Table of Contents

Abstract Code (AC) and SQL

Login Task

Abstract Code

- User enters employee_id `$UserInputEmployee_Id` and password `$UserInputPassword` input fields.
- If data validation is successful for both username and password input fields, then:
    - When **Enter** button is clicked:

-- Select the user's details based on employee_id

```
SELECT
        first_name,
       last_name
        ssn_last4
FROM
      `User`
WHERE
       employee_id = `$UserInputEmployee_Id `;
```

    - If User employee_id is not found or `User`.ssn_last4 || '-' || USER.last_name != `$UserInputPassword`):
        - Go back to **Login** form, with error message.
    - Else:
        - Store login information as session variable `$UserEmployee_Id`
          Go to **Menu** form.

View Menu Task

Abstract Code

- User clicked on the ***Login*** button from **Login** form and successfully passed
    - Display String: “Welcome, 'Users Name’”

```
SELECT
      first_name,
      last_name
FROM
      `User`
WHERE
      employee_Id = `$Employee_Id `;
```

    - Display Query: Select count for the following tables:

- Store, City, District, Manufacturer, Product, Category, and Holiday

```
SELECT
        *
FROM (
        SELECT 'Store' AS TableName, COUNT(*) AS TableCount FROM Store
        UNION SELECT 'City' AS TableName, COUNT(*) AS TableCount FROM City
        UNION SELECT 'District' AS TableName, COUNT(*) AS TableCount FROM District
        UNION SELECT 'Manufacturer' AS TableName, COUNT(*) AS TableCount FROM
        Manufacturer
        UNION SELECT 'Product' AS TableName, COUNT(*) AS TableCount FROM Product
        UNION SELECT 'Category' AS TableName, COUNT(*) AS TableCount FROM Category
        UNION SELECT 'Holiday' AS TableName, COUNT(*) AS TableCount FROM Holiday
        ) AS UnionCount;
```

- User will be able to click on link buttons to access to other pages:
  - Click ***Holidays*** button - Jump to **View Holiday** task
  - Click ***Manufacturers Product Report*** button - Jump to **Manufacturers Product Report** task
  - Click ***Category Report*** button - Jump to **Category Report** task
  - IF active session Employee has 1 or more Districts assigned to them:
    - Click *Actual* ***versus Predicted Revenue for GPS units*** button - Jump to **Actual versus Predicted Revenue for GPS units** task
    - Click ***Air Conditioners on Groundhog Day?*** button - Jump to **Air Conditioners on Groundhog Day?** task
  - IF active session Employee has ALL Districts assigned to them :
    - Click ***Store Revenue by Year by State*** button - Jump to **Store Revenue by Year by State** task
    - Click ***District with Highest Volume for Each Category*** button - Jump to **District with Highest Volume for Each Category** task
    - Click ***Revenue by Population*** button - Jump to **Revenue by Population** task
  - IF active session Employee_ID has Audit_Flag marked as TRUE
    - Click ***Audit*** button - Jump to **View Audit Log** task
- Click ***Logout*** button - Exit main session and return to **Login form**

View Holiday Task

Abstract Code

- User selects ***View Holidays*** button from the **Menu** form
- Display Query:
    - Select all from the Holiday table to retrieve all holiday records

```
SELECT holiday_date, holiday_name, created_by
FROM Holiday
ORDER BY holiday_date ASC;
```

- Close the report and return to the **Main Menu** when user clicks the ***exit*** button

Add Holiday Task

Abstract Code

- User select the ***Add Holidays*** button from the **Menu** form
- User enters Holiday date and Holiday name for the input fields.
    - Input Validation:
        - Ensure Holiday Date is a valid date format.
            - If not show error message
            - Else continue
        - Ensure Holiday name is a valid name format.
            - If not show error message
            - Else continue
    - Query the database to check if a holiday already exists on the date

```
SELECT holiday_date, holiday_name, created_by
FROM Holiday
WHERE holiday_date = `$UserInputHolidayDate`;
```

        - If holiday exists:
            - Show warning “A Holiday exists on that date”
        - Else:
            - Add new Holiday name and Holiday date to database

```
INSERT INTO Holiday
VALUES (`$UserInputHolidayDate `, `$UserInputHolidayName `, `$UserID `);
```

            - Display success message on successful creation.
            - Associate the holiday with the user who created it.
- Close the form and return to the **Main Menu** when user clicks the ***exit*** button

View Audit Log Task

Abstract Code

- User selects ***View Audit Log*** button – Jump to the **View Audit Log** task
- Display Query: Select top 100 most recent Audit logs from the Audit table

```
SELECT log_id, timestamp, audit_flag, employee_id, report_name
FROM Audit
ORDER BY timestamp DESC
LIMIT 100;
```

