Table of Contents

Abstract Code with SQL

View Stats (Main Menu)

Abstract Code

- Show *"**Store Revenue by Year by State Report**"*, *"**Store Revenue by Year by State Report**"*, *"**Store Revenue by Year by State Report**"*, *"**Outdoor Furniture on Groundhog Day Report**"*, *"**State with Highest Volume for each Category Report**"*, "***Revenue by Population Report***", *"**Childcare Sales Volume Report**"*, *"**Restaurant Impact on Category Sales Report**"*, and *"**Advertising Campaign Analysis Report**"* links.
- Show ***Holidays*** and ***City Population*** links
- Run the **ViewStats** task:
    - Run the **Store** task:
        - Count and display total of STORE instances.

```
SELECT COUNT(*) as TotalStores FROM Store;
```

        - Count and display total of STORE instances that have Restaurant or Snackbar

```
SELECT COUNT(*) as TotalNumberRestaurants FROM Store
WHERE restaurant = True OR snackbar = True;
```

        - Count and display total of STORE that offers Childcare

```
SELECT COUNT(*) as TotalStoresOfferChildcare FROM Store
WHERE `limit` IS NOT NULL;
```

    - Run **Product** task:
        - Count and display all PRODUCT instances

```
SELECT COUNT(*) as TotalProducts FROM Product;
```

    - Run **Ad Campaign** task:
        - Count and display total of distinct AD_CAMPAIGNs

```
SELECT COUNT(DISTINCT description) as TotalUniqueCampaigns
FROM AdCampaign;
```

- Upon:
    - Click ***Holidays*** link - Jump to the **View/Add Holidays** task
    - Click ***City Population*** link - Jump to the **Edit City Population** task
    - Click ***Category Report*** link - Jump to the **View Category** task
    - Click ***Actual vs. Predicted Revenue for Couches & Sofas*** link - Jump to the **Actual vs. Predicted Revenue for Couches & Sofas** task
    - Click ***Store Revenue by Year by State*** link - Jump to the **View Store Revenue by Year by State** task
    - Click ***Outdoor Furniture on Groundhog Day*** link - Jump to the **View Outdoor Furniture on Groundhog Day** task

- Click ***State with Highest Volume for each Category*** link - Jump to the **View State with Highest Volume for each Category** task
- Click ***Revenue by Population*** link - Jump to the **View Revenue by Population** task
- Click ***Childcare Sales Volume*** link - Jump to the **View Childcare Sales Volume** task
- Click ***Restaurant Impact on Category Sales*** link - Jump to the **View Restaurant Impact on Category Sales** task
- Click ***Advertising Campaign Analysis*** link - Jump to the **View Advertising Campaign Analysis** task

View/Add Holidays (Holiday Maintenance Form)

Abstract Code

- User clicked on ***Holidays*** link from the **Main Menu**:
- Run the **View/Add Holidays** task:
    - Run **View Holiday** task: query for list of holidays

```
SELECT holiday_name, `date` FROM Holiday;
```

- Display ***Add Holiday*** button at the bottom of the page along with $*Date* input field and $*Holiday Name* input field.
- If user clicked on ***Add Holiday*** button from the **Holiday Maintenance** form:
    - Data Validation: User can't enter string for date datatype in Date. User can't enter a HolidayName that already exists for that date.
    - User enters *$Date* and *$HolidayName* into corresponding input fields
    - If data validation is successful for both Date and HolidayName then:

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

        - Return to **Holiday Maintenance** form and Display confirmation message
    - Else HolidayName and Date are invalid, display **Holiday Maintenance** form, with error message

Edit City Population (City Population Update Form)

Abstract Code

- User clicked on ***City Population Update*** link from the **Main Menu**:
- Display form with input fields for *$City,* and *$Updated Population* along with ***Update Population*** button
- User enters *$City* and *Updated Population*
- Data Validation: User can only enter int datatype for Population. User can't enter int for string datatype in CityName. Population can't be updated to a null or blank value.
- Upon user clicking on ***Update Population*** button:
    - If data validation is correct for *City* and *Updated Population* then:
        - Run the **Edit City Population** task:

```
UPDATE City SET population = '$updated_population' WHERE
city = '$city';
```

            - Display confirmation message
    - Else City and Population is invalid, return to **City Population Update** form and display error

View Category Report (Report 1)

