Abstract Code with SQL

Team 017

Table of Contents

Notation:

- The Clickable Button is : "Underline"
    - Example: "Button"
- The Task is :**Bold**
    - Example: **Task Name**
- The Entity is in: Blue ink
    - Example: Entity Name
- The variable name is: '$VarName'
    - Example: '$Name'

Display Main Menu

- Show "Statistics Reports", "Maintain Holidays", and "Update Populations" tabs.
- The "Statistics Reports" tab is the default tab for initial viewing.
- Upon the "Statistics Report" tab:
    - A clickable button named "Report category" allows the user to jump to the **Report Category** task.
    - A clickable button named "Report outdoor furniture on Groundhog Day" allows the user to jump to the **Report Air Conditioners on Groundhog Day** task.
    - A clickable button named "Report restaurant impact on category sales" allows the user to jump to the **Report Restaurant Impact on Category Sales** task.
    - A clickable button named "View childcare sales volume" allows the user to jump to the **View Child Care Sales Volume** task.
    - A clickable button named "View highest volume for each category" allows the user to jump to the **Select Year and Month** task.
    - A clickable button named "View actual revenue vs predicated" allows the user to jump to the **View Actual Revenue vs Predicated** task.
    - A clickable button named "Description and active date of advertising campaign" allows the user to jump to the **Advertising Campaign Analysis** task.
    - A clickable button named "View revenue by state by year" allows the user to jump to the **Revenue by State by Year** task.
    - A clickable button named "View revenue by population" allows the user to jump to the **Select Population Category** task.
- Upon the "Maintain Holidays" tab:
    - A clickable button named "Upload holidays" allows the user to jump to the **Upload Holidays** task.
- Upon the "Update Population" tab:
    - A clickable button named "Edit city population" allows the user to jump to the **Edit City Population** task.

Report 1 - Category Report

Abstract Code:

- In the "Main Menu", users can click on "Category Report" button on "Statistics Reports".
- Run the "**Category Report**" task as SQL below.
- When finished, the user can select the next action from the **Main Menu**.

```
SELECT phc.category_name AS category_name,
COUNT(p.pid) AS product_count,
AVG(p.retail_price) AS avg_price
FROM Product AS p
INNER JOIN ProductHasCategory AS phc ON phc.pid = p.pid
GROUP BY category_name
```

Report 2 – Actual versus Predicted Revenue for Couches and Sofas

Abstract Code:

- The report can be executed by clicking the button "Actual vs Predicted Revenue" on **main menu** in displayed statistics information, the system will perform all the calculation relating to subtasks.
    - The user should click the tab "Select a Product" to specify the Product type.
    - The user should use "**Date Range**" to select a time period. Clicking the button should display a calendar.
    - The user can execute the report by clicking "View Report". The elements include: Product ID, Product Name, Retail Price, total number of units ever sold, total number of units sold at discount, total number of units sold at retail price, actual revenue in all sales, predicted revenue with no discount, and actual vs predicted revenue differences.
        - For each selected Date and Product, find Price and Quantity.
        - Calculate Actual Revenue using Price*Quantity.
        - Display table 1-"Actual Revenue" with element Date, Price, Quantity and Actual Revenue.
        - Create table 2-"Predicted Revenue" and fill the elements Date, Price, Quantity. The data is the same as table 1.

