Table of Contents

Main Menu

Abstract Code

- Show "**Calculate Statistics**" button.
- Show "**View/Edit Holidays**" button.
- Show "**View/Update City Profile**" button.
- Show the buttons of "***Query all Category***", "***Actual versus Predicted Revenue for Couches and Sofas***", "***Store Revenue by Year by State***", "***Analyze Outdoor Furniture on Groundhog Day***", "***State with Highest Volume for each Category***", "***Revenue by Population***", "***Childcare Sales Volume***", "***Analyze Sales by Restaurant Existence***", and "***Count Sold Products during and outside Advertising Campaign***".
- Upon:
    - Click ***Calculate Statistics button*** **-** jump to the **Calculate Statistics** task.
    - Click **View/Edit Holidays** ***button*** **-** jump to the **View/Edit Holidays** task.
    - Click **View/Update City Profile** ***button*** **-** jump to the **View/Update City Profile** task.
    - Click ***Query all Category button*** **-** jump to the **Query all Category** task.
    - Click ***Actual versus Predicted Revenue for Couches and Sofas button*** **-** jump to the **Actual versus Predicted Revenue for Couches and Sofas** task.
    - Click ***Store Revenue by Year by State button*** **-** jump to the **Store Revenue by Year by State** task.
    - Click ***Analyze Outdoor Furniture on Groundhog Day button*** **-** jump to the **Analyze Outdoor Furniture on Groundhog Day** task.
    - Click ***State with Highest Volume for each Category button*** **-** jump to the **State with Highest Volume for each Category** task.
    - Click ***Revenue by Population button*** **-** jump to the **Revenue by Population** task.
    - Click ***Childcare Sales Volume button*** **-** jump to the **Childcare Sales Volume** task.
    - Click ***Analyze Sales by Restaurant Existence button*** **-** jump to the **Analyze Sales by Restaurant Existence** task.
    - Click ***Count Sold Products during and outside Advertising Campaign button*** **-** jump to the **Count Sold Products during and outside Advertising Campaign** task.
- Exit "**Main Menu**" when closing this application.

Calculate Statistics

Abstract Code

- Run the **Calculate Statistics**: Query for the information forms about Stores, Products, and Advertising Campaign from the database.
    - Find all the stores in the form; sum the number of stores; display the sum of stores.

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

    - Find the stores using offering childcare in the Store form; sum the number of these stores; display the sum of the stores offering childcare.

```
SELECT COUNT (store_number) AS 'COUNT OF Stores WITH Childcare'
FROM Store
WHERE Store.fk_store_childcareID IS NOT NULL;
```

    - Find the stores using offering restaurant or snackbar in the Store form; sum the number of these stores; display the sum of the stores offering food.

```
SELECT COUNT (store_number) AS 'COUNT OF Stores WITH Food'
FROM Store
WHERE store.restaurant =TRUE OR store.snack_bar = TRUE;
```

    - Find all the products in the Product form; sum the number of the products; display the sum of the products.

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

    - Find all advertising campaign in the AdCampaign form; sum the number of advertising campaign; display this sum.

```
SELECT COUNT (campaignID) AS 'Count of Campaigns' FROM AdCampaign;
```

View/Add Holidays

Abstract Code

- Show "***View Holidays***" and "***Edit Holidays***" tabs
- Click "***View Holidays***" button:
    - Display all holiday names and dates from Holiday
    - User can select a range of dates and the specific holidays are displayed

```
SELECT fk_holiday_date AS holidayDate,
      name AS holidayName
FROM Holiday
ORDER BY holidayDate DESC;
```

- Click "***Add Holidays***" button :
    - User inputs a *holiday name* ($holiday_name) and *date* ($fk_holiday_date) and click enter.

```
CAST('$fk_holiday_date' AS DATE);
```

    - Issue an appropriate error message if any entry is invalid.

```
INSERT INTO Holiday(fk_holiday_date,name)
VALUES ('$fk_holiday_date', '$holiday_name');
```

      If the return shows success, display "Holiday successfully added". Otherwise, display an appropriate error message.
- When ready, the user can go back to the **Main Menu**.

