Table of Contents

1 Manufacturer's Product Report

Abstract Code

- User clicked the ***Manufacturer Product Report*** button:
- Run the **Manufacturer Product Report** task**:**
    - Group by manufacturer, select manufacture, count total number of product, calculate the average, max and min price.
    - Order the result by calculated average price descending, only list top 100
    -
```
SELECT Manufactuer_Name,COUNT(PID) AS num_product,
AVERAGE(retail_price) AS avg_price,
MIN(price) AS min_price, MAX(price) AS max_price
FROM Product
GROUP BY Manufactuer_Name
ORDER BY avg_price DESC
LIMIT 100
```

- For each row of the master table, when mouse is over the manufacturer name, there is additional information such as maximum discount shown.
```
SELECT Manufactuer_Name, Cap_Discount
FROM Manufactuer_Name
WHERE Manufactuer_Name=$Manufactuer_Name
```
- For each row in the master table, if click manufacturer name, a new table pops up with product information of that manufacturer:
    - Based on the selected PID, find the category from the category table. Concatenated if multiple categories are returned.
    - Order the product by retail price descending.
```
SELECT p.PID, CONCAT(c.Categories_Name, ",") AS category,
‘p.Product_Name, p.Retail_Price
FROM Product p, Category c
WHERE p.PID=c.PID AND p.Manufactuer_Name=$Manufactuer_Name
GROUP BY p.PID
ORDER BY Retail_Price DESC
```
- it shows the report table on the screen. Each row represents a manufacturer, column represents the related information.

2 Category Report

Abstract Code

- User clicked the ***Category Report*** button:
- Run the **Category Report** task:
    - Based on category table, merge with product and manufacturer table by PID as key to get retail price and manufacturer information. Group by category, count unique PID, count unique manufacturer and calculate average retail price.

```
SELECT c.Categories_Name, COUNT(c.PID) AS cnt_product,
COUNT(DISTINCT p.Manufactuer_Name) AS cnt_manufacturer,
AVERAGE(p.Retail_Price) AS avg_price
FROM Category c, Product p,
WHERE c.PID=p.PID
GROUP BY c.Categories_Name
ORDER BY c.Categories_Name
```

3 Actual versus Predicted Revenue for GPS units

Abstract Code

- User clicked the ***Actual versus Predicted Revenue for GPS units*** button:
- Run the **Actual versus Predicted Revenue for GPS units** task:
    - Find Product ID in the GPS Category from Category table.

```
SELECT PID AS GPS_id FROM Category WHERE Categories_Name = 'GPS';
```

    - Find retail price from Product table using selected Product ID.

```
SELECT Product_Name, Retail_Price FROM Product WHERE PID = GPS_id;
```

    - Find related transactions from *Transaction* table using selected Product ID.

```
SELECT PID, Quantity, Transaction_Date FROM Transaction WHERE PID IN
(GPS_id);
```

    - Create Boolean attribute is_sale to indicate whether a sale or not and make transaction price as sale_price if it is on sale, otherwise transaction price is retail price.

```
CASE WHEN t.Transaction_Date=s.Sales_Date THEN 1
ELSE  0
END AS  is_sale,
CASE WHEN t.Transaction_Date=s.Sales_Date THEN s.Sales_Price
ELSE p.Retail_Price
END AS Transaction_Price
FROM Transaction t
INNER JOIN Product p ON t.PID = p.PID
LEFT JOIN Sales s ON t.PID = p.PID
```

- Summarize information by product and generate **report**

- Sum up total quantity, sale quantity if is_sale=1, regular quantity if is_sale=0, and sum up the transaction price * total quantity to get actual revenue, and sum up the retail price *(quantity if is_sale=0, quantity*0.75 if is_sale=1) as predict revenue
- Get the revenue difference as predict revenue – actual revenue
- Get the revenue difference as predict revenue – actual revenue

```
SELECT PID, Product_Name, Retail_Price, SUM(Quantity) AS Total_sold,
SUM(Quantity*is_sale) AS discount_sold, SUM(Quantity*(1-is_sale)) AS
retail_sold,  SUM(Transaction_Price*Quantity) AS actual_revenue,
SUM(Retail_Price*(CASE WHEN is_sale=1 THEN 0.75*Quantity ELSE Quantity
END)) AS predict_revenue, actual_revenue-predict_revenue AS diff_revenue
FROM ( the above codes)
```

- For **report**, only keep PID with revenue difference>5000, sort by revenue difference in descending order.

```
HAVING diff_revenue>5000
ORDER BY diff_revenue DESC;
```

