Main Menu / Navigation Bar 2
View/Add Holiday 3
Update City Population 4
Top 100 Manufacturers and Drill-down 5
View Category Details 7
Revenue Gap Analysis 8
Yearly Store Revenue Filter by State 10
View Air Conditioners Sales on Groundhog Day 12
Top State for each Category Filter by Year-Month 13
View Revenue by Population 14
Membership Trend and Drill-down 16

Main Menu / Navigation Bar

Abstract Code

- When user successfully land on or navigate to **Main Menu** form:
  - Find Store and display the count of stores.

```
SELECT COUNT(DISTINCT store_number) FROM Store;
```

  - Find Manufacturer and display the count of manufacturers.

```
SELECT COUNT(DISTINCT manufacturerID) FROM Manufacturer;
```

  - Find Product and display the count of products sold.

```
SELECT COUNT(DISTINCT productID) FROM Product;
```

  - Find Membership and display the count of memberships sold.

```
SELECT COUNT(DISTINCT memberID) FROM Membership;
```

- Show "***View/Add Holiday***", "***Update City Population***", "***Manufacturer's Product Report***", "***Category Report***", "***Actual vs. Predicted Revenue for Speaker Units***", "***Store Revenue by Year by State***", "***Air Conditioners on Groundhog Day***", "***State with Highest Volume for each Category***", "***Revenue by Population***", and **"*Membership Trends*"** buttons.
- Upon:
  - Click ***View/Add Holiday*** button – Jump to the **View/Add Holiday** task.
  - Click ***Update City Population*** button – Jump to the **Update City Population** task.
  - Click ***Manufacturer's Product Report*** button – Jump to the **Top 100 Manufacturers and Drill-down** task.
  - Click ***Category Report*** button – Jump to the **View Category Details** task.
  - Click ***Actual vs. Predicted Revenue for Speaker Units*** button – Jump to the **Revenue Gap Analysis** task.
  - Click ***Store Revenue by Year by State*** button – Jump to the **Yearly Store Revenue Filter by State** task.
  - Click ***Air Conditioners on Groundhog Day*** button – Jump to the **View Air Conditioners Sales on Groundhog Day** task.
  - Click ***State with Highest Volume for each Category*** button – Jump to the **Top State for each Category Filter by Year-Month** task.
  - Click ***Revenue by Population*** button – Jump to the **View Revenue by Population** task.
  - Click ***Membership Trends*** button – Jump to the **View Memberships Sold per Year** task.

Table of Contents

View/Add Holiday

Abstract Code

- User click ***View Holiday*** button from **Main Menu** form or successfully submit add holiday requests.
- Run the **View Holiday** task: Find each Holiday; Display Holiday date and name.

```
SELECT holiday_date, holiday_name FROM Holiday;
```

- If user click ***Add Holiday*** button from **Holidays Information List** form, then:
  - User enter *date* ($date) and *holiday* ($holiday) input fields.
    - Data validation: check *date* and *holiday* input fields are not duplicated:
      - When ***Save*** button is clicked:

```
SELECT holiday_date, holiday_name FROM Holiday
WHERE holiday_date = '$date' and holiday_name = '$holiday';
```

        - If Holiday record is found:
          - Go back to **Add Holiday** form, with error message.
        - Else:
          - Insert Holiday record.

```
INSERT INTO Holiday (holiday_date, holiday_name) VALUE ('$date', '$holiday');
```

          - Go to **Holidays Information List** form.
    - Else *date* and *holiday* input fields are invalid, display **Add Holiday** form, with error message.
- Else no action is taken on **Holiday Information List** form then do nothing.
- When ready, user can select next action from Navigation Bar.

