S&E Technology Superstore Data Warehouse

Phase II

Project Report

Team024

[jchen857 jhu369 twu76 yshan34]

Table of Contents

Task Decomposition with Abstract Code

Main Menu

Abstract Code

- Display the count of stores from Store table.

```
SELECT COUNT(storeID) FROM Store;
```

- Display the count of manufacturer from manufacturer table.

```
SELECT COUNT(manufacturerID) FROM Manufacturer;
```

- Display the count of products from product table.

```
SELECT COUNT(DISTINCT pid) FROM Product;
```

- Show *"**Edit Holiday**", "**Edit Manager**", "**Edit Population**", "**View Manufacturer's Product Report**", "**View Category Report**", "**View Actual versus Predicted Revenue for GPS units Report**", "**View Store Revenue by Year by State Report**", "**View Air Conditioners on Groundhog Day Report**", "**View State with Highest Volume for each Category Report**", "**View Revenue by Population Report**", and "**Log out"*** tabs.
- Upon:
    - Click ***Edit Holiday*** button – Jump to the ***Edit Holiday*** task.
    - Click ***Edit Manager*** button – Jump to the ***Edit Manager*** task.
    - Click ***Edit Population*** button – Jump to the ***Edit Population*** task.
    - Click ***View Manufacturer's Product Report*** button – Jump to the ***View Manufacturer's Product Report*** task.
    - Click ***View Category Report*** button – Jump to *the **View Category Report*** task.
    - Click ***View Actual versus Predicted Revenue for GPS units Report*** button – Jump to the ***View Actual versus Predicted Revenue for GPS units Report*** task.
    - Click ***View Store Revenue by Year by State Report*** button – Jump to the ***View Store Revenue by Year by State Report*** task.
    - Click ***View Air Conditioners on Groundhog Day Report*** button – Jump to the ***View Air Conditioners on Groundhog Day Report*** task.
    - Click ***View State with Highest Volume for each Category Report*** button – Jump to the ***View State with Highest Volume for each Category Report*** task.

    - Click ***View Revenue by Population Report*** button – Jump to the ***View Revenue by Population Report*** task.
    - Click ***View Revenue by Population Report*** button – Jump to the ***View Revenue by Population Report*** task.
    - Click ***Log out*** button – Go back to **Login** form.

Edit Holiday

Abstract Code

- User clicked ***Edit Holiday*** button from **Main Menu**.
- Go to the default page for **Edit Holiday**
    - There is a ***View*** button for user to see the date holiday details, an ***Add*** button for user to add holidays, and an ***Update*** button for user to update holidays.
- When ***View*** button is pushed, pop up a window with a drop-down menu for date field and a holiday name field. The holiday name filed is empty. There are two buttons, ***Search*** and ***Cancel*** button.
    - If user selected one specific date '$date' through the system. When ***Search*** button is pushed, display the date selected by user and holiday name when it is not null and not blank at that date. If that date is not a holiday, show message "No holiday could be found at that date." at that window.

```
SELECT

        date,

        holiday_name

FROM Date

WHERE date = CAST('$date'  AS DATE)
AND LENGTH (holiday_name)  > 0 ;
```

    - If user selected two specific date '$start date' and '$end date'. If '$end date' selected by users is before '$start date'. A window will pop up display "Please select a date after start date." When ***Search*** button is pushed, display the dates that are holidays within that range as well as corresponding holiday names. If no holidays fall into that range, display message "No holiday could be found at that range." at that window.

```
SELECT
        date,
        holiday_name
FROM Date
WHERE date >= CAST('$start date' AS DATE)
AND date <= CAST('$end date' AS DATE)
AND LENGTH (holiday_name) > 0
ORDER BY date ASC;
```

  - If user push ***Cancel*** button, return to the default page for **Edit Holiday.**

- When ***Add*** button is pushed, pop up a window with a drop-down menu for date field and a holiday name field. The holiday name filed is empty.
- User can fill-in the information '$date' for the additional holiday '$holiday name' through system. There are two buttons, ***Save*** and ***Cancel*** button.
  - If user pushes ***Save*** button:
    - If user enter empty holiday name:
      - Stay at **Edit Holiday** page with error message, 'holiday name cannot be empty'.
    - Else:
      - If '$holiday name' already exists at this date, '$date'
        - Pop a window saying, "This holiday is already in the database" and go back to the default page.
      - If '$holiday name' doesn't exist at this date, '$date'
        - Set the holiday name for this date to be the concatenate of the original holiday name and the new holiday name.

```
UPDATE Date
SET holiday_name = CONCAT (holiday_name, ', ', '$holiday name')
WHERE date =CAST( '$date' AS DATE);
```