View/Update City Population

Abstract Code:

- User clicked on ***View Population*** on **Main menu**
- Display View Population page
    - User pick a *state* ($State) and *city name* ($CityName) in dropdown list
    - Run the View Population task: query the population in City where *city name* and *state* equals to the selection by the user

```
SELECT population FROM City WHERE name=$CityName AND state=$State;
```

    - Display population on **Main Menu**
- User clicked on ***Edit Population*** on ***View Population***
    - User select a *state* ($State) and *city name* ($CityName) in the dropdown list
    - User input an integer for *NewPopulation* ($Population)
    - If data validation is successful for *NewPopulation* then:
    - Run the Edit Population task: update *NewPopulation* into City.

```
UPDATE City SET population=$Population WHERE name=$CityName AND
state=$State;
```

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

Report 1: Query All Categories

Abstract Code:

- User clicks on ***Query All Categories*** button from **Main Menu:**
- Run the **Query All Categories** task: calculate statistics information about all the categories.
    - List all Categories
    - For each Category, find:
        - if it has Products (display results as NULL if not);
        - the Category name;
        - total number of Products in it;
        - the minimum regular Product.RetailPrice of all Products in it;
        - the average regular Product.RetailPrice of all Products in it: record the sum of all regular Product.RetailPrices, and divide the sum by the number of all Product.regularPrices.
        - the maximum regular Product.RetailPrice of all Products in it;
    - Sort and display the Category statistics results based on Category name in ascending order
- When ready, user selects next action from choices in **Main Menu**.

```
SELECT
    Category.name,
    COUNT(*),
    MIN(Product.retail_price) AS minRetailPrice,
    AVG(Product.retail_price) AS avgRetailPrice,
    MAX(Product.retail_price) AS maxRetailPrice
FROM Category
LEFT JOIN BelongsTo ON Category.name =
BelongsTo.fk_belongsto_category_name
LEFT JOIN Product ON Product.PID = BelongsTo.fk_belongsto_PID
GROUP BY Category.name
ORDER BY Catogory.name ASC;
```

Report 2: Actual Versus Predicted Revenue for Couches and Sofas

- User clicked on ***Actual Versus Predicted Revenue for Couches and Sofas Report*** button from **Main Menu**
- For each PID in Product
    - Save *RetailPrice*
    - If Category equals to Couches and Sofas
        - Query Sale, for each sale:
        - Save *Quantity*

        - If Date has Discount:
            - Save *DiscountPrice*
            - Calculate ActualRevenue using *DiscountPrice* and *Quantity*
            - Calculate *ModifedQuantity* using *Quantity*
            - Calculate PredictedRevenue using *ModifedQuantity* and *RetailPrice*
        - Else:
            - Calculate *ActualRevenue* using *RetailPrice* and *Quantity*
            - Calculate *PredictedRevenue* using *RetailPrice* and *Quantity*
- If RevenueDifference is greater than $5000 or smaller than -$5000, Show Report
- Return to **Main Menu**

Abstract Code:

```
SELECT
        b.PID,
        b.name,
        b.retail_price,
        b.dicountPriceQuantity + b.retailPriceQuantity AS total_units_sold,
        b.dicountPriceQuantity AS total_sold_discount_price,
        b.retailPriceQuantity AS total_sold_retail_price,
        SUM(b.actualRevenue) AS actualRevenue,
        SUM(b.predictedRevenue) AS predictedRevenue,
        SUM(b.actualRevenue) - SUM(b.predictedRevenue) AS difference
FROM
(SELECT
        a.PID,
        a.name,
        a.retail_price,
        CASE WHEN a.discounted = 1 THEN SUM(a.quantity) ELSE NULL END AS
discountPriceQuantity,
        CASE WHEN a.discounted = 0 THEN SUM(a.quantity) ELSE NULL END AS
retailPriceQuantity,
        SUM(a.actualRevenue) AS actual_revenue,
        SUM(a.predictedRevenue) AS predicted_revenue
FROM
(SELECT
        Sale.fk_product_PID,
        Sale.quantity,
        Product.name,
        Product.retail_price,
        Discount.discount_price,
        CASE WHEN Discount.discount_price IS NOT NULL THEN Discount.discount_price *
Sale.quantity ELSE Product.retail_price * Sale.quantity END AS actual_revenue,
```

```
        CASE WHEN Discount.discount_price IS NOT NULL THEN Product.retail_price*
0.75*Sale.quantity ELSE Product.retail_price * Sale.quantity END AS predicted_revenue,
        CASE WHEN Discount.discount_price IS NOT NULL THEN 1 ELSE 0 END AS
discounted
FROM Sale
        JOIN Product
        ON Sale.fk_product_PID = Product.PID
        LEFT JOIN Discount
        ON Sale.fk_product_PID = Discount.fk_product_PID AND
Sale.fk_sale_date=Discount.fk_discount_date
        JOIN BelongsTo ON
        Product.PID = BelongsTo.fk_belongsto_PID
        WHERE BelongsTo.CategoryName = 'Couches and Sofas') AS a
GROUP BY PID, name, retail_price, discounted) AS b
GROUP BY PID, name, retail_price
HAVING difference<-5000 OR difference>5000
ORDER BY difference DESC;
```