Abstract Code

- User clicked on ***Category Report*** link from the **Main Menu**:
- Run the **View Category Report** task:
- Find the Name for each CATEGORY
- For each CATEGORY:
    - Calculate the number of PRODUCTs in the CATEGORY; Display the number of PRODUCTS
    - Find the retail Price of the PRODUCTs in the CATEGORY
    - Calculate the average retail Price of the PRODUCTs in the CATEGORY; Display the average retail Price of the PRODUCTs
    - Find the minimum Price of the PRODUCTs in the CATEGORY; Display the minimum Price of the PRODUCTS
    - Find the maximum Price of the PRODUCTs in the CATEGORY; Display the maximum Price of the PRODUCTs
- Sort ascending the CATEGORY values by Name

```
SELECT
      Category.name                 as 'Category Name',
      count(Product.pid)            as 'Total Number of Products',
      min(Product.price)            as 'Minimum Regular Retail Price',
      round(avg(Product.price), 2)  as 'Average Regular Retail Price',
      max(Product.price)            as 'Maximum Regular Retail Price'
FROM Category
        LEFT JOIN ProductCategory ON Category.name = productCategory.name
        LEFT JOIN Product ON Product.pid = productCategory.pid
GROUP BY Category.name
ORDER BY Category.name ASC;
```

- If user clicks ***Main Menu*** button, return to **Main Menu**

View Actual vs Predicted Revenue for Couches & Sofas (Report 2)

Abstract Code

- User clicked on ***Actual versus Predicted Revenue*** link from the **Main Menu**:
- Run the **View Actual versus Predicted Revenue for Couches & Sofas** Task:
- Get PID, Name and Price from PRODUCT for all products within the CATEGORY "Couches and Sofas"
- For each PRODUCT with Category "Couches and Sofas"
    - Get the Revenue of and number of Sales at discount
        - Get the discount_price when the PRODUCT was sold at discount using Date and PID from the products in the category Couches and Sofas and the data in SOLD and HAS_DISCOUNT.
        - Multiply the Quantity of each product for the discount_price, add it and name it 'total_rev'
        - Add all the products sold at discount and name it 'totalSoldDiscount'
    - Get the Revenue of and number of Sales without discount
        - Get the DiscountPrice when the PRODUCT was sold without discount using Date and PID from the products in the CATEGORY Couches and Sofas and the data in the SOLD and HAS_DISCOUNT.
        - Multiply the Quantity of each product for the DiscountPrice, add it and name it 'total_rev'
    - Calculate Predicted Sales and obtain the total quantity of product sold:
        - Get the total amount of the product sold adding Quantity in the table SOLD at pid level
        - Obtain the (regular retail) price of the product, matching the PID of couchSofasProducts with the data in the table PRODUCT
        - Multiply the quantity sold of each product by 0.75 to calculate the predicted sales. Multiply that amount with the price of the product to obtain the predicted revenue. Name it TotalPredictedRevenue
    - Calculate the difference in actual vs predicted and display the data
        - Calculate the difference between 'ActualRevenue' and 'PredictedRevenue' name it 'Difference Actual vs Predicted'
        - Display, sorted in descending order by 'Difference Actual vs Predicted', PID, NAME, Price, total number of units sold, total number of units sold at discount price, the total revenue, the predicted revenue and 'Difference Actual vs Predicted' for each product where absolute value of 'Difference Actual vs Predicted' is > 5000

```
WITH couchSofasProducts as (SELECT Product.pid, Product.price,
Product.name -- assuming there is only one combination product price
FROM Product
JOIN ProductCategory
ON Product.pid = ProductCategory.pid
WHERE ProductCategory.name = 'Couches and sofas'),

-- Total sum of price of units sold at discount
totalNumberUnitWithDiscount as (SELECT couchSofasProducts.pid,
couchSofasProducts.name, SUM(Sold.quantity *
HasDiscount.discount_price) as TotalPrice, SUM(Sold.quantity) as
totalSoldDiscount
FROM couchSofasProducts
JOIN Sold on couchSofasProducts.pid = Sold.pid
JOIN HasDiscount on HasDiscount.pid = couchSofasProducts.pid
AND Sold.Date = HasDiscount.date
WHERE HasDiscount.date IS NOT NULL
GROUP BY couchSofasProducts.pid, couchSofasProducts.name),

-- Total sum of price of units sold without discount
totalNumberUnitWithoutDiscount as (
SELECT couchSofasProducts.pid, couchSofasProducts.name,
SUM(Sold.quantity * couchSofasProducts.Price) as TotalPrice
FROM couchSofasProducts
JOIN Sold on  couchSofasProducts.pid = Sold.pid
LEFT JOIN HasDiscount on HasDiscount.pid = couchSofasProducts.pid
AND Sold.Date = HasDiscount.date
WHERE HasDiscount.date IS NULL
GROUP BY couchSofasProducts.pid, couchSofasProducts.name),

-- Total predicted
totalPredicted as (SELECT couchSofasProducts.pid,
couchSofasProducts.name, SUM(Sold.quantity * couchSofasProducts.Price *
0.75) as TotalPredictedPrice, SUM(Sold.Quantity) as totalSold
FROM couchSofasProducts
JOIN Sold on  couchSofasProducts.pid = Sold.pid
GROUP BY couchSofasProducts.pid, couchSofasProducts.name)

SELECT couchSofasProducts.pid, couchSofasProducts.name,
couchSofasProducts.price, totalPredicted.totalSold,
totalNumberUnitWithDiscount.totalSoldDiscount,
(COALESCE(totalNumberUnitWithDiscount.TotalPrice,0) +
totalNumberUnitWithoutDiscount.TotalPrice) as TotalRevenue,
totalPredicted.TotalPredictedPrice,
(totalNumberUnitWithDiscount.TotalPrice +
totalNumberUnitWithoutDiscount.TotalPrice) -
totalPredicted.TotalPredictedPrice as 'Difference Actual vs Predicted'
FROM couchSofasProducts
LEFT JOIN totalNumberUnitWithDiscount ON couchSofasProducts.pid =
totalNumberUnitWithDiscount.pid
LEFT JOIN totalNumberUnitWithoutDiscount ON couchSofasProducts.pid =
totalNumberUnitWithoutDiscount.pid
LEFT JOIN totalPredicted ON couchSofasProducts.pid = totalPredicted.pid
WHERE ABS((totalNumberUnitWithDiscount.TotalPrice +
totalNumberUnitWithoutDiscount.TotalPrice) -
totalPredicted.TotalPredictedPrice ) > 5000
ORDER BY ABS((totalNumberUnitWithDiscount.TotalPrice +
totalNumberUnitWithoutDiscount.TotalPrice) -
```

```
totalPredicted.TotalPredictedPrice ) DESC;
```