        - After inserting, return to the default page for **Edit Holiday**. Now the Date table has been updated and the list of holidays would not have the original holiday but the user-entered holiday.
    - If user push ***Cancel*** button, return to the default page for **Edit Holiday**
- When ***Update*** button is pushed, pop up a window with a drop-down menu for date field and a holiday name field. There is a ***Save*** button and a ***Cancel*** button. The holiday name is filed with corresponding holiday name at that date. User will fill desired '$holiday name' at '$date' selected in drop-down menu through system.
    - If the '$date' is not a holiday, when User selected a specific '$date' through the menu. display message "The date selected is not a holiday. Please add holiday first."
    - If the '$date' selected is a holiday and '$holiday name' is specified, after user pushes ***Save*** button. Then update Date table:
    - If user-entered holiday name is blank, it means holiday name is removed.
    - After updating, return to the default page for **Edit Holiday**.
    - If user push ***Cancel*** button, return to the default page for **Edit Holiday**

```
UPDATE Date

SET holiday_name = '$holiday name'

WHERE date =CAST( '$date' AS DATE);
```

- When ***Main Menu*** button is pushed:
    - Go to the **Main Menu** form.
- When ***Log out*** button is pushed:
    - Go to the **Login** form.

Edit Manager

Abstract Code

- User clicked on ***Edit Manager*** button from **Main Menu**.
- When ***View*** button is pushed:
    - If User enters the information '$manager name', '$manager email' or '$store number' in corresponding field at that page.

    - Display manager name, manager email address from ActiveManager and InActiveManager tables, and corresponding store number in ActiveManager table.

```
WITH CTE_Manager AS
(SELECT DISTINCT activeManagerID AS managerID, email, manager_name
FROM ActiveManager
UNION
SELECT DISTINCT inactiveManagerID AS managerID, email, manager_name
FROM InActiveManager
)
SELECT
        CM.manager_name,
        CM.email,
        S.store_number
FROM CTE_Manager AS CM
LEFT JOIN Manage AS M
        ON M.activeManagerID = CM.managerID
LEFT JOIN Store AS S
        ON S.storeID = M.storeID
WHERE CM.manager_name = '$manager name'
OR CM.email = '$manager email'
OR S.store_number = '$store number';
```

- When ***Add Manager*** button is pushed, jump to **Add Manager** Task.
- When ***Remove Manager*** button is pushed, jump to **Remove Manager** Task.
- When ***Assign Manager*** button is pushed, jump to **Assign Manager** Task.
- When ***Unassign Manager*** button is pushed, jump to **Unassign Manager** Task.
- When ***Main Menu*** button is pushed:
    - Go to the **Main Menu** form.
- When ***Log out*** button is pushed:
    - Go to the **Login** form.

Add Manager

Abstract Code

- User clicked on ***Add Manager*** button from **Edit Manager**.
- User enters '$manager name and '$manager email' input fields.
- When ***Save*** button is pushed:
    - If '$manager name' is blank:
        - Stay on **Add Manager** page with error message, 'Please fill in manager's name'.
    - If 'email address' does not follow the standard format:
        - Stay on **Add Manager** page with error message, 'Manager email is not in standard format'.
    - If '$manager email' already exists:
        - Stay on **Add Manager** page with error message, 'Manager email already exists'.

```
SELECT 'Manager email already exists'
WHERE EXISTS (
                SELECT  1
                FROM ActiveManager AS M
                WHERE M.email = '$manager email'
                )
OR EXISTS (
                SELECT  1
                FROM InActiveManager AS M
                WHERE M.email = '$manager email'
                );
```

-
    - Else:
        - Add '$manager email', and '$manager name' into InActiveManager table with success message.

```
INSERT INTO InActiveManager

(

manager_name,

email

)

VALUES ('$manager name', '$manager email');
```

- When ***Main Menu*** button is pushed:
  - Go to the **Main Menu** form.
- When ***Log out*** button is pushed:
  - Go to the **Login** form.

