Table of Contents:

Main Menu/Dashboard

Abstract code:

- **View statistics information**
- Show "***View Manufacturer's Product Report***", "***View Category Report***", "***View Actual versus Predicted Revenue Report***", "***View Store Revenue Report***", "***View Air Conditioner Sold on Groundhog Day***", "***View Sate with Highest Volume for each Category***", "***View Revenue by Population***", "***Managers***", "***Holidays***" and "***Cities***" tabs
- User click on ***View Manufacturer's Product Report*** button in **Main Menu** - Jump to the **Product Report 1** task
- User click ***View Category Report*** button in **Main Menu** - Jump to the **Product Report 2** task
- User click ***View Actual versus Predicted Revenue Report*** button- Jump to the **Produce Report 3** task
- User click ***View Store Revenue Report*** button- Jump to the **Produce Report 4** task
- User click ***View Air Conditioner Sold on Groundhog Day*** button- Jump to the **Produce Report 5** task
- User click ***View Sate with Highest Volume for each Category*** button- Jump to the **Produce Report 6** task
- User click ***View Revenue by Population*** button- Jump to the **Produce Report 7** task
- User click "***View Managers***" button, jump to **View/List Managers**
- User click "***Holidays***" button, jump to **View Holiday Information**
- User click "***Cities***" button, jump to **View City Infomation**

Task: View statistics information

Abstract code:

- Retrieve STORE record; count and display "the count of stores"
- Retrieve MANUFACTURER record; count and display "the count of manufacturers"
- Retrieve PRODUCT record; count and display "the count of products"
- Retrieve MANAGER record; count and display "the count of managers"

```
SELECT COUNT(Store_number) AS Total_Store, COUNT(Manuf_name) AS
Total_Manufacturer, COUNT(PID) AS Total_Product, COUNT(Email) AS Total_Manager
FROM Store, Manufacturer, Product, Manager;
```

Holidays

Abstract code:

- User clicks on ***Holidays*** Button in the **Main Menu**
- Retrieve and display list of all holiday information:
  - Holiday name
  - Date

```
SELECT Year, Month, Day, Holiday_name AS Holiday FROM Holiday;
```

Task: Add Holiday Information

Abstract code:

- User clicks ***Add Holiday*** Button in the **Holidays** form
- Display additional room for another holiday, including input fields: *year (*$year*), month (*$month*), day* ($day) and *name* ($holiday_name)
- User input year, month, day and holiday name
- When user click ***Save*** button
-

```
SELECT Holiday_name FROM Holiday;
```

If user input '$holiday_name'=Holiday.Holiday_name:

Pops error message "Holiday has existed"

```
SELECT Year, Month, Day FROM Holiday;
```

If user input date '$year'= Holiday.Year AND '$month'= Holiday.Month AND '$day'= Holiday.Day:

Pops error message "Holiday has existed"

Else: insert a record into the Holiday table

```
INSERT INTO Holiday (Holiday_name, Year, Month, Day) VALUES ('$holiday_name', '$year', '$month', '$day');
```

Managers

Abstract code:

- User clicks ***Managers*** Button in the **Main Menu**
- Sort and display list of all mangers by last name ascending
  For each manger record, display:
  - Manager name
  - Manager E-mail
  - If active
  - Store Number of the store he/she is assigned

```
SELECT First_name, Middle_name, Last_name, Email, If_active, Store_number
FROM AssignedManager, Manager WHERE
Manager.Email=AssignedManager.Email;
```

Edit Managers

Abstract code:

- User click ***Edit Managers*** button in **Managers**

```
SELECT First_name, Middle_name, Last_name, Email, If_active, Store_number
FROM AssignedManager, Manager WHERE
Manager.Email=AssignedManager.Email;
```

- For each manager record, populate button ***Delete Manager, Modify Manager*** dropdown
- When a button is clicked, then do the following:
  - If ***Add Manager*** button is clicked: **Add Manager**; **View/List Managers**
  - If ***Delete Manager*** button is clicked; **Delete Manager**; **View/List Managers**
  - If ***Modify Manager*** button is clicked; **Modify Manager**; **View/List Managers**
  - If ***Cancel*** button is clicked; go to **View/List Mangers** for current records

