Table of Contents

Main Menu

Abstract Code

- Display statistics: query for information about the count of Store, count of Store that offer food, count of Store offering childcare, count of Product, and count of distinct advertising Campaign.
    - Display the count of stores:

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

    - Display the count of Store that offer food:

```
SELECT COUNT(store_number) AS 'Count of Stores offering food'
FROM Store
WHERE has_snackbar = True or has_restaurant = True;
```

    - Display the count of Store offering childcare:

```
SELECT COUNT(store_number) AS 'Count of Stores offering Childcare'
FROM Store
WHERE fk_has_childcareID IS NOT NULL;
```

    - Display the count of Product:

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

    - Display the count of distinct advertising Campaign

```
SELECT COUNT(description) AS 'Count of distinct campaigns'
FROM Campaign;
```

- Show ***View Category Report***, ***View Actual versus Predicted Revenue for Couches and Sofas***, ***View Store Revenue by Year by State***, ***View Outdoor Furniture on Groundhog Day and Average Number of Units per Day***, ***View State with Highest Volume for each Category***, ***View Revenue by Population***, ***View Childcare Sales Volume***, ***View Restaurant Impact on Category Sales***, and ***View Advertising Campaign Analysis*** buttons.
- Upon:

    - Click ***View Category Report*** button- Jump to the **View Category Report** task.
    - Click ***View Actual versus Predicted Revenue for Couches and Sofas*** button- Jump to the **View Actual versus Predicted Revenue for Couches and Sofas** task.
    - Click ***View Store Revenue by Year by State*** button- Jump to the **View Store Revenue by Year by State** task.
    - Click ***View Outdoor Furniture on Groundhog Day and Average Number of Units per Day*** button- Jump to the **View Outdoor Furniture on Groundhog Day and Average Number of Units per Day** task.
    - Click ***View State with Highest Volume for each Category*** button- Jump to the **View State with Highest Volume for each Category** task.
    - Click ***View Revenue by Population*** button- Jump to the **View Revenue by Population** task.
    - Click ***View Childcare Sales Volume*** button- Jump to the **View Childcare Sales Volum**e task.
    - Click ***View Restaurant Impact on Category Sales*** button- Jump to the **View Restaurant Impact on Category Sales** task.
    - Click ***View Advertising Campaign Analysis*** button- Jump to the **View Advertising Campaign Analysis** task.
- Show maintenance buttons: ***View Holidays***, ***Add Holidays*** and ***Edit Population***.
- Upon:
    - Click ***View Holidays*** button- Jump to the **View Holidays** task.
    - Click ***Add Holidays*** button- Jump to the **Add Holidays** task.
    - Click ***Edit Population*** button- Jump to the **Edit Population** task.

View Holidays

Abstract Code

- User clicked on ***View Holidays*** button
    - Display all the existing holidays for any dates***.***

```
SELECT fk_date      AS date,holiday_name AS 'Holiday Name'
FROM   holiday;
```

- Click ***Finish***, back to the **Main Menu**

Add Holidays

Abstract Code

- User clicked on ***Add Holidays*** button
- Generate a drop down menu to show all the dates for user to select

```
SELECT date
FROM Date
ORDER BY date DESC;
```

- User types in a new holiday name and selects the date. Then clicks the ***Add new holiday*** button to run the **Add the new holidays** subtask.

```
// read $Date, $HolidayName
```

```
INSERT INTO Holiday (fk_date, holiday_name) VALUES
($Date, $HolidayName);
```

    - If there's any error or the input is invalid (not a date), then show an error message explaining.
- Click ***Finish,*** back to the **Main Menu**
- If user click ***Cancel***, back to the **Main Menu**

Edit Population

Abstract Code

- User clicked on ***Edit Population*** button
- Run the **Display Drop-Down Menu** task

```
SELECT DISTINCT name, state, population
FROM City
ORDER BY state ASC;
```

    - Display drop-down menu for the user to choose CITY.city_address (state and city name), and a button called ***Cancel***.
    - Record the input from user, display chosen CITY.city_address and CITY.population, and buttons called ***Edit Population*** and ***Cancel***
    - If user click ***Edit Population***, go on with the next task
    - If user click ***Cancel***, back to the **Main Menu** Form
- Run the **Edit Population** task
    - Display chosen CITY.city_address and CITY.population, a text box, and buttons called ***Update*** and ***Cancel***.