Report 3: Store Revenue by Year by State

Abstract Code:

- User clicked on ***Store Revenue by Year by State Report*** button on **Main Menu**
- User select *State*($State) in dropdown box
- For each city in City:
    - If *city.state* == *State* selected by user:
        - Save city.id
        - Query store in Store
        - For each store:
            - Query Sale, save *Quantity*
            - Query Product, save *RetailPrice*
            - Query Date, save *Year, Month, Day*
                - ➢ If date has Discount on *Year, Month, Day*:
                    - ✓ Save discount price
                    - ✓ Revenue += discount price * quantity
                - ➢ Else:
                    - ✓ Revenue += retail price * quantity
- Sort results by year in ascending order
- Sort results by Revenue in descending order if user click “Revenue” on report

```
SELECT 'year',
        state,
```

```
            city_name,
            store_number,
            store_address,
            SUM(actual_revenue) AS total_revenue
FROM
  (SELECT City.state AS state,
         City.name AS city_name,
         Store.store_number AS store_number,
         Store.street_address AS store_address,
         CASE
            WHEN Discount.discount_price IS NOT NULL THEN Discount.discount_price *
Sale.quantity
            ELSE Product.retail_price * Sale.quantity
         END AS actual_revenue,
         YEAR(Sale.fk_sale_date) AS 'year'
   FROM Store
   JOIN City ON Store.fk_cityID = City.cityID
   JOIN Sale ON Sale.store_number = Store.store_number
   JOIN Product ON Product.PID = Sale.fk_sale_PID
   LEFT JOIN Discount ON Sale.fk_sale_PID = Discount.fk_discount_PID
   AND Sale.fk_sale_date = Discount.fk_discount_date)
WHERE state='$State'
GROUP BY 'year', state,
              city_name,
              store_number,
              store_address
ORDER BY 'year' ASC
ORDER BY total_revenue DESC;
```

Report 4: Analyze Outdoor Furniture Sales on Groundhog Day

Abstract Code

- User clicks on ***Query Outdoor Furniture on Groundhog Day*** from **Main Menu**
- Run the **Query Outdoor Furniture on Groundhog Day** task: calculate statistics information about outdoor furniture sold per year
    - List all Date.year that have Sales
    - For each Date.year, find/return:
        - the Date.year;
        - in the outdoor furniture Category, the total number of Products sold that year by looking through Sale; save the result as sum_sale;

        - the average number of units sold per day: divide sum_sale by the number of Date.days for the Date.year.
        - the total number of units sold on Groundhog Day of that year;
    - Sort the results based on the year in ascending order, and display if the average number of units sold per day is larger than the total number of units sold on Groundhog Day of that year.
- When ready, user selects next action from choices in **Main Menu**.

```
WITH Furniture_sale AS
  (SELECT Sale.fk_sale_date AS sale_date,
          Sale.quantity AS sale_quantity
   FROM Sale,
        Product,
        BelongsTo
   WHERE Sale.fk_sale_PID = Product.PID
     AND BelongsTo.fk_belongsto_PID = Product.PID
     AND BelongsTo.fk_belongsto_category_name = 'outdoor furniture' )
SELECT YEAR(sale_date) AS YEAR,
       SUM(Sale_quantity) AS total_furniture_sale,
       SUM(sale_quantity)/365 AS avg_furniture_sale,
       total_gh_furniture_sale,
       CASE
          WHEN avg_furniture_sale < total_gh_furniture_sale THEN 'YES'
          ELSE 'NO'
       END AS is_Groundhog_higher
FROM Furniture_sale
JOIN
  (SELECT YEAR(sale_date),
          SUM(sale_quantity) AS total_gh_furniture_sale
   FROM Furniture_sale
   WHERE MONTH(Date.date) = 'Feb'
     AND DAY(Date.date) = '02'
   GROUP BY YEAR(sale_date)) AS Furniture_sale_on_Groundhog_day ON
Furniture_sale.YEAR(sale_date) =
Furniture_sale_on_Groundhog_day.YEAR(sale_date)
GROUP BY YEAR
ORDER BY YEAR ASC;
```