- Close the report and return to the **Main Menu** when user clicks the ***exit*** button

Manufacturer's Product Report Task

Abstract Code

- User clicked on ***Manufacturer's Product Report*** button from **Main Menu** form
- Run the **View Manufacturer's Product Report** task to display information from the materialized view, created by querying the Product and Manufacturer tables:
    - The materialized view is created by running a query to join the two tables and performing a group by operation to calculate:
        - Total number of products offered by each manufacturer
        - Average retail price of all the manufacturer's products,
        - Minimum retail price for each manufacturer
        - Maximum retail price for each manufacturer
    - The materialized view returns the results, including the manufacturer name sorted by the average price in descending order; limited to 100 records

-- Step 1: Top 100 manufacturers' information with the highest average price on products

```
WITH manufacturer_report AS
(SELECT m.manufacturer_name,
    count(distinct p.pid) as number_of_products,
    avg(p.price) as avg_price,
    max(p.price) as max_price,
    min(price) as min_price
FROM Manufacturer m
    JOIN Product p ON m.manufacturer_name = p.manufacturer_name
GROUP BY m.manufacturer_name
ORDER BY avg(p.price) DESC
LIMIT 100)
SELECT manufacturer_name,
    number_of_products,
    avg_price,
    max_price,
    min_price
FROM manufacturer_report;
```

- Clicking a ***hyperlink*** connected to each row of data will display:
  - Manufacturers name
  - Manufacturers products id's
  - Manufacturers products names
  - Manufacturers products categories (in list form)
  - Manufacturers products Price
- The Hyperlink based view returns the results sorted by price in descending order

-- Step 2: Summary of manufacturer's information displayed upon clicking a hyperlink on a given row in the report above

```
SELECT manufacturer_name,
    number_of_products,
    avg_price,
    max_price,
    min_price
FROM manufacturer_report
WHERE manufacturer_name = `$manufacturer_name `;
```

-- Step 3: Detailed information of the manufacturer's information displayed upon clicking a hyperlink on a given row in the report above

```
SELECT p.pid,
    p.product_name,
    p.price,
   GROUP_CONCAT(category_name ORDER BY category_name SEPARATOR ',')
FROM Manufacturer m
    JOIN Product p ON m.manufacturer_name = p.manufacturer_name
   JOIN ProductCategory pc ON p.pid = pc.pid
WHERE manufacturer_name = `$manufacturer_name `
GROUP BY
    p.pid,
    p.product_name,
    p.price
ORDER BY p.price DESC;
```

- Close the report and return to the **Main Menu** form when user clicks the *exit* button

Category Report Task

Abstract Code

- User clicked on C*ategory Report* button from **Main Menu**
- Run the **View Category Report** task to display information from a materialized view, created by querying the Category, Product and ProductCategory tables:
    - The materialized view is created by running a query to join the three tables using the product ID variable and performing a group by operation on the category field to calculate:
        - Total number of products in each category
        - Total number of manufacturers offering products in each category
        - The average retail price (not including discount days) of all the products in each category
    - The materialized view returns the results, including the category name sorted in ascending order

```
SELECT c.category_name,
    count(distinct p.pid) as number_of_products,
    count(distinct p.manufacturer_name) as number_of_manufacturers,
    avg(price) as avg_price
FROM Category c
    JOIN ProductCategory pc ON c.category_name = pc.category_name
    JOIN Product p ON p.pid = pc.pid
GROUP BY c.category_name
ORDER BY c.category_name ASC;
```

- Close the report and return to the **Main Menu** when user clicks the *exit* button

