**PES UNIVERSITY**

**Department of Computer Science & Engineering**

****

DBMS - UE20CS301

Mini Project

Inventory Management System

## Submitted to: Submitted By:

Dr. Geetha D Name: Vishwa Mehul Mehta

Associate Professor SRN: PES2UG20CS389

Semester: V

Section: F

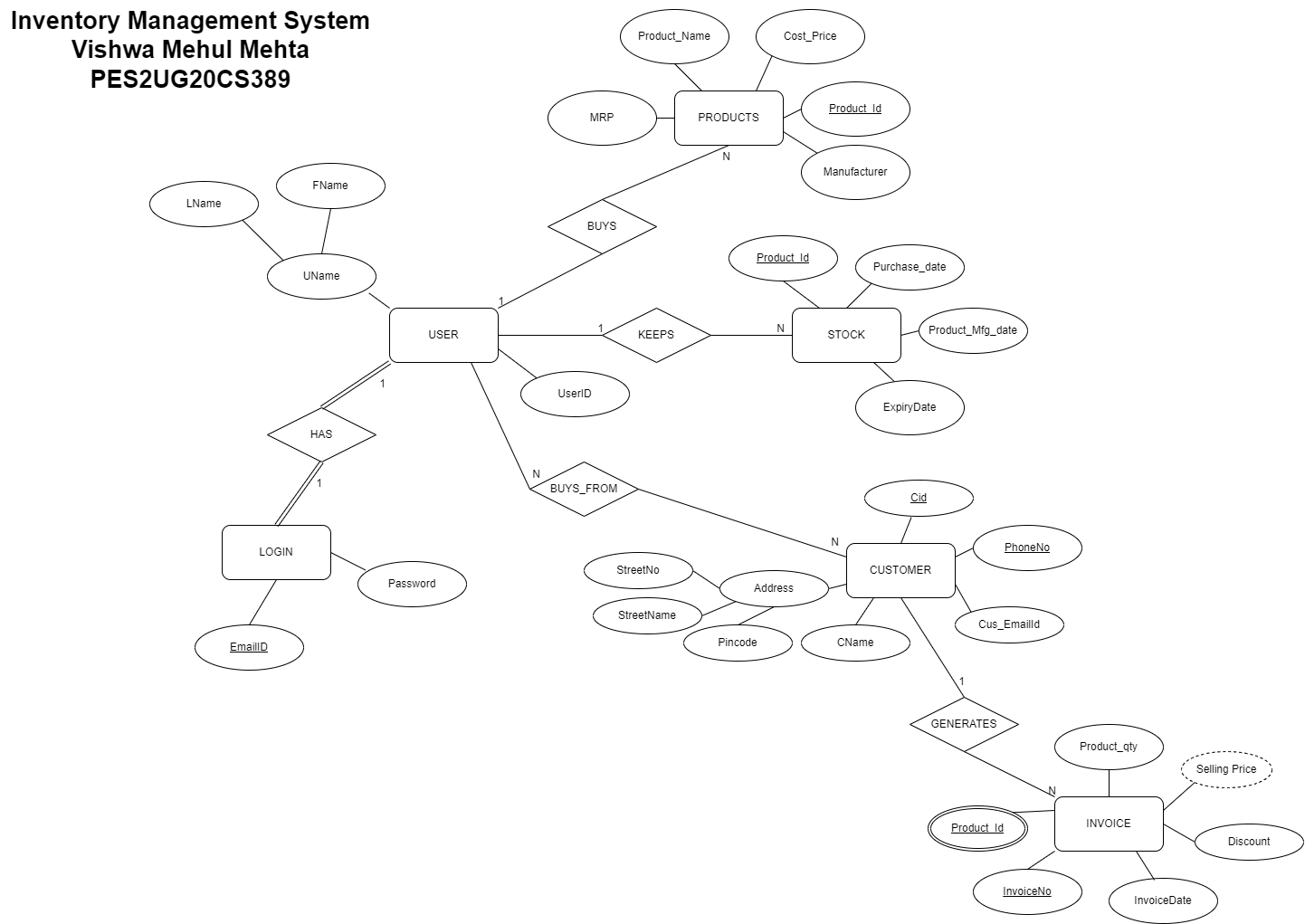
# Table of Contents

|  |  |  |
| --- | --- | --- |
| **Sl.No** | **Title** | **Page No** |
| 1. | Short Description and Scope of the Project | 3 |
| 2. | Entity Relationship Diagram | 4 |
| 3. | Relational Schema | 5 |
| 4. | DDL Statements-Building the  Database | 6 |
| 5. | Populating the Database | 9 |
| 6. | Join Queries | 13 |
| 7. | Attribute Functions | 15 |
| 8. | Set Operations | 17 |
| 9. | Functions and Procedures | 20 |
| 10. | Triggers and Cursors | 24 |
| 11. | Frontend Application | 30 |
| 12. | Conclusion | 58 |
| 13. | References | 59 |

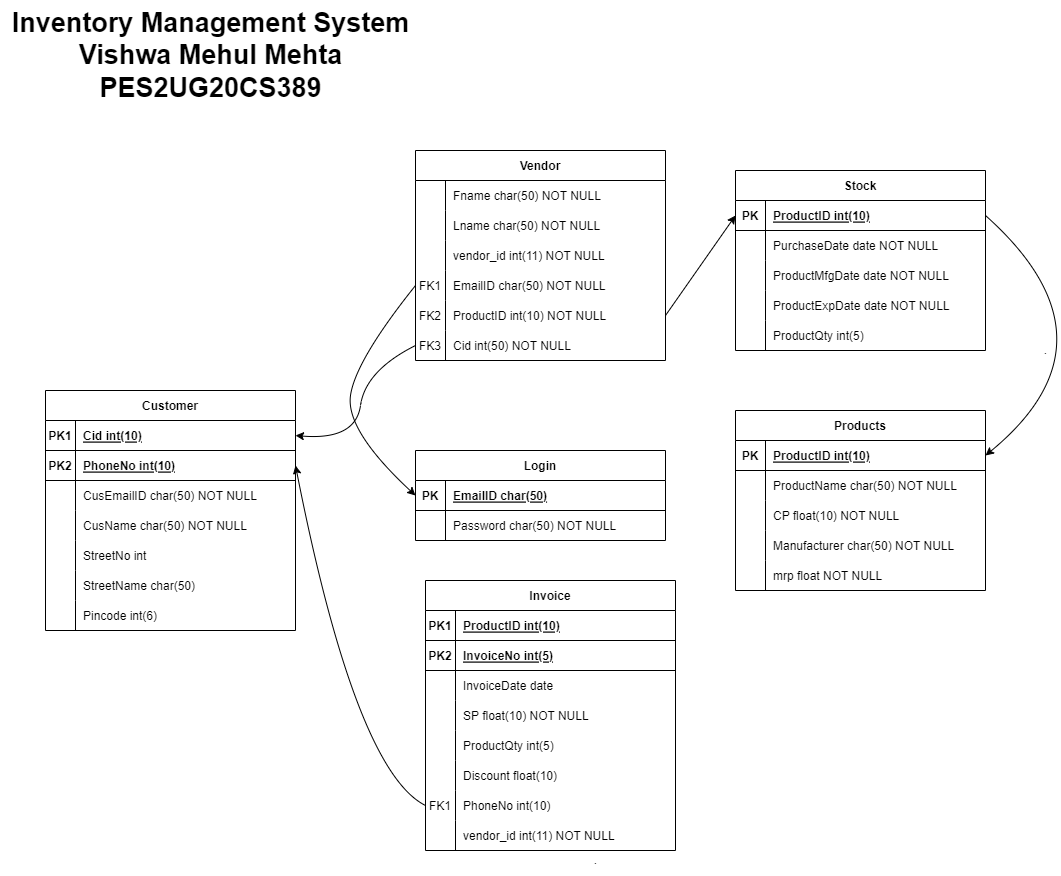
**1. Short Description and Scope of the Project**

ListAll is a web application that monitors and manages the goods and products in stock. The application uses a database and database operations to store information about the retailers and their respective products that are sold. It displays a list of all products and their quantities, customers who bought the items and their details along with invoice for each product purchased. It allows for performing CRUD operations for all the details. ListAll uses a simple frontend using Python based Streamlit and MySQL for the backend. The application aims to find profit/loss and generate it for a particular month or year as a future application.

**2. ER Diagram**



**3. Relational Schema**



**4. DDL statements - Building the database**

-- Table structure for table `customer`

--

CREATE TABLE IF NOT EXISTS `customer` (

  `cust\_id` int(10) NOT NULL,

  `phone\_no` bigint(10) NOT NULL,

  `cust\_email\_id` varchar(50) NOT NULL,

  `cust\_name` varchar(50) NOT NULL,

  `str\_no` int(11) DEFAULT NULL,

  `str\_name` varchar(50) DEFAULT NULL,

  `pincode` int(6) DEFAULT NULL,

  PRIMARY KEY (`cust\_id`,`phone\_no`)

) ENGINE=MyISAM DEFAULT CHARSET=utf8;

-- Table structure for table `invoice`

--

CREATE TABLE IF NOT EXISTS `invoice` (

  `prod\_id` int(10) NOT NULL,

  `invoice\_no` int(10) NOT NULL,

  `invoice\_date` date NOT NULL,

  `selling\_price` float NOT NULL,

  `prod\_qty` int(11) DEFAULT '0',

  `discount` float DEFAULT NULL,

  `phone\_no` bigint(10) DEFAULT NULL,

  `vendor\_id` int(11) NOT NULL,

  PRIMARY KEY (`prod\_id`,`invoice\_no`),

  KEY `phone\_no` (`phone\_no`)

) ENGINE=MyISAM DEFAULT CHARSET=utf8;

-- Table structure for table `login`

--

CREATE TABLE IF NOT EXISTS `login` (

  `email\_id` varchar(50) NOT NULL,

  `pass` varchar(50) NOT NULL,

  PRIMARY KEY (`email\_id`)

) ENGINE=MyISAM DEFAULT CHARSET=utf8;

-- Table structure for table `product`

--

CREATE TABLE IF NOT EXISTS `product` (

  `product\_id` int(10) NOT NULL,

  `product\_name` varchar(50) NOT NULL,

  `cost\_price` float NOT NULL,

  `manufacturer` varchar(50) NOT NULL,

  `mrp` float NOT NULL,

  PRIMARY KEY (`product\_id`)

) ENGINE=MyISAM DEFAULT CHARSET=utf8;

-- Table structure for table `stock`

--

CREATE TABLE IF NOT EXISTS `stock` (

  `pid` int(10) NOT NULL,

  `purchase\_date` date NOT NULL,

  `mfg\_date` date NOT NULL,

  `pqty` int(11) DEFAULT '0',

  PRIMARY KEY (`pid`)

) ENGINE=MyISAM DEFAULT CHARSET=utf8;

-- Table structure for table `vendor`

--

CREATE TABLE IF NOT EXISTS `vendor` (

  `fname` varchar(50) NOT NULL,

  `lname` varchar(50) NOT NULL,

  `vendor\_id` int(11) NOT NULL,

  `email\_id` varchar(50) NOT NULL,

  `product\_id` int(10) DEFAULT NULL,

  `cust\_id` int(10) DEFAULT NULL,

  KEY `email\_id` (`email\_id`),

  KEY `cust\_id` (`cust\_id`),

  KEY `product\_id` (`product\_id`)

) ENGINE=MyISAM DEFAULT CHARSET=utf8;

**5. Populating the Database**

-- Dumping data for table `customer`

--

INSERT INTO `customer` (`cust\_id`, `phone\_no`, `cust\_email\_id`, `cust\_name`, `str\_no`, `str\_name`, `pincode`) VALUES

(57317, 5289998172, 'inderpal@gmail.com', 'Inderpal Ravinder', 12, 'Goel Street', 649721),

(26864, 8043894922, 'ishi@gmail.com', 'Ishita Devraj', 14, 'Ramkissoon Street', 427425),

(58462, 8545588678, 'anikp@gmail.com', 'Anik Prabhakar', 56, 'Sekhon Street', 139845),

(63698, 3816889497, 'ninadram@gmail.com', 'Ninad Ram', 37, 'Kari Street', 456639),

(33295, 9489333654, 'devis@yahoo.com', 'Devi Sunder', 28, 'Subramaniam Street', 376921),

(22150, 9056364822, 'rama123@yahoo.com', 'Ramadevi Gopala', 57, 'Sen Street', 758086),

(76443, 7294778333, 'pravinajaya@gmail.com', 'Pravina Jaya', 22, 'Raja Street', 591648),

(45406, 4305436406, 'anilabhi@gmail.com', 'Anil Abhishek', 71, 'Shroff Street', 234740),

(26864, 8145956923, 'nilofara@hotmail.com', 'Nilofar Amardeep', 83, 'Vig Street', 158657),

(43143, 5636459498, 'vinayarvind24@hotmail.com', 'Vinay Aravind', 26, 'Chakraborty Street', 525343);

-- Dumping data for table `invoice`

--

INSERT INTO `invoice` (`prod\_id`, `invoice\_no`, `invoice\_date`, `selling\_price`, `prod\_qty`, `discount`, `phone\_no`, `vendor\_id`) VALUES

(59854, 38980, '2022-10-22', 22.8, 2, 0.2, 5289998172, 75071),

(59854, 85957, '2022-10-23', 22.8, 2, 0.2, 8043894922, 75071),

(59854, 99454, '2022-10-23', 22.8, 2, 0.2, 8545588678, 40101);

-- Dumping data for table `login`

--

INSERT INTO `login` (`email\_id`, `pass`) VALUES

('harry@gmail.com', 'har1234%'),

('ramesh@gmail.com', 'rameshH234'),

('ram@gmail.com', 'ram135&'),

('ranjeeta@gmail.com', 'ran234%'),

('rakesh@gmail.com', 'rakeshm345');

-- Dumping data for table `product`

--

INSERT INTO `product` (`product\_id`, `product\_name`, `cost\_price`, `manufacturer`, `mrp`) VALUES

(59854, 'milk', 15, 'Amul', 19),

(53028, 'curd', 30, 'Amul', 34),

(56145, 'butter', 25, 'Amul', 28),

(81010, 'cow milk', 15, 'Nandini', 17),

(99016, 'yougurt', 25, 'Nandini', 30),

(15292, 'paneer', 55, 'Milky Mist', 60),

(45124, 'cheese', 85, 'Amul', 90),

(14270, 'ghee', 440, 'Nandini', 450),

(86934, 'ice cream family pack', 180, 'Quality Walls', 200),

(98390, 'ice cream cone', 35, 'Cornetto', 40);

-- Dumping data for table `stock`

--

INSERT INTO `stock` (`pid`, `purchase\_date`, `mfg\_date`, `pqty`) VALUES

(59854, '2022-10-21', '2022-10-21', 80),

(53028, '2022-10-18', '2022-10-18', 90),

(56145, '2022-10-12', '2022-10-10', 60),

(81010, '2022-10-21', '2022-10-21', 60),

(99016, '2022-10-17', '2022-10-16', 50),

(15292, '2022-10-19', '2022-10-19', 30),

(45124, '2022-10-04', '2022-09-29', 35),

(14270, '2022-10-02', '2022-09-20', 40),

(86934, '2022-09-20', '2022-09-10', 50),

(98390, '2022-10-01', '2022-09-30', 35);

-- Dumping data for table `vendor`

--

INSERT INTO `vendor` (`fname`, `lname`, `vendor\_id`, `email\_id`, `product\_id`, `cust\_id`) VALUES

('harry', 'singh', 75071, 'harry@gmail.com', 59854, 57317),

('harry', 'singh', 75071, 'harry@gmail.com', 59854, 26864),

('harry', 'singh', 75071, 'harry@gmail.com', 59854, 58462),

('harry', 'singh', 75071, 'harry@gmail.com', 53028, 57317),

('harry', 'singh', 75071, 'harry@gmail.com', 14270, 58462),

('harry', 'singh', 75071, 'harry@gmail.com', 59854, 63698),

('harry', 'singh', 75071, 'harry@gmail.com', 98390, 63698),

('ramesh', 'sharma', 40101, 'ramesh@gmail.com', 59854, 33295),

('ramesh', 'sharma', 40101, 'ramesh@gmail.com', 15292, 33295),

('ramesh', 'sharma', 40101, 'ramesh@gmail.com', 59854, 33295),

('ramesh', 'sharma', 40101, 'ramesh@gmail.com', 14270, 22150),

('ramesh', 'sharma', 40101, 'ramesh@gmail.com', 59854, 22150),

('ram', 'c', 70978, 'ram@gmail.com', 53028, 76443),

('ram', 'c', 70978, 'ram@gmail.com', 59854, 76443),

('ranjeeta', 's', 94138, 'ranjeeta@gmail.com', 59854, 45406),

('ranjeeta', 's', 94138, 'ranjeeta@gmail.com', 45124, 26864),

('ranjeeta', 's', 94138, 'ranjeeta@gmail.com', 98390, 45406),

('rakesh', 'sharma', 29690, 'rakesh@gmail.com', 81010, 43143),

('rakesh', 'sharma', 29690, 'rakesh@gmail.com', 15292, 43143);

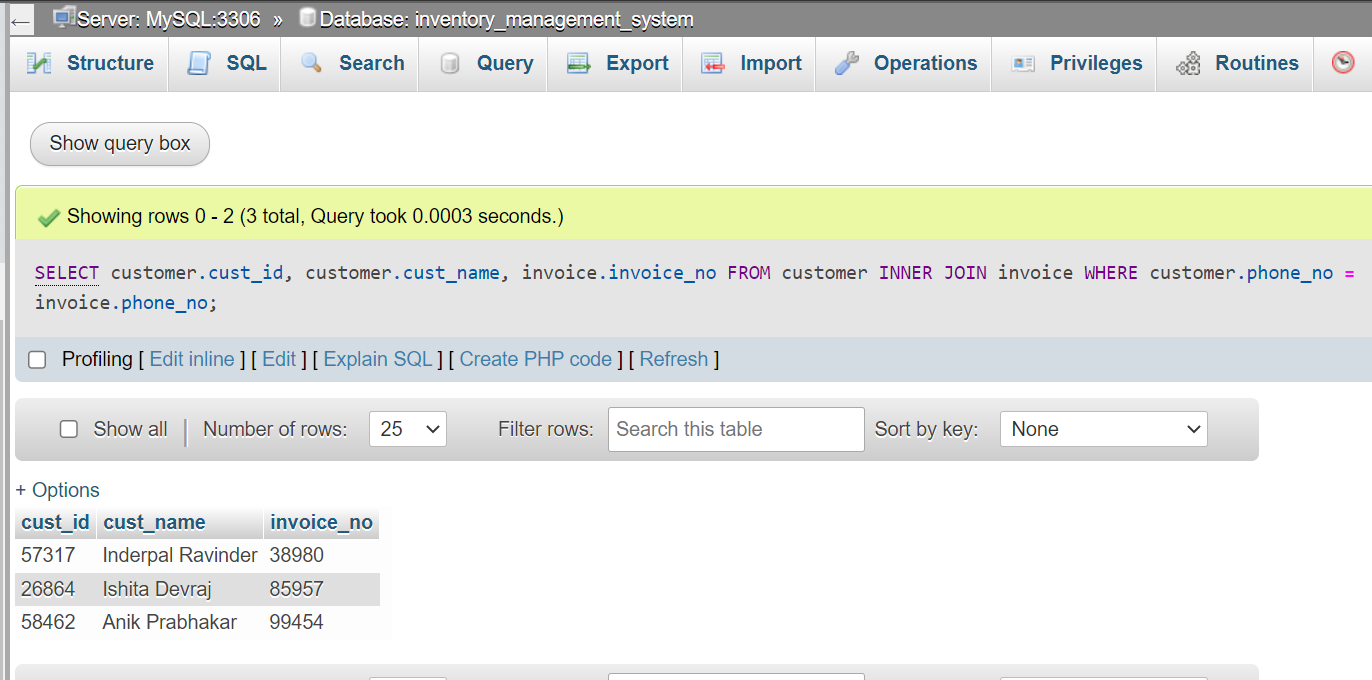
**6. Join Queries**

a. List of all customers that have an invoice associated with them.

Query:

SELECT customer.cust\_id, customer.cust\_name, invoice.invoice\_no FROM customer INNER JOIN invoice WHERE customer.phone\_no = invoice.phone\_no;

Screenshot:

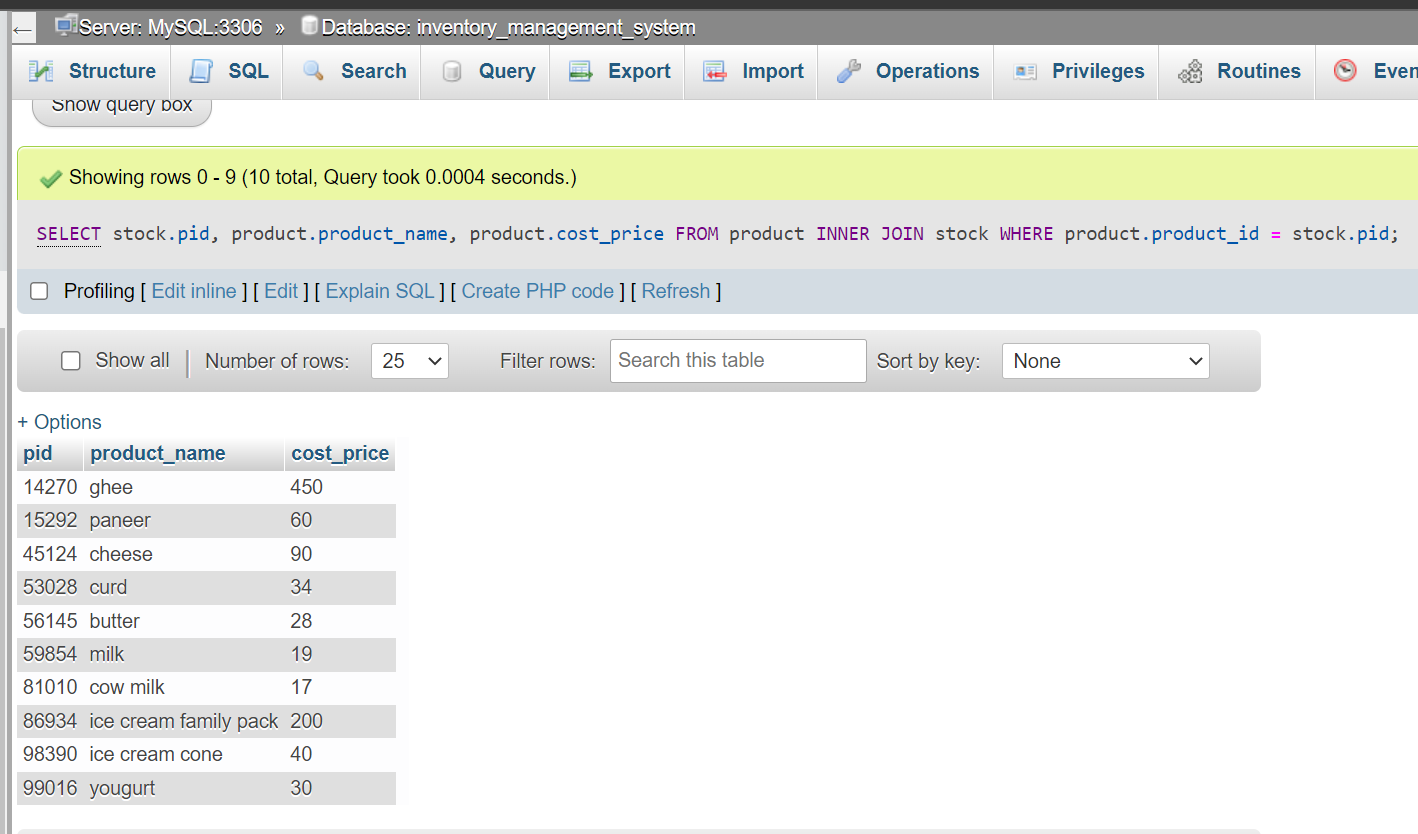


b. List all the cost prices of products in stock.

