## Table of Contents:

## Main Menu

Table of Contents

**Abstract Code**

- User clicks on ***View Statistics*** button from **Main Menu**:
- Run the **View Statistics:** query for information about the count of stores, manufacturer, products, and managers.
- Find and display count of stores;

```
SELECT COUNT(storeNumber) AS Count of Stores FROM Store;
```

- Find and display count of manufacturers;

```
SELECT COUNT(manufacturerName) AS Count of Manufacturers FROM
Manufacturer;
```

- Find and display count of products;

```
SELECT COUNT(productID) AS Count of Products FROM Product;
```

- Find and display count of managers;

```
SELECT COUNT(managerEmail) AS Count of Managers FROM Manager;
```

- There will be **Settings** form in the **Main Menu** where the user can edit Holiday, Manager, and population information in the data warehouse.
- Upon:
  - Click ***Edit Holidays button***- Jump to the **Edit Holidays** task.
    - User enters new *holiday name* ($holidayName) and *holiday date* ($holidayDate).

```
INSERT INTO Holiday VALUES (‘$holidayName’, ‘$holidayDate’)
```

  - Click ***Edit Manager’s Information*** button- Jump to **Edit Manager’s Information** task.
    - User clicks ***Add Manager’s Information*** button
      - User enters new Managers *name* ($managerName) and *email* ($managerEmail) and clicks enter

```
INSERT INTO Manager VALUES (‘$managerName’, ‘$managerEmail’)
```

    - User clicks ***Remove Manager’s Information*** button
      - User enters Managers *name* ($managerName) and *email* ($managerEmail) to be removed and clicks enter.

```
DELETE FROM Manager WHERE managerName =‘$managerName’ AND
managerEmail = ‘$managerEmail’
```

Table of Contents

        - User clicks ***Assign manager to store*** button
            - User selects *manager email* ($managerEmail) from list of available managers and *store Number* ($storeNumber) from list of available store numbers
            - If the manager is not already managing that store:

```
INSERT INTO Manages VALUES ('$managerEmail', '$storeNumber')
```

            - Else: Return error message stating manager is already managing store.
    - Click ***Edit Population of Cities*** button- Jump to **Edit Population of Cities** Task.
        - User enters a new *population* ($newPopulation) for a particular *city* ($cityName) and *state* ($state)

```
UPDATE City SET population = '$newPopulation' WHERE cityName = '$cityName'
and state = '$state'
```

- Show ***"View Manufacturer's Product Report"*** , ***"View Category Report", "View Actual versus Predicted Revenue", "View Store Revenue by Year by State", "View Air Conditioners Sale on GroundHog", "View State with Highest Volume for each Category", "View Revenue by Population"*** and **"Settings"** tabs in the navigation bar.
- Upon:
    - Click ***View Manufacturer's Product Report*** button- Jump to the **Report 1** task.
    - Click ***View Category Report*** button- Jump to the **Report 2** task.
    - Click ***View Actual versus Predicted Revenue*** button- Jump to the **Report 3** task.
    - Click ***View Store Revenue by Year by State*** button- Jump to the **Report 4** task.
    - Click ***View Air Conditioners Sale on GroundHog*** button- Jump to the **Report 5** task.
    - Click ***View State with Highest Volume for each Category*** button- Jump to the **Report 6** task.
    - Click ***View Revenue by Population*** button- Jump to the **Report 7** task.
    - Click **Settings** - Jump to the **Settings** task.

## Report 1 - Manufacturer's Product Report

**Abstract Code**

- User clicks on ***View Manufacturer's Product Report*** from **Main Menu**:

Table of Contents

- Run the **View Manufacturer's Product Report** task: query for information about the manufacturer and their products - manufacturer's ID, name, and total number of products offered by the manufacturer, average retail price of all the manufacturer's products, minimum retail price, and maximum retail price.
- Sort the results by average price with the highest average price appearing first, for only the top 100 manufacturers based on average price;

```
SELECT TOP 100 a.manufacturerName, COUNT(b.productId) as ProductCount,
AVG(b.retailPrice) as AverageRetailPrice,
MIN(b.retailPrice) as MinRetailPrice, MAX(b.retailPrice) as MaxRetailPrice
FROM dbo.Manufacturer as a
INNER JOIN dbo.Product as b
ON a.manufacturerName = b.manufacturerName
GROUP BY a.manufacturerName
ORDER BY AverageRetailPrice DESC;
```

- User clicks on ***manufacturer name*** hyperlink button in report to view drill down:
- Run the **View product drill-down** task. Drill down will query the details of manufacturer and its product - the manufacturer's details (name and maximum discount), the summary information from the parent report, and lists for each of the manufacturer's products' its product ID, name, category (or categories), and price, ordered by price descending (high to low).
- Display Manufacturer name and max discount;

```
SELECT Manufacturer.manufacturerName, Manufacturer.manufacMaxDiscount
FROM Manufacturer
WHERE Manufacturer.manufacturerName = `$manufacturerName`;
```