- The user enters a number *new_city_population.*
- If the *new_city_population* is valid (means it would be an integer), and the user clicks ***Update***, it'll update in the database and display CITY.name and the updated CITY.population, along with a success message and a button ***Finish***.

```
UPDATE City
SET population=$population
WHERE name = $city_name AND state = $city_state;
```

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

- Click ***Finish,*** back to the **Main Menu**
- If the *new_city_population* is an invalid value, or the user only clicks ***Update*** without putting in any value, then it'll display instructions, along with an error message and a button ***Cancel***.
- If the user clicks ***Cancel***, it'll go back to the **Main Menu** Form.

- When ready, the user selects the next action from choices in the **Main Menu** Form.

View Category Report

Abstract Code

- User click on ***View Category Report*** button on **Main Menu** form
- Run the **View Category Report** task

```
SELECT Category.name,
       Count(Product.pid),
       Max(Product.retail_price),
       Min(Product.retail_price),
       Avg(Product.retail_price)
FROM Category
  LEFT JOIN Assigned ON Category.name = Assigned.fk_category_name
  LEFT JOIN Product ON Product.pid = Assigned.fk_product_pid
GROUP BY Category.name
ORDER BY Category.name;
```

  - For each CATEGORY
    - Get CATEGORY.name
    - Count the total number of PRODUCTs in the CATEGORY through CATEGORY.name and "assigned" relationship.

        - Get the maximum, minimum, and the average retail prices by aggregating the PRODUCT.retail_price under the CATEGORY through the “assigned” relationship.
    - Sort the result by CATEGORY.name ascending.
    - Generate the report and display to the client.
- When user is ready, Click ***Finish***, back to the **Main Menu**

Actual versus Predicted Revenue for Couches and Sofas

Abstract Code

- User clicked on ***Actual versus Predicted Revenue for Couches and Sofas*** button on **Main Menu** form
- Run the **Actual versus Predicted Revenue for Couches and Sofas** task:
    - Run the **Select all "sold" records in category couches and sofa** subtask:
        - Find the list of PRODUCT.PID which category is Couches and Sofas in CATEGORY entities through "assigned" relationship
    - Once get the list of sold records, run the **Find revenue by “sold” records** subtask:
        - Calculate the total sold quantity by using SOLD entity
        - Calculate the sold at retail price quantity by using SOLD entity and "discount on" relationship
        - Calculate the sold at discount price quantity by using SOLD entity and "discount on" relationship
        - Calculate the actual revenue of units sold at discount by using the "discounted price" through "discount on" relationship and number of units sold at discount, save as actual_discount_revenue
        - Calculate the actual revenue of units sold at retail price by using the "retail price" and number of units sold at retail price, save as retail_revenue
        - Calculate the predicted revenue if units never discounted, using "retail price" and applying 0.75 factor to the number of units sold at discount, save as predict_discount_revenue
        - Calculate the actual - predict revenue difference: (actual_discount_revenue - predict_discount_revenue), save as actual_predict_difference.
        - get the list of sold records where actual_predict_difference > 5000 or actual_predict_difference < -5000
    - From the above sold records list, display the above calculated value, also display the PID, name, retail price.
- Sort the report by difference between the actual revenue and the predicted revenue in descending order
- Generate the report and display to the client.

```
SELECT Aggregated.pid AS product_id,
  Aggregated.retail_price,
  SUM(Aggregated.sold_total) AS total_sold_units,
  SUM(Aggregated.sold_discounted) AS discount_sold_units,
  SUM(Aggregated.sold_total) - SUM(Aggregated.sold_discounted) AS retail_sold_units,
  SUM(Aggregated.predicted_revenue) AS total_predict_revenue,
  SUM(Aggregated.actual_revenue) AS total_actual_revenue,
  SUM(Aggregated.actual_revenue) - SUM(Aggregated.predicted_revenue) revenue_diff
FROM (
    SELECT Sold.fk_product_pid AS pid,
      Product.name AS name,
      Product.retail_price AS retail_price,
      DiscountOn.discounted_price AS discounted_price,
      SUM(Sold.quantity) AS sold_total,
      IFNULL(
        DiscountOn.discounted_price * 0 + SUM(Sold.quantity),
        0
      ) AS sold_discounted,
      IFNULL(
        DiscountOn.discounted_price * SUM(Sold.quantity),
        SUM(Sold.quantity) * Product.retail_price
      ) AS actual_revenue,
      IFNULL(
        DiscountOn.discounted_price * 0 + SUM(Sold.quantity) * 0.75 * Product.retail_price,
        SUM(Sold.quantity) * Product.retail_price
      ) AS predicted_revenue
    FROM Sold
      LEFT JOIN DiscountOn ON Sold.fk_date = DiscountOn.fk_date
      AND Sold.fk_date = DiscountOn.fk_date
      JOIN Product ON Sold.fk_product_pid = Product.pid
      JOIN Assigned ON Assigned.fk_product_pid = Product.pid
    WHERE Assigned.fk_category_name = 'Couches and Sofas'
    GROUP BY Assigned.fk_category_name,
             Sold.fk_product_pid,
             Sold.quantity,
             DiscountOn.discounted_price
  ) AS Aggregated
GROUP BY Aggregated.pid
HAVING revenue_diff > 5000
  OR revenue_diff < -5000
ORDER BY revenue_diff DESC;
```

