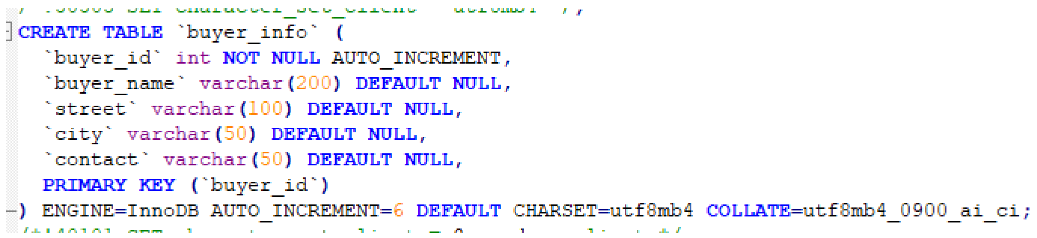
Implementation Document

Dominique Dalanni, Marshal Ben Manuel, Yash Prajapati, Krishna Chennamsetty

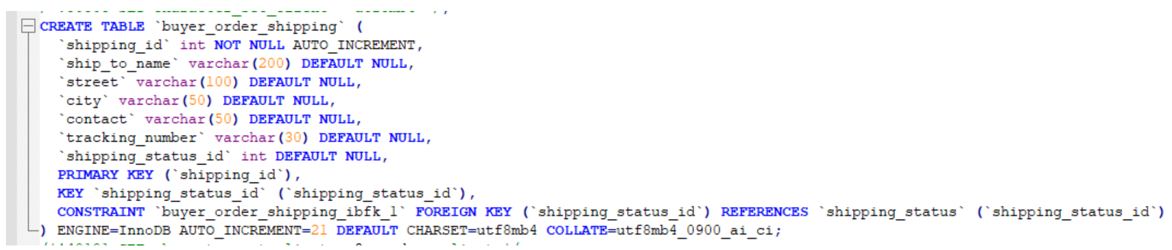
DDL

DDL (20pts) – represents the entire DB schema, including table structures, primary and foreign key constraints, data types, data constraints, table indexes and relationships. Your DB should consist of at least 12 entities of suitable complexity (3-10 attributes).

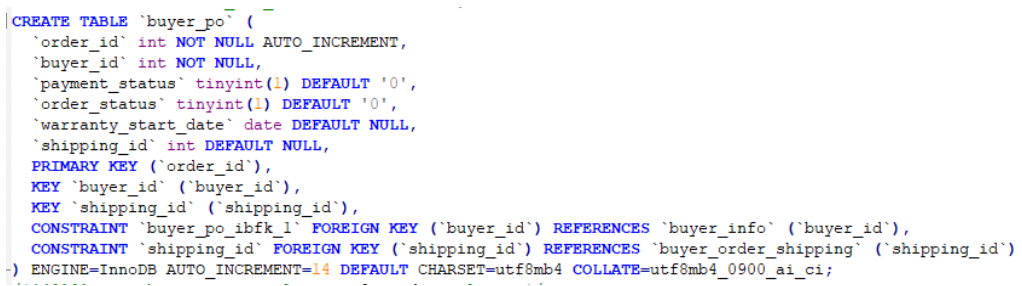
Within our submission files, we included our sql scripts utilized to create and update the database. We were also able to export our database structure and data into a sql file, which we have included in our submission entitled “Exported DDL and DML with Data.sql”. Screenshots of our fourteen tables from the sql file of our exported database and structure can be found below:



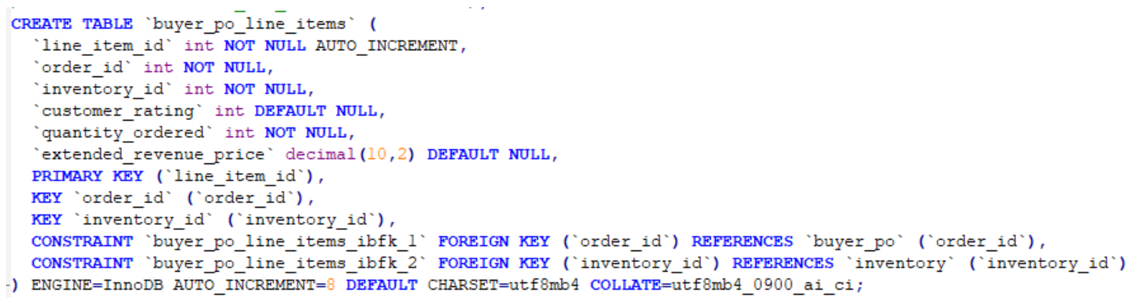
```
CREATE TABLE `buyer_info` (
  `buyer_id` int NOT NULL AUTO_INCREMENT,
  `buyer_name` varchar(200) DEFAULT NULL,
  `street` varchar(100) DEFAULT NULL,
  `city` varchar(50) DEFAULT NULL,
  `contact` varchar(50) DEFAULT NULL,
  PRIMARY KEY (`buyer_id`)
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```



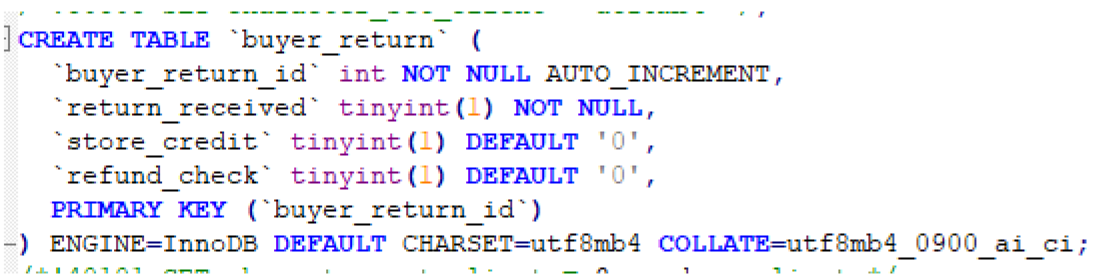
```
CREATE TABLE `buyer_order_shipping` (
  `shipping_id` int NOT NULL AUTO_INCREMENT,
  `ship_to_name` varchar(200) DEFAULT NULL,
  `street` varchar(100) DEFAULT NULL,
  `city` varchar(50) DEFAULT NULL,
  `contact` varchar(50) DEFAULT NULL,
  `tracking_number` varchar(30) DEFAULT NULL,
  `shipping_status_id` int DEFAULT NULL,
  PRIMARY KEY (`shipping_id`),
  KEY `shipping_status_id` (`shipping_status_id`),
  CONSTRAINT `buyer_order_shipping_ibfk_1` FOREIGN KEY (`shipping_status_id`) REFERENCES `shipping_status` (`shipping_status_id`)
) ENGINE=InnoDB AUTO_INCREMENT=21 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```



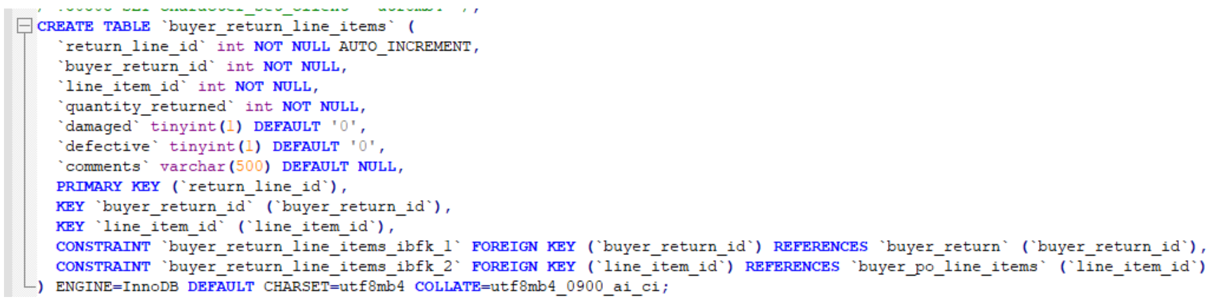
```
CREATE TABLE `buyer_po` (
  `order_id` int NOT NULL AUTO_INCREMENT,
  `buyer_id` int NOT NULL,
  `payment_status` tinyint(1) DEFAULT '0',
  `order_status` tinyint(1) DEFAULT '0',
  `warranty_start_date` date DEFAULT NULL,
  `shipping_id` int DEFAULT NULL,
  PRIMARY KEY (`order_id`),
  KEY `buyer_id` (`buyer_id`),
  KEY `shipping_id` (`shipping_id`),
  CONSTRAINT `buyer_po_ibfk_1` FOREIGN KEY (`buyer_id`) REFERENCES `buyer_info` (`buyer_id`),
  CONSTRAINT `shipping_id` FOREIGN KEY (`shipping_id`) REFERENCES `buyer_order_shipping` (`shipping_id`)
) ENGINE=InnoDB AUTO_INCREMENT=14 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```



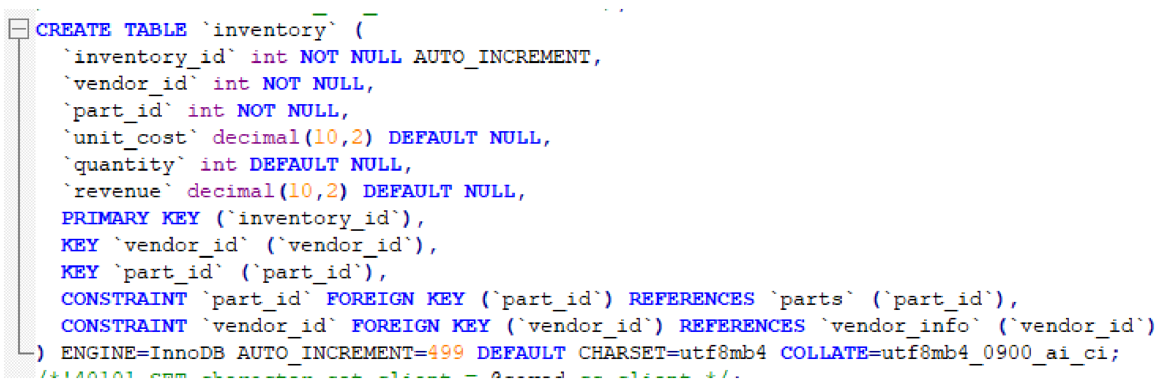
```
CREATE TABLE `buyer_po_line_items` (
  `line_item_id` int NOT NULL AUTO_INCREMENT,
  `order_id` int NOT NULL,
  `inventory_id` int NOT NULL,
  `customer_rating` int DEFAULT NULL,
  `quantity_ordered` int NOT NULL,
  `extended_revenue_price` decimal(10,2) DEFAULT NULL,
  PRIMARY KEY (`line_item_id`),
  KEY `order_id` (`order_id`),
  KEY `inventory_id` (`inventory_id`),
  CONSTRAINT `buyer_po_line_items_ibfk_1` FOREIGN KEY (`order_id`) REFERENCES `buyer_po` (`order_id`),
  CONSTRAINT `buyer_po_line_items_ibfk_2` FOREIGN KEY (`inventory_id`) REFERENCES `inventory` (`inventory_id`)
) ENGINE=InnoDB AUTO_INCREMENT=8 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```



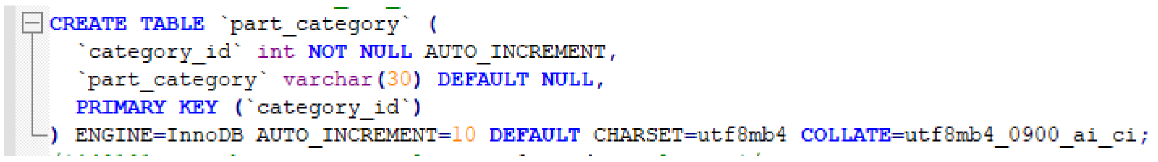
```
CREATE TABLE `buyer_return` (
  `buyer_return_id` int NOT NULL AUTO_INCREMENT,
  `return_received` tinyint(1) NOT NULL,
  `store_credit` tinyint(1) DEFAULT '0',
  `refund_check` tinyint(1) DEFAULT '0',
  PRIMARY KEY (`buyer_return_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```



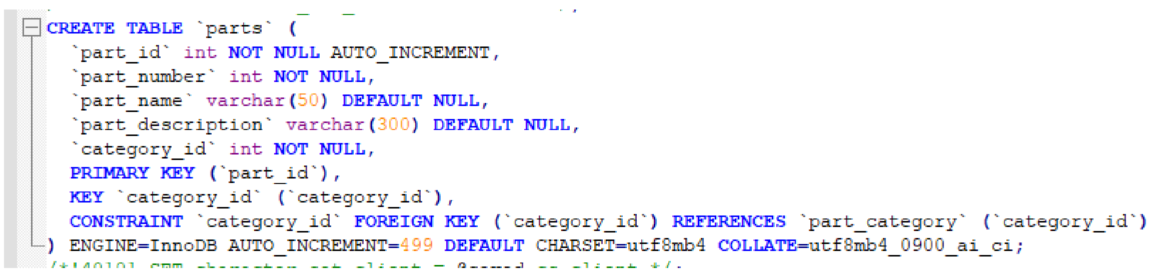
```
CREATE TABLE `buyer_return_line_items` (
  `return_line_id` int NOT NULL AUTO_INCREMENT,
  `buyer_return_id` int NOT NULL,
  `line_item_id` int NOT NULL,
  `quantity_returned` int NOT NULL,
  `damaged` tinyint(1) DEFAULT '0',
  `defective` tinyint(1) DEFAULT '0',
  `comments` varchar(500) DEFAULT NULL,
  PRIMARY KEY (`return_line_id`),
  KEY `buyer_return_id` (`buyer_return_id`),
  KEY `line_item_id` (`line_item_id`),
  CONSTRAINT `buyer_return_line_items_ibfk_1` FOREIGN KEY (`buyer_return_id`) REFERENCES `buyer_return` (`buyer_return_id`),
  CONSTRAINT `buyer_return_line_items_ibfk_2` FOREIGN KEY (`line_item_id`) REFERENCES `buyer_po_line_items` (`line_item_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```



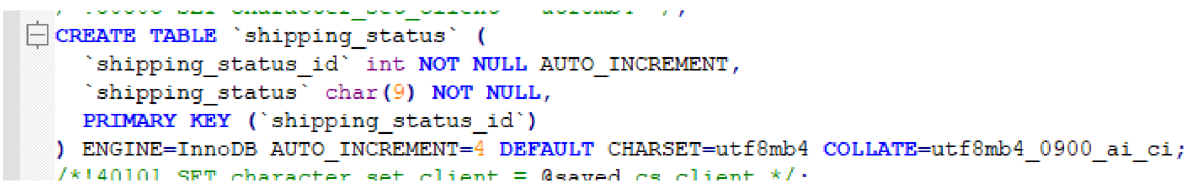
```
CREATE TABLE `inventory` (
  `inventory_id` int NOT NULL AUTO_INCREMENT,
  `vendor_id` int NOT NULL,
  `part_id` int NOT NULL,
  `unit_cost` decimal(10,2) DEFAULT NULL,
  `quantity` int DEFAULT NULL,
  `revenue` decimal(10,2) DEFAULT NULL,
  PRIMARY KEY (`inventory_id`),
  KEY `vendor_id` (`vendor_id`),
  KEY `part_id` (`part_id`),
  CONSTRAINT `part_id` FOREIGN KEY (`part_id`) REFERENCES `parts` (`part_id`),
  CONSTRAINT `vendor_id` FOREIGN KEY (`vendor_id`) REFERENCES `vendor_info` (`vendor_id`)
) ENGINE=InnoDB AUTO_INCREMENT=499 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```



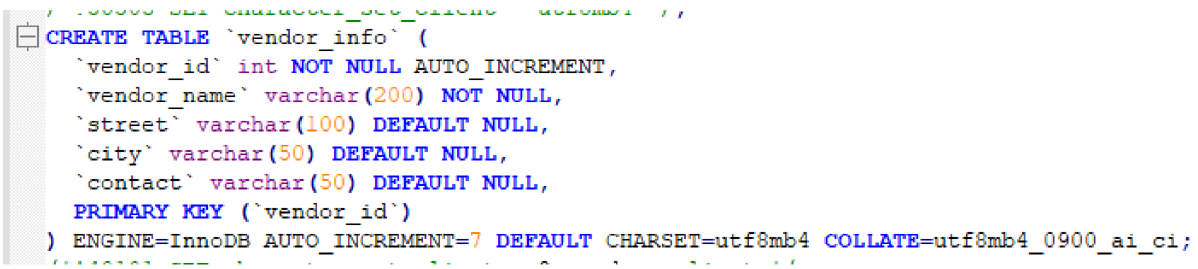
```
CREATE TABLE `part_category` (
  `category_id` int NOT NULL AUTO_INCREMENT,
  `part_category` varchar(30) DEFAULT NULL,
  PRIMARY KEY (`category_id`)
) ENGINE=InnoDB AUTO_INCREMENT=10 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```



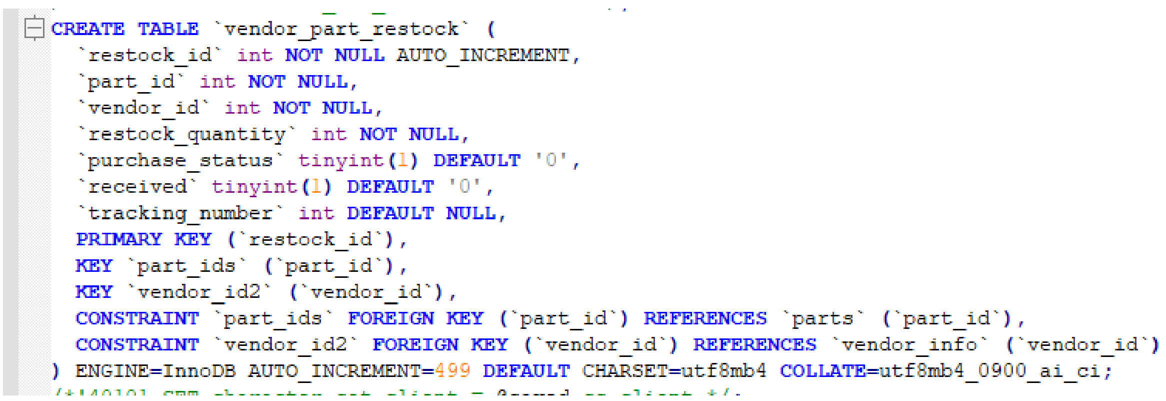
```
CREATE TABLE `parts` (
  `part_id` int NOT NULL AUTO_INCREMENT,
  `part_number` int NOT NULL,
  `part_name` varchar(50) DEFAULT NULL,
  `part_description` varchar(300) DEFAULT NULL,
  `category_id` int NOT NULL,
  PRIMARY KEY (`part_id`),
  KEY `category_id` (`category_id`),
  CONSTRAINT `category_id` FOREIGN KEY (`category_id`) REFERENCES `part_category` (`category_id`)
) ENGINE=InnoDB AUTO_INCREMENT=499 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```



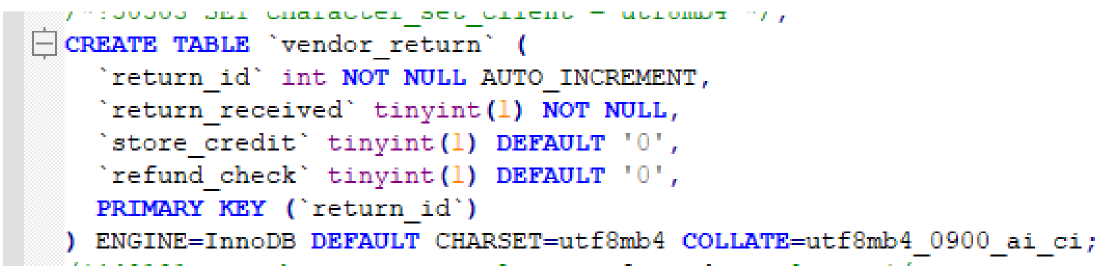
```sql
CREATE TABLE `shipping_status` (
  `shipping_status_id` int NOT NULL AUTO_INCREMENT,
  `shipping_status` char(9) NOT NULL,
  PRIMARY KEY (`shipping_status_id`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;
```



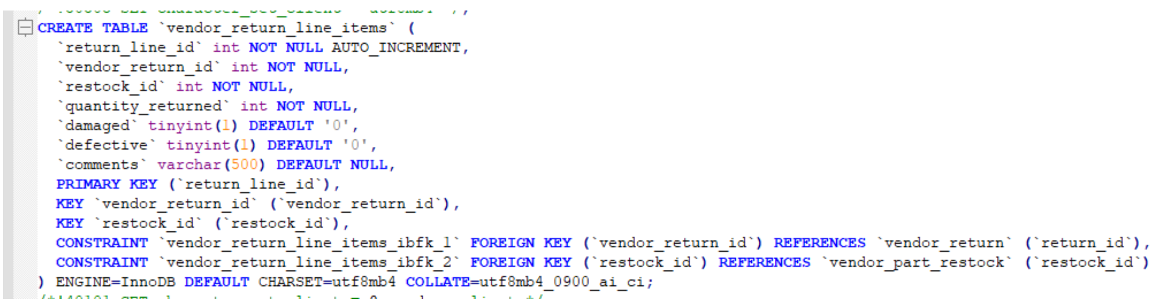
```sql
CREATE TABLE `vendor_info` (
  `vendor_id` int NOT NULL AUTO_INCREMENT,
  `vendor_name` varchar(200) NOT NULL,
  `street` varchar(100) DEFAULT NULL,
  `city` varchar(50) DEFAULT NULL,
  `contact` varchar(50) DEFAULT NULL,
  PRIMARY KEY (`vendor_id`)
) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```



```sql
CREATE TABLE `vendor_part_restock` (
  `restock_id` int NOT NULL AUTO_INCREMENT,
  `part_id` int NOT NULL,
  `vendor_id` int NOT NULL,
  `restock_quantity` int NOT NULL,
  `purchase_status` tinyint(1) DEFAULT '0',
  `received` tinyint(1) DEFAULT '0',
  `tracking_number` int DEFAULT NULL,
  PRIMARY KEY (`restock_id`),
  KEY `part_ids` (`part_id`),
  KEY `vendor_id2` (`vendor_id`),
  CONSTRAINT `part_ids` FOREIGN KEY (`part_id`) REFERENCES `parts` (`part_id`),
  CONSTRAINT `vendor_id2` FOREIGN KEY (`vendor_id`) REFERENCES `vendor_info` (`vendor_id`)
) ENGINE=InnoDB AUTO_INCREMENT=499 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```