Summary of SQL:

```
SELECT PID, Product_Name, Retail_Price, SUM(Quantity) AS Total_sold,
SUM(Quantity*is_sale) AS discount_sold, SUM(Quantity*(1-is_sale)) AS retail_sold,
SUM(Transaction_Price*Quantity) AS actual_revenue,  SUM(Retail_Price*(CASE WHEN
is_sale=1 THEN 0.75*Quantity ELSE Quantity END)) AS predict_revenue,
actual_revenue-predict_revenue AS diff_revenue
FROM (
        SELECT t.Transaction_Date AS transaction_date, t.Quantity AS quantity, p.Retail_Price
        AS retail_price, t.PID AS product_Pid, p.Product_Name AS product_name,
        CASE WHEN t.Transaction_Date=s.Sales_Date THEN 1
        ELSE  0
        END AS  is_sale,
        CASE WHEN t.Transaction_Date=s.Sales_Date THEN s.Sales_Price
        ELSE p.Retail_Price
        END AS Transaction_Price
        FROM Transaction t
        INNER JOIN Product p ON t.PID = p.PID
        LEFT JOIN Sales s ON t.PID = p.PID AND
        t.Transaction_Date=s.Sales_Date
        WHERE t.PID IN (SELECT PID FROM Category WHERE
                        Categories_Name = 'GPS')
)
   GROUP BY PID
   HAVING diff_revenue>5000
   ORDER BY diff_revenue DESC;
```

4 Store Revenue by Year by State

Abstract Code

- User clicked the ***Store Revenue by Year by State*** button. Run the **Store Revenue by Year by State** task:
- Enrich the transaction with the city and state
    - Merge city and store table by key: city to get the state
    - Merge the transaction and the new store table by key: StoreNumber. So for each transaction, it has the sold store,city and the state
- Enrich the transaction with price
    - Merge the new transaction table with product table by key: PID, to get retail price
    - Merge the transaction table with sales table by key: PID, to get sales price
    - Create transaction price as sales price if there is a sale, otherwise regular price

      City c INNER JOIN Store o ON
      c.City_Name = o.City_Name
      INNER JOIN Transaction t ON
      t.Store_Number = o.Store_Number
      INNER JOIN Product p ON
      t.PID = p.PID
      INNER JOIN Sales s ON
      t.PID = s.PID;
      CASE WHEN t.Transaction_Date = s.Sales_Date
      THEN s.Sales_Price ELSE p.Retail_Price END
      AS transaction_price
- Summarize the result
    - Based on the enriched transaction table, group by store and year, sum up the revenue calculated by product of transaction price times quantity.
    - Also keep the store ID, address, city name and state
    - Sort the result by year ascending and revenue descending

      Select t.Store_Number, t.Street_Address, t.City_Name, year(t.Transaction_Date) AS sale_year, t.Quantity * transaction_price AS revenue
      GROUP BY sale_year, t.Store_Number
      ORDER BY sale_year DESC, revenue DESC
      WHERE o.State_Name = `$State_Name`
- Generate **report** with a **drop down box** to select state
    - For each state, list all the stores with store ID, address, city name, sales year and total revenue. Order has been sorted in the previous step

Summary of SQL:

SELECT t.Store_Number, t.Street_Address, t.City_Name, year(t.Transaction_Date) AS sale_year, (t.Quantity*(CASE WHEN t.Transaction_Date=sa.Sales_Date THEN sa.Sales_Price ELSE t.Retail_Price END)) AS revenue

```
From City c INNER JOIN Store o ON
c.City_Name = o.City_Name
INNER JOIN Transaction t ON
t.Store_Number = o.Store_Number
INNER JOIN Product p ON
t.PID = p.PID
INNER JOIN Sales s ON
t.PID = s.PID
GROUP BY sale_year, t.Store_Number
ORDER BY sale_year DESC, revenue DESC
WHERE o.state = `$StateName`
```

5 Air Conditioners on Groundhog Day?

Abstract Code

- User clicked the ***Air Conditioners on Groundhog Day*** button. Run the **Air Conditioners on Groundhog Day** task:
- Select transactions only on category “air conditioning”
    - Merge with transaction table and category table by key: PID, only select category as “air conditioning”
    - Get the transaction year, and also add new attribute is_groundhog if it is on Feb 2nd.

      ```
      SELECT t.Transaction_Date, t.Quantity
      FROM Transaction t INNER JOIN Categories c ON t.PID = c.PID
      WHERE c.Categories_Name = “air conditioning”;
      year(t.Transaction_Date) AS sale_year, CASE WHEN
      month(t.Transaction_Date) = 2 AND day(t.Transaction_Date) = 2 THEN 1 ELSE
      0 END AS is_groundhog;
      ```
- Summarize the result and generate **report**
    - Group by year, sum up the total sales quantity divided by 365 to get average sales per day, sum up quantity*is_groundhog to get sales quantity at groundhog day
    - Sort by year ascending

      ```
      SELECT SUM(t.Quantity)/365 AS avg_quantity, SUM(quantity*is_groundhog) AS
      groundhog_quantity, sale_year
      GROUP BY sale_year
      ORDER BY sale_year;
      ```