- If user clicks ***Main Menu*** button, return to **Main Menu**

View Store Revenue by Year by State Report (Report 3)

Abstract Code

- User clicked on ***View Store Revenue by Year by State*** link from the **Main Menu**:
- Run the **Select State** task:
- Query for all unique States in CITY in the database to be displayed in the UI

```
SELECT DISTINCT(City.state) FROM City
ORDER BY City.state
```

- User selects `$State` from dropdown menu in the UI and clicks ***Generate Report*** button
- Upon clicking ***Generate Report*** button:
    - For each Year in SOLD
        - Find all CITY that have selected `$State`
        - For each STORE in a City of `$State` :
            - Display StoreNo, Address and the City Name
            - Calculate the revenue:
                - Calculate RevenuByDate: multiply Quantity by PRODUCT Price if there was not a discount for that PRODUCT during the Date it was SOLD, use Discount Price in case there a discount
            - Calculate the TotalRevenue: Add the RevenuByDate calculated for each Date where there was a sale
            - Display the revenue
    - Sort by Year in ascending order and then by TotalRevenue in descending order
- If user clicks ***Main Menu*** button, return to **Main Menu**

```
WITH storesInState AS (
   SELECT City.city,
          Store.store_no,
          Store.address,
          Store.State
   FROM City
            JOIN Store on (City.city = Store.city and City.state = Store.state)
   WHERE City.State = $State
),

-- Gets the revenue at discount and sales at discount per store
    revenueDiscount AS (
        SELECT storesInState.store_no,
               SUM(Sold.quantity * HasDiscount.discount_price) as revenue,
               YEAR(sold.date)                                 as year_sold
        FROM Sold
                 JOIN storesInState ON storesInState.store_no = SOLD.store_no
                 JOIN HasDiscount on HasDiscount.pid = SOLD.pid AND Sold.Date = HasDiscount.date
                 JOIN Product on Product.pid = Sold.pid
        WHERE HasDiscount.date IS NOT NULL
        GROUP BY YEAR(sold.date), storesInState.store_no
    ),
```

```
-- Gets the revenue at discount and sales at discount per store
    revenueRetail AS (
        SELECT storesInState.store_no,
               SUM(Sold.quantity * Product.Price) as revenue,
               YEAR(sold.date)                    as year_sold
        FROM Sold
                 JOIN storesInState ON storesInState.store_no = SOLD.store_no
                 LEFT JOIN HasDiscount ON HasDiscount.pid = SOLD.pid AND Sold.Date =
HasDiscount.date
                 JOIN Product ON Product.pid = Sold.pid
        WHERE HasDiscount.date IS NULL
        GROUP BY YEAR(sold.date), storesInState.store_no)

SELECT storesInState.store_no,
       storesInState.address,
       storesInState.city,
       YEAR(sold.date)                                                    AS  Sales_Year,
(COALESCE(revenueDiscount.revenue, 0) + COALESCE(revenueRetail.revenue, 0)) AS total_revenue
FROM storesInState
        LEFT JOIN sold ON storesInState.store_no = SOLD.store_no
        LEFT JOIN revenueDiscount ON
   (storesInState.store_no = revenueDiscount.store_no AND YEAR(sold.date) =
revenueDiscount.year_sold)
        LEFT JOIN revenueRetail
                  ON (storesInState.store_no = revenueRetail.store_no AND YEAR(sold.date) =
revenueRetail.year_sold)
GROUP BY storesInState.store_no,
        storesInState.address, storesInState.city,
        YEAR(sold.date), total_revenue
ORDER BY YEAR(sold.date) ASC, total_revenue DESC;
```

