# BuzzCars

## Table of Contents

Table of Contents

# Abstract Code with SQL

## Login

### Abstract Code

- The User enters the username (@username) and password (@password) in the input form
- When the user clicks the ***Login*** button:

```
SELECT u.password FROM User u WHERE u.username = @username;
```

  - If both the username and the password fields are not empty:
    - If username is not found from the database or if the user's password does not match:
      - Display an error message ("The combination of your username and passwords do not match")and clear the field
    - Else:
      - Navigate to **Display Cars List (Main Page)** with the Search Bar with a logged in session
      - Set @username to be the session username
  - Else:
    - Display a message saying that both username and password are required
- Click the ***back arrow*** button to navigate to the main page

## Display Cars List (Main Page)

### Abstract Code

- The default view is a navigation bar, and a search bar(for keywords) with a drop down button available next to the search bar
- @username= user from logged in
- If logged in (@username not null):
  - Run **SearchbyVIN** task(defined below)
  - For the *navigation bars*
    - If the logged in user is an owner or a manager
      - Show "***Seller History***", "***Average Time***", "***Price Per Condition Report***", "***Parts Statistics Report***", "***Monthly Sales Report***" tabs
      - Click the ***Seller History*** button to navigate to the **View Seller History Report** task
      - Click the ***Average Time*** in Inventory Report button to navigate to the **View Average Time in Inventory Report** task

Table of Contents

            - Click the ***Price Per Condition Report*** button to navigate to the **View Price Per Condition Report** task
            - Click the ***Parts Statistics Report*** button to navigate to the **View Parts Statistics Report** task
            - Click the ***Monthly Sales Report*** button to navigate to the **View Monthly Sales Report** task
        - If the logged in user is an Inventory Clerk or owner:
            - Show "***Add Vehicle***" tabs
                - Click the ***Add Vehicle*** button to navigate to the **Add/Edit Vehicle Info** task

- If the ***dropdown*** button is clicked:
    - criterias={"Vehicle Type":[], "Manufacturer":[], "Model Year":[], "Fuel Type":[], "Color":[]}
        - criterias["Vehicle Type"]=

```
SELECT type FROM VehicleType;
```

        - criterias["Manufacturer"]=

```
SELECT company FROM Manufacturer;
```

        - criterias["Model Year"]= [i for i in range (1950, currentYear+2)]
        - criterias["Fuel Type"]= ['Gas', 'Diesel', 'Natural Gas', 'Hybrid', 'Plugin Hybrid', 'Battery','Fuel 'Cell']
        - criterias["Color"]=

```
SELECT color FROM VehicleColor;
```

    - For each criteria in criterias:
        - Show criterias[criteria] in a dropdown

- When the ***Search*** button is clicked:
    - If none of the criterias are chosen and the input field is empty:
        - Display a message "Please enter some keywords or choose at least one filtering criteria."
    - Else:
        - For *criteria* in criterias:
            - If dropdown value is none:

        - Set the *criteria* to none
    - Else:
        - Set the *criteria* to the value
        - Set the @keyword from the textfield

- cars_result=

```
SELECT
   ot.type,
   modelYear,
   mb.company AS manufacturer,
   modelName,
   fuelType,
   mileage,
   (1.1 * COALESCE((SELECT -- might not be in partorder, default to have part price 0
         SUM(P.cost * P.quantity)
      FROM
         PartOrder PO
            JOIN
         Part P ON P.orderNumber = PO.orderNumber
      WHERE
         PO.vin = v.vin),0) + 1.25 * (SELECT
         s.purchasePrice
      FROM
         Sells_To s
      WHERE
         s.vin = v.vin)) AS price, -- This is a subquery to calculate the price
   GROUP_CONCAT(DISTINCT vc.color) AS colors -- concatenate all different colors in a
row
FROM
   Vehicle v
      JOIN
   Of_Type ot ON v.vin = ot.vin
      JOIN
   Manufactured_By mb ON v.vin = mb.vin
      JOIN
   Of_Color vc ON v.vin = vc.vin

WHERE (v.vin NOT IN ( -- filter out the cars that have not been installed
           SELECT p.vin
           FROM Part p
           WHERE p.status != 'installed'))
         AND
   (ot.type = @vehicleType -- This is to filter based on the filtering criterias that the user chose
```

Table of Contents

```
        OR @vehicleType IS NULL)
        AND (@modelYear <= v.modelYear
        OR @modelYear IS NULL)  -- if it is null, will set to true
        AND (mb.company = @manufacturer
        OR @manufacturer IS NULL)
        AND (v.modelName = @modelName
        OR @modelName IS NULL)
        AND (v.fuelType = @fuelType
        OR @fuelType IS NULL)
        AND (v.mileage <= @mileage
        OR @mileage IS NULL)
        AND (1.1 * (SELECT
            SUM(P.cost * P.quantity)
        FROM
            PartOrder PO
                JOIN
            Part P ON P.orderNumber = PO.orderNumber
        WHERE
            PO.vin = v.vin) + 1.25 * (SELECT
            s.purchasePrice
        FROM
            Sells_To s
        WHERE
            s.vin = v.vin) <= @price
        OR @price IS NULL) -- This is a subquery to compare the sale price and the entered
price.
        AND (ot.type LIKE CONCAT('%', @keyword, '%') -- This is to search through all the
different fields that may include a keyword
        OR modelYear = @keyword
        OR mb.company LIKE CONCAT('%', @keyword, '%')
        OR modelName LIKE CONCAT('%', @keyword, '%')
        OR v.description LIKE CONCAT('%', @keyword, '%'))
GROUP BY v.vin; -- this groupby is for concatenating colors
```