Remove Manager

Abstract Code

- User clicked on ***Remove Manager*** button from **Edit Manager**.
- User enters '$manager email' input field.
- When ***Save*** button is pushed:
  - If '$manager email' is blank:
    - Stay on **Remove Manager** page with error message, 'Please fill in the manager's email that you would like to remove'.
  - If '$manager email'is not in ActiveManager and InActiveManager tables.
    - Stay on **Remove Manager** page with error message, 'The Manager is not in system'.

```
SELECT 'Manager is not in system'
WHERE NOT EXISTS (
                SELECT 1
                FROM ActiveManager AS M
                WHERE M.email = '$manager email'
                )
AND NOT EXISTS (
                SELECT 1
                FROM InActiveManager AS M
                WHERE M.email = '$manager email'
                );
```

- If '$manager email' belongs to ActiveManager table, which means this manager is still assigned to store as an ActiveManager:
  - Stay on **Remove Manager** page with error message, 'Manager needs to be unassigned first'.

```
SELECT 'Manager needs to be unassigned first'
WHERE EXISTS (
                SELECT 1
                FROM ActiveManager AS M
                WHERE M.email = '$manager email'
                );
```

- Else:
  - Remove manager record from InActiveManager table based on '$manager email' with success message.

```
DELETE FROM InActiveManager

WHERE email = '$manager email';
```

- When ***Main Menu*** button is pushed:
    - Go to the **Main Menu** form.
- When ***Log out*** button is pushed:
    - Go to the **Login** form.

Assign Manager

Abstract Code

- User clicked on ***Assign Manager*** button from **Edit Manager**.
- User enters *manager email* ('$manager email') and *store* ('$store number') input fields.
- When ***Save*** button is pushed:
    - If '$manager email' is blank:
        - Stay on **Assign Manager** page with error message, 'Manager cannot be found, please enter an email first.'.
    - If '$manager email' is not in table InActiveManager and ActiveManager tables:
        - Stay on **Assign Manager** page with error message, 'manager cannot be found.'.

```
SELECT 'Manager cannot be found'

WHERE NOT EXISTS (

                SELECT  1

                FROM ActiveManager AS M

                WHERE M.email = '$manager email'

                )

AND NOT EXISTS (

                SELECT  1

                FROM InActiveManager AS M

                WHERE M.email = '$manager email'

                );
```

- If ‘$store number’ is blank:
  - Stay on **<u>Assign Manager</u>** page with error message, ‘Please enter a store number’.
- If ‘$store number’ not in store table:
  - Stay on **<u>Assign Manager</u>** page with error message, ‘Store cannot be found’.

```
SELECT 'Store cannot be found'

WHERE NOT EXISTS (

                SELECT  1

                FROM Store AS s

                WHERE store_number = '$store number'

                );
```

- Else:
  - Use ‘$manager email’ and ‘$store number’ relation to find storeID and activeManagerID and insert into Manage table and add ‘$manager email’ and associated manager name to ActiveManager table. Delete the manager with ‘$manager email’ from InActiveManager table.

```
INSERT INTO ActiveManager

(email, manager_name)

SELECT email, manager_name

FROM InActiveManager

WHERE email = '$manager email'

AND NOT EXISTS (SELECT email, manager_name

FROM ActiveManager

WHERE email = '$manager email');


DELETE FROM InActiveManager

WHERE email = '$manager email';
```

```
INSERT INTO Manage (activeManagerID, storeID)

WITH CTE_Store AS

(SELECT storeID FROM Store WHERE store_number = '$store number' )

,CTE_Manager AS

(SELECT activeManagerID FROM ActiveManager WHERE email = '$manager email')

,CTE_Result AS

(SELECT CM.activeManagerID, CS.storeID FROM CTE_Store CS INNER JOIN CTE_Manager
CM ON 1=1 )

SELECT CR.activeManagerID, CR.storeID

FROM CTE_Result CR

WHERE

NOT EXISTS (SELECT *

FROM manage M

INNER JOIN CTE_Result CR ON M.activeManagerID = CR.activeManagerID

                                        AND M.storeID = CR.storeID

);
```

        - Display success message, ‘Manger is already assigned to store’.
- When ***Main Menu*** button is pushed:
    - Go to the **Main Menu** form.
- When ***Log out*** button is pushed:
    - Go to the **Login** form.

