Abstract Code with SQL

Display Statistics for Main Menu

Abstract Code

Run the ***Display Statistics*** task

- Count and display the number of stores

```
SELECT count(store_Number) FROM `Store`;
```

- Count and display the number of products

```
SELECT count(pid) FROM `Product`;
```

- Count and display the number of Manufacturer

```
SELECT count(unique_name) FROM `Manufacturer`;
```

- Click ***Category report*** button – jump to the ***Return category summary*** task
- Click ***Predicted vs. actuals report*** button - jump to the **Return ACTL vs. PRED sales** task
- Click ***Store revenue report*** button - jump to the **Return revenue by store task & Return states list** task
- Click ***Outdoor furniture on Groundhog day report*** button - jump to the **Return Furniture sales** task
- Click ***Highest vol. per category report*** button - jump to the **Return year & month lst task & Return category/state volumes** task
- Click ***Revenue by population report*** button - jump to the **Return trend by city size** task
- Click ***Grand Showcase impact on revenue*** button - jump to **Return Category and Store type revenue** task
- Click ***Grand Showcase on category sales*** button - jump to **Regular sales and Grand Showcase** task

Update City Population

Abstract Code

- Upon clicking Update button:
  - Run the **Update city population** task
    - Store user entered text regarding the state as variable '$state_name', and user entered text regarding the city as '$city_name', and user entered text regarding the population as '$population'
    - Check if '$population' is an INT datatype

- **IF NOT** an INT then display error message for invalid entry for '$population'
- **ELSE**, run the following:

```
SELECT `State`, `Name` FROM `CITY` WHERE state = '$state_name' AND cityname = '$city_name';
```

- **IF** a city and state are returned:

```
UPDATE `CITY` SET population = '$population' WHERE state = '$state_name' AND cityname =
'$city_name';
```

- **ESLE,** update city population table with population information for a given city entered by the user

```
INSERT INTO `CITY` VALUES ('$city_name', '$state_name', '$population');
```

Report 1 – Manufacturer's Product Report

Task Decomp

Lock Types: 1 lock needed for read-only
Number of Locks: Single
Enabling Conditions: User clicks ***Manufacturer report*** button from main menu
Frequency: Frequent
Consistency (ACID): Not critical
Subtasks: Mother task is not needed. No decomposition needed.

Abstract Code

- Run the **Return manufactures' summary** task
    - For each manufacture:
        - For sale days the products not discounted
            - Return manufacturer's name
            - Count total number of products
            - Return minimum/maximum retail price across all products in that manufacture
            - Calculate the average retail price across all products in that manufacture
    - Sort by average retail price descending from top manufacturer

```
SELECT m.unique_name, COUNT(p.pID), AVG(p.retail_price), MAX(p.retail_price), MIN(p.retail_price)
FROM
        `Manufacturer` m JOIN `Product` p ON
        m.manufacturer_id = p.manufacturer_id
GROUP BY m.unique_name
ORDER BY AVG(p.retail_price) DESC
LIMIT 100;
```

- Click the **hyperlink** to view manufacturer details
    - For each manufacture name:
        - Report header

- Lookup name

```
SELECT m.unique_name,  AVG(p.retail_price)
FROM
        `Manufacturer` m JOIN `Product` p ON
        m.manufacturer_id = p.manufacturer_id
GROUP BY m.unique_name;
```

- Summary
  - Lookup manufacture product ID
  - Lookup manufacture product category and multiple categories concatenated
  - Lookup manufacture product by price descending

```
SELECT p.pID, p.name, group_concat(pt.category_name), p.retail_price
FROM
        `Product` AS p
JOIN `CategoryProductXref` AS pt ON
        p.pID = pt.pID
GROUP BY p.pID
ORDER BY p.retail_price DESC;
```

Report 2 – Category Report

Task Decomp

Lock Types: 1 lock needed for read-only
Number of Locks: Single
Enabling Conditions: User clicks ***Category report*** button from main menu
Frequency: Frequent
Consistency (ACID): Not critical
Subtasks: Mother task is not needed. No decomposition needed.