Table of Contents

Update City Population

Abstract Code

- User click ***Update City Population*** button from **Main Menu** form.
- Run **View City Profile** task: Find each City; Display CityName, State, and Population.

```
SELECT city_name, state, population FROM City;
```

- If user click ***Edit*** button on **City Information List** form, then all city population fields become editable.
    - If user edit *population* ($population) input field, then:
        - Data validation: check if *population* input field is numeric, then:
            - When ***Save*** button is clicked:

```
SELECT city_name, state, population FROM City
WHERE cityID = '$cityid';
```

                - If City record is found and City.Population != $population:
                    - Update City record.

```
UPDATE City SET population = '$population' WHERE cityID = '$cityid';
```

                    - Display **City Information List** form.
                - Else:
                    - Display **City Information List** form, with error message.
        - Else *population* input field is invalid, display **City Information List** form, with error message.
    - Else user click ***Cancel*** button, then go back to **City Information List** form.
- Else no action is taken on **City Information List** form then do nothing.
- When ready, user can select next action from Navigation Bar.

Table of Contents

Top 100 Manufacturers and Drill-down

Abstract Code

- User click ***Manufacturer's Product Report*** button from **Main Menu** form.
- Run the **Top 100 Manufacturers** task:
    - Find all Manufactuer and retrieve ManufacturerID and ManufacturerName.
    - For each Manufacturer, find all Product produced by that Manufacturer and retrieve ManufacturerID, ManufacturerName, ProductID and RetailPrice.
        - Count total number of products offered by that manufacturer using ProductID.
        - Calculate Average Retail Price, Minimum Retail Price, Maximum Retail Price by aggregating ManufacturerID and ManufacturerName.
        - Calculate Retail Price Range as the difference of maximum retail price and minimum retail price for products offered by that manufacturer.
    - Dispaly ManufacturerID, ManufacturerName, Total number of products offered by the manfuacturer, Average Retail Price, Minimum Retail Price, Maximum Retail Price, and Retail Price Range.
    - Sort by Average Retail Price in descending order.
    - Limit results to top 100 manufacturers based on Average Retail Price.

```
SELECT
    m.manufacturerID,
    manufacturer_name,
    COUNT(DISTINCT productID) AS total_products_offered,
    ROUND(AVG(retail_price),2) AS avg_retail_price,
    MIN(retail_price) AS min_retail_price,
    MAX(retail_price) AS max_retail_price,
    ROUND(MAX(retail_price) - MIN(retail_price),2) AS retail_price_range
FROM Manufacturer AS m
    LEFT JOIN Product AS p ON m.manufacturerID = p.manufacturerID
GROUP BY 1
ORDER BY 4 DESC
LIMIT 100;
```

- When ready, user can click drill-down or select next action from Navigation Bar.

Table of Contents

- If user click ***drill-down*** button on a ManufacturerName, then run **Top 100 Manufacturers Drill-down** task:
    - Query for information about selected Manufacturer where $manufactuerid is stored in the web session.
    - Find all Products offered by current Manufactuer using ManufacturerID.
    - For each Product:
        - Find all Category(s) associated with that product using ProductID.
        - Concatenate all CategoryName in alphabetical order, separated by forward slashes.
    - Display ManufacturerID, ManufacturerName, and MaximumDiscount in the report header; And for each product, display ProductID, ProductName, CategoryName(s) and Retail Price.
    - Sort by Retail Price in descending order then ProductID in ascending order.

```
SELECT
    p.productID,
    product_name,
    GROUP_CONCAT(DISTINCT category_name
                 ORDER BY category_name ASC SEPARATOR ' / ') AS category,
    retail_price
FROM Product AS p
    LEFT JOIN ProductCategory AS pc ON p.productID = pc.productID
WHERE manufacturerID = '$manufacturerid'
GROUP BY 1
ORDER BY 4 DESC, 1 ASC;
```

- Else no action is taken on **Manufacturer's Product Report** form then do nothing.
- When ready, user can select next action from Navigation Bar.