- If no cars have the matching criterias:
    - Display a message: “Sorry, it looks like we don’t have that in stock!”
- Else:
    - Order the cars with ascending VIN
    - For all the returned cars:
- Display the vehicle type, model year, manufacturer, model, fuel type, colors, mileage, and sale price in a ***Card*** component

Table of Contents

    - If a ***Card*** component of a car is clicked, navigate to the detailed page for the car

- **SearchbyVIN**:
    - User with username (@username) enters a full VIN (@vin) and clicks ***Search by VIN*** button
    - Determine if the input VIN is of legit format (alphanumeric)
    - Run the **Search by VIN** task: query for vehicle information where @vin is the vehicle identification number entered by the user
        - Determine if a User is Salesperson only and does not act in other roles such as Owner, Manager or InventoryClerk. If the following selection is NOT NULL, the User is only Salesperson and does not hold any other roles.

```
-- Check if the user is a Salesperson and not in other roles
    SELECT username
    FROM User u
    WHERE u.username = @username
      AND u.username NOT IN (
         SELECT username FROM Owner
         UNION
         SELECT username FROM Manager
         UNION
         SELECT username FROM InventoryClerk
      );
```

        - If User is SalesPeople only:
            - find the vehicle using Vehicle.vin that includes @vin and none of the associated part status is other than installed
                - if the vehicle information is found:
                    - Display Vehicle vin, vehicleType, modelYear, manufacturer, modelName, fuelType, color, mileage, and salesPrice.

```
SELECT
    v.vin,
    ot.type,
    v.modelYear,
    mb.company AS manufacturer,
    v.modelName,
    v.fuelType,
  v.mileage,
 (1.1 * COALESCE((SELECT -- might not be in partorder, default to have part price 0
       SUM(P.cost * P.quantity)
    FROM
       PartOrder PO
          JOIN
       Part P ON P.orderNumber = PO.orderNumber
    WHERE
       PO.vin = v.vin),0) + 1.25 * (SELECT
       s.purchasePrice
    FROM
       Sells_To s
    WHERE
       s.vin = v.vin)) AS price, -- This is a subquery to calculate the price
  GROUP_CONCAT(DISTINCT vc.color) AS colors -- concatenate all different colors in a row
FROM
  Vehicle v
    JOIN
  Of_Type ot ON v.vin = ot.vin
    JOIN
  Manufactured_By mb ON v.vin = mb.vin
    JOIN
  Of_Color vc ON v.vin = vc.vin

 WHERE
  (v.vin = @vin
  AND v.vin NOT IN (
         SELECT p.vin
         FROM Part p
         WHERE p.status != 'installed'
))
GROUP BY v.vin; -- this groupby is for concatenating colors
```

```
                ■ Display link to Sales Order Form next to each
                  vehicle
            ○ else
                ■ Display message "Sorry, it looks like we don't have
                  that in stock!"
    ■ Else (User is InventoryClerks, Managers, or Owner):
        ● find the vehicle using Vehicle.vin that matches @vin
            ○ if the vehicle information is found:
                ■ Display Vehicle vin, vehicleType, modelYear,
                  manufacturer, modelName, fuelType, color,
                  mileage, and salesPrice
```

Table of Contents

```
SELECT
   v.vin,
   ot.type,
   v.modelYear,
   mb.company AS manufacturer,
   v.modelName,
   v.fuelType,
   v.mileage,
  (1.1 * COALESCE((SELECT -- might not be in partOrder, default to have part price 0
         SUM(P.cost * P.quantity)
      FROM
         PartOrder PO
            JOIN
         Part P ON P.orderNumber = PO.orderNumber
      WHERE
         PO.vin = v.vin),0) + 1.25 * (SELECT
         s.purchasePrice
      FROM
         Sells_To s
      WHERE
         s.vin = v.vin)) AS price, -- This is a subquery to calculate the price
   GROUP_CONCAT(DISTINCT vc.color) AS colors -- concatenate all different colors in a row
FROM
   Vehicle v
      JOIN
   Of_Type ot ON v.vin = ot.vin
      JOIN
   Manufactured_By mb ON v.vin = mb.vin
      JOIN
   Of_Color vc ON v.vin = vc.vin

 WHERE
   v.vin = @vin
GROUP BY v.vin; -- this groupby is for concatenating colors
```

- else:
  - Display message “Sorry, it looks like we don’t have that in stock!”

## View Car Details

### Abstract Code

- ***Selected_car*** =

```
SELECT v.vin,ot.type, v.modelYear, mb.company as manufacturer, v.modelName, fuelType,
mileage, GROUP_CONCAT(DISTINCT vc.color) AS colors ,v.description,
 (1.1 *COALESCE((SELECT -- this is for calculating sale price
        SUM(P.cost * P.quantity)
     FROM
        PartOrder PO
          JOIN
        Part P ON P.orderNumber = PO.orderNumber
     WHERE
        PO.vin = v.vin),0) + 1.25 * (SELECT
        s.purchasePrice
     FROM
        Sells_To s
     WHERE
        s.vin = v.vin))
 AS price
FROM Vehicle v
        JOIN Of_Type ot ON v.vin=ot.vin
        JOIN Manufactured_By mb ON v.vin=mb.vin
  JOIN Of_Color vc ON v.vin = vc.vin
WHERE
v.vin = @vin AND -- select the designated car
v.vin NOT IN ( -- filter out not installed ones
         SELECT p.vin
         FROM Part p
         WHERE p.status != 'installed')
GROUP BY v.vin
ORDER BY v.vin ASC;
```

