Phase 2 AC+SQL | CS 6400 - Spring 2023 | **Team 021**

Table of Contents

Abstract Code:

Main Menu

Abstract Code

- Show **"Enter My Household Info"** and **"View Reports/Query data"** links.
- Upon:
  - **Enter My Household Info** - Jump to **Enter My Household Info** task.
  - **View Reports/Query data** - Jump to **View Reports** task.

Enter My Household Info

Basic Household Info

Abstract Code

- Display form to allow user to enter basic hosuehold info:
  - email, zip, type, square feet, thermostat settings, utilities
- If User clicks on Next from Basic Household Info screen:
  - Validate email:
    - If Email already exists in database (result of below query is 1), display a warning that email is already found, pick a new email. Disable next button until new email entered.

```
SELECT COUNT(email) FROM Household WHERE email = $Email;
```

- Validate Zip Code:
    - Check if Zip exists in Location (result of the below query is greater than zero). If not, display a warning that the zip code is invalid. Disable next button until new Zip or City or State entered.

```
SELECT COUNT(postal_code) FROM Location WHERE postal_code = $zip_code;
```

- Validate the required fields are filled in:
    - Type, Square Footage
- If no value is entered for *Thermostat Setting Heating*, validate the user checked the "No Heat" box.
- If no value is entered for *Thermostat Setting Cooling*, validate the user checked the "No Cooling" box.
- If any validation checks fail, present the user an error message saying what's missing.
- If all validation checks successful, insert data in the database, then go to the **Appliance - Add** task.

- Need to set an application variable called `on_grid` that is True if any Utilities box is checked, and False if no Utilities boxes are checked

```
INSERT INTO Household
(
    email,
    postal_code,
    type,
    square_footage,
    thermostat_setting_heating,
    thermostat_setting_cooling,
    on_grid
)
VALUES (
    $email,
    $zip_code,
    $home_type,
    $square_footage,
    $thermostat_heating,
    $thermostat_cooling,
    $on_grid
);
```

- Initialize 2 client-side variables for tracking the Order Entered of Appliances and Power Generators. `$Appliance_Order` and `$Power_Order`. Set them both to 1.
- Run this with one entry in VALUES for each utility type checked.

```
INSERT INTO HouseholdUtility (email, utility)
VALUES
($email, $Utility_Type0),
($email, $Utility_Type1),
...;
```

Appliance - Add

Abstract Code

- User selects an Appliance Type from a drop-down (Air Handler or Water Heater)
  - If user selects Air Handler, display the Air Handler fields:
    - Manufacturer, Model Name, EER, Energy Source, and Method (a multi-select allowing Air Conditioner, Heater, and/or Heat Pump)
    - Manufacturer will be a dropdown list of manufacturers who make Air Handlers:

```
SELECT name FROM Manufacturer;
```

- If user selects Air Conditioner method:
  - Add input for EER
- If user selects Heater:
  - Add input for Energy Source
- If user selects Heat Pump:
  - Add inputs for HSPF and SEER
- If user selects Water Heater, display Water Heater fields:
  - Energy Source, Current Temperature, Capacity
- If user hits Add:
  - Validate the required fields are filled in
  - Validate data types match:
    - HSPF, SEER, EER, Capacity are numbers, decimals allowed
    - BTU, Current Temperature are numbers, round to whole number when saving.
  - Insert to the database:

- Appliance Order Entered coming from client-side application variable "$Appliance_Order", created above. It incremements by 1 every time a new appliance is added. It doesn't go down when an appliance is deleted, meeting the requirement that deleted order numbers not be re-used. Because users can't edit their households later, we don't have to store this variable server-side, or look it up with SQL from the tables when inserting.
- `$email` is an application variable with the user's email (from previous steps).
- If more than one type is selected/validated, run through this whole process once for each appliance type. Order is important because the order entered in Appliance is a foreign key in WaterHeater, etc,

so do not insert multiple appliance types into Appliance without first inserting the subtype in the corresponding table.

```
INSERT INTO Appliance (
    appliance_number,
    email),
    manufacturer_id,
    btu_rating,
    model_name,
    type
)
VALUES (
    $Appliance_Order,
    $email,
    $manufacturer_id,
    $btu_rating,
    $model_name,
    $type
);
```

If user entered a **Water Heater**:

```
INSERT INTO WaterHeater (
    email,
    appliance_number,
    energy_source,
    current_tempurature,
    capacity)
VALUES (
    $email,
    $Appliance_Order,
    $energy_source,
    $current_tempurature,
    $capacity);
```

If user entered an **Air Handler:**

```
INSERT INTO AirHandler (email, appliance_number)
VALUES ($email, $Appliance_Order);
```

If user entered a **Heat Pump:**

```
INSERT INTO HeatPump (
    email,
    appliance_number,
    hspf,
    seer)
VALUES (
    $email,
    $Appliance_Order,
    $hspf,
    $seer);
```

If user entered an **Air Conditioner:**

```
INSERT INTO AirConditioner (
    email,
    appliance_number,
    eer)
VALUES (
    $email,
    $Appliance_Order,
    $eer);
```

If user entered a **Heater:**

```
INSERT INTO Heater (
    email,
    appliance_number,
    energy_source)
VALUES (
    $email,
    $Appliance_Order,
    $energy_source);
```

- Add 1 to `$Appliance_Order`

Appliance - List

Abstract Code

- Read household's appliances using household email.

```
SELECT * FROM Appliance WHERE Appliance.email = $email;
```

- If list is empty, run **Appliances - Add** (user must have deleted last appliance)
- Display:
    - List of appliances that are not Deleted.

