DATA C88C Summer 2025 — More SQL

Discussion 11: July 31, 2025

After you finish your Thanksgiving dinner, you realize that you still need to buy gifts for all your loved ones over the holidays. However, you also want to spend as little money as possible (you're not cheap, just looking for a great sale).

This question utilizes the following tables:

`products`

```
CREATE TABLE products AS
  SELECT 'phone' AS category, 'uPhone' AS name, 99.99 AS MSRP, 4.5 AS rating UNION
  SELECT 'phone'            , 'rPhone'        , 79.99        , 3            UNION
  SELECT 'phone'            , 'qPhone'        , 89.99        , 4            UNION
  SELECT 'games'            , 'GameStation'   , 299.99       , 3            UNION
  SELECT 'games'            , 'QBox'          , 399.99       , 3.5          UNION
  SELECT 'computer'         , 'iBook'         , 112.99       , 4            UNION
  SELECT 'computer'         , 'wBook'         , 114.29       , 4.4          UNION
  SELECT 'computer'         , 'kBook'         , 99.99        , 3.8               ;
```

`inventory`

```
CREATE TABLE inventory AS
  SELECT 'Hallmart' AS store, 'uPhone' AS item, 99.99 AS price UNION
  SELECT 'Targive'          , 'uPhone'        , 100.99         UNION
  SELECT 'RestBuy'          , 'uPhone'        , 89.99          UNION

  SELECT 'Hallmart'         , 'rPhone'        , 69.99          UNION
  SELECT 'Targive'          , 'rPhone'        , 79.99          UNION
  SELECT 'RestBuy'          , 'rPhone'        , 75.99          UNION

  SELECT 'Hallmart'         , 'qPhone'        , 85.99          UNION
  SELECT 'Targive'          , 'qPhone'        , 88.98          UNION
  SELECT 'RestBuy'          , 'qPhone'        , 87.98          UNION

  SELECT 'Hallmart'         , 'GameStation'   , 298.98         UNION
  SELECT 'Targive'          , 'GameStation'   , 300.98         UNION
  SELECT 'RestBuy'          , 'GameStation'   , 310.99         UNION

  SELECT 'Hallmart'         , 'QBox'          , 399.99         UNION
  SELECT 'Targive'          , 'QBox'          , 390.98         UNION
  SELECT 'RestBuy'          , 'QBox'          , 410.98         UNION

  SELECT 'Hallmart'         , 'iBook'         , 111.99         UNION
  SELECT 'Targive'          , 'iBook'         , 110.99         UNION
  SELECT 'RestBuy'          , 'iBook'         , 112.99         UNION

  SELECT 'Hallmart'         , 'wBook'         , 117.29         UNION
  SELECT 'Targive'          , 'wBook'         , 119.29         UNION
  SELECT 'RestBuy'          , 'wBook'         , 114.29         UNION

  SELECT 'Hallmart'         , 'kBook'         , 95.99          UNION
  SELECT 'Targive'          , 'kBook'         , 96.99          UNION
  SELECT 'RestBuy'          , 'kBook'         , 94.99              ;
```

`stores`

```
CREATE TABLE stores AS
  SELECT 'Hallmart' AS store, '50 Lawton Way' AS address, 25 AS Mbs UNION
  SELECT 'Targive'          , '2 Red Circle Way'        , 40        UNION
  SELECT 'RestBuy'          , '1 Kiosk Ave'             , 30            ;
```

Q1: Price Check

Let's start off by surveying our options. Using the `products` table, write a query that creates a table `average_prices` that lists categories and the average price of items in the category (using MSRP as the price). Finally, sort the resulting rows by highest to lowest average price.