Report 5: State with Highest Volume for each Category

Abstract Code

- User clicks on ***Calculate State with Highest Volume for each Category*** button from **Main Menu**
- Run the **Calculate State with Highest Volume for each Category** task: find the state with the greatest number of units for each category for the chosen year and month
    - User chooses *Year*($Year) and *Month*($Month) from the dropdown menu
    - List all Categories
    - For each Category, find:
        - the Category name;
        - the State that sold the highest number of Products in that Category, linked through City, Store, Sale, Product, Category;
        - the number of Products that were sold by Stores in that State.
    - Sort and display the results by Category name in ascending order.
- When ready, user selects next action from choices in **Main Menu**.

```
SELECT category_name,
      City.state,
      MAX(total_state_sale)
FROM
  (SELECT City.state,
         BelongsTo.fk_belongsto_category_name AS category_name,
         SUM(Sale.quantity) AS total_state_sale
   FROM Sale,
       Product,
       BelongsTo,
       Store,
       City, Date
   WHERE BelongsTo.fk_belongsto_PID = Product.PID
     AND Product.PID = Sale.fk_sale_PID
     AND Sale.fk_sale_store_number = Store.store_number
     AND Store.cityID = City.cityID
     AND Sale.fk_sale_date = Date.date
     AND YEAR(Date.date) = '$Year'
     AND MONTH(Date.date) = '$Month'
   GROUP BY category_name,
           City.state)
GROUP BY Category.name,
        City.state
ORDER BY Category.name ASC;
```

Report 6: Revenue by Population

Abstract Code

- User clicked on ***Revenue by Population*** button from **Main Menu**
- Run the **Summarize Revenue by Population Size**: Query for the information forms about all City.Population, Product.RetailPrice, Discount.DiscountPrice, Store, Sale and Date
    - Group City by Population into Small (population <3,700,000) Medium (>=3,700,000 and <6,700,000 and =9,000,000).
    - Calculate the total revenue for a store within a given year:
        - If the product wat sold on a date without discount: Store.Revenue = Product.RetailPrice * Sale.Quantity
        - If the product was sold with a discount: Store.Revenue = Discount.DiscountPrice*Sale.Quantity
    - Adding up the total Revenues for all stores located in a City;
    - Adding up the total revenues for all cities within a population size;
    - Do the above calculations for all years.
    - Order the revenue results
        - Row (Years): oldest to newest
        - Columns (Population Size): smallest to largest
- When ready, user selects ***Close*** button and go back to **Main Menu.**

```
SELECT 'year',
    AVG(small) AS small_rev,
    AVG(med) AS med_rev,
    AVG(LARGE) AS large_rev
FROM
  (SELECT 'year',
      CASE
        WHEN tmp1.population < 3700000 THEN tmp1.revenue
        ELSE NULL
      END AS small,
      CASE
        WHEN tmp1.population >= 3700000
          AND tmp1.population < 6700000 THEN tmp1.revenue
```

```
            ELSE NULL
        END AS med,
        CASE
            WHEN tmp1.population >= 6700000
                AND tmp1.population < 9000000 THEN tmp1.revenue
            ELSE NULL
        END AS LARGE,
        CASE
            WHEN tmp1.population >= 9000000 THEN tmp1.revenue
        END AS xlarge ELSE NULL END AS xlarge
  FROM
    (SELECT City.city_name,
            City.population,
            Store.store_city,
            CASE
                WHEN Discount.discount_price IS NOT NULL THEN Sale.quantity *
Discount.discount_price
                ELSE Sale.quantity * Product.retail_price
            END AS revenue,
            YEAR(Sale.fk_sale_date) AS 'year'
     FROM Store
     JOIN City ON Store.fk_store_cityID = City.cityID
     JOIN Sale ON Store.store_number = Sale.fk_sale_store_number
     JOIN Product ON Sale.fk_sale_PID = Product.PID
     LEFT JOIN Discount ON Sale.fk_sale_PID = Discount.fk_discount_PID
     AND Sale.fk_sale_date = Discount.fk_discount_date) AS tmp1) AS tmp2
GROUP BY 'year'
ORDER BY 'year' DESC;
```