  - Button to delete each appliance
  - Button to Add Another Appliance
  - Button saying Next.
- If user hits Add Another Appliance, run **Appliances - Add**.
- If user hits Delete, run **Appliances - Delete**
- If user hits Next, run **Power Generation - Add**.

Appliance - Delete

Abstract Code

Delete the selected appliance from **Appliance:**

- The application is already holding Appliance Order Entered from "Appliances - List".
- Rely on ON DELETE CASCADE to delete all subclass types of an Appliance

```
DELETE FROM Appliance
WHERE Appliance.email = $email
AND Appliance.appliance_number = $appliance_number;
```

Power Generator - Add

Abstract Code

- Display:
  - Inputs for Type, Monthly kWh, Capacity,
  - If household is not off-grid, display skip button.

```
SELECT on_grid From Household WHERE Household.email = $email;
```

- Display add button

- If user hits Add:

  - Validate required fields are filled in
  - Validate Monthly kWh and Capacity are numbers. Round to whole number when saving.
  - If above validation checks are successful:
    - Insert to database:

- Like with Appliance, Power Generator needs an order entered, and we can't use an auto-increment data type because it needs to start from 1 for each household. We'll use another client-side application variable `$Power_Order` that starts at 1 and adds 1 each time a power generator is added. This was created in the Household Add task. It never goes down, so pg_number doesn't re-use deleted numbers.

- `$email` is still stored in the application from previous steps.
- As with Appliance, when we save the order entered to the database, we'll also store it as an application variable to use in the Update.

```
INSERT INTO PowerGenerator (
    email,
    pg_number,
    type,
    avg_monthly_kilowatt_hours,
    capacity
)
VALUES (
    $email,
    $Power_Order,
    $type,
    $avg_monthly_kilowatt_hours,
    $capacity
);
```

- Add 1 to `$Power_Order`
- Run **Power Generator - List** (SQL in that task below)
- Else display a data validation error and disable Add button until a change to a user-input is made
- If user hits skip button, run **Thank User**

Power Generator - List

Abstract Code:

```
SELECT
*
FROM
PowerGenerator
WHERE PowerGenerator.email = $email
```

- Display:
  - Button saying "Add More Power"
  - Button saying "Finish"
  - Button saying "Delete" for each Power Generator listed

- If user hits "Add More Power", run **Power Generator - Add**
- If user hits "Delete", run **Power Generator - Delete**
- If user hits "Finish", validate household is on_grid OR at least 1 power generator exists for the household (below query returns at least 1), then run **Thank User**

```
SELECT
(SELECT COUNT(*) FROM PowerGenerator WHERE email = $email)
+
(SELECT COUNT(*) FROM Household WHERE email = $email AND on_grid);
```

- If 0 power generators exist and household is off-grid, run **Power Generator - Add**.

Power Generator - Delete

Abstract Code

- Delete the selected Power Generator
- `$pg_number` is known from the Power Generator - List task above. `$email` is from both that and earlier steps.

```
DELETE FROM PowerGenerator
WHERE PowerGenerator.email = $email
AND PowerGenerator.pg_number = $pg_number;
```

Thank User

Abstract Code

- Display a message thanking the user, and a link to the main menu. When clicked, run **Main Menu**
- No SQL here.

View Reports

Abstract Code:

- Display links for:
  - ✓ Top 25 popular manufacturers.
  - ✓ Manufacturer/model search
  - ✓ Heating/cooling method details
  - ✓ Water heater statistics by state
  - ✓ Off-the-grid household dashboard
  - ✓ Household averages by radius*
- Clicking each link should go to the respective report task

Top 25 Popular Manufacturers

Abstract Code:

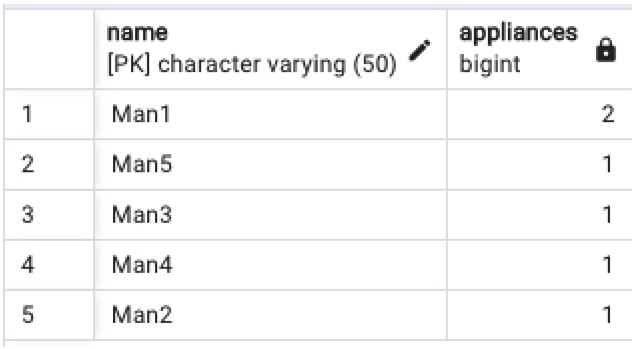
1. Query Appliance and Manufacturer to Get a count of Appliances grouped by Manufacturer *Name*. Sort by number of appliances descending, keep the top 25.
2. If user clicks on a manufacturer, query the Appliance entity grouped by the *Type* attribute to get a count of appliances by type for that Manufacturer *Name*. Display a drilldown table with that company's name as the title and columns for each type.