Table of Contents

View Category Details

Abstract Code

- User click ***Category Report*** button from **Main Menu** form.
- Run the **View Category Details** task:
  - For each Category, find all Product associated with that Category and retrieve CategoryName, ProductID, ManufacturerID, and Retail Price.
    - Count total number of products in that category using ProductID.
    - Count total number of unique manufacturers offering products in that category using ManufacturerID.
    - Calculate average retail price for all products in that category.
    - For each Product, find all Sales and OnSale related to that product.
      - If SaleDate equals to OnSaleDate, the Actual Selling Price is DiscountPrice else Retail Price.
      - Calculate total revenue for all products sold in that category by adding the result of QuantitySold multiply by Actual Selling Price.
  - Display CategoryName, Total number of products in that category, Total number of unique manufacturers offering products in that category, Average Retail Price, and Total Revenue for all products sold in that category.
  - Sort by CategoryName ascending.

```
SELECT category_name,
    COUNT(DISTINCT pc.productID) AS total_products,
    COUNT(DISTINCT manufacturerID) AS total_manufacturers,
    ROUND(AVG(retail_price), 2) AS avg_retail_price,
    ROUND(SUM(CASE WHEN discount_price IS NOT NULL THEN
                    discount_price * quantity_sold
                ELSE retail_price * quantity_sold END), 2) AS total_revenue
FROM ProductCategory AS pc
LEFT JOIN Product AS p ON pc.productID = p.productID
LEFT JOIN Sale AS t ON pc.productID = t.productID
LEFT JOIN OnSale AS o ON pc.productID = o.productID AND t.sale_date = o.onsale_date
GROUP BY 1
ORDER BY 1;
```

- When ready, user can select next action from Navigation Bar.

Table of Contents

Revenue Gap Analysis

Abstract Code

- User click ***Actual vs. Predicted Revenue for Speaker Units*** button form **Main Menu** form.
- Run the **Revenue Gap Analysis** task:
    - Find all products in Speaker category and retrieve ProductID.
    - For each Product in Speaker category:
        - Add all QuantitySold for this product as total number of units sold.
        - Find all Sales and OnSale for the product.
            - Calculate total number of units sold at discount by adding all QuantitySold if SaleDate equals to OnSaleDate.
            - Calculate total number of units sold at retail price by adding all QuantitySold if SaleDate not equal to OnSaleDate.
            - Calculate Actual Revenue by adding QuantitySold * Actual Selling Price for all sales (i.e., If the SaleDate equals to OnSaleDate, the actual selling price is DiscountPrice else Retail Price.)
            - Calculate the Predicted Revenue by adding Retail Price * Predicted QuantitySold for all sales (i.e., If the SaleDate equals to OnSaleDate, the Predicted Quantity Sold is 75% of QuantitySold else QuantitySold.)
            - Compute the difference between actual and predicted revenue.
    - Display ProductID, ProductName, Retail Price, Total number of units sold, Total number of units sold at discount, Total number of units sold at retail, Actual Revenue collected from all sales of the product, Predicted Revenue had the products never been put on sale and Difference between actual revenue and predicted revenue.
    - Limit only revenue difference greater than $5000 (positive or negative).
    - Sort by Predicted Revenue Difference in descending order.

Table of Contents

```sql
SELECT p.productID,
       product_name,
       retail_price,
       SUM(quantity_sold) AS total_units_sold,
       SUM(CASE WHEN discount_price IS NOT NULL THEN quantity_sold
                ELSE NULL END) AS total_units_sold_at_discount,
       SUM(CASE WHEN discount_price IS NULL THEN quantity_sold
                ELSE NULL END) AS total_units_sold_at_retail,
       ROUND(SUM(CASE WHEN discount_price IS NOT NULL THEN discount_price * quantity_sold
                      ELSE retail_price * quantity_sold END), 2) AS actual_revenue,
       ROUND(SUM(CASE WHEN discount_price IS NOT NULL THEN retail_price * quantity_sold * 0.75
                      ELSE retail_price * quantity_sold END), 2) AS predicted_revenue,
       ROUND(SUM(CASE WHEN discount_price IS NOT NULL THEN
                        discount_price * quantity_sold - retail_price * quantity_sold * 0.75
                    ELSE NULL END), 2) AS predicted_revenue_diff
FROM Product AS p
LEFT JOIN Sale AS t ON p.productID = t.productID
LEFT JOIN OnSale AS o ON p.productID = o.productID AND t.sale_date = o.onsale_date
WHERE p.productID IN ( SELECT productID FROM ProductCategory WHERE category_name = 'Speaker' )
GROUP BY 1
HAVING ABS(predicted_revenue_diff) > 5000
ORDER BY 9 DESC;
```

- When ready, user can select next action from Navigation Bar.

