Phase 2 Abstract Code w/SQL

Team 054
03/20/2021

Table of Contents

Abstract Code w/ SQL

Abstract Code Markups:

Bold Italics: Buttons

Bold underline: forms

Blue: TableName

Green: Report

Main Menu

Abstract code:

- Upon open main menu, display a generated view from Store, Childcare, Products, Date table and get a read-only report contain*:* count of stores, count of stores that offer food (have a restaurant, a snack bar, or both), count of stores offering childcare, count of products, and count of distinct advertising campaigns.
- Show "***Holiday Information***", "***City Population Information***", "***Category Report***" and "***Couches and Sofas Report***", "***Store Revenue by Year by State Report****", "Outdoor Furniture on Groundhog Day vs Avg", "Sale Report State with Highest Volume for Each Category by Month", "Revenue by Population Report Childcare Sales Volume", "Report Advertising Campaign Analysis Report"* buttons.
- Upon:
    - Click ***Holiday Information***, direct to the **View and Add Holiday Information** task .
    - Click ***City Population Information***, direct to the **View and Update the City Population** task.
    - Click ***Category Report***, direct to the **Generate Category Report** task.
    - Click ***Couches and Sofas Report***, direct to **General Actual vs Predict Revenue for Couches and Sofas Report** task.
    - Click ***Store Revenue by Year by state report***, direct to **View Store Revenue by Year and State** task.
    - Click ***Outdoor Furniture on Groundhog Day vs Avg Sale Report***, direct to **Generate Groundhog Day Report** task.
    - Click on ***State with Highest Volume for Each Category by Month***, direct to **Show State with Highest Volume for Each Category Report** task.
    - Click ***Revenue by Population Report***, direct to **Show Revenue by Population Report** task.
    - Click ***Childcare Sales Volume Report***, direct to **Generate Childcare Sales Volume Report** task.

- Click ***Restaurant Impact***, direct to **Generate Restaurant Impact Report** task.
- Click ***Advertising Campaign Analysis Report***, direct to **Generate Advertising Campaign Report** task.

```
SELECT (SELECT Count(store_number)
        FROM   store)                AS count_of_stores,
       (SELECT Count(store_number)
        FROM   store
        WHERE  restaurant IS TRUE
               OR snack_bar IS TRUE) AS store_that_offers_food,
       (SELECT Count(store_number)
        FROM   store
        WHERE  maximum_time != 0)     AS store_that_offers_childCare,
       (SELECT Count(productid)
        FROM   product)               AS count_of_products,
       (SELECT Count(DISTINCT campaign_description)
        FROM   belongto)              AS count_of_distinct_campaigns
FROM   store
       LEFT JOIN (SELECT *
                  FROM   (SELECT *
                          FROM   transaction
                                 LEFT JOIN product
                                        ON transaction.productid =
                                           product.productid) a
                                 LEFT JOIN belongto
                                        ON a.date = belongto.date) b
              ON store.store_number = b.store_number
LIMIT  1
```

OUTPUT:

	count_of_stores bigint	store_that_offers_food bigint	store_that_offers_childcare bigint	count_of_products bigint	count_of_distinct_campaigns bigint
1	15	4	11	5	3

View and Add Holiday Information

Abstract code:

- Click on the ***Holiday Information*** button from the Main Menu.
- Click on the ***View Holiday*** button from the **Holiday Information** form.
  - Choose Holiday or Date
  - Enter *HolidayName* (‘$HolidayName’) or *Date*(‘$Date’)
  - Display HolidayName and Date from Date table

```
SELECT holiday_name,
       date
FROM   holiday
```

```
WHERE  holiday_name = '$holiday_name'
    AND date = '$Date';
```

- Click on the ***Add Holiday*** button.
    - Enter *Date*('$Date')
    - Enter *HolidayName*('$HolidayName')
    - Store HolidayName and Date

```
INSERT INTO holiday VALUES
        ('$Date', '$holiday_name')
ON CONFLICT
        ( date )
        DO UPDATE
SET    holiday_name = holiday.holiday_name || ', '  || excluded.holiday_name;
```

- OUTPUT (conflict on 2019-01-09):

date	holiday_name
2018-06-21	Volvo
2017-04-14	Chevrolet
2020-09-29	Pontiac
2018-04-26	Chevrolet
2019-03-05	Oldsmobile
2019-08-04	Lexus
2020-11-04	Bentley
2019-10-29	Oldsmobile
2020-05-21	Nissan
2019-01-09	Dodge, Benz