---

1. Tested using seed data from playground/data.py

```
SELECT Manufacturer.name,
    COUNT(Appliance) AS Appliances
FROM Manufacturer
    INNER JOIN Appliance ON Manufacturer.name = Appliance.manufacturer_name
GROUP BY Manufacturer.name
ORDER BY Appliances DESC
LIMIT 25;
```

▶ Output



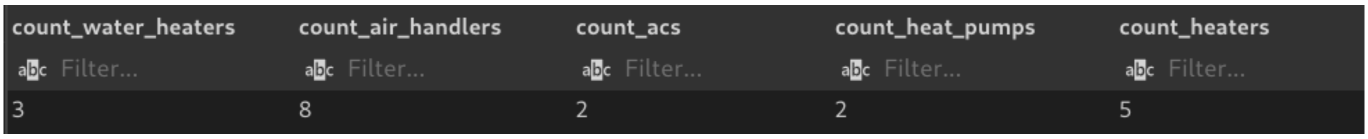
	name [PK] character varying (50)	appliances bigint
1	Man1	2
2	Man5	1
3	Man3	1
4	Man4	1
5	Man2	1

2. Tested using seed data from playground/data.py

- User selected manufacturer `Man1` in this example

```
SELECT COUNT(DISTINCT WaterHeater) AS count_water_heaters,
    COUNT(DISTINCT Appliance) FILTER (
        WHERE Appliance.type = 'Air Handler'
    ) AS count_air_handlers,
    COUNT(DISTINCT AirConditioner) AS count_acs,
    COUNT(DISTINCT HeatPump) AS count_heat_pumps,
    COUNT(DISTINCT Heater) AS count_heaters
FROM Appliance
    INNER JOIN Manufacturer ON Appliance.manufacturer_name =
Manufacturer.name
    LEFT JOIN AirConditioner USING (appliance_number, email)
    LEFT JOIN WaterHeater USING (appliance_number, email)
    LEFT JOIN HeatPump USING (appliance_number, email)
    LEFT JOIN Heater USING (appliance_number, email);
```

▶ Output



count_water_heaters	count_air_handlers	count_acs	count_heat_pumps	count_heaters
3	8	2	2	5

Manufacturer/Model Search

Abstract Code:

- If a user hits the "Search" button:
  - If there are no characters in the search text input, display a warning to the user.
  - If there are characters, query Manufacturer *Name* and Appliance *Model Name*. Display a table with all distinct models (and their manufacturer) where either the model or manufacturer name matched the search query.
  - Results should be ordered by manufacturer name (ascending), then model name (ascending).
  - Highlight matching strings in green.

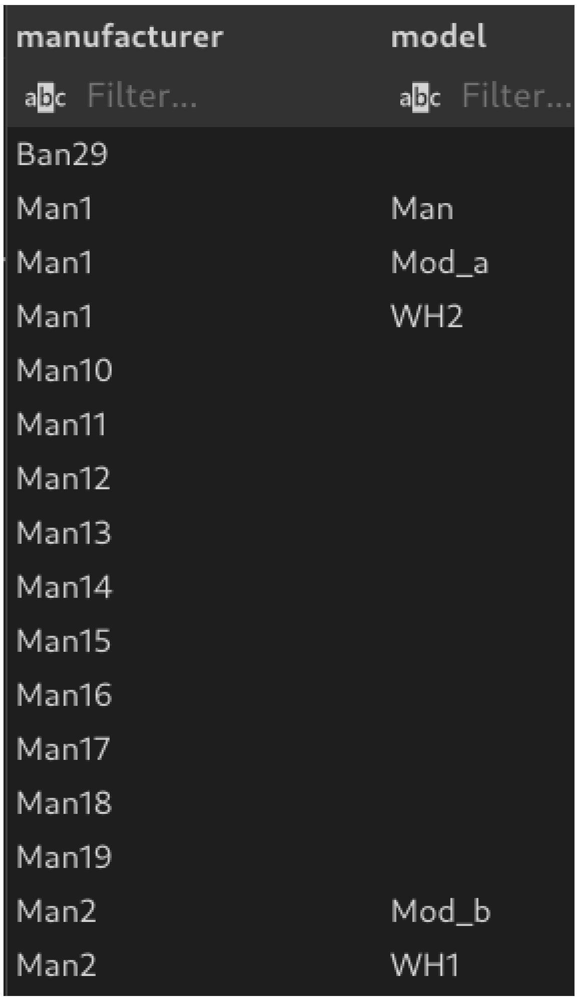
---

1. Tested using seed data from playground/data.py

- Manufacturer and Model match is partial and case insensitve
- Models can be null for a specific Manufacturer
- User input will be converted to lowercase before searching against lowercase versions of the model/manufacturer
- User search input was an in this example

```
SELECT Manufacturer.name as manufacturer,
    COALESCE(Appliance.model_name, '') AS model
FROM Manufacturer
    LEFT JOIN Appliance on Manufacturer.name = Appliance.manufacturer_name
WHERE POSITION('an' IN LOWER(Appliance.model_name)) > 0
    OR POSITION('an' IN LOWER(Manufacturer.name)) > 0
ORDER BY Manufacturer.name ASC,
    Appliance.model_name ASC;
```

▶ Output



manufacturer	model
Ban29	
Man1	Man
Man1	Mod_a
Man1	WH2
Man10	
Man11	
Man12	
Man13	
Man14	
Man15	
Man16	
Man17	
Man18	
Man19	
Man2	Mod_b
Man2	WH1