```sql
CREATE TABLE `vendor_return` (
  `return_id` int NOT NULL AUTO_INCREMENT,
  `return_received` tinyint(1) NOT NULL,
  `store_credit` tinyint(1) DEFAULT '0',
  `refund_check` tinyint(1) DEFAULT '0',
  PRIMARY KEY (`return_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```



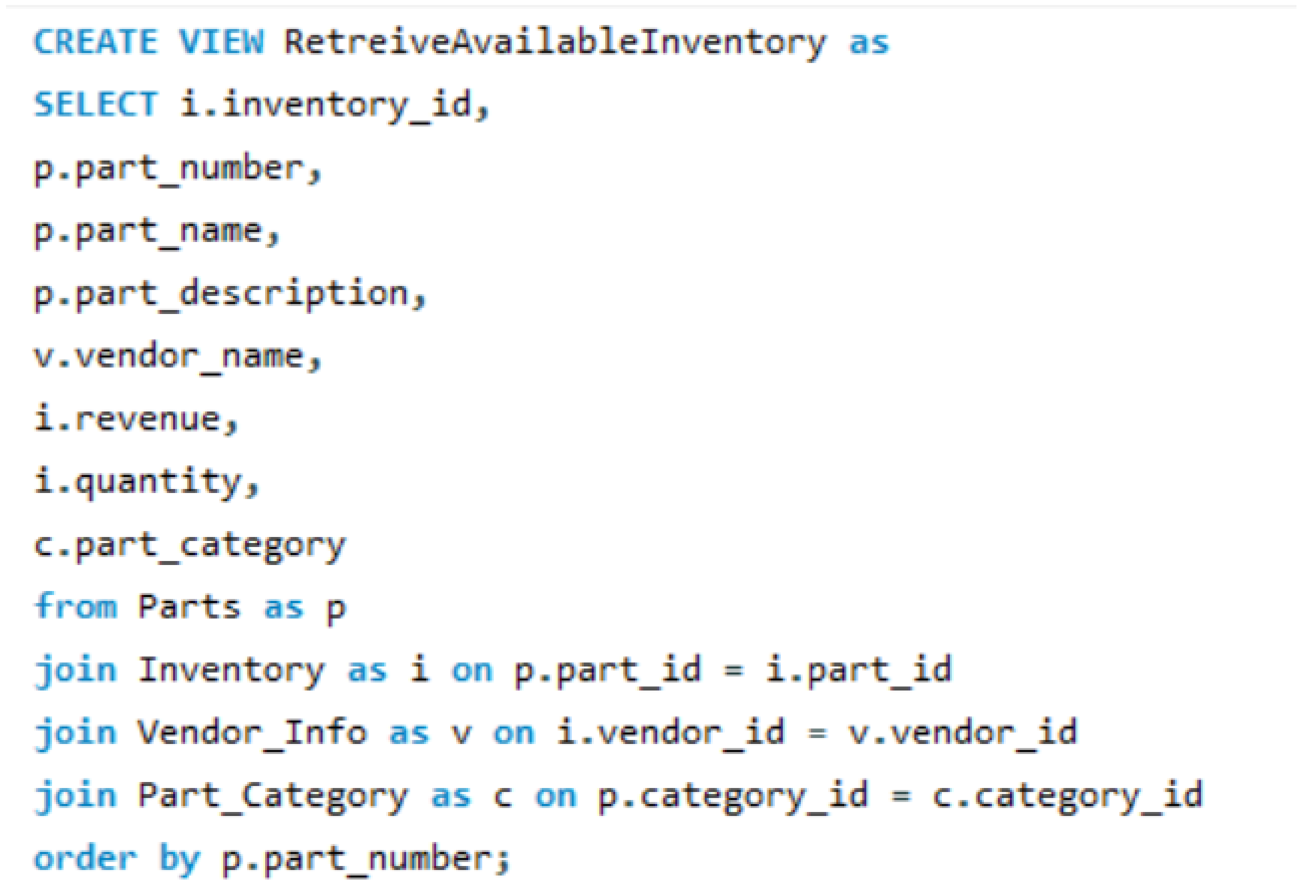
```sql
CREATE TABLE `vendor_return_line_items` (
  `return_line_id` int NOT NULL AUTO_INCREMENT,
  `vendor_return_id` int NOT NULL,
  `restock_id` int NOT NULL,
  `quantity_returned` int NOT NULL,
  `damaged` tinyint(1) DEFAULT '0',
  `defective` tinyint(1) DEFAULT '0',
  `comments` varchar(500) DEFAULT NULL,
  PRIMARY KEY (`return_line_id`),
  KEY `vendor_return_id` (`vendor_return_id`),
  KEY `restock_id` (`restock_id`),
  CONSTRAINT `vendor_return_line_items_ibfk_1` FOREIGN KEY (`vendor_return_id`) REFERENCES `vendor_return` (`return_id`),
  CONSTRAINT `vendor_return_line_items_ibfk_2` FOREIGN KEY (`restock_id`) REFERENCES `vendor_part_restock` (`restock_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

DML

DML (20pts) – Represents views (5), queries as stored procedures with varying levels of difficulty
(12) and triggers (3) to facilitate information retrieval across your system. Think about the DML in terms of business needs and data administration. In essence, queries are algorithms for information retrieval and should therefore be of a certain complexity to justify their existence. Therefore, a large percentage of your grade will consider such complexity and will be based on your understanding of table joins and query manipulation. Consequently, you are expected to create at least 20 database objects (views, stored procedures and triggers) of varying levels of complexity and provide a rationale for each query.

1. View To Retrieve A List of Available Inventory:

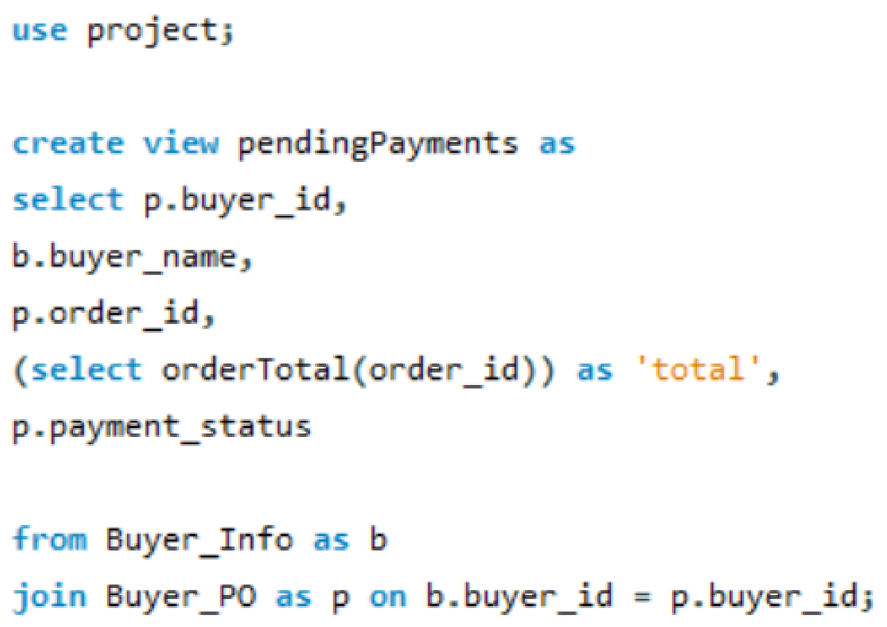
This view is used within the GUI's "View Inventory" feature to display a list of available inventory within our inventory management system:



```
CREATE VIEW RetreiveAvailableInventory as
SELECT i.inventory_id,
p.part_number,
p.part_name,
p.part_description,
v.vendor_name,
i.revenue,
i.quantity,
c.part_category
from Parts as p
join Inventory as i on p.part_id = i.part_id
join Vendor_Info as v on i.vendor_id = v.vendor_id
join Part_Category as c on p.category_id = c.category_id
order by p.part_number;
```

2. View To See Buyer Purchase Order Information Including Total:

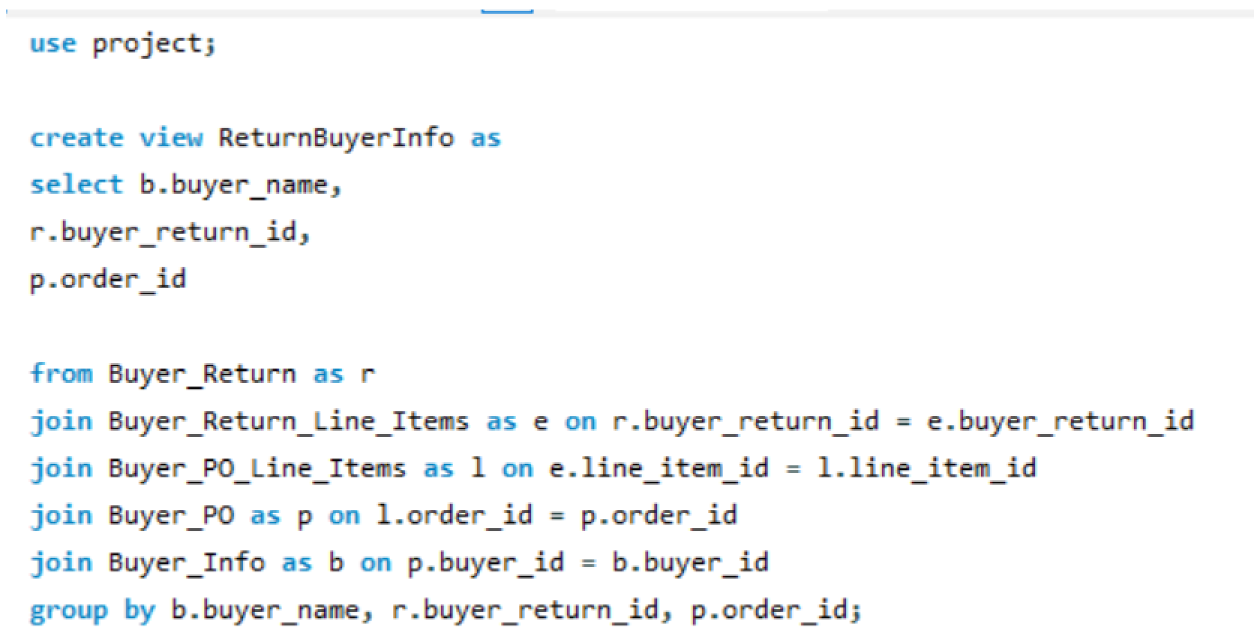
This view is utilized within the GUI to display buyer purchase order and payment status on the payment page. It additionally calls on the orderTotal function to retrieve the total of an order:



```sql
use project;

create view pendingPayments as
select p.buyer_id,
b.buyer_name,
p.order_id,
(select orderTotal(order_id)) as 'total',
p.payment_status

from Buyer_Info as b
join Buyer_PO as p on b.buyer_id = p.buyer_id;
```

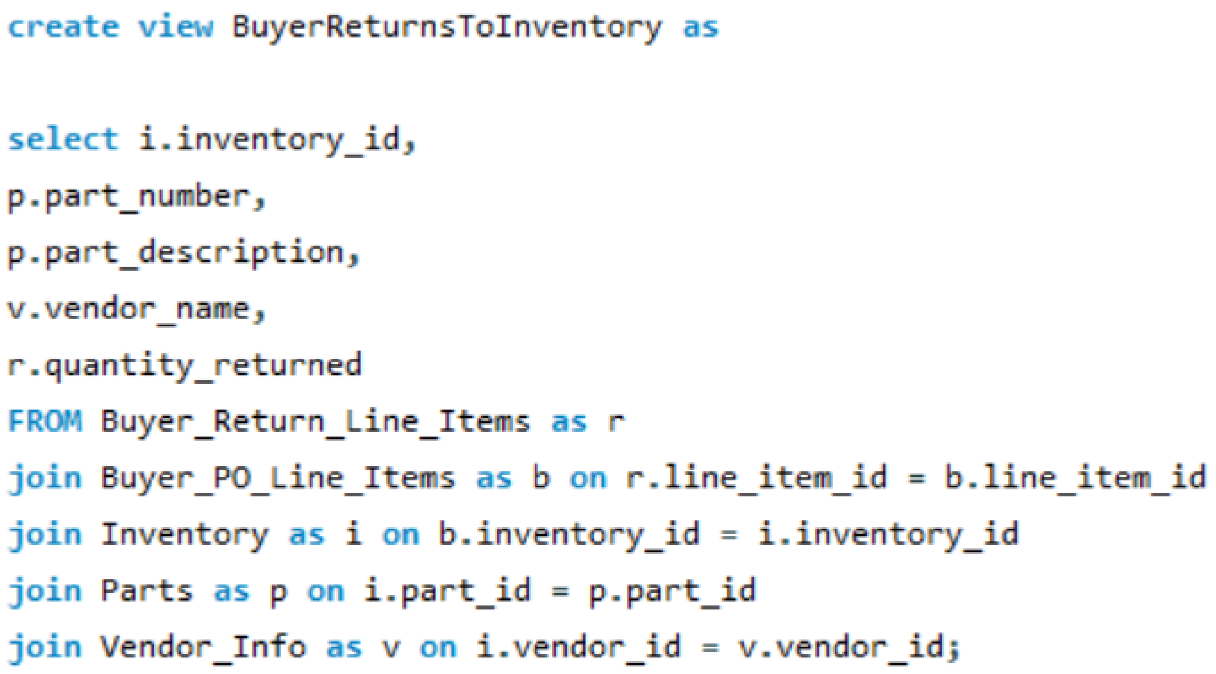
3. View to Return Buyer Name, Return ID, and Original Order ID:

This view returns buyer name, return ID, and the original order ID for each return in the system:



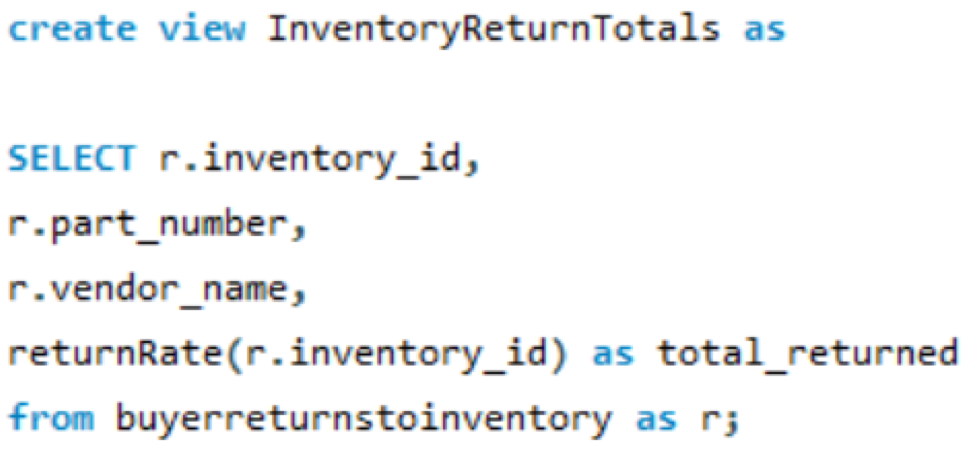
```sql
use project;

create view ReturnBuyerInfo as
select b.buyer_name,
r.buyer_return_id,
p.order_id

from Buyer_Return as r
join Buyer_Return_Line_Items as e on r.buyer_return_id = e.buyer_return_id
join Buyer_PO_Line_Items as l on e.line_item_id = l.line_item_id
join Buyer_PO as p on l.order_id = p.order_id
join Buyer_Info as b on p.buyer_id = b.buyer_id
group by b.buyer_name, r.buyer_return_id, p.order_id;
```

4. View to See Inventory Returned By Buyers and Quantity Returned:



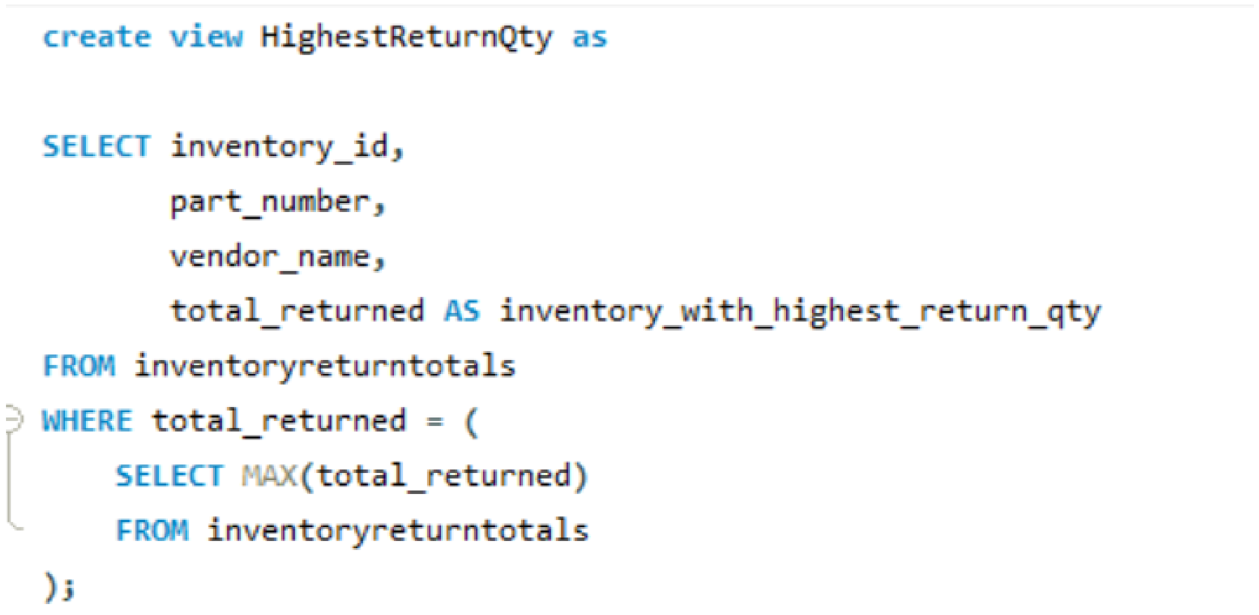
```
create view BuyerReturnsToInventory as

select i.inventory_id,
p.part_number,
p.part_description,
v.vendor_name,
r.quantity_returned
FROM Buyer_Return_Line_Items as r
join Buyer_PO_Line_Items as b on r.line_item_id = b.line_item_id
join Inventory as i on b.inventory_id = i.inventory_id
join Parts as p on i.part_id = p.part_id
join Vendor_Info as v on i.vendor_id = v.vendor_id;
```

5. View to See Inventory Return Totals from Buyers:



```
create view InventoryReturnTotals as

SELECT r.inventory_id,
r.part_number,
r.vendor_name,
returnRate(r.inventory_id) as total_returned
from buyerreturnstoinventory as r;
```

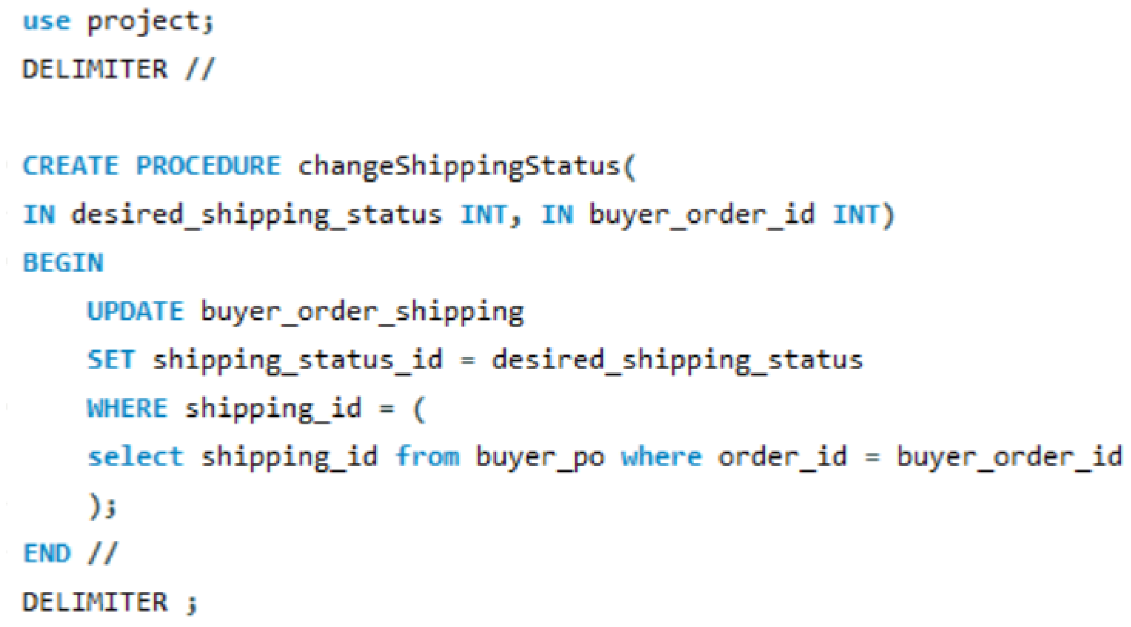
6. View to Retrieve Inventory Item(s) with Highest (Max) Return Rate Due to Highest Return QTY:



```
create view HighestReturnQty as

SELECT inventory_id,
       part_number,
       vendor_name,
       total_returned AS inventory_with_highest_return_qty
FROM inventoryreturntotals
WHERE total_returned = (
    SELECT MAX(total_returned)
    FROM inventoryreturntotals
);
```

7. Stored Procedure to Update Shipping Status:

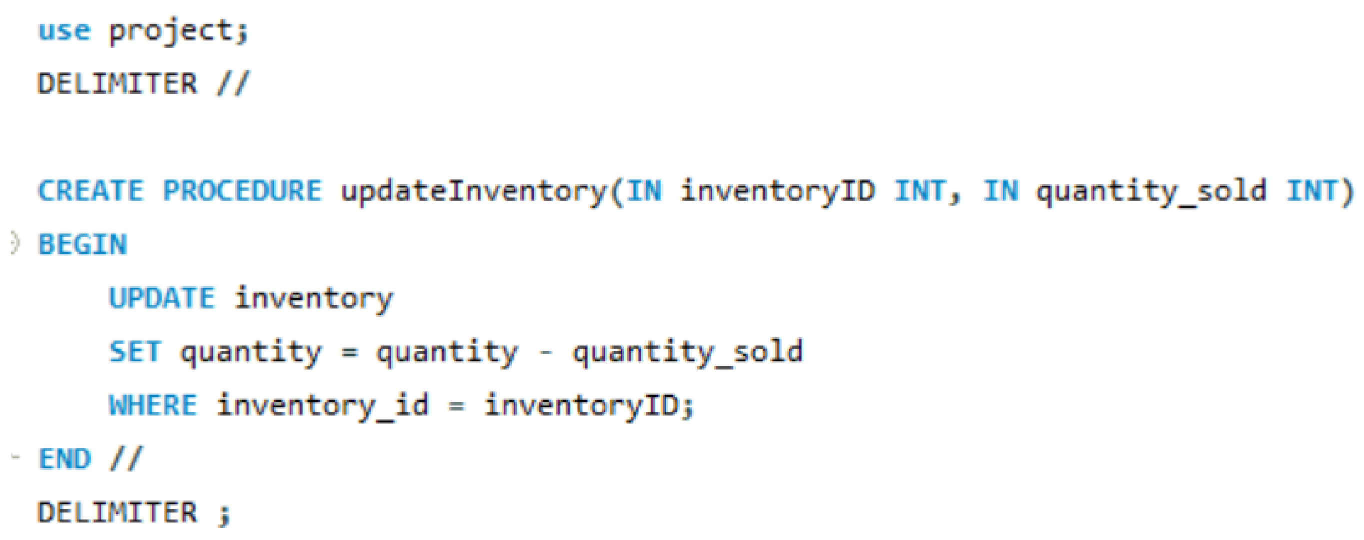
This stored procedure is utilized within the GUI to change the shipping status key of an order once payment has been made, and can be used to update shipping status at other stages within the shipping cycle:



```
use project;
DELIMITER //

CREATE PROCEDURE changeShippingStatus(
IN desired_shipping_status INT, IN buyer_order_id INT)
BEGIN
    UPDATE buyer_order_shipping
    SET shipping_status_id = desired_shipping_status
    WHERE shipping_id = (
    select shipping_id from buyer_po where order_id = buyer_order_id
    );
END //
DELIMITER ;
```

8. Procedure to Update Available Inventory:

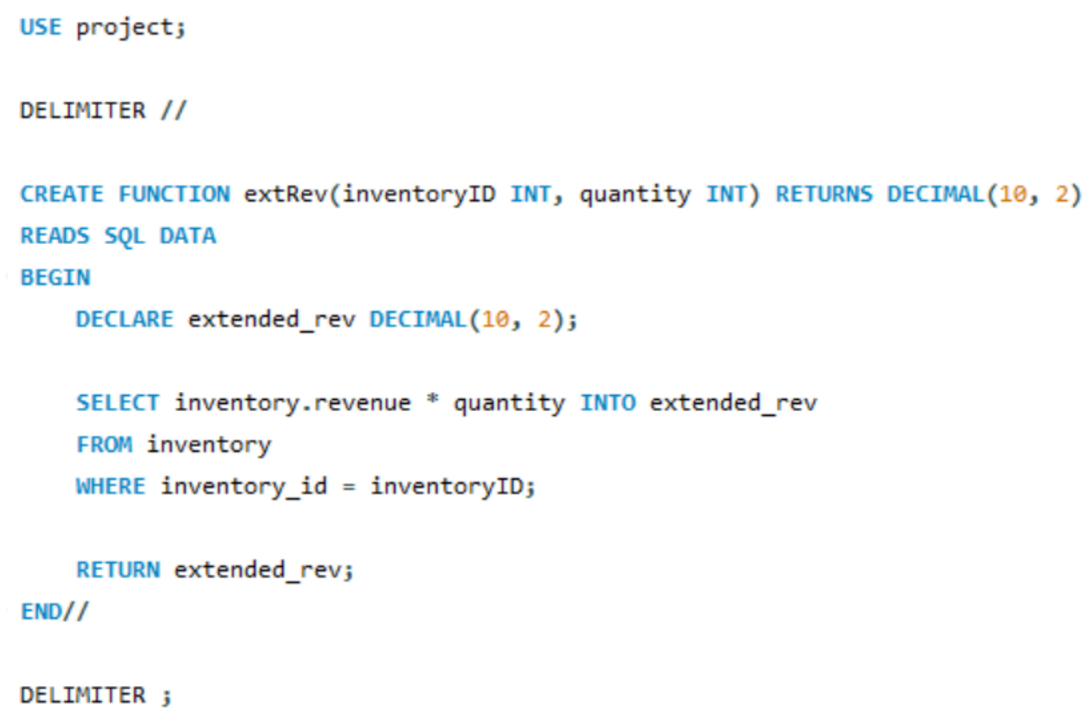
Once an order has been made, this stored procedure subtracts the quantity of an inventory item purchased from the quantity in inventory, and updates the Inventory table:



```
use project;
DELIMITER //

CREATE PROCEDURE updateInventory(IN inventoryID INT, IN quantity_sold INT)
BEGIN
    UPDATE inventory
    SET quantity = quantity - quantity_sold
    WHERE inventory_id = inventoryID;
END //
DELIMITER ;
```

9. Function To Return Extended Revenue:

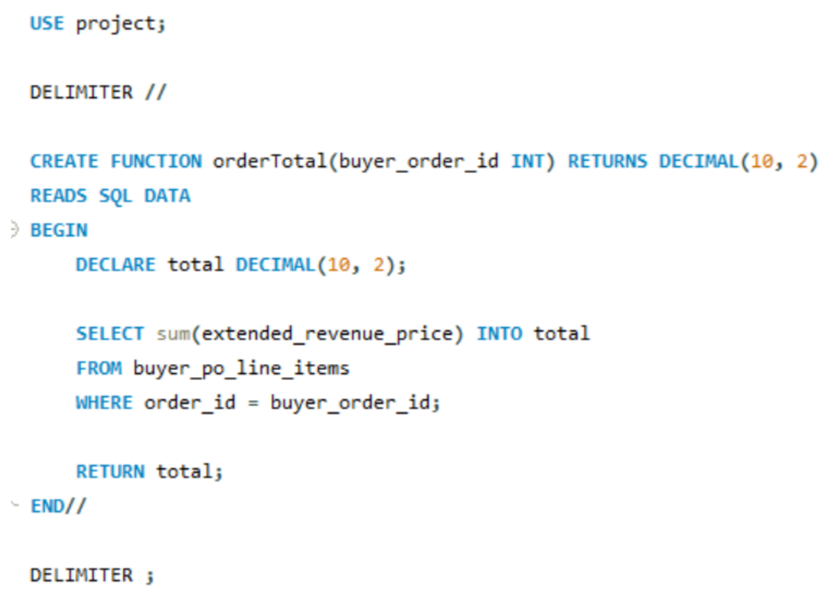
This function is utilized to update the extended_revenue_price field of the Buyer_PO_Line_Items table when an order is being made with inventory line items:



```
USE project;

DELIMITER //

CREATE FUNCTION extRev(inventoryID INT, quantity INT) RETURNS DECIMAL(10, 2)
READS SQL DATA
BEGIN
    DECLARE extended_rev DECIMAL(10, 2);

    SELECT inventory.revenue * quantity INTO extended_rev
    FROM inventory
    WHERE inventory_id = inventoryID;

    RETURN extended_rev;
END//

DELIMITER ;
```

10. Function to Return Total of an Order:

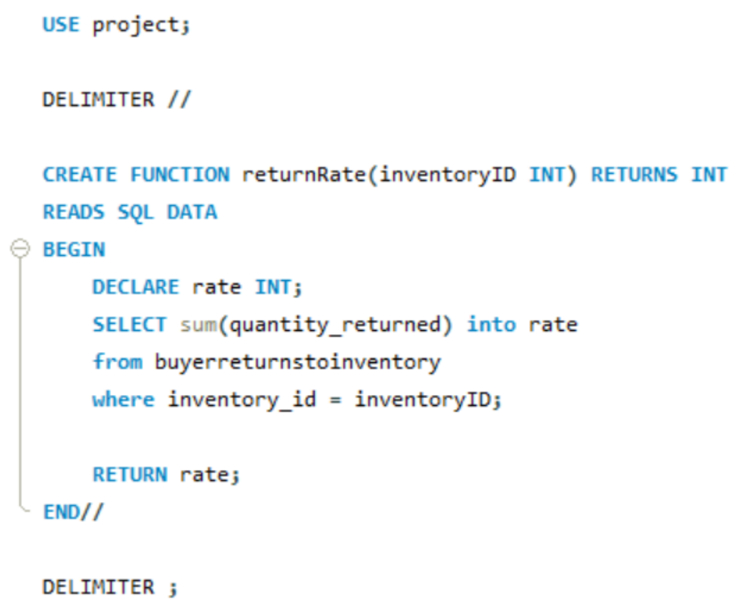
This function is utilized by the GUI to show a buyer’s purchase order total on the update payment status page:



```
USE project;

DELIMITER //

CREATE FUNCTION orderTotal(buyer_order_id INT) RETURNS DECIMAL(10, 2)
READS SQL DATA
BEGIN
    DECLARE total DECIMAL(10, 2);

    SELECT sum(extended_revenue_price) INTO total
    FROM buyer_po_line_items
    WHERE order_id = buyer_order_id;

    RETURN total;
END//

DELIMITER ;
```

11. Function to Retrieve Return Rate of Inventory Item:

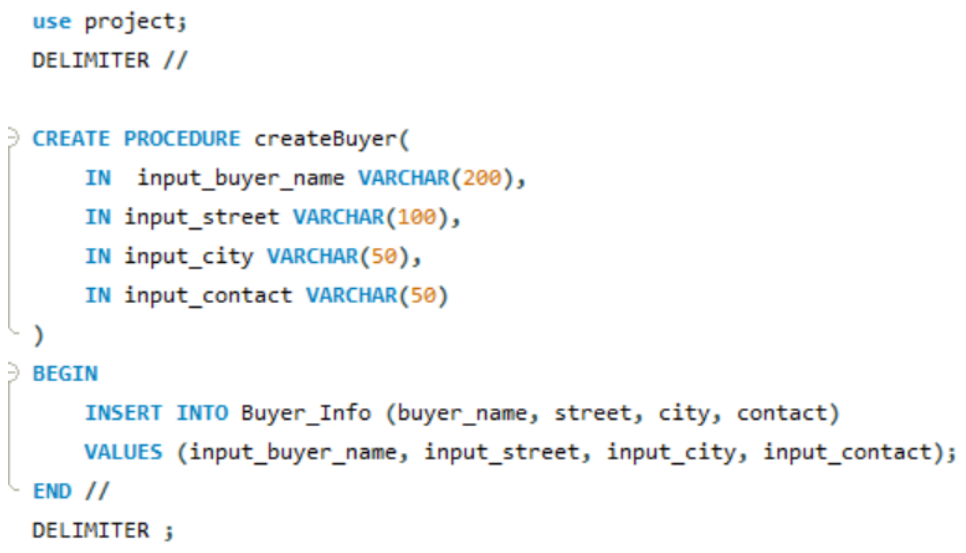
This function is used in the views pertaining to return rate and returns the sum of quantity returned for a particular inventory id number:



```
USE project;

DELIMITER //

CREATE FUNCTION returnRate(inventoryID INT) RETURNS INT
READS SQL DATA
BEGIN
    DECLARE rate INT;
    SELECT sum(quantity_returned) into rate
    from buyerreturnstoinventory
    where inventory_id = inventoryID;

    RETURN rate;
END//

DELIMITER ;
```

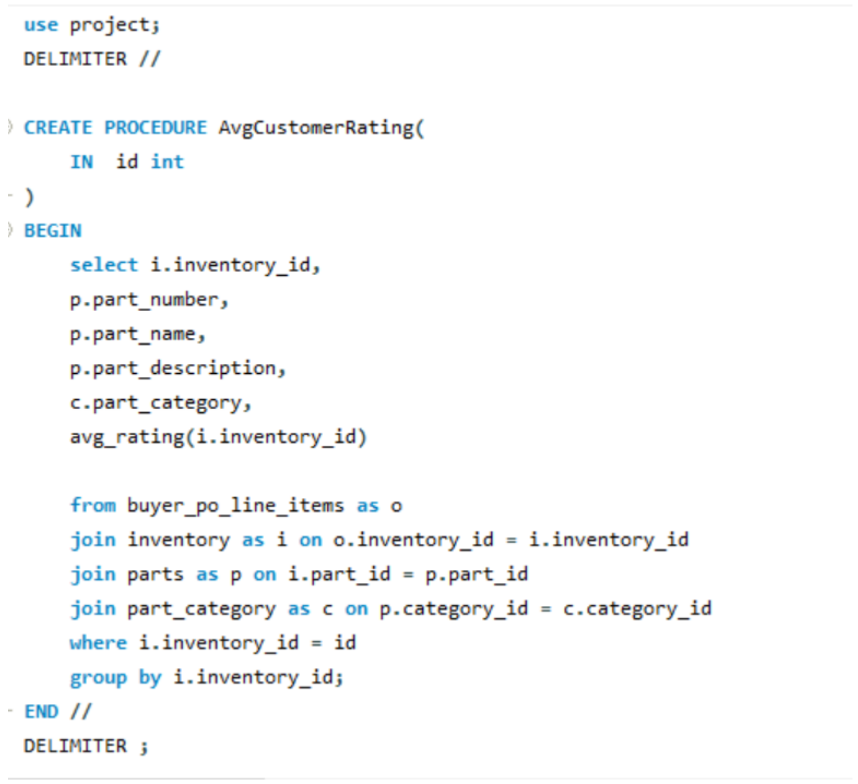
12. Stored Procedure to Create New Buyer Record:

This procedure is utilized in the GUI to create a new buyer in the system:



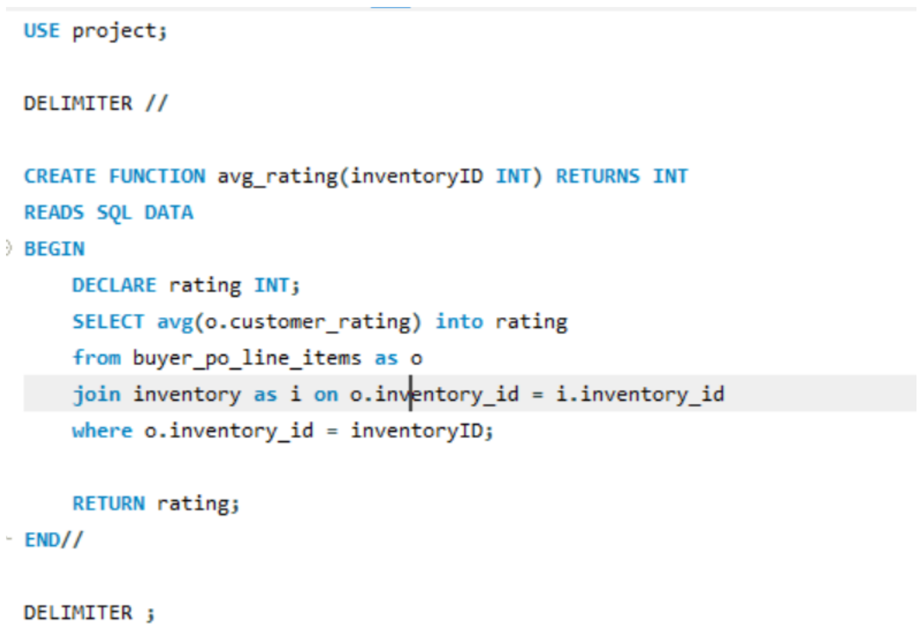
```sql
use project;
DELIMITER //

CREATE PROCEDURE createBuyer(
    IN  input_buyer_name VARCHAR(200),
    IN input_street VARCHAR(100),
    IN input_city VARCHAR(50),
    IN input_contact VARCHAR(50)
)
BEGIN
    INSERT INTO Buyer_Info (buyer_name, street, city, contact)
    VALUES (input_buyer_name, input_street, input_city, input_contact);
END //
DELIMITER ;
```

13. Stored Procedure to Return Inventory Information and Average Customer Rating for Specific Inventory Item Ordered:



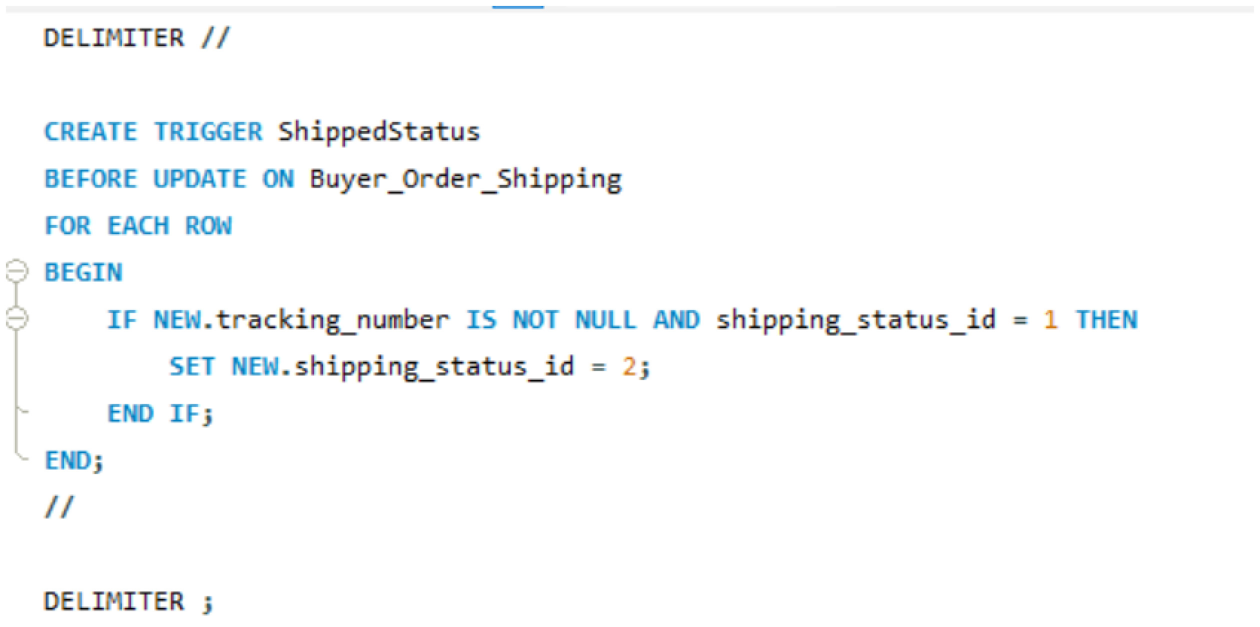
```sql
use project;
DELIMITER //

CREATE PROCEDURE AvgCustomerRating(
    IN  id int
)
BEGIN
    select i.inventory_id,
    p.part_number,
    p.part_name,
    p.part_description,
    c.part_category,
    avg_rating(i.inventory_id)

    from buyer_po_line_items as o
    join inventory as i on o.inventory_id = i.inventory_id
    join parts as p on i.part_id = p.part_id
    join part_category as c on p.category_id = c.category_id
    where i.inventory_id = id
    group by i.inventory_id;
END //
DELIMITER ;
```

14. Function to Return Average Customer Rating For Stored Procedure That Returns Full Information Including Average Customer Rating:



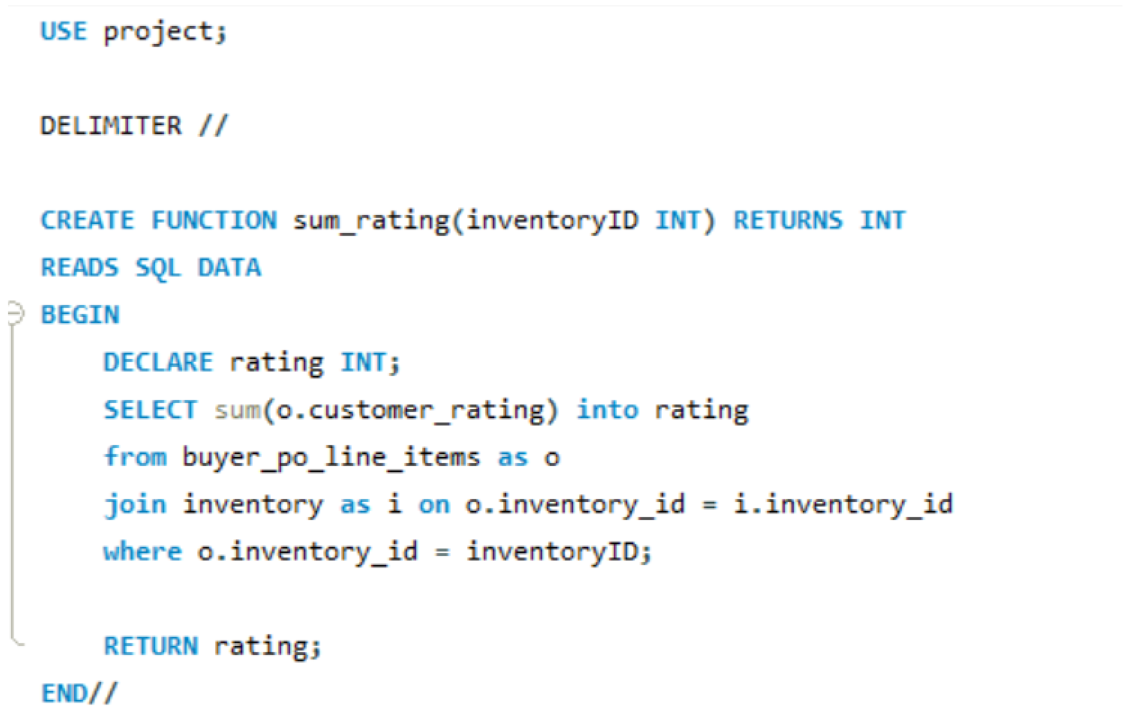
```
USE project;

DELIMITER //

CREATE FUNCTION avg_rating(inventoryID INT) RETURNS INT
READS SQL DATA
BEGIN
    DECLARE rating INT;
    SELECT avg(o.customer_rating) into rating
    from buyer_po_line_items as o
    join inventory as i on o.inventory_id = i.inventory_id
    where o.inventory_id = inventoryID;