Abstract Code

- Run the **Return category summary** task
    - For each category name:
        - Count total number of products
        - Count total unique manufacturers offering products in that category
        - Calculate the average retail price across all products in that category
    - sort by category name ascending

```
SELECT c.category_name AS category_name
        COUNT(p.pID) AS count_of_products
        COUNT(DISTINCT m.unique_name) AS count_of_manufacturer
        AVG(p.retail_price) AS avg_retail_price
        MAX(p.retail_price) AS max_retail_price
        MIN(p.retail_price) AS min_retail_price
FROM
        `Category` AS c
JOIN
        `CategoryProductXref` AS ptc ON
        c.category_name = ptc.category_name
JOIN
        `Product` AS p ON
        ptc.pID = p.pID
JOIN
        `Manufacturer` m ON
        p.manufacturer_id = m.manufacturer_id

GROUP BY c.category_name
ORDER BY c.category_name;
```

Report 3 – Actual versus Predicted Revenue for Couches and Sofas

Task Decomp

Lock Types: 1 lock needed for read-only
Number of Locks: Several different schema constructs are needed
Enabling Conditions: User clicks ***Predicted vs actuals report*** button from main menu
Frequency: Frequent
Consistency (ACID): Not critical
Subtasks: Order matters, need to populate actual revenue first based on discount, then populate predicted revenue based on retail price with 75% volume to get the differences between actual revenue and predicted revenue

Abstract Code

- Run the **Return ACTL** vs. **PRED** sales task
    - For each product in "Couches and Sofas" category:
        - Return Product ID
        - Return the name of the product
        - Return the retail price of the product
        - Return the total number of units sold of that product
        - Distinguish between on-sale date and others
            - Use 0.75 adjustment factor only calculated on sale days
            - Sum both predicted revenue and actual revenue with the rest days without sales
            - Calculate the difference between actual revenue and predicted revenue.
            - Display the differences between actual revenue and predicted revenue > $5000 (positive or negative)
    - Sort by the difference between actual revenue and predicted revenue descending

```
SELECT
        p.pID,
        p.name,
        p.retail_price,
        SUM(sa.quantity) AS total_items,
        SUM(case when dd.discount_date IS NOT NULL THEN sa.quantity else 0 end) AS discount_items,
```

```
        SUM(case when dp.discount_price IS NULL THEN sa.quantity*p.retail_price ELSE
sa.quantity*dp.discount_price END) AS total_revenue,
        SUM(case  dd.discount_date IS NOT NULL  THEN sa.quantity*.75*p.retail_price ELSE
sa.quantity*p.retail_price  END) AS predicted_revenue,
        SUM(case when dp.discount_price IS NULL THEN sa.quantity*p.retail_price ELSE
sa.quantity*dp.discount_price END) -SUM(case  dd.discount_date IS NOT NULL  THEN
sa.quantity*.75*p.retail_price ELSE sa.quantity*p.retail_price END) AS diff_revenue

FROM
        `Product` p
JOIN
        `StoreProductXref`  sp ON
        p.pID = sp.pID
JOIN
        `CategoryProductXref` ptc ON
        sp.pID = ptc.pID

JOIN
        `Category` c ON
        ptc.category_id = c.category_id

JOIN
        `Sales` sa ON
        ptc.pID = sa.pID

JOIN
        `SalesDate` sd ON
        sa.date_of_sales = sd.calendar_date

JOIN
        `GenericDate` gd ON
        sd.calendar_date = gd.calendar_date

JOIN    `DiscountDate` dd  ON
        gd.calendar_date = dd.calender_date

JOIN    `DiscountPrice` dp  ON
        dd.calender_date = dp.discount_date

WHERE
        c.category_name = 'Couches and Sofas'
GROUP BY
        1,
        2,
        3
HAVING
        ABS(diff_revenue) > 5000
ORDER BY
```

```
        diff_revenue DESC;
```

Report 4 – Store Revenue by Year by State

Task Decomp

Lock Types: 1 lock needed for read-only

Number of Locks: Several different schema constructs are needed

Enabling Conditions: User clicks Store revenue report button from main menu

Frequency: Frequent

Consistency (ACID): Not critical

Subtasks: Order matters, need to populate state drop down box first, then populate report with revenue by store by year for the given state

Abstract Code

- Run the **Return revenue by store** task
    - Run the **populate state dropdown** subtask
        - Fetch unique states from database

```
SELECT DISTINCT(state_name) FROM `City`;
```