Yearly Store Revenue Filter by State

Abstract Code

- User click ***Store Revenue by Year by State*** button form **Main Menu** form.
- If user select a *state* ($state) from a drop-down box, data validate if state is in the database.

```
SELECT state FROM City WHERE state = '$state';
```

  - If state is in the database, then run the **View Store Revenue by Year** task:
    - Query for information about the selected state where $state is stored in the web session.
    - Find all City in the selected state using City.State = $state and retrieve CityID and CityName.
      - For each City in selected state, find all Store(s) located in that City and retrieve Store's StreetAddress and ZipCode.
      - For each Store, find all Sales and Product(s) related to that store, as well as OnSale related to each product.
    - For each Store and Sales Year:
      - Calculate Total Revenue by adding QuantitySold * Actual Selling Price for all sales (i.e., If the SaleDate equals to OnSaleDate; the Actual Selling Price is DiscountPrice else Retail Price.)
      - Calculate Total Revenue Sold at Retail by adding QuantitySold * Retail Price for all Sales if Sale Date is not equal to OnSaleDate for that product.
      - Calculate Total Revenue Sold at Discount by adding QuantitySold * DiscountPrice for all Sales if Sale Date equals to OnSaleDate for that product.
    - Display StoreNumber, StreetAddress, ZipCode, CityName, Sales Year, Total Revenue, Total Revenue Sold at Retail, and Total Revenue Sold at Discount.
    - Sort by Sales Year in descending order and then Total Revenue in descending order.

Table of Contents

```
SELECT s.store_number,
       street_address,
       zip_code,
       city_name,
       YEAR(sale_date) AS sales_year,
       ROUND(SUM(CASE WHEN discount_price IS NOT NULL THEN discount_price * quantity_sold
                      ELSE retail_price * quantity_sold END), 2) AS total_revenue,
       ROUND(SUM(CASE WHEN discount_price IS NULL THEN retail_price * quantity_sold
                      ELSE NULL END), 2) as total_revenue_at_retail,
       ROUND(SUM(CASE WHEN discount_price IS NOT NULL THEN discount_price * quantity_sold
                      ELSE NULL END), 2) AS total_revenue_at_discount
FROM Store AS s
    INNER JOIN City AS c ON s.locatedcityID = c.cityID
    LEFT JOIN Sale AS t ON s.store_number = t.store_number
    LEFT JOIN Product AS p ON t.productID = p.productID
    LEFT JOIN OnSale as o ON t.productID = o.productID AND t.sale_date = o.onsale_date
WHERE state = '$state'
GROUP BY 1, 5
ORDER BY 5 DESC, 6 DESC;
```

    - If state is not in the database, then display error message.
- When ready, user can select next action from Navigation Bar.

View Air Conditioners Sales on Groundhog Day

Abstract Code

- User click ***Air Conditioners on Groundhog Day*** button from **Main Menu** form.
- Run **View Air Conditioners Sales on Groundhog Day** task:
  - Find all Sales and retrieve SaleDate, ProductID, and QuantitySold.
    - Extract products that belong to Air Conditioning category.
  - For each Sales Year:
    - Adding all QuantitySold as Total number of items sold that year in Air Conditioning category.
    - Calculate the average number of units sold per day by dividing Total number of Air Conditioning items sold by 365 days.
    - Adding all QuantitySold on Groundhog Day (February 2nd) of that year as total number of units sold on Groundhog Day.
  - Display Sales Year, Total number of Air Conditoning units sold, Average Air Conditioning units sold per day, and the Total number of units sold on Groundhog Day of that year.
  - Sort by Sales Year in ascending order.

```
SELECT YEAR(sale_date) AS sales_year,
       SUM(quantity_sold) AS total_AC_units_sold,
       ROUND(IFNULL(SUM(quantity_sold) / 365, 0)) AS avg_AC_units_sold_per_day,
       SUM(CASE WHEN DATE_FORMAT(sale_date, '%m-%d') = '02-02' THEN quantity_sold
                ELSE NULL END) AS total_AC_units_sold_on_groundhog_day
FROM Sale
WHERE productID IN (SELECT productid FROM ProductCategory
                    WHERE category_name = 'Air Conditioning' )
GROUP BY 1
ORDER BY 1;
```

- When ready, user can select next action from Navigation Bar.