- Display summary  information from parent report.
- Find and display lists for each of the manufacturer's products' its product ID, name, category (or categories), and price, ordered by price descending (high to low);

```
SELECT
a.productId,
a.productName,
STRING_AGG(b.categoryName, ',') as Categories,
a.retailPrice
FROM
dbo.Product as a
INNER JOIN dbo.AssignedTo as b
ON a.productId = b.productId
INNER JOIN dbo.Manufacturer as c
ON a.manufacturerName = c.manufacturerName
WHERE
```

Table of Contents

```
C.manufacturerName = '$manufacturerName'
GROUP BY
a.manufacturerName, a.productId, a.productName, a.retailPrice,
c.manufacMaxDiscount
ORDER BY a.retailPrice DESC;
```

## <u>Report 2 - View Category Report</u>

**Abstract Code**

- User clicks on ***View Category Report*** button from **<u>Main Menu</u>**:
- Run the **View Category Report** task: query for information about the category - category name, total number of products in that category, total number of unique

manufacturers offering products in that category, and the average retail price (not including sale days) of all the products in that category.

- Find and display category name, count of product id, count of manufacturer name, and average of retail price and sort the results by category name in ascending;

```
SELECT
Category.categoryName as catName,
COUNT(AssignedTo.productId) as totalProducts,
COUNT(DISTINCT(Product.manufacturerName)) as totalManufacturers,
AVG(Product.retailPrice) as avgRetailPrice
FROM Category
LEFT JOIN Assignedto on Category.categoryName = AssignedTo.categoryName
LEFT JOIN Product on AssignedTo.productId = Product.productId
GROUP BY Category.categoryName
```

## Report 3 - Actual versus Predicted Revenue for GPS units

**Abstract Code**

- User clicked on ***View Actual versus Predicted Revenue*** for GPS units button from user **Main Menu**:
- Run the **View Actual vs Predicted Revenue** task: For each product in the GPS category, it returns the product ID, the name of the product, the product's retail price, the

total number of units ever sold, the total number of units sold at a discount, the total number of units sold at retail price, the actual revenue collected from all the sales of the product, the predicted revenue had the product never been put on sale (based on 75% volume selling at retail price), and the difference between the actual revenue and the predicted revenue.

- Only predicted revenue differences greater than $5000 (positive or negative) will be displayed and sorted in descending order;

```
SELECT
b.productId,
b.productName,
b.retailPrice,
SUM(b.saleItemsPurchased) as saleItemsPurchased,
SUM(b.fullPriceItemsPurchased) as fullPriceItemsPurchased,
SUM(b.actualRevenue) as actualRevenue,
SUM(b.predictedRevenue) as predictedRevenue,
SUM(actualRevenue) - SUM(predictedRevenue) as revenueDifference
FROM (SELECT
a.productId,
a.productName,
a.retailPrice,
CASE WHEN a.onSale = 1 then SUM(a.quantityPurchased) else null
end as saleItemsPurchased,
CASE WHEN a.onSale = 0 then SUM(a.quantityPurchased) else null
end as fullPriceItemsPurchased,
SUM(a.actualRevenue) as actualRevenue,
SUM(a.predictedRevenue) as predictedRevenue
FROM (SELECT
Sale.productId,
Product.productName,
Product.retailPrice,
Sale.quantityPurchased,
Sale.percentageDiscount,
Product.retailPrice * (1 - Sale.percentageDiscount) * Sale.quantityPurchased as
actualRevenue, CASE WHEN percentageDiscount != 0 then Product.retailPrice * (0.75
* Sale.quantityPurchased) else Product.retailPrice * Sale.quantityPurchased
end as predictedRevenue,
CASE WHEN percentageDiscount != 0 then 1 else 0 end as onSale
FROM Sale
Join Product on
Sale.productId = Product.productId
Join AssignedTo on
Product.productId = AssignedTo.productId
WHERE AssignedTo.categoryName = 'GPS') as a
GROUP BY productId, productName, retailPrice, onSale) as b
GROUP BY productId, productName, retailPrice;
```

## Report 4 – Store Revenue by Year by State

**Abstract Code**

- User clicked on ***View Store Revenue by Year by State*** button from **Main Menu**:
- Run the ***View Store Revenue by Year by State tas***k: Display all the states in the Drop-down box.
- When report is loaded display a drop down selection menu for states in the USA;
- User chooses state ($state) based on drop-down selection menu. Display the table that results from the query below (revenue calculation is included in SQL);

Table of Contents

```
SELECT
state,
storeID,
streetAddress,
cityName,
year,
sum(saleRevenue) as revenue
FROM (SELECT
City.state as state,
Store.StoreNumber as storeID,
Store.streetAddress as streetAddress, City.cityName as cityName,
(1 - Sale.percentageDiscount) * Product.retailPrice * Sale.quantityPurchased as
saleRevenue, year(Sale.saleDate) as year
FROM Store
join City on Store.state = City.state
join Sale on Store.storeNumber = Sale.storeNumber
join Product on Sale.productId = Product.productId) as total
WHERE state = '$state'
GROUP BY state, storeID, streetAddress, cityName, year
ORDER BY year ASC, revenue DESC;
```