Task: Add Manager

Abstract code:

- User click ***Add Manager*** button in **Edit Managers**
- Display additional room for another manager, which contains:
  First_name, Middle_name, Last_Name input field
  Email input field
  A dropdown menu option for *Store Number*
- User enter manager name ($First_name, $Middle_name, $Last_name), email-address ($Email), and assigned store number ($Store_number)
- User click ***Save*** button

```
SELECT Email FROM Manager;
```

If exist: user input '$Email'=Manager.Email:
    Pops error message "This manager has existed"
Else: insert a record into the Holiday table

```
INSERT INTO Manager (Email, First_name, Middle_name, Last_name, If_active)
VALUES ('$Email', '$First_name', '$Middle_name', '$Last_name', 'Yes');
INSERT INTO AssignedManager (Store_number, Email) VALUES
('$Store_number', '$Email');
```

Task: Delete Manager

Abstract code:

- User click ***Delete Manage**r* button in **Edit Managers**

//Assume the application manages $selected_manager_email

```
SELECT Email, Store_number FROM AssignedManager, Manager
WHERE Manager.Email=AssignedManager.Email AND
Manager.Email='$selected_manager_email';
```

- If for the to-be-delete manger, exist AssignedManager.Store_number!='Null':
  Pops error message "A manger has to be unassigned from all stores before deleted"
- Else: The specific manager record will be deleted from the database

```
DELETE FROM Manager WHERE Manager.Email='$selected_manager_email';
```

Task: Modify Manager

Abstract code:

- User click ***Modify Manager*** button in **Edit Managers,**

//Assume the application manages $selected_manager_email

```
SELECT Email, First_name, Middle_name, Last_name, If-active,
Store_number FROM AssignedManager, Manager WHERE
Manager.Email=AssignedManager.Email AND
Manager.Email='$selected_manager_email';
```

- User may change manager's active status, delete or add assigned store number

//Assume the application mangers the user inputs of the $updated_status, $unassigned_store, $assign_new_store

- If user click ***Add Assigned Store*** button
  Display a new filed for user input new assigned store number $assign_new_store
  Use input new store number and click ***Save*** button, the record is updated in the database

```
INSERT INTO AssignedManager (Store_number, Email) VALUES
('$assign_new_store', '$selected_manager_email');
```

- User click ***Delete Assigned Store*** button in the current manager record table
  If the to be deleted store is the only store that is assigned to the manager, set the value of the store_number to the manger as 'Null'

```
UPDATE AssignedManager SET Store_number='Null' WHERE
AssignedManager.Email='$selected_manager_email';
```

  Else: assigned store number is deleted from the manager's record

```
DELETE FROM AssignedManager WHERE
AssignedManager.Email='$selected_manager_email' AND
AssignedManager.Store_number='$unassigned_store';
```

- If user changes the mannger's status and click **Save** button, the record is updated.

```
UPDATE Manager SET If_active='$updated_status' WHERE
AssignedManager.Email='$selected_manager_email';
```

Cities

Abstract code:

- User clicks ***Cities*** Button in the **Main Menu**
- Retrieve and display list of all city information:
  - City name
  - State
  - Population

```
SELECT City_name, State, Population FROM City;
```

- Adjacent to each row in form is *Update* Buttons

Task: Update city population

Abstract code:

- User clicks ***Update*** Button on the **Cities** form

// Assume the application mangers the $selected_city

- City population field entry change to blank.
- User enters population into the population field ($new_population) on form
- User click ***Save*** button, update the record in database

```
UPDATE City SET Population='$new_population' WHERE City_name='$selected_city';
```

Manufacturer' Product Report

Abstract code:

- User clicked on **View Manufacturer's Product Report** button from **Main Menu**
- Retrieve a complete list of all manufacturer records with their corresponding Product records
- For each manufacturer, retrieve the manufacturer's name, **COUNT** the total number of the products offered by the manufacturer; find the **average** retail price of all the manufacturer's products, the **maximum** retail price and the **minimum** retail price.
- Sort the list of manufacturer entries by average retail price descending
- Display Manufacturer's Product Report, for each manufacturer, display:
  - Manufacturer's name
  - Total number of products offered by the manufacturer
  - Average retail price
  - Minimum retail price
  - Maximum retail price

```
SELECT Manuf_name AS Manufacturer, AVG (Retail_price) AS Average_retail_price,
MAX (Retail_price) AS Max_price, MIN (Retail_price) AS Min_price FROM
Manufacturer, Product WHERE Manufacturer.Manuf_name=Product.Manuf_name
GROUP BY Manuf_name ORDER BY AVG (Retail_price) DESC;
```