Query:

SELECT stock.pid, product.product\_name, product.cost\_price FROM product INNER JOIN stock WHERE product.product\_id = stock.pid;

Screenshot:

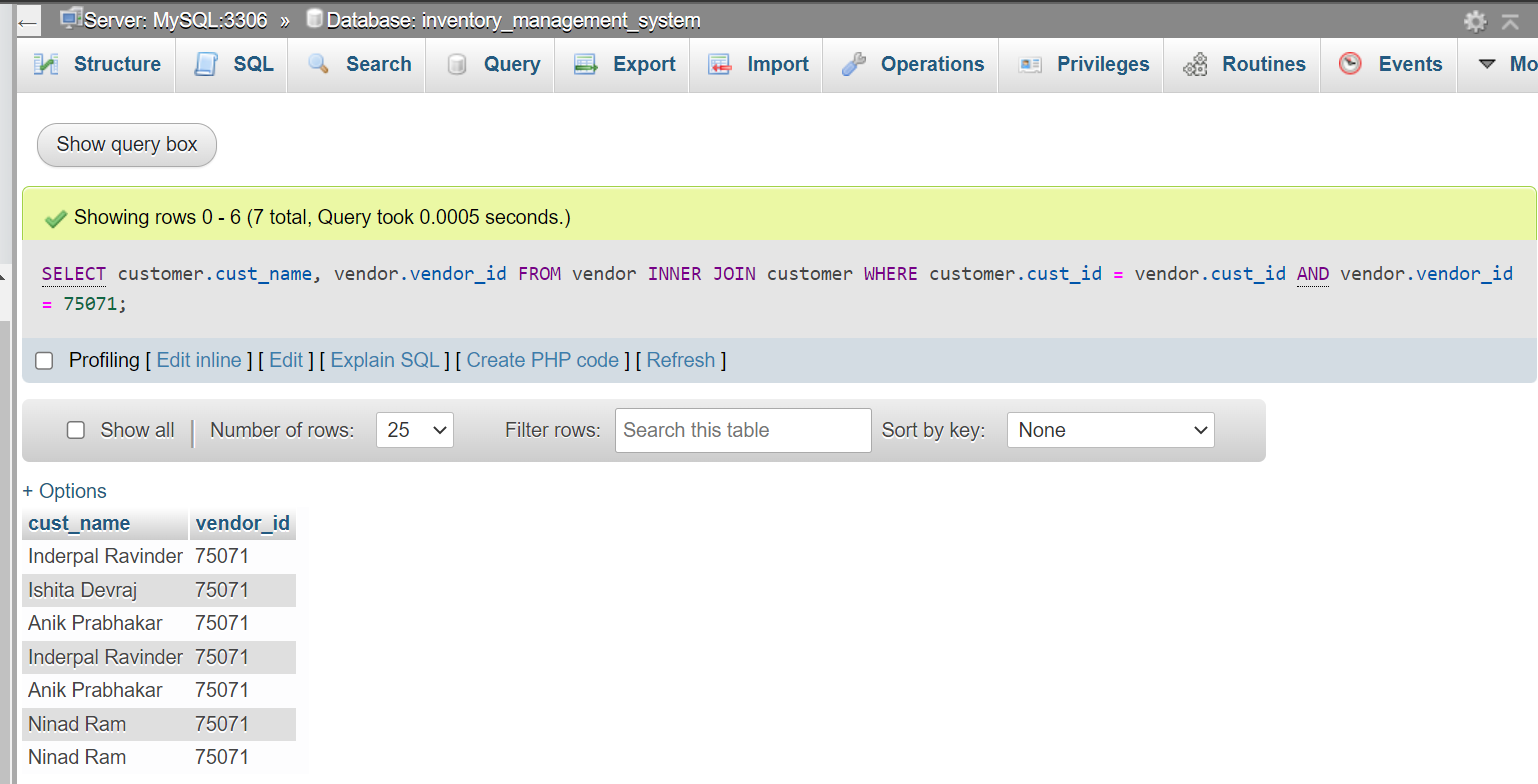


c. List all customer names who visit a certain vendor.

Query:

SELECT customer.cust\_name, vendor.vendor\_id FROM vendor INNER JOIN customer WHERE customer.cust\_id = vendor.cust\_id AND vendor.vendor\_id = 75071;

Screenshot:

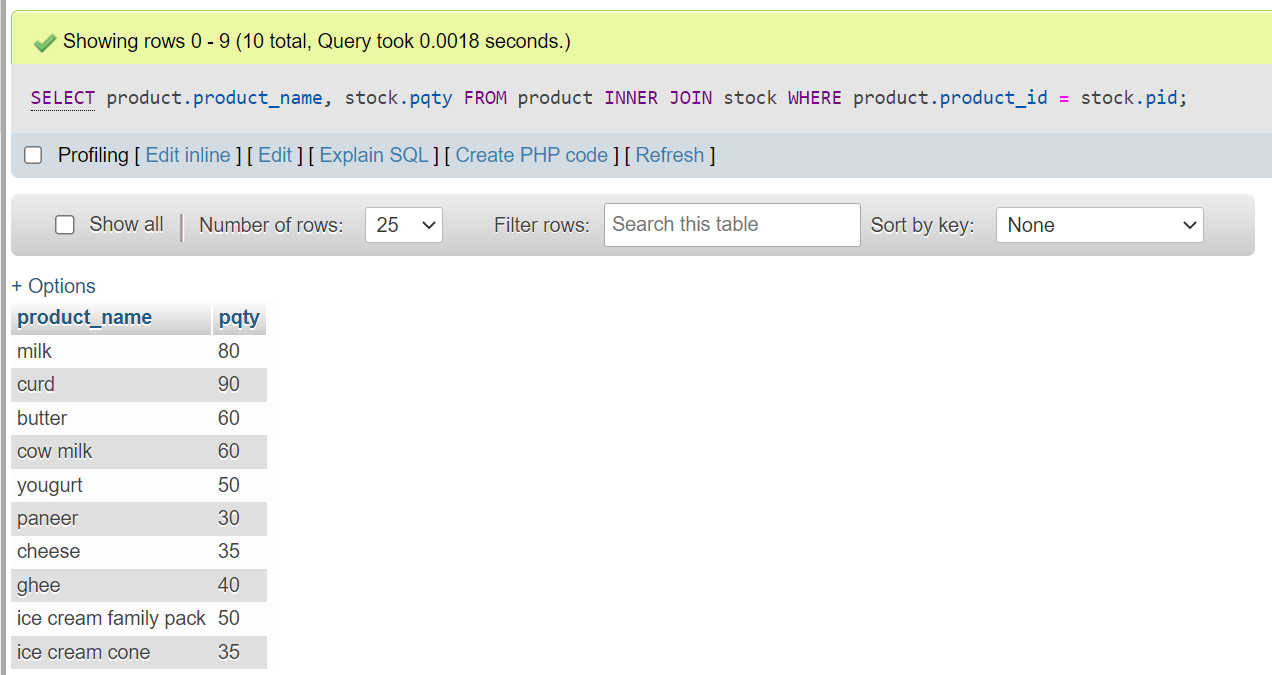


d. List quantity of all products in stock along with product details.

Query:

SELECT product.product\_name, stock.pqty FROM product INNER JOIN stock WHERE product.product\_id = stock.pid;

Screenshot:



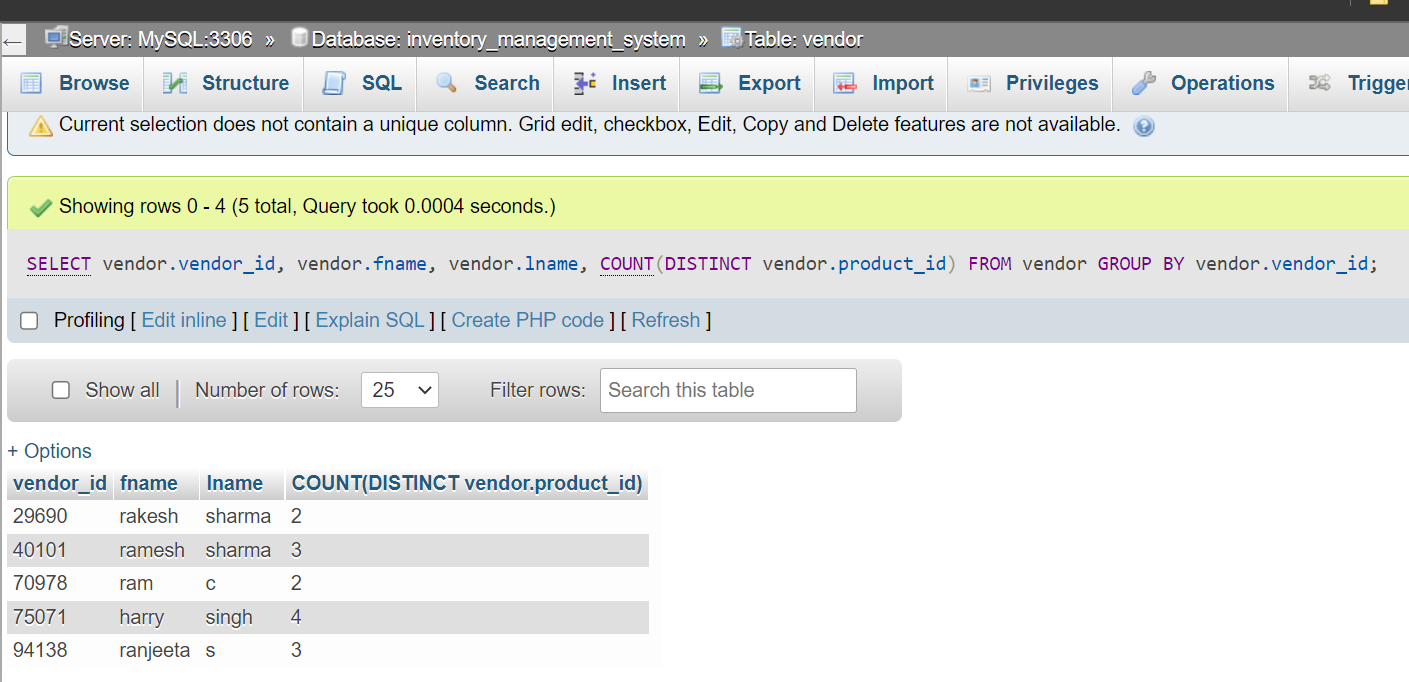
**7. Aggregate Functions:**

a. The total number of products each vendor sold.

Query:

SELECT vendor.vendor\_id, vendor.fname, vendor.lname, COUNT(DISTINCT vendor.product\_id) FROM vendor GROUP BY vendor.vendor\_id;

Screenshot:

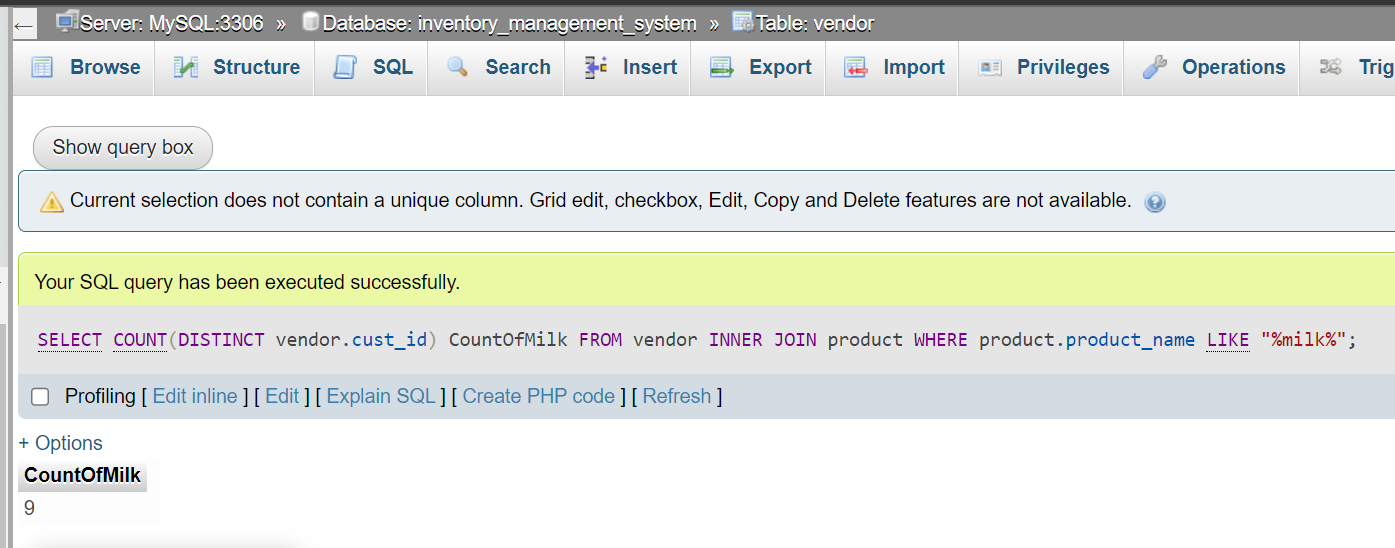


b. The number of customers who bought milk.

Query:

SELECT COUNT(DISTINCT vendor.cust\_id) CountOfMilk FROM vendor INNER JOIN product WHERE product.product\_name LIKE "%milk%";

Screenshot:

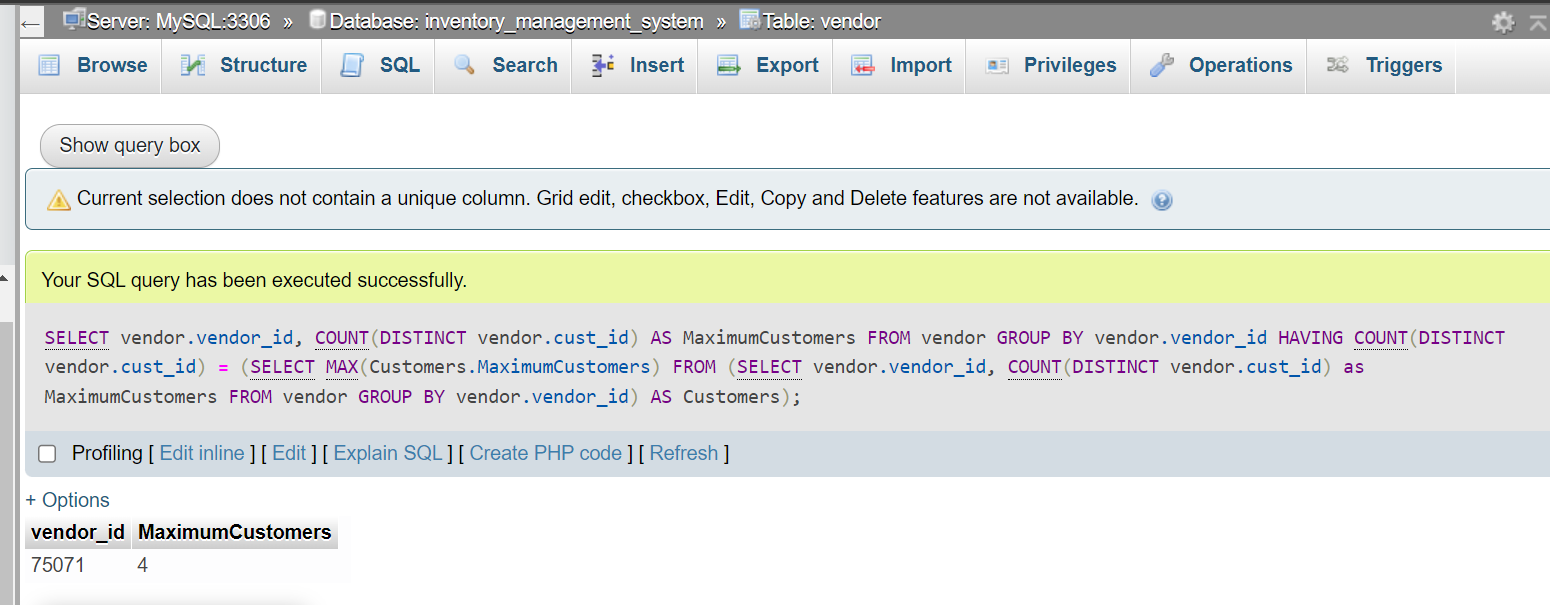


c. The vendor who sells to maximum number of customers.

Query:

SELECT vendor.vendor\_id, COUNT(DISTINCT vendor.cust\_id) AS MaximumCustomers FROM vendor GROUP BY vendor.vendor\_id HAVING COUNT(DISTINCT vendor.cust\_id) = (SELECT MAX(Customers.MaximumCustomers) FROM (SELECT vendor.vendor\_id, COUNT(DISTINCT vendor.cust\_id) as MaximumCustomers FROM vendor GROUP BY vendor.vendor\_id) AS Customers);

Screenshot:

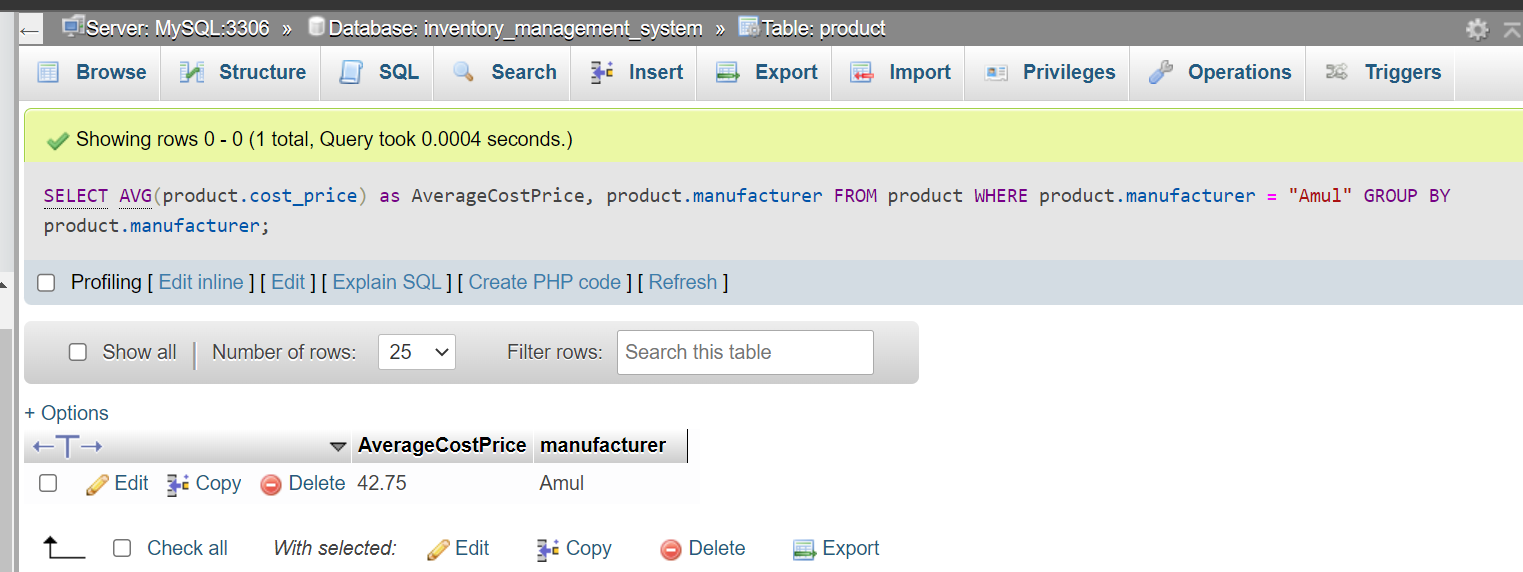


d. Average cost price of all Amul products.

Query:

SELECT AVG(product.cost\_price) AS AverageCostPrice, product.manufacturer FROM product WHERE product.manufacturer = “Amul” GROUP BY product.manufacturer;

Screenshot:



**8. Set Operations:**

a. Common products sold by vendor 75071 and 40101.

Query:

SELECT DISTINCT vendor.product\_id FROM vendor WHERE vendor.vendor\_id = 75071 AND vendor.product\_id IN (SELECT vendor.product\_id FROM vendor WHERE vendor.vendor\_id = 40101);

Screenshot:

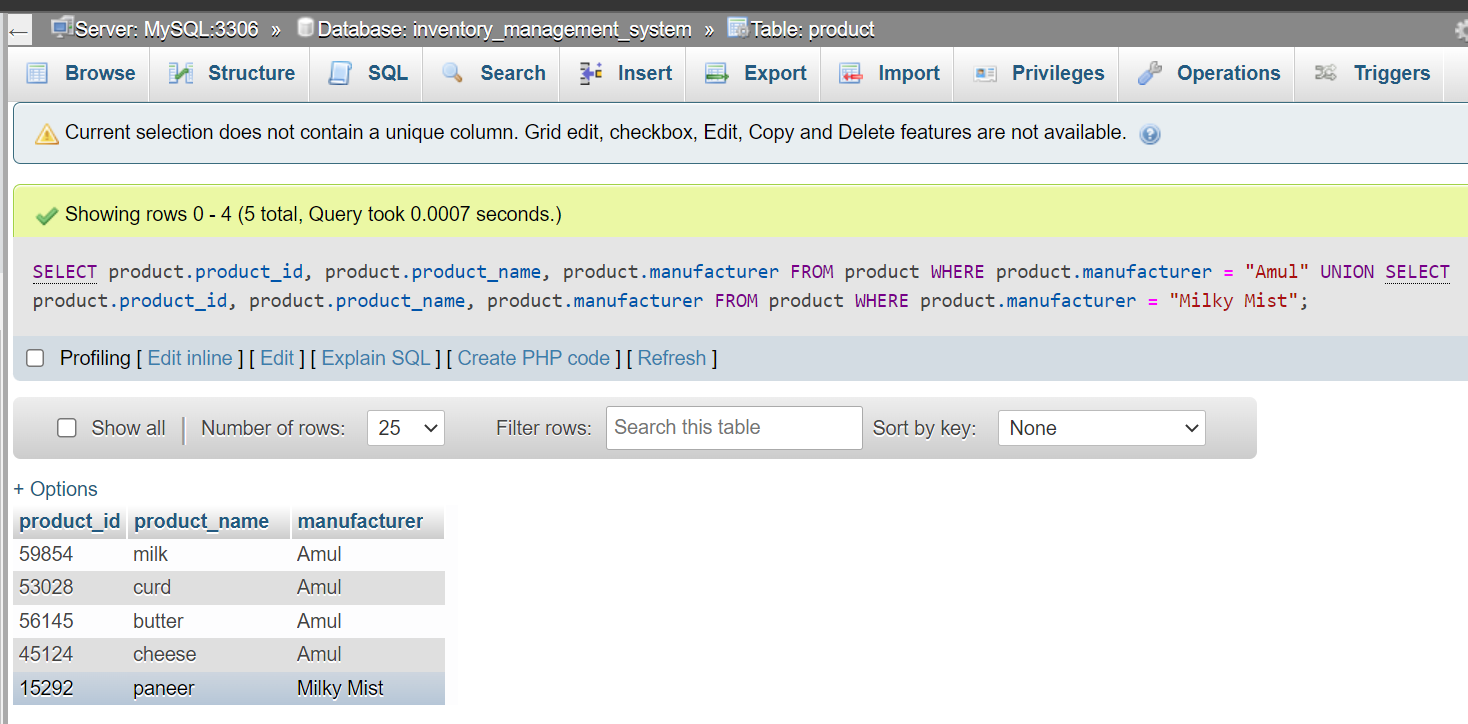


b. Products sold by Amul or Milky Mist.