- Show all the attributes of the ***selected_car*** in an unordered list <ul>
- Click the back button on the top left corner to go back to the list of cars from the searching result in **Display Car List/Main Menu**
- If the logged in user is a Manager or an Owner:

```
SELECT EXISTS(SELECT 1 FROM Manager m WHERE m.username = @username)
OR EXISTS(SELECT  1 FROM Owner o WHERE o.username = @username);
```

    - Display ***"View Purchase History"*** buttonClick the ***View Purchase History*** button to go to **View Purchase History** task
- If the logged in user is an Owner or Inventory Clerk

```
SELECT EXISTS(SELECT 1 FROM InventoryClerk ic WHERE ic.username = @username)
OR EXISTS(SELECT  1 FROM Owner o WHERE o.username = @username);
```

    - Display "***Update Part Order Status***", "***Add Part Order***" buttons
    - Click ***Update Part Order Status*** to display the **Part Order Status Form**
    - Click ***Add Part Order*** to display the **Add Parts Order Form**
- If the logged in user is an owner or Salespeople:
    - Display "***Sale Order***" buttons
    - Click the ***Sales Order*** button to navigate to the **Sale Order** form

## Search Part Vendor (Add Parts Order Form)

### Abstract Code

- The inventory clerk initiates the **Search Part Vendor** query while ordering parts within the **Add Parts Order Form**.
- The system provides a text-based search field that allows the clerk to enter keywords or phrases related to vendors.
- As the clerk types in the search field, the system dynamically updates the list of vendors displayed on the screen to match the search criteria.
- The list of vendors should include all information (***vendorID***, ***name***, ***address***, ***phoneNumber***) to help the clerk identify the correct vendor.
- The clerk can click on a vendor from the updated list to select it.
- When the clerk selects a vendor from the search results, the system associates this vendor with the current parts order.

```
SET @searchstring = 'Searchstring';

SELECT name, phoneNumber, street, city, state, postalCode
FROM Vendor
WHERE
   name LIKE CONCAT('%', @searchstring, '%')
   OR phoneNumber LIKE CONCAT('%', @searchstring, '%')
   OR street LIKE CONCAT('%', @searchstring, '%')
   OR city LIKE CONCAT('%', @searchstring, '%')
   OR state LIKE CONCAT('%', @searchstring, '%')
   OR postalCode LIKE CONCAT('%', @searchstring, '%');
```

## Add Part Vendor (Add Parts Order Form)

### Abstract Code

- Inventory clerk enters vendor information into the form fields.
- Click ***Add*** Button: The clerk initiates the process by clicking the ***Add*** button.
- The system verifies whether a vendor with the same name or identifier already exists in the Vendor Table.
- If Vendor Exists:
    - If a matching vendor is found, the system displays an error message indicating that the vendor already exists and cannot be added.
- Else Vendor Doesn't Exist:
    - The system creates a new vendor record in the Vendor Table using the entered information.

```
SET @name = 'name';
SET @phoneNumber = '111-111-1111';
SET @street = 'street';
SET @city = 'city';
SET @state = 'state';
SET @postalCode = '12345';

INSERT INTO Vendor (name, phoneNumber, street, city, state, postalCode)
SELECT
   @name,
   @phoneNumber,
   @street,
   @city,
   @state,
   @postalCode
FROM dual
WHERE NOT EXISTS (
   SELECT 1
   FROM Vendor
   WHERE name = @name
);

SELECT
   CASE
      WHEN ROW_COUNT() > 0 THEN 'Vendor added successfully'
      ELSE 'Vendor already exists and cannot be added'
   END AS Message;
```

## Search, View Part Order (Part Order Status form)

### Abstract Code

- The clerk can enter search criteria: ***orderNumber***, ***orderDate***, ***vendorID***, ***totalCost*** or ***status***, into the search fields provided on the screen.
- The system retrieves a list of part orders that match the search criteria from the database.
- The system displays the search results on the screen, including relevant information about each part order: ***orderNumber***, ***orderDate***, ***vendorID***, ***totalCost*** or ***status***.
- The clerk can click on a specific part order from the search results to view more details.
    - Click the **Update Part Order Status** button to move to the **Update Part Order Status** task
- The system displays the detailed information about the selected part order, ***orderNumber***, ***orderDate***, ***vendorID***, ***totalCost***, ***description***, and ***status***.

```
SELECT
      po.orderNumber, po.vendorName, po.vin, p.partNumber, p.quantity, p.cost, p.status
      FROM PartOrder po
      JOIN Part p ON po.orderNumber = p.orderNumber
      WHERE
 (@searchOrderNumber IS NULL OR po.orderNumber = @searchOrderNumber)
AND (@searchVendorName IS NULL OR po.vendorName = @searchVendorName)
AND (@searchVin IS NULL OR po.vin = @searchVin)
AND (@searchPartNumber IS NULL OR p.partNumber = @searchPartNumber)
 AND (@searchQuantity IS NULL OR p.quantity = @searchQuantity)
AND (@searchCost IS NULL OR p.cost = @searchCost)
AND (@searchStatus IS NULL OR p.status = @searchStatus);
```