- The application managers to display top 100 manufactures based on average price
- Adjacent to each row in form with ***VIEW Details*** Buttons
- Upon click View Details, execute the VIEW Manufacturer Details task

Task: View Manufacture Details

Abstract code:

- User clicked on the ***VIEW Details*** Buttons button in manufacturer table in **Report 1**
- For the specific manufacturer selected by the user, retrieve and display manufacturer name, and maximum discount of its products, the total number of the products offered by the manufacturer; average retail price, the maximum retail price and the minimum retail price in the header

//Assume the application managers the $selected_manufacturer

```
SELECT Manuf_name AS Manufacturer, Max_discount, AVG (Retail_price) AS Average_retail_price
FROM Manufacturer, Product
WHERE Manufacturer.Manuf_name=Product.Manuf_name AND
Manufacturer.Manuf_name='$selected_manufacturer';
```

- Sort the list of the products offered by the manufacturer by retail price descending
- For each product offered by the manufacturer, display:
  - Product ID
  - Name
  - Retail price

- Category (concatenated)

```
SELECT PID, Pname, Retail_price AS Price, Category_name AS Category FROM
Product, Category, ProductCategory WHERE Product.PID=ProductCategory.PID AND
Category.Category_name=ProductCategory.Category_name AND
Product.Manuf_name='$selected_manufacturer' ORDER BY Price DESC;
```

Category Report

Abstract code:

- User clicked on **View Category Report** button from **Main Menu**
- For each category:
  - Retrieve and display category name
  - Count and display the total number of products in the category
  - Retrieve and display average retail price of all the products in the category
  - Find and display total number of unique manufacturers offering products in the category

```
SELECT Category_name, COUNT (PID) AS Total_product, COUNT (DISTACT Manuf_name) AS
Total_manufacturer, AVG (Retail_price) AS Average_price
FROM Product, Category, ProductCategory, Manufacturer
WHERE Product.PID=ProductCategory.PID AND Category.Category_name=ProductCategory.Category_name
AND Product.Manuf_name=Manufacturer.Manuf_name
GROUP BY Category_name
ORDER BY Category_name ASCE;
```

Actual versus Predicted Revenue for GPS units

Abstract code:

- User clicked on ***Actual versus Predicted Revenue*** button after selecting ***GPS Products*** drop-down filter from **Main Menu**:
- Run the **Actual versus Predicted Revenue** task: query for the difference between the actual revenue and the predicted revenue.
- Find all Products belonging to Category Name of GPS.
- For each Product in GPS Category:
    - Display Product ID (PID), Product Name, Retail Price

```
CREATE VIEW GPSProduct AS
SELECT Product.PID AS GPSPID, Product.PName, Product.Retail_price
FROM ProductCategory, Product
WHERE ProductCategory.PID=Product.PID AND ProductCategory.Category_name='GPS';
```

    - Find all Dates and Sale price when Product was sold in Sale Price.
    - Find Sale Quantity sold on Dates of Sale Price, and the total number of GPS products sold at Sale Price by summing up Sale Quantity
    - Calculate Total Sale Revenue of Product sold in Sale Price by summing up the product of Sale Price and Sale Quantity.

```
CREATE VIEW GPSSale AS
SELECT SalePrice.PID, SalePrice.Year, SalePrice.Month, SalePrice.Day. SalePrice.Sale_price,
Sold.Sold_quantity, SalePrice.Sale_price*Sold.Sold_quantity AS GPSSale.Daily_sale_revenue,
SUM(SalePrice.Sale_price*Sold.Sold_quantity) AS TotalSaleRevenue, COUNT(Sold.Sold_quantity)
AS TotalSaleQuantity
FROM GPSProduct,Sold, SalePrice
WHERE GPSProduct.GPSPID=Sold.PID AND Sold.PID=SalePrice.PID
AND SalePrice.Year=Sold.Year AND SalePrice.Month=Sold.Month
AND SalePrice.Day=Sold.Day;
```