Query:

SELECT product.product\_id, product.product\_name, product.manufacturer FROM product WHERE product.manufacturer = "Amul" UNION SELECT product.product\_id, product.product\_name, product.manufacturer FROM product WHERE product.manufacturer = "Milky Mist";

Screenshot:

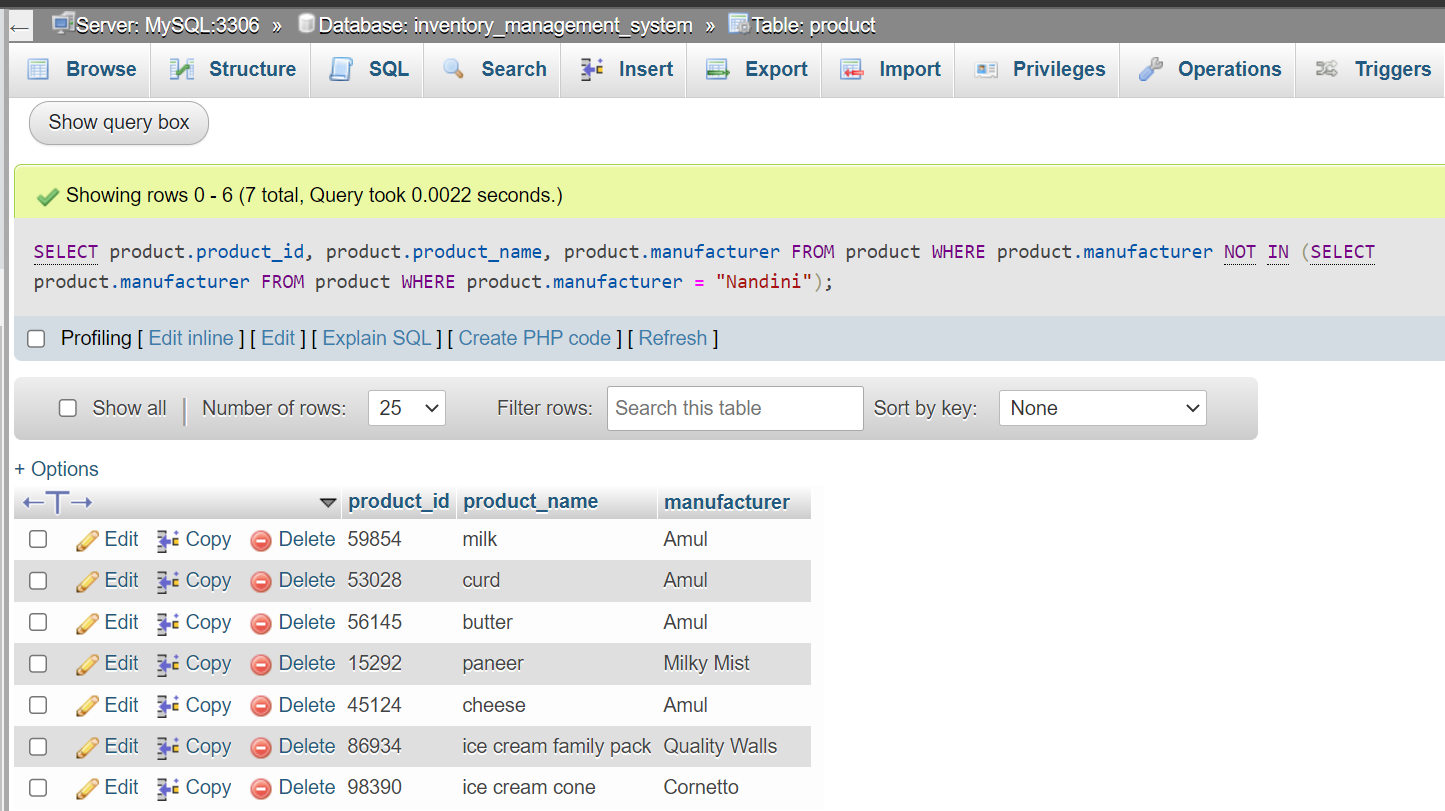


c. Products not sold by Nandini.

Query:

SELECT product.product\_id, product.product\_name, product.manufacturer FROM product WHERE product.manufacturer NOT IN (SELECT product.manufacturer FROM product WHERE product.manufacturer = "Nandini");

Screenshot:

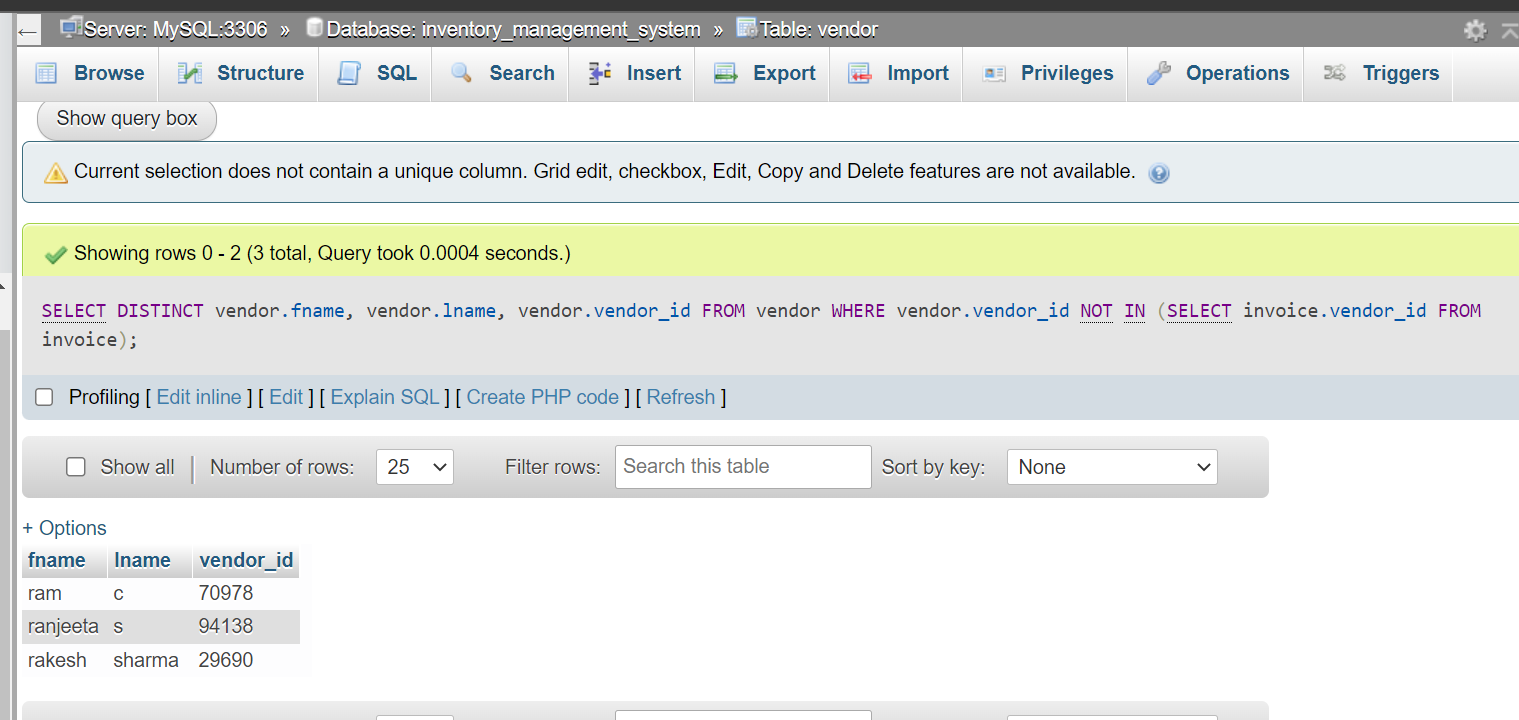


d. Vendors who don’t have any sold products in invoice.

Query:

SELECT DISTINCT vendor.fname, vendor.lname, vendor.vendor\_id FROM vendor WHERE vendor.vendor\_id NOT IN (SELECT invoice.vendor\_id FROM invoice);

Screenshot:



**9. Functions and Procedures:**

a. Function which shows which stocks need to be re-filled ie if the stock quantity is less than 50

Function creation:

DELIMITER $

CREATE FUNCTION check\_stock(qty int)

RETURNS VARCHAR(50)

BEGIN

DECLARE refill VARCHAR(50);

IF qty < 50 THEN

SET refill = "re-fill needed";

ELSE

SET refill = "no re-fill needed";

END IF;

RETURN refill;

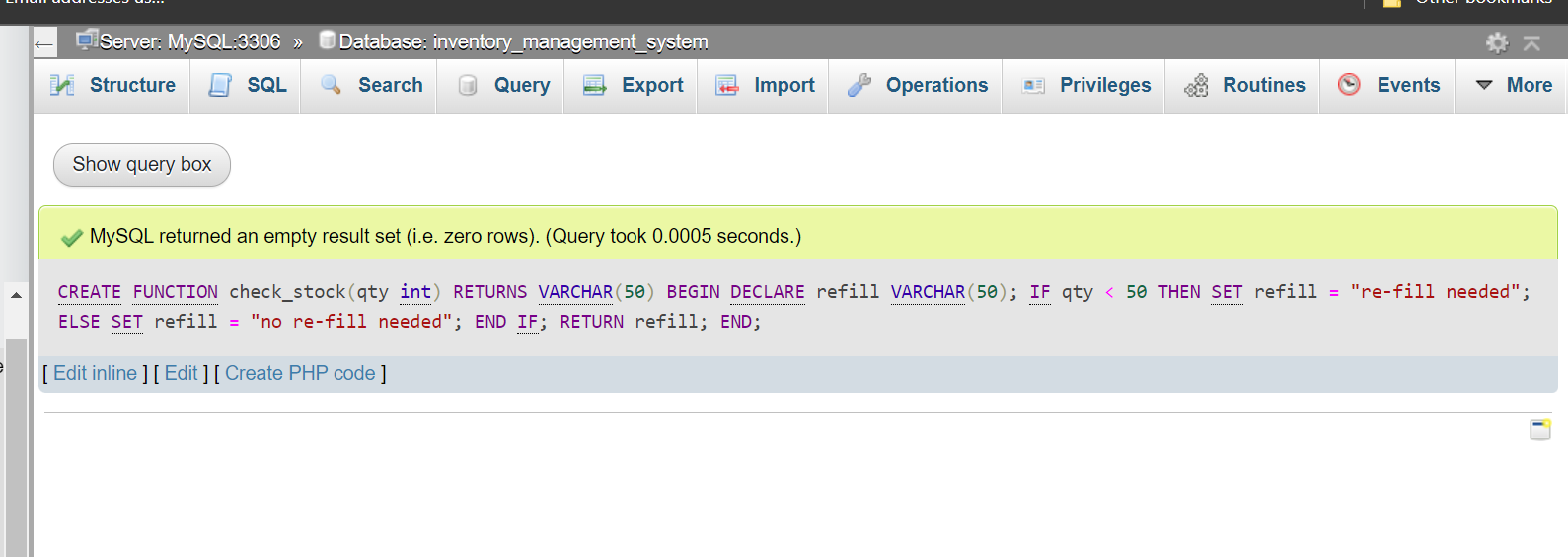
END $

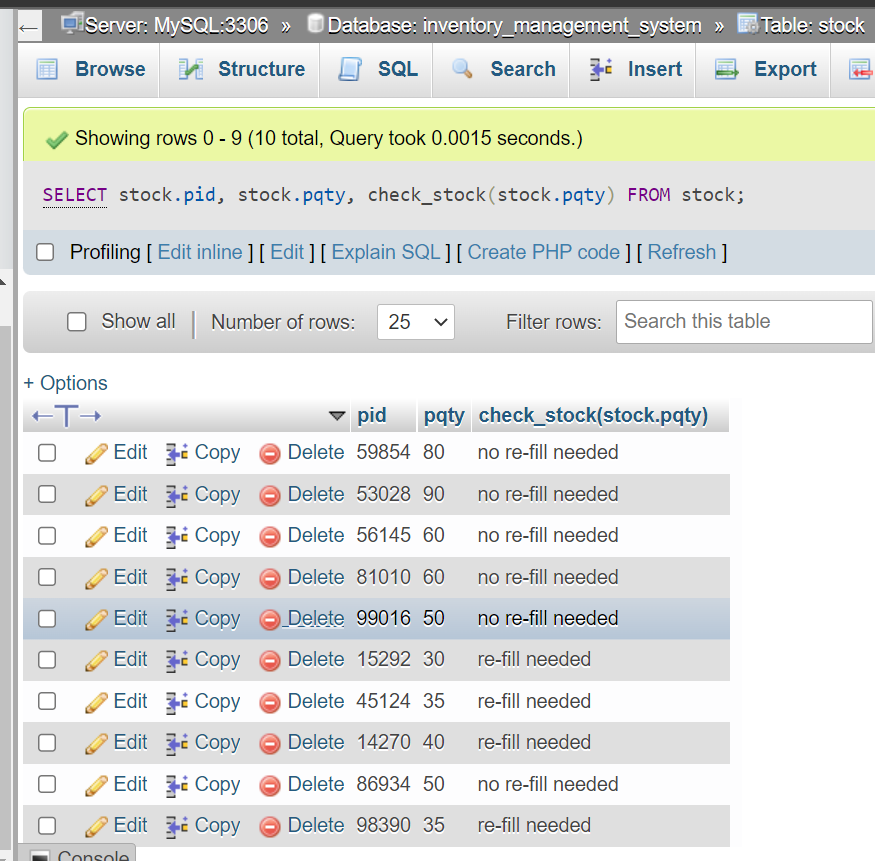
DELIMITER ;

Function call:

SELECT stock.pid, stock.pqty, check\_stock(stock.pqty) FROM stock;

Screenshots:





b. Procedure to update the product quantity to given value and purchase date to current date if the quantity is less than 50.

Procedure creation:

DELIMITER $

CREATE PROCEDURE update\_qty(IN qty int)

BEGIN

UPDATE stock SET stock.pqty = qty, stock.purchase\_date = CURRENT\_DATE WHERE stock.pqty < 50;

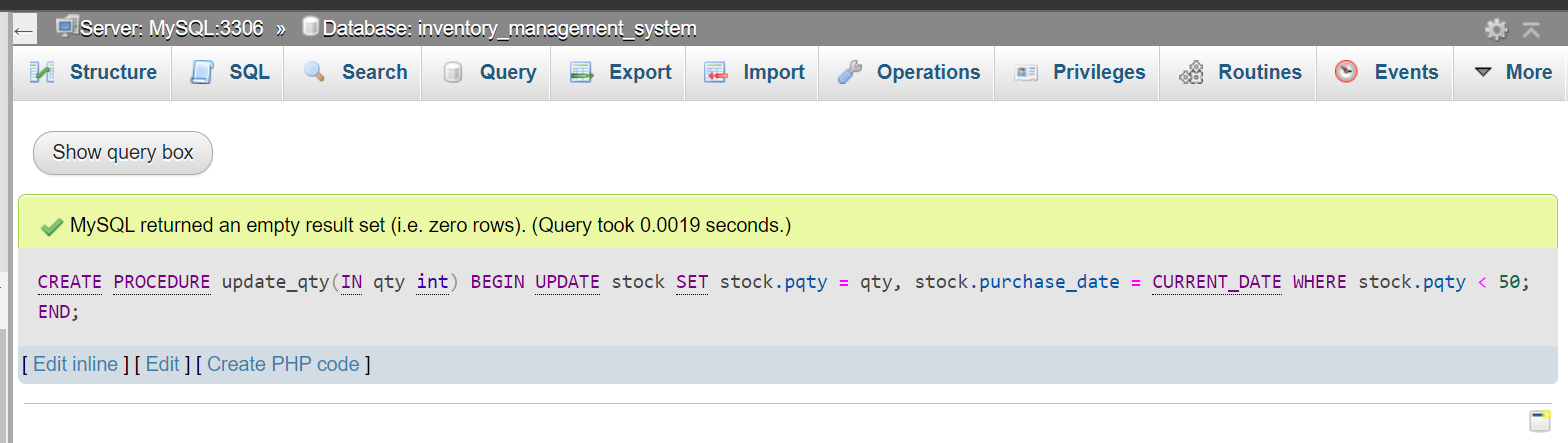
END $

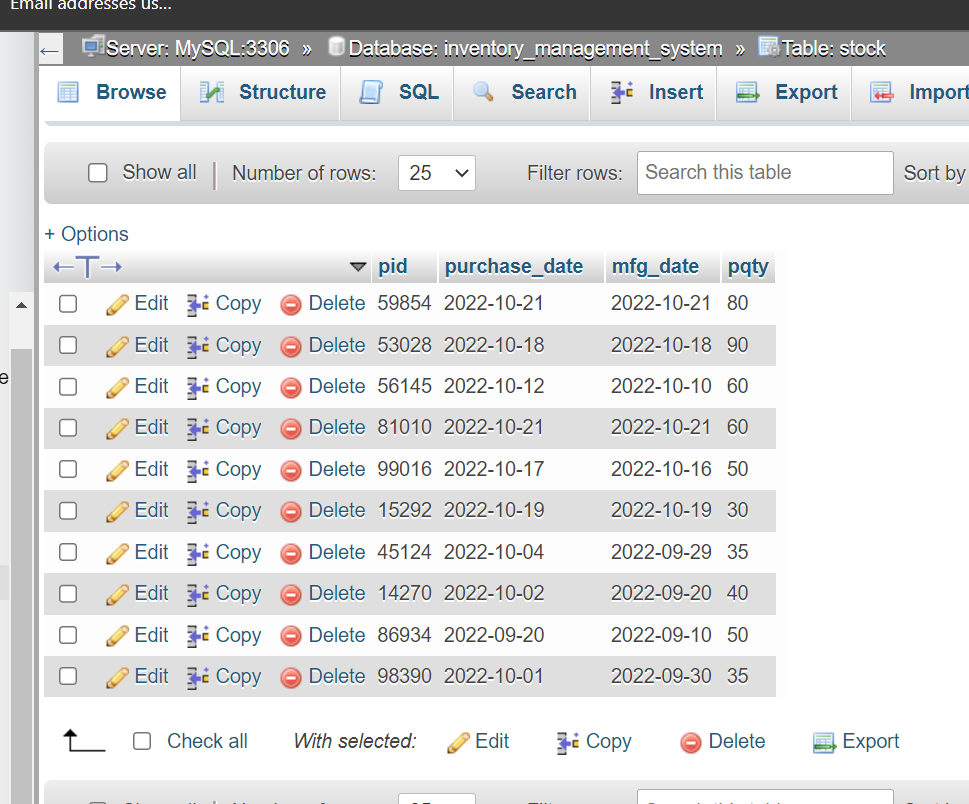
DELIMITER ;

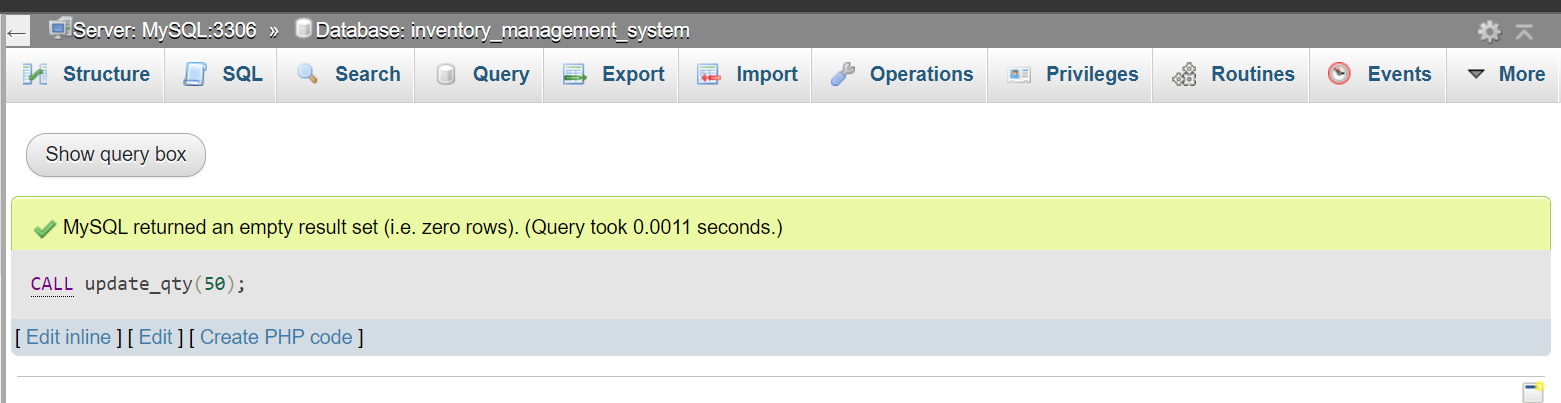
Procedure call:

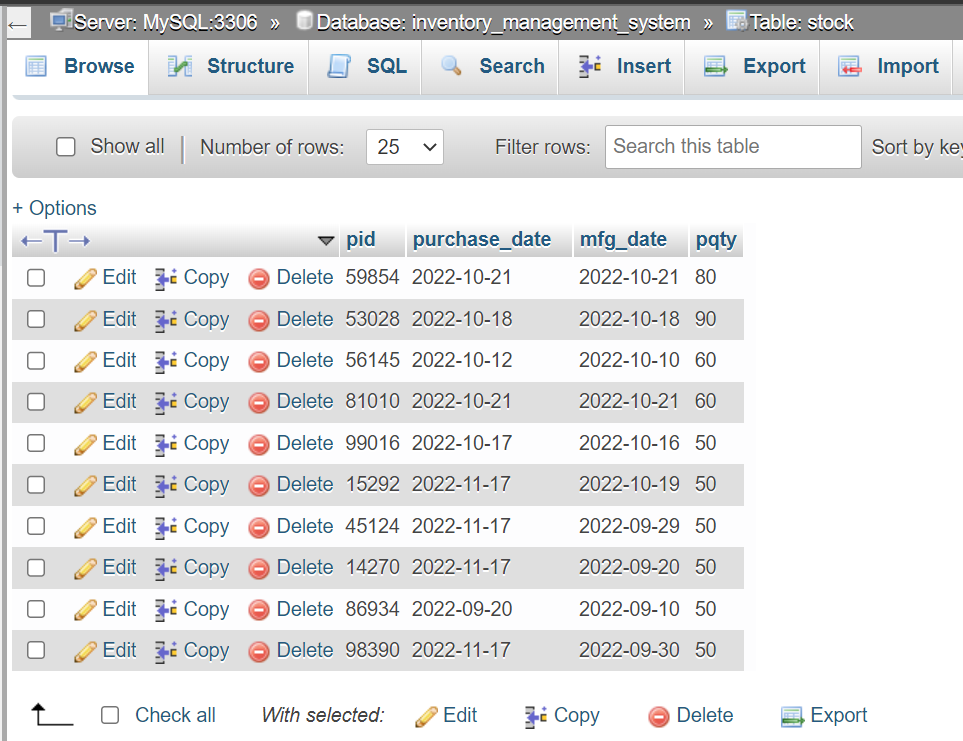
CALL update\_qty(50);

Screenshots:









**10. Triggers and Cursors:**

a. If manufacturing or purchase date is greater than the current date and manufacturing date should be less than purchase date then show an error.