    RETURN rating;
END//

DELIMITER ;
```

15. Trigger to Update Shipping Status to Shipped in Buyer_Order_Shipping Table if tracking_id is not null and shipping_status_id = 1:



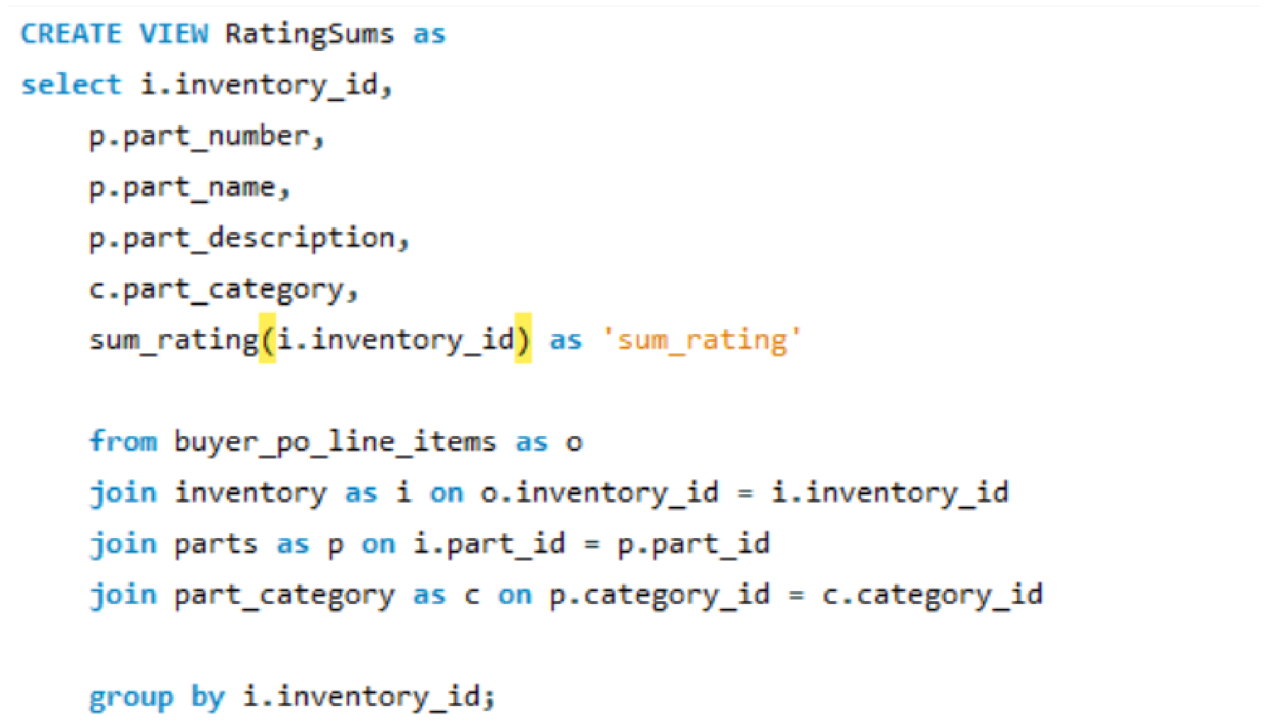
```
DELIMITER //

CREATE TRIGGER ShippedStatus
BEFORE UPDATE ON Buyer_Order_Shipping
FOR EACH ROW
BEGIN
    IF NEW.tracking_number IS NOT NULL AND shipping_status_id = 1 THEN
        SET NEW.shipping_status_id = 2;
    END IF;
END;
//

DELIMITER ;
```

16. Function To Sum Customer Rating Values For Inventory Part:



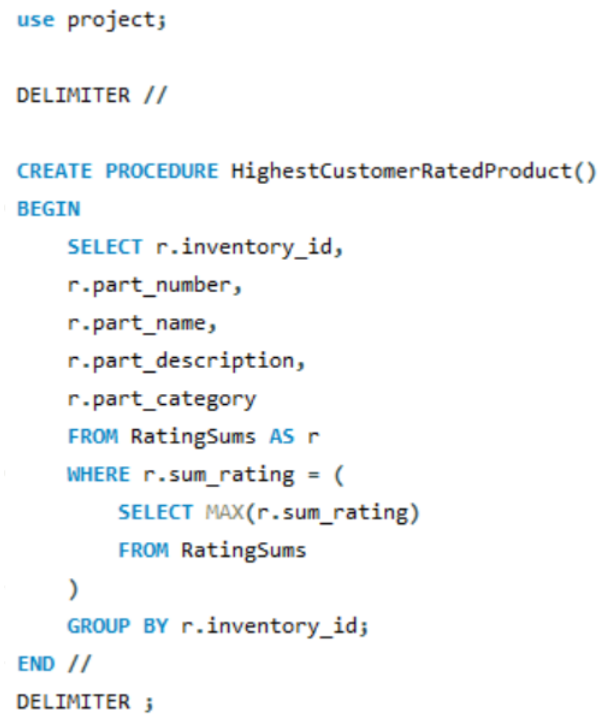
```
USE project;

DELIMITER //

CREATE FUNCTION sum_rating(inventoryID INT) RETURNS INT
READS SQL DATA
BEGIN
    DECLARE rating INT;
    SELECT sum(o.customer_rating) into rating
    from buyer_po_line_items as o
    join inventory as i on o.inventory_id = i.inventory_id
    where o.inventory_id = inventoryID;

    RETURN rating;
END//
```

17. View to Retrieve Sums of Customer Ratings With Inventory and Part Info:



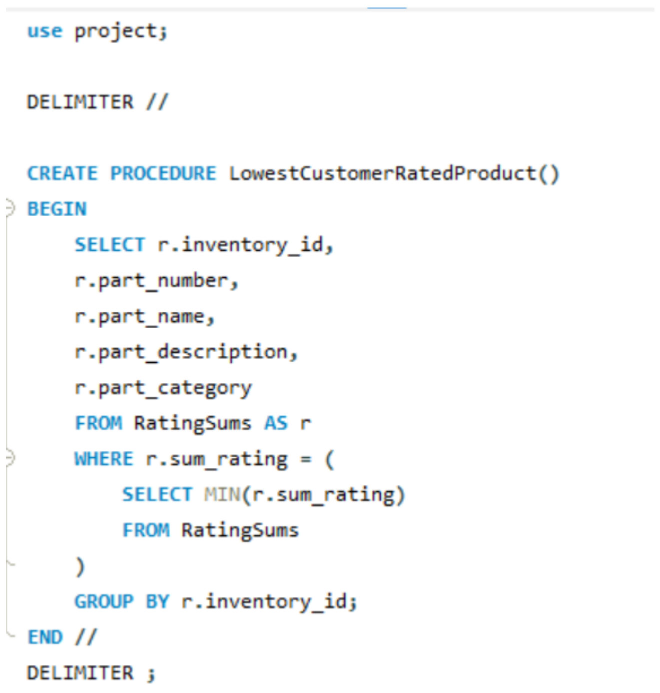
```
CREATE VIEW RatingSums as
select i.inventory_id,
    p.part_number,
    p.part_name,
    p.part_description,
    c.part_category,
    sum_rating(i.inventory_id) as 'sum_rating'

    from buyer_po_line_items as o
    join inventory as i on o.inventory_id = i.inventory_id
    join parts as p on i.part_id = p.part_id
    join part_category as c on p.category_id = c.category_id

    group by i.inventory_id;
```

18. Stored Procedure to Retrieve Inventory Item Information with Highest (Max) Customer Rating:



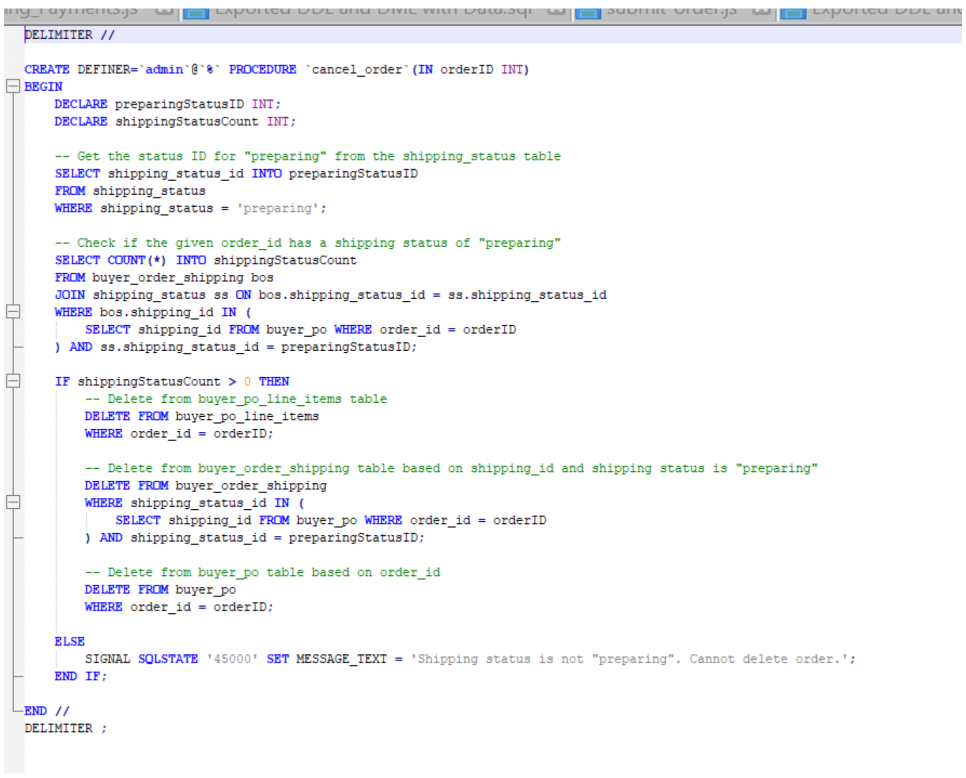
```
use project;

DELIMITER //

CREATE PROCEDURE HighestCustomerRatedProduct()
BEGIN
    SELECT r.inventory_id,
    r.part_number,
    r.part_name,
    r.part_description,
    r.part_category
    FROM RatingSums AS r
    WHERE r.sum_rating = (
        SELECT MAX(r.sum_rating)
        FROM RatingSums
    )
    GROUP BY r.inventory_id;
END //
DELIMITER ;
```

19. Stored Procedure to Retrieve Inventory Item Information with Lowest (Min) Customer Rating:



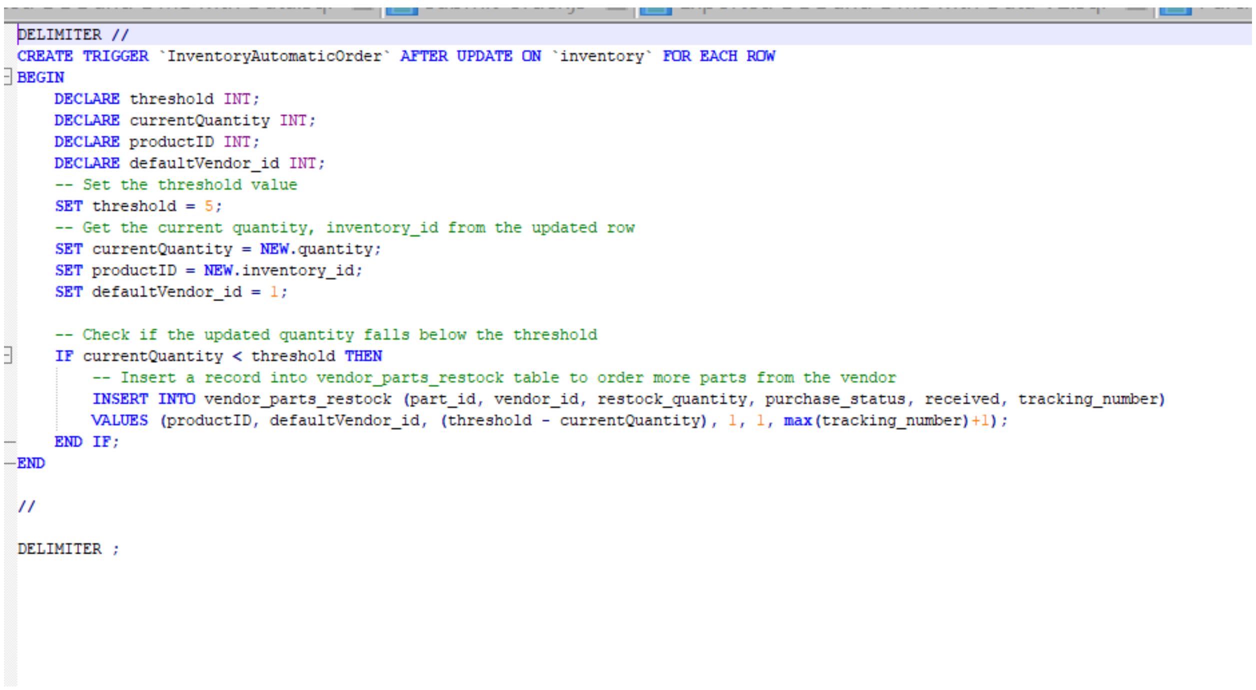
```
use project;

DELIMITER //

CREATE PROCEDURE LowestCustomerRatedProduct()
BEGIN
    SELECT r.inventory_id,
    r.part_number,
    r.part_name,
    r.part_description,
    r.part_category
    FROM RatingSums AS r
    WHERE r.sum_rating = (
        SELECT MIN(r.sum_rating)
        FROM RatingSums
    )
    GROUP BY r.inventory_id;
END //
DELIMITER ;
```

20. Procedure to Cancel Order:



```
DELIMITER //

CREATE DEFINER=`admin`@`%` PROCEDURE `cancel_order`(IN orderID INT)
BEGIN
    DECLARE preparingStatusID INT;
    DECLARE shippingStatusCount INT;

    -- Get the status ID for "preparing" from the shipping_status table
    SELECT shipping_status_id INTO preparingStatusID
    FROM shipping_status
    WHERE shipping_status = 'preparing';

    -- Check if the given order_id has a shipping status of "preparing"
    SELECT COUNT(*) INTO shippingStatusCount
    FROM buyer_order_shipping bos
    JOIN shipping_status ss ON bos.shipping_status_id = ss.shipping_status_id
    WHERE bos.shipping_id IN (
        SELECT shipping_id FROM buyer_po WHERE order_id = orderID
    ) AND ss.shipping_status_id = preparingStatusID;

    IF shippingStatusCount > 0 THEN
        -- Delete from buyer_po_line_items table
        DELETE FROM buyer_po_line_items
        WHERE order_id = orderID;

        -- Delete from buyer_order_shipping table based on shipping_id and shipping status is "preparing"
        DELETE FROM buyer_order_shipping
        WHERE shipping_status_id IN (
            SELECT shipping_id FROM buyer_po WHERE order_id = orderID
        ) AND shipping_status_id = preparingStatusID;

        -- Delete from buyer_po table based on order_id
        DELETE FROM buyer_po
        WHERE order_id = orderID;

    ELSE
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Shipping status is not "preparing". Cannot delete order.';
    END IF;

END //
DELIMITER ;
```

21. Trigger to Complete Vendor Restock if Inventory Quantity Drops Below 5:



```
DELIMITER //
CREATE TRIGGER `InventoryAutomaticOrder` AFTER UPDATE ON `inventory` FOR EACH ROW
BEGIN
    DECLARE threshold INT;
    DECLARE currentQuantity INT;
    DECLARE productID INT;
    DECLARE defaultVendor_id INT;
    -- Set the threshold value
    SET threshold = 5;
    -- Get the current quantity, inventory_id from the updated row
    SET currentQuantity = NEW.quantity;
    SET productID = NEW.inventory_id;
    SET defaultVendor_id = 1;

    -- Check if the updated quantity falls below the threshold
    IF currentQuantity < threshold THEN
        -- Insert a record into vendor_parts_restock table to order more parts from the vendor
        INSERT INTO vendor_parts_restock (part_id, vendor_id, restock_quantity, purchase_status, received, tracking_number)
        VALUES (productID, defaultVendor_id, (threshold - currentQuantity), 1, 1, max(tracking_number)+1);
    END IF;
END

//

DELIMITER ;
```

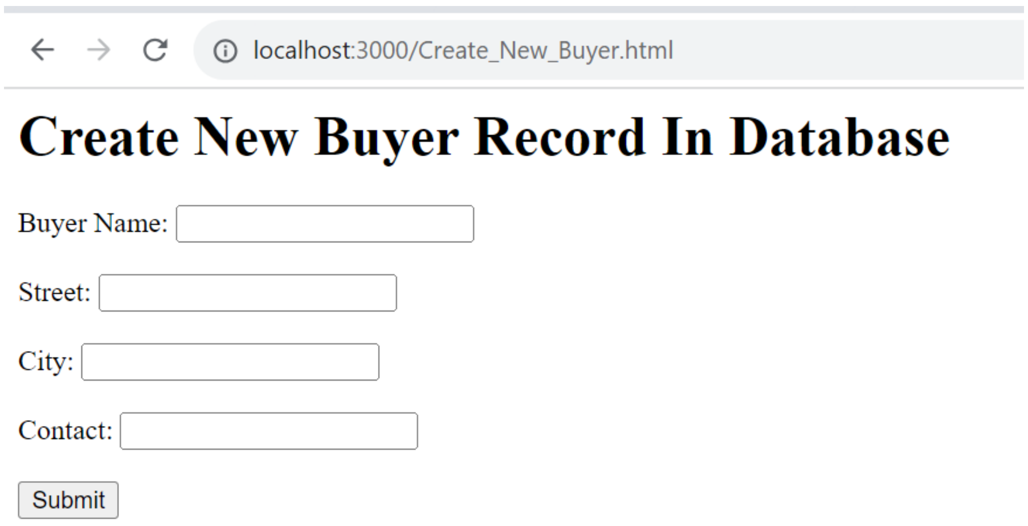
System GUI/CLI

System GUI / CLI (10pts) – Your database system should be integrated into a working system, which allows users to insert, modify and/or delete data and information from the database.

1. Create New Buyer:

Buyer information entered into fields are captured by an HTML file Create_New_Buyer.html, and are sent to the javascript file saveBuyerInfo.js:

Front-end GUI:



localhost:3000/Create_New_Buyer.html

Create New Buyer Record In Database

Buyer Name:

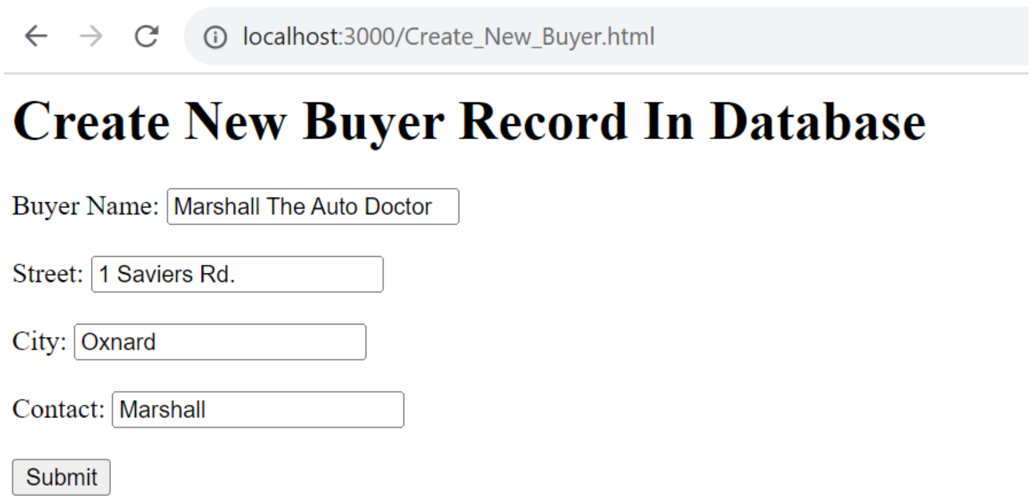
Street:

City:

Contact:

Submit

Info Submitted:



localhost:3000/Create_New_Buyer.html

Create New Buyer Record In Database

Buyer Name: Marshall The Auto Doctor

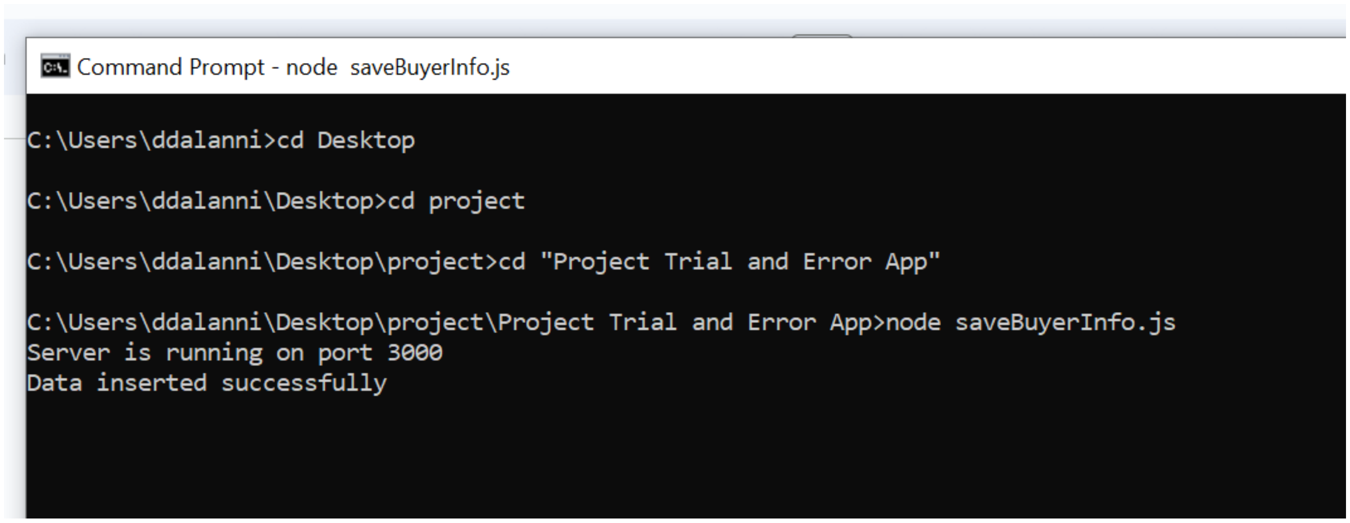
Street: 1 Saviers Rd.

City: Oxnard

Contact: Marshall

Submit

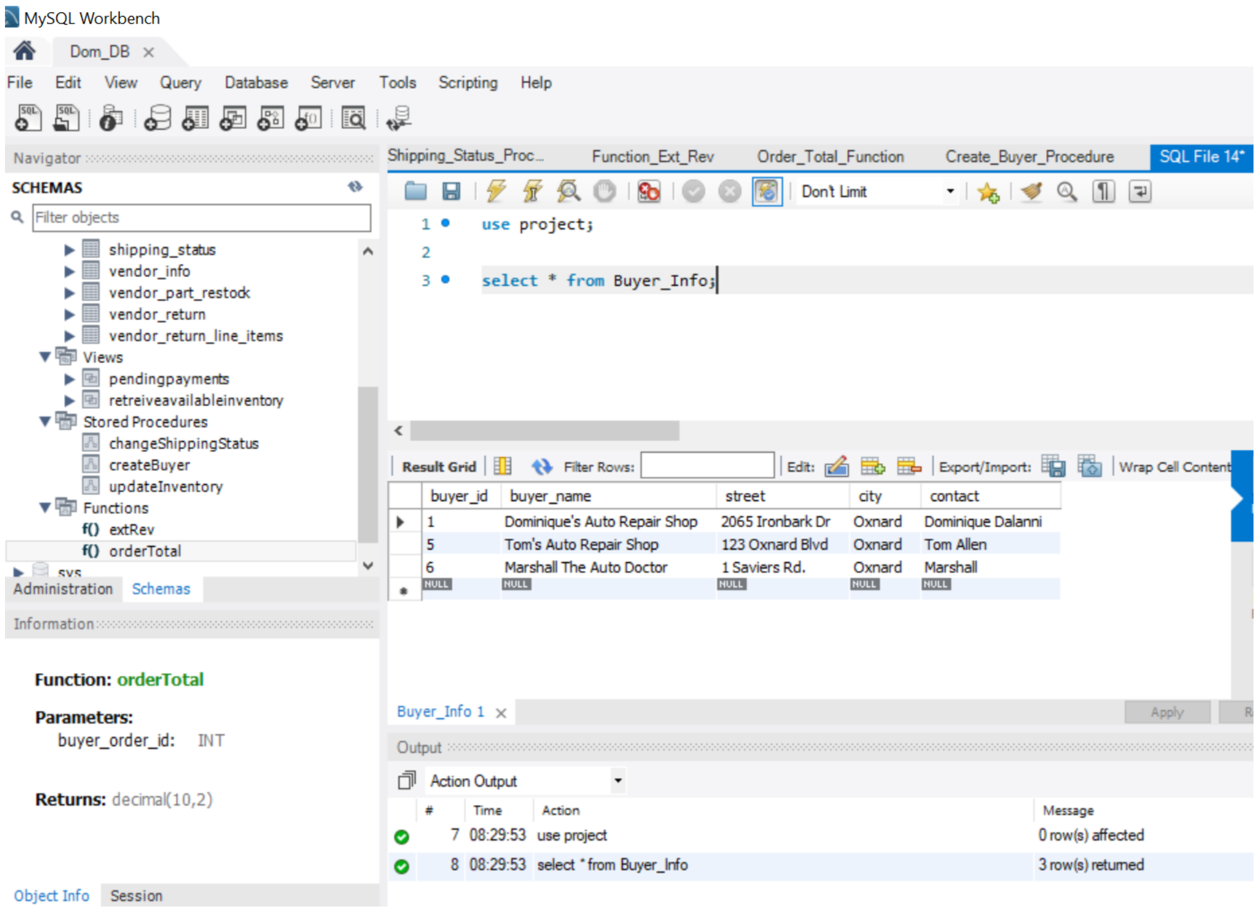
Backend command line response:



Command Prompt - node saveBuyerInfo.js