Unassign Manager

Abstract Code

- User clicked on ***Unassign Manager*** button from **Edit Manager**.
- User enters ‘$manager email’ and ‘$store number’ input fields.
- When ***Save*** button is pushed:
    - If ‘$manager email’ is blank:
        - Stay on **Edit Manager** page with error message, ‘Please fill in manager’s email’.
    - If ‘$store number’ is blank:
        - Stay on **Edit Manager** page with error message, ‘Store cannot be found’.
    - If ‘$manager email’ and ‘$store number’ cannot be found in any tuple in Manage, Active Manager, and Store tables with the manager email and specific store number:
        - Stay on **Edit Manager** page with error message, ‘The manager is not assigned to the store.’

```
SELECT  'The manager is not assigned to the store'
WHERE NOT EXISTS (
          SELECT  1
          FROM Manage M
          INNER JOIN ActiveManager AM ON M.activeManagerID = M.activeManagerID
          INNER JOIN Store S ON S.storeID = M.storeID
          WHERE S.store_number = '$store number'
          AND AM.email = '$manager email'
                    );
```

- Else:
  - Remove the records of ‘$manager email’ and ‘$store number’ from Manage relationship with success message ‘The has been successfully unassigned from the store’.

```
DELETE M

FROM Manage M

INNER JOIN ActiveManager AM ON AM.activeManagerID = M.activeManagerID

INNER JOIN Store S ON S.storeID = M.storeID

WHERE S.store_number = '$store number'

AND AM.email = '$manager email';
```

        - If this manager does not have any relation with store, it will be marked as InActiveManager.

```
INSERT INTO InActiveManager
(email, manager_name)
WITH CTE_Active_Manager AS
(
SELECT DISTINCT email, manager_name
FROM ActiveManager AM
WHERE AM.email = '$manager email'
AND NOT EXISTS (SELECT 1
          FROM Manage M
          INNER JOIN ActiveManager AM ON M.activeManagerID = M.activeManagerID
          WHERE AM.email = '$manager email'
                    )
)
SELECT email, manager_name
FROM CTE_Active_Manager;


DELETE FROM ActiveManager
WHERE email = '$manager email';
```

- When ***Main Menu*** button is pushed:
    - Go to the **Main Menu** form.
- When ***Log out*** button is pushed:
    - Go to the **Login** form.

Edit Population

Abstract Code

- User clicked on ***Edit Population*** button from **Main Menu**.
- When ***View*** button is pushed:
    - User enters any of the following fields: '$city name', and '$state name'
    - Display city, state, population:

```
SELECT city, state, population

FROM City

WHERE city= '$city name' AND state= '$state name';
```

- When ***Edit*** button is pushed:
    - User enters '$city name', '$state' and '$population' input fields.
    - When ***Save*** button is pushed:
        - If combination of '$city name' and '$state name' does not exist.
            - Stay on **Edit Population** page with error message, 'City and State combination cannot be found'.

```
SELECT 'City and State combination cannot be found'

WHERE NOT EXISTS (

                    SELECT  1

                    FROM City

                    WHERE city= '$city name' AND state = '$state name'

                    );
```

        - If '$city name' is blank:
            - Stay on **Edit Population** page with error message, 'city name cannot be blank'.
        - If '$population' does not follow the standard format or is blank:
            - Stay on **Edit Population** page with error message, 'population is incorrect'.
        - Else:

        - Update population using '$population' to corresponding '$city name' and '$state name'

```
UPDATE City

SET population = '$population'

WHERE city= '$city name' AND state = '$state name'
```

- When ***Main Menu*** button is pushed:
    - Go to the **Main Menu** form.
- When ***Log out*** button is pushed:
    - Go to the **Login** form.