View Outdoor Furniture on Groundhog Day Report (Report 4)

Abstract Code

- User clicked on ***Outdoor Furniture on Groundhog Day*** link from **Main Menu**:
- Run the **View Outdoor Furniture on Groundhog Day** task: query total and average quantity sold per year and on February 2.
    - Find PRODUCTCATEGORY, and SOLD
    - For each year in SOLD:
        - Calculate total quantity sold of 'Outdoor Furniture' from CATEGORY for entire year as 'total quantity sold'
        - Calculate average daily quantity by dividing total quantity sold (from above) by 365 as 'average daily quantity sold'
        - Calculate total quantity sold of 'Outdoor Furniture' from CATEGORY sold on February 2 as 'GroundHogDay'
    - Display Year, total quantity sold, average daily quantity sold, GroundHogDay
    - Sort by Year in ascending order

```
SELECT YEAR(x.date) as Year,
  sum(x.quantity) as TotalQuantity,
  sum(x.quantity)/365 as AvgDailyQuantity,
  sum(y.quant) as GroundHogDay
FROM
(SELECT date, quantity -- only outdoor sales
  FROM Sold JOIN ProductCategory ON Sold.PID = ProductCategory.PID
  WHERE ProductCategory.name = 'Outdoor Furniture') as x
LEFT JOIN
(SELECT date, quantity as quant -- only outdoor on Feb. 2
FROM Sold JOIN ProductCategory ON Sold.PID = ProductCategory.PID
  WHERE ProductCategory.name = 'Outdoor Furniture'
  AND MONTH(Sold.date) = 2
  AND DAY(Sold.date) = 2) as y
ON x.date = y.date
GROUP BY YEAR
ORDER BY YEAR ASC;
```

- If user clicks ***Main Menu*** button, return to **Main Menu**

View State with Highest Volume for each Category (Report 5)

Abstract Code

- User clicked on ***View State With Highest Volume of Sales for each Category*** link from **Main Menu**:
- Run **View State With Highest Volume for Each Category** task:
    - Run the **Select Dates** task: query for all available SOLD Date in the database to be displayed in the UI

```
SELECT *
FROM Sold AS SoldDuringSelectedDays
WHERE MONTH(`date`) = '$Month' AND YEAR(`date`) = '$Month';
```

