Table of Contents

Report Dashboard

Abstract Code 1

- Display form with below links to sections:
    - "Report 1 – Manufacturer's Product Report"
    - "Report 2 – Category Report"
    - "Report 3 – Actual versus Predicted Revenue for GPS units"
    - "Report 4 –Store Revenue by Year by State"
    - "Report 5 – Air Conditioners on Groundhog Day?"
    - "Report 6 – State with Highest Volume for each Category"
    - "Report 7 – Revenue by Population"

Abstract Code 2

- Upon:
    - Click Report 1 – Manufacturer's Product Report– Jump to **Generate Manufacturer's Product** task.
    - Click Report 2 – Category Report– Jump to **Generate Category** task.
    - Click Report 3 – Actual versus Predicted Revenue for GPS units– Jump to **Generate Actual versus Predicted Revenue** for GPS Units task.
    - Click Report 4 –Store Revenue by Year by State– **Jump to Generate Store Revenue by Year and State** task.
    - Click Report 5 – Air Conditioners on Groundhog Day? – Jump to **Generate Air Conditioners on Groundhog Day** task.
    - Click Report 6 – State with Highest Volume for each Category– Jump to **Generate State with Highest Volume for each Category** task.
    - Click Report 7 – Revenue by Population– Jump to **Get Revenue by Population** task.

<u>Data Warehouse Admin</u>

Abstract Code 1

- Display form with below links to sections:
    - “Update City Population”
    - “Maintain Holiday(s)”
    - “Maintain Manage(s)”

Abstract Code 2

- Upon:
    - Click Update City Population –Jump to **Update City Population** task.
    - Click Maintain Holiday(s) – Jump to **Maintain Holiday** task.
    - Click Maintain Manage(s) – Jump to **Maintain Manager** task.

Report 1 - Manufacturer's Product Report

Abstract Code

- User runs the **Manufacturer's Product Report** task:
    - Query for all MANUFACTURER by Name
        - For each MANUFACTURER:
            - Display the Name
            - Find each PRODUCT for the MANUFACTURER
            - For each PRODUCT:
                - Find the Retail_price of PRODUCT
            - Count total number of PRODUCT for MANUFACTURER
            - Sum all Retail_price of PRODUCT for MANUFACTURER
            - Average sum of all Retail_price over total number of PRODUCT
            - Display total number of PRODUCT for MANUFACTURER
            - Find minimum Retail_price for MANUFACTURER
                - Display minimum Retail_price for MANUFACTURER
            - Find maximum Retail_price for MANUFACTURER
                - Display maximum Retail_price for MANUFACTURER
            - Run **SubReport 1-Drill Down Detail** task
                - Display **Detail** button for accessing report
        - Sort Average of all Retail_price for all MANUFACTURER descending
            - Display first 100 MANUFACTURER

```
select
    m.name,
    count( distinct(p.pid)) as total_products,
    round( avg(p.retail_price)::numeric, 2 ) as avg_retail_price,
    round( min(p.retail_price)::numeric, 2 ) as min_retail_price,
    round( max(p.retail_price)::numeric, 2 ) as max_retail_price
from
    sedw.manufacturer m
inner join sedw.product p on
    p.manufacturer_id = m.id
group by
    m.name
order by
    avg( p.retail_price ) desc
limit 100;
```

SubReport 1 - Drill Down Detail

Abstract Code

User clicks for details in the **Manufacturer's Product Report** to display manufacturer details

- Display summary information from **Manufacturer's Product Report**
- Run the **Generate Drill Down Report 1** task:

```
select
    m.name,p.pid,p.name as product_name,
    round(p.retail_price::numeric, 2 ) as retail_price,
    string_agg(c."name",',')as category_name
from
    sedw.manufacturer m
inner join sedw.product p on
    p.manufacturer_id = m.id
    inner join sedw.belongs_to bt on bt.pid=p.pid
    inner join sedw.category c on c."name"=bt."name"
    where m."name"='garmin'
group by
    m.name,p.pid,p.name
order by
     p.retail_price desc;
```