Actual versus Predicted Revenue for GPS units Task

Abstract Code

- User clicked on *Actual versus Predicted Revenue for GPS units report* button from **Main Menu**
- Run the **View Actual versus Predicted Revenue for GPS units report** task to display information from a materialized view, created by querying the category, product, sales and discount tables:
    - The materialized view is created by running a query to perform multiple joins on the category table filtered by category_name = 'GPS' with the three tables using the variables: product ID, sale date, and discount date variable and performing multiple operations including different filtrations and grouping of data to calculate:
        - Total number of units ever sold for each product
        - Total number of units sold at a discount for each product
        - Total number of units sold at retail price for each product
        - Total revenue collected from all the sales of for each product
        - Total predicted revenue (based on 75% volume selling at retail price)
        - Predicted revenue differences = Total revenue - Total predicted revenue
    - The materialized view returns the results including the product ID, product name, product's retail price for products where abs(Predicted revenue differences) > 200, ordered by the revenue difference descending

```
SELECT pid,
   product_name,
   price,
   sum(quantity) AS total_quantity_sold,
   sum(discount_quantity) AS total_quantity_sold_discount,
   sum(retail_quantity) AS total_quantity_sold_retail,
```

```
    sum(actual_revenue) AS actual_revenue,
    sum(predicted_revenue) AS predicted_revenue,
    abs(sum(actual_revenue) - sum(predicted_revenue)) as revenue_diff
FROM (
        SELECT pid,
            product_name,
            price,
            quantity,
            CASE
                WHEN discount_price is NULL THEN quantity
                ELSE 0
            END AS retail_quantity,
            CASE
                WHEN discount_price is NOT NULL THEN quantity
                ELSE 0
            END AS discount_quantity,
            CASE
                WHEN discount_price is NULL THEN price * quantity
                ELSE discount_price * quantity
            END AS actual_revenue,
            CASE
                WHEN discount_price is NULL THEN price * quantity
                ELSE price * (0.75 * quantity)
            END AS predicted_revenue
        FROM (
                SELECT s.pid,
                    p.product_name,
                    p.price,
                    dap.discount_price,
                    s.quantity
                FROM Sales s
                    LEFT JOIN (
                        SELECT *
                        FROM DiscountAppliesToProduct dp
                            JOIN Discount d ON dp.discount_id = d.discount_id
                    ) dap ON s.date = dap.discount_date
                    AND s.pid = dap.pid
                    JOIN Product p ON s.pid = p.pid
                    JOIN ProductCategory pc ON s.pid = pc.pid
                WHERE category_name like '%Air Conditioning%'
            ) AS tbl0
    ) AS tbl1
GROUP BY pid,
    product_name,
```

```
    price
HAVING revenue_diff > 200
ORDER BY revenue_diff DESC;
```

- Close the report and return to the **Main Menu** when user clicks the *exit* button

Air Conditioners on Groundhog Day? Task

Abstract Code

- User selects ***Air Conditioners on Groundhog Day?*** button – Jump to the **Air Conditioners on Groundhog Day?** task
- Display Query in Ascending order by year:
    - Year
    - Count of products in the 'Air Conditioning' category sold
    - Average number of 'Air Conditioning' category products sold per day
    - Total number of 'Air Conditioning' category products sold on Feb 2nd

```
WITH
  ac_sales AS (
    SELECT
      YEAR( Sales.date) AS year,
      sum(Sales.quantity) AS annual_sales,
      sum(Sales.quantity) / 365 as daily_sales
    FROM
      Sales
      JOIN Product on Product.pid = Sales.pid
      JOIN ProductCategory on ProductCategory.pid = Product.pid
    WHERE
      ProductCategory.category_name like '%Air Conditioning%'
    GROUP BY
      year
  ),
  ghdSales AS (
    SELECT
      YEAR(Sales.date) as year,
      sum(Sales.quantity) AS sales
    FROM
      Holiday
      JOIN Sales on Sales.date = Holiday.date
      JOIN Product on Product.pid = Sales.pid
      JOIN ProductCategory on ProductCategory.pid = Product.pid
    WHERE
      ProductCategory.category_name like '%Air Conditioning%'
```

```
        AND Holiday.holiday_name = 'Groundhog Day'
    GROUP BY
        year
  )
SELECT
  ac_sales.year,
  ac_sales.annual_sales,
  ac_sales.daily_sales,
  ghdSales.sales
FROM
  ac_sales
  JOIN ghdSales ON ac_sales.year = ghdSales.year
ORDER BY
  ac_sales.year ASC
```

- Close the report and return to the **Main Menu** when user clicks the *exit* button

Store Revenue by Year by State Task

Abstract Code

- User clicked on *Store Revenue by Year by State* button from **Main Menu** form
  - Click **States** Dropdown menu
    - Click a single **State** from the dropdown

```
SELECT UNIQUE
        State
FROM
        City;
```

- Run the **Store Revenue by Year by State** task to display information from the materialized view, created by querying the City, Store and Sales tables:
  - The materialized view is created by running a query to join all three tables, performing a group by operation on a calculated Year field:
    - Store Number
    - Store Address
    - City Name
    - Year (extracted from the sales dates)
    - Total Revenue (calculated as a sum of sales totals for each year)
  - The materialized view returns the results, including the state name sorted by the revenue in descending order

```
SELECT ct.state,
  st.store_number,
  st.address,
  ct.city_name,
```

```
    YEAR(sa.date) AS Year,
    SUM(sa.total) AS Total_Revenue
FROM City AS ct
    LEFT JOIN Store AS st ON st.city_id = ct.city_id
    LEFT JOIN Sales AS sa ON sa.store_number = st.store_number
WHERE ct.State = `$State`
GROUP BY ct.state,
    st.store_number,
    st.address,
    ct.city_name,
    YEAR(sa.date) AS Year
ORDER BY Year ASC,
    Total_Revenue DESC;
```