    - Run the **return revenue for each state by year subtask**
        - For the selected states:
            - For each store in state:
                - Lookup store ID, address, city name
                - For each year:
                    - Sum revenue collected
    - Sort by year ascending
    - Sort by the revenue descending

```
SELECT
        st.store_number,
        st.street_address,
        st.city_name,
        EXTRACT(YEAR
FROM gd.calender_date) AS “year”,
```

```
        SUM(case when dp.discount_price IS NULL THEN sa.quantity*p.retail_price ELSE
sa.quantity*dp.discount_price END) AS total_revenue,


FROM `Store` AS st
JOIN
        `StoreProductXref` AS sp ON
        st.store_number = sp.store_number
JOIN
        `Product` AS p ON
        sp.pID = p.pID
JOIN
        `Sales` AS sa ON
         p.pID = sa.pID

JOIN
        `SalesDate` sd ON
        sa.date_of_sales = sd.calendar_date

JOIN
        `GenericDate` gd ON
        sd.calendar_date = gd.calendar_date

JOIN    `DiscountDate` dd  ON
        gd.calendar_date = dd.calender_date

JOIN    `DiscountPrice` dp  ON
        dd.calender_date = dp.discount_date

WHERE st.state_name= '$StateSelected'
GROUP BY st.store_number, Year(gd.calender_date)
ORDER BY “year” ASC, total_revenue DESC;
```

Report 5 – Outdoor Furniture on Groundhog Groundhog Day?

Task Decomp

Lock Types: 1 lock needed for read-only

Number of Locks: Single

Enabling Conditions: User clicks outdoor furniture on Groundhog Day report button from main menu

Frequency: Annually

Consistency (ACID): Not critical

Subtasks: No additional subtasks required.

Abstract Code

- Run the **Return outdoor furniture *on Groundhog Day's* summary** task
  - Group transactions of outdoor furniture categorized product transactions by year:
    - Return Year.
    - Return sum of quantity of product units under the outdoor furniture category that were sold that year.
      - Return daily average number of products units under the outdoor furniture category that were sold by dividing the annual total by 365.
    - Return quantity of product units under the outdoor furniture category that were sold on Groundhog day (February 2).
  - Sort by year in ascending order

```
#Part 1 : Get the yearly data
select YEAR(s.date_of_sales) as yr,
          sum(s.quantify) as yearly_total_quantity,
          (sum(s.quantify)/365) as average_yearly_quantity
          into #yearly_sales
from Sales s, Category c,  CategoryProductXref x
          where s.pid  = x.pid
          and x.category_id = c.category_id
          and c.category_name like "%Outdoor Furniture%">
group by yr
```

```
#Part 2: groundhog day info
select
        YEAR(s.date_of_sales) as yr,
        s.quantify  as groundhog_day_quantify,
        into #yearly_groundhog_day_sales
from Sales s, Category c,  CategoryProductXref x
        where s.pid  = x.pid
        and x.category_id = c.category_id
        and c.category_name like "%Outdoor Furniture%">
        and MONTH(s.date_of_sales) = 2
        and DAY(s.date_of_sales) = 2


#Part 3 : that return the actual resultset
select y.yr,  yearly_total_quantity,average_yearly_quantity,groundhog_day_quantify

        from #yearly_sales y, #yearly_groundhog_day_sales h
        where y.year  = h.year
        order by y.yr asc
```

Report 6 – State with Highest Volume for each Category

Task Decomp

Lock Types: 1 lock needed for read-only

Number of Locks: Several different schema constructs are needed

Enabling Conditions: User clicks State with Highest Volume for each report button from main menu

Frequency: monthly

Consistency (ACID): Not critical

Subtasks: Order matters, need to populate the month and year from the available dates in the database. Then populate the report with the category name, the state that sold the highest number of units in that category and the number of units that were sold by stores in that state.

Abstract Code

- Run ***state with highest volume*** task
  - Run the **populate available month and year dropdowns** subtask
    - Fetch unique available dates
  - Run the **get category max sales for state** subtask:
    - For the selected date :
      - For each state in state within time range:
        - For each category:
          - Sum volume sold
          - return state with max(volume)
  - Sort by category name ascending

```
#Allow selection of the unique month for dropdown
select distinct(month(calendar_date)) from GenericDate


#Allow selection of a unique year for dropdown.
select distinct(year(calendar_date)) from GenericDate

#Now user has the month and year. Go ahead and get the rest of the report


select
        c.category_name,
```

```
        st.state_name,
        sum(sa.quantity) as max_quantity_sold
from
        Store st,
        Sales sa,
        CategoryProductXref x,
        Category c
where
        st.store_number = sa.store_number
        and sa.pid = x.pid
        and x.category_id = c.category_id
        and month(sa.date_of_sales ) = '$Month'
        and year(sa.date_of_sales ) = '$Year'
group by c.category_name,st.state_name
        having max( sum(sa.quantity))
order by c.category_name asc
```

