Hemkraft

Table of Contents

Notes:

- **For SQL statements:**
  - All the SQL statements in this report are written based on MySQL 8.0 syntax.
  - All the SQL statements in this report are tested using MySQL 8.0 against our schema and against some dummy data.
    All the dummy data in testing are manually inserted;
    The .sql files for inserting dummy data and testing could be provided upon request if reviewers have any concerns with the query statements.
  - Utilized WITH(Common Table Expressions) to handle the temporary result set that could be referred multiple times within current statement(query). This is permitted by instructor in Ed discussion @583.
  - Java variable names: 'lowerCamelCase'
- **For Abstract Code:**
  - We will use Java to do the implementation in Phase3, so the naming convention of user input field attributes is 'lowerCamelCase'.

Main Menu

Abstract Code

- Show "***Enter my household info***" and "***View reports / query data***" buttons.
- Upon:
  - Click ***Enter my household info*** button - jump to the **Input Personal Information** task.
  - Click ***View reports / query data*** button - jump to the **View Reports** task.

Input Personal Information

Abstract Code

- User clicked on ***Enter my household info*** button from **Main Menu**:
- Run the **Input Personal Information** task:
  - First display the **Email** form:
    - User enters *Email*('email').
    - Check if the format of *Email*('email') is valid
    - If data validation is successful for *email* input field, then:
      - When the ***Submit*** button is clicked, query the database to check if the 'email' exists'

```
SELECT email FROM Household WHERE email = 'email';
```

  - If the *Email*('email') already exists in database:
    - Go back to **Email** form, with an error message.
  - Else:

        - 'email' is cached at user's end; Display **Location** form.
    - Else *email* input field is invalid, go back to **Email** Form, with an error message.
- After **Location** form displayed:
    - User enters *PostalCode*('postalCode'): required to be 5 digits.
    - If data validation is successful for *PostalCode* input field, then:
        - When the ***Submit Postal Code*** button is clicked,
            - Find the Location instance using 'postalCode'.

```
SELECT postal_code
FROM `Location`
WHERE postal_code = 'postalCode';
```

                - If no matched Location object is found in database, go back to **Location** form, with an error message.
                - If matched Location object found, display this Location's *'postalCode'*, *'city'*, *'state'*.
                    - When the ***YES*** button is clicked, *'postalCode'* is cached at user's end, and go to **Phone Number** form.
                    - When the ***NO*** button is clicked, go back to **Location** form with an error message and let user re-enter *PostalCode*.
    - Else *PostalCode* input field is invalid, go back to **Location** form, with an error message.
- After **Phone Number** form displayed:
    - If ***NO*** button is clicked, skip and go to the next task: **Input Household Information** task.
    - If ***YES*** button is clicked,
        - User enters:
            - *AreaCode*('areaCode'): required to be 3 digits;
            - *SevenDigit* ('sevenDigit'): required to be 7 digits other than dash(dash is permitted);
            - *PhoneType*('phoneType'): for *PhoneType* field, users are only allowed to select one from the drop-down list which comprises four types {"home", "mobile", "work", "other"}.
        - If data validation is successful for the input fields, then:
            - When the ***Next*** button is clicked, check if the phone number user entered is duplicated with any phone number in database:

```
SELECT area_code, seven_digit
FROM PhoneNumber
WHERE area_code = 'areaCode' AND seven_digit =
'sevenDigit';
```

    - IF the retrieval query returns non-empty result (duplicate exists in database):
        - Go back to **Phone Number** Form, with an error message to indicate that the phone number already exists.
    - Else:
        - Cache 'areaCode', 'sevenDigit', 'phoneType' at user's end; PhoneNumber information will be inserted into database during next task after Household insertion, because PhoneNumber table has a foreign key (email) that need to reference a key in Household table.
        - Then run next **Input Household Information** task.
- Else any of the input fields are invalid, go back to **Phone Number** Form, with an error message.

Input Household Information

Abstract Code

- ***Next*** Button from previous page clicked and data collected from previous task passed validation:
    - Display **Household Info** form.
    - User enters:
        - *HomeType*('householdType'): user can select one of the *HomeType* from the dropdown list {"House", "Apartment", "Townhome", "Condominium", "Mobile Home"};
        - *SquareFootage*('squareFootage'): needs to be more than 0;
        - *NumOfOccupants*('numOfOccupant'): needs to be >= 0;
        - *NumOfBedroom*('numOfBedroom'): cannot be a negative number;
- When ***Next*** button clicked:
    - IF any properties are missing, display "Blanks Error" message to prompt user to complete all 4 fields.
    - IF data validation is successful for all input fields,
        - save Household information collected in this task along with PhoneNumber information collected in last task into the database.

```
INSERT INTO Household (email, household_type, square_footage,
num_of_occupant, num_of_bedroom, postal_code)
VALUES ('email', 'householdType', 'squareFootage',
'numOfOccupant', 'numOfBedroom', 'postalCode');
```

```
INSERT INTO PhoneNumber
VALUES ('areaCode', 'sevenDigit', 'phoneType', 'email');
```

        - go to the next **Input Bathroom Information** task.

- ELSE data validation for any input fields is invalid,
  - go back to **Household Info** form with an error message.

Input Bathroom Information

Abstract Code

- User clicks on ***Next*** button from previous page and data collected from previous task passed validation.
- Run **Add Bathroom**, then:
  - When ***Half*** button is selected:
    - Display a form for entering HalfBathroom information.
    - User enters information:
      - *NumberOfSink* ('numOfSink'), *NumOfCommode* ('numOfCommode'), *NumOfBidet* ('numOfBidet'): the sum of these 3 input values should be larger than 0;
      - *Name* ('name')
  - When ***Full*** button is selected:
    - Display a form for entering FullBathroom information.
    - User enters information:
      - *NumberOfSink* ('numOfSink'), *NumOfCommode* ('numOfCommode'), *NumOfBidet* ('numOfBidet'): no constraint for the sum.
      - *NumOfBathtub* ('numOfBathtub'), *NumOfShower* ('numOfShower'), NumOfTub ('numOfTub'): the sum of these 3 input values should be larger than 0;
      - *IsPrimary* ('isPrimary') indicating whether the bathroom is a primary bathroom:
        - If exists a primary FullBathroom object in database, user will not be able to check the current one to be primary.
  - If data validation is successful for all input fields discussed above, then:
    - User clicked on ***Add*** button:
      - SQL insertion queries are merged into a single transaction:
      - First, query the total number of bathrooms associated with the current household using current 'email'. The query result plus 1 is the bathroom_no for the bathroom;
      - Second, insert in Bathroom table (parent table of HalfBathroom and FullBathroom);
      - Then, insert bathroom information
        - If ***Half*** button is selected:
          - Save data to database for HalfBathroom

```
START TRANSACTION;
```

```
SET @cur_bathroom_num :=
        (SELECT COUNT(bathroom_no)
        FROM Bathroom
        WHERE Bathroom.email = 'email') + 1;
INSERT INTO Bathroom
VALUES ('email', @cur_bathroom_num, 'numOfSink',
'numOfCommode', 'numOfBidet');
INSERT INTO HalfBathroom
VALUES ('email', @cur_bathroom_num, 'name');
COMMIT;
```

- Go to next **List/View Bathroom** subtask.