        - Replace all Sale Price with Retail Price.
        - Apply Quantity with multiplier 0.75.
        - Calculate “Predicted Revenue”:
        - Quantity = Quantity*0.75
        - Predicted Revenue = Price(Retail Price)*Quantity
    - Compute the difference between “Actual Revenue” and “Predicted Revenue”, by taking Actual Revenue - Predicted Revenue.
    - Display and sort table 3-”Differences in Actual and Predicted Revenue” in descending order only when absolute value for difference in 1,4 is greater than $5000.
- When finish the task, click on “<u>Main Menu</u>” button and return to the main menu.

```
SELECT
        P.pid, P.product_name, P.retail_price, B.discountSold, A.total_sold, A.actual_revenue,
        A.predicted_revenue, A.actual_revenue - A.predicted_revenue AS diff
FROM (
        SELECT T.pid, P.product_name, P.retail_price, SUM(T.sold_quantity) AS total_sold,
        SUM(P.retail_price * T.sold_quantity * 0.75) AS predicted_revenue, SUM(P.retail_price *
        T.sold_quantity * (DiscountPrice.discount_price)) AS actual_revenue
        FROM Transaction T JOIN Product P ON T.pid = P.pid
        JOIN DiscountPrice D ON T.pid = d.pid AND T.date_time = D.date_time
        WHERE P.pid IN(
                SELECT PHC.pid
                FROM ProductHasCategory PHC
                WHERE PHC.category_name = “Couches and Sofas”)
        GROUP BY T.pid ) AS A,
( SELECT T.pid AS pid, SUM(T.sold_quantity) AS DiscountSold
FROM Transaction AS T
INNER JOIN Product P ON T.pid = P.pid
INNER JOIN DiscountPrice D ON T.pid = D.pid
WHERE P.pid IN(
        SELECT PHC.pid
        FROM ProductHasCategory PHC
        WHERE PHC.category_name = “Couches and Sofas”)
GROUP BY T.pid) B
WHERE diff > 5000
ORDER BY diff DESC
```

Report 3 Store revenue by year by state

Abstract Code:

- Execute the report generation by clicking the task "**Store Revenue by Year by State**" on the **Main Menu** in displayed statistics information.
- Select the state through querying database
- The states available are presented in drop-down box

```
SELECT state_name AS StateName FROM City
```

- After selecting the State information, there are two choices for display results: view by sales year (ascending order) or view by revenue (descending order).
    - For the selected state, calculate the annual revenue of all cities in that state.
    - For each City in that state, find all Stores in that city; and then calculate the annual revenue for all the stores in that city.
    - In the selected state, sum all the revenues covering multiple sales years of the cities.

```
CREATE VIEW RevenuebyYearbyState AS
SELECT
        St.store_id AS store_id,
        St.store_address AS store_address,
        St.city_name AS city_name,
        DP.date_time AS date_time,
        T.quantity,
        DP.discount_price,
        P.retail_price*(1 - DP. discount_price) * T.quantity AS total
FROM Store AS St
INNER JOIN Transaction AS T ON St.store_id = T.store_id
INNER JOIN DiscountPrice AS DP ON DP.pid = T.pid AND DP.date_time = T.date_time
JOIN Product AS P ON P.pid= T.pid
WHERE St.state_name = 'State';
```

- If user choose to display by year, then the system can display store information in the state with listed tasks: store ID, store address, city name, total revenue and sales year in ascending order;

```
SELECT
      store_id,
      store_address,
      city_name,
    YEAR(date_time) AS SalesYear,
    SUM(total) AS TotalRevenue
FROM RevenuebyYearbyState
GROUP BY store_id, SalesYear
ORDER BY SalesYear ASC;
```

- If users choose to display by total revenue, then the system can display store information in the state with tasks: store ID, store address, city name, sales year and total revenue in descending order.

```
SELECT
    store_id,
    store_address,
    city_name,
    YEAR(date_time) AS SalesYear,
    SUM(total) AS TotalRevenue
FROM RevenuebyYearbyState
GROUP BY store_id, SalesYear
ORDER BY SalesYear ASC;
```

- When users finish the task, click the "**Main Menu**" button and return to the **Main Menu**.

Report 4 Outdoor Furniture on Groundhog Day

Abstract Code:

- User clicked on "**Outdoor Furniture on Groundhog Day**" button from the **Main Menu**.
- Run the **Outdoor Furniture on Groundhog Day** Task.
  - Look up the category identifier for the "Outdoor furniture" category from the Category table.
  - Look up all the Products' PID in "Outdoor furniture" category using category name as an identifier.
  - For all Products found above, find all the Transaction records associated with the Product using pid as an identifier, and lookup the quantity sold.
  - Look up the TimeDate for all the Transaction records found above.
  - Group all records by year in TimeDate.
  - Rank by year in ascending order.
    - Display Year
  - Sum all the quantities sold for each year.
  - Display total number of items sold in the outdoor furniture category that year.
  - Divide the sum from the previous step by 365.

