Phase 2 Team032 – CS6400 Spring 2021

Katie Barthelson, Erica Chia, Samantha Chu, Chris Edwards, Brandon Jackson

Table of Contents

Abstract Code

View Main Menu

Abstract Code

```
SELECT COUNT(campaign_description) AS campaign_count FROM advertising_campaign;

SELECT COUNT(store_number) AS food_store_count FROM store WHERE has_restaurant;

SELECT COUNT(pid) AS product_count FROM product;

SELECT COUNT(store_number) AS childcare_store_count FROM childcare_store;

SELECT COUNT(store_number) AS store_count FROM store;
```

Update Holidays

Abstract Code

User clicked on ***Update Holidays*** button from **Main Menu**.

- Display all Date and HolidayNames from Holiday entities
- While no buttons are pushed, do nothing.
- Click ***Save Holidays*** button – **Edit Holidays**.

```
SELECT date, holiday_names FROM holiday;

UPDATE holiday
SET holiday_names = '$HolidayNames'
WHERE date = '$Date';
```

Update City Population

Abstract Code

User clicked on ***City Population*** button from **Main Menu**.

- Display all CityNames, State, and Populations from City entities
- While no buttons are pushed, do nothing.
- Click ***Save City Populations*** button – **Edit Populations**.

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

UPDATE city
SET population = '$Population'
WHERE city_name = '$CityName' AND state = '$State';
```

Report 1 – Category Report

Abstract Code

User clicked on ***View Report 1*** button from **Main Menu**.

- For each Category, return:
  - CategoryName
  - The count of Product entities in the Category
  - Minimum RetailPrice across all products in the Category
  - Average RetailPrice across all products in the Category
  - Maximum RetailPrice across all products in the Category
- Sort results by CategoryName, ascending

```
SELECT
  c.category_name,
  COUNT(p.pid) AS product_count,
  MIN(p.retail_price) AS min_retail_price,
  ROUND(AVG(p.retail_price), 2) AS avg_retail_price,
  MAX(p.retail_price) AS max_retail_price
FROM category AS c
INNER JOIN belongs_to AS bt ON bt.category_name = c.category_name
INNER JOIN product AS p ON p.pid = bt.pid
GROUP BY c.category_name
ORDER BY c.category_name ASC;
```

Report 2 – Actual versus Predicted Revenue for Couches and Sofas

Abstract Code

User clicked on ***View Report 2*** button from **Main Menu**.

- For each Product in the "Couch" or "Sofa" Categories, return:
  - PID
  - ProductName
  - RetailPrice
  - Sum of all Quantities ever Sold ($TotalSold)
  - Sum of all Quantities Sold when the Product was DiscountedOn the sale Date
  - Sum of all Quantities Sold when the Product was *not* DiscountedOn the sale Date
  - Sum of all associated SaleTotals ($ActualRevenue)
  - Sum of $TotalSold * 0.75 * RetailPrice ($PredictedRevenue)
  - Difference of $ActualRevenue - $PredictedRevenue ($RevenueDifference).
- Filter the results to rows where the absolute value of $RevenueDifference > $5000
- Sort the results by $RevenueDifference, descending.

```
WITH sales_with_totals AS (
  SELECT
    s.store_number,
    s.date,
    s.pid,
    s.quantity,
```

```
    COALESCE(dis.discount_price, p.retail_price) AS sale_price,
    COALESCE(dis.discount_price, p.retail_price) * s.quantity AS sale_total,
    (NOT dis.discount_price IS NULL) AS is_discounted
  FROM sold AS s
  INNER JOIN date AS d ON d.date = s.date
  INNER JOIN product AS p ON p.pid = s.pid
  LEFT JOIN discounted_on AS dis ON dis.date = s.date AND dis.pid = s.pid
)
SELECT
  p.pid,
  p.product_name,
  p.retail_price,
  SUM(s.quantity) AS total_quantity_sold,
  SUM(CASE WHEN s.is_discounted THEN s.quantity ELSE 0 END) AS discounted_quantity_sold,
  SUM(CASE WHEN (NOT s.is_discounted) THEN s.quantity ELSE 0 END) AS
not_discounted_quantity_sold,
  SUM(s.sale_total) AS actual_revenue,
  ROUND(SUM(s.quantity) * p.retail_price * 0.75, 2) AS predicted_revenue,
  SUM(s.sale_total) - ROUND(SUM(s.quantity) * p.retail_price * 0.75, 2) AS revenue_difference
FROM product AS p
INNER JOIN sales_with_totals AS s ON s.pid = p.pid
INNER JOIN belongs_to AS bt ON bt.pid = p.pid
WHERE bt.category_name = 'Couches and Sofas'
GROUP BY p.pid, p.product_name, p.retail_price
HAVING ABS(SUM(s.sale_total) - ROUND(SUM(s.quantity) * p.retail_price * 0.75, 2)) > 5000
ORDER BY revenue_difference DESC;
```

Report 3 – Store Revenue by Year by State

Abstract Code

User clicked on ***View Report 3*** button from **Main Menu**.

- Return unique States from City Entities
- User clicked on a State
    - For all Stores in the State selected, return:
        - StoreNumber
        - StreetAddress
        - CityName
        - Sum of SaleTotal* for sales within each Date year ($YearSaleTotal)
        - Date year
    - Sort results by Date year, ascending, then by YearSaleTotal, descending.

*SaleTotal is a derived attribute that calculates the total revenue of a sale by first looking to see if the Product was DiscountedOn the sale Date, then multiplying the Quantity by the DiscountPrice if on sale, and the RetailPrice otherwise.