```
C:\Users\ddalanni>cd Desktop

C:\Users\ddalanni\Desktop>cd project

C:\Users\ddalanni\Desktop\project>cd "Project Trial and Error App"

C:\Users\ddalanni\Desktop\project\Project Trial and Error App>node saveBuyerInfo.js
Server is running on port 3000
Data inserted successfully
```

Database record showing update:



Function: orderTotal

Parameters:
buyer_order_id: INT

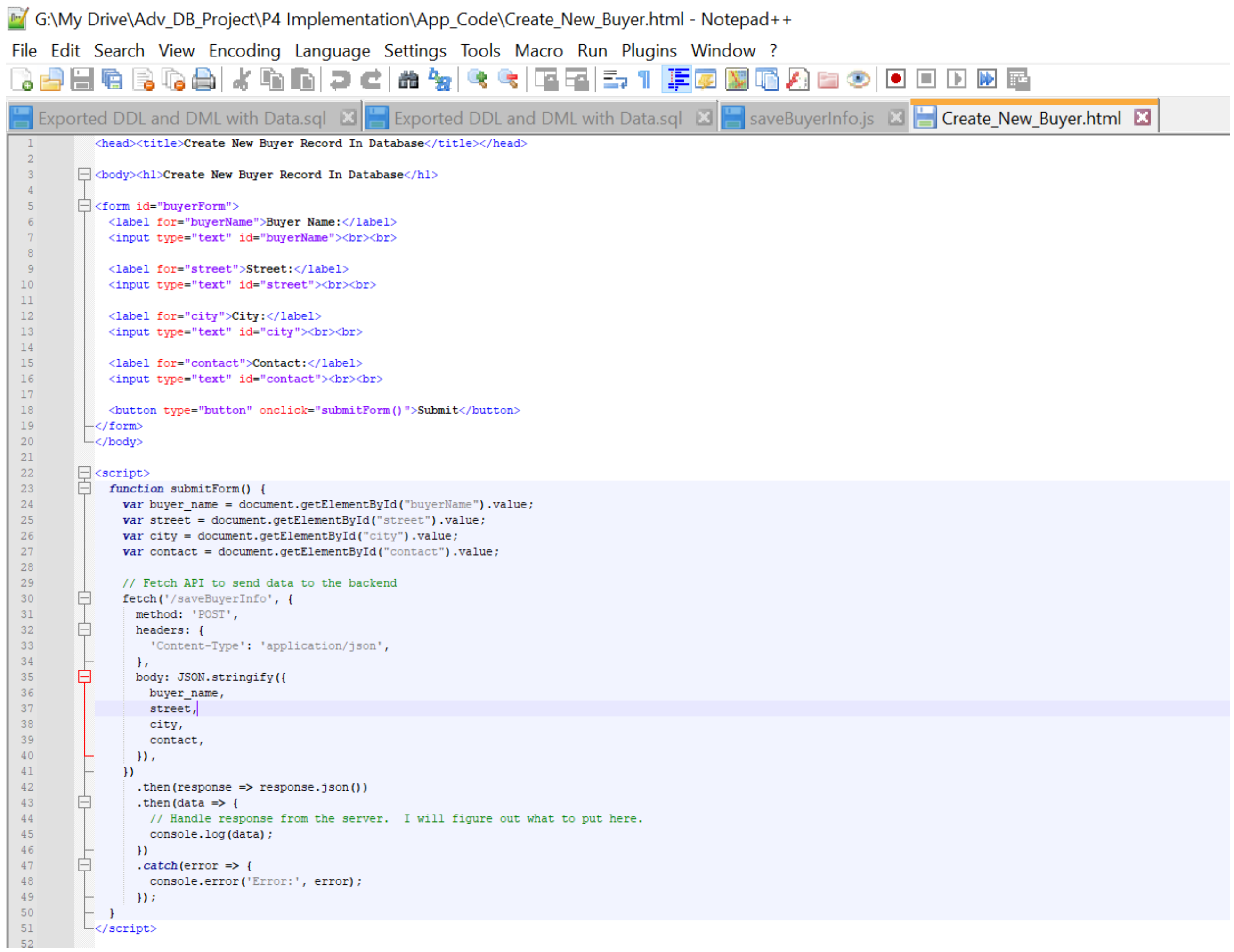
Returns: decimal(10,2)

```sql
use project;

select * from Buyer_Info;
```

buyer_id	buyer_name	street	city	contact
1	Dominique's Auto Repair Shop	2065 Ironbark Dr	Oxnard	Dominique Dalanni
5	Tom's Auto Repair Shop	123 Oxnard Blvd	Oxnard	Tom Allen
6	Marshall The Auto Doctor	1 Saviers Rd.	Oxnard	Marshall
NULL	NULL	NULL	NULL	NULL

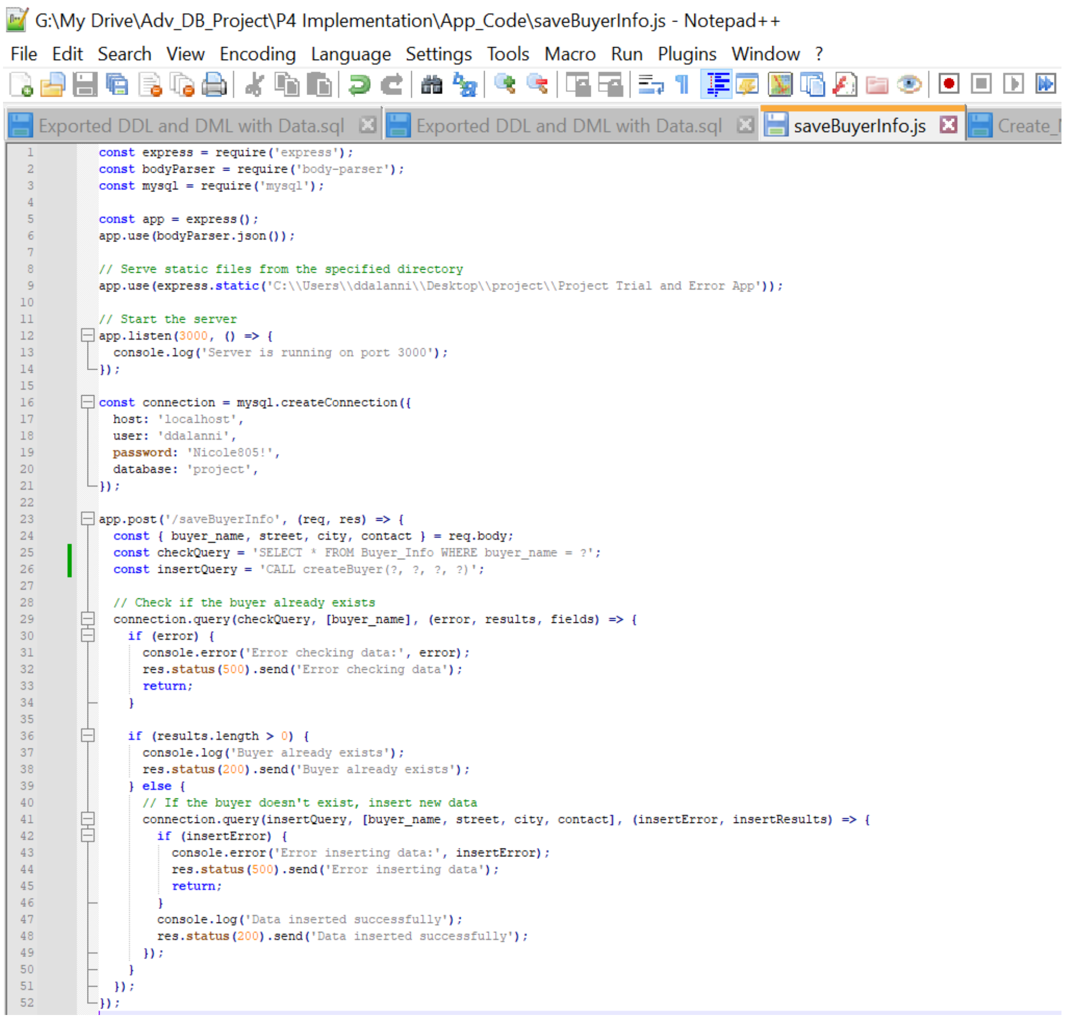
#	Time	Action	Message
7	08:29:53	use project	0 row(s) affected
8	08:29:53	select * from Buyer_Info	3 row(s) returned

HTML Code:


G:\My Drive\Adv_DB_Project\P4 Implementation\App_Code\Create_New_Buyer.html - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Exported DDL and DML with Data.sql
Exported DDL and DML with Data.sql
saveBuyerInfo.js
Create_New_Buyer.html

```html
<head><title>Create New Buyer Record In Database</title></head>

<body><h1>Create New Buyer Record In Database</h1>

<form id="buyerForm">
  <label for="buyerName">Buyer Name:</label>
  <input type="text" id="buyerName"><br><br>

  <label for="street">Street:</label>
  <input type="text" id="street"><br><br>

  <label for="city">City:</label>
  <input type="text" id="city"><br><br>

  <label for="contact">Contact:</label>
  <input type="text" id="contact"><br><br>

  <button type="button" onclick="submitForm()">Submit</button>
</form>
</body>

<script>
  function submitForm() {
    var buyer_name = document.getElementById("buyerName").value;
    var street = document.getElementById("street").value;
    var city = document.getElementById("city").value;
    var contact = document.getElementById("contact").value;

    // Fetch API to send data to the backend
    fetch('/saveBuyerInfo', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({
        buyer_name,
        street,
        city,
        contact,
      }),
    })
      .then(response => response.json())
      .then(data => {
        // Handle response from the server.  I will figure out what to put here.
        console.log(data);
      })
      .catch(error => {
        console.error('Error:', error);
      });
  }
</script>
```

Javascript backend:



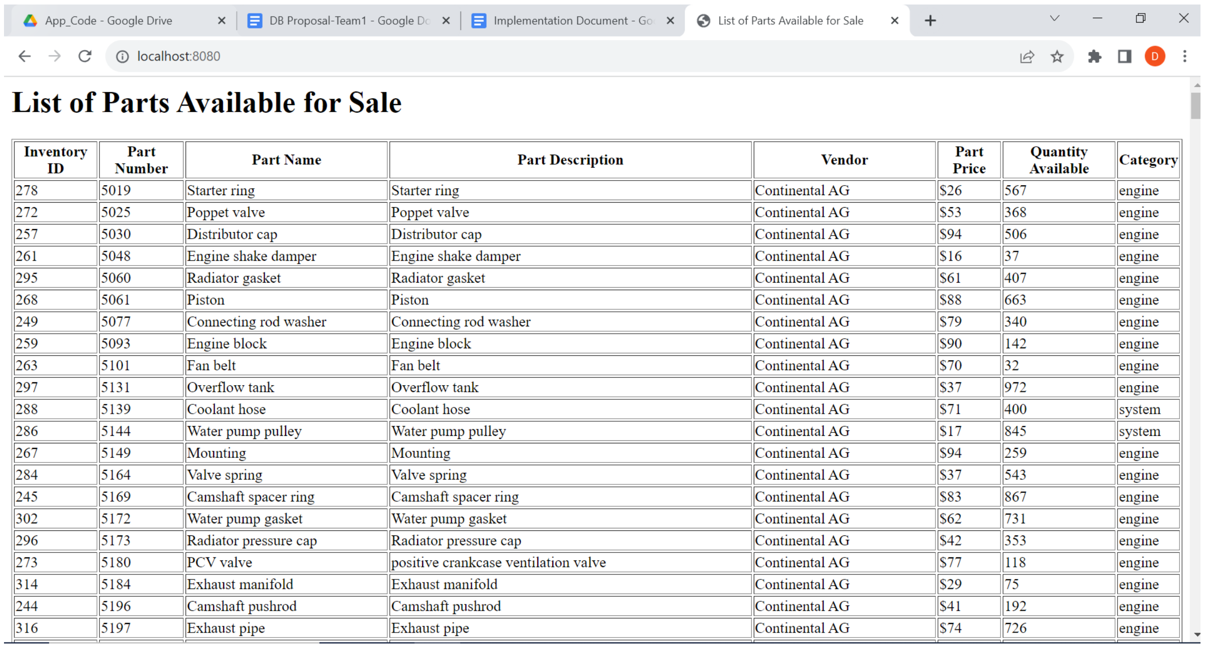
G:\My Drive\Adv_DB_Project\P4 Implementation\App_Code\saveBuyerInfo.js - Notepad++

```javascript
const express = require('express');
const bodyParser = require('body-parser');
const mysql = require('mysql');

const app = express();
app.use(bodyParser.json());

// Serve static files from the specified directory
app.use(express.static('C:\\Users\\ddalanni\\Desktop\\project\\Project Trial and Error App'));

// Start the server
app.listen(3000, () => {
  console.log('Server is running on port 3000');
});

const connection = mysql.createConnection({
  host: 'localhost',
  user: 'ddalanni',
  password: 'Nicole805!',
  database: 'project',
});

app.post('/saveBuyerInfo', (req, res) => {
  const { buyer_name, street, city, contact } = req.body;
  const checkQuery = 'SELECT * FROM Buyer_Info WHERE buyer_name = ?';
  const insertQuery = 'CALL createBuyer(?, ?, ?, ?)';

  // Check if the buyer already exists
  connection.query(checkQuery, [buyer_name], (error, results, fields) => {
    if (error) {
      console.error('Error checking data:', error);
      res.status(500).send('Error checking data');
      return;
    }

    if (results.length > 0) {
      console.log('Buyer already exists');
      res.status(200).send('Buyer already exists');
    } else {
      // If the buyer doesn't exist, insert new data
      connection.query(insertQuery, [buyer_name, street, city, contact], (insertError, insertResults) => {
        if (insertError) {
          console.error('Error inserting data:', insertError);
          res.status(500).send('Error inserting data');
          return;
        }
        console.log('Data inserted successfully');
        res.status(200).send('Data inserted successfully');
      });
    }
  });
});
```

2. View Available Inventory:

This page allows users to view all available inventory and the quantities available of each. This script was the first one we created and has HTML and Javascript code in one. This script relies on the RetreiveAvailableInventory view. The filename is “view_inventory.js” and the screenshots of the page and code can be found below:

Webpage:

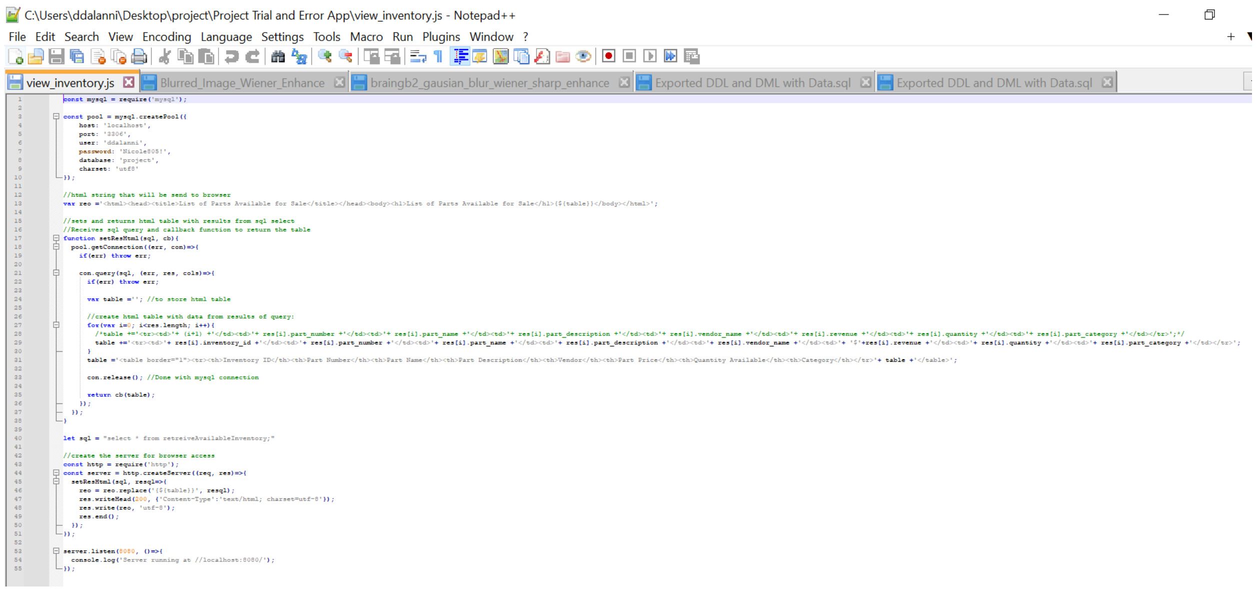


App_Code - Google Drive
DB Proposal-Team1 - Google Do
Implementation Document - Go
List of Parts Available for Sale
localhost:8080

List of Parts Available for Sale

Inventory ID	Part Number	Part Name	Part Description	Vendor	Part Price	Quantity Available	Category
278	5019	Starter ring	Starter ring	Continental AG	$26	567	engine
272	5025	Poppet valve	Poppet valve	Continental AG	$53	368	engine
257	5030	Distributor cap	Distributor cap	Continental AG	$94	506	engine
261	5048	Engine shake damper	Engine shake damper	Continental AG	$16	37	engine
295	5060	Radiator gasket	Radiator gasket	Continental AG	$61	407	engine
268	5061	Piston	Piston	Continental AG	$88	663	engine
249	5077	Connecting rod washer	Connecting rod washer	Continental AG	$79	340	engine
259	5093	Engine block	Engine block	Continental AG	$90	142	engine
263	5101	Fan belt	Fan belt	Continental AG	$70	32	engine
297	5131	Overflow tank	Overflow tank	Continental AG	$37	972	engine
288	5139	Coolant hose	Coolant hose	Continental AG	$71	400	system
286	5144	Water pump pulley	Water pump pulley	Continental AG	$17	845	system
267	5149	Mounting	Mounting	Continental AG	$94	259	engine
284	5164	Valve spring	Valve spring	Continental AG	$37	543	engine
245	5169	Camshaft spacer ring	Camshaft spacer ring	Continental AG	$83	867	engine
302	5172	Water pump gasket	Water pump gasket	Continental AG	$62	731	engine
296	5173	Radiator pressure cap	Radiator pressure cap	Continental AG	$42	353	engine
273	5180	PCV valve	positive crankcase ventilation valve	Continental AG	$77	118	engine
314	5184	Exhaust manifold	Exhaust manifold	Continental AG	$29	75	engine
244	5196	Camshaft pushrod	Camshaft pushrod	Continental AG	$41	192	engine
316	5197	Exhaust pipe	Exhaust pipe	Continental AG	$74	726	engine

Javascript/HTML Code:



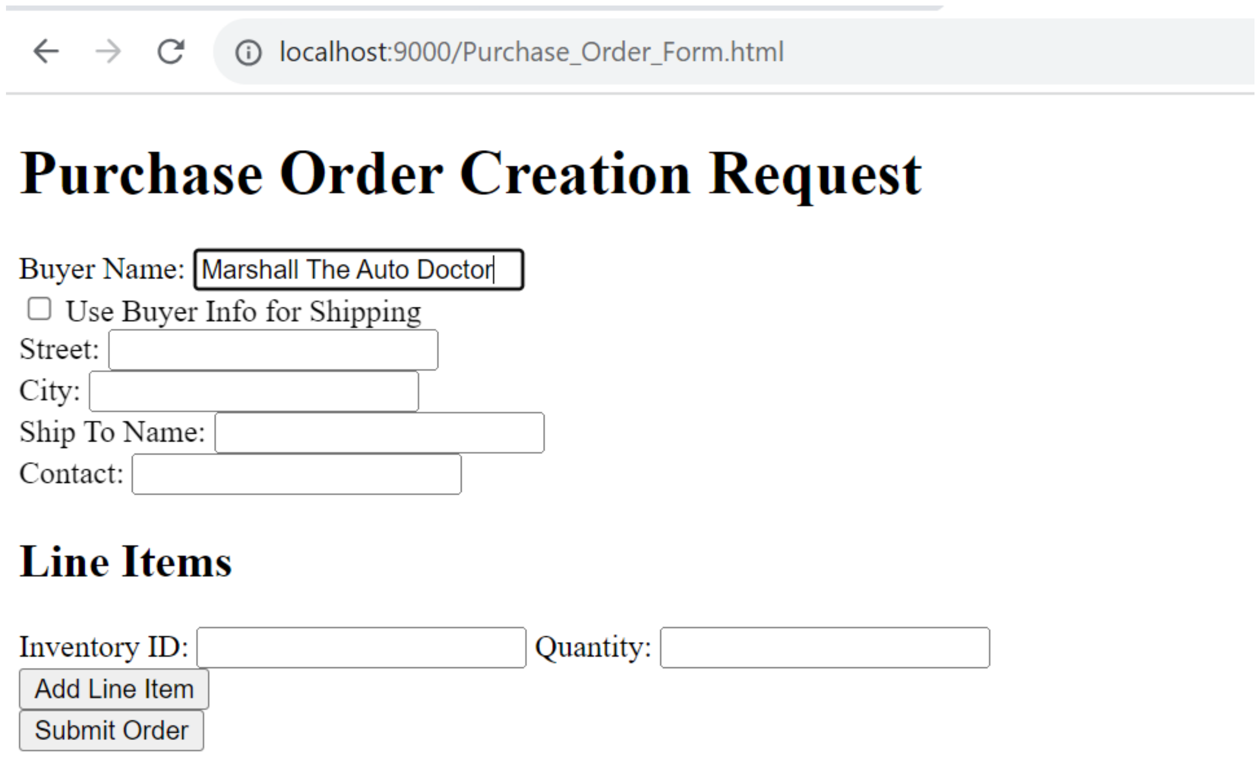
C:\Users\ddalanni\Desktop\project\Project Trial and Error App\view_inventory.js - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
view_inventory.js
Blurred_Image_Wiener_Enhance
braingb2_gausian_blur_wiener_sharp_enhance
Exported DDL and DML with Data.sql
Exported DDL and DML with Data.sql