Table of Contents

Top State for each Category Filter by Year-Month

Abstract Code

- User click ***State with Highest Volume for each Category*** button from **Main Menu** form.
- If user select *Year* ($year) and *Month* ($month) from drop-down box, check if input fields are numeric value.
    - If data validation is successful, then run **Find top state in each Category** task:
        - Find all Sales in selected year and month as well as related Store and Product; Retrieve SaleDate, StoreNumber, ProductID, and QuantitySold.
            - Find all Category(s) for each product sold using ProductID and retireve CategoryName.
            - Find all City for each Store and retrieve State.
        - For each Category and State:
            - Add all QuantitySold for all stores in selected year and month as Total units sold.
            - Rank States from highest to lowest total units sold in each category.
            - Find the top State with highest quantity sold in each category.
        - Display CategoryName, Highest Sold State, and Total units sold.
        - Sort by CategoryName in ascending order.

```
SELECT category_name, state AS highest_sold_state, total_units_sold
FROM (
    SELECT
        category_name,
        state,
        SUM(quantity_sold) AS total_units_sold,
        RANK() OVER (PARTITION BY category_name
                     ORDER BY SUM(quantity_sold) DESC) AS rnk
    FROM Sale AS t
        LEFT JOIN ProductCategory AS pc ON t.productID = pc.productID
        LEFT JOIN Store AS s ON t.store_number = s.store_number
        LEFT JOIN City AS ct ON s.locatedcityID = ct.cityID
    WHERE YEAR(sale_date) = '$year' AND MONTH(sale_date) = '$month'
    GROUP BY 1,2
) AS temp
WHERE rnk = 1
ORDER BY 1;
```

    - Else then display error message.
- When ready, user can select next action from Navigation Bar.

Table of Contents

View Revenue by Population

Abstract Code

- User click ***Revenue per Population*** button from **Main Menu** form.
- Run **View Revenue per Population** task:
    - Find all Sales and related Store and Product; Retrieve SaleDate, StoreNumber, ProductID, and QuantitySold.
    - Find all OnSale for each Product and retrieve ProductID, OnSaleDate, and DiscountPrice.
    - Calculate Total Rvenue by Sales Year for each City by adding QuantitySold * Actual Selling Price (i.e., If the SaleDate equals to OnSaleDate, the Actual Selling Price is DiscountPrice else Retail Price.).
    - Derived CitySizeCategory from City.Population.
    - Calculate Average Revenue by CitySizeCategory for each Year.
    - Display Sales Year, the average revenue for small cities, the average revenue for medium cities, the average revenue for large cities and the average revenue for extra large cities.
    - Sort by Sales Year in ascending order.

Table of Contents

```
WITH city_revenue AS (
    SELECT YEAR(sale_date) AS sales_year,
            locatedcityID,
            SUM(CASE WHEN discount_price IS NOT NULL THEN discount_price * quantity_sold
                    ELSE retail_price * quantity_sold END) AS revenue
    FROM Sale AS t
       LEFT JOIN Store AS s ON t.store_number = s.store_number
       LEFT JOIN Product AS p ON t.productID = p.productID
       LEFT JOIN OnSale AS o ON t.productID = o.productID AND t.sale_date = o.onsale_date
    GROUP BY 1, 2
), add_category AS (
    SELECT r.*,
        (CASE WHEN population < 3700000 THEN 'Small'
            WHEN population >= 3700000 AND population < 6700000 THEN 'Medium'
            WHEN population >= 6700000 AND population < 9000000 THEN 'Large'
            WHEN population >= 9000000 THEN 'Extra Large'
            ELSE NULL END) AS city_size_category
    FROM city_revenue AS r
    JOIN City AS ct ON r.locatedcityID = ct.cityID
),revenue_by_category AS (
    SELECT
        sales_year,
        city_size_category,
        ROUND(SUM(revenue) / COUNT(DISTINCT locatedcityID), 2) AS avg_revenue
    FROM add_category
    GROUP BY 1,2
)
SELECT sales_year,
       MAX(CASE WHEN city_size_category = 'Small' THEN avg_revenue
                ELSE NULL END) AS avg_revenue_small_cities,
       MAX(CASE WHEN city_size_category = 'Medium' THEN avg_revenue
                ELSE NULL END) AS avg_revenue_medium_cities,
       MAX(CASE WHEN city_size_category = 'Large' THEN avg_revenue
                ELSE NULL END) AS avg_revenue_large_cities,
       MAX(CASE WHEN city_size_category = 'Extra Large' THEN avg_revenue
                ELSE NULL END) AS avg_revenue_extra_large_cities
FROM revenue_by_category
GROUP BY 1
ORDER BY 1;
```

