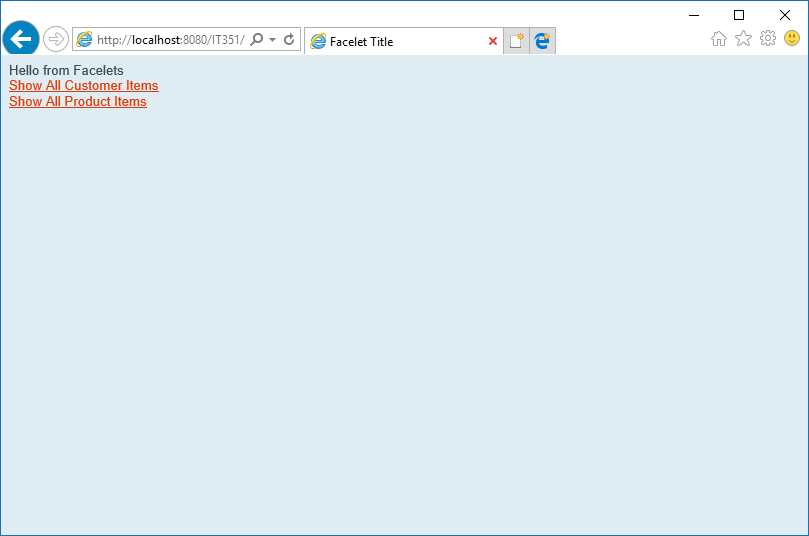
Confirmation displayed of item removal. Updated customer table displayed to page.

# Product Testing

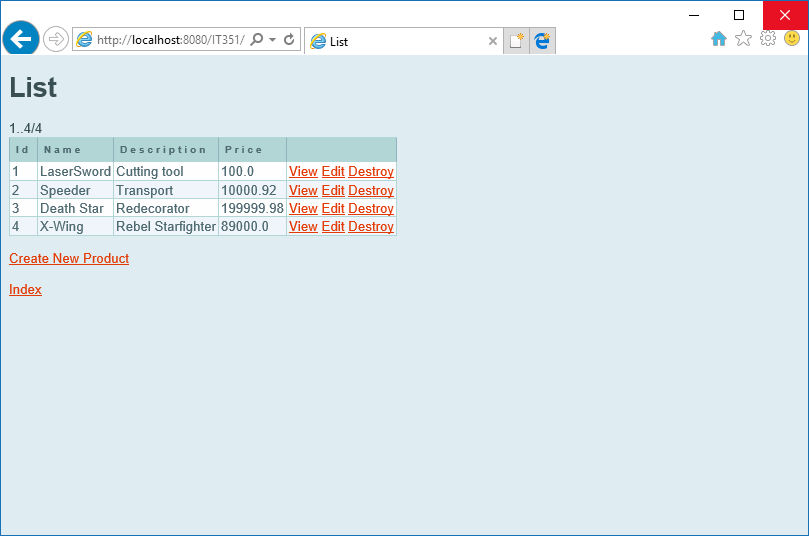
## Display Product Table (TC\_prodRead)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Database Manager Test Script** | | | | | | | | | | | | | |
| Test Script ID: | | TC\_prodRead | | |  | | | | | | | | |
| Test Description: | | Connect to web application and display product table | | | | | | | | | | | |
| System: | | Preproduction/Training and Testing | | | Test Type: | | | Functional | | | | | |
| Traceability: | | Functionality | | | Priority: | | | Critical | | | | | |
| Expected Results: | | Sales database product table will be displayed to user | | | | | | | | | | | |
| Pass/Fail Metric: | | Queried data will return in no more than 4 seconds | | | | | | | | | | | |
|  | | | | | | | | | | | | | |
|  | |  |  | |  | | |  |  | |  |  | |
|  | |  |  | |  | | |  |  | |  |  | |
|  | |  |  | |  | | |  |  | |  |  | |
|  | |  |  | |  | | |  |  | |  |  | |
| **Database Manager Test Case** | | | | | | | | | | | | |
| Test Script ID: | TC\_prodRead | | | Test Case ID: | | TC\_005 | | | | | | |
| Test Description: | Connect to web application and display product table | | | | | | | | | | | |
| Test Driver ID: | Tester02 | | | System State: | | Preproduction/Training and Testing | | | | | | |
| Input Data Values: | mysql -u ctuonline -p student | | | | | | | | | | | |
| Test Environment: | User LAN-connected Windows desktop connected to localhost | | | | | | | | | | | |
| Req. Traceability: | Module:Functionality | | | | | | | | | | | |
| Test Case Step Instruction | | | | Expected Results | | | Actual Results | | | Pass/Fail | | |
| 1. Open web browser and click link to: <http://localhost:8080/IT351> | | | | Web app will load with option to display items | | | Web app loaded | | | Pass | | |
| 1. Click link to show all Product items | | | | List of products will load to browser | | | Products Displayed | | | Pass | | |
| 1. Click Index link to return to start page | | | | Browser will return to start page | | | Returned to Index | | | Pass | | |
|  | | | |  | | |  | | |  | | |