- Else if ***Full*** button is selected:
  - Save data to database for FullBathroom

```
START TRANSACTION;
SET @cur_bathroom_num :=
        (SELECT COUNT(bathroom_no)
        FROM Bathroom
        WHERE Bathroom.email = 'email') + 1;
INSERT INTO Bathroom
VALUES ('email', @cur_bathroom_num, 'numOfSink',
'numOfCommode', 'numOfBidet');
INSERT INTO FullBathroom
VALUES ('email', @cur_bathroom_num, 'numOfBathtub',
'numOfShower', 'numOfTub', 'isPrimary');
COMMIT;
```

- Go to next **List/View Bathroom** subtask.

- Else any input field is invalid, go back for user to enter Cooker information again, with an error message.

- Then, run **List/View Bathroom** subtask:
  - Find all the saved bathrooms under the current house, identified by 'email', and output Bathroom # (bathroom_no), bathroom type, and 'isPrimary' values

```
SELECT `Bathroom #`, `Type`, `Primary`

FROM
 (SELECT bathroom_no AS `Bathroom #`,
         'Full' AS `Type`,
         CASE WHEN is_primary IS NOT NULL THEN 'Yes' ELSE NULL END AS
`Primary`
  FROM FullBathroom
  WHERE email = 'email'

  UNION

  SELECT bathroom_no AS `Bathroom #`,
         'Half' AS `Type`,
         NULL AS `Primary`
  FROM HalfBathroom
  WHERE email = 'email') AS AllBathroom

ORDER BY `Bathroom #`;
```

- Display result (if no bathrooms are saved for this household, display empty table)
- Upon clicking ***Add Another Bathroom*** button, go back and run **Input Bathroom Information** task.
- Upon Clicking ***Next*** button, go to **Input Appliance Information** task.

Input Appliance Information

Abstract Code

- Run **Input Appliance Information** task, display **Appliance** form;
- User is prompted to choose *ApplianceType*('applianceType') for the Appliance; User can only select one of the *ApplianceType* in a drop down menu includes {"Refrigerator/freezer", "Cooker", "Washer", "Dryer", "TV"}.
- After user selects *ApplianceType*, run **View Manufacturer List** subtask:
  - Find all the Manufacturer objects; Display manufacturers' *ManufacturerName* in a drop-down list.

```
SELECT manufacturer_name
FROM Manufacturer;
```

  - User selects 'manufacturerName' information for current Appliance.
- User can optionally enter *ModelName* ('modelName') for Appliance; Set to NULL if no input for it.
- When selecting *ApplianceType,*
  - IF user chooses ***Cooker***, display a form for entering Cooker information.
    - User can click on ***Oven*** or ***Cooktop*** button to enter information, also can click both buttons to enter information for both.
      - If ***Oven*** button is clicked, user will be able enter information required by Oven:
        - *OvenType*('ovenType') is selected from a drop-down menu includes: {"Convection", "Conventional"};
        - HeatSource information is also entered by user. The *OvenHeatSource*('ovenHeatSource') can only be chosen from {"Gas", "Electric", "Microwave"}, but the user can choose multiple of the types.
      - If ***Cooktop*** button is clicked, user will be able enter information required by Cooktop:
        - HeatSource information can be entered by user. The *CooktopHeatSource*('cooktopHeatSource') can only be chosen from {"Gas", "Electric", "Radiant electric", "Induction"}. User can only choose one of them.
    - If data validation is successful for all input fields discussed above, then:
      - When the ***Add*** button is clicked, save information to database:

- A set of SQL insertion queries are executed based on what appliance user chooses to add. SQL insertion queries are merged into a single transaction. General steps:
  - First, query the total number of appliances that associated with current household using current 'email'. The query result plus 1 is the appliance_no for the appliance that user want to add at this point;
  - Second, insert in Appliance table (parent table of Cooker);
  - Then, insert in Cooker table (parent table of Oven and Cooktop);
  - At the end, insert the Oven and/or Cooktop information (insert in child tables)
- IF ***Oven*** button is clicked, save Oven information to database (insert into both Oven and OvenHeatSource table); Because the user can choose multiple 'ovenHeatSource', for each 'ovenHeatSource' user checked, insert into OvenHeatSource table (Inserting multiple rows simultaneously into OvenHeatSource table is possible).

```
START TRANSACTION;
SELECT @cur_appliance_num :=
        (SELECT COUNT(appliance_no)
        FROM Appliance
        WHERE Appliance.email = 'email') + 1;
INSERT INTO Appliance VALUES ('email',
@cur_appliance_num, 'modelName',
'manufacturerName');
INSERT INTO Cooker VALUES ('email',
@cur_appliance_num);
INSERT INTO Oven VALUES ('email',
@cur_appliance_num, 'ovenType');
INSERT INTO OvenHeatSource VALUES ('email',
@cur_appliance_num, 'ovenHeatSource1'), ('email',
@cur_appliance_num, 'ovenHeatSource2');
COMMIT;
```

- ELSE IF ***Cooktop*** button is clicked, save Cooktop information to database.

```
START TRANSACTION;
SELECT @cur_appliance_num :=
        (SELECT COUNT(appliance_no)
        FROM Appliance
        WHERE Appliance.email = 'email') + 1;
INSERT INTO Appliance VALUES ('email',
@cur_appliance_num, 'modelName',
'manufacturerName');
INSERT INTO Cooker VALUES ('email',
```

```
@cur_appliance_num);
INSERT INTO Cooktop VALUES ('email',
@cur_appliance_num, 'cooktopHeatSource');
COMMIT;
```

- ELSE IF both ***Oven*** and ***Cooktop*** button are clicked, save both Oven and Cooktop information with one Cooker:

```
START TRANSACTION;
SELECT @cur_appliance_num :=
        (SELECT COUNT(appliance_no)
        FROM Appliance
        WHERE Appliance.email = 'email') + 1;
INSERT INTO Appliance VALUES ('email',
@cur_appliance_num, 'modelName',
'manufacturerName');
INSERT INTO Cooker VALUES ('email',
@cur_appliance_num);
INSERT INTO Oven VALUES ('email',
@cur_appliance_num, 'ovenType');
INSERT INTO OvenHeatSource VALUES ('email',
@cur_appliance_num, 'ovenHeatSource1'), ('email',
@cur_appliance_num, 'ovenHeatSource2');
INSERT INTO Cooktop VALUES ('email',
@cur_appliance_num, 'cooktopHeatSource');
COMMIT;
```

- Go to **View Appliance List** subtask.