Heating/Cooling Method Details

Abstract Code:

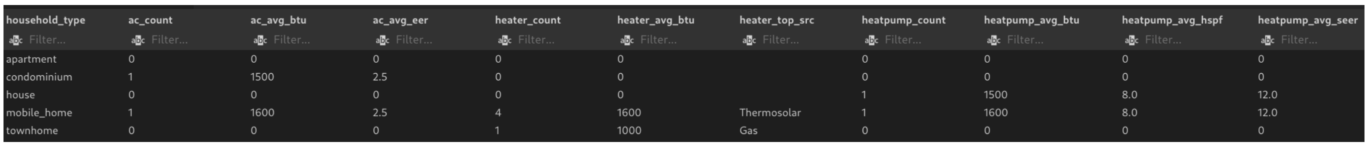
1. Query **Air Conditioner** grouped by **Household** *Type* (found via tracing identifying relationships from Air Conditioner to Household), returning a count of Air Conditioners, average BTU (rounded to whole number) and average EER (rounded to 0.1).
2. Query **Heater** grouped by **Household** *Type* (found via tracing identifying relationships from Air Conditioner to Household), returning a count of Heaters, average *BTU* (rounded to whole number), and most common **Heater** *Energy Source* for each **Household** *Type*.
3. Query **Heat Pump** grouped by **Household** *Type* (found via tracing identifying relationships from Air Conditioner to Household), returning a count of Heat Pumps, average *BTU* (rounded to whole number), and average SEER and HSPF (rounded to 0.1).
4. Group by **Household** *Type* so that all results can be displayed in a single table with 1 column for each measure, and a row for each household type.

```
SELECT Household.household_type,
    COUNT(AirConditioner) AS ac_count,
    COALESCE(ROUND(AVG(btu_rating) FILTER (WHERE eer IS NOT NULL)),0) AS
ac_avg_btu,
    COALESCE(ROUND(AVG(eer), 1),0) AS ac_avg_eer,

    COUNT(Heater) AS heater_count,
    COALESCE(ROUND(AVG(btu_rating) FILTER (WHERE energy_source IS NOT
NULL)),0) AS heater_avg_btu,
    COALESCE(MODE() WITHIN GROUP (ORDER BY energy_source), '') AS
heater_top_src,

    COUNT(HeatPump) AS heatpump_count,
    COALESCE(ROUND(AVG(btu_rating) FILTER (WHERE hspf IS NOT NULL)), 0) AS
heatpump_avg_btu,
    COALESCE(ROUND(AVG(hspf), 1), 0) AS heatpump_avg_hspf,
    COALESCE(ROUND(AVG(seer), 1), 0) AS heatpump_avg_seer
FROM Household
    LEFT JOIN Appliance USING (email)
    LEFT JOIN AirConditioner USING (email, appliance_number)
    LEFT JOIN Heater USING (email, appliance_number)
    LEFT JOIN HeatPump USING (email, appliance_number)
GROUP BY Household.household_type;
```

▶ Output



household_type	ac_count	ac_avg_btu	ac_avg_eer	heater_count	heater_avg_btu	heater_top_src	heatpump_count	heatpump_avg_btu	heatpump_avg_hspf	heatpump_avg_seer
Filter...	Filter...	Filter...	Filter...	Filter...	Filter...	Filter...	Filter...	Filter...	Filter...	Filter...
apartment	0	0	0	0	0		0	0	0	0
condominium	1	1500	2.5	0	0		0	0	0	0
house	0	0	0	0	0		1	1500	8.0	12.0
mobile_home	1	1600	2.5	4	1600	Thermosolar	1	1600	8.0	12.0
townhome	0	0	0	1	1000	Gas	0	0	0	0

Water Heater Statistics By State

Abstract Code:

1. Query **Location**, **Water Heater**, **Appliance**, **Household**, group by *State*, return:

- Average *BTU* (from **Appliance**, round to whole number)
- Average water heater *Capacity* (round to whole number)
- Average *Current Temperature* (round to 0.1)
- Count of All Water Heaters
- Count of Water Heaters that have a **Current Temperature** recorded
- Count of Water Heaters that do not have a **Current Temperature** recorded
- If there are no results for any of the counts, display 0.
- Sort results by State abbreviation, ascending, and display. Each State should be a link.