    - User selects *$Month* and *$Year* from dropdown menu in the UI and clicks ***Generate Report*** button
    - Run **View Report** task:
        - For each PRODUCT_CATEGORY:
            - Display the CATEGORY Name
            - Find each SOLD PRODUCT which Date corresponds to the *$Month* and *$Year* provided by the user that belongs to the CATEGORY
            - Find the STORE State or each PRODUCT SOLD
            - Add the SOLD Quantity of the products in the same State as “total_number_sold”
        - Display the State name with maximum “total_number_sold” and its corresponding “total_number_sold” value
        - If two states have the same “total_number_sold” then duplicate the category in the final report for each of them
        - Sort the list of categories by name in ascending order

```
WITH TotalNumberSold(category_name, state, total_number_sold)
        AS (SELECT category_name, state, SUM(quantity) AS total_number_sold
            FROM (SELECT name AS category_name, state, quantity
                  FROM ProductCategory
                           JOIN Sold ON ProductCategory.pid = Sold.pid
                           JOIN Store ON Store.store_no = Sold.store_no
                  WHERE MONTH (Sold.`date`) = '$Month' AND YEAR (Sold.`date`) =
'$Year') AS ProductsSoldPerCategory
            GROUP BY category_name, state
   )
SELECT TotalNumberSold.category_name, TotalNumberSold.state,
TotalNumberSold.total_number_sold
FROM (SELECT category_name, MAX(total_number_sold) AS max_total_number_sold
     FROM TotalNumberSold
     GROUP BY category_name) AS TotalNumberSoldPerCategory
        JOIN TotalNumberSold ON TotalNumberSoldPerCategory.category_name =
TotalNumberSold.category_name
   AND TotalNumberSoldPerCategory.max_total_number_sold =
```

```
TotalNumberSold.total_number_sold
ORDER BY category_name ASC;
```

- If user clicks ***Main Menu*** button, return to **Main Menu**

View Revenue by Population (Report 6)

Abstract Code

- User clicked on ***Revenue by Population*** link from **Main Menu**:
- Run the **View Revenue by Population** task: queries TotalSale by City population categories and compare per annum
- Find SOLD, DATE, CITY, PRODUCT, HAS_DISCOUNT
- For each year in DATE:
    - For each CITY:
        - Calculate Total Revenue as:
            - Calculate revenue of retail priced items: multiply Quantity by PRODUCT Price for dates where there are no discounts
            - Calculate revenue of discounted items: multiply Quantity by HAS_DISCOUNT Price for dates where there are discounted products
            - Calculate total revenue: Sum retail price items and discount price items
- Aggregate Total Revenue by population category as:
    - For each CITY aggregate revenue by population category:
        - IF population is < 3,700,000
            - Then population category is “Small”
        - ELIF population is between 3,7000,000 and 6,699,999
            - Then population category is “Medium”
        - ELIF population is between 6,700,000 and 8,999,999
            - Then population category is “Large”
        - ELIF: population is > 9,000,000
            - Then population category is “Extra Large”
    - Display Year, Small, Medium, Large, Extra Large and the corresponding Total Revenue for each Population Category
- Sort by year ascending (i.e., oldest at the top, newest at the bottom)

```
-- Calculate retail revenue for each city
WITH retail_rev as (select YEAR(Sold.date) as Year, city, state, sum(price *
quantity) as retailRev
  FROM sold
     LEFT JOIN Product P on Sold.pid = P.pid
     LEFT JOIN HasDiscount ON P.pid = HasDiscount.pid
     LEFT JOIN Store on Sold.store_no = Store.store_no
  WHERE HasDiscount.date IS null
  GROUP BY city, state, year),

-- Calculate discounted revenue for each city
```

```
 discount_rev as (select YEAR(Sold.date) as Year, city, state,
sum(discount_price * quantity) as discountRev
  FROM sold
     LEFT JOIN Product P on Sold.pid = P.pid
     LEFT JOIN HasDiscount ON P.pid = HasDiscount.pid
     LEFT JOIN Store on Sold.store_no = Store.store_no
  GROUP BY city, state, year),
citySize as (SELECT City.city,City.state,
             City.population as pop, -- Calculate city population categories
       CASE
          WHEN (City.population < 3700000) THEN 'small'
          WHEN (City.population BETWEEN 3700000 AND 6699999) THEN 'medium'
          WHEN (city.population BETWEEN 6700000 AND 8999999) THEN 'large'
          WHEN (City.population >= 9000000) THEN 'Xlarge'
          ELSE null
       END         as CityPopulation
  FROM City),

totalRev as (select retail_rev.Year,
                        retail_rev.city,
                        retail_rev.state,
                        (COALESCE(retail_rev.retailRev, 0) +
COALESCE(discount_rev.discountRev, 0)) as total_rev
  FROM retail_rev
     JOIN discount_rev
     ON retail_rev.Year = discount_rev.Year AND
        retail_rev.city = discount_rev.city AND
        retail_rev.state = discount_rev.state),

revWithCategory as (Select citySize.CityPopulation, totalRev.Year,
SUM(totalRev.total_rev) as total
  FROM citySize
     JOIN totalRev ON
          citySize.city = totalRev.city AND
          citySize.state = totalRev.state
  GROUP BY totalRev.Year, citySize.CityPopulation)

SELECT revWithCategory.Year,
      SUM(CASE WHEN revWithCategory.CityPopulation = 'small' THEN total END)
AS SMALL,
      SUM(CASE WHEN revWithCategory.CityPopulation = 'medium' THEN total END)
AS MEDIUM,
      SUM(CASE WHEN revWithCategory.CityPopulation = 'large' THEN total END)
AS LARGE,
      SUM(CASE WHEN revWithCategory.CityPopulation = 'xlarge' THEN total END)
AS Extra_Large
FROM revWithCategory
GROUP BY revWithCategory.Year;
```