- When user is ready, Click ***Finish***, back to the **Main Menu**

View Store Revenue by Year by State

Abstract Code

- User click on ***View Store Revenue by Year by State*** button on **Main Menu**
- Get All the all the states in drop-down box using CITY entity

```
SELECT DISTINCT state
FROM City;
```

- User select the CITY.state in drop-down box on **View Store Revenue by Year by State** form
- Run the **View Store Revenue by Year by State** task:
    - Find the list of sold products which store is in the user selected state through "sold" and "in" relationships in CITY, STORE, and PRODUCT entities
    - Calculate the actual revenue of units sold at discount by using the "discounted price" through "discount on" relationship and number of units sold at discount, save as actual_discount_revenue
    - Calculate the actual revenue of units sold at retail price by using the "retail price" and number of units sold at retail price, save as retail_revenue
    - Calculate the revenue (actual_discount_revenue + retail_revenue)
    - Display the following information in the report: city name, store ID, store street address, sales year, total revenue of each year
- Sort the report by year in ascending order and then by revenue in descending order
- Generate the report and display to the client

```
SELECT store_number,
       street_address,
       city_name,
       sales_year,
       SUM(actual_revenue) AS total_revenue
FROM (
    SELECT Sold.fk_product_pid AS product_pid,
           Store.store_number,
           Store.street_address,
           City.name AS city_name,
           YEAR(Sold.fk_date) AS sales_year,
           SUM(Sold.quantity) AS sold_total,
           IFNULL(
             DiscountOn.discounted_price * SUM(Sold.quantity),
             SUM(Sold.quantity) * Product.retail_price
           ) AS actual_revenue
    FROM City
      JOIN Store ON City.name = Store.fk_in_city_name
      AND City.state = Store.fk_in_city_state
      JOIN Sold ON Store.store_number = Sold.fk_store_number
```

```
      LEFT JOIN DiscountOn ON Sold.fk_date = DiscountOn.fk_date
      AND Sold.fk_date = DiscountOn.fk_date
      JOIN Product ON Sold.fk_product_pid = Product.pid
    WHERE City.state = $state
    GROUP BY Sold.fk_product_pid,
      Store.store_number,
      city_name,
      DiscountOn.discounted_price,
      Sold.quantity,
      sales_year
  ) AS Aggregated
GROUP BY store_number,
         street_address,
         city_name,
         sales_year
ORDER BY sales_year ASC,
         total_revenue DESC;
```