### (TC\_prodRead) Output



Step 1 – Connected to Web App

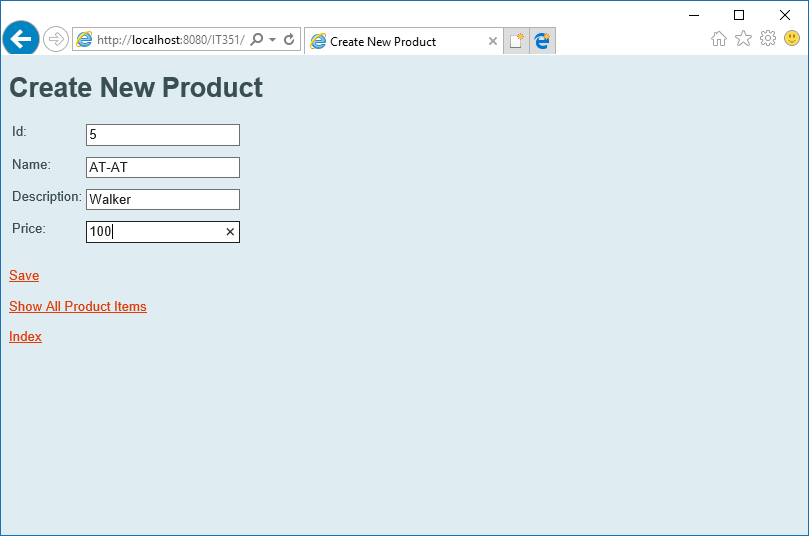


Step 2 – Display Product Items

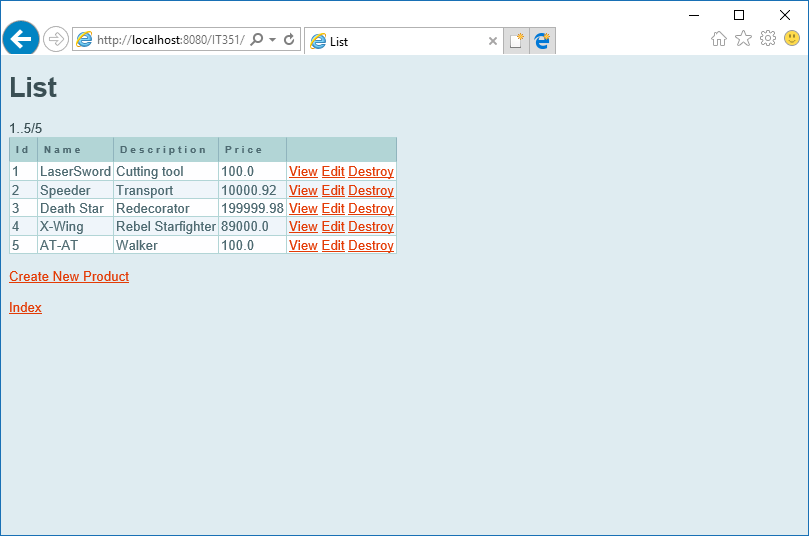
## Create Product (TC\_prodCreate)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Database Manager Test Script** | | | | | | | | | | | | | |
| Test Script ID: | | TC\_prodCreate | | |  | | | | | | | | |
| Test Description: | | Create new product table item | | | | | | | | | | | |
| System: | | Preproduction/Training and Testing | | | Test Type: | | | Functional | | | | | |
| Traceability: | | Module: Functionality | | | Priority: | | | Critical | | | | | |
| Expected Results: | | New product will be created | | | | | | | | | | | |
| Pass/Fail Metric: | | New product ID 5 will be displayed in web browser | | | | | | | | | | | |
|  | | | | | | | | | | | | | |
|  | |  |  | |  | | |  |  | |  |  | |
|  | |  |  | |  | | |  |  | |  |  | |
|  | |  |  | |  | | |  |  | |  |  | |
|  | |  |  | |  | | |  |  | |  |  | |
| **Database Manager Test Case** | | | | | | | | | | | | |
| Test Script ID: | TC\_prodCreate | | | Test Case ID: | | TC\_006 | | | | | | |
| Test Description: | Create new product record | | | | | | | | | | | |
| Test Driver ID: | Tester02 | | | System State: | | Preproduction/Training and Testing | | | | | | |