- Else any input field is invalid, go back for user to enter Cooker information again, with an error message.

- ELSE IF user chooses ***Refrigerator/freezer***, display a form for entering Refrigerator information.
  - User enters information required by Refrigerator:
    - *RefrigeratorType* ('refrigeratorType'): user can choose one type from {"Bottom freezer refrigerator", "French door refrigerator", "Side-by-side refrigerator", "Top freezer refrigerator", "Chest freezer", "Upright freezer"}.
  - If data validation is successful for input fields, then:
    - When the ***Add*** button is clicked,
      - Save Refrigerator information. (Similar process as inserting Cooker information: Insert in both Appliance table and Refrigerator table as single transaction)

```
START TRANSACTION;
SELECT @cur_appliance_num :=
        (SELECT COUNT(appliance_no)
        FROM Appliance
        WHERE Appliance.email = 'email') + 1;
INSERT INTO Appliance VALUES ('email',
@cur_appliance_num, 'modelName', 'manufacturerName');
INSERT INTO Refrigerator VALUES ('email',
@cur_appliance_num, 'refrigeratorType');
COMMIT;
```

        - Go to **View Appliance List** subtask.
    - Else any input field is invalid, go back for user to enter Refrigerator information again, with an error message.
- ELSE IF user chooses ***Washer***, display a form for entering Washer information.
    - User enters information required by Refrigerator:
        - *LoadingType*('loadingType'): choose one from {"Top", "Front"}.
    - If data validation is successful for input fields, then:
        - When the ***Add*** button is clicked,
            - Save Washer information. (Similar process as inserting Cooker information: Insert in both Appliance table and Washer table as single transaction)

```
START TRANSACTION;
SELECT @cur_appliance_num :=
        (SELECT COUNT(appliance_no)
        FROM Appliance
        WHERE Appliance.email = 'email') + 1;
INSERT INTO Appliance VALUES ('email',
@cur_appliance_num, 'modelName', 'manufacturerName');
INSERT INTO Washer VALUES ('email',
@cur_appliance_num, 'loadingType');
COMMIT;
```

            - Go to **View Appliance List** subtask.
    - Else any input field is invalid, go back for user to enter Washer information again, with an error message.
- ELSE IF user chooses ***Dryer***, display a form for entering Dryer information.
    - User enters information required by Dryer:
        - HeatSource information for Dryer is also entered by user. The *DryerHeatSourceType*('dryerHeatSource') can be chosen from {"Gas", "Electric", "None"}, user can choose one from the list.
    - If data validation is successful for input fields, then:
        - When the ***Add*** button is clicked,
            - Save Dryer information. (Similar process as inserting Cooker information: Insert in both Appliance table and Dryer table as single transaction)

```
START TRANSACTION;
SELECT @cur_appliance_num :=
        (SELECT COUNT(appliance_no)
        FROM Appliance
        WHERE Appliance.email = 'email') + 1;
INSERT INTO Appliance VALUES ('email',
@cur_appliance_num, 'modelName', 'manufacturerName');
INSERT INTO Dryer VALUES ('email',
@cur_appliance_num, 'dryerHeatSource');
COMMIT;
```

            - Go to **View Appliance List** subtask.

        - Else any input field is invalid, go back for user to enter Dryer information again, with an error message.
    - ELSE IF user chooses ***TV***, display a form for entering TV information.
        - User enters information required by TV:
            - *DisplayType*('displayType'): user can select one from {"Tube", "DLP", "Plasma", "LCD", "LED"};
            - *DisplaySize*('displaySize'): user can only enter fractional inches to the tenth of an inch;
            - *MaxResolution*('maxResolution'): user can select one from {"480i", "576i", "720p", "1080i", "1080p", "1440p", "2160p (4K)", "4320p (8K)"}.
        - If data validation is successful for input fields, then:
            - When the ***Add*** button is clicked,
                - Save TV information. (Similar process as inserting Cooker information: Insert in both Appliance table and TV table as single transaction)

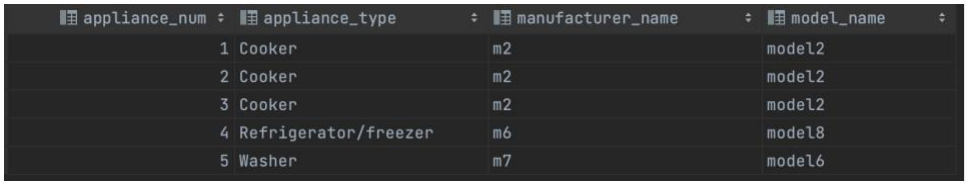
```
START TRANSACTION;
SELECT @cur_appliance_num :=
        (SELECT COUNT(appliance_no)
        FROM Appliance
        WHERE Appliance.email = 'email') + 1;
INSERT INTO Appliance VALUES ('email',
@cur_appliance_num, 'modelName', 'manufacturerName');
INSERT INTO TV VALUES ('email', @cur_appliance_num,
'displayType', 'displaySize', 'maxResolution');
COMMIT;
```

                - Go to **View Appliance List** subtask.
        - Else any input field is invalid, go back for user to enter TV information again, with an error message.
- Then, run **View Appliance List** subtask:
    - Find all the saved appliances; Retrieve each appliance's 'applianceType', 'manufacturerName', 'modelName' from database based on the 'email' oof the current household. And the appliance's number is appliance_num column in the output, which is assigned based on the insertion order of an appliance.

```
SELECT A.appliance_no AS appliance_num,
       appliance_type,
       manufacturer_name,
       model_name
FROM
    Appliance AS A,
    (
    SELECT email, appliance_no, 'Cooker' AS appliance_type from Cooker
    UNION
    SELECT email, appliance_no, 'Refrigerator/freezer' AS appliance_type
FROM Refrigerator
    UNION
    SELECT email, appliance_no, 'TV' AS appliance_type FROM TV
```

```
    UNION
    SELECT email, appliance_no, 'Washer' AS appliance_type FROM Washer
    UNION
    SELECT email, appliance_no, 'Dryer' AS appliance_type FROM Dryer
    ) AS D
WHERE
    A.email = 'email' AND A.email = D.email AND A.appliance_no =
D.appliance_no
ORDER BY A.appliance_no;
```

- Display the result table to user: (example of output with dummy data in MySQL 8.0)



appliance_num	appliance_type	manufacturer_name	model_name
1	Cooker	m2	model2
2	Cooker	m2	model2
3	Cooker	m2	model2
4	Refrigerator/freezer	m6	model8
5	Washer	m7	model6

- If no appliance data stored in database, display an empty table to user with an indication message.
- Upon clicking ***Add Another Appliance*** button, go back and run **Input Appliance Information** task.
- Upon Clicking ***Next*** button, go to **Wrap Up** task.