Report 7 – Revenue by Population

Task Decomp

Lock Types: 1 lock needed for read-only

Number of Locks: Several different schema constructs are needed

Enabling Conditions: User clicks ***Revenue by Population*** for each report button from main menu

Frequency: Frequent

Consistency (ACID): Not critical

Subtasks: OrderMatters, Need to first populate the city population category. Then we will need to first calculate revenue by year.Then for each city calculate the revenue. Then apply categories to only include cities within that population category.

Abstract Code

- Run **calculate revenue by year** task
    - Run the **calculate revenue for each city** subtask
        - Fetch each city revenue on an annual basis
    - Run the **display population category state** subtask:
        - For the selected category :
            - For each city in city within population range:
                - City Size broken down:
                    - 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)
                - Sum revenue sold
    - Sort by city size ascending, sort ascending year

```
SELECT Store.city_name, EXTRACT(YEAR FROM S.date_of_sales) AS YOS, SUM(S.total_sales)
FROM Store
       JOIN Sales S on Store.store_number = S.store_number
GROUP BY Store.city_name, YOS;

-- ###display population category state
-- ##For the selected category, For each city in city within population range

--       City Size broken down: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)
--       Sum revenue sold. Sort by city size ascending, sort ascending year
SELECT Store.city_name,
     (CASE WHEN C2.population < 3700000 THEN 'Small' WHEN C2.population >= 3700000 AND
C2.population < 6700000 THEN 'Medium' WHEN C2.population >= 6700000 AND  C2.population <
9000000 THEN 'Large' WHEN C2.population >= 9000000 THEN 'Extra Large' END ) AS SIZE,
     EXTRACT(YEAR FROM S.date_of_sales) AS YOS,
     SUM(S.total_sales)
FROM Store
       JOIN Sales S on Store.store_number = S.store_number
       JOIN Product P on S.pid = P.pID
       JOIN CategoryProductXref CPX on P.pID = CPX.pID
       JOIN Category C on C.category_id = CPX.category_id
       JOIN City C2 on C2.city_name = Store.city_name and C2.state_name = Store.state_name
WHERE C.category_name = '<category_input>'
GROUP BY Store.city_name, YOS
ORDER BY FIELD(SIZE, 'Small', 'Medium', 'Extra Large'), YOS;
```

Report 8 – Grand Showcase Store Revenues Comparison

Task Decomp

Lock Types: 1 lock needed for read-only

Number of Locks: Several different schema constructs are needed

Enabling Conditions: User clicks ***Revenue of Grand Showcase*** for each report button from main menu

Frequency: Frequent

Consistency (ACID): Not critical

Subtasks: OrderMatters, need to first populate the Grand Showcase stores/normal Will-Mart stores. Then we will need to first calculate revenue by year. Then for the Grand Showcase stores/normal Will-Mart stores calculate the revenue including minimum, average and maximum and total revenues, count the number of Grand Showcase stores/normal Will-Mart stores. Finally calculate the minimum, average, total revenue of the Grand Showcase stores/normal Will-Mart stores .

Abstract Code

- Run **calculate the Grand Showcase stores/normal Will-Mart revenue by year** task
  - Run the **calculate counts** subtask
    - Count the total number of Grand Showcase stores
    - Count the total number of Will-Mart stores
  - Run the **display Grand Showcase stores/normal Will-Mart revenue** subtask
    - Calculate the minimum, average, maximum and total revenue of all Grand Showcase stores.
    - Calculate the minimum, average, maximum and total revenue of all Will-Mart stores.
  - Sort by ascending year

Report 9 – Grand Showcase Store Category Comparison

Task Decomp

Lock Types: 1 lock needed for read-only
Number of Locks: Single
Enabling Conditions: User clicks ***Grand Showcase Store Category Comparison report*** button from main menu
Frequency: Frequent
Consistency (ACID): Not critical
Subtasks: Mother task is not needed. No decomposition needed.