Trigger creation:

DELIMITER $

CREATE TRIGGER on\_insert\_stock

BEFORE INSERT

ON stock FOR EACH ROW

BEGIN

DECLARE err\_msg1 varchar(100);

DECLARE err\_msg2 varchar(100);

DECLARE err\_msg3 varchar(100);

SET err\_msg1 = "Mfg/Purchase Date must be before the current date";

SET err\_msg2 = "Mfg Date should be before Purchase Date";

SET err\_msg3 = "Product quantity should be minimum 50";

IF (new.purchase\_date > CURRENT\_DATE OR new.mfg\_date > CURRENT\_DATE) THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = err\_msg1;

ELSEIF new.mfg\_date > new.purchase\_date THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = err\_msg2;

ELSEIF new.pqty < 50 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = err\_msg3;

END IF;

END $

DELIMITER ;

Incorrect insert:

INSERT INTO stock VALUES (86934, '2022-11-01', '2022-12-01', 50);

INSERT INTO stock VALUES (86934, '2022-12-11', '2022-12-01', 50);

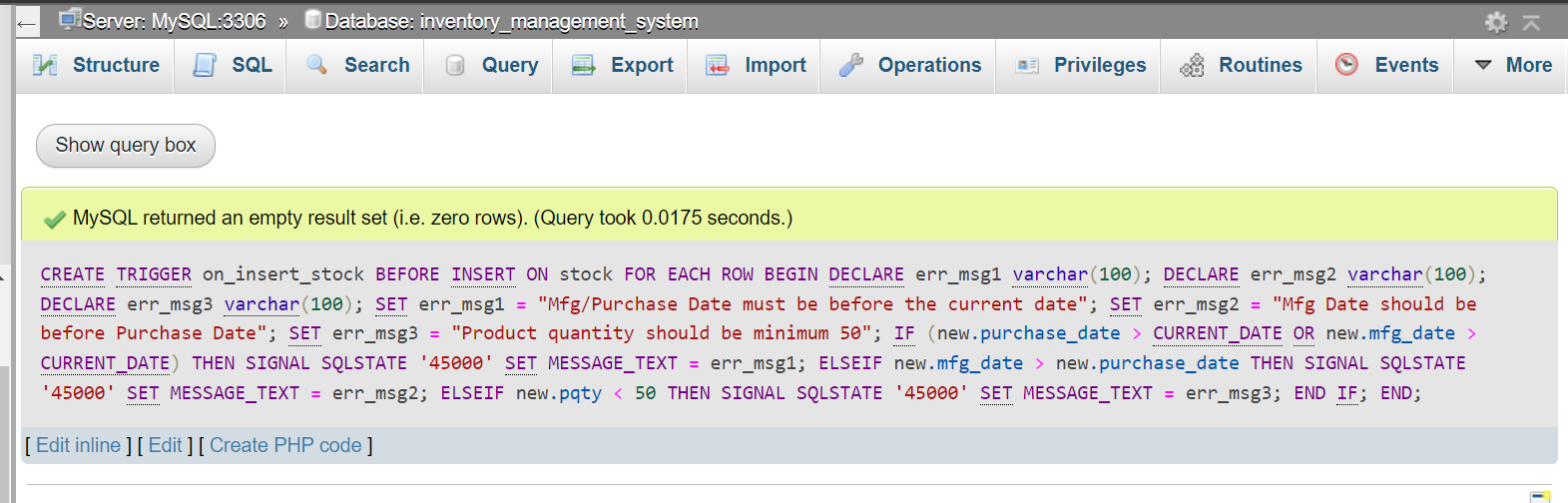
INSERT INTO stock VALUES (86934, '2022-10-01', '2022-11-01', 50);

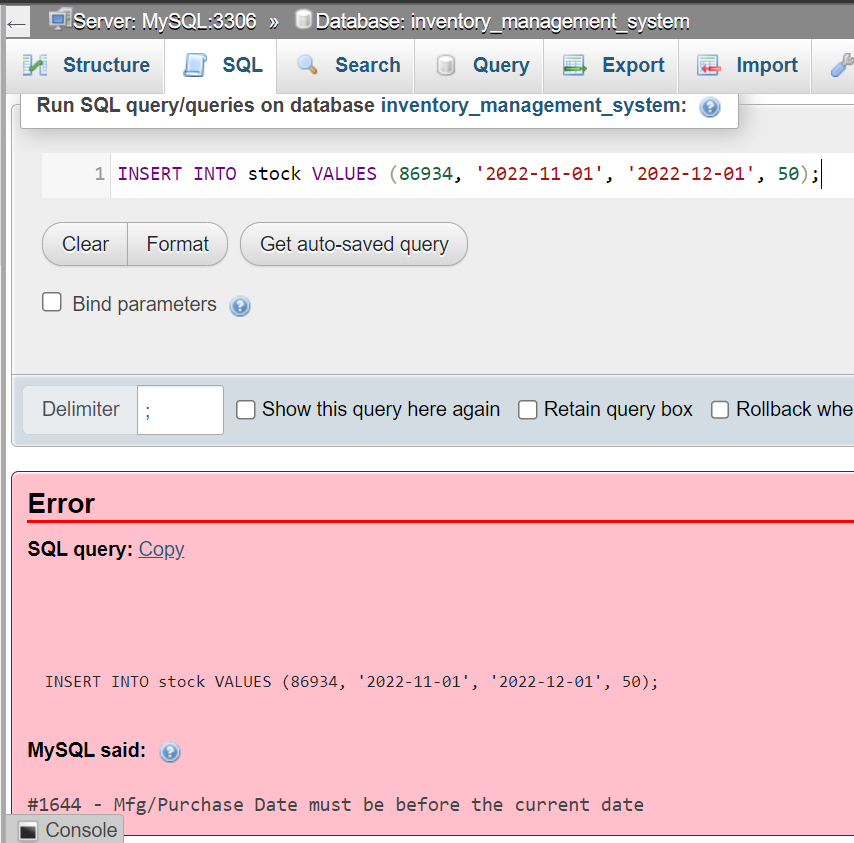
INSERT INTO stock VALUES (86934, '2022-11-10', '2022-11-01', 5);

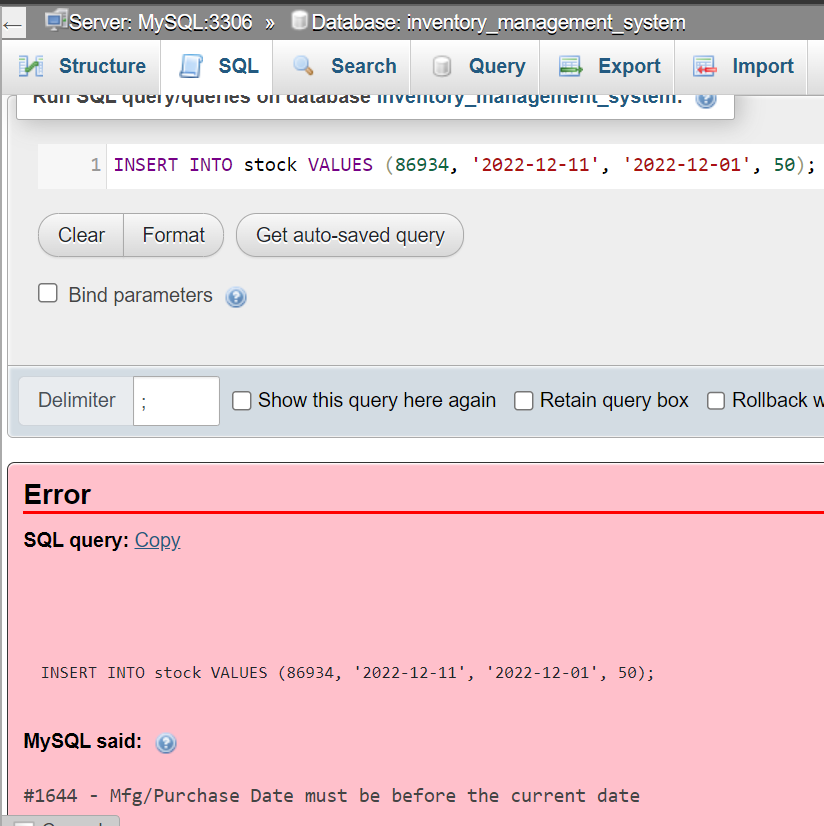
Correct insert:

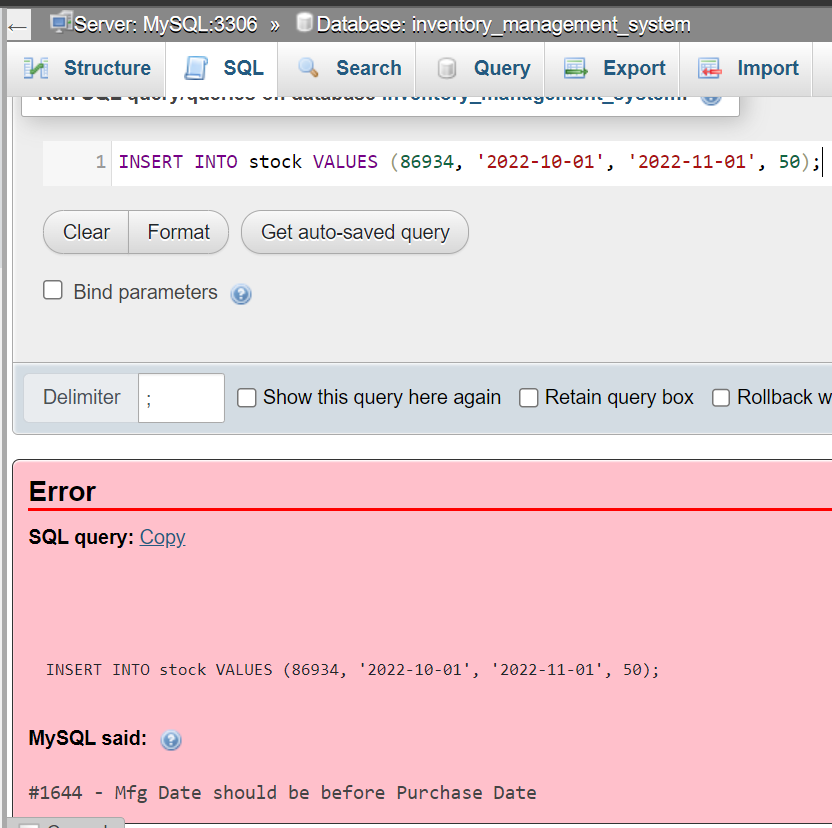
INSERT INTO stock VALUES (86934, '2022-11-01', '2022-11-10', 50);

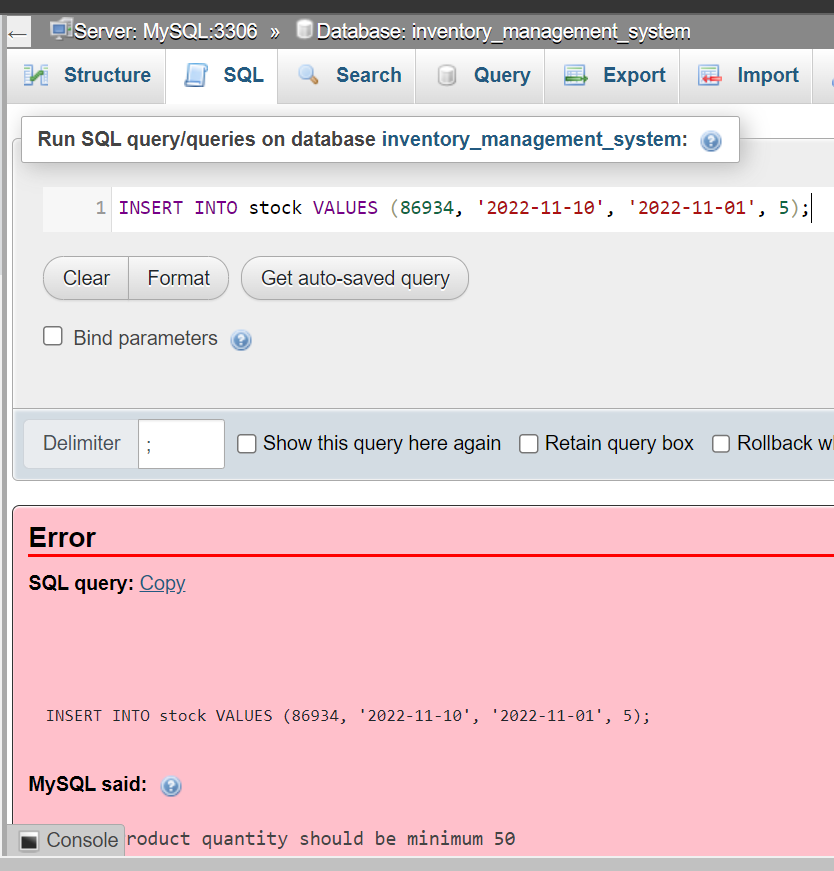
Screenshots:











b. Procedure with a Cursor to delete entries in vendor table with NULL values for product and customer that have at least one entry in the invoice table.

Procedure creation:

DELIMITER $

CREATE PROCEDURE del\_null()

BEGIN

DECLARE vid int;

DECLARE finished INTEGER DEFAULT 0;

DECLARE del\_null CURSOR FOR

SELECT DISTINCT vendor.vendor\_id FROM vendor INNER JOIN invoice WHERE vendor.vendor\_id = invoice.vendor\_id and vendor.product\_id IS NULL;

DECLARE CONTINUE HANDLER

FOR NOT FOUND SET finished = 1;

OPEN del\_null;

getRows: LOOP

FETCH del\_null INTO vid;

IF finished THEN

LEAVE getRows;

END IF;

DELETE FROM vendor WHERE vendor.vendor\_id = vid AND vendor.product\_id IS NULL;

END LOOP;

CLOSE del\_null;

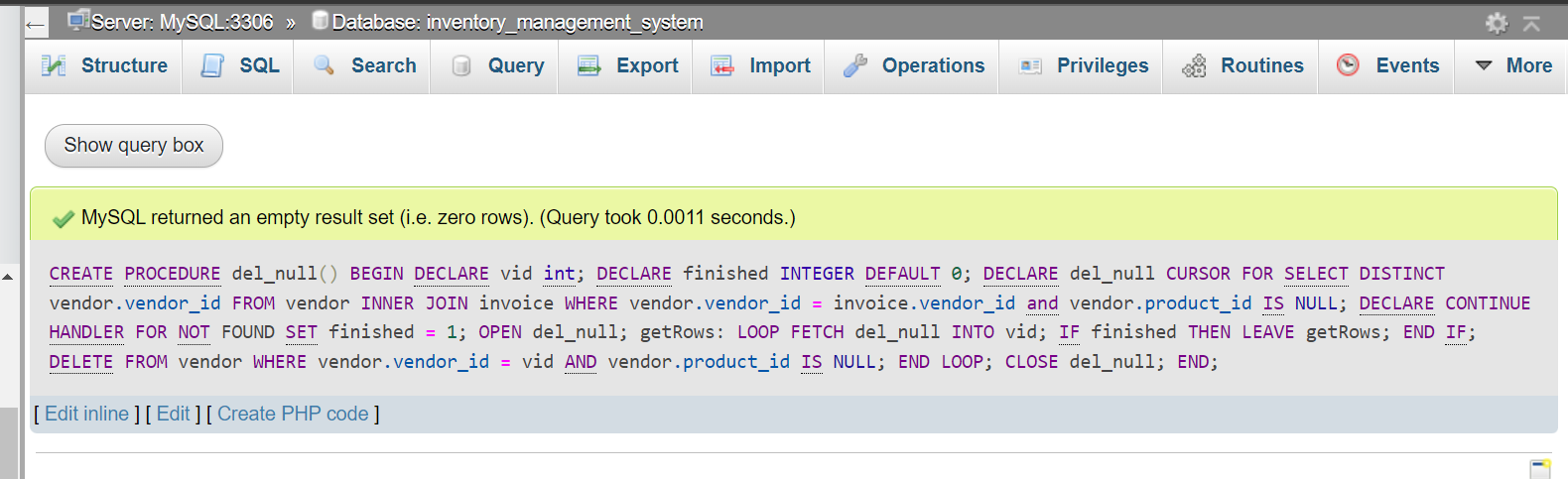
END $

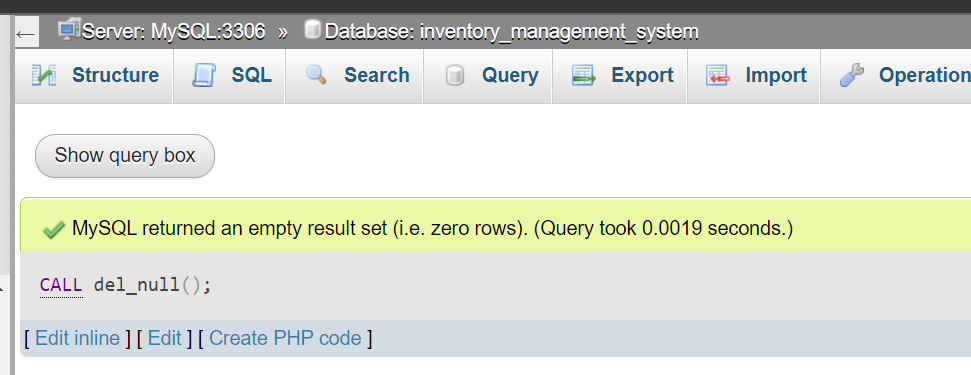
DELIMITER ;

Procedure Call:

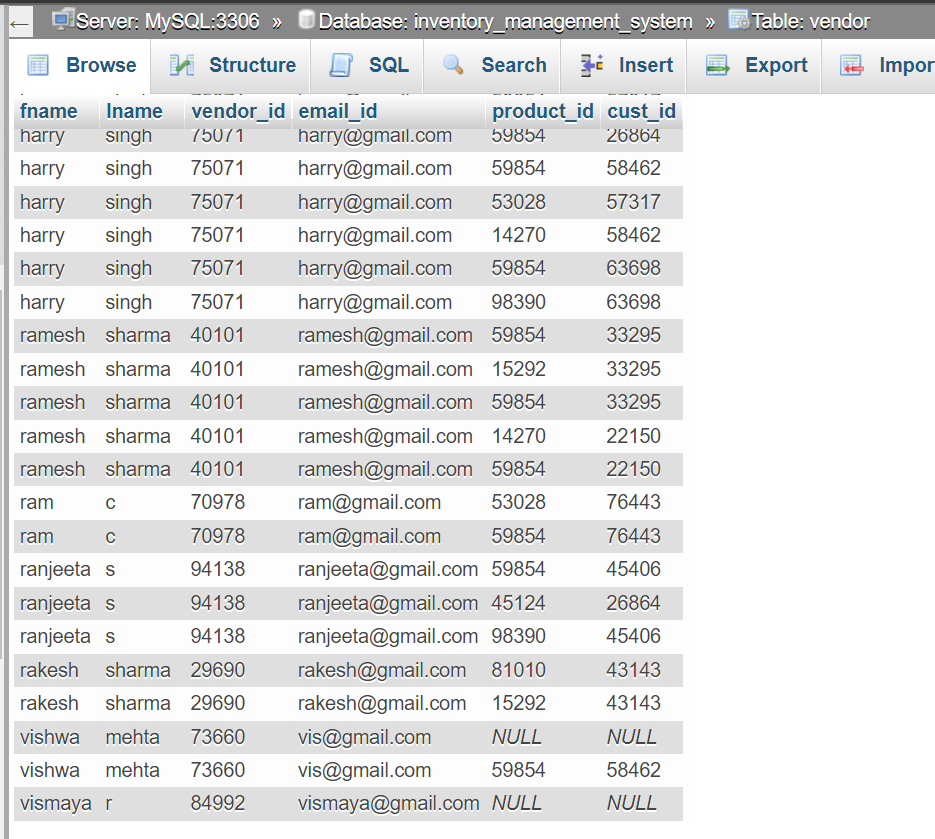
CALL del\_null();

Screenshots:

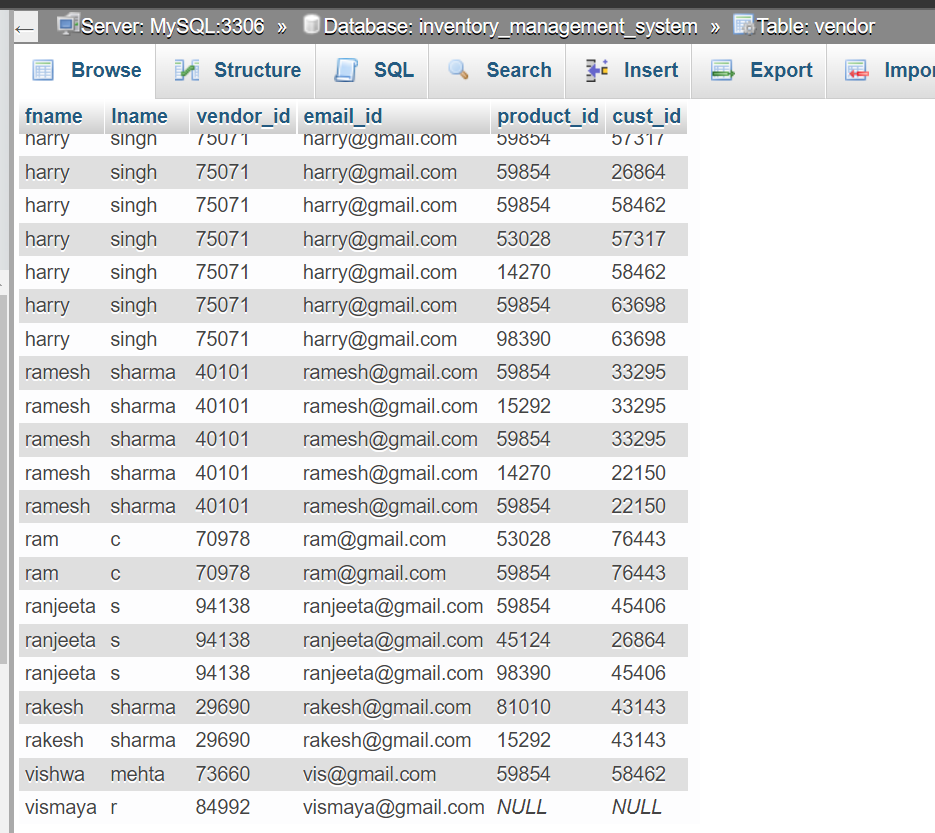




Before procedure call:



After procedure call:



**11. Developing a Frontend**

1. app.py

import streamlit as st

from database import check\_login, add\_login

def main():

    st.title("Inventory Management System")

    col1, col2 = st.columns(2)

    with col1:

        st.subheader("Login")

        email0 = st.text\_input("Email: ")

        password0 = st.text\_input("Password: ", type = "password")

        if st.button("Login"):

            check\_login(email0, password0)

    with col2:

        st.subheader("SignUp:")

        fname = st.text\_input("First Name:")

        lname = st.text\_input("Last Name")

        email = st.text\_input("Email:")

        password = st.text\_input("Password:", type = "password")

        if st.button("Signup"):

            add\_login(fname, lname, email, password)

if \_\_name\_\_ == '\_\_main\_\_':

    main()

2. database.py

import streamlit as st

import mysql.connector

import random

mydb = mysql.connector.connect(

    host = "localhost",

    user = "root",

    password = "",

    database = "inventory\_management\_system"

)

c = mydb.cursor()

from reload import reload

def check\_login(email, password):

    # print(email)

    values = email

    c.execute("SELECT \* FROM login WHERE email\_id = %s", (values,))

    passws = c.fetchall()

    for passw in passws:

        if passw[1] == password:

            reload("home")

            break

        else:

            st.error("Incorrect credentials, please try again.")

def add\_login(fn, ln, email, password):

    values = email

    c.execute("SELECT \* FROM login WHERE email\_id = %s", (values,))

    x = c.fetchall()

    if x != []:

        print(x)

        st.error("User already exists :/")

    else:

        c.execute("INSERT INTO login (email\_id, pass) VALUES (%s,%s)", (email, password))

        vid = random.randint(10000, 99999)

        c.execute("INSERT INTO vendor (fname, lname, vendor\_id, email\_id) VALUES (%s, %s, %s, %s) ", (fn, ln, vid, email))

        st.success("Successfully Added User")

    # else:

    #     st.error("User already exists :/")

def update\_password(mail, old\_pass, new\_pass):

    c.execute("UPDATE login SET pass = %s WHERE email\_id = %s AND pass = %s", (new\_pass, mail, old\_pass))

    if c.rowcount is not None:

        st.success("Password updated successfully!")

def view\_stock():

    c.execute("SELECT \* FROM stock")

    data = c.fetchall()

    return data

def add\_stock(pid, pdate, mdate, qty):

    c.execute("INSERT INTO stock VALUES (%s, %s, %s, %s)", (pid, pdate, mdate, qty))

    st.success("Values inserted")

def prod\_not\_in\_stock():

    c.execute("SELECT product.product\_id FROM product WHERE product.product\_id NOT IN (SELECT stock.pid FROM stock)")

    data = c.fetchall()

    return data

def prod\_in\_stock():

    c.execute("SELECT product.product\_id FROM product WHERE product.product\_id IN (SELECT stock.pid FROM stock)")

    data = c.fetchall()

    return data

def get\_product(pid):

    c.execute("SELECT \* FROM product WHERE product.product\_id = {}".format(pid))

    data = c.fetchall()

    print(data)

    return data

def get\_stock(pid):

    c.execute("SELECT \* FROM stock WHERE stock.pid = {}".format(pid))

    data = c.fetchall()

    print(data)

    return data

def update\_stock(pid, pdate, mdate, qty):

    c.execute("UPDATE stock SET purchase\_date = %s, mfg\_date = %s, pqty = %s WHERE pid = %s", (pdate, mdate, qty, pid))

    st.success("Stock updated successfully")

def remove\_stock(pid):

    c.execute("DELETE FROM stock WHERE stock.pid = {}".format(pid))

    st.success("Product Deleted successfully")

def view\_product():

    c.execute("SELECT \* FROM product")

    data = c.fetchall()

    return data

def add\_product(pid, pname, cp, mfg, mrp):

    c.execute("INSERT INTO product VALUES (%s, %s, %s, %s, %s)", (pid, pname, cp, mfg, mrp))

    st.success("Inserted Product")

def get\_product\_names():

    c.execute("SELECT product.product\_name FROM product")

    data = c.fetchall()

    return data

def get\_product\_id(pname):

    value = pname

    c.execute("SELECT product.product\_id FROM product WHERE product\_name = %s", (value,))

    data = c.fetchall()

    return data[0][0]

def get\_prod\_mrp(pid):

    value = pid

    c.execute("SELECT product.mrp FROM product WHERE product.product\_id = %s", (value,))

    data = c.fetchall()

    return data[0][0]

def update\_prod\_price(pid, cp):

    c.execute("UPDATE product SET product.cost\_price = %s WHERE product.product\_id = %s", (cp, pid))

    st.success("Cost price updated")

def view\_customer():

    c.execute("SELECT \* FROM customer")

    data = c.fetchall()

    return data

def add\_customer(cid, pn, mail, cname, sno, sname, pin):

    c.execute("INSERT INTO customer VALUES (%s, %s, %s, %s, %s, %s, %s)", (cid, pn, mail, cname, sno, sname, pin))

    st.success("Inserted Customer")

def get\_customer(cid):

    c.execute("SELECT \* FROM customer WHERE customer.cust\_id = {}".format(cid))

    data = c.fetchall()

    print(data)

    return data

def get\_cust\_names():

    c.execute("SELECT customer.cust\_name FROM customer")

    data = c.fetchall()

    return data

def get\_cust\_ids(cname):

    value = cname

    c.execute("SELECT customer.cust\_id FROM customer WHERE customer.cust\_name = %s", (value,))

    data = c.fetchall()

    return data

def update\_customer(cid, phno, email, cname, stno, stname, pin):

    c.execute("UPDATE customer SET phone\_no = %s, cust\_email\_id = %s, cust\_name = %s, str\_no = %s, str\_name = %s, pincode = %s WHERE cust\_id = %s", (phno, email, cname, stno, stname, pin, cid))

    st.success("Successfully Updated Customer")

def view\_invoice():

    c.execute("SELECT \* FROM invoice")

    data = c.fetchall()

    return data

def get\_vendor\_id(mail):

    print(mail)

    c.execute("SELECT DISTINCT vendor.vendor\_id FROM vendor WHERE vendor.email\_id = %s", (mail, ))

    data = c.fetchall()

    print("data", data[0][0])

    return data[0][0]

def add\_invoice(pid, ino, idate, sp, pqty, dis, phno, vid):

    c.execute("INSERT INTO invoice VALUES (%s, %s, %s, %s, %s, %s, %s, %s)", (pid, ino, idate, sp, pqty, dis, phno, vid))

    c.execute("SELECT customer.cust\_id FROM customer WHERE customer.phone\_no = %s", (phno,))

    cid = c.fetchall()[0][0]

    c.execute("SELECT DISTINCT vendor.fname, vendor.lname, vendor.vendor\_id, vendor.email\_id FROM vendor WHERE vendor.vendor\_id = %s", (vid,))

    d = c.fetchall()

    fn, ln, vid0, mail = d[0][0], d[0][1], d[0][2], d[0][3]

    c.execute("INSERT INTO vendor VALUES (%s, %s, %s, %s, %s, %s)", (fn, ln, vid0, mail, pid, cid))

    st.success("Invoice inserted successfully")

def get\_invoice(iid):

    c.execute("SELECT \* FROM invoice WHERE invoice.invoice\_no = {}".format(iid))

    data = c.fetchall()

    return data

def get\_phnos():

    c.execute("SELECT customer.phone\_no FROM customer")

    data = c.fetchall()

    return data

3. home.py

import streamlit as st

from database import \*

import pandas as pd

import random

st.title("Welcome to ListAll")

def main():

    menu = ["View Stock", "Add item to stock", "Update Stock", "Remove Stock", "View Products", "Add Product", "Update Product Price", "View Customers", "Add Customer", "Update Customer", "View Invoice", "Add Invoice", "Change Password"]

    choice = st.sidebar.selectbox("Menu", menu)

    if choice == "View Stock":

        st.subheader("Current Stock:")

        data = view\_stock()

        df = pd.DataFrame(data, columns = ['pid', 'purchase\_date', 'mfg\_date', 'pqty'])

        st.dataframe(df)

    elif choice == "Add item to stock":

        st.subheader("Add New Product to Current Stock:")

        list\_of\_products = [i[0] for i in prod\_not\_in\_stock()]

        prod = st.selectbox("Products that can be added to stock", list\_of\_products)

        result = get\_product(prod)

        if result:

            pid = st.text\_input("Product ID:", result[0][0])

            pname = st.text\_input("Product Name:", result[0][1])

            pur\_date = st.text\_input("Purchase Date(YYYY-MM-DD):")

            mfg\_date = st.text\_input("Manufacturing Date(YYYY-MM-DD):")

            qty = st.text\_input("Product Quantity:")

            if st.button("Add"):

                add\_stock(pid, pur\_date, mfg\_date, qty)

        with st.expander("View Updated table"):

            data = view\_stock()

            df = pd.DataFrame(data, columns = ['pid', 'purchase\_date', 'mfg\_date', 'pqty'])

            st.dataframe(df)

    elif choice == "Update Stock":

        st.subheader("Update Stock:")

        list\_of\_products = [i[0] for i in prod\_in\_stock()]

        prod = st.selectbox("Products that can be updated in stock", list\_of\_products)

        result = get\_stock(prod)

        res = get\_product(prod)

        # print(result[0])

        if result:

            # print(type(result[0][1]), type(result[0][2]))

            pid = result[0][0]

            pdate = result[0][1]

            mdate = result[0][2]

            pqty = result[0][3]

            col1, col2= st.columns(2)

            with col1:

                pid0 = st.text\_input("Product ID:", pid, disabled = True)

                new\_pdate = st.text\_input("Purchase Date(YYYY-MM-DD):", pdate)

            with col2:

                pname = st.text\_input("Product Name:", res[0][1], disabled = True)

                new\_mdate = st.text\_input("Manufacturing Date(YYYY-MM-DD):", mdate)

            new\_pqty = st.text\_input("Product Quantity:", pqty)

            if st.button("Update"):

                update\_stock(pid0, new\_pdate, new\_mdate, new\_pqty)

        with st.expander("View Updated table"):

            data = view\_stock()

            df = pd.DataFrame(data, columns = ['pid', 'purchase\_date', 'mfg\_date', 'pqty'])

            st.dataframe(df)

    elif choice == "Remove Stock":

        st.subheader("Remove Stock:")

        list\_of\_products = [i[0] for i in prod\_in\_stock()]

        prod = st.selectbox("Products that can be added to stock", list\_of\_products)

        result = get\_product(prod)

        if result:

            pid = result[0][0]

            if st.button("Remove "+str(result[0][1])):

                remove\_stock(pid)

        with st.expander("View Updated table"):

            data = view\_stock()

            df = pd.DataFrame(data, columns = ['pid', 'purchase\_date', 'mfg\_date', 'pqty'])

            st.dataframe(df)

    elif choice == "View Products":

        st.subheader("Products:")

        data = view\_product()

        df = pd.DataFrame(data, columns = ['product\_id', 'product\_name', 'cost\_price', 'manufacturer', 'mrp'])

        st.dataframe(df)

    elif choice == "Add Product":

        st.subheader("Enter product details:")

        gen\_pid = random.randint(10000, 99999)

        d = get\_product(gen\_pid)

        if not d:

            print(gen\_pid)

            pid = st.text\_input("Product ID:", gen\_pid, disabled = True)

        else:

            st.error("Please reload the page.")

        pname = st.text\_input("Product Name:")

        cp = st.text\_input("Cost Price:")

        mfg = st.text\_input("Manufacturer Name:")

        mrp = st.text\_input("MRP:")

        if st.button("Add"):

            add\_product(pid, pname, cp, mfg, mrp)

        with st.expander("View Updated table"):

            data = view\_product()

            df = pd.DataFrame(data, columns = ['product\_id', 'product\_name', 'cost\_price', 'manufacturer', 'mrp'])

            st.dataframe(df)

    elif choice == "Update Product Price":

        st.subheader("Update Product:")

        list\_of\_products = [i[0] for i in get\_product\_names()]

        pname = st.selectbox("Product Name:", list\_of\_products)

        prod = get\_product\_id(pname)

        result = get\_product(prod)

        # print(result[0])

        if result:

            # print(type(result[0][1]), type(result[0][2]))

            pid = result[0][0]

            pname = result[0][1]

            cp = result[0][2]

            mfg = result[0][3]

            mrp = result[0][4]

            col1, col2= st.columns(2)

            with col1:

                pid0 = st.text\_input("Product ID:", pid, disabled = True)

                pname0 = st.text\_input("Name:", pname, disabled = True)

                mrp0 = st.text\_input("MRP:", mrp, disabled = True)

            with col2:

                new\_cp = st.text\_input("Cost Price:", cp)

                mfg0 = st.text\_input("Manufacturer Name:", mfg, disabled = True)

            if st.button("Update"):

                update\_prod\_price(pid0, new\_cp)

            with st.expander("View Updated table"):

                data = view\_product()

                df = pd.DataFrame(data, columns = ['product\_id', 'product\_name', 'cost\_price', 'manufacturer', 'mrp'])

                st.dataframe(df)

    elif choice == "View Customers":

        st.subheader("Customers:")

        data = view\_customer()

        df = pd.DataFrame(data, columns = ['cust\_id', 'phone\_no', 'cust\_email\_id', 'cust\_name', 'str\_no', 'str\_name', 'pincode'])

        st.dataframe(df)

    elif choice == "Add Customer":

        st.subheader("Enter customer details:")

        gen\_cid = random.randint(10000, 99999)

        d = get\_customer(gen\_cid)

        if not d:

            print(gen\_cid)

            cid = st.text\_input("Customer ID:", gen\_cid, disabled = True)

        else:

            st.error("Please reload the page.")

        cname = st.text\_input("Customer Name:")

        pn = st.text\_input("Phone Number:")

        mail = st.text\_input("Email ID:")

        col1, col2, col3 = st.columns(3)

        with col1:

            sno = st.text\_input("Street Number:")

        with col2:

            sname = st.text\_input("Street Name:")

        with col3:

            pin = st.text\_input("Pincode:")

        if st.button("Add"):

            add\_customer(cid, pn, mail, cname, sno, sname, pin)

        with st.expander("View Updated table"):

            data = view\_customer()

            df = pd.DataFrame(data, columns = ['cust\_id', 'phone\_no', 'cust\_email\_id', 'cust\_name', 'str\_no', 'str\_name', 'pincode'])

            st.dataframe(df)

    elif choice == "Update Customer":

        st.subheader("Update Customer:")

        list\_of\_cnames = [i[0] for i in get\_cust\_names()]

        # print(list\_of\_cnames)

        cname = st.selectbox("Select customer to be updated", list\_of\_cnames)

        # print(cname)

        cid = get\_cust\_ids(cname)

        # print(cid[0][0])

        result = get\_customer(cid[0][0])

        # print(result[0])

        if result:

            cid = result[0][0]

            phno = result[0][1]

            email = result[0][2]

            cname = result[0][3]

            stno = result[0][4]

            stname = result[0][5]

            pin = result[0][6]

            cid0 = st.text\_input("Customer ID:", cid, disabled = True)

            new\_phno = st.text\_input("Phone Number", phno)

            new\_email = st.text\_input("Email ID:", email)

            new\_cname = st.text\_input("Name:", cname)

            new\_stno = st.text\_input("Street Number:", stno)

            new\_stname = st.text\_input("Street Name:", stname)

            new\_pin = st.text\_input("Pincode:", pin)

            if st.button("Update"):

                update\_customer(cid0, new\_phno, new\_email, new\_cname, new\_stno, new\_stname, new\_pin)

        with st.expander("View Updated table"):

            data = view\_customer()

            df = pd.DataFrame(data, columns = ['cust\_id', 'phone\_no', 'cust\_email\_id', 'cust\_name', 'str\_no', 'str\_name', 'pincode'])

            st.dataframe(df)

    elif choice == "View Invoice":

        st.subheader("Invoice:")

        data = view\_invoice()

        df = pd.DataFrame(data, columns = ['prod\_id', 'invoice\_no', 'invoice\_date', 'selling\_price', 'prod\_qty', 'discount', 'phone\_no', 'vendor\_id'])

        st.dataframe(df)

    elif choice == "Add Invoice":

        st.subheader("Add Invoice:")

        list\_of\_products = [i[1] for i in view\_product()]

        prod = st.selectbox("Product Name:", list\_of\_products)

        pid = get\_product\_id(prod)

        gen\_iid = random.randint(10000, 99999)

        d = get\_invoice(gen\_iid)

        if not d:

            print(gen\_iid)

            iid = st.text\_input("Invoice ID:", gen\_iid, disabled = True)

        else:

            st.error("Please reload the page.")

        idate = st.text\_input("Invoice Date (YYYY-MM-DD):")

        dis = st.text\_input("Discount Given(%):")

        mrp = get\_prod\_mrp(pid)

        if dis:

            dis = float(dis) \* 0.01

            sp = mrp - (mrp \* dis)

            sp0 = st.text\_input("Selling Price:", sp, disabled = True)

        pqty = st.text\_input("Product Quantity:")

        list\_of\_phnos = [i[0] for i in get\_phnos()]

        pno = st.selectbox("Customer Phone Number:", list\_of\_phnos)

        mail = st.text\_input("Vendor Email ID:")

        # vid = 0

        if mail:

            vid = get\_vendor\_id(mail)

            # pass

        # phno = get\_cus\_phone(pno)

        if st.button("Add Invoice"):

            add\_invoice(pid, iid, idate, sp0, pqty, dis, pno, vid)

        with st.expander("View Updated Invoice"):

            data = view\_invoice()

            df = pd.DataFrame(data, columns = ['prod\_id', 'invoice\_no', 'invoice\_date', 'selling\_price', 'prod\_qty', 'discount', 'phone\_no', 'vendor\_id'])

            st.dataframe(df)

    elif choice == "Change Password":

        st.subheader("Enter New Credentials Below:")

        email = st.text\_input("Email:")

        old\_pass = st.text\_input("Old Password:", type = "password")

        new\_pass = st.text\_input("New Password:", type = "password")

        if st.button("Change Password"):

            update\_password(email, old\_pass, new\_pass)

    else:

        st.subheader("Home Page")

if \_\_name\_\_ == '\_\_main\_\_':

    main()

4. reload.py

import streamlit as st

import os

import keyboard

import threading

import time

wait\_second = 10

def threadFunc():

   time.sleep(wait\_second)

def reload(page):

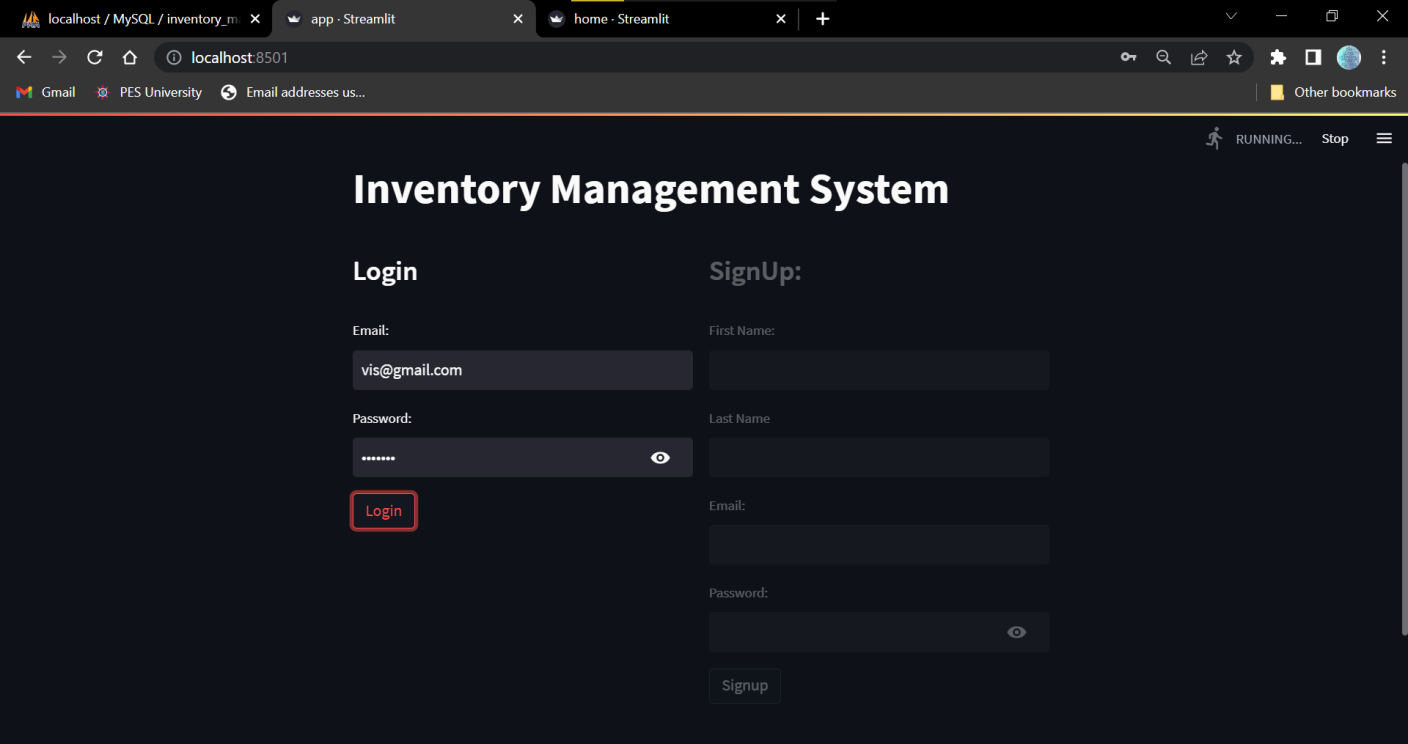
    th = threading.Thread(target=threadFunc)

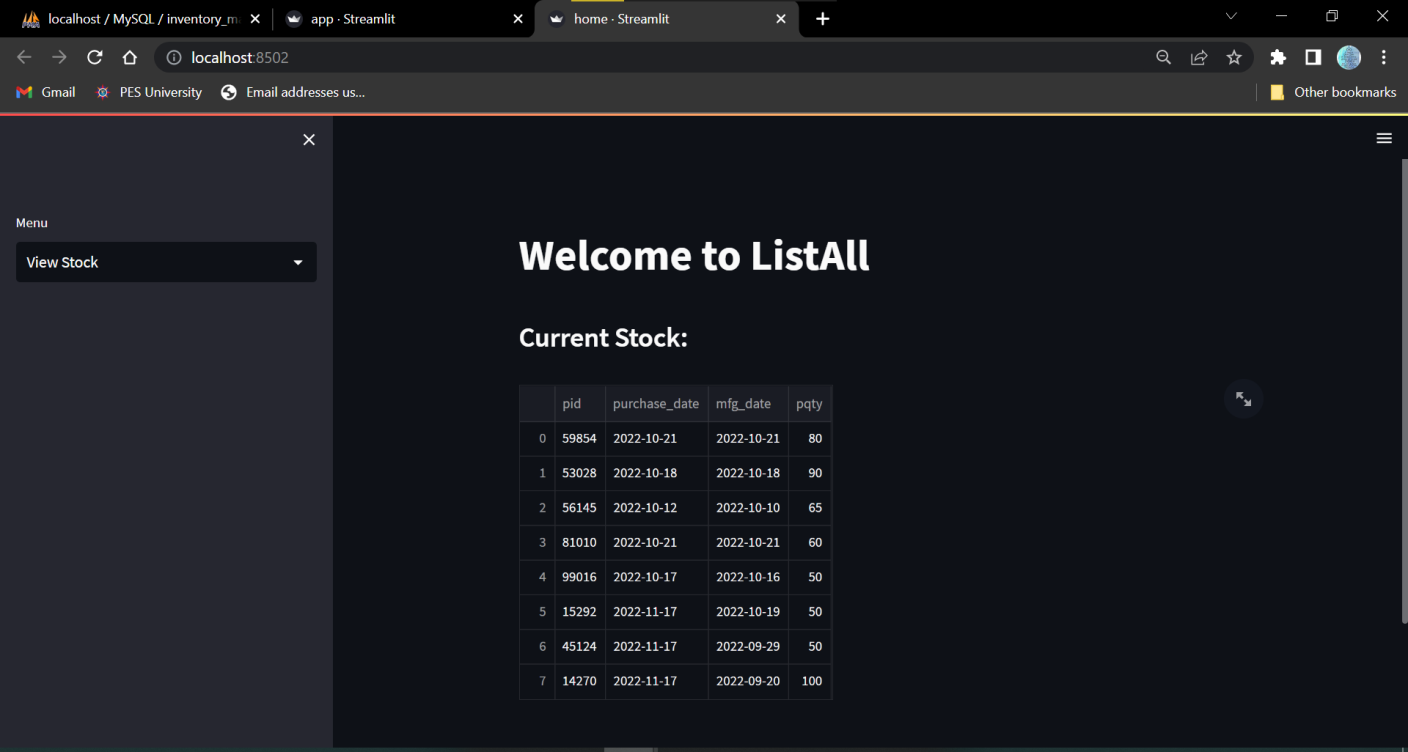
    th.start()