View Reports

Abstract Code

- Show "***View Top 25 Popular Manufactures***", "***View Bathroom Statistics***", "***View Laundry Center Report***", "***View Extra Fridge/freezer Report***", "***View Average TV Display Size by State***", "***Search Manufacture / Model***", and "***View Household Averages by Radius***" buttons.
- Upon:
  - Click ***View Top 25 Popular Manufactures*** button, jump to the **View Top 25 Popular Manufactures** task.
  - Click ***View Bathroom Statistics*** button, jump to the **View Bathroom Statistics** task.
  - Click ***View Laundry Center Report*** button, jump to the **View Laundry Center Report** task.
  - Click ***View Extra Fridge/freezer Report*** button, jump to the **View Extra Fridge/freezer Report** task.
  - Click ***View Average TV Display Size by State*** button, jump to the **View Average TV Display Size by State** task.
  - Click ***Search Manufacture / Model*** button, jump to the **Search Manufacture / Model** task.
  - Click ***View Household Averages by Radius*** button, jump to the **View Household Averages by Radius** task.

View Top 25 Popular Manufacturers

Abstract Code

- User clicked on the ***View Top 25 Popular Manufacturers*** button from the **View Reports.**
- Run **View Top 25 Popular Manufacturers** subtask:
  - Find information about Manufacturer and Appliance
    - Count: for each Manufacturer object, how many Appliance objects associated with it – how many Appliance objects are PRODUCED BY this manufacturer.
    - Sort the *ManufacturerName* values by the count as descending order, with limit of top 25 manufactures.

```
SELECT manufacturer_name
FROM
    (SELECT manufacturer_name,
            COUNT(appliance_no) as appliance_count
     FROM Appliance
     GROUP BY manufacturer_name
     ORDER BY appliance_count DESC
     LIMIT 25) Top_Manuf;
```

  - Display **Top 25 Popular Manufacturers** and each manufacturer in the list will be associated with a link for users to view detailed info about a specific manufacturer;
- Run **View Top 25 Popular Manufacturers Each Appliance Raw Count** subtask:
  - When the link of a specific manufacturer 'interestedManufacturer' (the manufacturer that user is interested in) in **Top 25 Popular Manufacturers** is clicked,
    - Query information of Appliance coupled with 'interestedManufacturer': appliance type, and the frequency of each appliance type for the selected Manufacturer.

```
SELECT Appl_Type.appliance_type,
       COUNT(Appliance.appliance_no) as appliance_count
FROM
    (SELECT email, appliance_no, appliance_type
     FROM(
         SELECT email, appliance_no, 'Cooker' as appliance_type
         FROM Cooker
         UNION
         SELECT email, appliance_no, 'Dryer' as appliance_type
         FROM Dryer
         UNION
         SELECT email, appliance_no, 'Refrigerator' as
appliance_type
```

```
          FROM Refrigerator
          UNION
          SELECT email, appliance_no, 'Washer' as appliance_type
          FROM Washer
          UNION
          SELECT email, appliance_no, 'TV' as appliance_type
          FROM TV) Appl_Type_V1
     )Appl_Type
INNER JOIN Appliance ON appliance.email  = Appl_Type.email
and Appliance. appliance_no = Appl_Type.appliance_no
GROUP BY Appliance. manufacturer_name, Appl_Type.appliance_type
HAVING Appliance. manufacturer_name = 'interestedManufacturer'
ORDER BY appliance_count DESC;
```

  - Display 'interestedManufacturer' at the top of the user interface page, along with a table that includes the results from the query.
- If there is no data stored in the system corresponding to this report, an error message will be prompted to user to indicate this.