Table of Contents

## Update Part Order Status (Part Order Status form)

### Abstract Code

- Choose a new status: The clerk chooses the new status for the selected part. This status can typically be "ordered," "received," or "installed." (@status)
- The clerk submits the updated ***status***.
- Update the Part: The system updates the ***status*** of the selected part in the PartOrder Table with the new ***status*** chosen by the clerk.
- The system provides a confirmation message indicating that the part ***status*** has been successfully updated.
- The clerk select the vehicle(@vin), select the Part Order(@orderNumber), and the part(@partNumber) that he/she wants to modify
- **currentStatus**=

```
SELECT p.status from Part p WHERE  p.orderNumber = @orderNumber AND
p.partNumber=@partNumber and p.vin=@vin;
```

- Update=True
- If the **currentStatus** = “ordered” and @status = “ordered” :
  - Display message: “It is already ordered”.
  - Update=False
- If the **currentStatus** is “received” and (@status = “ordered” OR @status= “received) :
  - Display message: “Status cannot be reverted. ”
  - Update=False
- If the **currentStatus** is installed:
  - Display message: “Status cannot be changed
  - Update=False
- If Update=True:

```
UPDATE Part p
SET
   p.status = @status
WHERE
   p.orderNumber =@orderNumber
      AND p.partNumber = @partNumber
      AND p.vin =@vin;
```

Table of Contents

## Order Parts (Add Parts Order Form)

### Abstract Code

- Select or Add Parts:
  Inventory clerk is presented with options to either select existing parts or add new parts to the part order.
- Check for Part Existence:
  Before adding a new part to the Part table, the system checks if a part with the same part number and same orderNumber already exists.

```
SELECT EXISTS (SELECT 1 FROM Part WHERE @orderNumber=orderNumber
AND @description=description AND @quantity = quantity);
```

  If True (1):
  a message is displayed indicating that the part in that part order already exists and cannot be added again.
- @username=username from the login session
- If selecting existing parts:
  Inventory clerk chooses parts from a list of available parts. This is populated in a drop down menu via loop from each item in the part table.
  Specifies the quantity in the text field
  - If the quantity is entered as data type other than integer or is a negative number:
    -Display a message: “The format of quantity needs to be a positive integer.”

  and clicks the ***Order*** button.
  The status of selected parts is automatically set to 'ordered.'
- Else adding a new part:
  - Present a text field for the Inventory Clerk to enter the partNumber (show hint that it should be a combination of 3 letters and 3 numbers), quantity, description, and the part cost:
    - If the partNumber is not in the right format, OR quantity and part cost are not entered as numeric or entered as a negative number:
      - Display a message: “Please make sure the input is in the correct format”
  - Vendor table fields ***@name***, ***@phoneNumber***, ***@street***, ***@city***, ***@state***, **and** ***@postalcode***,
    - Validate user input data:
    - Validate phone number format(xxx-xxx-xxxx)
    - Validate Postal Code(xxxxx)

Table of Contents

-If data validation fails, display an error message to the user indicating the issue.

VendorExist=

```
SELECT EXISTS (SELECT 1 FROM Vendor  WHERE @name=name AND
@phoneNumber=phoneNumber AND @street = street AND city=@city AND @state=state
and @postalCode=@postalCode);
```

- If the user needs to add a vendor that is already existing:
  Use the **Search Part Vendor** to populate the vendor fields.
  Set @vendorName
- Else vendor is not existing:
  Use **Add Part Vendor** through the **Add Parts Order Form**.

  User clicks the ***Add Part*** button to send the part to the order.
- Generate Purchase Order Number:
  The system generates an automatic unique purchase order number for the part order based on the ***VIN*** of the associated vehicle and an ordinal number for the order (e.g., 123-01 for the first order for vehicle VIN 123).
  PurchaseOrderNumber=

```
SELECT CONCAT(SUBSTRING(@vin, 4), '-', LPAD((SELECT COUNT(*) + 1 FROM
PartOrder WHERE vin = @vin), 2, '0')) AS PurchaseOrderNumber ;
```

- Link PartOrder to Vehicle:
  When the user confirms the part order, a new record is created in the PartOrder table. The PartOrder table includes a reference (e.g., foreign key) to the selected vehicle.
- Update Part Inventory:
  This action is performed automatically when a new part is added to the Part table. If a new part is added to the Part table, the system updates the inventory to reflect the addition of the new part.

```
INSERT INTO PartOrder (vin, orderNumber, username, vendorName)
  VALUES (@vin, @PurchaseOrderNumber, @username, @vendorName);
 INSERT INTO Part (partNumber, vin, orderNumber, quantity, status, description, cost)
  VALUES (
     @PartNumber,
     @vin,
     @PurchaseOrderNumber,
     @quantity,
     'ordered',
     @description,
     @cost );
```

Table of Contents

If successfully added:
    Display a message: “Part order placed successfully.”