Update City Population

Abstract Code:

- Click on the ***Update City Population*** button from the Main Menu.
- Click on the ***Find City*** button: find the number of populations on a particular city, display CityName and CityPopulation from the CITY table.
- Click on ***Update City Population*** botton and type in the updated number.
    - Enter *Date*('$Date')
    - Enter *CityPopulation*('$CityPopulation')

```
UPDATE city
SET    city_population = '$CityPopulation'
WHERE  state = '$state'
    AND city_name = '$city_name'
```

	state [PK] character varying (50)	city_name [PK] character varying (50)	city_population integer
1	NY	New York City	4768503
2	NY	Syracuse	8884406
3	GA	Atlanta	3059488
4	MI	Ann Arbor	3635119
5	GA	Duluth	3291916
6	MI	Grand Rapids	8031583
7	GA	Macon	9411111
8	GA	Lawrenceville	5505344
9	MA	Watertown	2457074
10	MA	Waltham	10970015
11	NY	Buffalo	4407856
12	NY	Jamaica	5823348
13	MA	Boston	1482087
14	NY	Rochester	2838114
15	NY	Albany	10000

1- Generate Category Report

Abstract Code:

- User clicked on the Category ***Report*** button from the Main Menu.
- Run the **Generate Category Report** task:
    - For each category_name in Category, find all matching productID using the InCategory table.
    - For each productID, find all the matching regular_price using the Product table.
    - Group by category_name with aggregate functions of count, min, avg, and max on regular_price.
        - For each category_name:
            - count: count the number products that belong to the category_name.
            - min: the minimum regular_price in the category_name.
            - avg: the sum of regular_price divided by the count of regular_price in the category_name.
            - max: the maximum regular_price in the category_name.
    - Sort the rows by category_name ascending.

- Display the Category Report.

```
SELECT G.category_name,
       Count(*),
       Min(G.regular_price),
       Avg(G.regular_price),
       Max(G.regular_price)
FROM   (SELECT T.category_name,
               P.regular_price
        FROM   (SELECT C.category_name,
                       I.productid
                FROM   category AS C
                       LEFT JOIN incategory AS I
                              ON C.category_name = I.category_name) AS T
               LEFT JOIN product AS P
                      ON T.productid = P.productid) AS G
GROUP  BY category_name
ORDER  BY G.category_name ASC;
```

OUTPUT:

category_name	count	min	avg	max
Couches	2	3	4	5
Movies	2	3	6.5	10
Outdoor Furniture	5	3	10	20
Pots and Pans	1	12	12	12
Toy	1	NULL	NULL	NULL

2- Generate Actual vs Predict Revenue for Couches and Sofas Report

Abstract Code:

- User clicks on ***Couches and Sofas Report*** from Main Menu.
- Run **Generate Actual vs Predict Revenue for Couches and Sofas Report** task:
  - Query information about the product (couches and sofas) using product ID.
  - Filter all the products that belong to couches and sofas based on CATEGORY.Name.

- Gather information of DATE.Date of transaction, DATE.Date of discount, TRANSACTION.quantity, and PRODUCT.RegularPrice and “Actual Revenue/Predicted Revenue” for each product, calculate actual revenue and predicted revenue.
    - If no discount is applied to the product at the specified date:
        - Revenue is Actual Revenue, which is calculated as the product of TRANSACTION.quantity times PRODUCT.RegularPrice.
    - Else:
        - Revenue is Predicted Revenue, which is calculated as 75% of the product of TRANSACTION.quantity times PRODUCT.DiscountPrice.
- Calculate revenue difference as subtraction of predicted revenue from actual revenue.