- When ready, user selects Close button and go back to Main Menu.

Report 7: Childcare Sales Volume

Abstract Code

- User clicked on ***Childcare Sales Volume*** button from **Main Menu**
- Run the **Calculate Sales Volume by Childcare**: Query for the information forms about all Childcare.TimeLimit, Product.RetailPrice, Discount.DiscountPrice, Store, Sale and Date
    - Categorize Childcare into different levels (eg, No Childcare, Low, Medium and High) according to its time limit. Group Stores by their Childcare category.

- Calculate the total revenue for a store within a given month:
  - If the product was sold on a date without discount:
    Store.Revenue = Product.RetailPrice * Sale.Quantity
  - If the product was sold with a discount:
    Store.Revenue = Discount.DiscountPrice*Sale.Quantity
- Adding up the total Revenues for all stores within a Childcare category
- Do the above calculations for 12 recent months.
- Order the revenue results:
  Row (Months): oldest to newest
- Pivot the table

- When ready, user selects ***Close*** button and go back to **Main Menu.**

```
SELECT 'MONTH', ISNULL(CAST(childcare AS varchar), 'No Childcare'),
            SUM(revenue) AS revenue INTO #ChildcareReport
FROM
  (SELECT MONTH(Sale.fk_sale_date) AS 'month',
        Childcare.time_limit AS childcare,
        CASE
           WHEN Discount.discount_price IS NOT NULL THEN Sale.quantity *
Discount.discount_price
           ELSE Sale.quantity * Product.retail_price
        END AS revenue
   FROM Store
   LEFT JOIN Childcare ON Store.fk_store_childcareID= Childcare.childcareID
   JOIN Sale ON Store.store_number = Sale.fk_sale_store_number
   JOIN Product ON Sale.fk_sale_PID = Product.PID
   AND DATEDIFF(MONTH, Sale.fk_sale_date, CURRENT_TIMESTAMP)
BETWEEN 0 AND 2
   LEFT JOIN Discount ON Sale.fk_sale_PID = Discount.discount_PID
   AND Sale.fk_sale_date = Discount.fk_discount_date) AS tmp1
GROUP BY 'MONTH', childcare
ORDER BY 'MONTH' DESC

DECLARE @cols AS NVARCHAR(MAX),
       @query AS NVARCHAR(MAX)
SELECT @cols = STUFF(
                  (SELECT DISTINCT ',' + QUOTENAME(childcare_time)
                   FROM #ChildcareReport
                   FOR XML PATH(''), TYPE).value('.', 'NVARCHAR(MAX)') , 1, 1, '')
SET @query = 'SELECT month, ' + @cols + ' from
```

```
        (
          select month, childcare_time, revenue
          from #ChildcareReport
        ) x
        pivot
        (
          SUM(revenue)
          for childcare_time in (' + @cols + ')
        ) p ' execute(@query)

DROP TABLE #ChildcareReport;
```

- When ready, user selects Close button and go back to Main Menu.

Report 8: Analyze Sales by Restaurant Existence

Abstract Code

- Click on the ***Analyze Sales by Restaurant Existence*** button on **Main Menu**
- Calculate and generate result
    - Query the Sales.Quantity, Store.Restaurant (if restaurant exists or not), Product.PID, and Category.Name of all sales record
    - Group data by Store.Restaurant and Category.Name, sum all Sales.Quantity in a group
    - Sort result by non-restaurant before restaurant
    - Sort result by category name ascending

```
SELECT BelongsTo.belongsto_category_name AS Category,
      CASE
         WHEN Store.restaurant ='True' THEN 'Restaurant'
         ELSE 'Non-restaurant'
      END AS 'Store Type',
      SUM(Sale.quantity) AS 'Quantity Sold'
FROM BelongsTo
JOIN Product ON BelongsTo.fk_belongsto_PID=Product.PID
JOIN Sale ON Sale.fk_sale_PID=Product.PID
RIGHT OUTER JOIN Store ON
Store.store_number=Sale.fk_sale_store_number
GROUP BY Category,
       'Store Type',
ORDER BY Category,
```

```
        'Store Type' ASC;
```