- If user clicks ***Main Menu*** button, return to **Main Menu**

View Childcare Sales Volume (Report 7)

Abstract Code

- User clicked on ***Childcare Sales Volume*** link from the **MainMenu**:
- Run the **View Childcare Sales Volume** task:
    - Find the last 12 months starting from the last available SOLD Date
    - Find each PRODUCT in SOLD which Date is part of the previous 12 month
        - Find the STORE Limit who SOLD the product
        - Find the TotalAmount of each SOLD product by multiplying SOLD Quantity by PRODUCT Price or HAS_DISCOUNT DiscountPrice (if there is one available on this date)
    - Add the TotalAmount of all SOLD products with the same limit and display this values as “childcare_limit”
    - Note that the “childcare_limit” will be pivoted to distinct columns using the host language

```
SELECT month_of_year, childcare_limit, SUM(total_amount) AS total_sales
FROM (SELECT MONTH(Sold.`date`)
AS month_of_year,
            IFNULL(Store.`limit`, 0)
AS childcare_limit,
            Sold.quantity * IFNULL(HasDiscount.discount_price,
Product.price)   AS total_amount
     FROM Sold
              LEFT JOIN Product ON Product.pid = Sold.pid
              LEFT JOIN HasDiscount ON HasDiscount.pid = Sold.pid AND
HasDiscount.`date` = Sold.`date`
              LEFT JOIN Store ON Store.store_no = Sold.store_no
     WHERE Sold.`date` > (SELECT MAX(Sold.date) FROM Sold) - INTERVAL 12
month) AS SalesPerChildcareLimit
GROUP BY month_of_year, childcare_limit;
```

- If user clicks ***Main Menu*** button, return to **Main Menu**

View Restaurant Impact on Category Sales (Report 8)

Abstract Code

- User clicked on ***Restaurant Impact on Category Sales*** link from the **MainMenu**:
- Run the **View Restaurant Impact on Category Sales** task:
    - For each PRODUCTCATEGORY:
        - Display CATEGORY Name
        - Find each SOLD where STORE has RESTAURANT:
            - Add Quantity from SOLD of all STOREs and display this value as "Quantity Sold for Restaurant"
        - Find each SOLD where STORE does not have RESTAURANT:
            - Add all Quantity of all STOREs and display this value as "Quantity Sold for Non-Restaurant"
    - Sort the list of categories by name in ascending order
    - For each CATEGORY group by "Quantity Sold for Non-Restaurant" and "Quantity Sold for Restaurant". Sort by Non-Restaurant first.
    - The host language will group the rows, so that each category is shown once

```
SELECT ProductCategory.name as Category_name, S.restaurant as non
FROM ProductCategory
JOIN Sold on ProductCategory.pid = Sold.pid
JOIN Store S on Sold.store_no = S.store_no
GROUP BY Category_name, restaurant;

SELECT SUM(store_no) as Stores_quantity
FROM Store
GROUP BY restaurant;

WITH getProductCategoryInSol as (SELECT Sold.store_no,
   ProductCategory.name,
   SUM(quantity) as Quantity_Sold
FROM Sold
LEFT JOIN  ProductCategory ON Sold.pid = ProductCategory.pid
GROUP BY Sold.store_no, ProductCategory.name)

SELECT
      getProductCategoryInSol.name as Category,
CASE
   WHEN Store.restaurant = 1 Then "Restaurant"
   WHEN Store.restaurant = 0 Then "Non-restaurant"
END AS Store_type,
SUM(Quantity_Sold) as Quantity_Sold
FROM Store
JOIN getProductCategoryInSol
ON Store.store_no = getProductCategoryInSol.store_no
GROUP BY getProductCategoryInSol.name, Store_type
ORDER BY getProductCategoryInSol.name, Store_type ASC;
```