- When user is ready, Click ***Finish***, back to the **Main Menu**

View Outdoor furniture sales on Groundhog Day and average sale per day

Abstract code

- Users clicked on ***View Outdoor furniture sales on Groundhog Day and average sale per day*** button from **Main Menu.**
- Run the **View Outdoor furniture sales on Groundhog Day and average sale per day** task: Query for information about the outdoor furniture product and their average sale per day.
    - Find the total “Sold”.quantity of PRODUCT where CATEGORY.name = “Outdoor Furniture”.
    - Calculate the average number of units sold per day by:
      Average sale per day = total annual Sold quantity/ 365.
    - Find the “Sold”.quantity of PRODUCT where CATEGORY.name = OutdoorFurniture on February 2 (Groundhog Day) of that year.
    - Display the year, the total sale of PRODUCT where CATEGORY.name = “Outdoor Furniture” in the specific year, the average number of units sold per day in the year, and sale PRODUCT where CATEGORY.name = “Outdoor Furniture” on Groundhog Day of that year.

```
WITH sold_prop AS (
SELECT fk_product_pid,
       fk_date,
       quantity,
       Year(fk_date)  AS date_year,
       Month(fk_date) AS date_month,
```

```
       Day(fk_date)    AS date_day
FROM   Sold
)
SELECT T.date_year,
       T.annual_quantity,
       T.avr_quantity_by_year,
       G.quantity AS quantity_at_groundhog_day
FROM   (SELECT sold_prop.date_year           AS date_year,
               Sum(sold_prop.quantity)       AS annual_quantity,
               Sum(sold_prop.quantity) / 365 AS avr_quantity_by_year
        FROM   sold_prop,
               Product AS product,
               Assigned AS assigned
        WHERE  sold_prop.fk_product_pid = product.pid
               AND product.pid = assigned.fk_product_pid
               AND assigned.fk_category_name = 'Outdoor Furniture'
        GROUP  BY date_year) AS T
       INNER JOIN
       (SELECT sold_prop.date_year     AS date_year,
               Sum(sold_prop.quantity) AS quantity
        FROM   sold_prop,
               Product AS product,
               Assigned AS assigned
        WHERE  sold_prop.fk_product_pid = product.pid
               AND product.pid = assigned.fk_product_pid
               AND assigned.fk_category_name = 'Outdoor Furniture'
               AND sold_prop.date_month = 2
               AND sold_prop.date_day = 2
        GROUP  BY date_year) AS G
        ON T.date_year = G.date_year
GROUP  BY T.date_year,
          quantity_at_groundhog_day
ORDER  BY T.date_year ASC;
```

- When user is ready, click ***Finish***, back to the **Main Menu**

View State with Highest Volume in each Category

Abstract code

- A user clicked on the ***View state with highest volume in each category*** button from the **Main Menu.**
- A User selected a year and a month from the drop-down box.
  `//populate dropdown`

```
SELECT YEAR(Date.date), MONTH(Date.date)
FROM Date;
```

- Run the **View state with highest volume in each category task*:***
    - Read the selected year (*$year*) and month (*$month*)

```
//read $year, $month
```

```
WITH Saleperstate AS
(
    SELECT   Assigned.fk_category_name AS category_name,
             Store.fk_in_city_state    AS state,
             Sum(Sold.quantity)        AS sold_quantity
    FROM     Sold,
             Assigned,
             Store,
             Product
    WHERE    Sold.fk_store_number = Store.store_number
             AND Assigned.fk_product_pid = Product.pid
             AND Sold.fk_product_pid = Product.pid
             AND Year(Sold.fk_date)  = $year
             AND Month(Sold.fk_date) = $month
    GROUP BY Category_name, State
    ORDER BY Sold_quantity),
Maxperstate AS
(
    SELECT   Saleperstate.category_name      AS category_name,
             max(Saleperstate.sold_quantity) AS max_quantity
    FROM     Saleperstate
    GROUP BY category_name
)
SELECT Saleperstate.category_name,
       Saleperstate.state,
       Maxperstate.max_quantity
FROM   Saleperstate,
       Maxperstate
WHERE  Saleperstate.category_name = Maxperstate.category_name
AND    Saleperstate.sold_quantity = Maxperstate.max_quantity
ORDER BY Saleperstate.category_name ASC;
```

    - Find category.name of each PRODUCT through the “Assigned” relationship.For each category:
        - Find the total “Sold”.quantity of PRODUCT during the time selected by the user (*$year, $month*) in each state through the “Sold” relationship.
        - Find the maximum “Sold”.quantity in the category.
    - Display CATEGORY.name, CITY.state which has the maximum quantity value in that category and “Sold”.quantity of the state.
- Group the results by CATEGORY.name.
- When user is ready, click ***Finish***, back to the **Main Menu**