```
const mysql = require('mysql');

const pool = mysql.createPool({
    host: 'localhost',
    port: '3306',
    user: 'ddalanni',
    password: 'Nicole805!',
    database: 'project',
    charset: 'utf8'
});

//html string that will be send to browser
var reo ='<html><head><title>List of Parts Available for Sale</title></head><body><h1>List of Parts Available for Sale</h1>{${table}}</body></html>';

//sets and returns html table with results from sql select
//Receives sql query and callback function to return the table
function setResHtml(sql, cb){
  pool.getConnection((err, con)=>{
    if(err) throw err;

    con.query(sql, (err, res, cols)=>{
      if(err) throw err;

      var table =''; //to store html table

      //create html table with data from results of query:
      for(var i=0; i<res.length; i++){
        /*table +='<tr><td>'+ (i+1) +'</td><td>'+ res[i].part_number +'</td><td>'+ res[i].part_name +'</td><td>'+ res[i].part_description +'</td><td>'+ res[i].vendor_name +'</td><td>'+ res[i].revenue +'</td><td>'+ res[i].quantity +'</td><td>'+ res[i].part_category +'</td></tr>';*/
        table +='<tr><td>'+ res[i].inventory_id +'</td><td>'+ res[i].part_number +'</td><td>'+ res[i].part_name +'</td><td>'+ res[i].part_description +'</td><td>'+ res[i].vendor_name +'</td><td>'+ '$'+res[i].revenue +'</td><td>'+ res[i].quantity +'</td><td>'+ res[i].part_category +'</td></tr>';
      }
      table ='<table border="1"><tr><th>Inventory ID</th><th>Part Number</th><th>Part Name</th><th>Part Description</th><th>Vendor</th><th>Part Price</th><th>Quantity Available</th><th>Category</th></tr>'+ table +'</table>';

      con.release(); //Done with mysql connection

      return cb(table);
    });
  });
}

let sql = "select * from retreiveAvailableInventory;"

//create the server for browser access
const http = require('http');
const server = http.createServer((req, res)=>{
  setResHtml(sql, resql=>{
    reo = reo.replace('{${table}}', resql);
    res.writeHead(200, {'Content-Type':'text/html; charset=utf-8'});
    res.write(reo, 'utf-8');
    res.end();
  });
});

server.listen(8080, ()=>{
  console.log('Server running at //localhost:8080/');
});
```

3. Create Buyer Purchase Order:

This webpage gives buyers the option to ship to the location they have on file, or ship to a different address. They are able to add multiple line items with a specified quantity of inventory. While the javascript backend checks whether or not a buyer is in the system, a future release will also confirm that a buyer is not purchasing more items than available in stock. This page relies on an HTML file called Purchase_Order_Form.html on the front end, and a Javascript file on the backend called submit-order.js. Screenshots of the form and code can be found below:

Webpage:



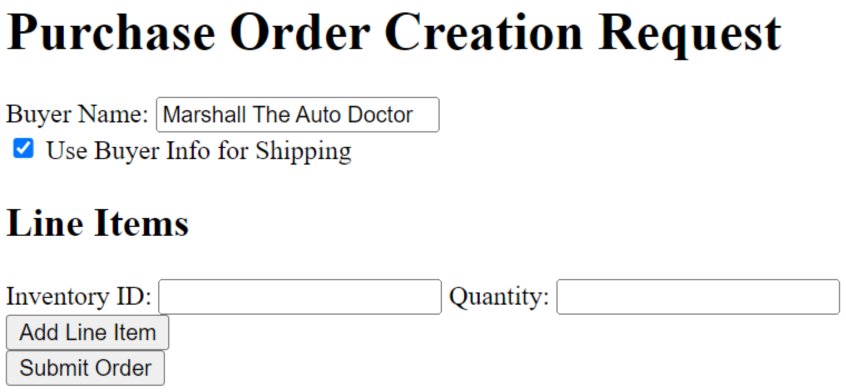
localhost:9000/Purchase_Order_Form.html

Purchase Order Creation Request

Buyer Name: Marshall The Auto Doctor
☐ Use Buyer Info for Shipping
Street:
City:
Ship To Name:
Contact:

Line Items

Inventory ID: Quantity:

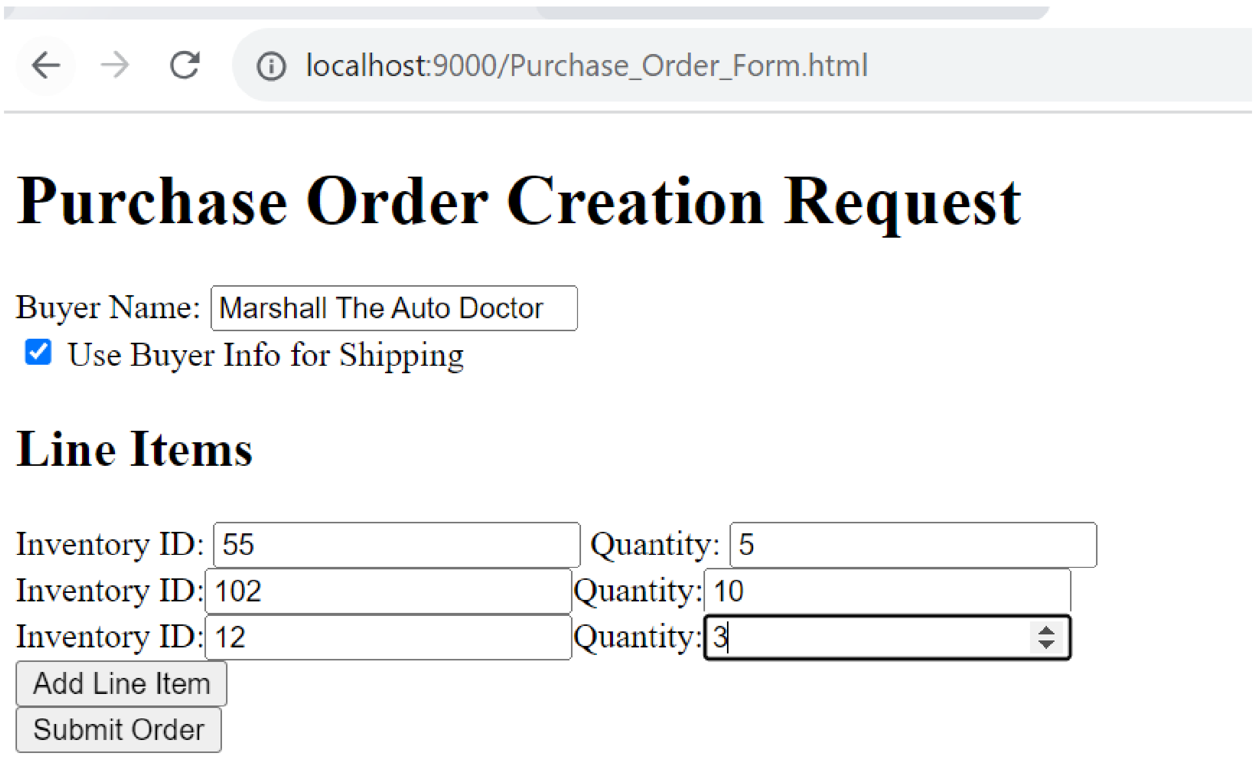


Purchase Order Creation Request

Buyer Name: Marshall The Auto Doctor
☑ Use Buyer Info for Shipping

Line Items

Inventory ID: Quantity:

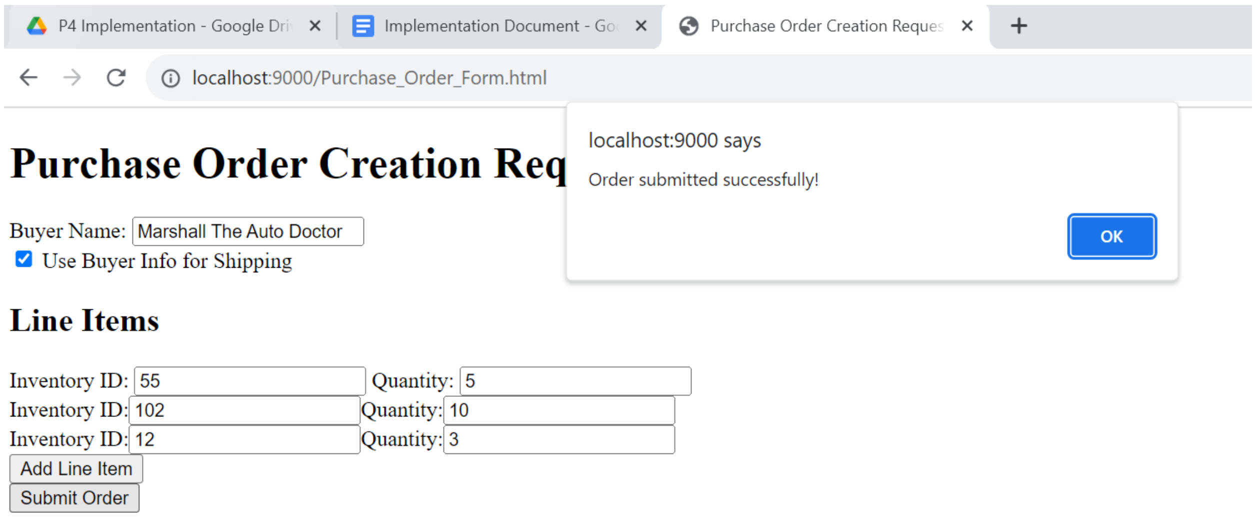

localhost:9000/Purchase_Order_Form.html

Purchase Order Creation Request

Buyer Name: Marshall The Auto Doctor
☑ Use Buyer Info for Shipping

Line Items

Inventory ID: 55 Quantity: 5
Inventory ID: 102 Quantity: 10
Inventory ID: 12 Quantity: 3


P4 Implementation - Google Dri
Implementation Document - Go
Purchase Order Creation Reques

localhost:9000/Purchase_Order_Form.html

localhost:9000 says
Order submitted successfully!

Purchase Order Creation Req

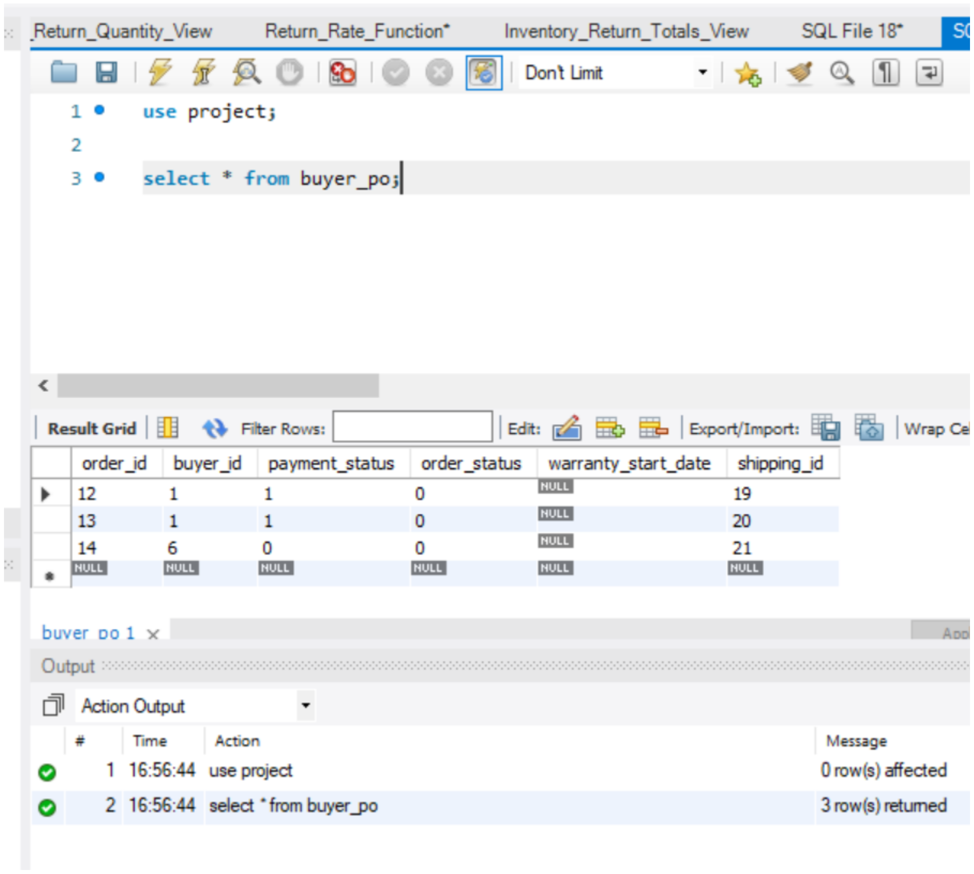
Buyer Name: Marshall The Auto Doctor
☑ Use Buyer Info for Shipping

Line Items

Inventory ID: 55 Quantity: 5
Inventory ID: 102 Quantity: 10
Inventory ID: 12 Quantity: 3

Add Line Item
Submit Order

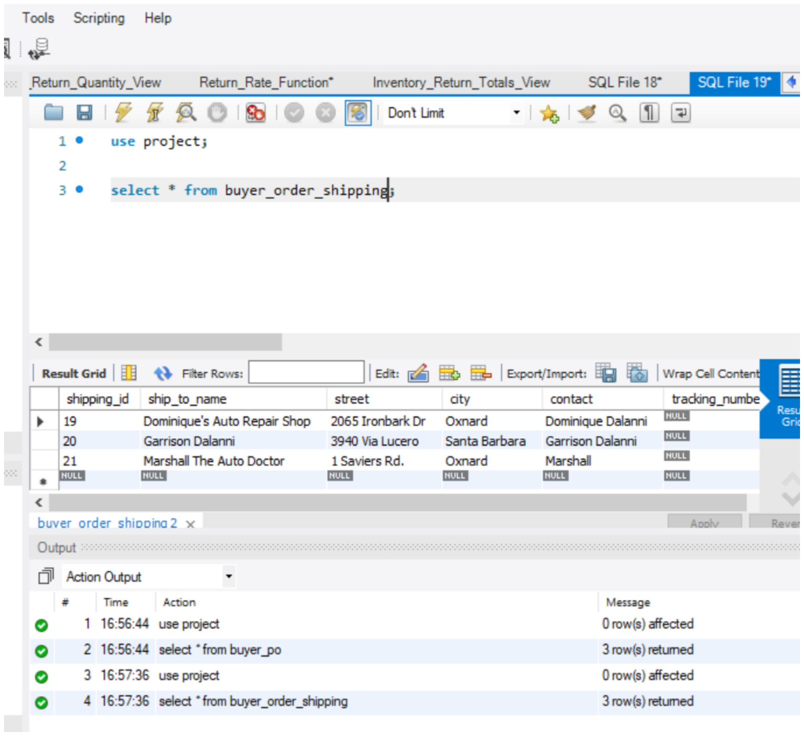
Database Screenshots to Show Successful Update On Backend:



```
use project;

select * from buyer_po;
```

order_id	buyer_id	payment_status	order_status	warranty_start_date	shipping_id
12	1	1	0	NULL	19
13	1	1	0	NULL	20
14	6	0	0	NULL	21
NULL	NULL	NULL	NULL	NULL	NULL

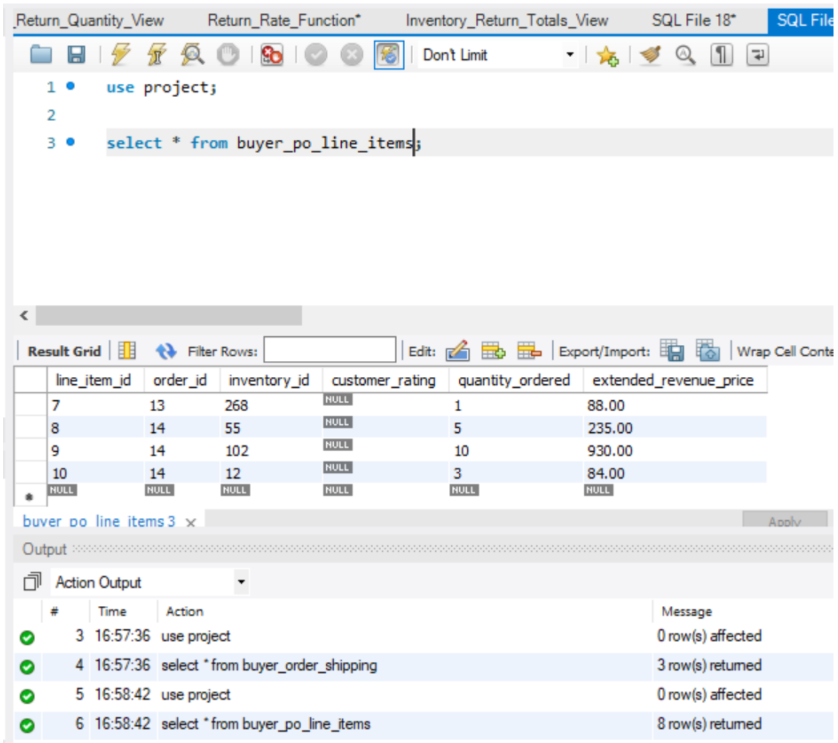
#	Time	Action	Message
1	16:56:44	use project	0 row(s) affected
2	16:56:44	select * from buyer_po	3 row(s) returned



```
use project;

select * from buyer_order_shipping;
```

shipping_id	ship_to_name	street	city	contact	tracking_numbe
19	Dominique's Auto Repair Shop	2065 Ironbark Dr	Oxnard	Dominique Dalanni	NULL
20	Garrison Dalanni	3940 Via Lucero	Santa Barbara	Garrison Dalanni	NULL
21	Marshall The Auto Doctor	1 Saviers Rd.	Oxnard	Marshall	NULL
NULL	NULL	NULL	NULL	NULL	NULL

#	Time	Action	Message
1	16:56:44	use project	0 row(s) affected
2	16:56:44	select * from buyer_po	3 row(s) returned
3	16:57:36	use project	0 row(s) affected
4	16:57:36	select * from buyer_order_shipping	3 row(s) returned



Return_Quantity_View | Return_Rate_Function* | Inventory_Return_Totals_View | SQL File 18* | SQL File

Don't Limit

```
use project;

select * from buyer_po_line_items;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Conte

line_item_id	order_id	inventory_id	customer_rating	quantity_ordered	extended_revenue_price
7	13	268	NULL	1	88.00
8	14	55	NULL	5	235.00
9	14	102	NULL	10	930.00
10	14	12	NULL	3	84.00
NULL	NULL	NULL	NULL	NULL	NULL