- Display report of Actual Revenue and Predicted Revenue of the Product, for product with revenue difference higher than $5000 (positive or negative) in descending order of revenue difference, including product ID, product name and retail price, total quantity of item sold, quantity of item sold at regular price, quantity of item sold at discounted price and difference between actual revenue and predicted revenue.

```
SELECT Sum(p.regular_price * quantity * 0.75) - Sum(( CASE
                                                      WHEN (
                d.productid = p.productid
                AND d.date = t.date ) THEN d.discount_price
                                                      ELSE p.regular_price
                                                    END ) * quantity) AS
       difference,
       p.product_name                                                 AS
       product_name,
       p.productid                                                  AS
       product_ID,
       p.regular_price                                              AS
       retail_price,
       Sum(t.quantity)                                              AS
       total_quantity
FROM   TRANSACTION AS t
       LEFT JOIN product AS p
              ON t.productid = p.productid
       LEFT JOIN discount AS d
              ON d.productid = p.productid
       LEFT JOIN incategory AS ic
              ON ic.category_name = 'Couches'
                 OR ic.category_name = 'Sofa'
GROUP  BY p.product_name,
          p.productid
HAVING Sum(p.regular_price * quantity * 0.75) - Sum(( CASE
                                                      WHEN (
                d.productid = p.productid
```

```
        AND d.date = t.date ) THEN d.discount_price
                                   ELSE p.regular_price
                                 END ) * quantity) < -5000
    OR Sum(p.regular_price * quantity * 0.75) - Sum((
       CASE
       WHEN ( d.productid =
            p.productid
            AND d.date = t.date ) THEN d.discount_price
       ELSE p.regular_price
                                   END ) * quantity) >
       5000
ORDER  BY difference DESC;
```

OUTPUT:

Data Output | Explain | Messages | Notifications

	difference double precision	product_name character varying (50)	product_id character varying (50)	retail_price double precision	total_quantity bigint
1	-32008	E	ID.5	3	42784
2	-45770	B	ID.2	5	36680
3	-62104	A	ID.1	10	24880
4	-100404	C	ID.3	12	33500
5	-303240	D	ID.4	20	60876

3- View Store Revenue by Year by State

Abstract Code:

- Click on ***View Store Revenue by Year by State*** botton on Main Menu.
- Run the view report task on screen:
    - Drop down displayed for States:
        - Users select a specific state('$state').
            - combine the Store, City tables to select all the stores that are in the selected state.
            - Combine the Store, Transaction, Product and Discount tables to calculate the revenue for each transaction from Transaction.quantity * (regular_price or discount_price if discount_price is not NULL)
            - Group the revenue by year and store_number as total revenue.
    - Sort the report first by year in ascending order and then by revenue in descending order.
- Display the Store Revenue by Year by State report showing the store ID, store address, city name, sales year, and total revenue.

```
SELECT s.store_number,
```

```
        address,
        city_name,
        Extract(year FROM t.date) AS Year,
        Sum(quantity * ( CASE
                           WHEN d.discount_price IS NULL THEN regular_price
                           WHEN d.discount_price IS NOT NULL THEN
                           d.discount_price
                         END ))   AS Revenue
FROM    store AS s
        natural JOIN city
        natural JOIN TRANSACTION AS t
        natural JOIN product AS p
        LEFT OUTER JOIN discount AS d
                     ON t.productid = d.productid
                       AND t.date = d.date
WHERE  city_name IN (SELECT city_name
                     FROM   city
                     WHERE  state = '$state')
GROUP  BY s.store_number,
          year
ORDER  BY year,
          revenue DESC;
```

OUTPUT with ‘NY” as INPUT:

	store_number [PK] character varying (50)	address character varying (50)	city_name character varying (50)	year double precision	revenue double precision
1	No.13	666 Dummy Street	Syracuse	2015	966
2	No.15	666 Dummy Street	New York City	2015	827
3	No.4	666 Dummy Street	Buffalo	2015	590
4	No.3	666 Dummy Street	Jamaica	2015	579
5	No.14	666 Dummy Street	Albany	2015	570
6	No.1	666 Dummy Street	Rochester	2015	110
7	No.13	666 Dummy Street	Syracuse	2016	1485
8	No.15	666 Dummy Street	New York City	2016	1178
9	No.14	666 Dummy Street	Albany	2016	1163
10	No.4	666 Dummy Street	Buffalo	2016	817

4- Generate Outdoor Furniture on Groundhog Day vs Average Sale Report

Abstract Code:

- Click on ***View Outdoor Furniture Category Sales on Groundhog Day Report*** button from the Main Menu.
- Run the **Generate Outdoor Furniture on Groundhog Day vs Average Sale Report** task:
    - Run count all products subtask: count all products which belong to the "Outdoor Furniture" category from the Transaction table and Category table.
    - Run Find all years subtask: find all years which have Outdoor Furniture sold.
    - For each Year:
        - Count the total number of quantities which Date falls in this year as TotalNumber.
        - Calculate the AveNumber: TotalNumber / 365.
        - Count the total number of quantities which Date is on February 2nd this year as GroundNum.
        - If no sales made on that year, return year on the year column, and 0 for all columns.
    - Sort the report on Year ascending order.
- Display Year, TotalNumber, AverageNumber, and GroundhogDayNumber in Outdoor Furniture on Groundhog Day vs Average Sale Report.

```
SELECT a.year,
    avenumber,
    groundhognum
FROM   (SELECT Extract(year FROM t.date) AS Year,
           Sum(quantity) * 1.0 / 365 AS AveNumber
     FROM   TRANSACTION AS t
          natural JOIN product
          natural JOIN incategory
     WHERE  productid IN (SELECT productid
                  FROM   incategory
                  WHERE  category_name = 'Outdoor Furniture')
     GROUP  BY year
     ORDER  BY year) a
    LEFT JOIN (SELECT Sum(quantity)         AS Groundhognum,
                  Extract(year FROM date) AS year
```

```
        FROM   TRANSACTION
        WHERE  Extract(month FROM date) = 2
            AND Extract(day FROM date) = 2
        GROUP  BY date
        ORDER  BY Extract(year FROM date)) b
    ON a.year = b.year;
```

OUTPUT:

	year double precision	avenumber numeric	groundhognum bigint
1	2015	5.4054794520547945	40
2	2016	7.4630136986301370	31
3	2017	7.1178082191780822	44
4	2018	9.5068493150684932	55
5	2019	6.6520547945205479	27
6	2020	7.5561643835616438	26
7	2021	1.2383561643835616	19

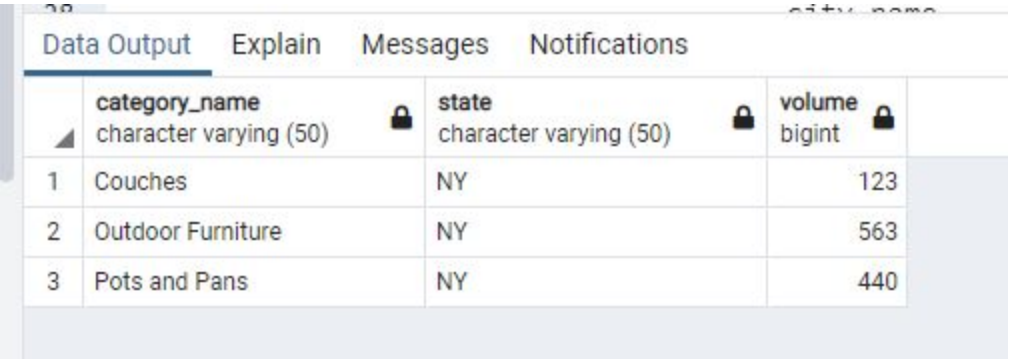
5- Show State with Highest Volume for Each Category by Month

Abstract Code:

- Click on the ***State with Highest Volume for Each Category by Month*** button from the Main Menu.
- Enter *Date*('$Year$Month').
- Run the **Show State with Highest Volume for Each Category by Month** task:
    - Find the quantity of the selected month in each store of each product by combining the Transaction table.
    - Find the category_name for each product combining Product and InCategory.
    - Find store location for each transaction using City.
    - Group the count of transactions for each product by CATEGORY.category_name and City.
    - Find the HighestVolume product in each category in all states.
    - Sort the rows by CategoryName ascending.
- Display monthly HighestVolume, CategoryName and State in State with Highest Volume for Each Category by Month.

```
SELECT category_name,
       state,
       volume
FROM  (SELECT category_name,
              state,
              volume,
              Rank()
                OVER (
                  partition BY category_name
                  ORDER BY volume DESC ) AS Rank
       FROM   (SELECT category_name,
                      Sum(quantity)AS Volume,
                      state
               FROM  (SELECT category_name,
                             quantity,
                             store_number
                      FROM  (SELECT category_name,
                                    quantity,
                                    store_number,
                                    Extract(year FROM T.date) AS YEAR,
                                    Extract(month FROM T.date)AS month
                             FROM   incategory AS I
                                    INNER JOIN TRANSACTION AS T
                                            ON I.productid = T.productid)AS A
                      WHERE  year = $year
                             AND month = $month)AS C
                     INNER JOIN (SELECT S.store_number,
                                        city_name,
                                        state
                                 FROM   store AS S
                                        INNER JOIN TRANSACTION AS T
                                                ON
                                        S.store_number = T.store_number)AS
                                L
                             ON C.store_number = L.store_number
               GROUP  BY category_name,
                         state
               ORDER  BY category_name ASC) a) a
WHERE  rank = 1
```