Summary of SQL:

```
SELECT SUM(Quantity)/365 AS avg_quantity, SUM(Quantity*(CASE WHEN
month(t.Transaction_Date)=2 AND day(t.Transaction_Date)=2 THEN 1 ELSE 0 END)) AS
groundhog_quantity, sale_year
```

```
FROM
(SELECT year(t.Transaction_Date) AS sale_year, t.Quantity, t.Transaction_Date
 FROM Transaction t, Categories c
 ON t.PID=c.PID
 WHERE c.Categories_Name="air conditioning")
 GROUP BY sale_year
 ORDER BY sale_year;
```

6 State with Highest Volume for each Category

Abstract Code

User clicked the ***State with Highest Volume for each Category*** button:

- Run the **State with Highest Volume for each Category**:
- Choose Date to get *year and month.*
- Create a master table which includes all the necessary attributes for the transaction
    - First merge store and city table by key: city to get each store's state, then merge with the transaction table by: key store number. So each transaction record has sold store and state.
    - Get the year and month for each transaction

```
CREATE TABLE Newtransaction AS(
SELECT t.PID, concat(year(t.transaction_date), '-',  month(t.transaction_date)) as year_month, t.quantity, c.state, s.store_number, s.stress_address
FROM Store s
INNER JOIN City c ON s.city_name= c.city_name
INNER JOIN Transaction t ON t.store_number = s.store_number);
```

- Get transaction information for category table
    - Merge the category table with new transaction table by PID. So each category has the transaction information for all the products it sold.

```
CREATE TABLE Newcategory AS(
SELECT n.*, ca.categories_name
FROM Newtransaction n
INNER JOIN Categories ca ON n.PID = ca.PID);
```

- Summarize all the information and generate **report**

- Based on the enriched category table, group by year and month, as well as category and state, calculate the summation of sold quantity.Sort the output by sold quantity descending

```
CREATE TABLE summary AS(
SELECT  s.year_month, s.state, s.categories_name, s.total_quantity,
max(s.total_quantity) OVER (PARTITION BY s.categories_name,s.year_month)
as max_sale
FROM
(SELECT year_month, state, categories_name, SUM(quantity) AS total_quantity
FROM Newcategory
GROUP BY categories_name, year_month, state) s);
```

- Only select the highest state.
- Create summary view, which has a drop down window to select year and month, and the report will show all the categories' names, the highest sold state and quantity.Sort by **category name** ascending

```
SELECT categories_name, state, total_quantity
FROM summary
WHERE total_quantity=max_sale AND year_month=$year_month
ORDER BY categories_name
```

- Drill-down details with store and manager information
  - Each row of the report has a hyperlink to include store **IDs, addresses, and cities, managers names and email addresses**
  - Sort **store IDs** ascending, parent header include **category, year/month, state**

```
SELECT nc.store_number, nc.categories_name, nc.year_month,
nc.stress_address, m.manager_email, m.first_name, m.last_name
From Newcategory nc, Manager m
WHERE nc.store_number=m.store_number
AND nc.state=$state AND nc.categories_name=$category AND
nc.year_month=$year_month
ORDER BY nc.store_number
```

7 Revenue by Population

Abstract Code

- User clicked the ***Revenue by Population*** button. Run the **Revenue by Population** task:
- Transaction with the city and Population
  - Based on the criterion, create the attribute CitySize of small, medium, large and extra large based on Population. Merge city and store table by key: city to enrich store table.

```
CREATE TABLE newstore AS(
SELECT (CASE WHEN c.population<3,700,000 THEN 'Small'
WHEN c.population>=3,700,000 AND c.population<6,700,000 THEN 'Medium'
WHEN c.population>=6,700,000 AND c.population<9,000,000 THEN 'Large'
WHEN c.population>=9,000,000 THEN 'Extra Large'  END) AS city_size,
c.city_name, c.state, s.store_number,s.phone_number,  s.street_address
FROM City c
INNER JOIN Store s
ON s.city_name = c.city_name);
```

- Merge the transaction and the new store table by key: StoreNumber. So for each transaction, it has the sold city and the citySize,
- Merge the new transaction table with product table by key: PID, to get retail price
- Merge the transaction table with sales table by key: PID and Date, to get sales price
- Create transaction price as sales price if there is a sale, otherwise regular price

```
CREATE TABLE Newtransaction AS(
SELECT ns.city_size, ns.city_name, ns.state,
ns.store_number,ns.phone_number,  ns.street_address, t.PID,
t.transaction_date, t.quantity,year(t.transaction_date) AS year
(CASE WHEN t.transaction_date=s.sales_date THEN s.sales_price
ELSE p.retail_price END) AS transaction_price
FROM Newstore ns
INNER JOIN Transaction t  ON  ns.store_number = t.store_number
INNER JOIN Product p ON t.PID=p.PID
LEFT JOIN Sales s ON t.PID=p.PID AND t.transaction_date=s.sales_date);
```

- Summarize the result and create **pivot table**
  - Based on the enriched transaction table, group by year and citySize, sum up the product of transaction price and quantity to get the revenue.
  - Make a pivot table with year as row, citysize as column. Each cell represents the total revenue.

```
SELECT SUM(quantity*transaction_price) AS revenue, year, city_size
FROM Newtransaction
GROUP BY year, city_size
```