```
SELECT DISTINCT state FROM city;
```

```
WITH sales_with_totals AS (
 SELECT
   s.store_number,
   s.date,
   s.pid,
   s.quantity,
   COALESCE(dis.discount_price, p.retail_price) AS sale_price,
   COALESCE(dis.discount_price, p.retail_price) * s.quantity AS sale_total,
   (NOT dis.discount_price IS NULL) AS is_discounted
  FROM sold AS s
  INNER JOIN date AS d ON d.date = s.date
  INNER JOIN product AS p ON p.pid = s.pid
  LEFT JOIN discounted_on AS dis ON dis.date = s.date AND dis.pid = s.pid
)
SELECT
  st.store_number,
  st.street_address,
  st.city_name,
  SUM(so.sale_total) AS yearly_revenue,
  EXTRACT(YEAR FROM so.date) AS sale_year
FROM store AS st
INNER JOIN sales_with_totals AS so ON so.store_number = st.store_number
WHERE st.state = '$State'
GROUP BY st.store_number, st.street_address, st.city_name, sale_year
ORDER BY sale_year ASC, yearly_revenue DESC;
```

Report 4 – Outdoor Furniture on Groundhog Day?

Abstract Code

User clicked on ***View Report 4*** button from **Main Menu**.

- For each Date year, return:
  - Date year
  - Sum of Quantity Sold for Products in the "Outdoor Furniture" Category ($TotalUnitsSold),
  - Average Quantity sold per day assuming a 365-day year ($AvgQuantity = $TotalUnitsSold / 365)
  - Quantity Sold on the Date February 2nd for Products in the "Outdoor Furniture" category
- Sort the result by Date year, ascending.

```
SELECT
  EXTRACT(YEAR FROM s.date)::int as year,
  SUM(quantity) AS total_units_sold,
  ROUND(SUM(quantity)/365.0, 2) AS avg_quantity,
  SUM(CASE WHEN EXTRACT(MONTH FROM s.date) = 2 AND EXTRACT(DAY FROM s.date) = 2 THEN
quantity ELSE 0 END) AS groundhog_day_sales
FROM Sold AS s
```

```
LEFT JOIN belongs_to AS bt ON s.PID = bt.PID
WHERE bt.category_name = 'Outdoor Furniture'
GROUP BY EXTRACT(YEAR FROM s.date)
ORDER BY Year ASC;
```

Report 5 – State with Highest Volume for Each Category

Abstract Code

User clicked on ***View Report 5*** button from **Main Menu**.

- User chooses year and month from Date entities
- For each Category in specified year and month return:
  - CategoryName
  - The State with largest sum of Quantity sold in that Category
  - Number of Products sold by Stores in that State
- Sort output by CategoryName, ascending

```
SELECT DISTINCT EXTRACT(YEAR FROM date)::int AS year FROM date;
SELECT DISTINCT TO_CHAR(date, 'Month') AS month, EXTRACT(MONTH FROM date) AS month_num
FROM date;