View Revenue by Population

Abstract Code

- User click on ***View Revenue by Population*** button on **Main Menu**
- Run the **View Revenue by Population** task

```
WITH rawdata
     AS (SELECT city.population                                  AS
population,
                Year(sold.fk_date)                               AS year,
                Ifnull(discounton.discounted_price *
Sum(sold.quantity),
                Sum(sold.quantity) * product.retail_price) AS
segregated_revenue
         FROM   sold AS sold
                JOIN store AS store
                  ON sold.fk_store_number = store.store_number
                JOIN city AS city
                  ON store.fk_in_city_name = city.NAME
                     AND store.fk_in_city_state = city.state
                JOIN product AS product
                  ON product.pid = sold.fk_product_pid
                LEFT JOIN discounton AS discounton
                       ON product.pid = discounton.fk_product_pid
                          AND sold.fk_date = discounton.fk_date
         GROUP  BY Year(sold.fk_date),
                   city.population,
                   discounton.discounted_price,
```

```
                product.retail_price),
    tosum
    AS (SELECT year,
               Sum(segregated_revenue) AS small,
               0                       AS medium,
               0                       AS large,
               0                       AS extra_large
        FROM   rawdata
        WHERE  rawdata.population < 3700000
        GROUP  BY year
        UNION ALL
        SELECT year,
               0                       AS small,
               Sum(segregated_revenue) AS medium,
               0                       AS large,
               0                       AS extra_large
        FROM   rawdata
        WHERE  rawdata.population >= 3700000
               AND rawdata.population < 6700000
        GROUP  BY year
        UNION ALL
        SELECT year,
               0                       AS small,
               0                       AS medium,
               Sum(segregated_revenue) AS large,
               0                       AS extra_large
        FROM   rawdata
        WHERE  rawdata.population >= 6700000
               AND rawdata.population < 9000000
        GROUP  BY year
        UNION ALL
        SELECT year,
               0                       AS small,
               0                       AS medium,
               0                       AS large,
               Sum(segregated_revenue) AS extra_large
        FROM   rawdata
```

```
          WHERE  rawdata.population >= 9000000
          GROUP  BY year)
SELECT year,
       Sum(small),
       Sum(medium),
       Sum(large),
       Sum(extra_large)
FROM   tosum
GROUP  BY year
ORDER  BY year ASC;
```

- Calculate “Revenue” using different sold prices multiply the “Sold”.quantity by year
- Group “Revenue” based on the population size
  - The categories for population size are: Small (CITY.population <3,700,000), Medium (CITY.population >=3,700,000 and <6,700,000), Large (CITY.population >=6,700,000 and <9,000,000) and Extra Large (CITY.population >=9,000,000)
- Generate the report with population sizes (small/medium/large/extra_large) as columns, “Revenue” as content, and each year as rows
- Sort the report by “Revenue” and “City Size” in ascending order
- Display the report to user, with a button ***Finish***
- When user is ready, click ***Finish***, back to the **Main Menu**

View Childcare Sales Volume

Abstract Code

- User click on ***View Childcare Sales Volume*** button on **Main Menu**
- Run the **View Childcare Sales Volume** task

```
WITH rawdata
     AS (SELECT Month(sold.fk_date)  AS month,
                childcare.time_limit AS time_limit,
                SUM(sold.quantity)   AS grouped_sales
         FROM   Sold AS sold
                JOIN Store AS store
                  ON store.store_number = sold.fk_store_number
                LEFT JOIN ChildCare AS childcare
```

```
                        ON childcare.childcareid =
store.fk_has_childcareid
         WHERE  sold.fk_date >= Date_sub(Now(), interval 1 month)
                AND sold.fk_date <= Now()
         GROUP  BY Month(sold.fk_date),
                   store.fk_has_childcareid),
     tosum
     AS (SELECT month,
                SUM(grouped_sales) AS no_childcare,
                0                  AS min20,
                0                  AS min40,
                0                  AS min50
         FROM   rawdata
         WHERE  time_limit IS NULL
         GROUP  BY month
         UNION ALL
         SELECT month,
                0                  AS no_childcare,
                SUM(grouped_sales) AS min20,
                0                  AS min40,
                0                  AS min50
         FROM   rawdata
         WHERE  time_limit = 20
         GROUP  BY month
         UNION ALL
         SELECT month,
                0                  AS no_childcare,
                0                  AS min20,
                SUM(grouped_sales) AS min40,
                0                  AS min50
         FROM   rawdata
         WHERE  time_limit = 40
         GROUP  BY month
         UNION ALL
         SELECT month,
                0                  AS no_childcare,
                0                  AS min20,
```

```
                0                   AS min40,
                SUM(grouped_sales)  AS min50
         FROM   rawdata
         WHERE  time_limit = 50
         GROUP  BY month)
SELECT month,
       SUM(no_childcare),
       SUM(min20),
       SUM(min40),
       SUM(min50)
FROM   tosum
GROUP  BY month;
```