## View Seller History (Report Page)

### Abstract Code

- User clicks the ***View Seller History*** Button.
- If role validation is successful for manager or owner, then:
    - Initialize report data by creating an empty list to store data for the report.
    - For each ***vehicle*** in the Vehicle table:
        - Retrieve seller information (***CustomerID***, ***Address***, ***TaxID*** or ***driverLicense***).
        - Create variables to store calculations of the number of parts ordered and the total parts cost for the vehicle.
        - For each seller, calculate:
            - ***totalNumberOfVehiclesSold***.
            - ***averageSoldPrice*** for vehicles sold. Use the formula to calculate each individual vehicle’s price first: sells_to.purchasePrice * [1.25 + (parts.cost * 1.1) * quantity]
            - ***averageNumberOfPartsOrderedPerVehicle***.
            - ***averageCostOfPartsPerVehicle***.
            - For sellers meeting the criteria (average part cost >= @500 or average parts ordered >= 5):
            - Mark the seller's entry for highlighting (This is marked by a red_highlighted column with values true or false).
    - Sort by ***totalNumberOfVehiclesSold*** (descending) and Sort by ***averageSoldPrice*** (ascending) and group sellers together.
    - Present the report data, including seller names, metrics, and highlights. Abstract code.
    - The name of the seller (either first name and last name or company name, which should be displayed as a single column, not two different columns for each seller type)
- Else role validation fails then displays no user rights message.

```
SELECT
  CASE
    WHEN B.businessName IS NOT NULL THEN B.businessName
    ELSE CONCAT(I.firstName, ' ', I.lastName)
  END AS sellerName,
-- Counting the total number of vehicles sold by each seller
   COUNT(DISTINCT ST.vin) AS totalNumberOfVehiclesSold,

  -- Calculating the average sold price per vehicle
   ROUND(AVG(1.1 * COALESCE(PT.partsCost, 0) + 1.25 * s.purchasePrice), 2) AS
averageSoldPrice,

-- Computing the average number of parts ordered per vehicle
   ROUND(SUM(COALESCE(PO.partsOrdered, 0)) / NULLIF(COUNT(DISTINCT ST.vin),
0), 2) AS averageNumberOfPartsOrderedPerVehicle,

  -- Determining the average cost of parts per vehicle
   CASE
      WHEN COALESCE(SUM(PT.partsCost), 0) / NULLIF(COUNT(DISTINCT ST.vin), 0) =
0 THEN 'N/A'
      ELSE ROUND(COALESCE(SUM(PT.partsCost), 0) / NULLIF(COUNT(DISTINCT
ST.vin), 0), 2)
   END AS averageCostOfPartsPerVehicle,

  -- Flagging for red highlighting if average parts ordered or cost per vehicle exceeds certain
thresholds
   (SUM(COALESCE(PO.partsOrdered, 0)) / NULLIF(COUNT(DISTINCT ST.vin), 0) >= 5
   OR COALESCE(SUM(PT.partsCost), 0) / NULLIF(COUNT(DISTINCT ST.vin), 0) >= 500)
AS red_highlighted
FROM
   Buys_From BF
LEFT JOIN Sells_To ST ON ST.vin = BF.vin
LEFT JOIN Business B ON BF.customerID = B.customerID
LEFT JOIN Individual I ON BF.customerID = I.customerID

-- Calculating the total parts ordered for each vehicle
LEFT JOIN (
   SELECT
      PO.vin,
      SUM(P.quantity) AS partsOrdered
   FROM PartOrder PO
   JOIN Part P ON PO.vin = P.vin AND PO.orderNumber = P.orderNumber
   GROUP BY PO.vin
) AS PO ON BF.vin = PO.vin
```

Table of Contents

```
-- Calculating the total cost of parts for each vehicle
LEFT JOIN (
   SELECT
      PO.vin,
      SUM(P.quantity * P.cost * 1.10) AS partsCost
   FROM PartOrder PO
   JOIN Part P ON PO.vin = P.vin AND PO.orderNumber = P.orderNumber
   GROUP BY PO.vin
) AS PT ON BF.vin = PT.vin
LEFT JOIN Sells_To s ON s.vin = BF.vin
GROUP BY sellerName -- Grouping the results by seller's name

-- Sorting the output by the total number of vehicles sold and the average sold price
ORDER BY totalNumberOfVehiclesSold DESC, averageSoldPrice ASC;
```

## View Average Time in Inventory Report (Report Page)

### Abstract Code

- User clicks ***Average Time in Inventory*** button.
- If role validation is successful for manager or owner, then:
- Create ***averageTimeList*** variable.
- Create ***totalNumberOfVehicles variable***.
- For each ***vehicle*** in the dataset:
    - Calculate the time spent in inventory by subtracting the ***purchaseDate*** from the ***saleDate***.
    - Accumulate the calculated times for all vehicles.
    - Store each individual vehicle's average time in ***averageTimeList***
- Calculate the overall average time spent in inventory by dividing the accumulated time by the ***totalNumberOfVehicles*** .
- Display the **Average Time in Inventory** report to the user, including the calculated average time.
- Else role validation fails then the display message user doesn't have the necessary rights.
- Report=

```
SELECT VehicleType.type,
   IFNULL(CAST(AVG(DATEDIFF(Buys_From.transactionDate, Sells_To.purchaseDate))
AS CHAR), 'N/A') AS averageTime -- calculate the date difference between buying from
salesperson and selling to inventory clerk
FROM VehicleType
```

Table of Contents

```
LEFT JOIN Of_Type ON VehicleType.type = Of_Type.type
LEFT JOIN Vehicle ON Vehicle.vin = Of_Type.vin
LEFT JOIN Sells_To ON Sells_To.vin = Of_Type.vin
LEFT JOIN Buys_From ON Of_Type.vin = Buys_From.vin
GROUP BY VehicleType.type;
```