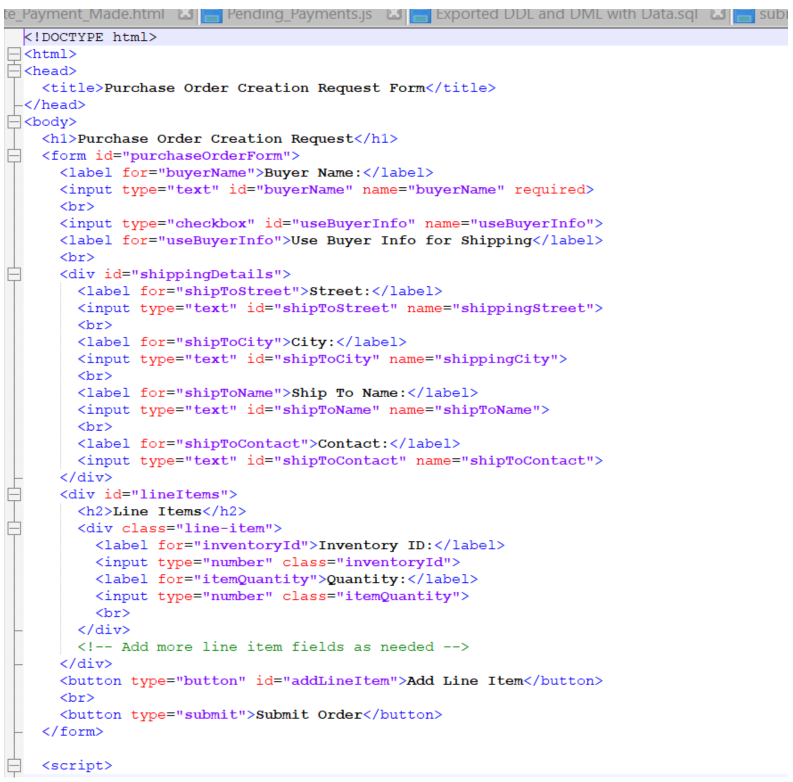
buyer_po_line_items 3 ×

Output

Action Output

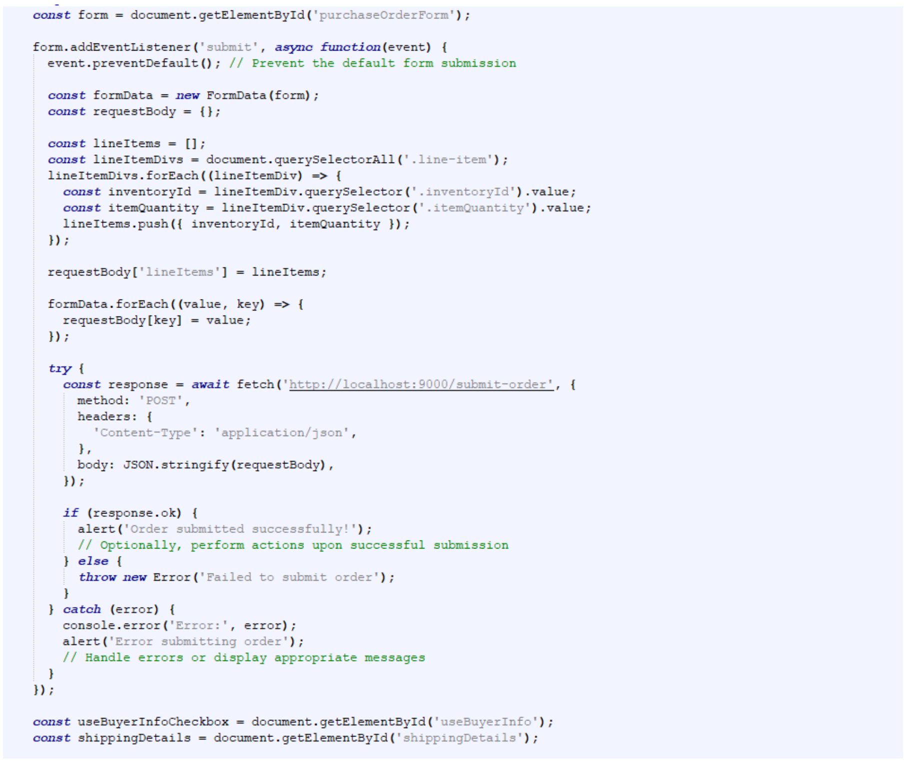
#	Time	Action	Message
3	16:57:36	use project	0 row(s) affected
4	16:57:36	select * from buyer_order_shipping	3 row(s) returned
5	16:58:42	use project	0 row(s) affected
6	16:58:42	select * from buyer_po_line_items	8 row(s) returned

HTML Code:

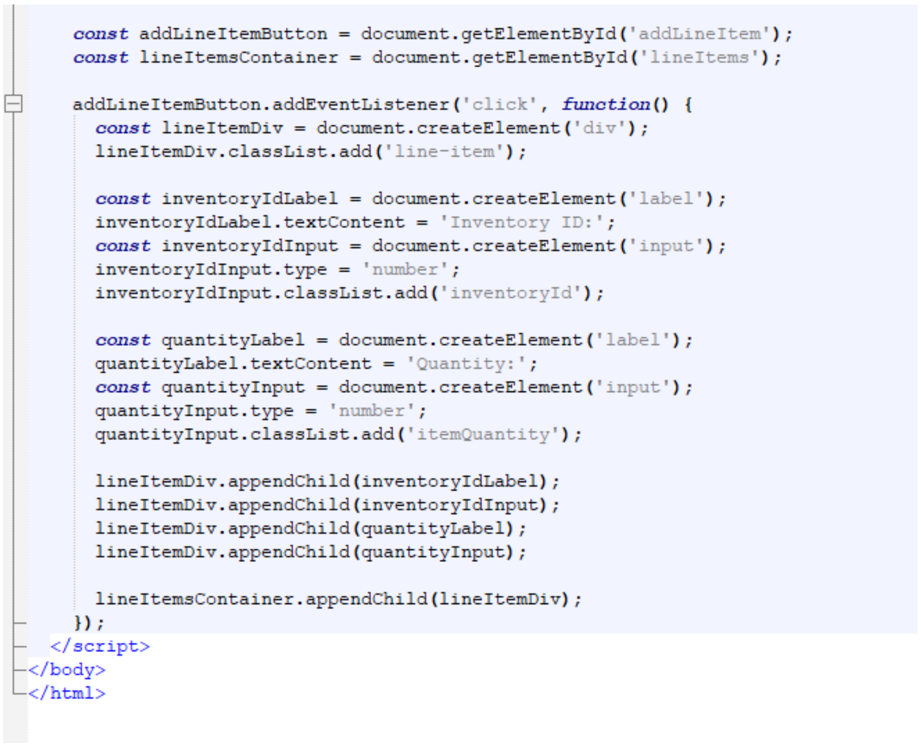


te_Payment_Made.html | Pending_Payments.js | Exported DDL and DML with Data.sql | subi

```
<!DOCTYPE html>
<html>
<head>
  <title>Purchase Order Creation Request Form</title>
</head>
<body>
  <h1>Purchase Order Creation Request</h1>
  <form id="purchaseOrderForm">
    <label for="buyerName">Buyer Name:</label>
    <input type="text" id="buyerName" name="buyerName" required>
    <br>
    <input type="checkbox" id="useBuyerInfo" name="useBuyerInfo">
    <label for="useBuyerInfo">Use Buyer Info for Shipping</label>
    <br>
    <div id="shippingDetails">
      <label for="shipToStreet">Street:</label>
      <input type="text" id="shipToStreet" name="shippingStreet">
      <br>
      <label for="shipToCity">City:</label>
      <input type="text" id="shipToCity" name="shippingCity">
      <br>
      <label for="shipToName">Ship To Name:</label>
      <input type="text" id="shipToName" name="shipToName">
      <br>
      <label for="shipToContact">Contact:</label>
      <input type="text" id="shipToContact" name="shipToContact">
    </div>
    <div id="lineItems">
      <h2>Line Items</h2>
      <div class="line-item">
        <label for="inventoryId">Inventory ID:</label>
        <input type="number" class="inventoryId">
        <label for="itemQuantity">Quantity:</label>
        <input type="number" class="itemQuantity">
        <br>
      </div>
      <!-- Add more line item fields as needed -->
    </div>
    <button type="button" id="addLineItem">Add Line Item</button>
    <br>
    <button type="submit">Submit Order</button>
  </form>

  <script>
```

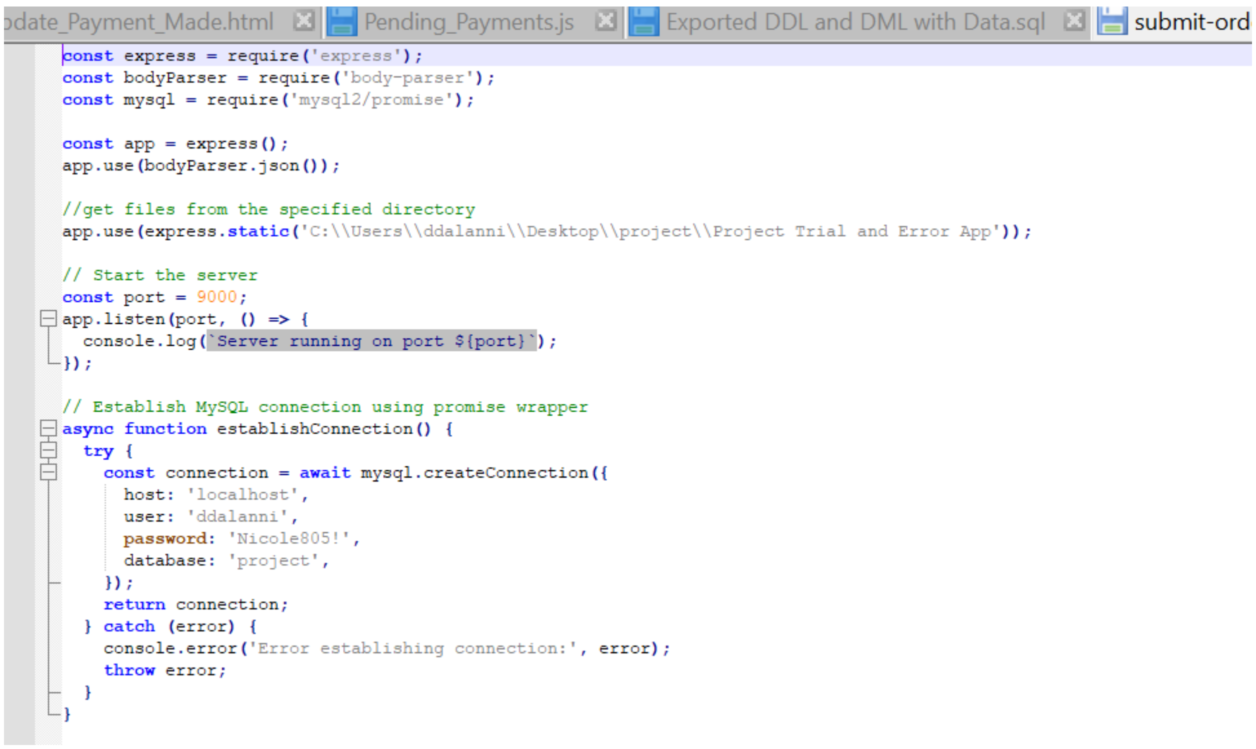


```
const form = document.getElementById('purchaseOrderForm');

form.addEventListener('submit', async function(event) {
  event.preventDefault(); // Prevent the default form submission

  const formData = new FormData(form);
  const requestBody = {};

  const lineItems = [];
  const lineItemDivs = document.querySelectorAll('.line-item');
  lineItemDivs.forEach((lineItemDiv) => {
    const inventoryId = lineItemDiv.querySelector('.inventoryId').value;
    const itemQuantity = lineItemDiv.querySelector('.itemQuantity').value;
    lineItems.push({ inventoryId, itemQuantity });
  });

  requestBody['lineItems'] = lineItems;

  formData.forEach((value, key) => {
    requestBody[key] = value;
  });

  try {
    const response = await fetch('http://localhost:9000/submit-order', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify(requestBody),
    });

    if (response.ok) {
      alert('Order submitted successfully!');
      // Optionally, perform actions upon successful submission
    } else {
      throw new Error('Failed to submit order');
    }
  } catch (error) {
    console.error('Error:', error);
    alert('Error submitting order');
    // Handle errors or display appropriate messages
  }
});

const useBuyerInfoCheckbox = document.getElementById('useBuyerInfo');
const shippingDetails = document.getElementById('shippingDetails');
```

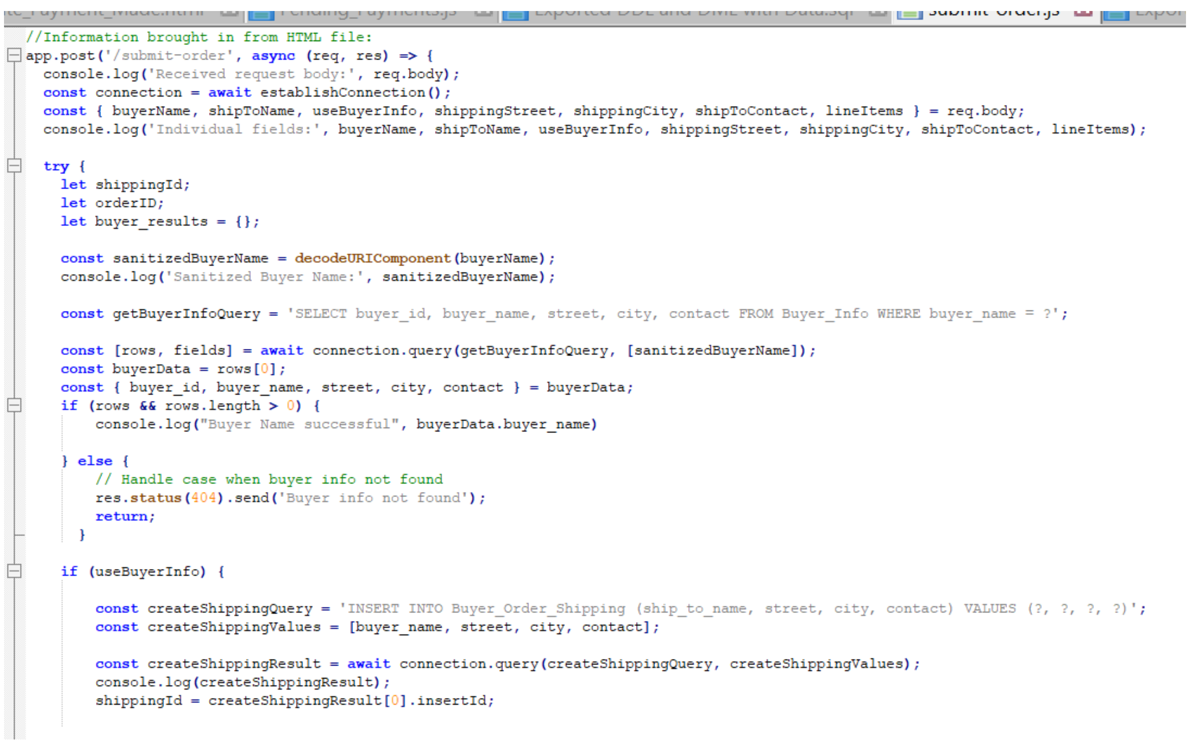


```
    const addLineItemButton = document.getElementById('addLineItem');
    const lineItemsContainer = document.getElementById('lineItems');

    addLineItemButton.addEventListener('click', function() {
      const lineItemDiv = document.createElement('div');
      lineItemDiv.classList.add('line-item');

      const inventoryIdLabel = document.createElement('label');
      inventoryIdLabel.textContent = 'Inventory ID:';
      const inventoryIdInput = document.createElement('input');
      inventoryIdInput.type = 'number';
      inventoryIdInput.classList.add('inventoryId');

      const quantityLabel = document.createElement('label');
      quantityLabel.textContent = 'Quantity:';
      const quantityInput = document.createElement('input');
      quantityInput.type = 'number';
      quantityInput.classList.add('itemQuantity');

      lineItemDiv.appendChild(inventoryIdLabel);
      lineItemDiv.appendChild(inventoryIdInput);
      lineItemDiv.appendChild(quantityLabel);
      lineItemDiv.appendChild(quantityInput);

      lineItemsContainer.appendChild(lineItemDiv);
    });
  </script>
</body>
</html>
```

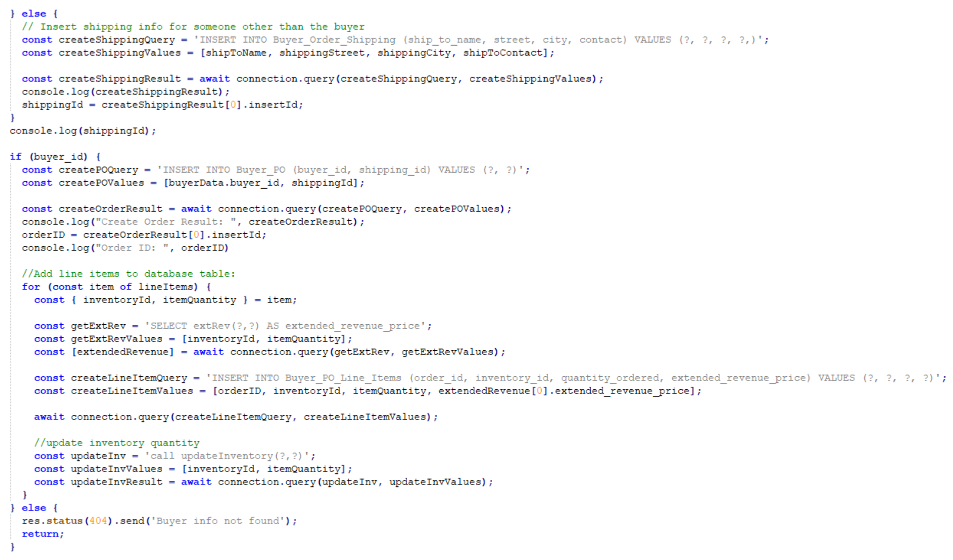
Javascript File:



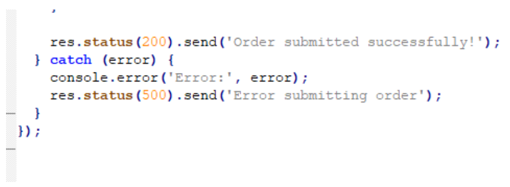
```
const express = require('express');
const bodyParser = require('body-parser');
const mysql = require('mysql2/promise');

const app = express();
app.use(bodyParser.json());

//get files from the specified directory
app.use(express.static('C:\\Users\\ddalanni\\Desktop\\project\\Project Trial and Error App'));

// Start the server
const port = 9000;
app.listen(port, () => {
  console.log(`Server running on port ${port}`);
});

// Establish MySQL connection using promise wrapper
async function establishConnection() {
  try {
    const connection = await mysql.createConnection({
      host: 'localhost',
      user: 'ddalanni',
      password: 'Nicole805!',
      database: 'project',
    });
    return connection;
  } catch (error) {
    console.error('Error establishing connection:', error);
    throw error;
  }
}
```



```
//Information brought in from HTML file:
app.post('/submit-order', async (req, res) => {
  console.log('Received request body:', req.body);
  const connection = await establishConnection();
  const { buyerName, shipToName, useBuyerInfo, shippingStreet, shippingCity, shipToContact, lineItems } = req.body;
  console.log('Individual fields:', buyerName, shipToName, useBuyerInfo, shippingStreet, shippingCity, shipToContact, lineItems);

  try {
    let shippingId;
    let orderID;
    let buyer_results = {};

    const sanitizedBuyerName = decodeURIComponent(buyerName);
    console.log('Sanitized Buyer Name:', sanitizedBuyerName);

    const getBuyerInfoQuery = 'SELECT buyer_id, buyer_name, street, city, contact FROM Buyer_Info WHERE buyer_name = ?';

    const [rows, fields] = await connection.query(getBuyerInfoQuery, [sanitizedBuyerName]);
    const buyerData = rows[0];
    const { buyer_id, buyer_name, street, city, contact } = buyerData;
    if (rows && rows.length > 0) {
        console.log("Buyer Name successful", buyerData.buyer_name)

    } else {
        // Handle case when buyer info not found
        res.status(404).send('Buyer info not found');
        return;
      }

    if (useBuyerInfo) {

        const createShippingQuery = 'INSERT INTO Buyer_Order_Shipping (ship_to_name, street, city, contact) VALUES (?, ?, ?, ?)';
        const createShippingValues = [buyer_name, street, city, contact];

        const createShippingResult = await connection.query(createShippingQuery, createShippingValues);
        console.log(createShippingResult);
        shippingId = createShippingResult[0].insertId;
```



```
} else {
  // Insert shipping info for someone other than the buyer
  const createShippingQuery = 'INSERT INTO Buyer_Order_Shipping (ship_to_name, street, city, contact) VALUES (?, ?, ?, ?,)';
  const createShippingValues = [shipToName, shippingStreet, shippingCity, shipToContact];

  const createShippingResult = await connection.query(createShippingQuery, createShippingValues);
  console.log(createShippingResult);
  shippingId = createShippingResult[0].insertId;
}
console.log(shippingId);

if (buyer_id) {
  const createPOQuery = 'INSERT INTO Buyer_PO (buyer_id, shipping_id) VALUES (?, ?)';
  const createPOValues = [buyerData.buyer_id, shippingId];

  const createOrderResult = await connection.query(createPOQuery, createPOValues);
  console.log("Create Order Result: ", createOrderResult);
  orderID = createOrderResult[0].insertId;
  console.log("Order ID: ", orderID)

  //Add line items to database table:
  for (const item of lineItems) {
    const { inventoryId, itemQuantity } = item;

    const getExtRev = 'SELECT extRev(?,?) AS extended_revenue_price';
    const getExtRevValues = [inventoryId, itemQuantity];
    const [extendedRevenue] = await connection.query(getExtRev, getExtRevValues);

    const createLineItemQuery = 'INSERT INTO Buyer_PO_Line_Items (order_id, inventory_id, quantity_ordered, extended_revenue_price) VALUES (?, ?, ?, ?)';
    const createLineItemValues = [orderID, inventoryId, itemQuantity, extendedRevenue[0].extended_revenue_price];

    await connection.query(createLineItemQuery, createLineItemValues);

    //update inventory quantity
    const updateInv = 'call updateInventory(?,?)';
    const updateInvValues = [inventoryId, itemQuantity];
    const updateInvResult = await connection.query(updateInv, updateInvValues);
  }
} else {
  res.status(404).send('Buyer info not found');
  return;
}
```