        - Display average number of units sold per day in that year
    - For all Transaction records found previously, filter the date to 2020-02-02.
    - Sum all the quantities sold from the previous step.
        - Display Count

```
SELECT
    YEAR(date_time),
    SUM(quantity),
    SUM(quantity)/365,
    (SELECT SUM(quantity)
    FROM Transaction
    JOIN ProductHasCategory ON Transaction.pid = ProductHasCategory.pid
    WHERE ProductHasCategory.category_name = "outdoor furniture" AND
    Transaction.date_time = (select date_time from Holiday where Holiday.holiday_name =
    "GroundHog Day" or date_time = Get.date('2020-02-02'))
    )
FROM Transaction
JOIN ProductHasCategory ON Transaction.pid = ProductHasCategory.pid
WHERE ProductHasCategory.category_name = "outdoor furniture"
GROUP BY YEAR(Transaction.date_time)
ORDER BY TEAR(Transaction.date_time) ASC;
```

- User clicked on "**Main Menu**" button: go back to **Main Menu** page

Report 5 – View State with Highest Volume of Each Category

Abstract Code:

- When the user clicked on the "View state highest volume for each category" button on the "statistics reports" tab from the "**Main menu**".

```
CREATE VIEW JoinedTable AS
SELECT
        ProductHasCategory.category_name AS category_name,
        Transaction.pid AS pid,
        Store.state_name AS state_name,
        Transaction.sold_quantity AS sold_quantity,
        Transaction.date AS date
FROM Transaction
INNER JOIN ProductHasCategory ON Transaction.pid = ProductHasCategory.pid
INNER JOIN Store ON Store.store_id = Transaction.store_id;

CREATE VIEW filterByYearAndMonth AS
SELECT
        category_name,
        state_name,
        SUM(sold_quantity) AS sum_uantity
FROM JoinedTable
WHERE YEAR(date) = '$userInputYear' AND MONTH(date)='$userInputMonth'
GROUP BY category_name, state_name;

SELECT
        filter.category_name AS category_name,
        filter.state_name AS state_name,
        filter.sum_quantity AS max_quantity
FROM filterByYearAndMonth AS filter
WHERE (category_name, sum_quantity) IN (
        SELECT category_name, MAX(sum_quantity)
        FROM filterByYearAndMonth
        GROUP BY category_name
        );
```

- Return for each category details
- When ready, the user clicks on the "Main Menu" button: go back to the main menu page.

Report 6 Revenue by Population

Abstract Code:

- Users can click the button of "**View Revenue by Population**" on "Statistics Reports" from the **Main Menu**, then the report will display a PIVOT table.
- Use city Population divide all cities into four groups:
    - o Small = CityPopulation < 3,700,000;
    - o Medium = CityPopulation>=3,700,000 and <6,700,000;
    - o Large = CityPopulation >=6,700,000 and <9,000,000;
    - o Extra Large = CityPopulation >=9,000,000

```
DROP VIEW IF EXISTS 'Population_Category';
CREATE VIEW Population_Category AS
SELECT city_name, state_name,
CASE
WHEN population < 3700000 THEN ‘Small’
WHEN 3700000 <= population < 6700000 THEN ‘Medium’
WHEN 6700000 <= population < 9000000 THEN ‘Large’
ELSE ‘Extra Large’
END AS “Population_Category”
FROM City;
```

Report 7 – Childcare Sales Volume

Abstract Code:

- When the user clicked on the “View child care sales volume” button on the “Category Reports” from the “Main Menu”, run the **View child care sales volume** task.
- The “View child care sales volume” task will create and display a table which lists the sales volume of each month with different child care limits at each store in the 12 months. ○ From a UI point of view, there is a table to show the static result.
    - Look up and find the store ID which provides the child care service.
    - If a store does not provide childcare service, it will show “No childcare” or “0 min”.
    - Find all the Sale record associated with the Product using PID as an identifier, and
    - lookup the quantity sold.
    - Find the total number of sales volume/month (monthly sale volume) corresponding to all categories and each store with different childcare time limit category.
    - Sum all the quantities sold of each category in each month.
    - Sort by time (month) with ascending order and display the months.

```
SELECT SUM(A.amount)
FROM(
  SELECT YEAR(A.date) AS year, MONTH(A.date) AS month, DAY(A.date) AS day, A.pid, A.store_id,
A.amount, C.time_limit
 FROM
        ((SELECT T.pid, T.date_time, T,store_id, T.sold_quantity * D.discount_price as amount
        FROM Transaction T
        INNER JOIN DiscountPrice D ON D.pid = T.pid AND D.date_time = T.date_time)
        UNION
        (SELECT T.pid, T.date, T,store_id, T.sold_quantity * P.retail_price as amount
        FROM Transaction T
        INNER JOIN Product P ON P.pid = T.pid
        WHERE pid NOT IN (SELECT DiscountPrice.pid FROM DiscountPrice ))) A
 LEFT JOIN ChildCare C ON C.store_iD = A.store_id
 WHERE A.date_time > DATEADD(year, -1, GETDATE()))
GROUP BY month, time_limit
```

Report 8 – Report Restaurant Impact on Category Sales

Abstract Code:

- When the user clicked on the "Report restaurant impact on category sales" button on the "Category Reports" from the "Main Menu", run the **Report restaurant impact on category sales** task.
- **Report restaurant impact on category sales** task will:
  - Display the categories which are selling in the Store. Check in the Restaurant entity to determine if a Store provides dining services.
  - Display if the Store has the restaurant service or not.
  - Find all products belonging to each Category, then get the Transaction record for all the Products.
  - Display the total number of items sold belonging to each Category under the condition of w/o Restaurant within one year.
  - Sorted by Category with ascending order.

```
SELECT
        ProductHasCategory.category_name AS Category,
        Store.restaurant AS Store_Type
        SUM(Transaction.sold_quantity) AS Quantity_Sold
FROM Transaction
INNER JOIN Store ON Transaction.store_id = Store.store_id
INNER JOIN ProductHasCategory ON Transaction.pid=ProductHasCategory.pid
GROUP BY Store_Type
ORDER BY Category;
```

- When ready, the user clicks on the "Main Menu" button: go back to the main menu page.

Report 9. Advertising Campaign Analysis

Abstract Code:

- The system will start to run the **Advertising Campaign Analysis** task when clicking on the "**Advertising Campaign Analysis**" tab.