2. If a user clicks on a State link:

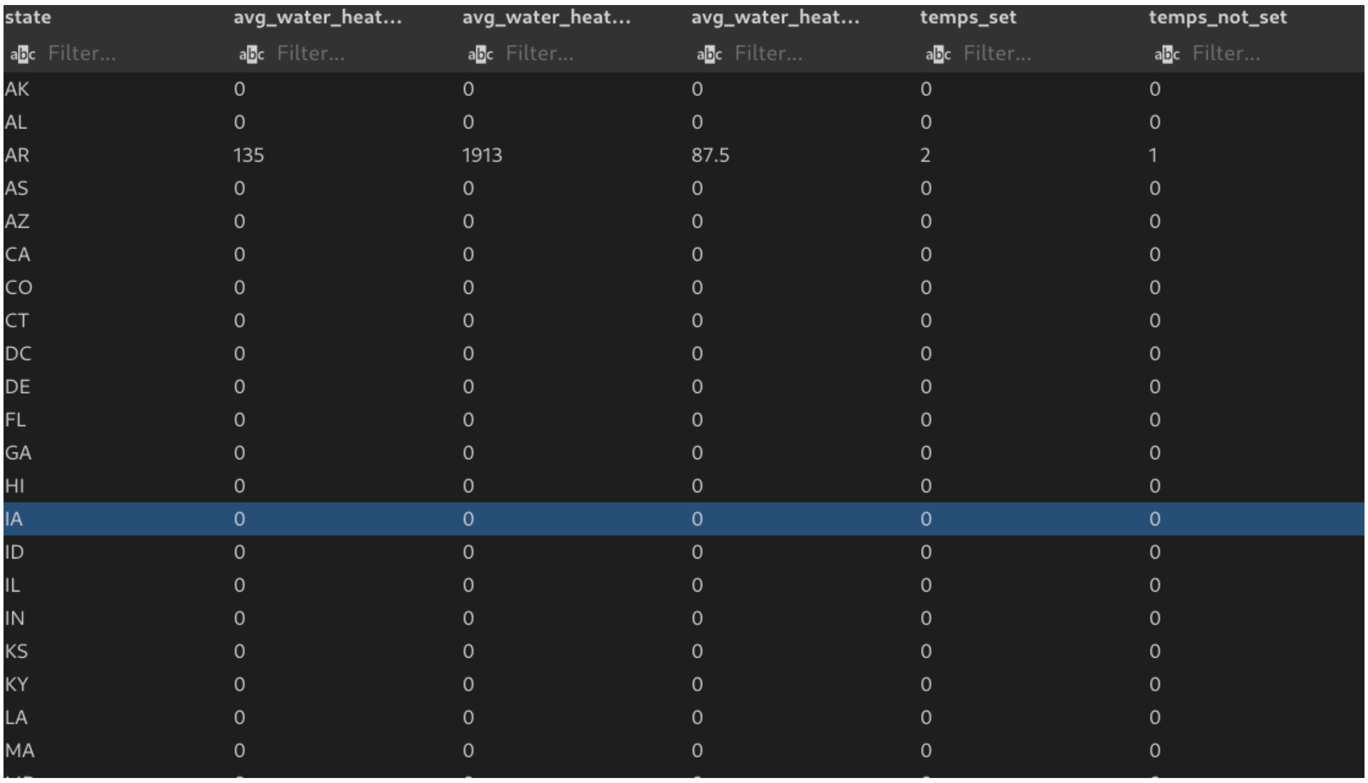
- Query **Water Heater**, **Location**, **Household**, **Appliance**, filter by the State clicked on and grouped by *Energy Source*, return:
  - Min, Avg, Max of *Capacity*, rounded to whole number
  - Min, Avg, Max of *Current Temperature*, rounded to 0.1.
  - Order by *Energy Source* ascending and display results.

---

1. Tested using seed data from playground/data.py

```
SELECT Location.state,
    COALESCE(ROUND(AVG(WaterHeater.capacity)), 0) AS
avg_water_heater_capacity,
    COALESCE(ROUND(AVG(Appliance.btu_rating) FILTER (WHERE
WaterHeater.email IS NOT NULL)), 0) AS avg_water_heater_btu,
    COALESCE(ROUND(AVG(WaterHeater.current_temperature), 1), 0) AS
avg_water_heater_temp,
    COALESCE(COUNT(WaterHeater.current_temperature), 0) AS temps_set,
    COALESCE(COUNT(
        CASE
            WHEN WaterHeater.email IS NOT NULL AND
WaterHeater.current_temperature IS NULL THEN 1
        END
    ), 0) AS temps_not_set
FROM Location
    LEFT JOIN Household USING (postal_code)
    LEFT JOIN Appliance USING (email)
    LEFT JOIN WaterHeater USING (email, appliance_number)
GROUP BY Location.state
ORDER BY Location.state ASC;
```

▶ Output



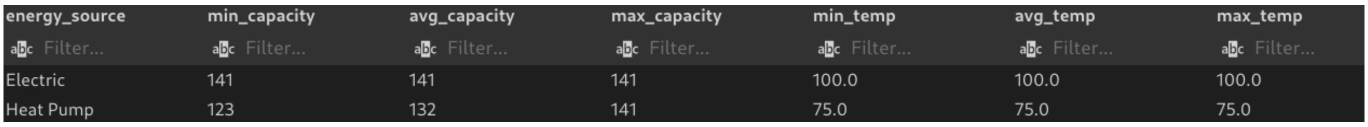
state	avg_water_heat...	avg_water_heat...	avg_water_heat...	temps_set	temps_not_set
Filter...	Filter...	Filter...	Filter...	Filter...	Filter...
AK	0	0	0	0	0
AL	0	0	0	0	0
AR	135	1913	87.5	2	1
AS	0	0	0	0	0
AZ	0	0	0	0	0
CA	0	0	0	0	0
CO	0	0	0	0	0
CT	0	0	0	0	0
DC	0	0	0	0	0
DE	0	0	0	0	0
FL	0	0	0	0	0
GA	0	0	0	0	0
HI	0	0	0	0	0
IA	0	0	0	0	0
ID	0	0	0	0	0
IL	0	0	0	0	0
IN	0	0	0	0	0
KS	0	0	0	0	0
KY	0	0	0	0	0
LA	0	0	0	0	0
MA	0	0	0	0	0