- When ready, click ***Close*** to go back to **Main Menu**

Report 9: Count Sold Products during and outside Advertising Campaign

Abstract Code

- User clicks **on Count *Sold Products during and outside Advertising Campaign*** button from **Main Menu**:
- Run the ***Count Sold Products during and outside Advertising Campaign***: Query for the information forms about all Products, Date, Sale, Adcampaign.
    - Find the products where the product has sold and discounted from Sale and Date; display the PID, product name and quantity.
    - Find the discount_date with AdCampaign in Date form as *Dis_Adv*;
        - Find the product in Sale form using Dis_Adv;
        - Find the product ID(PID) and sold quantity;
        - Sum the quantity as Sold_During_Campaign;
    - Find the discount_date without AdCampaign in Date form as Dis_NoAdv;
        - Find the product in Sale form using Dis_NoAdv;
        - Find product ID(PID) and sold quantity;
        - Sum the quantity as Sold_Outside_Campaign;
    - Combine the Sold_During_Campaign and Sold_Outside_Campaign using PID;
        - Display the product name using PID;
        - Subtract the Sold During Campaign from Sold Outside Campaign; Display the result as Difference;
    - Sort the list as Difference descending; Display top 10 and bottom 10 rows from the list to this report.

```
SELECT TOP 10 * FROM
        (SELECT
        tmp1.'Product PID', tmp1.'Product Name',
        tmp1.'Sold During Campaign', tmp1.'Sold Outside Campaign', tmp1.'Sold During
        Campaign' -  tmp1.'Sold Outside Campaign' AS Difference,
        FROM
                (SELECT S.fk_sale_PID AS 'Product PID',
                P.name(P.PID=S.fk_sale_PID) AS 'Product Name',
                CASE WHEN S.fk_sale_date IN  (SELECT fk_eventdate_date from
                EventDate) THEN SUM(S.quantity) END AS 'Sold During Campaign' ,
```

```
                CASE WHEN S.fk_sale_date NOT IN  (SELECT fk_eventdate_date from
                EventDate) THEN SUM(S.quantity) END  AS 'Sold Outside Campaign'
                FROM
                        (Sale RIGHT JOIN  Discount
                        ON Discount.fk_discount_date = Sale.fk_sale_date
                        LEFT JOIN EventDate
                        ON Sale. fk_sale_date = EventDate .fk_event_date
                        )
                AS tmp1
                )
        ORDER BY Difference DESC
        )
        AS top10
UNION
SELECT TOP 10 * FROM
         (SELECT
        tmp2.'Product PID', tmp2.'Product Name',
        tmp2.'Sold During Campaign', tmp2.'Sold Outside Campaign', tmp2.'Sold During
        Campaign' -  tmp2.'Sold Outside Campaign' AS Difference,
        FROM
                (SELECT S.fk_sale_PID AS 'Product PID',
                P.name(P.PID=S.fk_sale_PID) AS 'Product Name',
                CASE WHEN S.fk_sale_date IN  (SELECT fk_eventdate_date from
                EventDate) THEN SUM(S.quantity) END AS 'Sold During Campaign' ,
                CASE WHEN S.fk_sale_date NOT IN  (SELECT fk_eventdate_date from
                EventDate) THEN SUM(S.quantity) END  AS 'Sold Outside Campaign'
                FROM
                        (Sale RIGHT JOIN  Discount
                        ON Discount.fk_discount_date = Sale.fk_sale_date
                        LEFT JOIN EventDate
                        ON Sale. fk_sale_date = EventDate .fk_event_date
                        )
                AS tmp2
                )
        ORDER BY Difference ASC
        )
        AS bot10
ORDER BY Difference DESC;
```

- Return to the **Main Menu** when closing this report.