SELECT
  bt.category_name,
  st.state,
  SUM(s.quantity)
FROM belongs_to bt
LEFT JOIN sold s ON s.pid = bt.pid
LEFT JOIN store st ON st.store_number = s.store_number
WHERE EXTRACT(YEAR FROM s.date) = '$year' AND EXTRACT(MONTH FROM s.date) = '$month_num'
GROUP BY bt.category_name, st.state
ORDER BY bt.category_name;
```

Report 6 – Revenue by Population

Abstract Code

User clicked on ***View Report 6*** button from **Main Menu**.

- For each Date year and PopulationCategory* return:
  - The Date year
  - PopulationCategory
  - The sum of SaleTotals for that Date year in Cities with that PopulationCategory ($YearCategoryRevenue)
- Sort results by Date year, ascending, and by PopulationCategory in the order [“Small”, “Medium”, “Large”, “Extra Large”]
- Pivot the results such that the PopulationCategories form the columns and the Date years form the row index, with one $YearCategoryRevenue per each PopulationCategory in each Date year’s row.

*PopulationCategory is a derived attribute on City, calculated using the logic: if Population < 3.7 million, then “Small”; if Population >= 3.7 million and < 6.7 million, then “Medium”; if Population >= 6.7 million and < 9 million then “Large”; otherwise, if Population >= 9 million, “Extra Large”

```
WITH population_category AS (
  SELECT
    city_name,
    state,
    population,
    (CASE
      WHEN population < 3700000 THEN 'Small'
      WHEN population < 6700000 THEN 'Medium'
      WHEN population < 9000000 THEN 'Large'
      ELSE 'Extra Large'
    END) AS category
  FROM city
), sales_with_totals AS (
  SELECT
    s.store_number,
    s.date,
    s.pid,
    s.quantity,
    COALESCE(dis.discount_price, p.retail_price) AS sale_price,
    COALESCE(dis.discount_price, p.retail_price) * s.quantity AS sale_total,
    (NOT dis.discount_price IS NULL) AS is_discounted
  FROM sold AS s
  INNER JOIN date AS d ON d.date = s.date
  INNER JOIN product AS p ON p.pid = s.pid
  LEFT JOIN discounted_on AS dis ON dis.date = s.date AND dis.pid = s.pid
)
SELECT
  EXTRACT(YEAR FROM s.date)::int AS year,
  SUM(CASE WHEN pc.population_category = 'Small' THEN s.sale_total ELSE 0 END) as small,
  SUM(CASE WHEN pc.population_category = 'Medium' THEN s.sale_total ELSE 0 END) as medium,
  SUM(CASE WHEN pc.population_category = 'Large' THEN s.sale_total ELSE 0 END) as large,
  SUM(CASE WHEN pc.population_category = 'Extra Large' THEN s.sale_total ELSE 0 END) as
extra_large
FROM sales_with_totals s
INNER JOIN store AS st ON st.store_number = s.store_number
INNER JOIN population_category pc ON pc.city_name = st.city_name AND pc.state = st.state
GROUP BY EXTRACT(YEAR FROM s.date)
ORDER BY year;
```

Report 7 – Childcare Sales Volume

Abstract Code

User clicked on ***View Report 7*** button from **Main Menu**.

- For each Date month in the last 12 months, return:
  - Date month
  - TimeLimit Minutes (or, for non-ChildcareStores, "No childcare") ($ChildcareCategory)
  - Sum of SaleTotal* for that Date month in stores with that $ChildcareCategory ($MonthCategorySales)
- Pivot the results such that the $ChildcareCategories form the columns and the Date months form the row index, with one $MonthCategorySales value per each $ChildcareCategory in each Date month's row

*SaleTotal is a derived attribute that calculates the total revenue of a sale by first looking to see if the Product was DiscountedOn the sale Date, then multiplying the Quantity by the DiscountPrice if on sale, and the RetailPrice otherwise.

```
SELECT minutes FROM time_limit;

SELECT * FROM crosstab('
WITH sales_with_totals AS (
  SELECT
    s.store_number,
    s.date,
    s.pid,
    s.quantity,
    COALESCE(dis.discount_price, p.retail_price) AS sale_price,
    COALESCE(dis.discount_price, p.retail_price) * s.quantity AS sale_total,
    (NOT dis.discount_price IS NULL) AS is_discounted
  FROM sold AS s
  INNER JOIN date AS d ON d.date = s.date
  INNER JOIN product AS p ON p.pid = s.pid
  LEFT JOIN discounted_on AS dis ON dis.date = s.date AND dis.pid = s.pid
), stores_childcare_categories AS (
  SELECT
    s.store_number,
    COALESCE(tl.minutes::text, ''No childcare'') AS childcare_category
  FROM store s
  LEFT JOIN childcare_store cs ON s.store_number = cs.store_number
  FULL OUTER JOIN time_limit tl ON cs.minutes = tl.minutes
), stores_childcare_months AS (
  SELECT DISTINCT
    s.store_number,
    s.childcare_category,
    date_trunc(''month'', d.date) as month
```