2. Tested using seed data from playground/data.py

- User selected AR in this example

```
WITH AllEnergySources AS (
    SELECT *
    FROM (
            VALUES('Electric'),
                ('Gas'),
                ('Thermosolar'),
                ('Heat Pump')
        ) AS t(energy_source)
),
HeatersInState AS (
    SELECT capacity,
        current_temperature,
        energy_source
    FROM WaterHeater
        INNER JOIN Appliance USING (email, appliance_number)
        INNER JOIN Household USING (email)
        INNER JOIN Location USING (postal_code)
    WHERE state = 'AR'
)
SELECT AllEnergySources.energy_source,
    COALESCE(ROUND(MIN(W.capacity)), 0) AS min_capacity,
    COALESCE(ROUND(AVG(W.capacity)), 0) AS avg_capacity,
    COALESCE(ROUND(MAX(W.capacity)), 0) AS max_capacity,
    COALESCE(ROUND(MIN(W.current_temperature), 1), 0) AS min_temp,
    COALESCE(ROUND(AVG(W.current_temperature), 1), 0) AS avg_temp,
    COALESCE(ROUND(MAX(W.current_temperature), 1), 0) AS max_temp
FROM AllEnergySources
    LEFT JOIN HeatersInState W USING (energy_source)
    GROUP BY AllEnergySources.energy_source
    ORDER BY AllEnergySources.energy_source ASC;
```

▶ Output



energy_source	min_capacity	avg_capacity	max_capacity	min_temp	avg_temp	max_temp
Electric	141	141	141	100.0	100.0	100.0
Heat Pump	123	132	141	75.0	75.0	75.0

Off-the-grid Household Dashboard

Abstract Code:

1. Query **Location** and **Off-Grid Household** grouped by State, return the State with the most Off-Grid Households and the count for that state.
2. Query the average Battery Storage capacity (Round to whole number) of all PowerGenerators in off-grid households.
3. Query the Percentages (round to 1 decimal) of off-grid households that have only-Wind, only-Solar, and mixed (Wind + Solar) PowerGenerator setups.
4. Query the average Water Heater Capacity (round to 1 decimal) grouped by whether the household is on-grid or not
5. Query the minimum, avg, and maximum BTU (round to whole number) of all appliances in off-grid households, grouped by appliance type

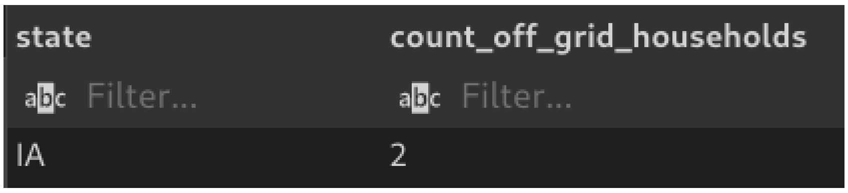
Each of these queries' results will be displayed as a separate table on a single page as part of the off-grid report.

---

1. Tested using seed data from playground/data.py

```sql
SELECT Location.state,
    COUNT(Household) AS count_off_grid_households
FROM Household
    INNER JOIN Location USING (postal_code)
WHERE NOT Household.on_grid
GROUP BY Location.state
ORDER BY count_off_grid_households DESC
LIMIT 1;
```

▶ Output



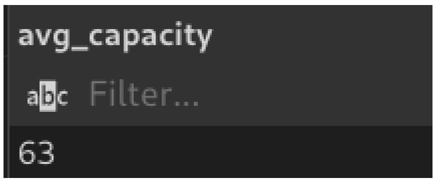
state	count_off_grid_households
abc Filter...	abc Filter...
IA	2

---

2. Tested using seed data from playground/data.py

```sql
SELECT ROUND(AVG(PowerGenerator.capacity)) as avg_capacity
FROM PowerGenerator
    INNER JOIN Household USING (email)
WHERE NOT Household.on_grid;
```

▶ Output



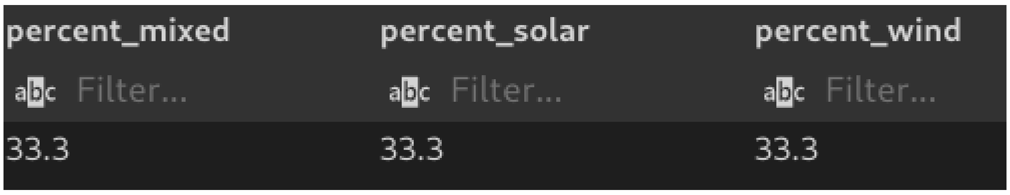
avg_capacity
abc Filter...
63

---

3. Tested using seed data from playground/data.py

```sql
WITH Temp AS (
    SELECT ARRAY_AGG(DISTINCT PowerGenerator.type) as pg_types
    FROM PowerGenerator INNER JOIN Household USING(email)
    WHERE NOT Household.on_grid
    GROUP BY PowerGenerator.email
),
HouseholdPowerGenTypes AS (
    SELECT (CASE WHEN ARRAY_LENGTH(pg_types, 1) = 2 THEN 'Mixed' ELSE
pg_types[1] END) as pg_type
    FROM Temp
)
SELECT
ROUND((COUNT(H) FILTER (WHERE pg_type = 'Mixed'))::decimal/NULLIF(COUNT(H),
0) * 100, 1) as percent_mixed,
ROUND((COUNT(H) FILTER (WHERE pg_type = 'Solar'))::decimal/NULLIF(COUNT(H),
0) * 100, 1) as percent_solar,
ROUND((COUNT(H) FILTER (WHERE pg_type = 'Wind'))::decimal/NULLIF(COUNT(H),
0) * 100, 1) as percent_wind
FROM HouseholdPowerGenTypes H;
```