View Manufacturer's Product Report

Abstract Code

- User clicked on ***View Manufacturer's Product Report*** button from **Main Menu**.
- Run the **View Manufacturer's Product Report:** query for information about products condition for each manufacturer.
- Find the Manufacturer using Manufacturer [manufacturer_name].
- Find the relevant Product for each Manufacturer using Product [pid] and Manufacturer [manufacturer_name].
- Count the number of products for each manufacturer (count (Product [pid]))
- Compute the average, minimum and maximum retail price for each manufacturer.
- Display the manufacturer's name, total number of products offered by the manufacturer, average retail price, minimum retail price and maximum retail price for each manufacture; Sort by highest average retail price and only list the top 100 manufacturers.

```
SELECT

M.manufacturer_name,

COUNT(DISTINCT P.PID) AS numbers_of_products,

AVG(P.retail_price) AS average_retail_price,

MIN(P.retail_price) AS minimum_retail_price,

MAX(P.retail_price) AS maximum_retail_price

FROM Product P

INNER JOIN Manufacturer M ON P.manufacturerID = M.manufacturerID

GROUP BY M.manufacturer_name

ORDER BY AVG (P.retail_price) DESC

LIMIT 100:
```

- Each manufacturer has a ***Detail*** button.
- If the user pushes Detail button for a specific manufacturer (‘$manufacturer_name’), display the manufacturer’s name, maximum discount, and all the information mentioned above for this manufacturer. Then perform the following query:
    - Filter the products from the Product and Manufacturer tables by the target manufacturer as target products
    - Find the Category/categories for the target products from Label relation. If a product has multiple categories, concatenate them as one string.
    - Display the product ID, name, retail price and Category/categories for each product with the target manufacturer; Order by retail price from high to low.

```
SELECT
MF.manufacturer_name,
MF.max_discount AS maximum_discount,
P.pid AS product_ID,
P.product_name,
GROUP_CONCAT(DISTINCT C.category_name ORDER BY C.category_name ASC SEPARATOR
',') AS category_name,
P.retail_price AS price
FROM Manufacturer AS MF
LEFT JOIN Product AS P ON P.manufacturerID = MF.manufacturerID
LEFT JOIN Label AS L ON P.pid = L.pid
LEFT JOIN Category AS C ON C.categoryID = L.categoryID
GROUP BY
MF.manufacturer_name,
MF.max_discount,
P.pid,
P.product_name,
P.retail_price
ORDER BY P.retail_price DESC;
```

- When ***Main Menu*** button is pushed:
    - Go to the **Main Menu** form.
- When ***Log out*** button is pushed:
    - Go to the **Login** form.

View Category Report

Abstract Code

- User clicked on ***View Category Report*** button from **Main Menu**.
- Run the **View Category Report:** query for information about products and manufacturer condition for each category.

- Find the Category using Category[category_name]
- For each Category – Product relationship (Label relationship), find the category name, manufacturer name from Manufacturer table, retail price from Product table
- Count product as number of products, count distinct(manufacturer) as number of unique manufacturers, get average of retail price as average retail price for each category for records in Label relationship
- Display the category name, number of products, number of unique manufacturers, average retail price for each category; sorted by category name ascending.
- When ***Main Menu*** button is pushed:
    - Go to the **Main Menu** form.
- When ***Log out*** button is pushed:
    - Go to the **Login** form.

```
SELECT

C.category_name AS category_name,

COUNT(DISTINCT P.pid) AS numbers_of_products,

COUNT(DISTINCT P.manufacturerID) AS numbers_of_manufacturer,

AVG(P.retail_price) AS average_retail_price

FROM Category AS C

LEFT JOIN Label AS L ON C.categoryID = L.categoryID

LEFT JOIN Product AS P ON P.pid = L.pid

GROUP BY C.category_name

ORDER BY C.category_name ASC;
```

View Actual versus Predicted Revenue for GPS units Report

Abstract Code

- User clicked on ***View Actual versus Predicted Revenue for GPS units Report*** button from **Main Menu**.
- Run the **View Actual versus Predicted Revenue for GPS units Report:** query for information about revenue and predicted revenue condition for each product in specific date range.
- Find all the products under GPS category from Category – Product relationship (Label relationship) as the target products