- Query for MANUFACTURER by Name
    - Display the Name
    - Find Maximum_discount
    - Display Maximum_discount
    - Find each PRODUCT for the MANUFACTURER
    - For each PRODUCT:
        - Find PID of PRODUCT
        - Display PID
        - Find Name of PRODUCT
        - Display Name
        - Find each CATEGORY for PRODUCT
            - For each CATEGORY:
                - Concatenate into one comma-delimited value
        - Display concatenated CATEGORY for PRODUCT
        - Find the Retail_price of PRODUCT
        - Display Retail_price
    - Display each PRODUCT, sorted descending by Price

Report 2 - Category Report

Abstract Code

- User runs the **Category Report** task from the Report Dashboard:

```
select
    c.name,
    count( distinct( bt.pid )) as products_in_category,
    count( distinct( m.id )) as unique_manufacturers,
    round( avg( retail_price )::decimal, 2 ) as avg_retail_price
from
    category c
inner join belongs_to bt on
    bt."name"=c."name"
inner join product p on p.pid=bt.pid
inner join manufacturer m on
    m.id = p.manufacturer_id
group by
    c.name
order by
    c.name asc;
```

- Query for all categories by Name
    - For each CATEGORY:
        - Find each Product in the CATEGORY:
            - Find the MANUFACTURER of PRODUCT
            - Find the Retail_price of PRODUCT
        - Count total number of PRODUCT in CATEGORY
        - Sum all Retail_price of PRODUCT in CATEGORY
        - Average sum of all Retail_price over total number of PRODUCT
        - Count unique MANUFACTURER in CATEGORY
        - Display total number of PRODUCT in CATEGORY
        - Display total number of unique MANUFACTURER in CATEGORY
        - Display average of Retail_price in CATEGORY
    - Display each CATEGORY, sorted ascending by Name

Report 3 - Actual versus Predicted Revenue for GPS Units

Abstract Code

```
● User clicks the "Actual vs. Predicted Revenue for GPS Units Report" button
● For each PRODUCT.Name.BELONGS_TO.CATEGORY.Name = "GPS"
    ● For each PURCHASE_DAY.Date
        ● Sum Store Quantity for all STORE.Store_number on that date
        ● If there is a matching PRODUCT.HAS_A.SALE.Sale_date /*sale price*/
            ● Add (75% * Store Quantity) to Predicted Sales Quantity
            ● Add Store Quantity to DIscounted Sales Quantity
            ● Add (Store Quantity * PRODUCT.HAS_A.SALE.Sale_price) to Actual
              Revenue
        ● Otherwise /*no sale*/
            ● Add Store Quantity to Predicted Sales Quantity
            ● Add Store Quantity to Retail Sales Quantity
            ● Add (Store Quantity * PRODUCT.Retail_price) to Actual Revenue
    ● Calculate Predicted Revenue = (Predicted Sales Quantity * PRODUCT.Retail_Price)
    ● Calculate Delta = (Actual Revenue – Predicted Revenue)
    ● Store PRODUCT.Name, PRODUCT.PID, PRODUCT.Retail_price, Discounted Sales
      Quantity, Retail Sales Quantity, Actual Revenue, Predicted Revenue, and Delta
● Sort Deltas from high to low
● Display report headers
● For each Delta where Delta > 5000 or Delta < -5000
    ● Display PRODUCT.PID, PRODUCT.Name, PRODUCT.Retail_price, Retail Sales Quantity,
      Actual Revenue, Predicted Revenue, Delta
● Display report footers
```

```sql
select
      *
from
      (
      select
            bt.pid,
            p.name,
            p.retail_price,
            get_total.total_units_sold,
            get_discounted_units.units_sold_at_discount,
            get_total.total_units_sold -
get_discounted_units.units_sold_at_discount as units_sold_at_retail_price,

round(((get_total.total_units_sold-get_discounted_units.units_sold_at_disco
unt) * p.retail_price + get_discounted_units.discounted_revenue)::numeric,
            2) as actual_revenue,
            round((get_discounted_units.units_sold_at_discount * 0.75 +
get_total.total_units_sold-get_discounted_units.units_sold_at_discount) *
p.retail_price::numeric,
            2) as predicted_revenue,

round(((get_total.total_units_sold-get_discounted_units.units_sold_at_disco
unt) * p.retail_price + get_discounted_units.discounted_revenue)::numeric,
            2) - round((get_discounted_units.units_sold_at_discount * 0.75
+ get_total.total_units_sold-get_discounted_units.units_sold_at_discount) *
p.retail_price::numeric,
            2) as difference
      from
            category c
      inner join belongs_to bt on
            c.name = bt.name
      inner join product p on
            bt.pid = p.pid
      inner join (
            select
                  tracks_sold.pid,
                  sum(quantity) as units_sold_at_discount,
                  sum(tracks_sold.quantity * sale.sale_price) as
discounted_revenue
            from
                  tracks_sold
            inner join sale on
                  tracks_sold.pid = sale.pid
            inner join sale_sale_date on
                  tracks_sold.date = sale_sale_date.sale_date
                  and sale.sale_id = sale_sale_date.sale_id
            group by
                  tracks_sold.pid ) as get_discounted_units on
            get_discounted_units.pid = bt.pid
      inner join (
            select
                  pid,
                  sum(quantity) as total_units_sold
            from
                  tracks_sold
            group by
```

```
                  pid ) as get_total on
            bt.pid = get_total.pid
      where
            c.name = 'gps') report
where
      abs(report.difference) > 5000
order by
      report.difference desc;
```