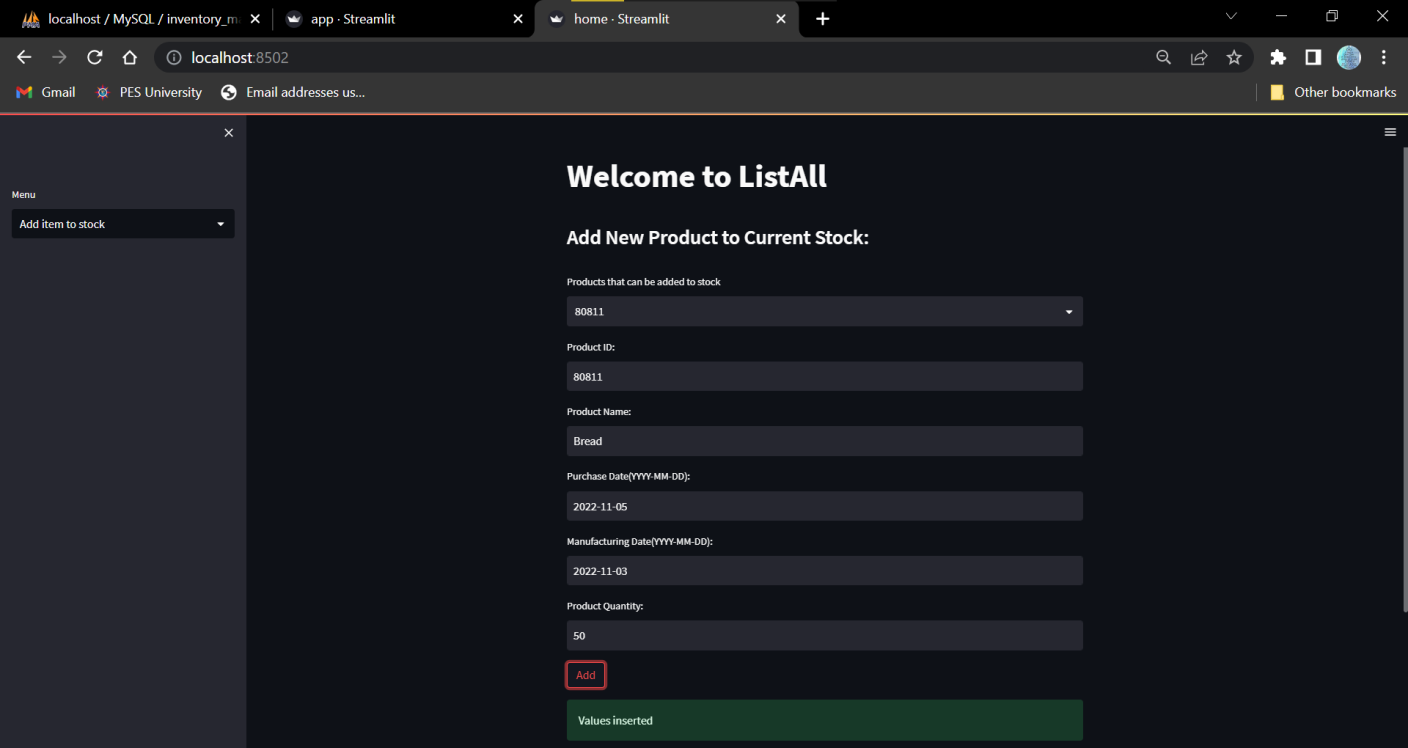
    os.system(r"streamlit run " + page + ".py")

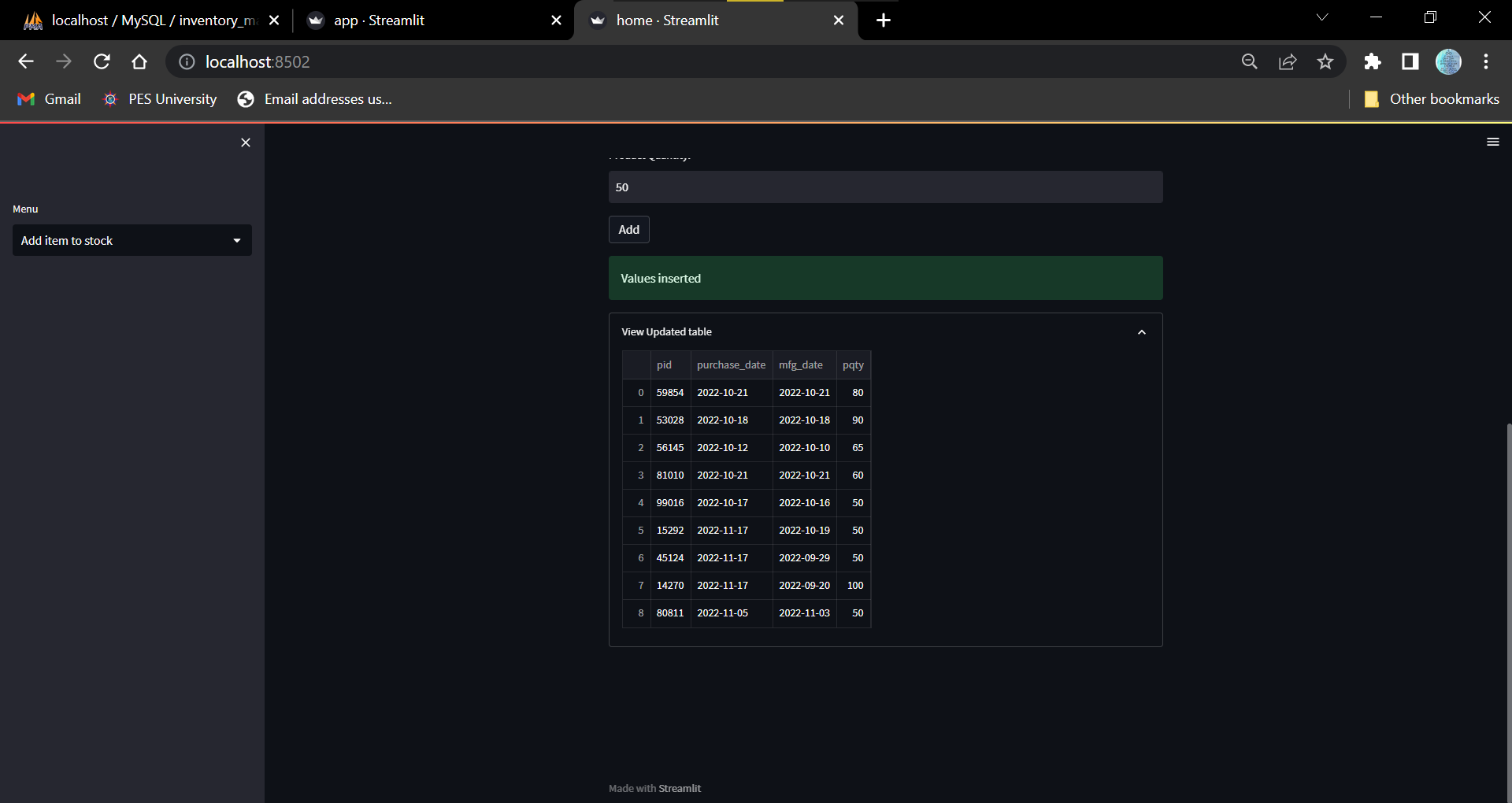
    th.join()

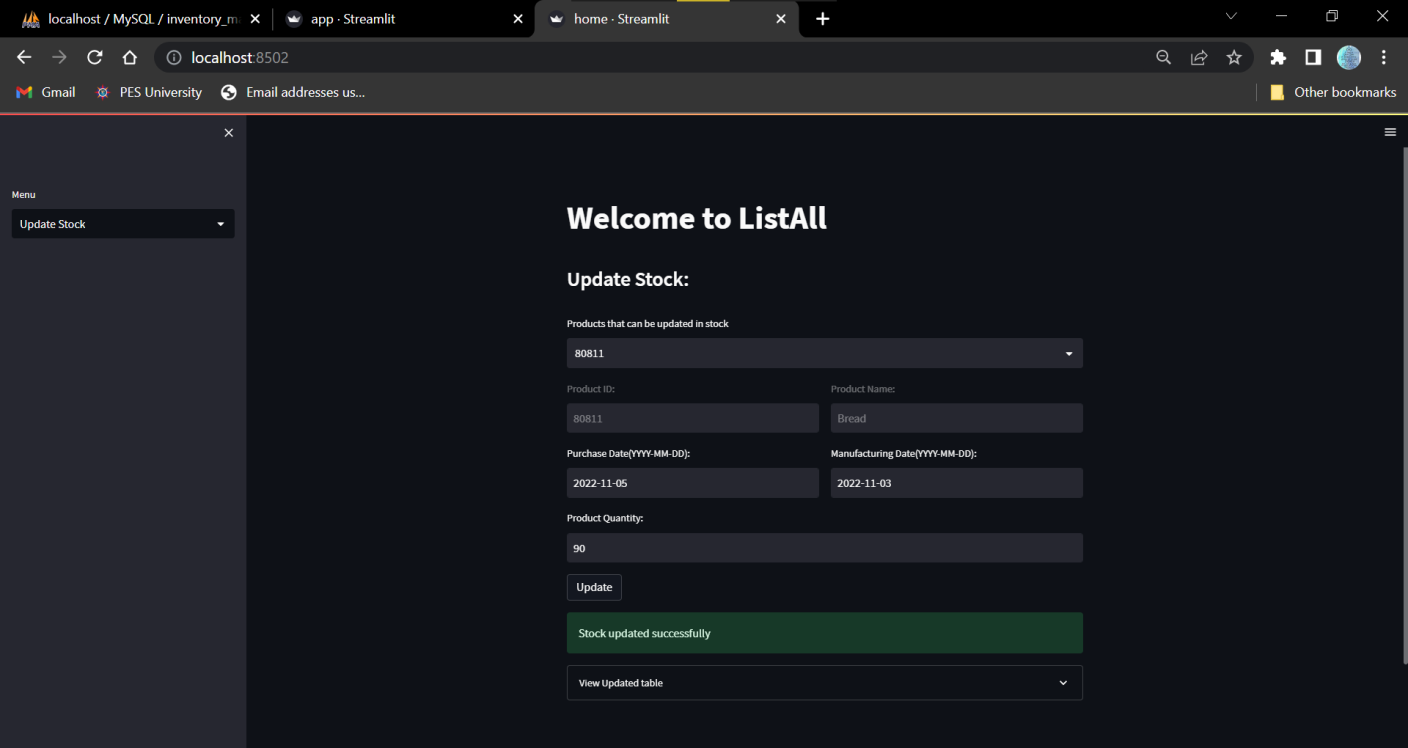
Screenshots:

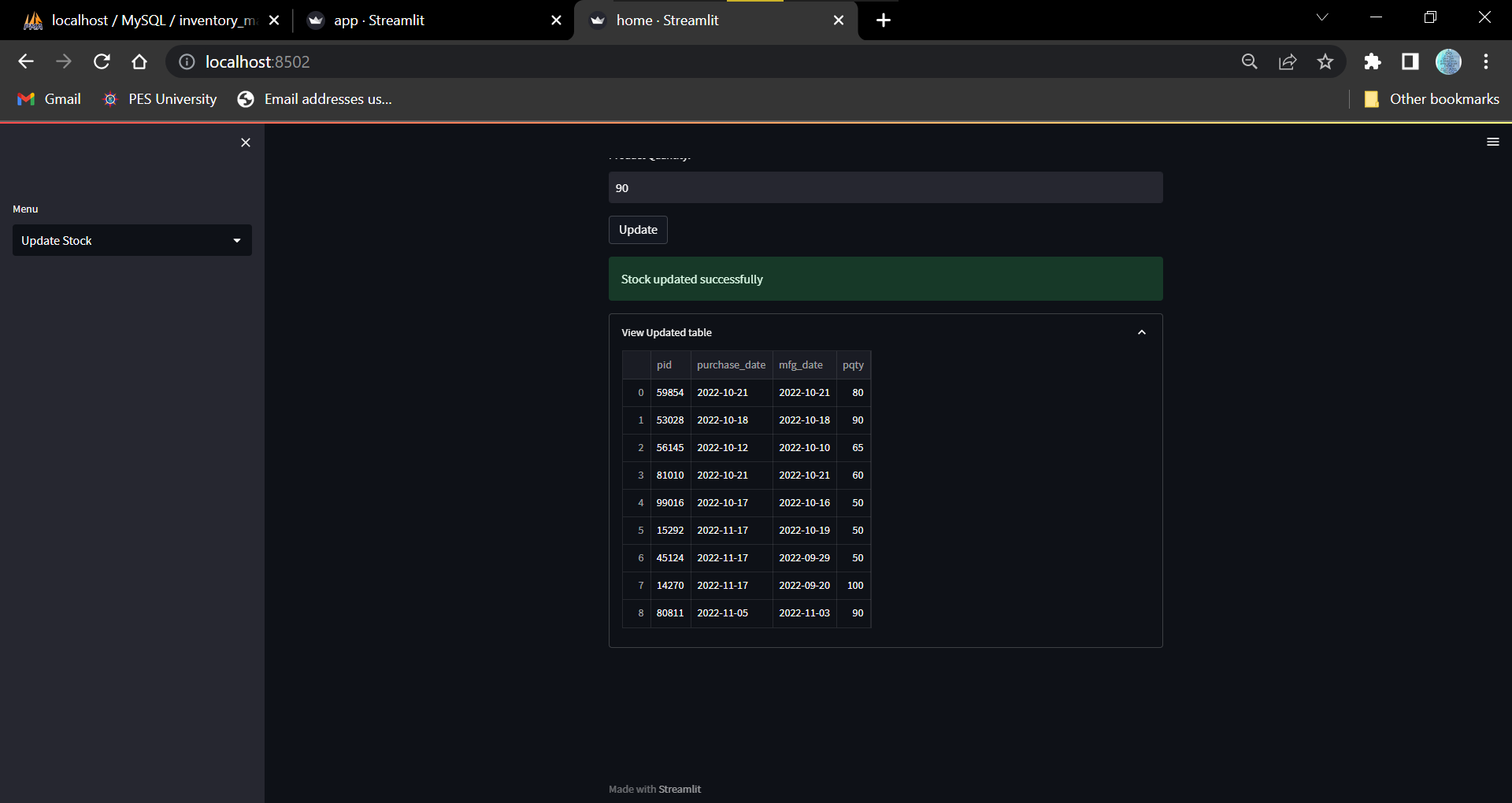


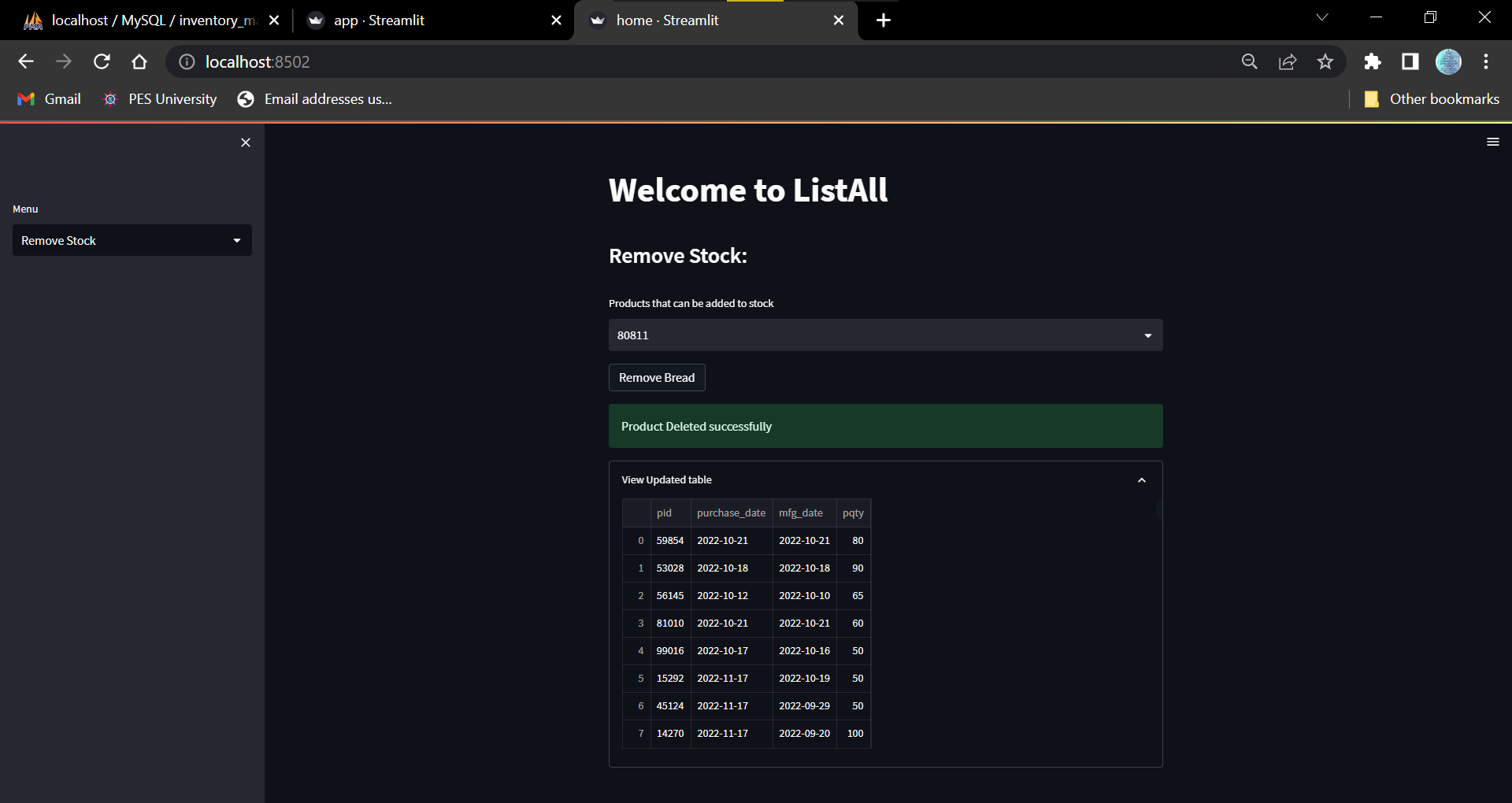


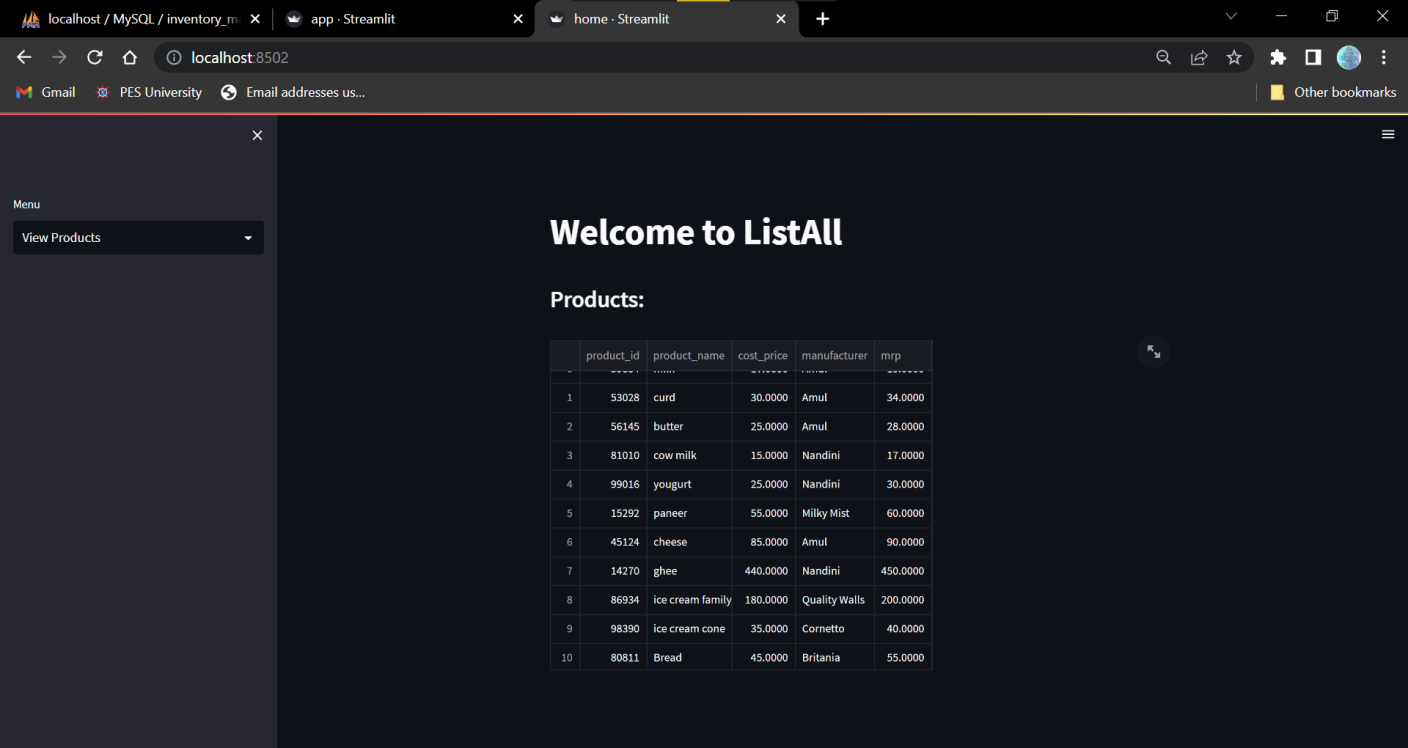


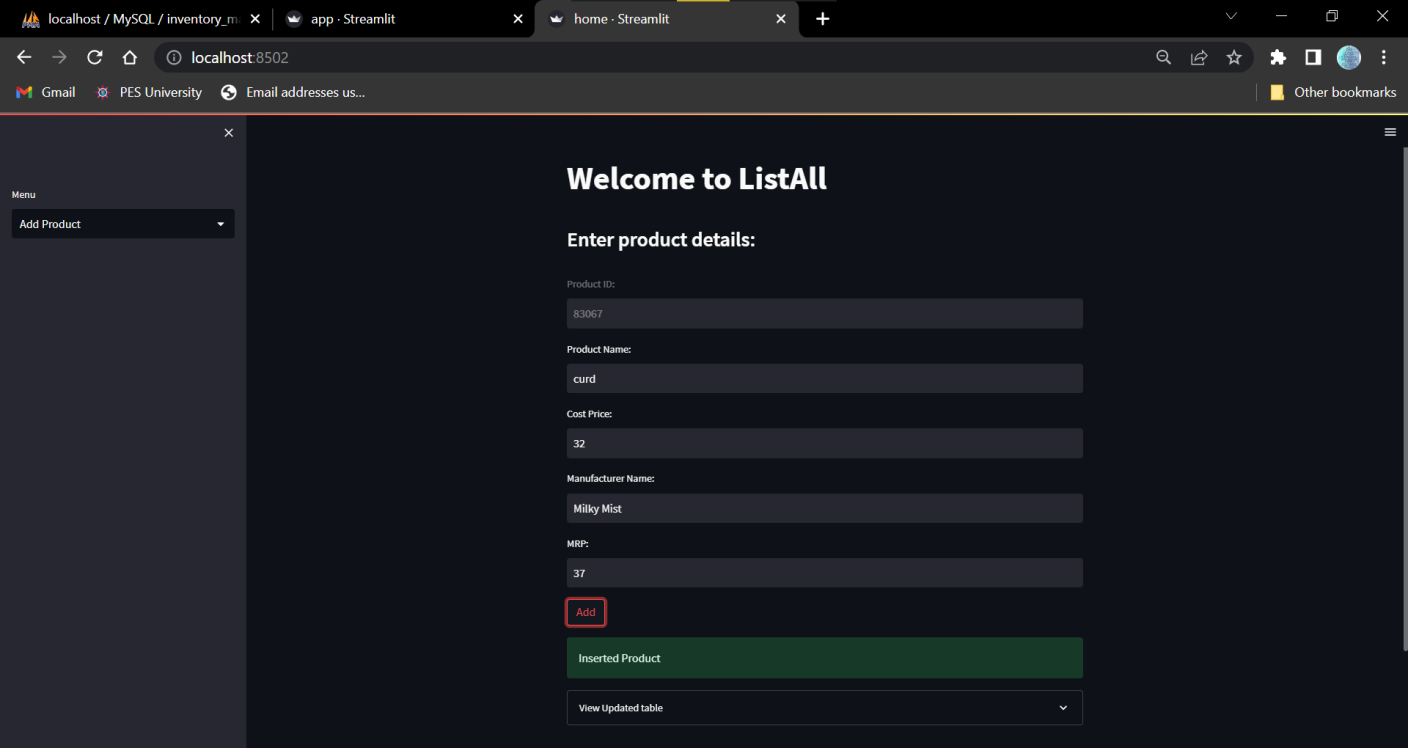


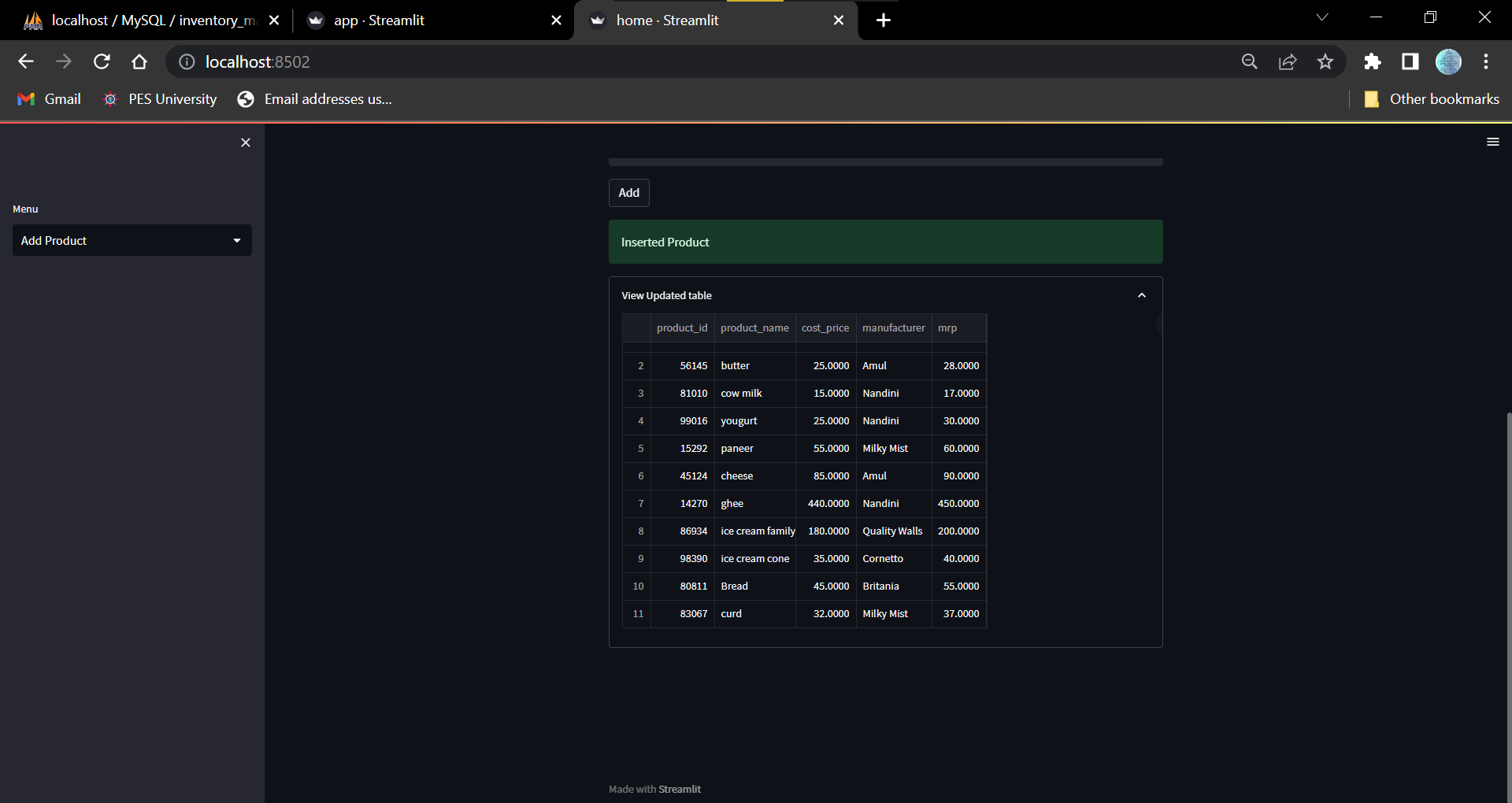


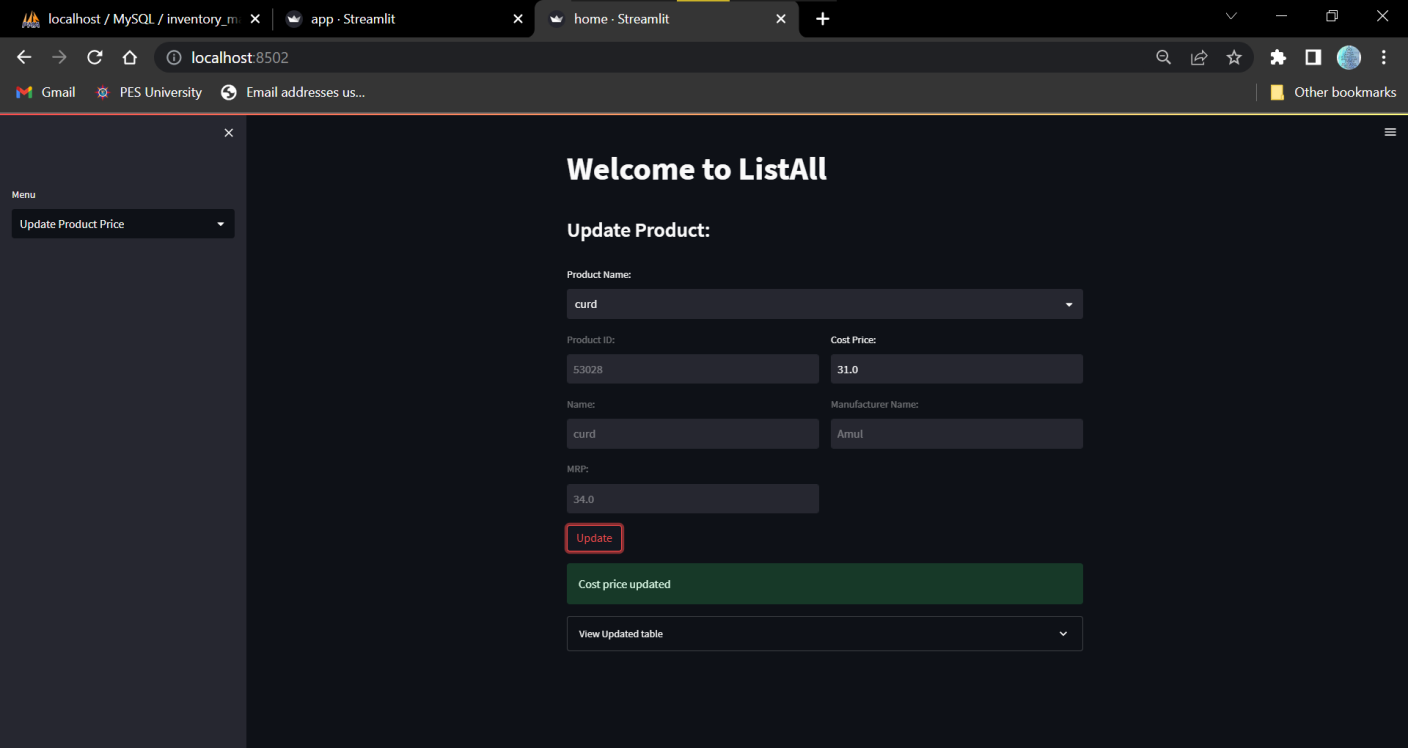


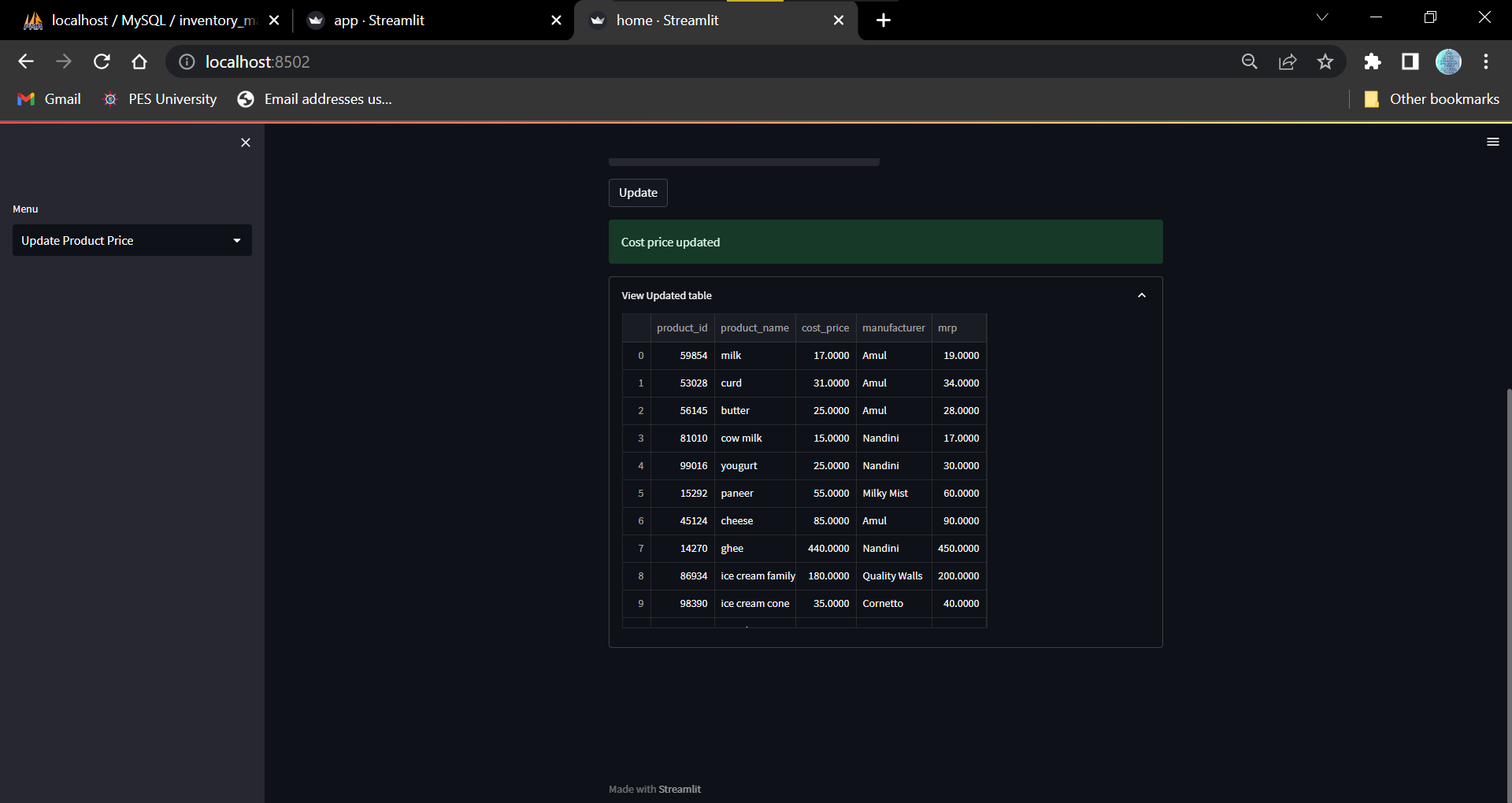


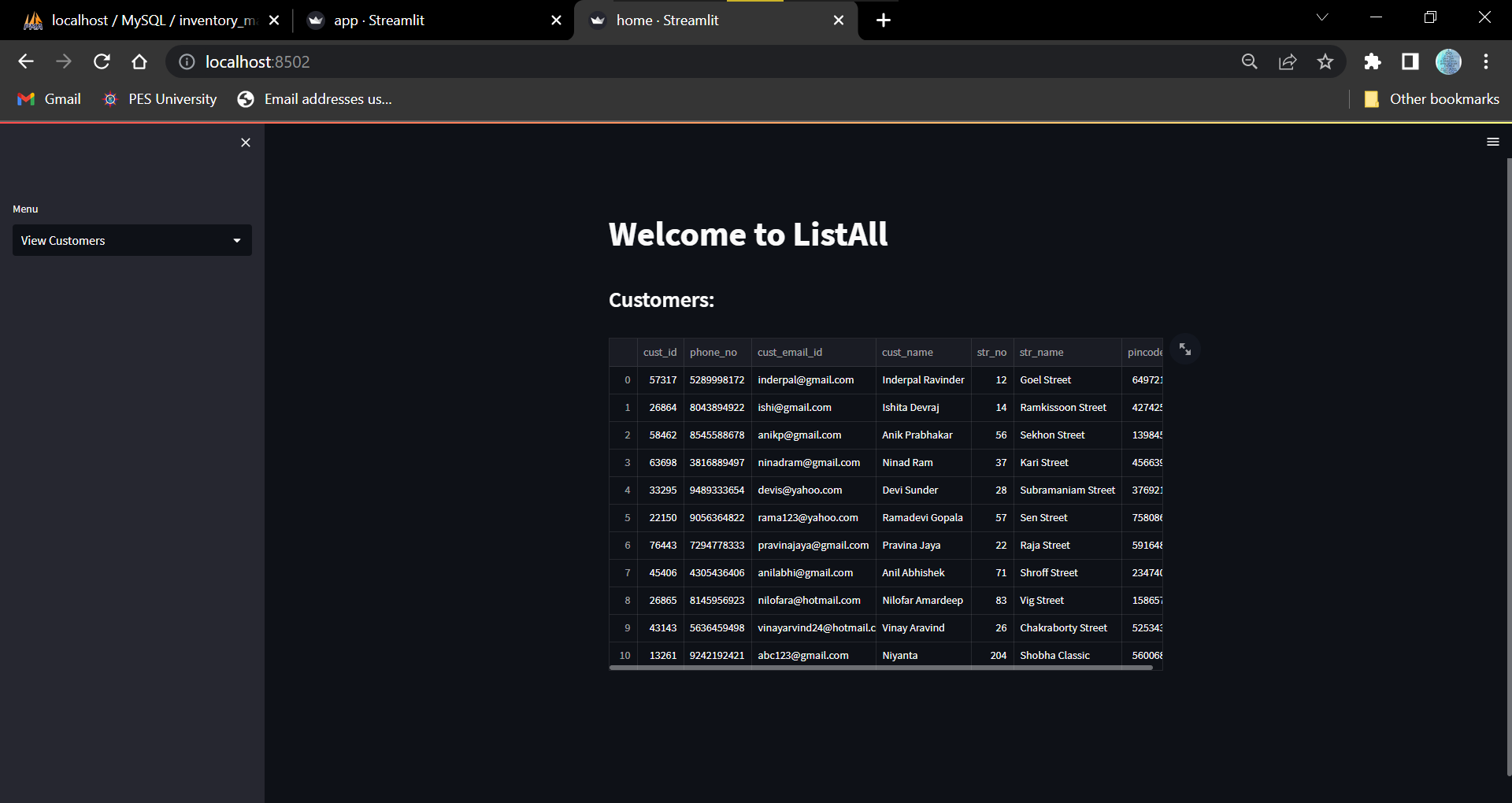


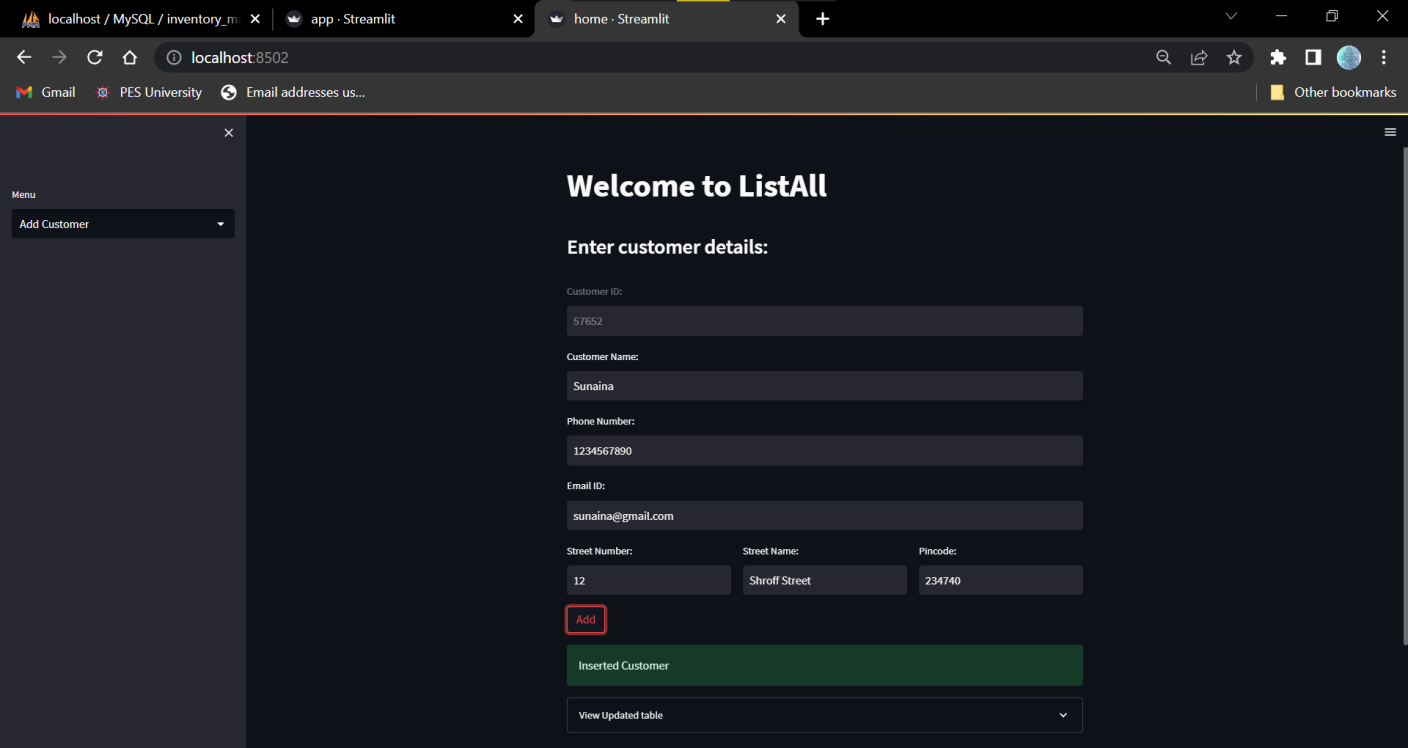


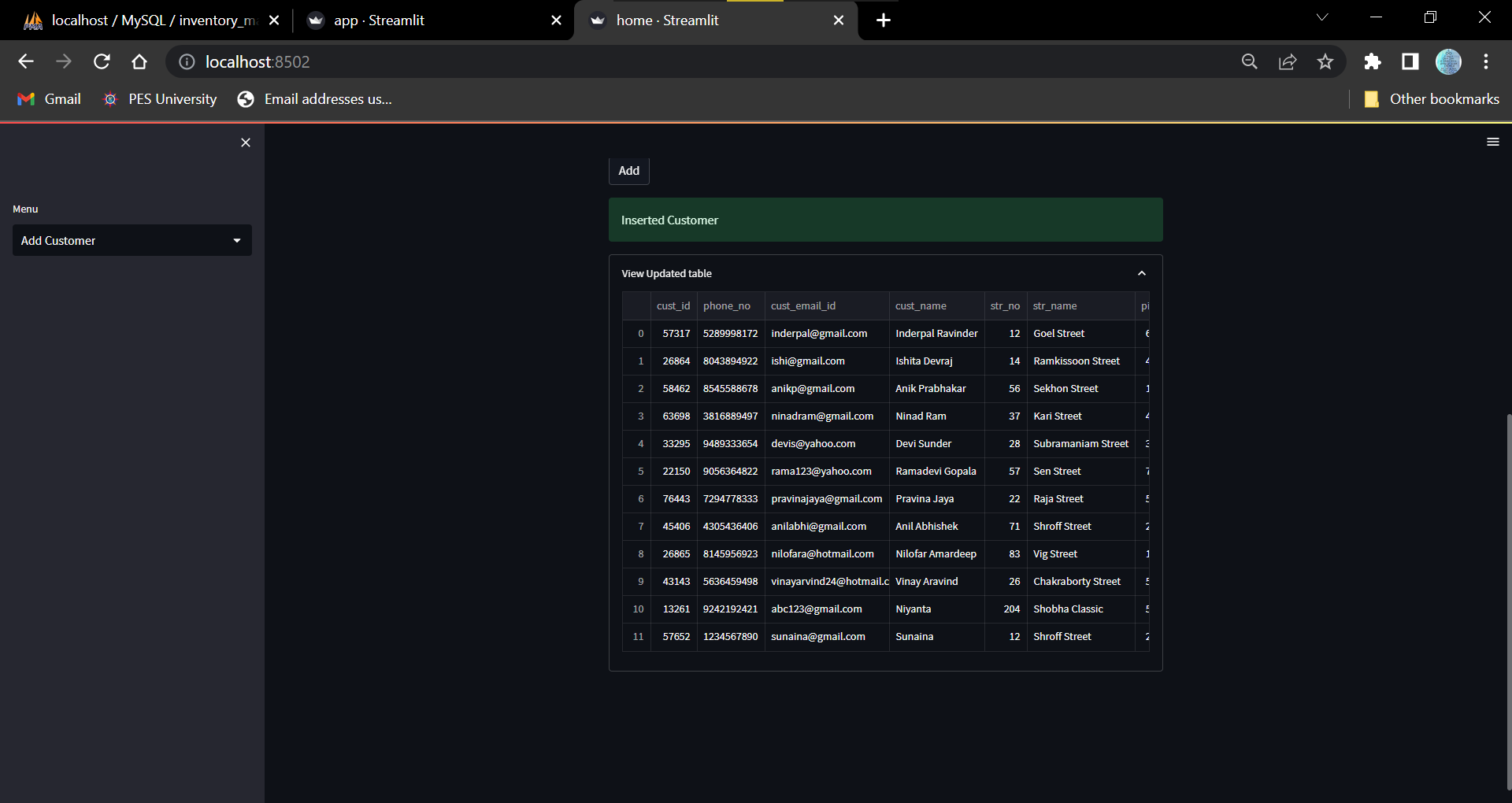


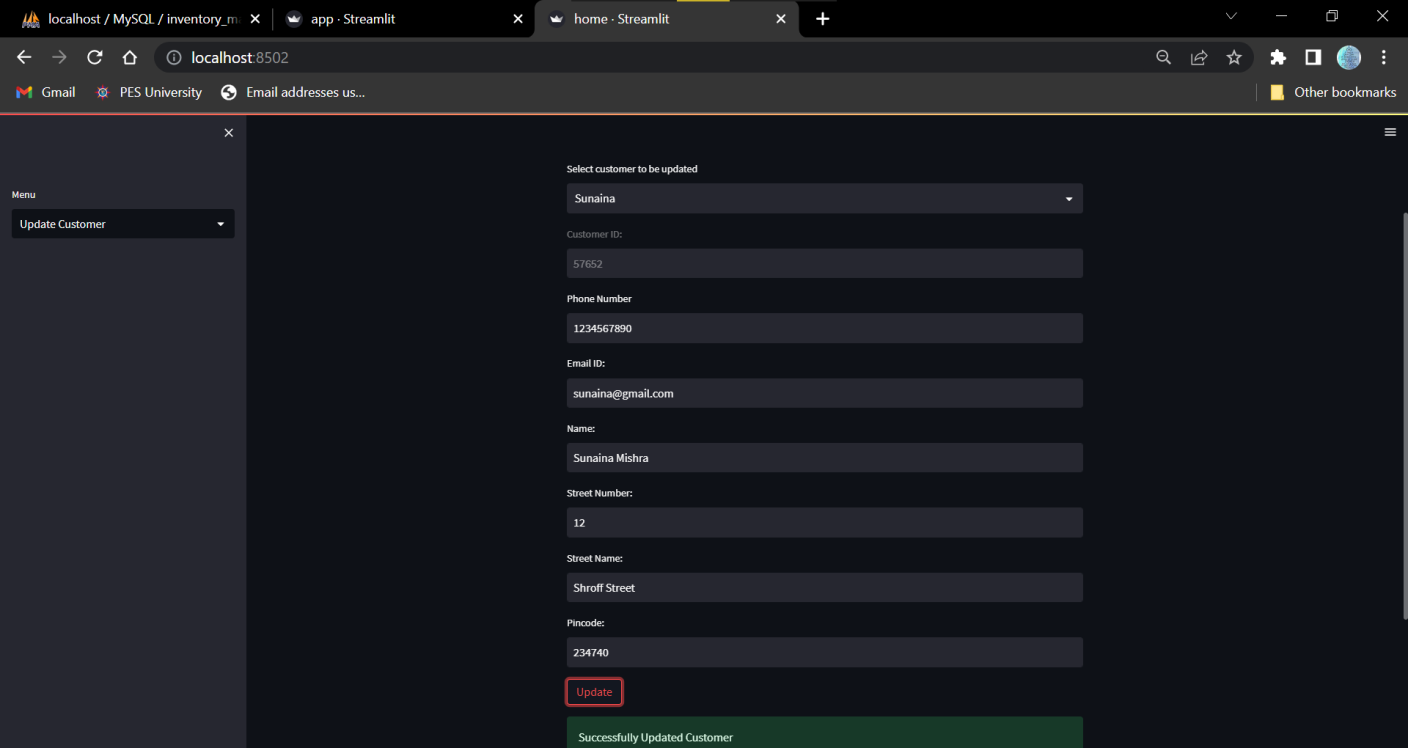


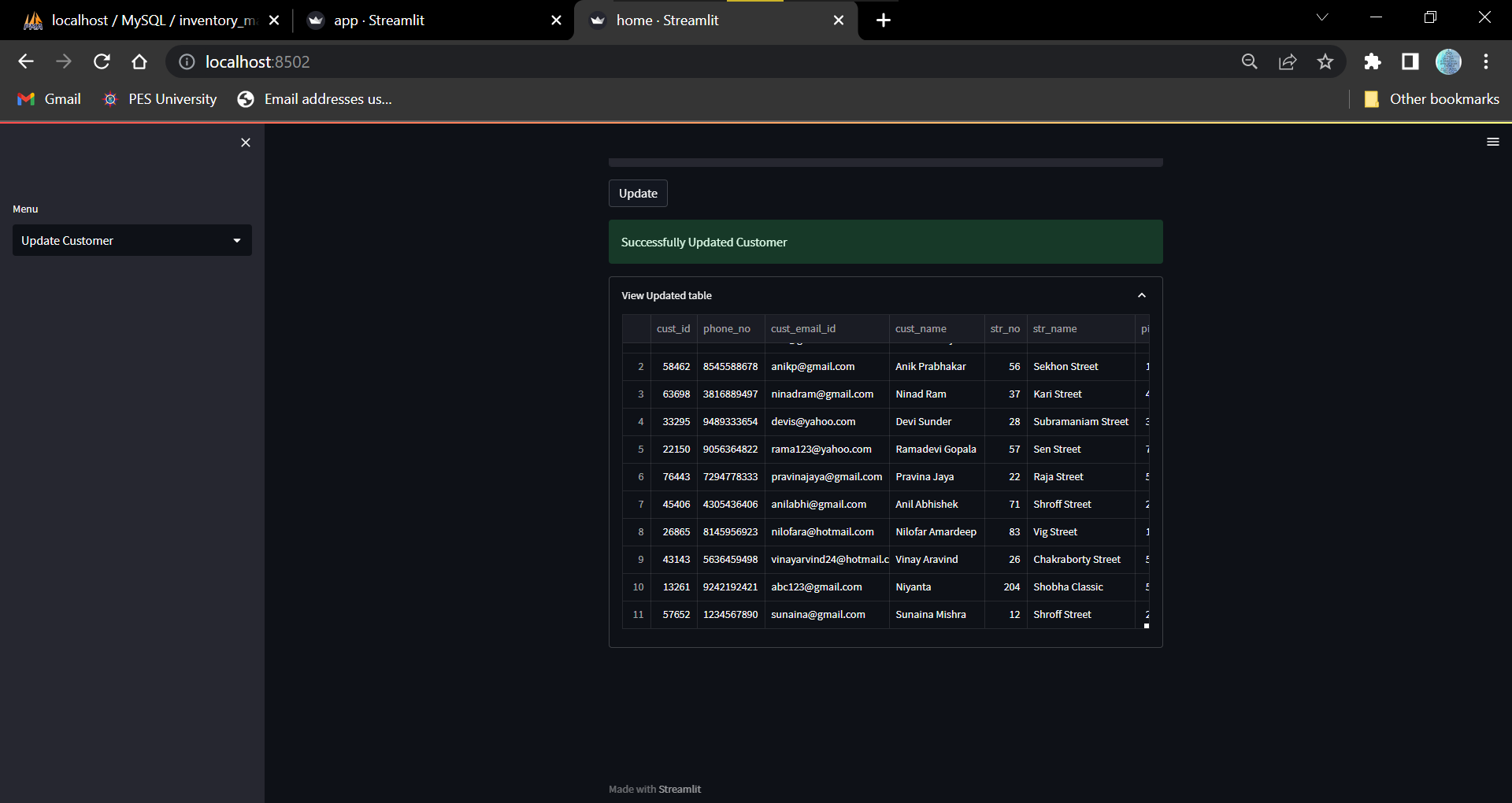