| Input Data Values: | mysql -u ctuonline -p student //  New product:  ID – 5 / Name – AT-AT / Description - Walker / Price - 100 | | | | | | | | | | | |
| Test Environment: | User LAN-connected Windows desktop connected to localhost | | | | | | | | | | | |
| Req. Traceability: | Module:Functionality | | | | | | | | | | | |
| Test Case Step Instruction | | | | Expected Results | | | Actual Results | | | Pass/Fail | | |
| 1. Open web browser and click link to: <http://localhost:8080/IT351> | | | | Web app will load with option to display items | | | Web app loaded | | | Pass | | |
| 1. Click link to show all product items | | | | List of products will load to browser | | | products Displayed | | | Pass | | |
| 1. Click “Create New Product” link to add item | | | | Create New Product webpage will be displayed | | | Page displayed | | | Pass | | |
| 1. Enter new product values using data values provided above. When entered, click “Save” | | | | “Product was successfully created” message displayed to page | | | Confirmation Displayed | | | Pass | | |
| 1. Click “Show All Product Items” link to confirm customer is added | | | | List will contain 5 records, including exact data as provided in data values | | | Item accurately added and displayed to list | | | Pass | | |

### (TC\_prodCreate) Output



Step 4 – New product created



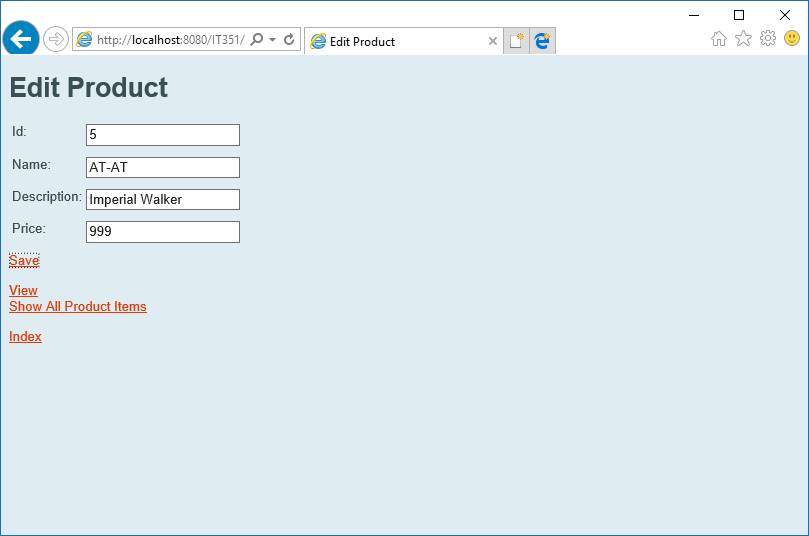
Step 5 – Updated product list to show added item