    - Calculate Predicted Revenue of Product sold in Sale Price by multiplying Retail Price and Predicted Quantity.
    - Calculate Total Actual Revenue of GPS Products by adding up Retail Revenue and Sale Revenue.
    - Calculate Total Predicted Revenue of GPS Products by adding up Retail Revenue and Predicted Revenue.

```
CREATE VIEW GPSPredict AS
SELECT SalePrice.PID, SalePrice.Year, SalePrice.Month, SalePrice.Day.
GPSProduct.Retail_price AS New_sale_price, Sold.Sold_quantity*0.75 AS
Predict_quantity, GPSProduct.Retail_price *Sold.Sold_quantity*0.75 AS
GPSSale.Daily_predict_revenue, SUM(GPSProduct.Retail_price
*Sold.Sold_quantity*0.75) AS TotalPredictRevenue
FROM GPSProduct,Sold, SalePrice
WHERE GPSProduct.GPSPID=Sold.PID AND Sold.PID=SalePrice.PID
AND SalePrice.Year=Sold.Year AND SalePrice.Month=Sold.Month
AND SalePrice.Day=Sold.Day;
```

  - Calculate the difference between total Actual Revenue and total Predicted Revenue.

- For each Product in GPS Category:
  - Display Total Actual Revenue, Total Predicted Revenue and their difference if the difference is larger than $5000 (positive or negative) with the difference sorted in descending order.

```
SELECT GPSProduct.GPSPID AS PID, GPSProduct.PName,
GPSProduct.Retail_price,SUM(Sold.Sold_quantity) AS TotalSoldQuantity,
GPSSale.TotalSaleQuantity, (GPSSale.TotalSaleRevenue –
GPSPredict.TotalPredictRevenue) AS Difference
FROM GPSProduct, GPSSale, GPSPredict, Sold
WHERE Sold.PID=GPSProduct.PID
AND ABS(GPSSale.TotalSaleRevenue–GPSPredict.TotalPredictRevenue)>5000
ORDER BY Difference DESC;
```

Store Revenue

Abstract code:

- User clicks ***View Store Revenue Report*** button
- For each distinct state in the drop down box

```
SELECT State FROM City;
```

- Find all the cities related to the state, display the city name.
- For each store related to the cities, display the store number, street address.
- Find the quantity of product sold with retail price and the quantity of product on sale.
- Find all the sold products, Display the year of the date that the products are sold.
- Find the quantity of product sold with retail price and the quantity of product on sale.
- Calculate revenue of each product in the store by:
- Each product revenue= *quantity* of product sold with retail price X retail price + *quantity* of product on sale X sale price.
- Sum up all the product revenue in each store to get the total revenue of each store.
- The report is ordered by year, then by revenue in descending order.

//Assume the application managers the selected state $selected_state

```
CREATE VIEW Regular_sold_date AS
((SELECT PID, Year, Month, Day FROM Sold)
EXCEPT
(SELECT PID, Year, Month, Day FROM SalePrice));
```

```
CREATE VIEW Regular_sale AS
SELECT Store_number, Address, City_name, State, Year, Month, Day, PID, Sold_quantity
FROM Regular_sold_date, Sold, Store, City
WHERE Regular_sold_date.PID=Sold.PID AND Regular_sold_date.Year=Sold.Year AND
Regular_sold_date.Month=Sold.Month AND Regular_sold_date.Day=Sold.Day AND
Sold.Store_number=Store.Store_number AND Store.City_name=City.Name AND
Store.State=City.State AND City.State='$selected_state';
```

```
CREATE VIEW Regular_sale_value AS
SELECT Store_number, Address, City_name, State, Year, Month, Day, PID,
Sold_quantity*Retail_price AS Sold_value
FROM Regular_sale, Product
WHERE Regular_sale.PID=Product.PID;
```

```
CREATE VIEW On_sale_value AS
SELECT Store_number, Address, City_name, State, Year, Month, Day, PID,
Sold_quantity*Sale_price AS Sold_value
FROM SalePrice, Sold, Store, City
WHERE SalePrice.PID=Sold.PID AND SalePrice.Year=Sold.Year AND
SalePrice.Month=Sold.Month AND SalePrice.Day=Sold.Day AND
Sold.Store_number=Store.Store_number AND Store.City_name=City.Name AND
Store.State=City.State AND City.State='$selected_state';
```

```
SELECT Store_number, Address, City_name, Year, SUM (Sold_value) AS Revenue
FROM (Regular_sale_value JOIN On_sale_value)
WHERE Year in (SELECT Year
                    FROM (Regular_sale_value JOIN On_sale_value))
GROUP BY Store_number
ORDER BY Revenue DESC
ORDER BY Year ASCE;
```