## View Monthly/Yearly Summary Report (Report Page)

### Abstract Code

- User clicks the '***View Monthly/Yearly summary report***' button.
- If role validation is successful for manager or owner, then:
  - Create variables ***SalesSummary*** list to accumulate sales data by year and month, ***totalNumberOfVehiclesSold***, ***totalSalesIncome***, and ***totalNetIncome***. Calculate sales income.
  - For each vehicle in the dataset:
    - Extract the ***saleDate***, ***salesPrice***, ***purchasePrice***, and ***partsCosts***.
    - If the selected year or month has no sales data
      - Exclude the selected date from the report.
    - Determine the year and month of the sale date.
    - Update the relevant variables to ***SalesSummary***.
  - Sort the ***SalesSummary*** by year and month in descending order, with the most recent year and month as the first result.
  - Loop through the sorted ***SalesSummary*** and display the summary report to the user, listing for each year and month:
    - ***Total Number of Vehicles sold***.
    - ***Total Sales Income***.
    - ***Total Net Income***.
- isUserAuthenticated=

```
SELECT EXISTS(SELECT 1 FROM Manager m WHERE m.username = @username)
OR EXISTS(SELECT  1 FROM Owner o WHERE o.username = @username);
```

- If isUserAuthenticated==False:
  - Display message: "Sorry you don't have the permission to view this report".
- Else:

```
SELECT
```

Table of Contents

```
    YEAR(BF.transactionDate) AS SaleYear,
    MONTH(BF.transactionDate) AS SaleMonth,
    COUNT(DISTINCT BF.vin) AS TotalVehiclesSold,
    ROUND(SUM(
        1.1 * COALESCE(PartsCost, 0) + 1.25 * COALESCE(ST.purchasePrice, 0)
    ), 2) AS TotalSalesIncome,
    ROUND(SUM(
        1.1 * COALESCE(PartsCost, 0) + 1.25 * COALESCE(ST.purchasePrice, 0) -
        COALESCE(ST.purchasePrice, 0) -
        COALESCE(PartsCost, 0)
    ), 2) AS TotalNetIncome -- sale price minus the cost of purchase and cost of part
FROM Sells_To ST
JOIN Buys_From BF ON ST.vin = BF.vin
LEFT JOIN (
    SELECT
        PO.vin,
        COALESCE(SUM(P.cost * P.quantity), 0) AS PartsCost -- default 0 instead of Null
    FROM PartOrder PO
    JOIN Part P ON P.orderNumber = PO.orderNumber
    GROUP BY PO.vin
) PartsPrice ON ST.vin = PartsPrice.vin
GROUP BY SaleYear, SaleMonth
ORDER BY SaleYear DESC, SaleMonth DESC;
```

## View Monthly/Yearly Drilldown Report (Report Page)

### Abstract Code

- User selects the month and year for the drill down report.
- Create variables ***salespeople***, ***numberOfVehicles***, ***totalVehiclesSold***, and ***totalSales***.
- User clicks the '***View Monthly/Yearly Drilldown report***' button.
- If role validation is successful for manager or owner, then:
    - Extract the selected year and month from the user's request.
    - For each vehicle in the selected year and month:
        - Calculate the price of vehicles and sort them in descending order for the ***salespeople.***
        - Identify the top-performing ***salespeople*** for the selected year and month.
        - Calculate the ***numberOfVehicles*** they sold and their ***totalSales***.
    - Sort the salespeople data by ***totalVehiclesSold*** (descending) and ***totalSales*** (descending) using a sorting algorithm

Table of Contents

- In the event of a tie in total ***vehiclesSold***, prioritize the salesperson element with the highest total sales.

```
SELECT
   U.firstName AS FirstName,
   U.lastName AS LastName,
   COUNT(DISTINCT BF.vin) AS TotalVehiclesSold,
   ROUND(SUM(
      COALESCE(1.1 * COALESCE(PartsCost, 0) + 1.25 * COALESCE(ST.purchasePrice,
0), 0)
   ), 2) AS TotalSales -- This is to calculate the sales price and sum up
FROM Salesperson sp
JOIN User U ON sp.username = U.username
LEFT JOIN Buys_From BF ON BF.username = sp.username
LEFT JOIN Sells_To ST ON ST.vin = BF.vin
LEFT JOIN (
   SELECT PO.vin, SUM(COALESCE(P.cost * P.quantity, 0)) AS PartsCost
   FROM PartOrder PO
   LEFT JOIN Part P ON PO.orderNumber = P.orderNumber
   GROUP BY PO.vin
) Parts ON BF.vin = Parts.vin -- This is to grab the PartsCost
WHERE YEAR(BF.transactionDate) = @SelectedYear
AND MONTH(BF.transactionDate) = @SelectedMonth
GROUP BY U.username
ORDER BY TotalVehiclesSold DESC, TotalSales DESC;
```

## View Price Per Condition Report (Report Page)

### Abstract Code

- User clicks the **View Price Per Condition Report** button.
- Calculate the price of vehicles
- If role validation is successful for manager or owner, then:
    - Sort data by ***vehicleType***
    - Sort data by ***condition***
    - Sort data by ***averagePrice***
- For each unique combination of ***vehicleType*** and ***condition***:
    - Filter cars that match the current combination of ***vehicleType*** and ***condition***.