## 

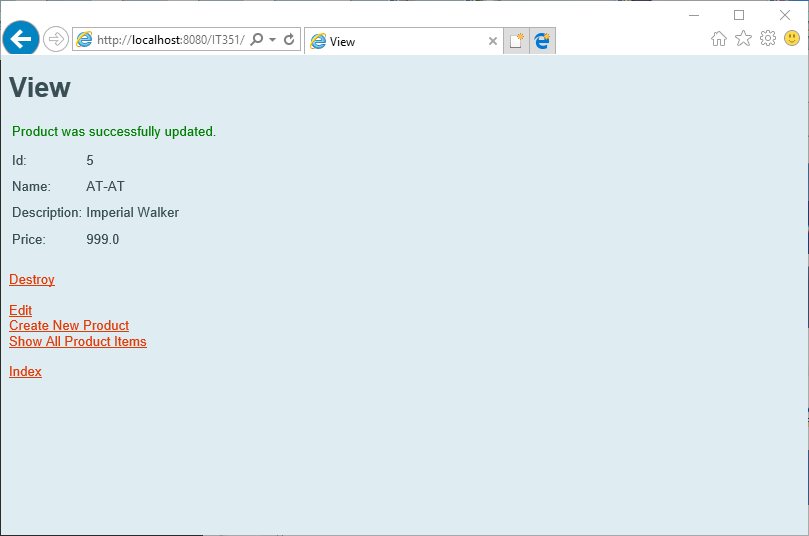
## Update Product (TC\_prodUpdate)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Database Manager Test Script** | | | | | | | | | | | | | |
| Test Script ID: | | TC\_prodUpdate | | |  | | | | | | | | |
| Test Description: | | Update an existing product item 5 | | | | | | | | | | | |
| System: | | Preproduction/Training and Testing | | | Test Type: | | | Functional | | | | | |
| Traceability: | | Module: Functionality | | | Priority: | | | Critical | | | | | |
| Expected Results: | | Existing product item will be updated and displayed to web page | | | | | | | | | | | |
| Pass/Fail Metric: | | Product 5 will be updated Item will be displayed to web page | | | | | | | | | | | |
|  | | | | | | | | | | | | | |
|  | |  |  | |  | | |  |  | |  |  | |
|  | |  |  | |  | | |  |  | |  |  | |
|  | |  |  | |  | | |  |  | |  |  | |
|  | |  |  | |  | | |  |  | |  |  | |
| **Database Manager Test Case** | | | | | | | | | | | | |
| Test Script ID: | TC\_prodUpdate | | | Test Case ID: | | TC\_007 | | | | | | |
| Test Description: | Update an existing product item 5 | | | | | | | | | | | |
| Test Driver ID: | Tester02 | | | System State: | | Preproduction/Training and Testing | | | | | | |
| Input Data Values: | mysql -u ctuonline -p student //  Update product ID 5 – AT-AT with updated values:  description – Imperial Walker/ Price - 999 | | | | | | | | | | | |
| Test Environment: | User LAN-connected Windows desktop connected to localhost | | | | | | | | | | | |
| Req. Traceability: | Module: Functionality | | | | | | | | | | | |
| Test Case Step Instruction | | | | Expected Results | | | Actual Results | | | Pass/Fail | | |
| 1. Open web browser and click link to: <http://localhost:8080/IT351> | | | | Web app will load with option to display items | | | Web app loaded | | | Pass | | |
| 1. Click link to show all product items. | | | | List of products will load to browser | | | products Displayed | | | Pass | | |
| 1. Click “Edit” link in row for product ID 5, Name AT-AT. | | | | Edit product webpage will be displayed | | | Edit page displayed | | | Pass | | |
| 1. Update description and price using (description) “Imperial Walker” and (price) “999” values and click “Save”. | | | | “Product was successfully updated” message displayed to page | | | Confirmation Displayed with updated item values | | | Pass | | |

### (TC\_prodUpdate) Output



Step 4 – Product with updated description and price values

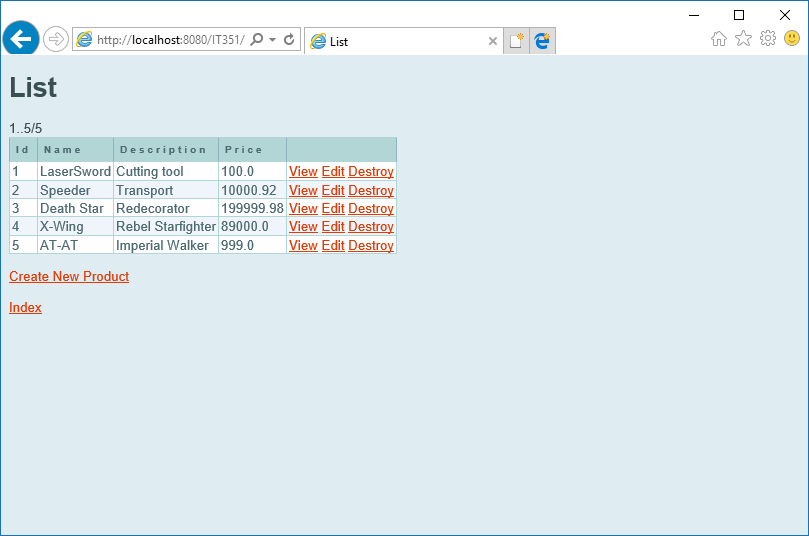


Verification with updated values after clicking “Save”