```
  FROM stores_childcare_categories s
  CROSS JOIN date d
  WHERE d.date > date_trunc(''month'', CURRENT_DATE) - INTERVAL ''1 year''
)
SELECT
  TO_CHAR(scm.month, ''Month'') || '' '' || EXTRACT(YEAR FROM scm.month) AS month,
  scm.childcare_category,
  COALESCE(SUM(s.sale_total), 0.00) AS sale_total
FROM stores_childcare_months scm
LEFT JOIN sales_with_totals s ON s.store_number = scm.store_number AND date_trunc(''month'',
s.date) = scm.month
GROUP BY scm.month, scm.childcare_category
ORDER BY scm.month DESC, scm.childcare_category;')
AS ct(sale_month TEXT, $time_limit_1 NUMERIC, $time_limit_1 NUMERIC, ..., "No childcare"
NUMERIC);
```

Report 8 – Restaurant Impact on Category Sales

Abstract Code

User clicked on ***View Report 8*** button from **Main Menu**.

- For each Category, return:
  - Category
  - Sum of Quantity Sold in Stores where HasRestaurant = True ($Restaurant)
  - Sum of Quantity Sold in Stores where HasRestaurant = False ($NonRestaurant)
- Un-pivot the Restaurant and NonRestaurant columns into a "StoreType" column and a "TotalQuantitySold" column, grouped by Category.
- Sort the results by Category, ascending and StoreType, ascending

```
WITH store_type_counts AS (
  SELECT
    bt.category_name,
    SUM(CASE WHEN st.has_restaurant THEN quantity ELSE 0 END) AS restaurant,
    SUM(CASE WHEN NOT st.has_restaurant THEN quantity ELSE 0 END) AS non_restaurant
  FROM store AS st
  LEFT JOIN sold AS s ON st.store_number = s.store_number
  LEFT JOIN belongs_to AS bt ON s.pid = bt.pid
  GROUP BY bt.category_name
)
SELECT
  category_name,
  'Restaurant' AS store_type,
  restaurant AS quantity_sold
FROM store_type_counts
UNION
SELECT
```

```
  category_name,
  'Non-restaurant' AS store_type,
  non_restaurant AS quantity_sold
FROM store_type_counts
ORDER BY category_name, store_type;
```

Report 9 – Advertising Campaign Analysis

Abstract Code

User clicked on ***View Report 9*** button from **Main Menu**.

- Construct $CampaignSaleQuantities dataset by returning, for each Product:
  - PID
  - ProductName
  - Sum of Quantity Sold on Dates where an AdvertisingCampaign was ActiveOn that Date and the Product was DiscountedOn that Date ($SoldDuringCampaign)
  - Sum of Quantity Sold on Dates where an AdvertisingCampaign was *not* ActiveOn that Date and the Product was DiscountedOn that Date ($SoldOutsideCampaign)
  - The difference between $SoldDuringCampaign and $SoldOutsideCampaign ($AdDifference)
- Sort $CampaignSaleQuantities dataset by $AdDifference, descending. Limit the output to only return 10 results ($CSQTop10).
- Sort $CampaignSaleQuantities dataset by $AdDifference, ascending. Limit the output to only return 10 results ($CSQBottom10). Re-sort this result by $AdDifference, descending.
- Return the union of $CSQTop10 and $CSQBottom10

```
WITH campaign_sale_quantities AS (
  SELECT
    p.pid,
    p.product_name,
    SUM(CASE WHEN ao.date IS NOT NULL THEN Quantity ELSE 0 END) AS sold_during_campaign,
    SUM(CASE WHEN ao.date IS NULL THEN Quantity ELSE 0 END) AS sold_outside_campaign,
    SUM(CASE WHEN ao.date IS NOT NULL THEN Quantity ELSE 0 END) - SUM(CASE WHEN ao.date IS
NULL THEN Quantity ELSE 0 END) AS ad_difference
  FROM product AS p
  INNER JOIN sold AS s ON s.pid = p.pid
  LEFT JOIN active_on AS ao ON ao.date = s.date
  GROUP BY p.pid, p.product_name
), top10 AS (
  SELECT
    *
  FROM campaign_sale_quantities
  ORDER BY ad_difference DESC
  LIMIT 10
), bottom10 AS (
  SELECT
    *
  FROM campaign_sale_quantities
```

```
  ORDER BY ad_difference ASC
  LIMIT 10
)
SELECT * FROM top10
UNION
SELECT * FROM bottom10
ORDER BY ad_difference DESC;
```