Air Conditioners on Groundhog Day

Abstract Code

- User click ***View Air Conditioner Sold on Groundhog Day*** button on **Main Menu**
- Filter products to leave those in Air Conditioner Category
- Group the sold information by year, sum up all the quantities to get the total number of items sold that year.
    - Get average number of units sold per day by divide the total number by 365.

```
CREATE VIEW ACAllYear AS
SELECT year, SUM(sold_quantity)/365.0 AS sold_allyear FROM (Sold INNER JOIN
ProductCategory ON Sold.PID = ProductCategory.PID)
WHERE Category_name='Air Conditioner'
GROUP BY year;
```

- Group the sold information by each year's Groundhog Day, sum up all the quantities to get total number of units sold at that day.

```
CREATE VIEW ACGroundHog AS
SELECT s.year, p.sold_quantity AS sold_groundhog FROM (Sold AS s INNER JOIN
ProductCategory AS p ON s.PID = p.PID INNER JOIN Holiday AS h ON (h.year = s.year AND
h.month = s.month AND h.day = s.day))
WHERE (p.category_name = 'Air Conditioner' AND h.holiday_name = 'Groundhog Day');
```

- Join those two information above by year.
    - Sort on the year in ascending order

```
SELECT a.year, a.sold_all_year, g.sold_groudhog FROM (ACAllYear AS a INNER JOIN
ACGroundHog AS g ON a.year = g.year)
ORDER BY a.year;
```

State with Highest Volume for each Category

Abstract Code

- User click ***View Sate with Highest Volume for each Category*** button on **Main Menu**
- User input *Year* and *Month* into input fields and click *look up* button.
- Do **View Highest Volume State task:**
    - Filter sold date with user input's year and month.
    - Group the product by category, and for each category:
        - Group the store by state, sum up the quantity to get total sale volume for that sate
        - Find the state with the maximum volume, return the state name and the volume
    - Sort by category name in ascending order.

```
CREATE VIEW TempView1 AS
SELECT s.store_number, s.PID, s.sold_quantity, p.category_name FROM (Sold AS s
INNER JOIN ProductCategory AS p ON s.PID = p.PID)
WHERE s.year = '$Year' AND s.month = '$Month';

CREATE VIEW TempView2 AS
SELECT s.store_number, c.state FROM (Store AS s
INNER JOIN City AS p ON s.city_name = c.city_name);

SELECT state, category_name, MAX(sum) FROM
  (SELECT state, category_name, SUM(sold_quantity) AS sum FROM
    (SELECT t1.store_number, t1.PID, t1.sold_quantity, t1.category_name, t2.state
      FROM TempView1 AS t1 INNER JOIN TempView2 AS t2
      ON t1.store_number = t2.store_number))
  GROUP BY state, category_name;
```

- Attach a hyperlink to each *row* of the main report
- Upon user click a *row*:
    - Do **View State Info:**
        - Get the category, year/month, state information from the main menu
        - Find all store in the that state
        - For each store:
            - Find city and managers belongs to this store.
            - Return store ID, address, city, and every unique manager as a row
        - Sort the store by ID in ascending order.
        - Display the date with the header (category, year/month, state)

```
SELECT s.store_number, s.address, s.city, m.email, m.first_name, m.last_name
FROM (Store as s INNER JOIN City as ON s.city_name = c.city_name
INNER JOIN AssignedManager AS a ON a.store_number = s.store_number
INNER JOIN Manager AS m ON m.email = a.email)
WHERE c.state = '$State'
ORDER BY s.store_number;
```

Revenue by Population

Abstract Code

- User clicked on ***Revenue by Population*** button from **Main Menu**:
- Run the **Revenue by Population** task: query for information about average annual revenue for specific city population categories.
  - Find all unique Cities available in database by looking up both City Name and Store Number (ID) of Stores located in the City.

```
SELECT Store_number,City_name FROM 'Store';
```

  - For each unique City:

    Find City Population.