▶ Output



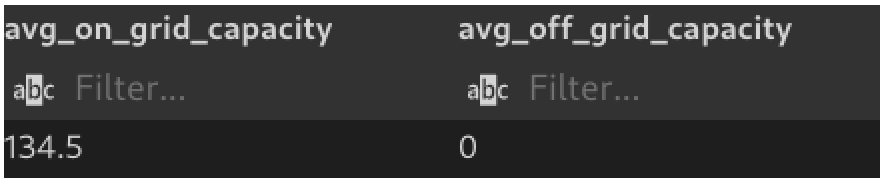
percent_mixed	percent_solar	percent_wind
abc Filter...	abc Filter...	abc Filter...
33.3	33.3	33.3

---

4. Tested using seed data from playground/data.py

```
WITH Averages AS (
    SELECT AVG(capacity) FILTER (WHERE NOT on_grid) as off_grid,
    AVG(capacity) FILTER (WHERE on_grid) as on_grid
    FROM Household
        INNER JOIN WaterHeater USING (email)
)
SELECT COALESCE(ROUND(on_grid, 1), 0) as avg_on_grid_capacity,
    COALESCE(ROUND(off_grid, 1), 0) as avg_off_grid_capacity
FROM Averages;
```

▶ Output



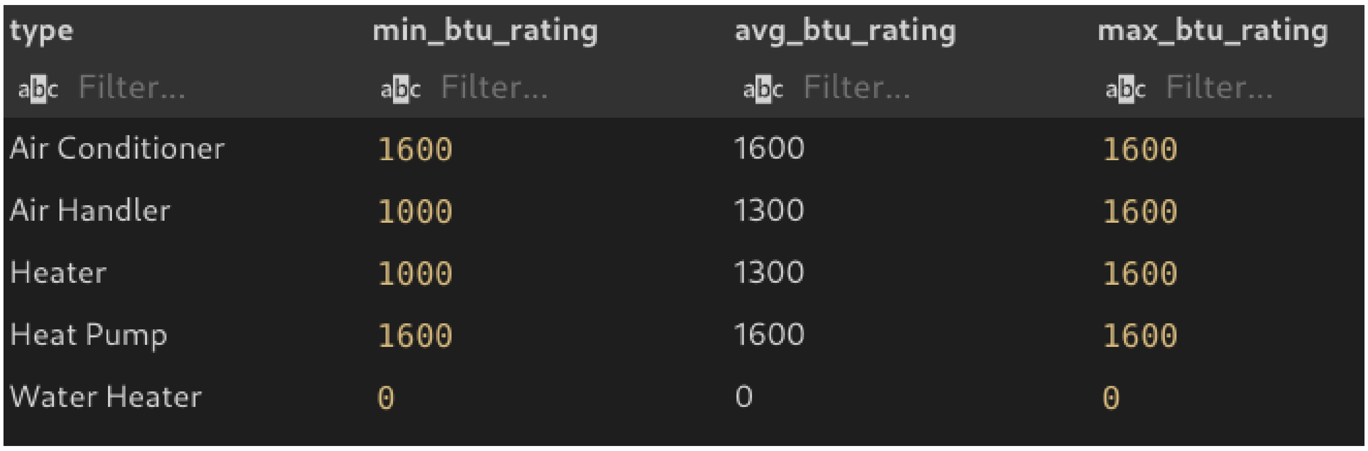
avg_on_grid_capacity	avg_off_grid_capacity
abc Filter...	abc Filter...
134.5	0

---

5. Tested using seed data from playground/data.py

```
With OffGrid AS (
    SELECT email,
        appliance_number,
        btu_rating AS btu,
        type
    FROM Household
        INNER JOIN Appliance USING (email)
    WHERE NOT on_grid
),
AllTypes AS (
    SELECT * FROM (
      VALUES ('Air Handler'),('Water Heater'),('Air Conditioner'),('Heat
Pump'),('Heater')
    ) AS Temp (t)
),
ApplianceBTUs AS (
    (SELECT 'Air Handler' AS t, btu FROM OffGrid WHERE type = 'Air
Handler')
    UNION
    (SELECT 'Water Heater' AS t, btu FROM OffGrid WHERE type = 'Water
Heater')
    UNION
    (SELECT 'Air Conditioner' AS t, btu FROM AirConditioner INNER JOIN
OffGrid USING (email, appliance_number))
    UNION
    (SELECT 'Heat Pump' AS t, btu FROM HeatPump INNER JOIN OffGrid USING
(email, appliance_number))
    UNION
    (SELECT 'Heater' AS t, btu FROM Heater INNER JOIN OffGrid USING (email,
appliance_number))
)
SELECT t as type,
    COALESCE(ROUND(MIN(btu)), 0) AS min_btu_rating,
    COALESCE(ROUND(AVG(btu)), 0) AS avg_btu_rating,
    COALESCE(ROUND(MAX(btu)), 0) AS max_btu_rating
FROM AllTypes
    LEFT JOIN ApplianceBTUs USING (t)
GROUP BY t;
```