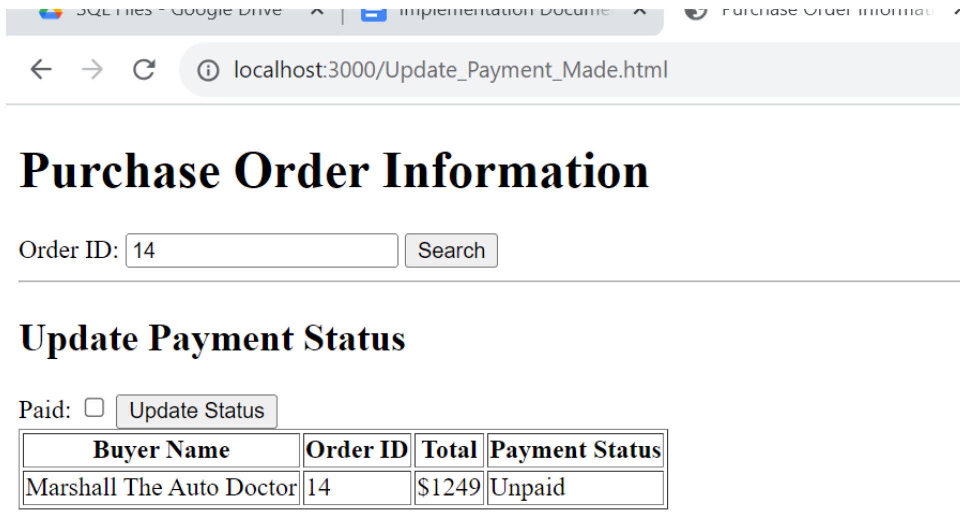


```
    ,

    res.status(200).send('Order submitted successfully!');
  } catch (error) {
    console.error('Error:', error);
    res.status(500).send('Error submitting order');
  }
});
```

3. Update Buyer Purchase Order Payment Status Page:

This page allows for a buyer purchase order search and marks the order as paid. This changes the payment status of the order from 0 to 1 and updates the shipping status to pending:

Webpage:



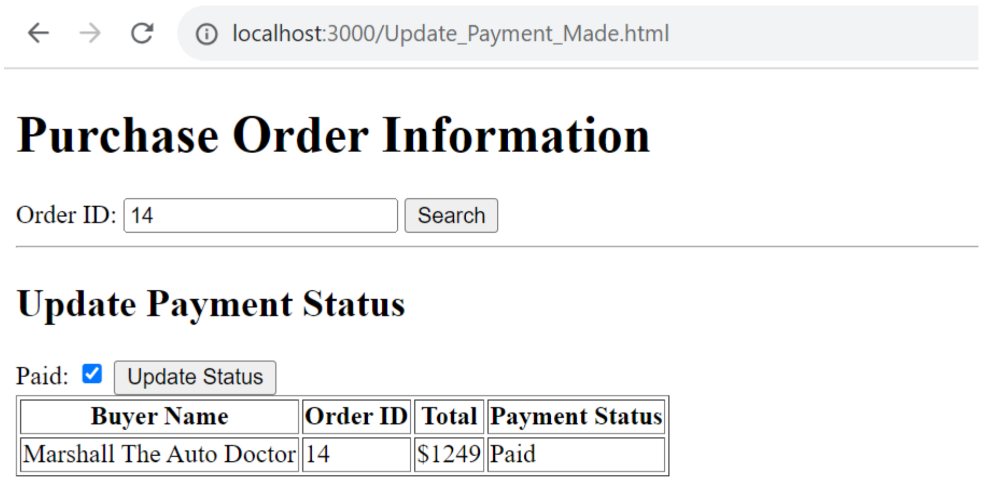
Purchase Order Information

Order ID: 14 Search

Update Payment Status

Paid: ☐ Update Status

Buyer Name	Order ID	Total	Payment Status
Marshall The Auto Doctor	14	$1249	Unpaid


localhost:3000/Update_Payment_Made.html

Purchase Order Information

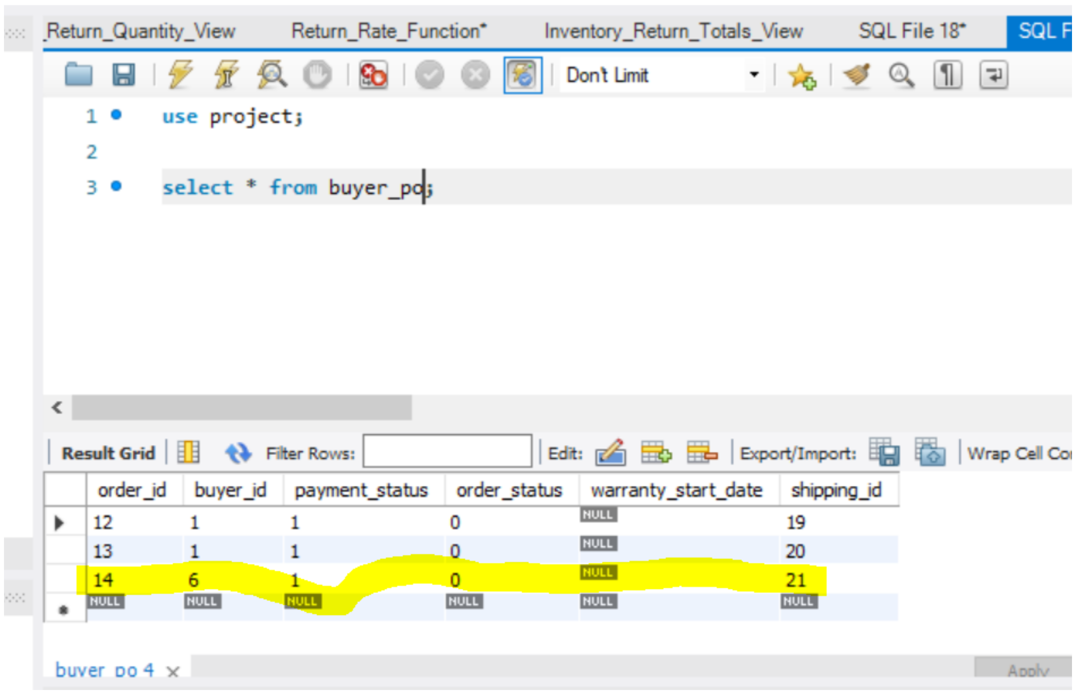
Order ID: 14 Search

Update Payment Status

Paid: ☑ Update Status

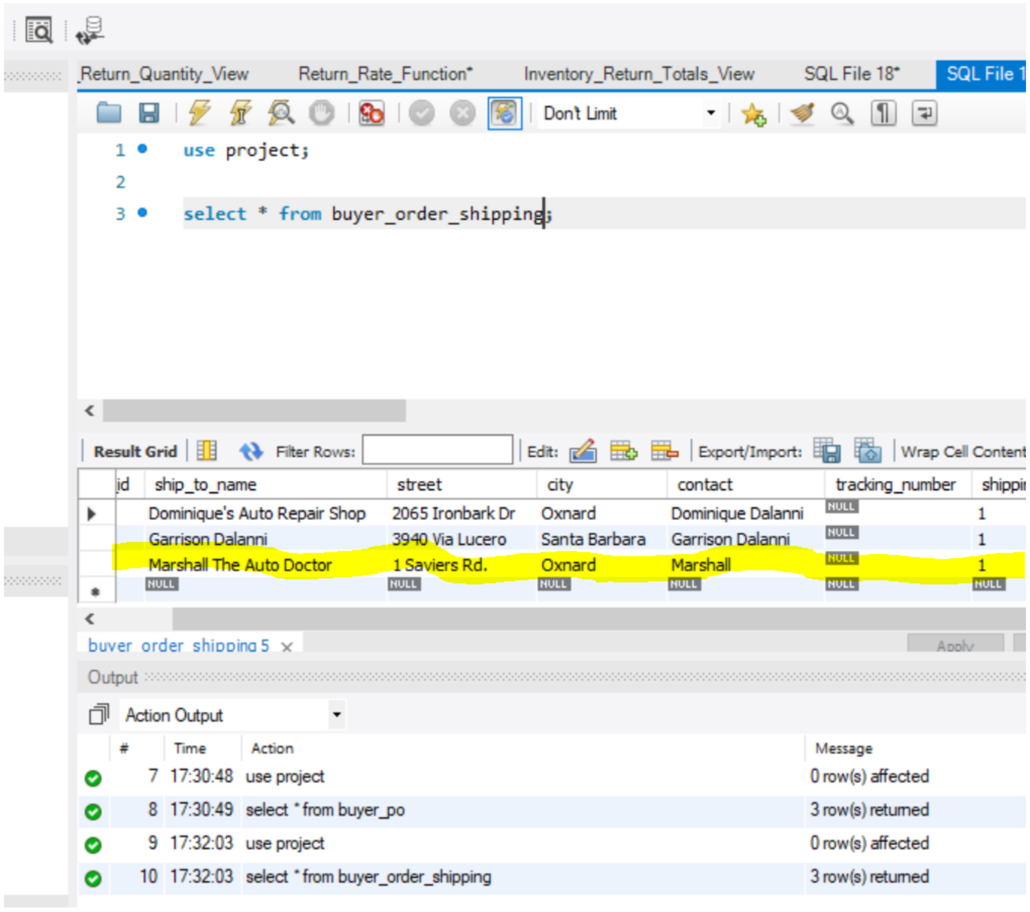
Buyer Name	Order ID	Total	Payment Status
Marshall The Auto Doctor	14	$1249	Paid

Updates in Database Showing Changes:



```
use project;

select * from buyer_po;
```

order_id	buyer_id	payment_status	order_status	warranty_start_date	shipping_id
12	1	1	0	NULL	19
13	1	1	0	NULL	20
14	6	1	0	NULL	21
NULL	NULL	NULL	NULL	NULL	NULL

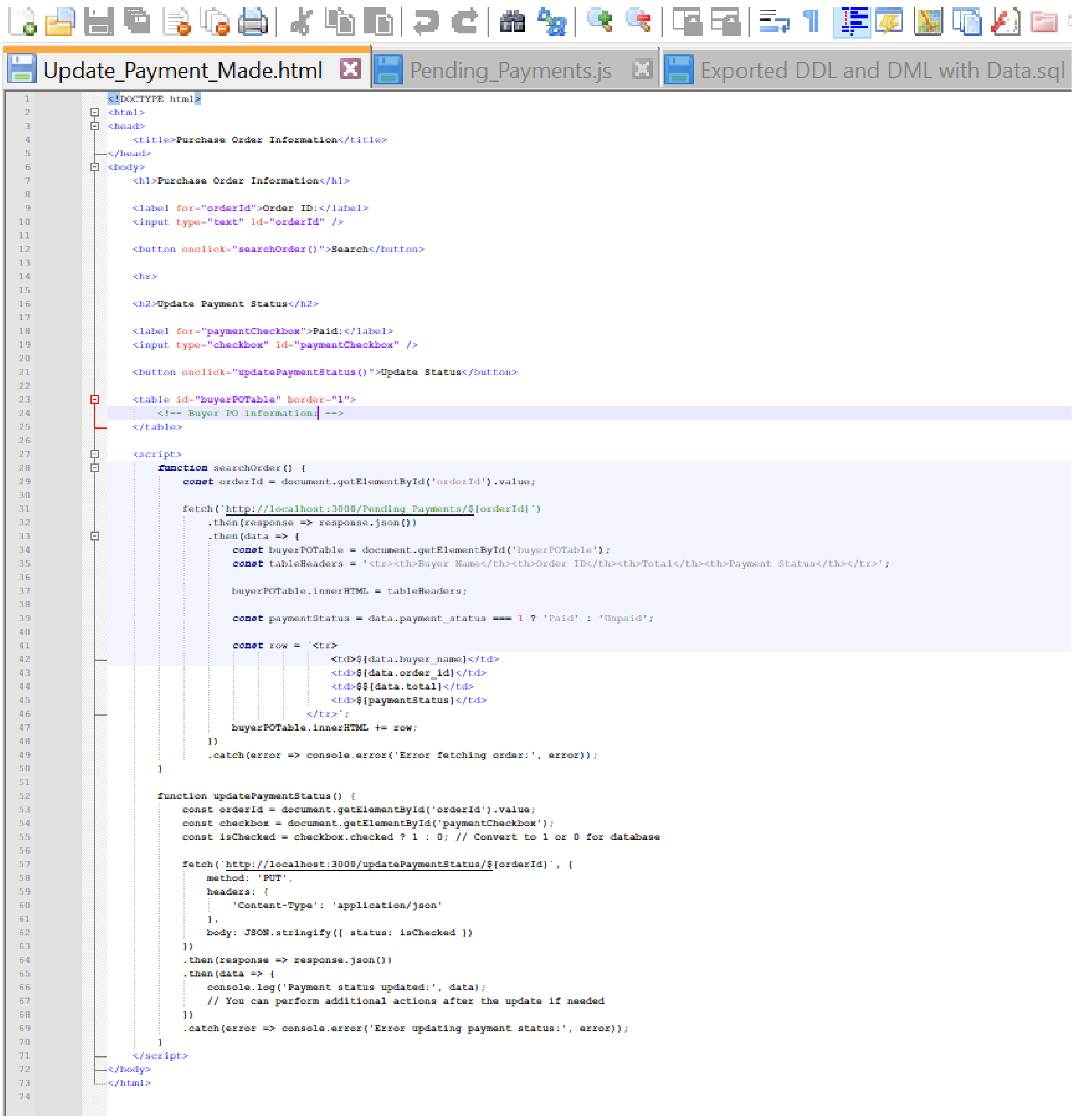


```
use project;

select * from buyer_order_shipping;
```

id	ship_to_name	street	city	contact	tracking_number	shippi
	Dominique's Auto Repair Shop	2065 Ironbark Dr	Oxnard	Dominique Dalanni	NULL	1
	Garrison Dalanni	3940 Via Lucero	Santa Barbara	Garrison Dalanni	NULL	1
	Marshall The Auto Doctor	1 Saviers Rd.	Oxnard	Marshall	NULL	1
	NULL	NULL	NULL	NULL	NULL	NULL

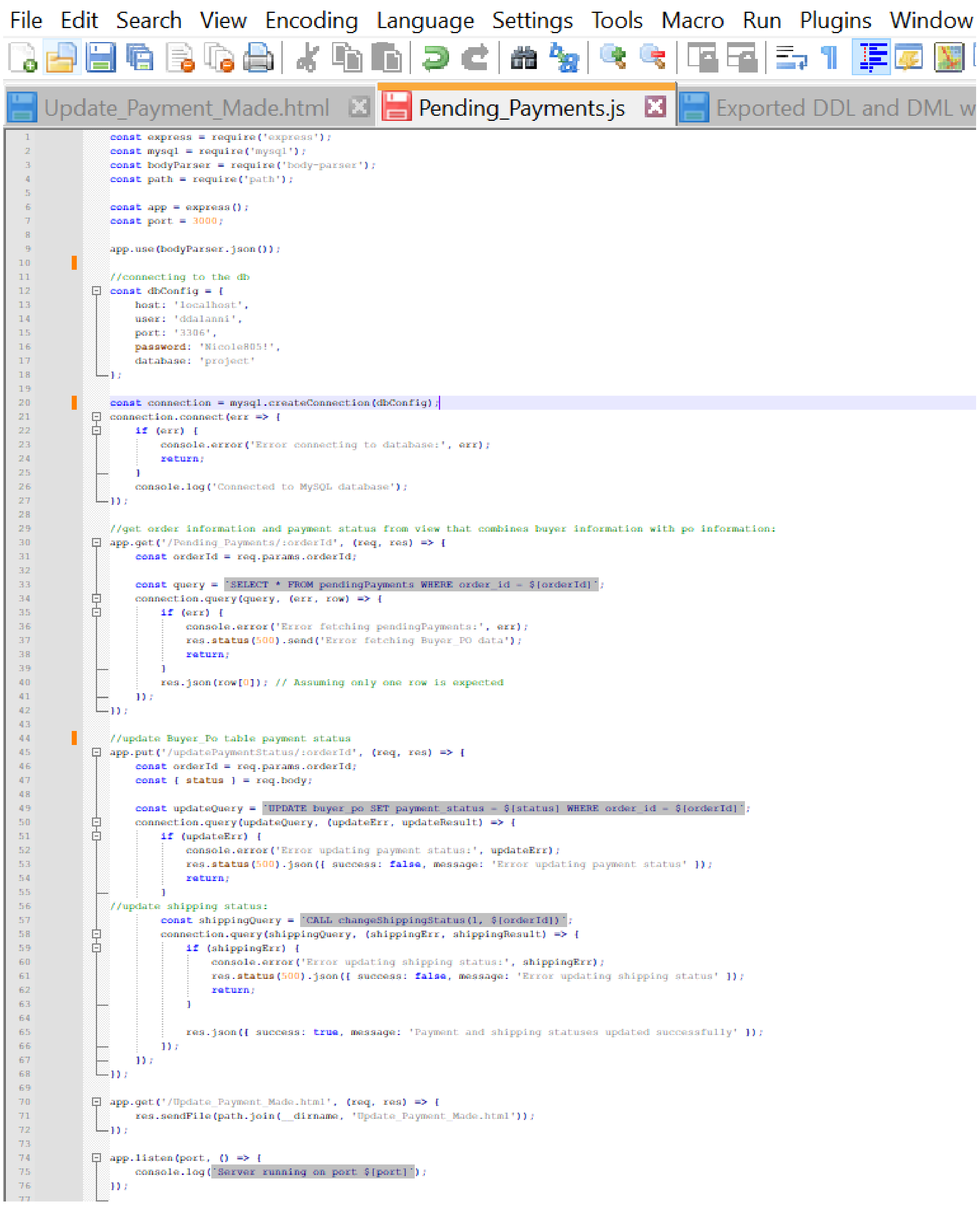
#	Time	Action	Message
7	17:30:48	use project	0 row(s) affected
8	17:30:49	select * from buyer_po	3 row(s) returned
9	17:32:03	use project	0 row(s) affected
10	17:32:03	select * from buyer_order_shipping	3 row(s) returned

HTML Code:



```html
<!DOCTYPE html>
<html>
<head>
    <title>Purchase Order Information</title>
</head>
<body>
    <h1>Purchase Order Information</h1>

    <label for="orderId">Order ID:</label>
    <input type="text" id="orderId" />

    <button onclick="searchOrder()">Search</button>

    <hr>

    <h2>Update Payment Status</h2>

    <label for="paymentCheckbox">Paid:</label>
    <input type="checkbox" id="paymentCheckbox" />

    <button onclick="updatePaymentStatus()">Update Status</button>

    <table id="buyerPOTable" border="1">
        <!-- Buyer PO information: -->
    </table>

    <script>
        function searchOrder() {
            const orderId = document.getElementById('orderId').value;

            fetch(`http://localhost:3000/Pending_Payments/${orderId}`)
                .then(response => response.json())
                .then(data => {
                    const buyerPOTable = document.getElementById('buyerPOTable');
                    const tableHeaders = '<tr><th>Buyer Name</th><th>Order ID</th><th>Total</th><th>Payment Status</th></tr>';

                    buyerPOTable.innerHTML = tableHeaders;

                    const paymentStatus = data.payment_status === 1 ? 'Paid' : 'Unpaid';

                    const row = `<tr>
                                <td>${data.buyer_name}</td>
                                <td>${data.order_id}</td>
                                <td>$${data.total}</td>
                                <td>${paymentStatus}</td>
                            </tr>`;
                    buyerPOTable.innerHTML += row;
                })
                .catch(error => console.error('Error fetching order:', error));
        }

        function updatePaymentStatus() {
            const orderId = document.getElementById('orderId').value;
            const checkbox = document.getElementById('paymentCheckbox');
            const isChecked = checkbox.checked ? 1 : 0; // Convert to 1 or 0 for database

            fetch(`http://localhost:3000/updatePaymentStatus/${orderId}`, {
                method: 'PUT',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify({ status: isChecked })
            })
            .then(response => response.json())
            .then(data => {
                console.log('Payment status updated:', data);
                // You can perform additional actions after the update if needed
            })
            .catch(error => console.error('Error updating payment status:', error));
        }
    </script>
</body>
</html>
```

Javascript Code:


File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window
Update_Payment_Made.html
Pending_Payments.js
Exported DDL and DML w

```javascript
const express = require('express');
const mysql = require('mysql');
const bodyParser = require('body-parser');
const path = require('path');

const app = express();
const port = 3000;

app.use(bodyParser.json());

//connecting to the db
const dbConfig = {
    host: 'localhost',
    user: 'ddalanni',
    port: '3306',
    password: 'Nicole805!',
    database: 'project'
};

const connection = mysql.createConnection(dbConfig);
connection.connect(err => {
    if (err) {
        console.error('Error connecting to database:', err);
        return;
    }
    console.log('Connected to MySQL database');
});

//get order information and payment status from view that combines buyer information with po information:
app.get('/Pending_Payments/:orderId', (req, res) => {
    const orderId = req.params.orderId;

    const query = `SELECT * FROM pendingPayments WHERE order_id = ${orderId}`;
    connection.query(query, (err, row) => {
        if (err) {
            console.error('Error fetching pendingPayments:', err);
            res.status(500).send('Error fetching Buyer_PO data');
            return;
        }
        res.json(row[0]); // Assuming only one row is expected
    });
});

//update Buyer_Po table payment status
app.put('/updatePaymentStatus/:orderId', (req, res) => {
    const orderId = req.params.orderId;
    const { status } = req.body;

    const updateQuery = `UPDATE buyer_po SET payment_status = ${status} WHERE order_id = ${orderId}`;
    connection.query(updateQuery, (updateErr, updateResult) => {
        if (updateErr) {
            console.error('Error updating payment status:', updateErr);
            res.status(500).json({ success: false, message: 'Error updating payment status' });
            return;
        }
//update shipping status:
        const shippingQuery = `CALL changeShippingStatus(1, ${orderId})`;
        connection.query(shippingQuery, (shippingErr, shippingResult) => {
            if (shippingErr) {
                console.error('Error updating shipping status:', shippingErr);
                res.status(500).json({ success: false, message: 'Error updating shipping status' });
                return;
            }

            res.json({ success: true, message: 'Payment and shipping statuses updated successfully' });
        });
    });
});

app.get('/Update_Payment_Made.html', (req, res) => {
    res.sendFile(path.join(__dirname, 'Update_Payment_Made.html'));
});

app.listen(port, () => {
    console.log(`Server running on port ${port}`);
});
```

Closing Comments:

Due to time constraints we were unable to fully implement the GUI interface for our system. However we are proud of the progress we have made, and fully stand behind this product. Version 2 will include the following:

- Implement tracking number input page once items have been shipped which automatically updates shipping status to shipped.
- Implement customer review entry.
- Implement list of buyer order history
- Implement form for customer return
- Implement form for processing receival of customer return
- Implement vendor return

References:

- https://github.com/atduskgreg/hypecyclist/blob/master/lists/List%20of%20auto%20parts.csv (For beginning bits of data)

- https://www.kaggle.com/datasets/humansintheloop/car-parts-and-car-damages (For beginning bits of data)

- https://coursesweb.net/nodejs/select-mysql-output-html-table (Tutorial to view database select statement on HTML Webpage)

- https://chat.openai.com/ (For javascript and html for code debugging)