Report 4 - Store Revenue By Year by State

Abstract Code

- User clicks the "***Store Revenue by Year by State***" button
- Create *list of states* from CITY.STATE.{}

```
select state_name from sedw.state;
```

```
- Present drop-down menu containing list of states
- User selects State from menu
- For each STORE LOCATED_IN State
    - For each PURCHASE_DAY.Date
        - Calculate Year from PURCHASE_DAY.Date
        - For each PRODUCT
            - If PRODUCT.HAS_A.SALE.Date = PURCHASE_DAY.Date, add
              (PRODUCT.PURCHASE_DAY.Quantity *
              PRODUCT.HAS_A.SALE.Sale_Price) to Store Revenue.Year
            - else add (PRODUCT.PURCHASE_DAY.Quantity * PRODUCT.Retail_price)
              to Store Revenue.Year
    - Store STORE.Store_number, STORE.Street_address, STORE.LOCATED_IN.City_name,
      Store Revenue.{}
- For each Year from oldest to newest
    - Sort by Store Revenue.Year
    - Display report headers
    - For each STORE.Store_number
        - Display STORE.Store_number, STORE.Street_address,
          STORE.LOCATED_IN.City_name, Year, Store Revenue.Year
    - Display report footers
```

```
\echo
'---------------------------------------------------------------------------
--------------------------'
\echo 'Report 4 State select '
```

```
\echo  'Below has New York hard coded; this will be taken from drop-down'
\echo
'-----------------------------------------------------------------------------
--------------------------'
-- Report 4 State select
-- Below has New York hard coded; this will be taken from drop-down
select
store_number,
street_number ,
street_name ,
city_name,
year,
sum(revenue)as total_revenue
from
(  -- First we need to get product sales with discount applied, then we can
aggregate
   select
      s.store_number,
      s.street_number ,
      s.street_name ,
      c.city_name,
      extract ( year from ts."date" )::integer as year,
      -- ts.quantity*p.retail_price,
      p.pid,
      p."name",
      sale.pid,
      ts."date",
      ts.quantity,
      ROUND((COALESCE(sale.sale_price,
p.retail_price)*ts.quantity)::numeric,2) as revenue
      -- s.street_number || ' ' || s.street_name || ' ' || c.city_name ||
',' || c.state_name as address
from
      sedw."store" s
inner join sedw.city c on c.id = s.city_id and c.state_name = 'New York'
inner join sedw.tracks_sold ts on ts.store_number = s.store_number
inner join sedw.product p on p.pid = ts.pid
left join sedw.sale on sale.pid=p.pid
left join sedw.sale_sale_date sd on sd.sale_id=sale.sale_id
)apply_discount
group by
```

```
      store_number,
      street_number ,
      street_name ,
      city_name,
      year
      order by year asc, total_revenue desc;
```