▶ Output



type	min_btu_rating	avg_btu_rating	max_btu_rating
abc Filter...	abc Filter...	abc Filter...	abc Filter...
Air Conditioner	1600	1600	1600
Air Handler	1000	1300	1600
Heater	1000	1300	1600
Heat Pump	1600	1600	1600
Water Heater	0	0	0

Household averages by radius

Abstract Code:

- If user hits "Search" button:
  - Validate Postal Code exists in **Location**. If it doesn't, display a warning and stop.

```
SELECT Location.postal_code
FROM Location
WHERE Location.postal_code = $postal_code;
```

1. Query **Location** to return all postal codes within the user-selected distance of the user-input Postal Code.
   - For each row in Location:
     - Convert each Longitude and Latitude to Radians
     - Get Delta Lat and Delta Lon by subtracting the row's Lat/Lon from the Lat/Lon for the Postal Code the user entered.
     - Calculate "a" as:
       - $sin^2(\Delta lat/2) + cos(lat2) * sin^2(\Delta lon/2$
     - Calculate "c" as:
       - $2 * atan2(\sqrt{a}, \sqrt{1-a})$
     - Finally, the distance is $R * c$, where R is the radius of Earth.
     - Return the Zip Codes where distance is less than or equal to the user selected max distance.
2. Query **Household** (joined on `PowerGenerator` and `HouseholdUtilities`), filter for *Postal Code* from the previous query. Return:
   - Count of households
   - Count of households grouped by Type
   - Avg square footage (round to whole number)
   - Avg *Thermostat Setting Heating* (round to 0.1)
   - Avg *Thermostat Setting Cooling* (round to 0.1)
   - Comma-separated list of utilities used
   - Count of off-grid households
   - Count of households with PowerGenerators
   - Most common Power Generation method
   - Avg Monthly Power Generation
   - Count of households with Battery Storage

```
-- In this example, 71937 is the queried ZIP, and 100 miles is the radius.
WITH CenterLocation AS (
    SELECT latitude as lat,
        longitude AS lng
    FROM Location
    WHERE postal_code = '71937'
),
NearbyHouseholds AS (
    SELECT (household_type = 'house')::int AS is_house,
        (household_type = 'apartment')::int AS is_apartment,
        (household_type = 'townhome')::int AS is_townhome,
        (household_type = 'condominium')::int AS is_condo,
        (household_type = 'mobile_home')::int AS is_mobile,
        square_footage,
        thermostat_setting_heating,
        thermostat_setting_cooling,
        (NOT on_grid)::int AS off_grid,
        (
            CASE
                WHEN ARRAY_LENGTH(ARRAY_AGG(DISTINCT PowerGenerator.type),
1) = 2 THEN 'Mixed'
                WHEN ARRAY_LENGTH(ARRAY_AGG(DISTINCT PowerGenerator.type),
1) = 0 THEN NULL
                ELSE (ARRAY_AGG(DISTINCT PowerGenerator.type)) [1]
            END
        ) as pg_type,
        STRING_AGG(DISTINCT HouseholdUtilities.utilities, ',') AS
utilities,
        COUNT(DISTINCT PowerGenerator.email) > 0 AS has_power_gen,
        (
            COUNT(DISTINCT PowerGenerator.email) FILTER (
                WHERE PowerGenerator.capacity > 0
            ) > 0
        )::int as has_battery,
        SUM(PowerGenerator.avg_mon_kilo_hours) AS powergen_kwh
    FROM Location L
        CROSS JOIN CenterLocation C
        INNER JOIN Household USING (postal_code)
        LEFT JOIN HouseholdUtilities USING (email)
        LEFT JOIN PowerGenerator USING (email)
    WHERE 2 * 3958 * ASIN(
            SQRT(
                POW(SIN(RADIANS(L.latitude - C.lat) / 2), 2) +
COS(RADIANS(C.lat)) * COS(RADIANS(L.latitude)) * POW(
                    SIN(RADIANS(L.longitude - C.lng) / 2),
                    2
                )
            )
        ) <= 100
    GROUP BY Household.email
)
SELECT COUNT(*) AS household_count,
    SUM(is_house) AS house_count,
    SUM(is_apartment) AS apartment_count,
    SUM(is_townhome) AS townhome_count,
    SUM(is_mobile) AS mobile_count,
    SUM(is_condo) AS condo_count,
    ROUND(AVG(square_footage)) AS avg_sq_footage,
    ROUND(AVG(thermostat_setting_heating), 1) AS avg_therm_heating,
    ROUND(AVG(thermostat_setting_cooling), 1) AS avg_therm_cooling,
    STRING_AGG(DISTINCT utilities, ',') AS utilities,
    SUM(off_grid) AS off_grid_count,
    COUNT(pg_type) AS count_with_power_gen,
    MODE() WITHIN GROUP (
        ORDER BY pg_type
    ) as most_common_power_gen,
    ROUND(AVG(powergen_kwh)) AS avg_powergen_kwh
    SUM(has_battery) AS count_with_battery
FROM NearbyHouseholds;
```