- Query for all the products where the Product ID is unique.
- For every product we query:
  - Display the information for Product Name together with Product ID.
  - Display the discount price effect time.
- Sum up total sold with active advertising campaign based on discount effect TimeDate.
  - Sum up total sold when there is no advertising campaign, by excluding the effect discount price TimeDate.
  - Compare and display the two totals between with/without Advertising Campaign by taking the difference.
- Sort the total Transaction differences in descending order.
- The table will display the final report with features Product ID, Product Name, Sold During Campaign, Sold Outside Campaign and Differences.
- When users finish the task, click the "**Main Menu**" button and return to the **Main Menu**.

```
CREATE VIEW AdvertisingCampaignAnalysis AS
SELECT
    pid
    date_time,
    SUM(sold_quantity) AS sold_quantity with discount,
    (SELECT SUM(sold_quantity)
    FROM Transaction
    JOIN DiscountPrice ON Transaction.pid = DiscountPrice.pid
      JOIN Product ON Transaction.pid = Product.pid
    WHERE Transaction.date_time = DiscountPrice.date_time AND DiscountPrice.discount_price
<Product.retail_price)
    )
FROM Transaction
JOIN DiscountPrice ON Transaction.pid = DiscountPrice.pid
WHERE Transaction.date_time = DiscountPrice.date_time AND DiscountPrice.discount_price
<Product.retail_price
GROUP BY Transaction.date_time
ORDER BY Transaction.date_time ASC;

SELECT
    pid
    date_time,
    SUM(sold_quantity) AS sold_quantity without discount,
    (SELECT SUM(sold_quantity)
    FROM Transaction
    JOIN DiscountPrice ON Transaction.pid = DiscountPrice.pid
      JOIN Product ON Transaction.pid = Product.pid
    WHERE Transaction.date_time = DiscountPrice.date_time AND DiscountPrice.discount_price
=Product.retail_price)
    )
FROM Transaction
JOIN DiscountPrice ON Transaction.pid = DiscountPrice.pid
WHERE Transaction.date_time = DiscountPrice.date_time AND DiscountPrice.discount_price
=Product.retail_price
GROUP BY Transaction.date_time
ORDER BY Transaction.date_time ASC;
```

Edit City Population

Abstract Code:

- Click the button of "Edit city population" in the **Main Menu** to run the **Edit City Population** task.
- Two drop-lists, and an input box for entry are needed for this task UI.
- The first drop-list contains a list of states available for selection. The user can select one of the States and click on "Select State".
- Once the state is selected, the user can select the city from the second drop-list and click on "Select City". The list of the City is within the selected state.

```
SELECT city_name FROM City
WHERE state_name = '$stateName';
```

- The input box can be used to enter a new number for the population of the selected City. The input number '$populationNumber' expects to be a non-negative number. Any other data type will lead to an error message with the indication of the expected data type.

```
SET population = '$populationNumber'
WHERE state_name = '$stateName' AND city_name = '$cityName'
LIMIT 1;
```

- A City should be selected before applying for the number from the input box. Otherwise, a warning will be issued for any violation.
- After finishing the task, click on the "Apply" button to refresh.
- Click "**Main Menu**" button to go back to the **Main Menu** page

Upload holidays

Abstract Code:

- User clicked on the "**Upload holidays**" in the "**Main Menu**", the **upload holidays** task is run.
    - The Upload Holiday task has two subtasks: Display Existing Holidays and Upload Holiday.
- The Display Existing Holiday task shows a list of Holidays already in the database.
    - All the Holidays should be unique in the database.
    - A refresh button is needed for reloading the Holidays information whenever the holiday information has been uploaded.
    - A scroll bar is needed in case the list of existing Holidays is too long to display in the window.

```
SELECT holiday_name, date_time FROM Holiday;
```

- The Upload Holiday task allows the user to add a new Holiday into the database.
    - UI design: Two input boxes are designed for users to enter the new Date and holiday name. One button of "Apply" is designed to apply the changes to the database.
    - The input restrictions:
        - The format of TimeDate should be the same as the existing date in the database.
        - The name of the Holiday should be a non-zero-length string and unique.
    - Error message reminders:
        - Any format errors of date and Holiday names will lead to an error message reminder.
        - If the name of the Holiday is not unique, an error message will show up.
- After entering the new TimeDate and Holiday name, click "Apply" button to refresh.
    - Insert a new Holiday name on a new date;

```
INSERT INTO TimeDate(date_time)
VALUES ('2020-07-04');
INSERT INTO TimeDate(date_time)
VALUES ('2020-01-01');
INSERT INTO Holiday(date_time)
VALUES ('NEW YEAR', '2020-01-01');
```

    - Insert a new Holiday name on an existing date;

```
INSERT INTO Holiday(holiday_name, date_time)
```

```
SELECT ‘independence day’, date_time
FROM TimeDate
WHERE date_time=’2020-07-04’
LIMIT 1;
```

- Click “**Main Menu**” button to go back to the **Main Menu** page