Report 5 - Air Conditioners on Groundhog Day

Abstract Code

- User clicks the "***Air Conditioners on Groundhog Day Report***" button
- For each PRODUCT.Name.BELONGS_TO .CATEGORY.Name = "Air Conditioner"
    - For each PURCHASE_DAY.Date
        - Calculate *Year*
        - Add Product.PURCHASE_DAY.Quantity to *Total Units.Year*
    - For each PURCHASE_DAY.Date that is a February 2
        - Calculate *Year*
        - add PRODUCT.PURCHASE_DAY.Quantity to *Groundhog Units.Year*
- For each *Year* (from lowest to highest)
    - Display *Year*, *Total Units.Year*, =(*Total Units.Year* / 365), *Groundhog Units.Year*

/* this assumes every year has air conditioner sales, and that every year has air conditioner sales on
Groundhog Day */

```
select * from (
                select extract(YEAR from ssd.sale_date)
as year,
                       count(sale.sale_id)
AS sales,
                       to_char(count(sale.sale_id)::FLOAT / 365,
'99999.000') AS daily_sales
                from sale
                       join sale_sale_date ssd on sale.sale_id =
ssd.sale_id
                       join product p on sale.pid = p.pid
                       join belongs_to bt on p.pid = bt.pid
                where bt.name = 'Air Conditioner'
                group by extract(YEAR from ssd.sale_date)
                order by extract(YEAR from ssd.sale_date) ASC
              ) as tab1
```

```
JOIN (
  select extract(YEAR from ssd.sale_date) as year,
         count(sale.sale_id)              AS GroundhogDay_Sales
  from sale
         join sale_sale_date ssd on sale.sale_id = ssd.sale_id
         join product p on sale.pid = p.pid
         join belongs_to bt on p.pid = bt.pid
  where bt.name = 'Air Conditioner'
    and extract(DOY from ssd.sale_date) = 33
  group by extract(YEAR from ssd.sale_date)
  order by extract(YEAR from ssd.sale_date) ASC
) as tab2
on tab1.year = tab2.year;
```

Report 6 - State with Highest Volume for each Category

Abstract Code

- Show ***year and month drop down menus*** populated with available dates from database

```
select
    extract( year from ts.date ) as the_year,
    extract( month from ts.date ) as the_month
from
    sedw.tracks_sold ts
group by
    the_year,
    the_month;
```

- Show ***"generate report button"***
- Upon choosing *a year and month* and clicking the ***generate report button***:
  - Query the database to return the State and the highest sales in each Category
  - Display a row with the Category name, the State with highest sales in that Category, the number of sales in that Category, ***a button*** that generates SubReport 6.
    - *If* a Category has more than one State as equal in highest sales, Display a row for each state.

```
select final_output.* from
(select
      state_name,
      category,
      the_year,
      the_month,
      units_sold,
      total_revenue,
      rank() over ( partition by state_name, category order by units_sold
desc )
from (
      select
state_name,
category,
the_year,
the_month,
sum(quantity) as units_sold,
sum(revenue)as total_revenue
from
(  -- First we need to get product sales with discount applied, then we can
aggregate
   select
    state.state_name,
    extract( year from ts.date )::integer as the_year,
    extract( month from ts.date )::integer as the_month,
    ROUND((COALESCE(sale.sale_price,
p.retail_price)*ts.quantity)::numeric,2) as revenue,
    ts.quantity,
    b."name" as category
from sedw."store" s
inner join sedw.city c on c.id = s.city_id -- and c.state_name = 'New York'
inner join sedw.tracks_sold ts on ts.store_number = s.store_number
inner join sedw.product p on p.pid = ts.pid
left join sedw.sale on sale.pid=p.pid
left join sedw.sale_sale_date sd on sd.sale_id=sale.sale_id
inner join sedw.state on state.state_name=c.state_name
inner join sedw.belongs_to b on b.pid=p.pid
where 1=1
and extract( year from ts.date )::integer='2018'
and  extract( month from ts.date )::integer='01'
)apply_discount
group by
    state_name,
    category,
    -- store_number,
    -- street_number ,
    -- street_name ,
    -- city_name,
    the_year,
    the_month)category_by_state  order by category)final_output where
rank=1;
```