View Bathroom Statistics

Abstract Code

- User clicked on the ***View Bathroom Statistics*** button from the **View Reports.**
- Run ***View Bathroom Statistics*** task:
  - Find information about Bathroom, HalfBathroom, FullBathroom, Household, Location.
  - Run **Get Min, Avg, Max of the Count of X Per Household** task:
    - Count the number of X(Bathroom, HalfBathroom, FullBathroom) objects associated with each Household object respectively.
    - Count the number of X(commodes, sinks, bidets, bathtubs, showers, tub/showers) per Household.
    - Calculate the Avg count of all X per household as Double data type: using number of Household objects in system and count of number of X per Household.
    - Calculate the Min and Max count of all X per household as Integer data type: order the count of X per household.
    - Display Min, Avg, Max of the count of X per Household.

```
-- count Bathroom:

SELECT MIN(count),
       MAX(count),
       ROUND(AVG(count), 1)
FROM (SELECT email,
             COUNT(email) AS count
```

```
      FROM Bathroom
      GROUP BY email
      ORDER BY COUNT(email) DESC) AS BathroomCount;
```

```
-- count HalfBathroom:

WITH SumHouseholdNum AS (SELECT COUNT(email) AS householdCount
                         FROM Household)
SELECT min_count, max_count,
       ROUND(sum_count/SumHouseholdNum.householdCount, 1) AS avg_count
FROM (SELECT MIN(count) AS min_count,
             MAX(count) AS max_count,
             SUM(count) AS sum_count
      FROM (SELECT email, COUNT(email) AS count
            FROM HalfBathroom
            GROUP BY email
            ORDER BY COUNT(email) DESC) AS HalfBathroomCount) AS FC,
     SumHouseholdNum;
```

```
-- count FullBathroom:

WITH SumHouseholdNum AS (SELECT COUNT(email) AS householdCount
                         FROM Household)
SELECT min_count, max_count,
       ROUND(sum_count/SumHouseholdNum.householdCount, 1) AS avg_count
FROM (SELECT MIN(count) AS min_count,
             MAX(count) AS max_count,
             SUM(count) AS sum_count
      FROM (SELECT email, COUNT(email) AS count
            FROM FullBathroom
            GROUP BY email
            ORDER BY COUNT(email) DESC) AS FullBathroomCount) AS FC,
     SumHouseholdNum;
```

```
-- count Commodes:

WITH SumHouseholdNum AS (SELECT COUNT(email) AS householdCount
                         FROM Household)
SELECT min_count, max_count,
       ROUND(sum_count/SumHouseholdNum.householdCount, 1) AS avg_count
FROM (SELECT MIN(count) AS min_count,
             MAX(count) AS max_count,
             SUM(count) AS sum_count
      FROM (SELECT email, SUM(num_of_commode) AS count
            FROM Bathroom
```

```
            GROUP BY email
            ORDER BY count DESC) AS CommodesCount) AS FC,
     SumHouseholdNum;
```

```
-- count Sinks:

WITH SumHouseholdNum AS (SELECT COUNT(email) AS householdCount
                         FROM Household)
SELECT min_count, max_count,
       ROUND(sum_count/SumHouseholdNum.householdCount, 1) AS avg_count
FROM (SELECT MIN(count) AS min_count,
             MAX(count) AS max_count,
             SUM(count) AS sum_count
      FROM (SELECT email, SUM(num_of_sink) AS count
            FROM Bathroom
            GROUP BY email
            ORDER BY count DESC) AS SinksCount) AS FC,
     SumHouseholdNum;
```

```
-- count Bidets:

WITH SumHouseholdNum AS (SELECT COUNT(email) AS householdCount
                         FROM Household)
SELECT min_count, max_count,
       ROUND(sum_count/SumHouseholdNum.householdCount, 1) AS avg_count
FROM (SELECT MIN(count) AS min_count,
             MAX(count) AS max_count,
             SUM(count) AS sum_count
      FROM (SELECT email, SUM(num_of_bidet) AS count
            FROM Bathroom
            GROUP BY email
            ORDER BY count DESC) AS BidetsCount) AS FC,
     SumHouseholdNum;
```

```
-- count Bathtubs:

WITH SumHouseholdNum AS (SELECT COUNT(email) AS householdCount
                         FROM Household)
SELECT min_count, max_count,
       ROUND(sum_count/SumHouseholdNum.householdCount, 1) AS avg_count
FROM (SELECT MIN(count) AS min_count,
             MAX(count) AS max_count,
             SUM(count) AS sum_count
```

```
      FROM (SELECT email, SUM(num_of_bathtub) AS count
            FROM FullBathroom
            GROUP BY email
            ORDER BY count DESC) AS BathtubsCount) AS FC,
     SumHouseholdNum;
```

```
-- count Showers:

WITH SumHouseholdNum AS (SELECT COUNT(email) AS householdCount
                         FROM Household)
SELECT min_count, max_count,
       ROUND(sum_count/SumHouseholdNum.householdCount, 1) AS avg_count
FROM (SELECT MIN(count) AS min_count,
             MAX(count) AS max_count,
             SUM(count) AS sum_count
      FROM (SELECT email, SUM(num_of_shower) AS count
            FROM FullBathroom
            GROUP BY email
            ORDER BY count DESC) AS ShowersCount) AS FC,
     SumHouseholdNum;
```

```
-- count tubShowers:

WITH SumHouseholdNum AS (SELECT COUNT(email) AS householdCount
                         FROM Household)
SELECT min_count, max_count,
       ROUND(sum_count/SumHouseholdNum.householdCount, 1) AS avg_count
FROM (SELECT MIN(count) AS min_count,
             MAX(count) AS max_count,
             SUM(count) AS sum_count
      FROM (SELECT email, SUM(num_of_tub) AS count
            FROM FullBathroom
            GROUP BY email
            ORDER BY count DESC) AS TubShowersCount) AS FC,
     SumHouseholdNum;
```

- Run **View Max Count of Bidets State** task:
  - Count the number of bidets for each State based on information about Location, Household, Bathroom.
  - Get the state with max number of bidets, then display values of this *State* and its total number of bidets.

```
SELECT `state`,
```

```
       MAX(count)
FROM (SELECT `state`, SUM(num_of_bidet) AS count
      FROM ((SELECT Bathroom.email, num_of_bidet,
                    bathroom_no, Household.postal_code
             FROM Bathroom NATURAL JOIN Household) AS BH
             NATURAL JOIN `Location`)
      GROUP BY `state`
      ORDER BY count DESC) AS MaxBidetsState
GROUP BY `state`
LIMIT 1;
```

- Run **View Max Count of Bidets Postcode** task:
  - Based on information about Location, Household, Bathroom, Count the number of bidets for each *PostalCode* in Location.
  - Get the PostalCode with max number of bidets, then display values of this *PostalCode* and its total number of bidets.

```
SELECT postal_code, MAX(count)
FROM (SELECT postal_code, SUM(num_of_bidet) AS count
      FROM ((SELECT Bathroom.email, num_of_bidet,
                    bathroom_no, Household.postal_code
             FROM Bathroom NATURAL JOIN Household) AS BH
             NATURAL JOIN `Location`)
      GROUP BY postal_code
      ORDER BY count DESC) AS MaxBidetsCode
GROUP BY postal_code
LIMIT 1;
```

- Run **View Count of Household with Only Primary Bathroom** task:
  - Based on information about Location, Household, Bathroom, FullBathroom, count the number of Household objects which has only one bathroom, and this bathroom is primary.
  - Display value of count.

```
SELECT COUNT(OnlyOne.email) AS count
FROM (SELECT email
      FROM Bathroom
      GROUP BY email
      HAVING COUNT(email) = 1) AS OnlyOne

     INNER JOIN

     (SELECT email, bathroom_no, is_primary
      FROM FullBathroom
      WHERE is_primary = TRUE) AS IsPrimary

     ON OnlyOne.email = IsPrimary.email;
```

- If there is no data stored in the system corresponding to this report, an error message will be prompted to user to indicate this.