- When ready, user can select next action from Navigation Bar.

Table of Contents

Membership Trend and Drill-down

Abstract Code

- User click on ***Membership Trend*** button from **Main Menu** form.
- Run **View Memberships Sold per Year** task:
  - Find all Membership and retrieve MemberID and SingupDate.
  - Count total number of memberships sold for each year.
  - Display year and total number of memberships sold for that year.
  - Sort by year in descending order.

```sql
SELECT
    YEAR(signup_date) AS signup_year,
    COUNT(DISTINCT memberID) AS total_memberships_sold
FROM Membership
GROUP BY 1
ORDER BY 1 DESC;
```

- When ready, user can click drill-down or select next action from Navigation Bar.

Table of Contents

- If user click ***drill-down*** button on a year, then run **Top and Bottom 25 Cities Drill-down** task:
    - Query for information about selected year where $year is stored in the web session.
    - Display selected year in the report header.
    - Find all Membership sold in the selected year using Membership.SignupDate.
        - Find all Stores from which each Membership is purchased as well as related City; Retrieve MemberID, CityName, and State.
        - Count total number of memberships sold for each City in the selected year using MemberID.
    - The first section presents the top 25 cities that sold the most memberships; Display CityName, State, and total number of memberships sold for that year.
        - Sort first by Total memberships sold in descending order then by CityName in ascending alphabetical order.

```
SELECT
    city_name,
    state,
    COUNT(DISTINCT memberID) AS total_memberships_sold
FROM Membership AS m
    LEFT JOIN Store AS s ON m.signup_store = s.store_number
    LEFT JOIN City AS ct ON s.locatedcityID = ct.cityID
WHERE YEAR(signup_date) = '$year'
GROUP BY 1,2
ORDER BY 3 DESC, 1 ASC
LIMIT 25;
```

    - The second section presents the bottom 25 cities that sold the least memberships; Display CityName, State, and total number of memberships sold for that year.
        - Sort first by Total memberships sold in ascending order then by CityName in ascending alphabetical order.

```
SELECT
    city_name,
    state,
    COUNT(DISTINCT memberID) AS total_memberships_sold
FROM Membership AS m
    LEFT JOIN Store AS s ON m.signup_store = s.store_number
    LEFT JOIN City AS ct ON s.locatedcityID = ct.cityID
WHERE YEAR(signup_date) = '$year'
GROUP BY 1,2
ORDER BY 3, 1
LIMIT 25;
```

    - When ready, user can click drill-down or select next action from Navigation Bar.

Table of Contents

- If user click ***drill-down*** button on a city, then run **City to Store level Drill-down** task:
  - Query for information about selected CityID ($cityid) where is stored in the web session.
  - Find all Memberships sold in selected Year, City, and Store; Retrieve MemberID, SignupStore, StreetAddress, ZipCode, CityName, and State.
    - Count total number of memberships sold for each store using MemberID.
  - Display selected year in the report header; Then display StoreNumber, StreetAddress, ZipCode, CityName, State, and total number of memberships sold during that year in this store.

```
SELECT
    store_number,
    street_address,
    zip_code,
    city_name,
    state,
    COUNT(DISTINCT memberID) AS total_memberships_sold
FROM Membership AS m
    LEFT JOIN Store AS s ON m.signup_store = s.store_number
    LEFT JOIN City AS ct ON s.locatedcityID = ct.cityID
WHERE YEAR(signup_date) = '$year' AND ct.cityID = '$cityid'
GROUP BY 1
ORDER BY 6 DESC;
```

  - When ready, user can select next action from Navigation Bar.

Table of Contents