    Categorize City based on Population. City categories: Small (population <3,700,000), Medium (population >=3,700,000 and <6,700,000), Large (population >=6,700,000 and <9,000,000) and Extra Large (population >=9,000,000).

```
GroupPopulation AS (SELECT City.Population, CASE WHEN
City.Population<3,700,000 THEN 'SMALL' WHEN City.Population BETWEEN
3,700,000 AND 6,700,000 THEN 'MEDIUM' WHEN City.Population BETWEEN
3,700,000 AND 9,000,000 THEN 'LARGE' WHEN City.Population>=9,000,000
```

    Find Store Number (ID) of all Stores located in the City.

```
CREATE VIEW CityStore AS
SELECT Store_number, City_name, State, Population, Populationgroup
FROM Store, City
WHERE Store.City_name=City.City_name AND Store.State=City.State;

CREATE VIEW SoldWithPrice AS
SELECT Year, Store_number, PID, Sold_quantity, IF(Sold.Year=SalePrice.Year AND
Sold.Month=SalePrice.Month AND Sold.Day=SalePrice.Day, SalePrice.Sale_price,
Product.Retail_price) AS Sold_price
FROM Product, Sold, SalePrice
WHERE Sold.PID=SalePrice.PID AND SalePrice.PID=Product.PID;
```

  - Find all unique Years available in database.
  - For each Year:
    - For each City:
      - For each Store in the City:
        - Find all Dates when Products were sold in Retail Price.
        - Find Product Quantity sold on Dates of Retail Price.

- Calculate Revenue of each Product sold in Retail Price in the Store by multiplying Retail Price and Product Quantity.
- Sum up Retail Revenue of all Products sold in Retail Price.
- Find all Dates when Products were sold in Sale Price.
- Find Product Quantity sold on Dates of Sale Price.
- Calculate Revenue of each Product sold in Sale Price in the Store by multiplying Sale Price and Product Quantity.
- Sum up Sale Revenue of all Products sold in Sale Price.
- Calculate total Revenue of Products in the Store of the City of the Year by adding up both Retail Revenue and Sale Revenue.

▪ Sum up total Revenue of all Stores in the City of the Year.

```
CREATE VIEW CitySold AS
SELECT Year, City_name, State, Population, Populationgroup, (Sold_quantity*Sold_price)
AS CityAnnualRevenue
FROM CityStore, SoldWithPrice;
```

▫ Sum up Revenue of all Cities of the Year of each City Category.
▫ Calculate average Revenue of a City Category in the Year by dividing total Revenue by number of Cities within a City Category.

▪ Sort results by Year (from oldest to newest) first and then by City Size Category (from smallest to largest).

```
SELECT Year, Populationgroup AS CityCategory,
SUM(CityAnnualRevenue)/COUNT(distinct (City_name, State)) AS CityCatAvenue FROM CitySold
WHERE Populationgroup='SMALL' GROUP BY Year
UNION
SELECT Year, Populationgroup AS CityCategory,
SUM(CityAnnualRevenue)/COUNT(distinct (City_name, State)) AS CityCatAvenue    FROM CitySold
WHERE Populationgroup='MEDIUM' GROUP BY Year
UNION
SELECT Year, Populationgroup AS CityCategory,
SUM(CityAnnualRevenue)/COUNT(distinct (City_name, State)) AS CityCatAvenue FROM CitySold
WHERE Populationgroup='LARGE' GROUP BY Year
UNION
SELECT Year, Populationgroup AS CityCategory,
SUM(CityAnnualRevenue)/COUNT(distinct (City_name, State)) AS CityCatAvenue FROM CitySold
WHERE Populationgroup='EXTRALARGE' GROUP BY Year
ORDER BY Year, CityCategory ASC;
```