SubReport 6 - Drill Down Detail:

Abstract Code

- Upon a ***button*** click in report 6:
    - Show ***a header*** containing original criteria from parent report (*category, year/month, state*).
    - Show column headings of: Store address, Store id, city, Manager name, Manager email
    - Query database for Stores that are located in the passed-in state from parent report
        - get the Stores' address and store_id
    - Query database for City of the store
    - Query database for all Managers located in a store
        - get the Managers name and email_address
    - *for each* store in the arranged list of stores Display a row with Store name, Store id, City, Manager name, Manager email
        - *if* a Store has multiple Managers, display a new row *for each* manager in that store

```
echo
'---------------------------------------------------------------------------
--------------------------'
\echo 'Report 6 Drill-Down Hard coded in where clause'
\echo
'---------------------------------------------------------------------------
--------------------------'
select
topstore.store_number,
topstore.address,
topstore.city_name,
m.first_name,
m.last_name,
m.email_address
from
(
   select
   s.store_number,
   s.street_number || ' ' || s.street_name || ' ' || c.city_name || ',' ||
c.state_name as address,
    state.state_name,
    c.city_name,
    extract( year from ts.date )::integer as the_year,
    extract( month from ts.date )::integer as the_month,
    b."name" as category
from sedw."store" s
inner join sedw.city c on c.id = s.city_id -- and c.state_name = 'New York'
inner join sedw.tracks_sold ts on ts.store_number = s.store_number
inner join sedw.product p on p.pid = ts.pid
left join sedw.sale on sale.pid=p.pid
left join sedw.sale_sale_date sd on sd.sale_id=sale.sale_id
inner join sedw.state on state.state_name=c.state_name
inner join sedw.belongs_to b on b.pid=p.pid
where 1=1
and extract( year from ts.date )::integer='2018'
and  extract( month from ts.date )::integer='01'
group by s.store_number, address,
city_name,state.state_name,the_year,the_month,b."name"
)topstore
inner join sedw.managed_by mb on mb.store_number=topstore.store_number
inner join sedw.manager m on m.email_address=mb.email_address
    where 1=1
    and topstore.category='electronics' and topstore.state_name='New York'
and the_year='2018' and the_month='1'
      order by topstore.store_number asc
   ;
```

Report 7 - Revenue by Population

Abstract Code

- **Query** the database for the number_of_years in the database
  - Arrange years in ascending order (oldest to newest)
- *For each* year in the database:
  - Query the database to get the population of each City.
    - Arrange cities into predefined categories in ascending order (smallest to largest)
  - Query database to get the revenue for each store in a City.
  - Calculate average_revenue for population category
  - Display the data for the current year.
    - Initially the display will have city categories as the rows and years as the columns
- Show ***pivot button*** for switching "years"/"city category" as either columns or rows.
  - Upon pressing the ***button*** switch which category is rows.

- If rows are represented by years, make years the columns and the city categories the rows
- Else make rows the years and city categories the columns.

```
select
      size,
      round(avg(revenue)::numeric,2) as average_revenue,
      year
from
      (
values ('small',
0,
3700000),
('medium',
3700000,
6700000),
('large',
6700000,
9000000),
('extra large',
9000000,
10000000000)) as t (size,
      min,
      max)
inner join (
      select
            city_id,
            s.store_number,
            year,
            population,
            sum(revenue)as revenue
      from
            city c
      inner join store s on
            c.id = s.city_id
      inner join store_revenue sr on
            s.store_number = sr.store_number
      group by
            city_id,
            s.store_number,
            sr.year,
            population ) as t2 on
      t2.population >= t.min
      and t2.population < t.max
group by
      size,
      year
order by
      size desc, --the reason size is desc is because alphabetically, this
would make the categories ascending
      year asc;
```

Update City Population

Abstract Code

- User clicks on the Update City link from the **Data Warehouse Admin** form.
- Show **Cities** form
- For each City in CITY.
    - Display CITY attributes: CITY.City_name, CITY.State and CITY.Population

```
 select city_name,state_name,population from sedw.city;
```

- User updates population (‘$population’) input field to desired value.
- When the **Update** button is clicked
    - Validate the *$population* is an integer
        - If valid: Update the populate for CITY.Population to the value stored in *$population.*

```
the_id:=select id from sedw.city where city_name='$city_name' and
state_name='$state';

update sedw.city set population='$population' where id=@the_id;
```

- Else: Prompt user to use a valid integer, validate and Update.