- Calculate sales volume using different sold prices multiply the “Sold”.quantity by month
- Group sales volume by time_limit
  - The categories for time_limit are: no_childcare, min20, min40, min50
- Generate the report with time_limit as columns, “sales volume as content, and each month as rows
- Sort the report by sales volume and time_limit in ascending order
- Display the report to user, with a button ***Finish***
- When user is ready, click ***Finish***, back to the **Main Menu**

View Restaurant Impact on Category Sales

Abstract Code

- A User clicked on ***View Restaurant Impact on Category Sales*** button on **Main Menu** form.
- Run the **View Restaurant Impact on Category Sales** task

```
SELECT Assigned.fk_category_name AS "Category",
       CASE
         WHEN Store.has_restaurant THEN 'Restaurant'
         ELSE 'Non-Restaurant'
       END                        AS "Store Type",
       Sum(Sold.quantity)         AS "Quantity Sold"
FROM   Store,
       Sold,
       Product,
       Assigned
```

```
WHERE  Product.pid = Sold.fk_product_pid
       AND Sold.fk_store_number = Store.store_number
       AND Assigned.fk_product_pid = Product.pid
GROUP  BY Assigned.fk_category_name,
          Store.has_restaurant
ORDER  BY Assigned.fk_category_name,
          Store.has_restaurant;
```

- Through the SOLD relationship, we find all tuples of STORE.has_restaurant, PRODUCT, and SOLD.quantity associated with them. It must only contain PRODUCTs that have been sold. For each of the tuples, we sum up the quantity and sort the result first by the category name and then the store type.
- Generate the report and display to the client.
- When ready, user clicks ***Finish*** button to choose next action from choices in **Main Menu**

View Advertising Campaign Analysis

Abstract Code

- User clicked on ***View advertising Campaign Analysis*** button on **Main Menu** form
- User select the “all store” in drop-down box on **View advertising Campaign Analysis** form.

```
SELECT store_number
FROM   Store;
```

- Run the **View advertising Campaign Analysis:**

```
 With campaign_difference AS (
  SELECT Product.pid, Product.name,
  SUM(
    CASE
    When ActiveOn.fk_campaign_description IS NOT NULL THEN
Sold.quantity
    ELSE Null
    END
  ) AS Sold_during_campaign,
  SUM(
    CASE
    When ActiveOn.fk_campaign_description IS NULL THEN
Sold.quantity
    ELSE Null
    END
```

```
  ) AS Sold_outside_campaign
FROM Sold
INNER JOIN Product ON Sold.fk_product_pid = Product.pid
INNER JOIN DiscountOn ON DiscountOn.fk_product_pid =
Sold.fk_product_pid
AND Sold.fk_date = DiscountOn.fk_date
LEFT JOIN ActiveOn ON Sold.fk_date = ActiveOn.fk_date
GROUP BY 1, 2
)

SELECT top.pid, top.name,
top.Sold_during_campaign,
top.Sold_outside_campaign,
top.difference
FROM (
SELECT campaign_difference.pid, campaign_difference.name,
campaign_difference.Sold_during_campaign,
campaign_difference.Sold_outside_campaign,
(campaign_difference.Sold_during_campaign -
campaign_difference.Sold_outside_campaign) AS difference
FROM campaign_difference
ORDER BY difference DESC
LIMIT 10) as top

UNION ALL

SELECT bottom.pid, bottom.name,
bottom.Sold_during_campaign,
bottom.Sold_outside_campaign,
bottom.difference
FROM (
SELECT campaign_difference.pid, campaign_difference.name,
campaign_difference.Sold_during_campaign,
campaign_difference.Sold_outside_campaign,
(campaign_difference.Sold_during_campaign -
campaign_difference.Sold_outside_campaign) AS difference
FROM campaign_difference
ORDER BY difference ASC
LIMIT 10) as bottom;
```

- Find the product sold information from all stores including PRODUCT.PID, PRODUCT.name, PRODUCT.quantity and sold date.
- Find the campaign information including CAMPAIGN.description and Date.date.

    - Find the discount information including PRODUCT.PID, PRODUCT.name, discount Date.date and PRODUCT.discounted_price.
    - Combine all of the above information together based on the PID and date.
    - Perform mathematical manipulation to calculate the difference.
    - Sort the results by difference in descending (highest to lowest) order" before display
    - Display the top 10 and the bottom 10.
- When ready, user clicks ***Finish***, back to the **Main Menu**