- Close the report and return to the **Main Menu** form when user clicks the ***exit*** button

District with Highest Volume for each Category Task

Abstract Code

- User selects ***District with Highest Volume for each Category*** button from the **Main Menu**
- Run **View District with Highest Volume for each Category** task
- User chooses a year `$UserInputYear` and month `$UserInputMonth` from the available dates in the database
    - For the selected month, query Category, Sales, Product and Store tables to show a report for each Category that includes:
        - Category name
        - The district that sold the highest number of units in that category
        - Names of Product sold by all stores in the district
        - Number of units sold by stores in that district
    - Clicking a ***hyperlink*** connected to each row of data will display a sub report containing:
        - District ID
        - Addresses of the stores
        - States the stores are located in
        - Cities the stores are located in
    - The sub-report is ordered by store ID ascending and the header includes the original criteria from the parent report.
- Close the report and return to the **Main Menu** form when user clicks the *exit* button

-- Step 1: Aggregate total units sold per store for each category in the selected month and year

```
WITH CategorySales AS (
  SELECT
    c.category_name,
    st.district_number,
    SUM(s.quantity) AS total_units_sold
  FROM
    Sales s
    JOIN ProductCategory pc ON s.pid = pc.pid
    JOIN Category c ON pc.category_name = c.category_name
    JOIN Store st ON s.store_number = st.store_number
  WHERE
          EXTRACT(YEAR FROM s.date) = `$UserInputYear`
    AND EXTRACT(MONTH FROM s.date) = `$UserInputMonth`
  GROUP BY
    c.category_name,
    st.district_number
),
```

-- Step 2: Find the district with the highest total units sold for each category

```
MaxDistrictSales AS (
  SELECT
    category_name,
    MAX(total_units_sold) AS max_units_sold
  FROM
    CategorySales
  GROUP BY
    category_name
);
```

-- Step 3: Retrieve the district with the highest sales for each category and list the products sold

```
SELECT
  cs.category_name,
  cs.district_number,
  cs.total_units_sold
FROM
  CategorySales cs
  JOIN MaxDistrictSales mds ON cs.category_name = mds.category_name
    AND cs.total_units_sold = mds.max_units_sold
ORDER BY
```

```
  cs.category_name ASC;
```

-- Step 4: Sub-Report for Drill-Down Details
-- (This query can be run when the user clicks on a specific district in the main report)

```
SELECT
  st.district_number,
  st.store_number,
  st.address,
  ct.state,
  ct.city_name
FROM
  Store st
  JOIN City ct ON st.city_id = ct.city_id
WHERE
  st.district_number = `$DistrictNumber `
ORDER BY
  st.store_number ASC;
```

Revenue by Population Task

Abstract Code

- User clicked on ***Revenue by Population*** button from **Main Menu**
- Run the **Revenue by Population** task to display information from the materialized view, created by querying the City, Store and Sales tables:
    - The materialized view is created by running a query to join all three tables (City, Store and Sales):
        - Year (extracted from the sales dates)
        - City Size (calculated using the population field)
            - 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)
        - Retrieve annual sales data for each city
        - Calculate Average Revenue as the average sum of sales totals for each year and population demographic combination
    - The materialized view returns the results, sorted by Year and City Size in Ascending order.

```
SELECT YEAR(sa.Date) AS Year,
   CAST(
      CASE
         WHEN ct.population < 3700000 THEN ‘ Small ’
         WHEN ct.population < 6700000
```

```
        AND ct.population >= 3700000 THEN ‘ Medium ’
        WHEN ct.population < 9000000
        AND ct.population >= 6700000 THEN ‘ Large ’
        ELSE ‘ Extra Large ’
      END AS VARCHAR
  ) AS City_Size,
  AVG(sa.Total) AS Average_Revenue
FROM City AS ct
  LEFT JOIN Store AS st ON st.city_id = c.city_id
  LEFT JOIN Sales AS sa ON sa.store_number = st.store_number
GROUP BY Year,
  City_Size
ORDER BY Year ASC;
```

- Close the report and return to the **Main Menu** when user clicks the *exit* button