- Filter the products by the target products in SalesRecord relationship as target sale records
- Find the sales date, product name, quantity, and retail price from Product table for each target sale record
- Find the sale price from OnSale relationship for each target sale record. If there is match record in OnSale relationship with the product ID (Product [pid]) and sale date from the SalesRecord relationship, price would be the sale price which comes from the OnSale relationship; otherwise, price would be retail price.
- Calculate the actual revenue = sales price * quantity.
- If sale price == Product [retail price]:
  - predicted revenue = actual revenue
  - predicted quantity = quantity
- Else:
  - predicted quantity = quantity * (1- 25%)
  - predicted revenue = Product [retail price] * predicted quantity
- Display the product ID, product name, total number of units ever sold, total number of units sold at a discount, total actual revenue, total predicted revenue, the difference between the actual revenue and predicted revenue for each product; Only predicted revenue differences > 5,000 or < -5,000 will be displayed; Sort by revenue differences in descending order.
- When ***Main Menu*** button is pushed:
  - Go to the **<u>Main Menu</u>** form.
- When ***Log out*** button is pushed:
  - Go to the **<u>Login</u>** form.

```
SELECT

P.pid AS productID,

P.product_name,

P.retail_price,

SUM(SR.quantity) AS numbers_of_sold,

SUM(CASE WHEN SLS.sale_price IS NULL THEN SR.quantity

                ELSE 0 END) AS numbers_of_sold_at_discount,

SUM(SR.quantity * IFNULL(SLS.sale_price, P.retail_price)) AS actual_revenue,

SUM(CASE WHEN SLS.sale_price IS NULL THEN SR.quantity * P.retail_price

                ELSE SR.quantity *(1-0.25)*P.retail_price END) AS predicted_revenue,

SUM(SR.quantity * IFNULL(SLS.sale_price, P.retail_price) -

(CASE WHEN SLS.sale_price IS NULL THEN SR.quantity * P.retail_price

                ELSE SR.quantity *(1-0.25)*P.retail_price END) )AS difference_of_revenue

FROM SalesRecord AS SR

INNER JOIN Label AS L ON SR.pid = L.pid

INNER JOIN Category C ON C.categoryID = L.categoryID

INNER JOIN Product AS P ON P.pid = L.pid

LEFT JOIN OnSale AS SLS ON SLS.date = SR.date AND SLS.pid = P.pid

WHERE C.category_name = 'GPS'

GROUP BY P.pid, P.product_name, P.retail_price

HAVING ABS(SUM(SR.quantity *   -

(CASE WHEN SLS.sale_price IS NULL THEN SR.quantity * P.retail_price

                ELSE SR.quantity *(1-0.25)*P.retail_price END) )) >5000

ORDER BY ABS(SUM(SR.quantity * IFNULL(SLS.sale_price, P.retail_price) -

                        (CASE WHEN SLS.sale_price IS NULL THEN SR.quantity * P.retail_price

                                ELSE SR.quantity *(1-0.25)*P.retail_price END) )) DESC;
```

View Store Revenue by Year by State

Abstract Code

- User clicks on ***View Store Revenue by Year by State*** button from **Main Menu**.
- When the user selects a specific state ('$state') from the state drop-down box, perform the following query

  - Select the stores from the target cities as target stores in the Store table
  - Keep the sale records in the target stores as the target records from the SalesRecord relationship
  - Find the year from the SalesRecord table for each record for the target records by matching date
  - Find the retail price from the Product table for each record in SalesRecord relationship using PID
  - Find the sale price from the OnSale relationship for each record in SalesRecord relationship by matching PID and date
  - Calculate the revenue for each record in SalesRecord relationship:
    If sale price is null: revenue = retail price × quantity
    otherwise: revenue = sale price × quantity
  - Sum up the revenue for each target store by year
  - Find the store ID, store address from the Store table using store number
  - Find the city name from the Store table
- Display the store ID, store address, city name, sales year and total revenue for each target store; sort by year in ascending order and then by revenue in descending order
- When ***Main Menu*** button is pushed:
  - Go to the **Main Menu** form.
- When ***Log out*** button is pushed:
  - Go to the **Login** form.

```
SELECT
        S.store_number,
        S.street_address AS store_address,
        C.city,
        YEAR(SR.date) AS sales_year,
        SUM(SR.quantity * IFNULL(SLS.sale_price, P.retail_price)) AS Total_Revenue
FROM Store S
INNER JOIN City C ON S.cityID = C.cityID
LEFT JOIN SalesRecord SR ON S.storeID = SR.storeID
LEFT JOIN Product P ON SR.pid = P.pid
LEFT JOIN OnSale SLS ON SLS.date = SR.date AND SR.pid = SLS.pid
WHERE S.state = '$state'
GROUP BY
        S.store_number,
        S.street_address,
        C.city,
        YEAR(SR.date)
ORDER BY YEAR(SR.date)  ASC,
           SUM(SR.quantity * IFNULL(SLS.sale_price, P.retail_price)) DESC;
```