View Laundry Center Report

Abstract Code

- User clicked on the ***View Laundry Center Report*** button from the **View Reports**
- Run ***View Laundry Center Report*** task:
    - Find the information for Washer, Household, Location, Dryer, HeatSource.
    - Run **View Most Popular Washer Type By State and View Most Popular Dryer Heat Source By State** subtasks together in a single SQL query:
        - Based on the information for Washer, Household, Location.
            - Count each *WasherType* for each *State*, and get the most common *WasherType* for each *State.*
            - Get Most Popular Washer Type By State that is ordered by state ascending.
        - Based on the information for Dryer, Household, Location, HeatSource.
            - Count number of each HeatSource associated with Dryer for each *State*, and get the most common *HeatSourceType* for each *State.*
            - Get Most Popular Dryer HeatShource Type By State that is ordered by state ascending.
        - Merge results from last two steps (full outter join), and the output table is ordered by state ascending. (NULL values will be replaced by empty string to display)

```
WITH cte_washer AS (SELECT state, loading_type,
                           COUNT(loading_type) AS count
                    FROM (SELECT state, loading_type
                          FROM Household
                          NATURAL JOIN Location
                          NATURAL JOIN Washer) AS t
                    GROUP BY state, loading_type
                    ORDER BY state ASC),

     cte_washer_final AS (SELECT state, loading_type, count
                          FROM cte_washer
                          WHERE (state, count) IN
                             (SELECT state, MAX(count) AS count
                              FROM cte_washer
                              GROUP BY state)),

     cte_dryer AS (SELECT state, dryer_heat_source,
                          COUNT(dryer_heat_source) AS count
                   FROM (SELECT state, dryer_heat_source
                         FROM Household
                         NATURAL JOIN Location
                         NATURAL JOIN Dryer) AS t3
                   GROUP BY state, dryer_heat_source
                   ORDER BY state ASC),

     cte_dryer_final AS (SELECT state, dryer_heat_source, count
                         FROM cte_dryer
                         WHERE (state, count) IN
                           (SELECT state, MAX(count) AS count
                            FROM cte_dryer
```

```
                                GROUP BY state))

SELECT state,
       MAX(loading_type) AS common_washer_type,
       MAX(dryer_heat_source) AS common_dryer_heat_source
FROM
      (SELECT w.state, w.loading_type, d.dryer_heat_source
      FROM cte_washer_final w
      LEFT JOIN cte_dryer_final d ON w.state = d.state
      UNION
      SELECT w.state, w.loading_type, d.dryer_heat_source
      FROM cte_washer_final w
      RIGHT JOIN cte_dryer_final d ON w.state = d.state) AS t5
GROUP BY state;
```

- Run **View Household with Washer No Dryer By State** task:
  - Based on the information for Dryer, Washer, Household, Location.
    - For each *State, c*ount number of Household with only Washer but no Dryer.
    - The output table is ordered by household count descending.

```
WITH cte_washer AS (SELECT DISTINCT email, state
                    FROM Household
                    NATURAL JOIN Location
                    NATURAL JOIN Washer),

     cte_dryer AS (SELECT DISTINCT email, state
                   FROM Household
                   NATURAL JOIN Location
                   NATURAL JOIN Dryer)

SELECT state,
       COUNT(state) AS household_count
FROM cte_washer
WHERE cte_washer.email NOT IN (SELECT email FROM cte_dryer)
GROUP BY state
ORDER BY household_count DESC;
```

- Two tables from the two subtasks in this **View Laundry Center Report** task are displayed together on one GUI page.
- If there is no data stored in the system corresponding to this report, an error message will be prompted to user to indicate this.

View Extra Fridge/freezer Report

Abstract Code

- User clicked on the ***View Laundry Center Report*** button from the **View Reports**
- Run **View Extra Fridge/freezer Report task:**
  - Count and display the number of Households that have count of Refrigerator greater than 1

```
SELECT COUNT(MultiHouse.`house`) AS household_count
FROM (SELECT Household.email AS `house`,
             Location.state,
             COUNT(Refrigerator.email) AS refrigerator_count
      FROM Refrigerator
      JOIN Household ON Refrigerator.email=Household.email
      JOIN Location ON Location.postal_code=Household.postal_code
      GROUP BY Household.email
      HAVING refrigerator_count > 1) AS MultiHouse;
```

- Query the top ten states that possess most Households with multiple fridge/freezers, along with count of Households with multiple fridge/freezers in this state and 3 percentage statistics: (The SQL statement (attach below) is one huge query that completes all the steps introduced here together)
  - Step 1: Based on list of Households with multiple fridge/freezers, count number of Households by state. Order the list of states by descending order of the count and display.
  - Step 2: For each state in the top ten list:
    - Find all Households in the state
    - Based on list of Households, count number of households with:
      - multiple fridge/freezers and chest freezers
      - multiple fridge/freezers and upright freezers
      - multiple fridge/freezers and freezers other than chest/upright freezers
    - Calculate and display the 3 percentages by dividing each count by total number of Households in the state

```
WITH MultiHouse AS (SELECT Household.email AS `house`,
                           Location.state AS `state`,
                           COUNT(Refrigerator.email) AS RefrigeratorCount
                    FROM Refrigerator
                    JOIN Household ON Refrigerator.email=Household.email
                    JOIN Location ON Location.postal_code=Household.postal_code
                    GROUP BY Household.email
                    HAVING RefrigeratorCount > 1),
     ChestHouse AS (SELECT Household.email AS `chest_house`,
                           Location.state AS `chest_state`,
                           COUNT(Refrigerator.email) AS ChestCount
                    FROM Refrigerator
                    JOIN Household ON Refrigerator.email = Household.email
                    JOIN Location ON Location.postal_code = Household.postal_code
                    WHERE Refrigerator.refrigerator_type = 'chest'
                    GROUP BY Household.email),
     UprightHouse AS (SELECT Household.email AS `upright_house`,
                      Location.state AS `upright_state`,
                      COUNT(Refrigerator.email) AS UprightCount
                      FROM Refrigerator
                      JOIN Household ON Refrigerator.email = Household.email
                      JOIN Location ON Location.postal_code = Household.postal_code
                      WHERE Refrigerator.refrigerator_type = 'upright'
                      GROUP BY Household.email),
     MultiWithChest AS (SELECT MultiHouse.`house`, MultiHouse.`state`
```

```
                        FROM MultiHouse
                        JOIN ChestHouse
                            ON MultiHouse.`house` = ChestHouse.`chest_house`),
    MultiWithUpright AS (SELECT MultiHouse.`house`, MultiHouse.`state`
                         FROM MultiHouse
                         JOIN UprightHouse
                           ON MultiHouse.`house` = UprightHouse.`upright_house`),
    MultiChestAndMultiUpright AS (
               SELECT MultiWithChest.`house` FROM MultiWithChest
               UNION
               SELECT MultiWithUpright.`house` FROM MultiWithUpright),
    MultiWithOther AS (
               SELECT MultiHouse.`house`, MultiHouse.`state`
               FROM MultiHouse
               LEFT JOIN MultiChestAndMultiUpright
                    ON MultiHouse.`house` = MultiChestAndMultiUpright.`house`
               WHERE MultiChestAndMultiUpright.`house` IS NULL)

SELECT MULTI.`State`,
       MULTI.`StateMultiCount`,
       ROUND(IF(ISNULL(CHEST.`StateChestCount`), 0,
CHEST.`StateChestCount`)/STATEHOUSE.`StateHouseCount`*100, 0) AS `State Chest
Percent`,
       ROUND(IF(ISNULL(UPRIGHT.`StateUprightCount`), 0,
UPRIGHT.`StateUprightCount`)/STATEHOUSE.`StateHouseCount`*100, 0) AS `State Upright
Percent`,
       ROUND(IF(ISNULL(OTHER.`StateOtherCount`), 0,
OTHER.`StateOtherCount`)/STATEHOUSE.`StateHouseCount`*100, 0) AS `State Other
Percent`

FROM
    (SELECT Location.state AS `State`,
            COUNT(MultiHouse.`house`) AS `StateMultiCount`
     FROM Location
     JOIN MultiHouse ON Location.state = MultiHouse.`state`
     GROUP BY Location.state) AS MULTI

     LEFT JOIN

    (SELECT Location.state AS `State`,
            COUNT(MultiWithChest.`house`) AS `StateChestCount`
     FROM Location
     JOIN MultiWithChest ON Location.state = MultiWithChest.`state`
     GROUP BY Location.state) AS CHEST

     ON MULTI.`State` = CHEST.`State`

     LEFT JOIN

    (SELECT Location.state AS `State`,
            COUNT(MultiWithUpright.`house`) AS `StateUprightCount`
     FROM Location
     JOIN MultiWithUpright ON Location.state = MultiWithUpright.`state`
     GROUP BY Location.state) AS UPRIGHT
```

```
     ON MULTI.`State` = UPRIGHT.`State`

     LEFT JOIN

    (SELECT
     Location.state AS `State`,
     COUNT(MultiWithOther.`house`) AS `StateOtherCount`
     FROM Location
     JOIN MultiWithOther ON Location.state = MultiWithOther.`state`
     GROUP BY Location.state) AS OTHER

     ON MULTI.`State` = OTHER.`State`

     LEFT JOIN

   (SELECT Location.state AS `State`,
     COUNT(Household.email) AS `StateHouseCount`
    FROM Household JOIN Location ON Household.postal_code = Location.postal_code
    GROUP by Location.state) AS STATEHOUSE

     ON MULTI.`State` = STATEHOUSE.`State`

ORDER BY MULTI.`StateMultiCount` DESC
LIMIT 10;
```