OUTPUT:



Data Output | Explain | Messages | Notifications

	category_name character varying (50)	state character varying (50)	volume bigint
1	Couches	NY	123
2	Outdoor Furniture	NY	563
3	Pots and Pans	NY	440

6- Show Revenue by Population Report

Abstract Code:

- Click on the ***Revenue by Population Report*** button from the Main Menu.
- Run the ***Show Revenue by Population Report*** task:
  - Find the yearly revenue in each store by combining the Transaction table.
  - Find store location for each transaction using City.
  - Calculate the yearly revenue for each city..
  - Find the city.size for each depending on following the conditions:
    - Small (population <3,700,000)
    - Medium (population >=3,700,000 and <6,700,000)
    - Large (population >=6,700,000 and <9,000,000)
    - Extra Large (population >=9,000,000)
  - Combine the yearly revenue by the city.size.
  - Sort the rows by year ascending and sort the columns by city.size ascending.
- Display Revenue, City.Size and Year in Revenue by Population Report.

```
SELECT *
FROM   crosstab ( 'WITH CityRevenue(year, city_size, revenue) AS (SELECT   year,
       city_size,
       Sum(revenue)AS revenue
FROM    (
            SELECT year,
                 city_name,
                 state,
                 revenue
            FROM  (
                        SELECT    Extract(year FROM t.date) AS year,
                              store_number,
                              quantity*(
                              CASE
```

```
                                WHEN d.discount_price IS NULL THEN p.regular_price
                                WHEN d.discount_price IS NOT NULL THEN d.discount_price
                            END)       AS revenue
                    FROM      TRANSACTION AS t
                    LEFT JOIN discount    AS d
                    ON        t.productid=d.productid
                    AND       t.date=d.date
                    LEFT JOIN product AS p
                    ON        p.productid=t.productid )AS a natural
            JOIN   city                                AS c) AS f natural
JOIN
        (
            SELECT city_name,
                state,
                CASE
                    WHEN city_population<3700000 THEN "small"
                    WHEN city_population>=3700000
                    AND    city_population<6700000 THEN "medium"
                    WHEN city_population>=6700000
                    AND    city_population<9000000 THEN "large"
                    WHEN city_population>=9000000 THEN "extra_large"
                END AS city_size
            FROM   city) AS c
GROUP BY year,
        city_size
ORDER BY year)
SELECT year, city_size, revenue FROM CityRevenue ORDER BY 1,2' ) AS PIVOT(year double
PRECISION, small DOUBLE PRECISION, medium DOUBLE PRECISION, large DOUBLE PRECISION,
extra_large DOUBLE PRECISION);
```

OUTPUT:

Data Output | Explain | Messages | Notifications

	year double precision	small double precision	medium double precision	large double precision	extra_large double precision
1	2015	21170	21170	52925	63510
2	2016	33722	33722	84305	101166
3	2017	27604	27604	69010	82812
4	2018	30886	30886	77215	92658
5	2019	20432	20432	51080	61296
6	2020	27640	27640	69100	82920
7	2021	4248	4248	10620	12744

7- Generate Childcare Sales Volume Report

Abstract Code:

- Click on the ***Childcare Sales Volume Report*** button from the Main Menu.
- Run the **Generate Childcare Sales Volume Report** task:
    - Query the maximum_time of Childcare for each store in the Store table.
    - Calculate the revenue for all stores in the past 12 months.
        - Filter out all the transactions before “2020-04-01” as of today (“2021-03-20”).
        - For each productID in the Transaction table, find the regular_price by looking up the Product table.
        - Find all the discount records by looking up the Discount table.
        - Create a new column named “revenue” by multiplying the quantity and the price (discount_price if available).
    - Aggregate the total revenue by month and maximum_time.
        - Group by the current table on month and maximum_time.
        - Using aggregate function SUM to get the total revenue for every month with different childcare time.
    - Create a pivot table with month as rows, childcare time limit as columns, with the cell value the total revenue for a specific month and childcare time.
    - Sort the table with the month in ascending order (‘Apr’, ‘May’, ‘Jun’, etc.) and the childcare time limit in ascending order (‘no_childcare’, ‘max_15’, ‘max_30’, ‘max_60’).
- Display the Childcare Sales Volume Report.

```
SELECT  *
FROM    crosstab (
        'WITH aggtable(mon, maximum_time, sum) AS
                        (SELECT Sales.mon, Sales.maximum_time, SUM(Sales.revenue)
                        FROM
                                (SELECT to_char(PriceChild.date,"Mon") as mon,
        PriceChild.maximum_time,
                                PriceChild.quantity*LEAST(PriceChild.regular_price, D.discount_price)
                                AS revenue
                                FROM
                                        (SELECT Price.date, Price.productID, Price.regular_price,
                                        Price.quantity, Store.maximum_time
                                        FROM
                                                (SELECT T.store_number, T.productID, T.date,
        T.quantity,
                                                P.regular_price
                                                FROM Transaction AS T
                                                LEFT JOIN Product AS P
                                                ON T.productID = P.productID
                                                WHERE T.date >=
                                                date_trunc("month", CURRENT_DATE) - INTERVAL "1
        year") AS Price
```

```
                                        LEFT JOIN Store
                                        ON Price.store_number = Store.store_number) AS PriceChild
                                LEFT JOIN Discount AS D
                                ON PriceChild.productID = D.productID AND PriceChild.date = D.date)
        AS Sales
                        GROUP BY Sales.mon, Sales.maximum_time)
                SELECT mon, maximum_time, sum
                FROM AggTable
                ORDER BY 1,2'
) AS aggtable(mon text, no_childcare float8, max_15 float8, max_30 float8, max_60 float8)
ORDER BY
    CASE
        WHEN mon = 'Jan' THEN 1
        WHEN mon = 'Feb' THEN 2
        WHEN mon = 'Mar' THEN 3
        WHEN mon = 'Apr' THEN 4
        WHEN mon = 'May' THEN 5
        WHEN mon = 'Jun' THEN 6
        WHEN mon = 'Jul' THEN 7
        WHEN mon = 'Aug' THEN 8
        WHEN mon = 'Sep' THEN 9
        WHEN mon = 'Oct' THEN 10
        WHEN mon = 'Nov' THEN 11
        WHEN mon = 'Dec' THEN 12
    END;
```

OUTPUT:

mon	no_childcare	max_15	max_30	max_60
Jan	40	777	280	135
Feb	75	478	120	50
Mar	180	65	852	185
Apr	612	420	134	669
May	195	560	359	NULL
Jun	234	460	898	72
Jul	284	192	250	27
Aug	83	205	48	NULL
Sep	666	174	154	390
Oct	457	102	167	528
Nov	210	601	146	542
Dec	210	60	216	NULL

8- Generate Restaurant impact on Category Sales Report

Abstract Code:

- Click on the ***Restaurant impact on Category Sales Report*** button from the Main Menu.
- Run the **Generate Restaurant Impact on Category Sales Report** task.
    - Left join Store and Transaction table on store_number to get all store's restaurant status and the productID and it;s quantity
    - Join InCategory table with above table on productID, get the category of the product and quantity sold for each product
    - Select all column names from above table, and categorize the Null value from store_type as "Non-Restaurant"
- Display Restaurant impact on Category Sales Report showing quantity sold by store type and product category.

```
SELECT category,
    CASE
      WHEN store_type IS NULL THEN 'Non-Restaurant'
      ELSE store_type
    end AS Store_Type,
    quantity_sold
FROM   (SELECT category_name AS Category,
```

```
        CASE
          WHEN restaurant IS TRUE THEN 'Restaurant'
        end          AS Store_Type,
        Sum(quantity) AS Quantity_Sold
FROM   incategory
        LEFT JOIN(SELECT restaurant,
                         store.store_number,
                         quantity,
                         productid
                  FROM   store
                         INNER JOIN transaction
                                 ON store.store_number =
                                    transaction.store_number) a
               ON incategory.productid = a.productid
GROUP  BY category_name,
          restaurant
ORDER  BY category_name) a;
```