View Air Conditioners on Groudhog Day

Abstract Code

- User clicks on ***View Air Conditioners On Groudhog Day*** button from **Main Menu**.
- Perform the following query
    - Select the products (PID) that belong to the category of air conditioners from the Label relation as the target products
    - Select the records for the target products in SalesRecord relationship as target records
    - Find the holiday name, and year from the Date table for the target records

    - Sum up the quantities of the target records for each year as the total number of items sold of that year
    - Divide the sum of quantities by 365 for each year as the average number of units sold per day of that year
    - Sum up the quantities of the target records on Groundhog Day for each year as the total number of units sold on Groundhog Day of that year

- Display the total number of items sold that year, the average number of units sold per day, and the total number of units sold on Groundhog Day of that year for each year; sort by year in ascending order.
- When ***Main Menu*** button is pushed:
    - Go to the **Main Menu** form.
- When ***Log out*** button is pushed:
    - Go to the **Login** form.

```
SELECT

YEAR(DD.date) AS sales_year,

SUM(SR.quantity)/365 AS average_numbers_of_sold,

SUM(CASE WHEN DD.holiday_name = 'Groundhog Day' THEN SR.quantity

          ELSE 0 END) AS numbers_of_sold_on_groundhog

FROM Date DD

INNER JOIN SalesRecord SR ON DD.date = SR.date

INNER JOIN Label L ON SR.pid = L.pid

INNER JOIN Category C ON L.categoryID = C.categoryID

WHERE C.category_name = 'air conditioning'

GROUP BY YEAR (DD.date)

ORDER BY YEAR (DD.date) ASC;
```

View State with Highest Volume for each Category

Abstract Code

- User clicks on ***View State with Highest Volume for Each Category*** button from **Main Menu**.
- User select the year (‘$year’) and month (‘$month’)
- Perform the following query

  - Find the year and month for each record in SalesRecord relationship by matching date
  - Select the records in the target month and target year from the SalesRecord relationship as the target records
  - Find the category from the Label relation for each record in target records using PID
  - Find the state of the store from the Store table for each record in the target records using store number
  - Sum up the quantities of the unit by each category and each state as the total number of units
  - Find the state with the highest total number of units for each category

- Display the category name, state that sold the highest number of units in that category, and the number of unites that were sold in the states; sort by the category name ascending.

```
WITH CTE_Category

AS (

SELECT C.category_name, S.state, SUM(SR.quantity) AS
numbers_of_sold_by_category

FROM SalesRecord SR

INNER JOIN Label L ON L.pid = SR.pid

INNER JOIN Category C ON L.categoryID = C.categoryID

INNER JOIN Store S ON SR.storeID = S.storeID

WHERE YEAR(SR .date) = '$year' AND MONTH(SR.date) = '$month'

GROUP BY C.category_name, S.state

)

, CTE_Category_Rank

AS (

SELECT category_name, state, numbers_of_sold_by_category,

RANK () OVER (PARTITION BY category_name ORDER BY
numbers_of_sold_by_category DESC) AS R

FROM CTE_Category

)

SELECT

CCR.category_name,

CCR.state AS highest_sales_state,

SUM(SR.quantity) AS total_units

FROM SalesRecord AS SR

INNER JOIN Store AS S ON SR.storeID= S.storeID

INNER JOIN CTE_Category_Rank AS CCR ON CCR.state = S.state AND R = 1

GROUP BY CCR.category_name, CCR.state

ORDER BY CCR.category_name ASC;
```

- Each category will have a ***DETAILS*** button.
- If user clicks on ***DETAILS*** button for a specific category ('$category_name'), the state('$state name') with the highest number of units sold in that category in that year ('$year'), and month ('$month') would be the target state; perform the following query

  - Select the records in the target month and target year from the SalesRecord relationship as the target records
  - Find the category from the Label relation for each record in target records.
  - Filter the records in the target records by the category to the target category as the target records
  - Find the city and state of the store from the Store table for each record in the target records
  - Filter the records in the target records by the state to the target state as the target records
  - Select the distinct stores from the target records as the target stores
  - Find the store ID, address and cities from the Store Table of the target stores
  - Find the manager names and email address for each store from ActiveManager table