Maintain Holiday

Abstract Code

- User clicks on the *Maintain Holiday* link from the **Data Warehouse Admin** form.
    - Query database for all holidays and sort in ascending order by date.
        - The **Maintain/View Holiday** form will show with a table of holidays sorted in ascending order by date.
        -
    - Click **Add** button – User will be prompted for Holiday name $holiday_name and $date. The **CREATE** task will be run with $holiday_name and $date.
        If holiday exists (count > 0): query databases for manager where Holiday.name = $holiday_name and Date=$date

```
select
    count( id )
from
    holiday h
where
    lower( h."name" )= '$holiday' and date = '$date'
```

- Inform the user that the holiday already exists in the database.
- Else: Add the new Holiday into the database.

```
insert
    into
       holiday ( id, date, name )
    values ( nextval( 'sedw.holiday_id_seq' ), '$date', '$holiday_name' );
```

Maintain Manager

Abstract Code

- User clicks on the Maintain Manager link from the **Data Warehouse Admin** form.
    - The **Manager** form will show a table of managers along with a drop down for Stores and Manager Names for selection.
        - Query database for unique STORE.Store_number

        ▪ Query database for unique MANAGER.Manager_name

- Populate Drop downs with results from queries
- Click **Add** button – User will be prompted for MANAGER.name *$manager* and STORE.Store_number *$store*. The **CREATE** task will be run with these arguments.

```
insert into sedw.manager
(email_address, first_name, last_name, status)
values('$email_address', '$first_name', '$last_name',
'active'::employment_status);
```

- ▪ User selects either a *$store* or a manager name $manager_name
- ▪ If store: query databases for manager where STORE.Store_number = $store
- ▪ If manager: query databases for manager where MANAGER.Manager_name = $name
    - Update the **<u>Manager</u>** form with information retrieved from the database. If the manager manages multiple stores, each row will be displayed on the table with the STORE.Store_number.
    - Click **Delete Manager** button – Present the user with confirmation that the manager and his associations will be removed from database, if yes proceed, else cancel.

```
delete from sedw.manager
where email_address='$email_address';
```

- Click **Unassign** button – This will remove the association between the MANAGER and STORE.

```
delete from sedw.managed_by
where store_number=$store_number AND email_address='$email_address';
```

- Click **Assign** button – The user will be prompted to select the STORE.Store_number from a drop down. Once selected the database will create the association with MANAGER and STORE.

```
insert into sedw.managed_by
(store_number, email_address)
values($store_number, '$email_address');
```

Store_Revenue View

Abstract Code

- This view returns the revenue each store made on a given item. It was a commonly used query for many reports but we didn't realize it until report 7.
- It is used in the queries for report 7.

```
CREATE OR REPLACE VIEW sedw.store_revenue
AS SELECT s.store_number,
    s.street_number,
    s.street_name,
    c.city_name,
    date_part('year'::text, ts.date)::integer AS year,
    p.pid,
    p.name,
    ts.date,
    ts.quantity,
    round((COALESCE(sale.sale_price, p.retail_price) * ts.quantity::double
precision)::numeric, 2) AS revenue
   FROM store s
     JOIN city c ON c.id = s.city_id AND c.state_name::text = 'New
York'::text
     JOIN tracks_sold ts ON ts.store_number = s.store_number
     JOIN product p ON p.pid = ts.pid
     LEFT JOIN sale ON sale.pid = p.pid
     LEFT JOIN sale_sale_date sd ON sd.sale_id = sale.sale_id;

-- Permissions

ALTER TABLE sedw.store_revenue OWNER TO team075;
GRANT ALL ON TABLE sedw.store_revenue TO team075;
```