- If there is no data stored in the system corresponding to this report, an error message will be prompted to user to indicate this.

View Average TV Display Size by State

Abstract Code

- User clicked on the ***View Household Averages by Radius*** button from the **View Reports**
- Run **View Average TV Display Size by State** (all steps below are completed in one single query)
    - Find information for TV, Household, Location.
    - Count the total number of TVs in the state.
    - Calculate sum of display size of all TVs in the state.
    - Calculate the average TV display size (data type is Double) by state: dividing sum of display size by number of TVs.
    - Display Average TV Display Size by State ordered by state ascending

```
SELECT Location.state,
       ROUND (avg(TV.display_size),1) as avg_display_size
FROM Appliance INNER JOIN TV
       ON Appliance.email=TV.email AND Appliance.appliance_no=TV.appliance_no
     JOIN Household ON Appliance.email=Household.email
     INNER JOIN Location on Location.postal_code=Household.postal_code
GROUP BY Location.state
ORDER BY Location.state ASC;
```

- For each state, if user clicks on ***View Drilldown Report*** button associated with the state(the state that user is interested in 'interestedState'),
    - Run **View Drilldown Report for the State** subtask:
        - For this specific state, for each *DisplayType* and *MaxResolution* combination, (all steps below are completed in one single query)
            - Count the number of TVs with this combination of *DisplayType* and *MaxResolution.*
            - Calculate average screen size by dividing sum of all TV screen size by number of TVs (data type is Double).
            - Output query result: table of *DisplayType*, *MaxResolution*, and the corresponding average screen size, ordered by average screen size in descending order.

```
SELECT Location.state, TV.display_type, TV.max_resolution,
       ROUND(avg(TV.display_size),1) as avg_display_size
FROM Appliance
INNER JOIN TV ON Appliance.email=TV.email and
Appliance.appliance_no=TV.appliance_no
INNER JOIN Household ON Appliance.email=Household.email
INNER JOIN Location on Location.postal_code=Household.postal_code
GROUP BY Location.state,TV.display_type,TV.max_resolution
HAVING Location.state = 'interestedState'
ORDER BY avg_display_size DESC;
```

        - Display result.
- If there is no data stored in the system corresponding to this report, an error message will be prompted to user to indicate this.

Search Manufacture / Model

Abstract Code

- User clicked on ***Search Manufacture / Model*** button, run **Search Manufacture / Model** task.
- User enters a keyword 'inputKeyword' (string).
- If data validation is successful for the input field, then:
    - When the ***Search*** button is clicked,
        - Get all the matched *ManufacturerName* and *ModelName* in database. (all steps below are completed in one single query)
            - Get all the Manufacturers with part of *ManufacturerName* matches the input;

- Get all the Appliance associated with each Manufacturer to further get all the values of *ModelName* in Appliance that related to this Manufacturer.
- Get all the Appliance, then get all the values of *ModelName* which part of it matches the input.

```
SELECT distinct manufacturer_name, model_name
FROM Appliance
WHERE manufacturer_name like concat('%','inputKeyword','%')
      OR model_name like concat('%','inputKeyword','%')
ORDER BY manufacturer_name ASC, model_name ASC;
```

  - If no match is found,
    - Let user enter it again, with an error message.
  - Else matches are found,
    - Display the query result: a distinct list ordered by *ManufacturerName* and *ModelName* ascending. (matched *ManufacturerName* and *ModelName* will be highlighted)
- Else the input field is invalid, let user enter it again, with an error message.
- If there is no data stored in the system corresponding to this report, an error message will be prompted to user to indicate this.

View Household Averages by Radius

Abstract Code

- Click on the ***View Household Averages by Radius Report*** button from the **View Reports**
- Run **Input Postal Code and Search Radius** subtask
  - Populate *PostalCode* and *SearchRadius* input Options, User enter postal code (5 digits) and select one of the radius from the dropdown list {0,5,10,25,50,100,250}.
  - If data validation for format is successful for the input fields *PostalCode* 'inputPostalCode'(string) and *SearchRadius* 'searchRadius'(integer), then:
    - When ***Submit*** button is clicked,
      - IF 'inputPostalCode' exists in Location table (this query returns non-empty result), run **View Household Statistics Results** subtask.

```
SELECT postal_code
FROM Location
WHERE postal_code = 'inputPostalCode';
```

      - ELSE IF 'inputPostalCode' does not exist in Location table (the query returns empty result), let user enter them again, with an error message to indicate the input postal code does not exist in database.