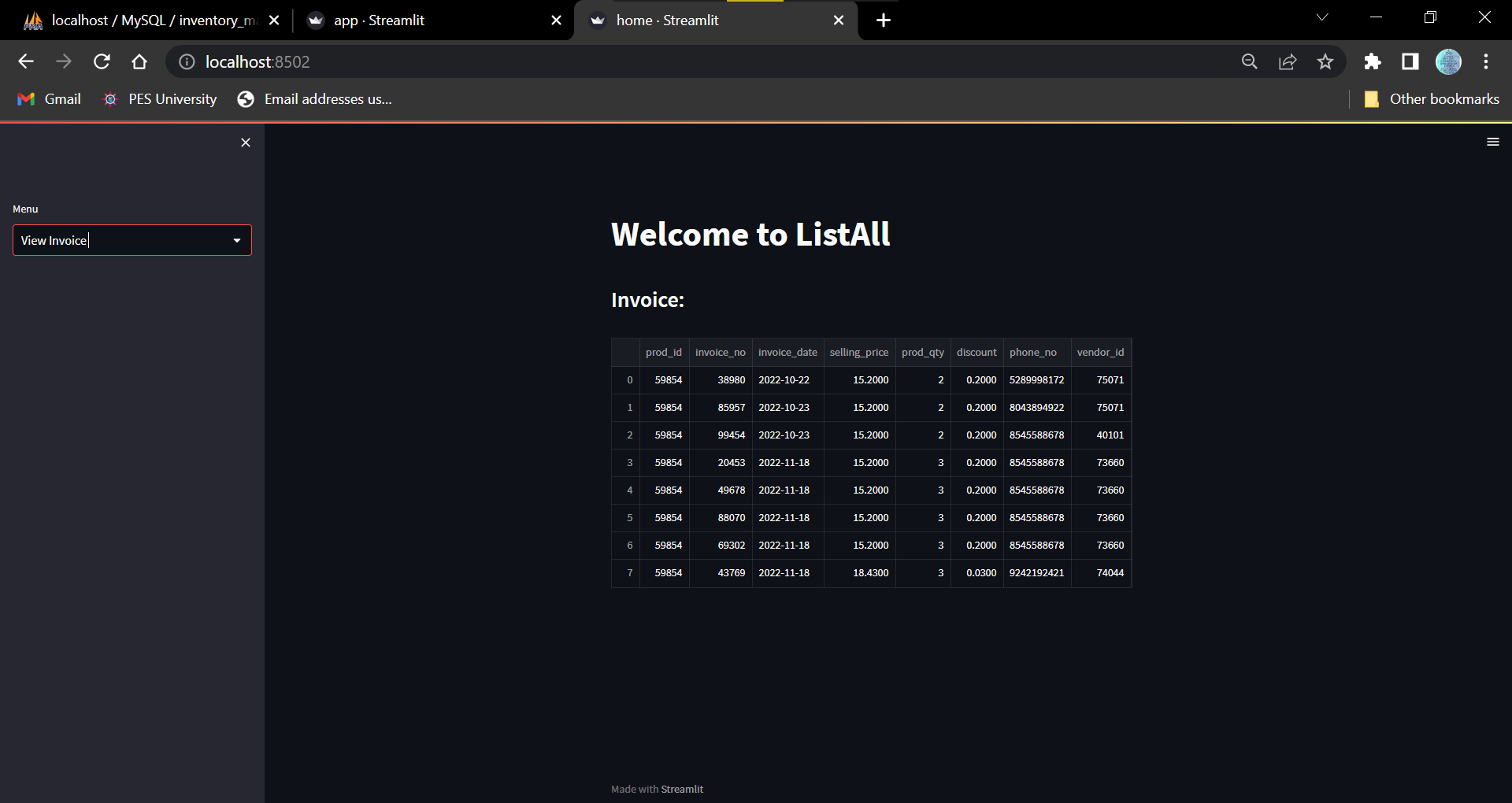


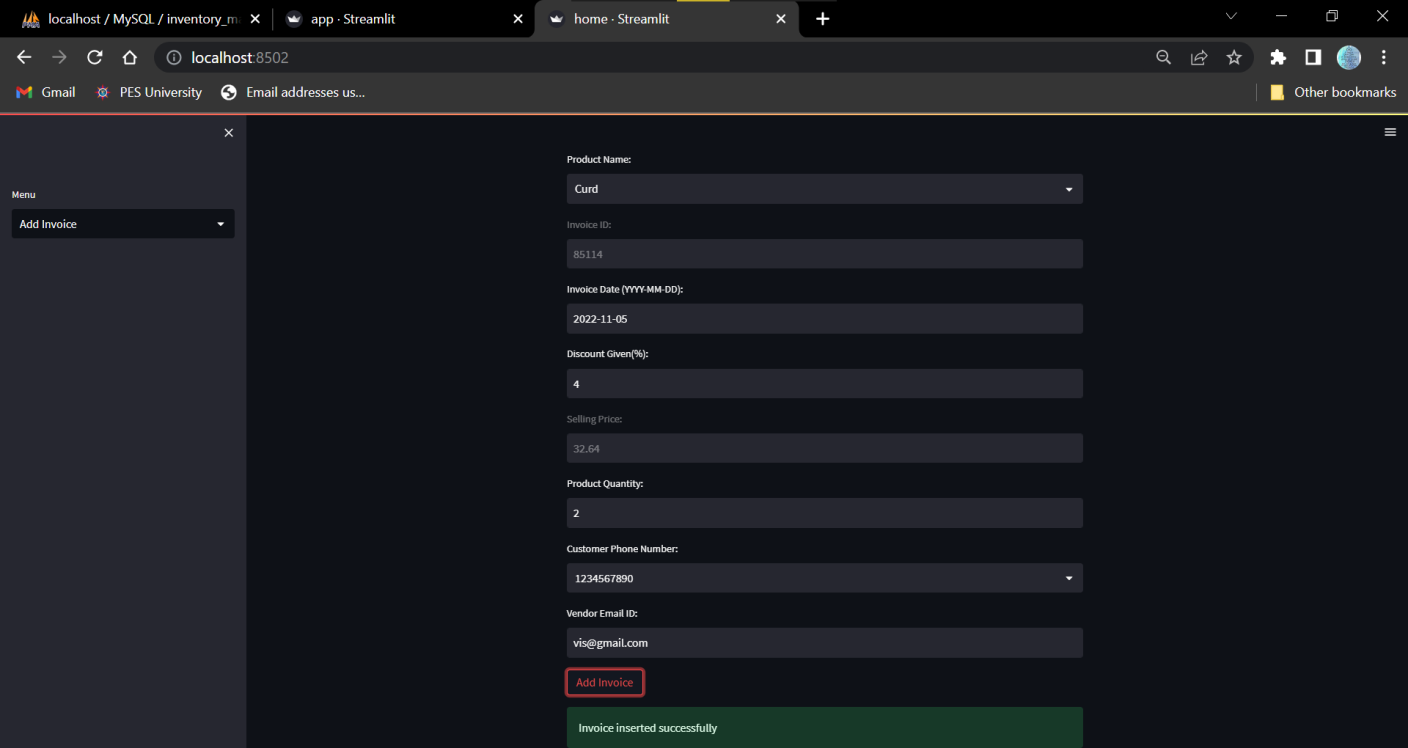


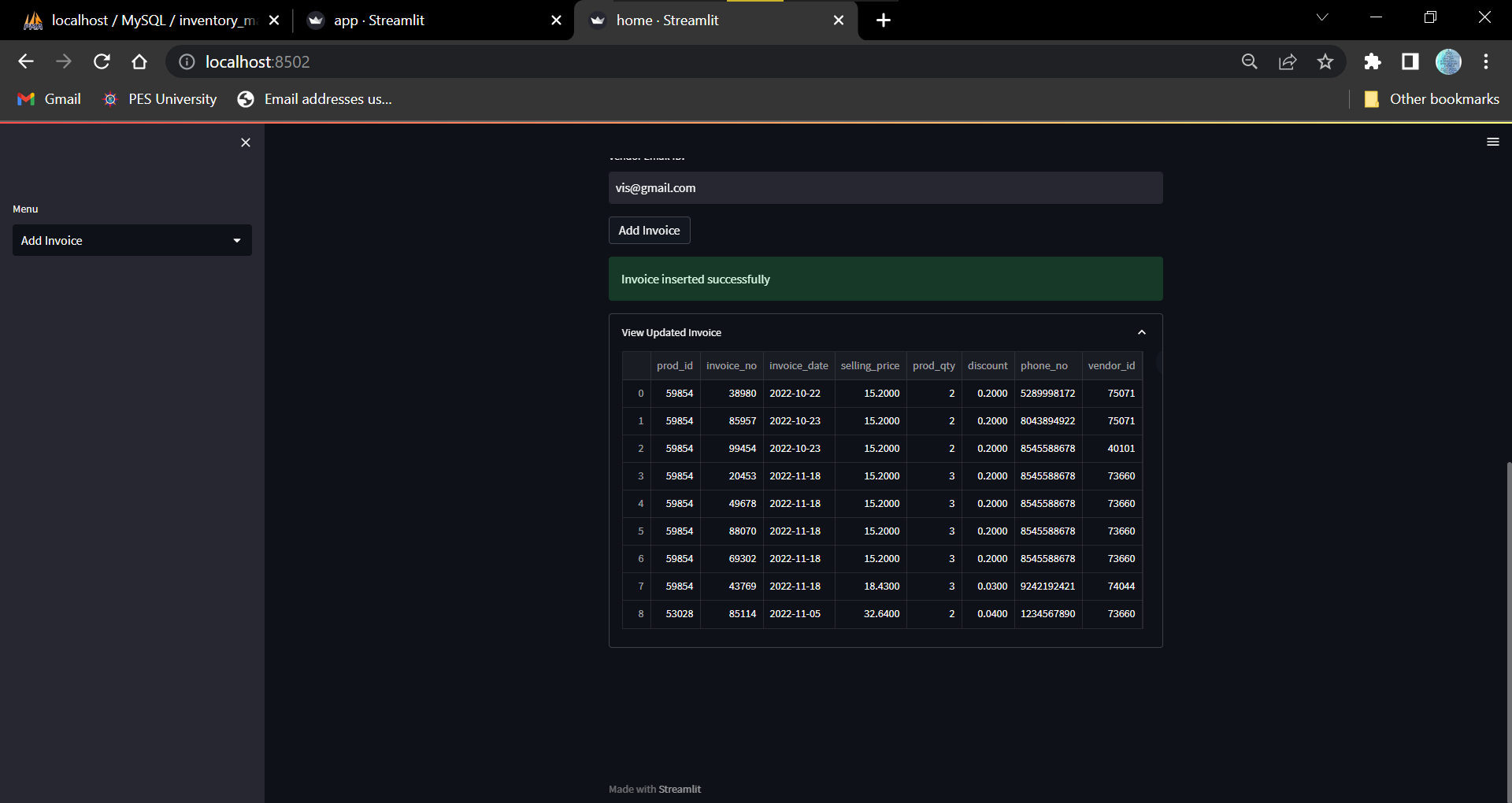


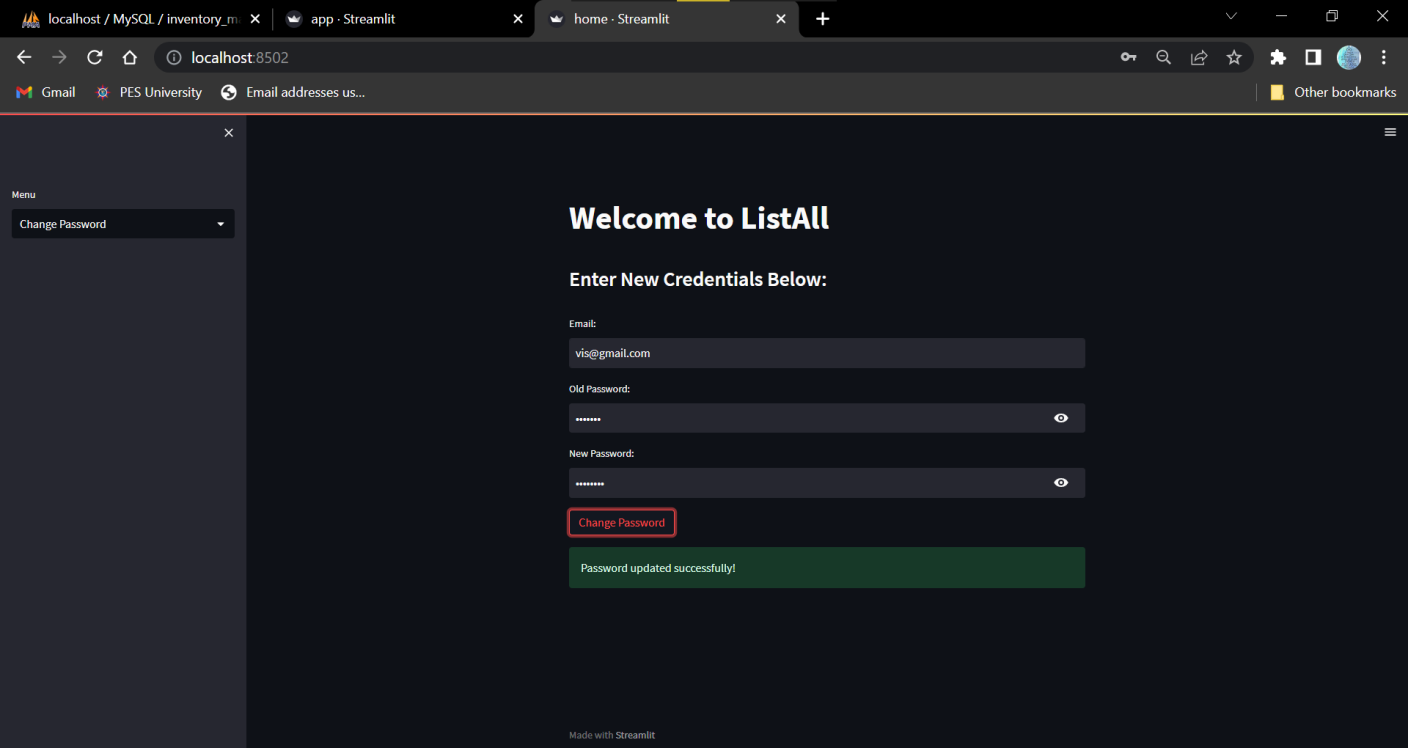












**12. Conclusion**

The project currently performs most basic CRUD operations on the database and comprises of triggers, functions, procedures, cursors etc. to help refine these CRUD operations. There is much more scope to this project and I will continue to build it going forward.

**13. References**

[1] PESU Academy Slides and Notes

[2] MySQL official documentation