- Display the target category, target year, target month, target state as the summary. Note, the stores here would only include the stores that have sold any unit in the target category for the specific year and month.
- Also display the store ID, address, city, manger name and email address for each manager in each target store; order by store ID ascending.

```
SELECT DISTINCT
S.store_number,
S.street_address AS store_address,
City.city,
MR.manager_name,
MR.email AS manager_email
FROM SalesRecord SR
INNER JOIN Label L ON SR.pid = L.pid
INNER JOIN Category C ON L.categoryID = C.categoryID
INNER JOIN Store S ON SR.storeID = S.storeID
INNER JOIN City ON City.cityID = S.cityID
LEFT JOIN Manage M ON S.storeID = M.storeID
LEFT JOIN ActiveManager MR ON MR.activeManagerID = M.activeManagerID
WHERE S.state = '$state name' AND C.category_name = '$category_name'
          AND YEAR(SR.date) = '$year' AND MONTH(SR.date) = '$month'
```

- When ***Main Menu*** button is pushed:
    - Go to the **Main Menu** form.
- When ***Log out*** button is pushed:
    - Go to the **Login** form.

View Revenue by Population

Abstract Code

- User clicks on ***View Revenue by Population*** button from **Main Menu**.
- Perform the following query

    - Find the year from the Date table for each record in SalesRecord relationship by matching on date

- Find the city of the store from the Store table for each record in SalesRecord relationship using store number
- Find the retail price from the Product table for each record in SalesRecord relationship using PID
- Find the sale price from the OnSale relationship for each record in SalesRecord relationship using PID and date information
- Calculate the revenue for each record in SalesRecord relationship:
  If sale price is null: revenue = retail price × quantity
  otherwise: revenue = sale price × quantity
- Sum the revenue for each city and each year in the SalesRecord relationships as the total revenue for city by year
- Find the population from City table for each city and categorize the population as below: Small (population < 3,700,000), Medium (population >= 3,700,000 and < 6,700,000), Large (population >= 6,700,000 and < 9,000,000) and Extra Large (population >= 9,000,000).
- Get the total revenue for each city category and each year.

- Display the revenue by city category in ascending order and year in ascending order
- When ***Main Menu*** button is pushed:
  - Go to the **Main Menu** form.
- When ***Log out*** button is pushed:
  - Go to the **Login** form.

```
WITH CTE_result AS (

SELECT

YEAR(SR.date) AS sales_year,

CASE WHEN C.population < 3700000 THEN 'Small'

        WHEN C.population >= 3700000 AND C.population < 6700000 THEN 'Medium'

        WHEN C.population >= 6700000 AND C.population < 9000000 THEN 'Large'

        WHEN C.population >= 9000000 THEN 'Extra_Large'

END AS city_category,

SUM(SR.quantity * IFNULL(SLS.sale_price, P.retail_price)) AS revenue

FROM SalesRecord SR

INNER JOIN Product P ON P.pid = SR.pid

INNER JOIN Store S ON SR.storeID = S.storeID

INNER JOIN City C ON S.cityID = C.cityID

LEFT JOIN OnSale SLS ON SLS.date = SR.date AND SLS.pid = SR.pid

WHERE C.population IS NOT NULL

GROUP BY YEAR(SR.date),

CASE WHEN C.population < 3700000 THEN 'Small'

        WHEN C.population >= 3700000 AND C.population < 6700000 THEN 'Medium'

        WHEN C.population >= 6700000 AND C.population < 9000000 THEN 'Large'

        WHEN C.population >= 9000000 THEN 'Extra_Large'

END

)

SELECT sales_year,

SUM(CASE WHEN city_category = 'Small' THEN revenue ELSE 0 END) AS Small,

SUM(CASE WHEN city_category = 'Medium' THEN revenue ELSE 0 END) AS Medium,

SUM(CASE WHEN city_category = 'Large' THEN revenue ELSE 0 END) AS Large,

SUM(CASE WHEN city_category = 'Extra_Large' THEN revenue ELSE 0 END) AS Extra_Large

FROM CTE_result

GROUP BY sales_year

ORDER BY sales_year ASC;
```