  - If any of the inputs' format is invalid, let user enter them again, with an error message.
- Run **View Household Statistics Results** subtask:
  - The SQL statement (attach below) is one huge query that completes all the steps introduced here together. And the output is just one row with 8 columns/attributes required by this task. All calculations and output data casting/formating are done within this query.
    - Step1: get latitude and longitude of the user 'inputPostalCode' (center point) in Location table;
    - Step2: based on the center point's latitude, longitude, and input 'searchRadius', select all the postal_code in Location within the search range (calculate the distance between a postal_code and the center point; if the distance <= searchRadius, this postal_code is valid)
    - Step3: get all Household, Bathroom, Appliance within the search range
    - Step4: calculate and output the required statistics; The casting and formatting of output data are also done within database.
      - input_center_postal_code,
      - input_search_radius,
      - avg_bathroom_count_per_household_within_range,
      - avg_bedroom_count_per_household_within_range,
      - avg_occupant_count_per_household_within_range,
      - ratio_of_commode_to_occupant_within_range,
      - avg_appliance_count_per_household_within_range,
      - most_common_heat_source_within_range.
        - Notes to most common heat source: If all the households within the range have no appliances that contain heat sources (check query result using `IFNULL in MySQL`[1]), the output for this var is set to be empty

Single SQL query for **View Household Averages by Radius** task:

```
# One huge query for all the information needed in this task; Might want to utilize view
later in Phase3 to decrease redundant in statement;

# Utilized WITH(Common Table Expressions) to handle the temporary result set (alias is DP
in this query) that could be referred multiple times within current query. This is
permitted by instructor in Ed discussion @583.
WITH DP AS (SELECT L.postal_code
            FROM `Location` AS L,
                (SELECT latitude  AS centerLat,
                        longitude AS centerLon
                 FROM `Location`
                 WHERE postal_code = 'inputPostalCode') AS CP
            WHERE (ACOS(SIN(RADIANS(CP.centerLat)) *
```

[1] IFNULL(expr1,expr2): If expr1 is not NULL, IFNULL() returns expr1; otherwise it returns expr2.

```
                    SIN(RADIANS(L.latitude)) +
                    COS(RADIANS(CP.centerLat)) *
                    COS(RADIANS(L.latitude)) *
                    COS(RADIANS(L.longitude) -
                    RADIANS(CP.centerLon))) * 3958.75) <= 'searchRadius')

SELECT
      'inputPostalCode' AS input_center_postal_code,
      'searchRadius' AS input_search_radius,
      CAST((FB.count_of_bathroom_within_range/FH.count_of_household_within_range) AS
DECIMAL(11, 1)) AS avg_bathroom_count_per_household_within_range,
      CAST(FH.avg_bedroom_num_per_household AS DECIMAL(11, 1)) AS
avg_bedroom_count_per_household_within_range,
      ROUND(FH.avg_occupant_num_per_household) AS
avg_occupant_count_per_household_within_range,
      CONCAT('1:', CAST((CAST((FH.sum_occupant_num / FB.sum_commode_num) AS
DECIMAL(12, 2))) AS CHAR)) AS ratio_of_commode_to_occupant_within_range,
      CAST((FA.count_of_appliance_within_range/FH.count_of_household_within_range) AS
DECIMAL(11, 1)) AS avg_appliance_count_per_household_within_range,
      FS.most_common_heat_source AS most_common_heat_source_within_range

FROM
    # FH as final household res
    (SELECT AVG(H.num_of_occupant) AS avg_occupant_num_per_household,
            SUM(num_of_occupant)   AS sum_occupant_num,
            AVG(num_of_bedroom)    AS avg_bedroom_num_per_household,
            COUNT(DISTINCT email)  AS count_of_household_within_range
     FROM Household AS H,
          DP
     WHERE H.postal_code = DP.postal_code) AS FH,

    # FB as final bathroom res
    (SELECT SUM(B.num_of_commode) AS sum_commode_num,
            COUNT(B.bathroom_no)  AS count_of_bathroom_within_range
     FROM Household AS H,
          Bathroom AS B,
          DP
     WHERE H.postal_code = DP.postal_code
       AND H.email = B.email) AS FB,

    # FA as final appliance res
    (SELECT COUNT(A.appliance_no) AS count_of_appliance_within_range
     FROM Household AS H,
          Appliance AS A,
          DP
     WHERE H.postal_code = DP.postal_code
       AND H.email = A.email) AS FA,

    # FS as final heat source res
    # If no heat sources exists at all for all households within range, return empty
    (SELECT IFNULL(
    (SELECT heat_source AS most_common_heat_source
     FROM (SELECT heat_source,
                  COUNT(heat_source) AS count_of_each_heatsource
           FROM (SELECT A.email,
```

```
                    A.appliance_no
             FROM Household AS H,
                  Appliance AS A,
                  DP
             WHERE H.postal_code = DP.postal_code
               AND H.email = A.email) AS DA,

            ((SELECT email, appliance_no,
                     dryer_heat_source AS heat_source FROM Dryer)
             UNION ALL
             (SELECT email, oven_no AS appliance_no,
                     oven_heat_source AS heat_source FROM OvenHeatSource)
             UNION ALL
             (SELECT email, cooktop_no AS appliance_no,
                     cooktop_heat_source AS heat_source FROM Cooktop)) AS DHS
       WHERE DA.email = DHS.email
         AND DA.appliance_no = DHS.appliance_no
       GROUP BY heat_source
       ORDER BY count_of_each_heatsource DESC) AS HS
LIMIT 1), '') AS most_common_heat_source) AS FS;
```

- Display report results: (example of output with dummy data in MySQL 8.0)
  - Dummy data with households that have heat source appliances

1 row

	1
input_center_postal_code	71937
input_search_radius	250
avg_bathroom_count_per_household_within_range	0.7
avg_bedroom_count_per_household_within_range	3.3
avg_occupant_count_per_household_within_range	5
ratio_of_commode_to_occupant_within_range	1:5.17
avg_appliance_count_per_household_within_range	1.2
most_common_heat_source_within_range	gas

  - Dummy data with households that DO NOT have any heat source appliance

	1
input_center_postal_code	71937
input_search_radius	250
avg_bathroom_count_per_household_within_range	0.7
avg_bedroom_count_per_household_within_range	3.3
avg_occupant_count_per_household_within_range	5
ratio_of_commode_to_occupant_within_range	1:5.17
avg_appliance_count_per_household_within_range	0.3
most_common_heat_source_within_range	

- If there is no data stored in the system corresponding to this report, an error message will be prompted to user to indicate this.