- If user clicks ***Main Menu*** button, return to **Main Menu**

View Advertising Campaign Analysis (Report 9)

Abstract Code

- User clicked on ***Advertising Campaign Analysis*** link from **Main Menu**:
- Run the **View Advertising Campaign Analysis** task:
    - Find each PRODUCT that HAS_DISCOUNT
        - Find each PRODUCT that was SOLD on a Date that HAS_DISCOUNT
        - Add the SOLD Quantity of all PRODUCTs which Date is also in DATE_ADCAMPAIGN and display the result as “sold_during_campaign”
        - Add the SOLD Quantity of all PRODUCTs which Date is not in DATE_ADCAMPAIGN and display the result as “sold_outside_campaign”
        - Compute the difference between the values “sold_during_campaign” and “sold_outside_campaign” and display it as “difference”
        - Display PRODUCT PID, PRODUCT Name
    - Sort the products by Difference in descending order
    - Display the top 10 and bottom 10 products only from the resulting list of products

```
WITH ALLResult (pid, name, total_sold_during_campaign,
total_sold_outside_campaign, difference) AS (
   SELECT pid,
          name,
          SUM(sold_during_campaign)                                  AS
total_sold_during_campaign,
          SUM(sold_outside_campaign)                                 AS
total_sold_outside_campaign,
          SUM(sold_during_campaign) - SUM(sold_outside_campaign) AS
difference
   FROM (SELECT Product.pid,
                Product.name,
                IF(DateAdCampaign.description IS NOT NULL, Sold.quantity,
0) AS sold_during_campaign,
                IF(DateAdCampaign.description IS NULL, Sold.quantity, 0)
AS sold_outside_campaign
         FROM Product
                  JOIN HasDiscount ON Product.pid = HasDiscount.pid
                  LEFT JOIN Sold ON Product.pid = Sold.pid AND
HasDiscount.`date` = Sold.`date`
                  LEFT JOIN DateAdCampaign ON HasDiscount.`date` =
DateAdCampaign.`date`
                  ) AS ProductsWithDiscountSalesSummary
   GROUP BY pid, name
)
   (SELECT *
    FROM ALLResult
    ORDER BY difference DESC
    limit 10)
UNION
(SELECT *
FROM (SELECT *
      FROM ALLResult
      ORDER BY difference ASC
      limit 10) AS ProductResultsOrdered
```

```
ORDER BY difference DESC);
```

- If user clicks ***Main Menu*** button, return to **Main Menu**

APPENDIX: Revisions since Phase 1

The summary of revisions below have been incorporated into this report and the enhanced EER diagram based TA feedback subsequent to Phase 1.

EER:

- HOLIDAY now a subtype of DATE
- HOLIDAY Name is now an attribute (previously was multi-valued attribute)
- Removed FOOD_SERVICE and its subtypes
- Moved Restaurant and Snackbar as a attributes on STORE
- Revised CHILDCARE from weak entity to entity with Limit as unique identifier and revised cardinality of the relationship
- Revised SOLD to ternary relationship from entity type
- Revised CITY to hold City and State as composite attribute
- Revised AD_CAMPAIGN to have mandatory relationship with DATE

Report:
We have revised the abstract code for each report to consider the updated EER diagram and new relations/entities. This included changes to lookup tables/relationships as needed and changed variable names where required. Below is a summary of significant revisions to the reports.

Report 2: changed abstract code to reflect that there is no task decomposition for this task and is executed in one query, with four main steps.
Report 4:

- Revised references from CATEGORY to PRODUCTCATEGORY relationship instead.

Report 7: Updated all references to SOLD_DATE to reference SOLD and DATE relations.

- Similarly updated references to STORE_SOLD and changed to STORE and SOLD.
- Changed OFFERS reference for CHILDCARE.

Report 8: Uses PRODUCTCATEGORY relation instead of CATEGORY
Report 9: Added more details regarding the relations we need to access for this query
Report 9: Uses the DATE_ADCAMPAIGN relation instead of ADCAMPAIGN and SOLD