## Report 5 – Air Conditioners on Groundhog Day?

**Abstract Code**

- User clicked on ***View Air Conditioners Sale on GroundHog*** Day button from **Main Menu**:
- Run the **View Air Conditioners Sale on GroundHog Day** task: Display the year, the total number of items sold that year in the air conditioning category, the average number of units sold per day, and the total number of units sold on Groundhog Day (February 2) of that year.
- Sort and display the report on the year in ascending order;

Table of Contents

```
SELECT YEAR(Sale.saleDate) as [Sales Year]
 ,SUM(Sale.[quantityPurchased]) [Items Sold]
,(SUM(Sale.[quantityPurchased])/365) as [Average Units Sold]
FROM Sale
JOIN AssignedTo ON Sale.productId = AssignedTo.productId
WHERE AssignedTo.[categoryName] = 'air conditioning'
GROUP BY YEAR(Sale.saleDate)
ORDER BY YEAR(Sale.saleDate) ASC
```

- FInd and display units sold on groundhog day;

```
SELECT YEAR(Sale.saleDate) as [Sales Year]
 ,SUM(Sale.[quantityPurchased]) [Items Sold]
,(SUM(Sale.[quantityPurchased])/365) as [Average Units Sold]
FROM Sale
JOIN AssignedTo ON Sale.productId = AssignedTo.productId
WHERE AssignedTo.[categoryName] = 'air conditioning'
AND MONTH(Sale.saleDate) = 2
AND DAY(Sale.saleDate) = 2
GROUP BY YEAR(Sale.saleDate)
ORDER BY YEAR(Sale.saleDate) ASC
```

## Report 6 - State with Highest Volume for each Category

**Abstract Code**

- User clicks on ***View State with Highest Volume for each Category*** button from **Main Menu:**
- Select a year and month from the available dates in the database.
- Run the **View the State with Highest Volume for each Category** task: For each category: the category name, the state that sold the highest number of units in that category (i.e., include items sold by all stores in the state), and the number of units that were sold by stores in that state.

Table of Contents

- This output shall be sorted by category name ascending. Note that each category will only be listed once unless two or more states tied for selling the highest number of units in that category;

```
SELECT [Category Name]
, [State]
, MAX([Units Sold]) AS [Units Sold]
FROM
(SELECT  AssignedTo.categoryName AS [Category Name]
 , Store.state AS [State]
, SUM(Sale.quantityPurchased) AS [Units Sold]
FROM Sale
JOIN AssignedTo ON Sale.productId = AssignedTo.productId
JOIN Store ON Sale.storeNumber = Store.StoreNumber
GROUP BY Store.state, AssignedTo.categoryName) Total
GROUP BY [State], [Category Name]
ORDER BY [Category Name] ASC;
```

- User clicks ***drill-down detail*** button for category; Drop-down detail for each row uses the criteria of state, category, and year/month to provide the IDs, names, and cities of all the stores and their managers names and email addresses so that they can be recognized for their efforts.
- This sub report is ordered by store ID ascending and the header includes the original criteria from the parent report (category,year/month, state).

```
SELECT Store.state AS [State]
,AssignedTo.categoryName AS [Category Name]
,CAST(YEAR(Sale.saleDate) AS CHAR(4)) + '/' + RIGHT('0' +
RTRIM(MONTH(Sale.saleDate)), 2) [Year/Month]
,Store.StoreNumber [Store Number]
,Store.cityName [City Name]
FROM Sale
JOIN Store ON Store.storeNumber = Store.StoreNumber
JOIN AssignedToON Sale.productId = AssignedTo.productId;
```

- If there is more than one manager, it is displayed as multiple rows.

## Report 7 - Revenue by Population

### Abstract Code

- User clicked on ***View Revenue by Population*** button from **Main Menu:**
- Run the **View Revenue by Population** task: Display a pivot table with years or city category as columns or as rows, with both attributes arranged in ascending order (oldest to newest for years, smallest to largest for city size).
- Display the following pivot table;

```
SELECT year,
avg(smallRevenue) AS avgSmallRevenue,
avg(medRevenue) AS avgMedRevenue,
avg(largeRevenue) AS avgLargeRevenue,
avg(xlRevenue) AS avgXLRevenue
FROM (
SELECT
year,
Case when population < 3700000 then saleRevenue
ELSE null
END AS smallRevenue,
Case when population >= 3700000 and population < 6700000 then saleRevenue
 ELSE null
END AS medRevenue,
Case when population >= 6700000 and population < 9000000 then saleRevenue
 ELSE null
END AS largeRevenue,
Case when population >= 9000000 then saleRevenue
ELSE null
END as xlRevenue
FROM (
SELECT
City.cityName AS city,
City.state AS state,
City.population AS population,
Store.state AS storeID,
Store.streetAddress AS streetAddress,
City.cityName AS cityName,                    (1-
Sale.percentageDiscount)*Product.retailPrice*Sale.quantityPurchased as
saleRevenue,year(Sale.saleDate) AS year
FROM Store
JOIN City
ON Store.state = City.state
JOIN Sale
ON Store.storeNumber = Sale.storeNumber
JOIN Product
ON Sale.productID = Product.productID) AS total) AS pivoted_total
GROUP BY year order BY year ASC;
```