Table of Contents

- If there are matching cars:
    - Calculate the ***averagePrice*** for these cars.
    - Display the ***averagePrice*** in the report.
- Else display "@0" for this combination in the report.

```
SELECT
   VT.type AS VehicleType, -- Selecting the VehicleType from VehicleType table
   Conditions.CarCondition, -- Selecting the CarCondition from the generated list of conditions
   ROUND(AVG(
      CASE
         WHEN V.carCondition = Conditions.CarCondition -- Checking condition match
         THEN 1.1 * COALESCE(IndividualPartsCost, 0) + 1.25 *
COALESCE(ST.purchasePrice, 0) -- Price calculation
         ELSE 0 -- Default case if condition doesn't match
      END
   ), 2) AS AveragePrice -- Displaying the rounded average price
FROM VehicleType VT -- Selecting from VehicleType table
CROSS JOIN ( -- Generating a list of conditions to pair with VehicleType
   SELECT 'Excellent' AS CarCondition
   UNION ALL
   SELECT 'Very Good'
   UNION ALL
   SELECT 'Good'
   UNION ALL
   SELECT 'Fair'
) Conditions
LEFT JOIN Of_Type OT ON VT.type = OT.type -- Joining the VehicleType and Of_Type tables
LEFT JOIN Vehicle V ON OT.vin = V.vin -- Joining the Vehicle and Of_Type tables
LEFT JOIN (
   SELECT
      P.vin,
      COALESCE(SUM(P.cost * COALESCE(P.quantity, 0)), 0) AS IndividualPartsCost --
Calculating total part costs considering the quantity
   FROM Part P
   GROUP BY P.vin -- Grouping by vehicle VIN to calculate total part costs per vehicle
) Parts ON V.vin = Parts.vin -- Joining Parts table with Vehicle table based on VIN
LEFT JOIN Sells_To ST ON V.vin = ST.vin -- Joining the Sells_To table with Vehicle table
GROUP BY VT.type, Conditions.CarCondition -- Grouping the results by VehicleType and
CarCondition
```

Table of Contents

```
ORDER BY VT.type, FIELD(Conditions.CarCondition, 'Excellent', 'Very Good', 'Good', 'Fair');
-- Sorting the output by VehicleType and specified CarCondition order
```

## View Parts Statistics Report (Report Page)

### Abstract Code

- User clicks the ***View Parts Statistics Report*** button.
- Initialize lists/arrays for ***part***, ***part order***, ***vendor*** data, and ***report statistics***.
- For each part in the **Part**, **Vendor**, and **Part Order** table:
    - Store ***part***, ***partOrder***, ***vendor*** data in report statistics
- Sort report statistics by ***partNumber*** and ***orderNumber.***

```
SELECT
   V.name AS VendorName,
   COUNT(P.partNumber) AS TotalPartsSupplied,
   SUM(P.quantity) AS TotalPartsQuantity,
   SUM(P.quantity * P.cost) AS TotalDollarAmount
FROM
   Part P
JOIN
   PartOrder PO ON P.vin = PO.vin AND P.orderNumber = PO.orderNumber
JOIN
   Vendor V ON PO.vendorName = V.name
GROUP BY
   VendorName;
```

## Add Vehicle Info

### Abstract Code

- **View car details**. More details of this subtask are listed in its relevant section.
- Present the user with the button of ***Add Vehicle*** next to each vehicle from the previous step.
- When no button is pushed, do nothing; when ***Add Vehicle*** is pushed then load the **Add Vehicle Form**.

Table of Contents

- User clicks ***Search/view customer*** button (jump to **Search/View Customer Info**) to search existing customer; if not an existing customer, user clicks ***add customer info*** button (jump to **Add Customer Info**)
- User clicks on ***Link to a customer*** button to link the customer from previous step (with Customer ID saved as @customerID) and the fields to enter vehicle information including vin (@vin), vehicle type (@vehicleType), model year (@modelYear), manufacturer (@manufacturer), model name (@modelName), fuel type (@fuelType), color (@color), mileage (@mileage), description (@description), purchase price (@purchasePrice), and car condition (@carCondition).
    - If ***Save***: Update and store Vehicle info. View Vehicle Info.

```
INSERT INTO Vehicle (vin, modelYear, modelName, fuelType, mileage, description,
carCondition)
VALUES (@vin, @modelYear, @modelName, @fuelType, @mileage, @description,
@carCondition);

INSERT INTO Of_Color (vin, color)
VALUES (@vin, @color);

INSERT INTO Of_Type (vin, type)
VALUES (@vin, @vehicleType);

INSERT INTO Manufactured_By (vin, company)
VALUES (@vin, @manufacturer);

INSERT INTO Sells_To (customerID, username, vin, purchaseDate, purchasePrice)
VALUES (@customerID, @username, @vin, @purchaseDate, @purchasePrice);
```

- If a wrong data type is entered, display an ‘Error: wrong data type, please recheck your entry’ message when the user clicks ***Save***.
- If no value is entered for the required fields, display an ‘Error: empty entry, please recheck your entry’ message when the user clicks ***Save***.
- If value outside of what is presented in the VehicleColor, VehicleType and Manufacturer tables is entered, display an ‘Error: please use values from dropdown list only and recheck your entry’ message when the user clicks ***Save***.
- If ***Add another color***: Display the drop down list of colors, users enter vin (@vin) and color (@color).

```
INSERT INTO Of_Color (vin, color)
VALUES (@vin, @color);
```

- If ***Cancel***: Go to **Display Cars List (Main Page)** with user logged in.