OUTPUT:

	category character varying (50)	store_type text	quantity_sold bigint
1	Couches	Non-Restaurant	2406
2	Couches	Restaurant	956
3	Movies	Non-Restaurant	2231
4	Movies	Restaurant	852
5	Outdoor Furniture	Non-Restaurant	6198
6	Outdoor Furniture	Restaurant	2085
7	Pots and Pans	Non-Restaurant	1284
8	Pots and Pans	Restaurant	391

9- Generate Advertising Campaign Analysis Report

Abstract Code:

- Click on the ***Advertising Campaign Analysis Report*** button from the main menu.
- Run the **Generate Advertising Campaign Analysis Report** task.
    - Combine the Campaign table and BelongTo table to get the date that campaign happens
    - Combine above view with Transaction table to get the date and quality of transaction, to see if the transaction happen within or outside of the campaign date:

- Cast the above view to get the count of sold within campaign and outside campaign by productID.
- Count the sum of products sold within and outside of campaign date, get the difference in quantity. Then join with the Product table to get the product name.
- Sort the report by the difference in descending order, keep the top 10 rows. Union with report by difference in ascending order keeping top 10 rows-- end up with top 10 and bottom 10 differences.

- Display Advertising Campaign Analysis Report showing table of product ID, product name, quantity sold during campaign and outside campaign and difference.

```
(SELECT b.productid,
        product_name,
        Sum(inside)              AS sold_during_campaign,
        Sum(outside)             AS sold_outside_campaign,
        Sum(inside) - Sum(outside) AS difference
 FROM   (SELECT Max(CASE
                     WHEN camp = 'inside' THEN quantity
                     ELSE 0
                   end) AS inside,
                Max(CASE
                     WHEN camp = 'outside' THEN quantity
                     ELSE 0
                   end) AS outside,
                productid,
                a.date
         FROM   (SELECT CASE
                          WHEN transaction.date BETWEEN
                               a.start_date AND a.end_date THEN
                          'inside'
                          ELSE 'outside'
                        end            AS camp,
                        productid,
                        transaction.date AS date,
                        quantity
                 FROM   transaction
                        LEFT JOIN (SELECT date,
                                          start_date,
```

```
                        end_date
                    FROM   belongto
                INNER JOIN campaign
                    ON belongto.campaign_description =
                     campaign.campaign_description) a
                   ON transaction.date = a.date) AS a
      GROUP  BY productid,
                camp,
                a.date) b
     LEFT JOIN product
          ON b.productid = product.productid
GROUP  BY b.productid,
       product_name
ORDER  BY difference DESC
-- our dummy data can only show top 2 and bottom 2 due to limited number of data
-- if using our data to run please change the LIMIT from 10 to 2.
LIMIT  10)
UNION
(SELECT b.productid,
     product_name,
     Sum(inside)              AS sold_during_campaign,
     Sum(outside)             AS sold_outside_campaign,
     Sum(inside) - Sum(outside) AS difference
FROM   (SELECT Max(CASE
                 WHEN camp = 'inside' THEN quantity
                 ELSE 0
               end) AS inside,
            Max(CASE
                 WHEN camp = 'outside' THEN quantity
                 ELSE 0
               end) AS outside,
            productid,
            a.date
     FROM   (SELECT CASE
                WHEN transaction.date BETWEEN
                    a.start_date AND a.end_date THEN
               'inside'
```

```
                ELSE 'outside'
            end             AS camp,
            productid,
            transaction.date AS date,
            quantity
        FROM   transaction
            LEFT JOIN (SELECT date,
                        start_date,
                        end_date,
                        campaign.campaign_description
                    FROM   belongto
                INNER JOIN campaign
                    ON belongto.campaign_description =
                    campaign.campaign_description) a
                ON transaction.date = a.date) AS a
    GROUP  BY productid,
        camp,
        a.date) b
    LEFT JOIN product
        ON b.productid = product.productid
GROUP  BY b.productid,
    product_name
ORDER  BY difference ASC
-- our dummy data can only show top 2 and bottom 2 due to limited number of data
-- if using our data to run please change the LIMIT from 10 to 2.
LIMIT  10)
ORDER  BY difference DESC;
```

OUTPUT:

	productid character varying (50)	product_name character varying (50)	sold_during_campaign bigint	sold_outside_campaign bigint	difference bigint
1	ID.1	A	21	1511	-1490
2	ID.5	E	0	1505	-1505
3	ID.3	C	0	1667	-1667
4	ID.2	B	19	1751	-1732