Abstract Code

- Run **Grand Showcase Store Category Comparison *report*** task
  - For each product:
    - Display Product ID
    - Display Product Name
    - Count the number of each product sold by Grand Showcase stores
    - Count the number of each product sold by Will-Mart stores
    - Subtract the numbers of each product sold by Grand Showcase stores from the numbers of each product sold by Will-Mart stores to obtain the differences
  - Run **Show Grand Showcase Store Category Comparison drill down** subtask:
    - Display top five of Product ID, Product Name, Grand Showcase Qty, Regular Qty and Difference
    - Display bottom five of of Product ID, Product Name, Grand Showcase Qty, Regular Qty and Difference
  - Sort by difference in descending order, product ID in ascending order

```
 -- ####Grand Showcase Store Category Comparison
-- For each product: Display Product ID,Display Product Name,Count the number of each product sold by Grand Showcase stores
-- Count the number of each product sold by Will-Mart stores
--  Subtract the numbers of each product sold by Grand Showcase stores from the numbers of each product sold by Will-Mart stores to obtain the differences

SELECT P.product_name, P.pid, SUM(Sales.quantity) AS ALL_SALES,
     SUM(CASE WHEN S.is_showcasestore = 1 THEN Sales.quantity WHEN S.is_showcasestore = 0 THEN
```

```
0 END ) AS SHOWCASE_STORE_SALES,
     SUM(Sales.quantity - CASE WHEN S.is_showcasestore = 1 THEN Sales.quantity WHEN
S.is_showcasestore = 0 THEN 0 END) AS DIFF_IN_SALES
FROM Sales
      JOIN Product P on P.pID = Sales.pid
      JOIN Store S on S.store_number = Sales.store_number
GROUP BY pid;

-- ##how Grand Showcase Store Category Comparison drill down subtask:
-- ##Display top five of Product ID, Product Name, Grand Showcase Qty, Regular Qty and Difference

SELECT P.product_name, P.pid, SUM(Sales.quantity) AS ALL_SALES,
     SUM(CASE WHEN S.is_showcasestore = 1 THEN Sales.quantity WHEN S.is_showcasestore = 0 THEN
0 END ) AS SHOWCASE_STORE_SALES,
     SUM(Sales.quantity - CASE WHEN S.is_showcasestore = 1 THEN Sales.quantity WHEN
S.is_showcasestore = 0 THEN 0 END) AS DIFF_IN_SALES
FROM Sales
      JOIN Product P on P.pID = Sales.pid
      JOIN Store S on S.store_number = Sales.store_number
GROUP BY P.pid
ORDER BY DIFF_IN_SALES DESC, P.pID ASC
LIMIT 5;

-- ## Display bottom five of of Product ID, Product Name, Grand Showcase Qty, Regular Qty and
Difference

SELECT P.product_name, P.pid, SUM(Sales.quantity) AS ALL_SALES,
     SUM(CASE WHEN S.is_showcasestore = 1 THEN Sales.quantity WHEN S.is_showcasestore = 0 THEN
0 END ) AS SHOWCASE_STORE_SALES,
     SUM(Sales.quantity - CASE WHEN S.is_showcasestore = 1 THEN Sales.quantity WHEN
S.is_showcasestore = 0 THEN 0 END) AS DIFF_IN_SALES
FROM Sales
      JOIN Product P on P.pID = Sales.pid
      JOIN Store S on S.store_number = Sales.store_number
GROUP BY P.pid
ORDER BY DIFF_IN_SALES DESC, P.pID DESC
LIMIT 5;

-- ##Sort by difference in descending order, product ID in ascending order

SELECT P.product_name, P.pid, SUM(Sales.quantity) AS ALL_SALES,
     SUM(CASE WHEN S.is_showcasestore = 1 THEN Sales.quantity WHEN S.is_showcasestore = 0 THEN
0 END ) AS SHOWCASE_STORE_SALES,
     SUM(Sales.quantity - CASE WHEN S.is_showcasestore = 1 THEN Sales.quantity WHEN
S.is_showcasestore = 0 THEN 0 END) AS DIFF_IN_SALES
FROM Sales
      JOIN Product P on P.pID = Sales.pid
```

```
    JOIN Store S on S.store_number = Sales.store_number
GROUP BY P.pid
ORDER BY DIFF_IN_SALES DESC, P.pID ASC;
```