Table of Contents

## Edit/Confirm Sale Order

### Abstract Code

- **Search Vehicle** using VIN. View Vehicle.

```
SELECT
   v.vin,
   ot.type,
   v.modelYear,
   mb.company AS manufacturer,
   v.modelName,
   v.fuelType,
   v.mileage,
   v.description,
  (1.1 * COALESCE((SELECT -- might not be in partorder, default to have part price 0
         SUM(P.cost * P.quantity)
      FROM
         PartOrder PO
            JOIN
         Part P ON P.orderNumber = PO.orderNumber
      WHERE
         PO.vin = v.vin),0) + 1.25 * (SELECT
         s.purchasePrice
      FROM
         Sells_To s
      WHERE
         s.vin = v.vin)) AS price, -- This is a subquery to calculate the price
   GROUP_CONCAT(DISTINCT vc.color) AS colors -- concatenate all different colors in a row
FROM
   Vehicle v
      JOIN
   Of_Type ot ON v.vin = ot.vin
      JOIN
   Manufactured_By mb ON v.vin = mb.vin
      JOIN
   Of_Color vc ON v.vin = vc.vin

 WHERE
   (v.vin = @vin
   AND v.vin NOT IN (
           SELECT DISTINCT  p.vin
           FROM Part p
           WHERE p.status != 'installed'
))
GROUP BY v.vin; -- this groupby is for concatenating colors
```

Table of Contents

- Click on the ***Sale the Car*** button then load the **Sale Order Form**. Do nothing if no button is clicked.
- On **Sale Order Form**, the user clicks ***Search for the customer*** button and fields for driver's license and tax ID are presented. Run **Search/View Customer Info** task (described as another task below).
- If the customer does not exist from the previous step, display 'No customer is found' message and present **Add Customer Info** (described as another task below).
- Click on ***Link this customer*** to link the customer from either of the above steps.
    - If ***Save***: Update and store Sale Order info. View Vehicle Info.
    - If a wrong data type is entered or required fields are missing, display an 'Error: please recheck your entry' message when the user clicks ***Save***.
    - If no value is entered for the required fields, display an 'Error: empty entry, please recheck your entry' message when the user clicks ***Save***.
    - If ***Cancel***: Go to View Vehicle info for existing VIN.

```
INSERT INTO Buys_from (customerID, username, vin, transactionDate)
VALUES (@customerID, @username, @vin, @transactionDate)
```

## Search/View Customer Info

### Abstract Code

- Search by either driver's license or tax id. Find the current customer by driver's license (@driverLicense) if customer is an individual, or find customer using tax id (@taxID) if customer is a business.

```
SELECT C.CustomerID, C.email, C.phoneNumber, C.street, C.city, C.state, C.postalCode,
I.driverLicense, I.firstName, I.lastName
FROM Customer AS C JOIN Individual AS I ON C.customerID = I.customerID
WHERE I.driverLicense = @driverLicense;

SELECT C.CustomerID, C.email, C.phoneNumber, C.street, C.city, C.state, C.postalCode,
B.taxID, B.businessName, B.name, B.title
FROM Customer AS C INNER JOIN Business AS B ON C.customerID = B.customerID
WHERE B.taxID = @taxID;
```

- If customer profile is found:
    - Display email, phone number, customerID and address(state/city/street/postal code)
    - Additionally, display driver's license, name(firstName/lastName) for individual customers and display taxid, business name, primary contact(name/title) for business customers.

Table of Contents

  - Display button ***Link this customer***. Store CustomerID (@CustomerID) for other tasks.
- If no existing customer profile is found, jump to **Add Customer Info** form as described below.

## Add Customer Info

### Abstract Code

- Present ***Individual*** and ***Business*** buttons in page bottom and do the following:
  - If click on ***Individual***:
    - Display empty fields for user to enter email (@email), phone number (@phoneNumber), street (@street), city (@city), state (@state), postal code (@postalCode), driver license (@driverLicense), first name (@firstName), and last name (@lastName).
    - After entering the above information and clicking the ***Save*** button, store the Customer and Individual information and display the newly saved customer info.

```
INSERT INTO Customer (email, phoneNumber, street, city, state, postalCode)
VALUES (@email, @phoneNumber, @street, @city, @state, @postalCode);

INSERT INTO Individual (driverLicense, firstName, lastName, customerID)
VALUES (@driverLicense, @firstName, @lastName, (SELECT customerID FROM Customer
WHERE email = @email AND  phoneNumber = @phoneNumber AND street = @street AND
city = @city AND  state = @state AND  postalCode = @postalCode));

SELECT C.CustomerID, C.email, C.phoneNumber, C.street, C.city, C.state, C.postalCode,
I.driverLicense, I.firstName, I.lastName
FROM Customer AS C JOIN Individual AS I ON C.customerID = I.customerID
WHERE I.driverLicense = @driverLicense;
```

  - If click on ***Business***:
    - Display empty fields for user to email (@email), phone number (@phoneNumber), street (@street), city (@city), state (@state), postal code (@postalCode), tax ID (@taxID), business name (@businessName), contact person name (@name) and title (@title)
    - After entering the above information and clicking the ***Save*** button, store the Customer and Business information and display the newly saved customer info.

```
INSERT INTO Customer (email, phoneNumber, street, city, state, postalCode)
VALUES (@email, @phoneNumber, @street, @city, @state, @postalCode);
```

```
INSERT INTO Business (taxID, businessName, name, title, customerID)
VALUES (@taxID, @businessName, @name, @title, (SELECT customerID FROM Customer
WHERE email = @email AND phoneNumber = @phoneNumber AND  street = @street AND
city = @city AND  state = @state AND  postalCode = @postalCode));

SELECT C.CustomerID, C.email, C.phoneNumber, C.street, C.city, C.state, C.postalCode,
B.taxID, B.businessName, B.name, B.title
FROM Customer AS C INNER JOIN Business AS B ON C.customerID = B.customerID
WHERE B.taxID = @taxID;
```

- Display button ***Link this customer***. Store CustomerID (@CustomerID) for other tasks.
- If a wrong data type is entered or required fields are missing, display an 'Error: please recheck your entry' message when the user clicks ***Save***.
- if ***Cancel***, jump to **Search/View Customer Info** page.

Table of Contents