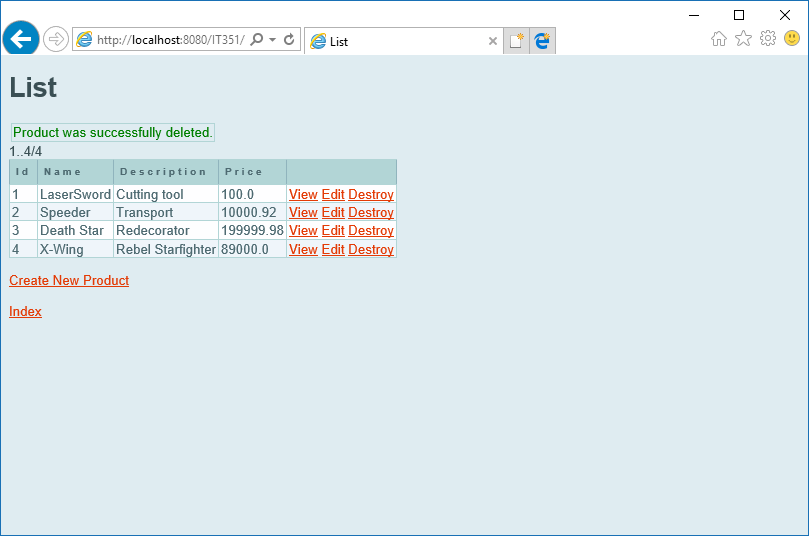
## Delete Product (TC\_prodDelete)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Database Manager Test Script** | | | | | | | | | | | | | |
| Test Script ID: | | TC\_prodDelete | | |  | | | | | | | | |
| Test Description: | | Delete an existing product item | | | | | | | | | | | |
| System: | | Preproduction/Training and Testing | | | Test Type: | | | Functional | | | | | |
| Traceability: | | Module: Functionality | | | Priority: | | | Critical | | | | | |
| Expected Results: | | Existing product item will be deleted and verified by viewing list | | | | | | | | | | | |
| Pass/Fail Metric: | | Item 5 is successfully deleted | | | | | | | | | | | |
|  | | | | | | | | | | | | | |
|  | |  |  | |  | | |  |  | |  |  | |
|  | |  |  | |  | | |  |  | |  |  | |
|  | |  |  | |  | | |  |  | |  |  | |
|  | |  |  | |  | | |  |  | |  |  | |
| **Database Manager Test Case** | | | | | | | | | | | | |
| Test Script ID: | TC\_prodDelete | | | Test Case ID: | | TC\_008 | | | | | | |
| Test Description: | Delete an existing product item | | | | | | | | | | | |
| Test Driver ID: | Tester02 | | | System State: | | Preproduction/Training and Testing | | | | | | |
| Input Data Values: | mysql -u ctuonline -p student // | | | | | | | | | | | |
| Test Environment: | User LAN-connected Windows desktop connected to localhost | | | | | | | | | | | |
| Req. Traceability: | Module: Functionality | | | | | | | | | | | |
| Test Case Step Instruction | | | | Expected Results | | | Actual Results | | | Pass/Fail | | |
| 1. Open web browser and click link to: <http://localhost:8080/IT351> | | | | Web app will load with option to display items | | | Web app loaded | | | Pass | | |
| 1. Click link to show all product items. | | | | List of products will load to browser | | | Products Displayed | | | Pass | | |
| 1. Click “Destroy” link in row for product ID 5, Name AT-AT. | | | | List will update and item associated with ID 5 will be removed. Verification message displayed to webpage along with updated list to show item removal | | | Item removed and updated list displayed. “Product was successfully deleted” confirmation. | | | Pass | | |

### (TC\_prodDelete) Output



“Destroy” clicked for row 5 to remove item from database.



Confirmation displayed of item removal. Updated customer table displayed to page.