You should get the following output:

category	average_price
games	350.0
computer	109.0
phone	90.0

Due to floating point errors, you may get average price values that are slightly off, such as 349.99 instead of 350.

```
CREATE TABLE average_prices AS
  SELECT category, AVG(msrp) AS average_price -- alternatively: SELECT category, SUM(msrp
    ) / COUNT(*) AS average_price
  FROM products
  GROUP BY category
  ORDER BY average_price DESC;
```

Q2: Lowest Prices

Now, you want to figure out which stores sell each item in products for the lowest price. Write a SQL query that uses the `inventory` table to create a table `lowest_prices` that lists items, the stores that sells that item for the lowest price, and the price that the store sells that item for. Finally, sort the resulting rows alphabetically by item name.

You should expect the following output:

store	item	lowest_price
Hallmart	GameStation	298.98
Targive	QBox	390.98
Targive	iBook	110.99
RestBuy	kBook	94.99
Hallmart	qPhone	85.99
Hallmart	rPhone	69.99
RestBuy	uPhone	89.99
RestBuy	wBook	114.29

In other variants of SQL such as PostgreSQL, you are not allowed to include columns in the `SELECT` clause if they are not included in the `GROUP BY` clause or an aggregation function in the `SELECT` clause. However, SQLite allows this and this variant is what we use in this class.

```
CREATE TABLE lowest_prices AS
  SELECT store, item, MIN(price) AS lowest_price
  FROM inventory
  GROUP BY item
  ORDER BY item;
```

Q3: Shopping List

You want to make a shopping list by choosing the item that is the best deal possible for every category. For example, for the "phone" category, the uPhone is the best deal because the MSRP price of a uPhone divided by its ratings yields the lowest cost. That means that uPhones cost the lowest money per rating point out of all of the phones. Note that the item with the lowest MSRP price may not necessarily be the best deal.

Write a query to create a table `shopping_list` that lists the items that you want to buy from each category.

After you've figured out which item you want to buy for each category, add another column that lists the store that sells that item for the lowest price.

Finally, sort the resulting table alphabetically by item name.

Hint: What table have you already defined that gives you the store that sells a given item for the lowest price? How can you use that to determine which item in each category is the best deal?

You should expect the following output:

item	store
GameStation	Hallmart

item	store
uPhone	RestBuy
wBook	RestBuy

Note 1: In other variants of SQL such as PostgreSQL, you are not allowed to include columns in the `SELECT` clause if they are not included in the `GROUP BY` clause or an aggregation function in the `SELECT` clause. However, SQLite allows this and this variant is what we use in this class.

Note 2: In other variants of SQL such as PostgreSQL, you cannot provide a numeric type as the predicate of the `HAVING` clause (the staff solution uses `HAVING MIN(...)` which isn't allowed since `MIN` returns a number but `HAVING` expects a boolean). However, SQLite allows this and this variant is what we use in this class.

```
CREATE TABLE shopping_list AS
  SELECT p.name, l.store
  FROM products AS p, lowest_prices AS l
  WHERE l.item = p.name
  GROUP BY p.category
  HAVING MIN(p.msrp / p.rating)
  ORDER BY p.name;

  -- Alternate solution using implicit inner join:
  -- SELECT p.name, l.store
  -- FROM products AS p, lowest_prices AS l
  -- WHERE l.item = p.name
  -- GROUP BY p.category
  -- HAVING MIN(p.msrp / p.rating)
  -- ORDER BY p.name;
```

Q4: Bandwidth

Using the Mbs (`megabits`) column from the `stores` table, write a query to calculate the total amount of bandwidth needed to get everything in your shopping list (assume the `shopping_list` table from the previous question is already defined).

You should expect the following output:

total_bandwidth
85

```
CREATE TABLE bandwidth AS
  SELECT SUM(s.mbs) AS total_bandwidth
  FROM stores AS s, shopping_list AS sl
  WHERE s.store = sl.store;

  -- Alternate solution using explicit inner join:
  -- SELECT SUM(s.mbs) AS total_bandwidth
  -- FROM stores AS s
  -- JOIN shopping_list AS sl
  -- ON s.store = sl.store;
```