0.1 Question 1: Unboxing the Data

0.1.1 Question 1a

Note that the `data` table, in the full database, is billions of rows. What do you notice about the design of the database schema that helps support the large amount of data?

Despite the data table being billions of rows, having multiple tables that hold similar data significantly decreases the time it takes to query the data if the user knows exactly what they are looking for since normalization is being used to eliminate redundancy.

0.1.2 Question 1d

Do you see any issues with the schema given? Here are two worthwhile things to consider: - Can you uniquely determine the building given the sensor data? Why? - Could `buildings_site_mapping.building` be a valid foreign key pointing to `real_estate_metadata.building_name`?

There is an issue being that you may not determine a uniquely associated row in the metadata table given a row from the data table because since there is 5 places where the real_estate_metadata has the 2 or more of the same building_name so these cannot be uniquely identified. Even if we tried to set buildings_site_mapping.building to be a foreign key for real_estate_metadata.building_name, this wouldn't be valid because building_name is not a unique identifier for the real_estate_metadata table as determined by 1b.

```
In [19]: # %%sql
         # SELECT COUNT(*), COUNT(DISTINCT(building_name))
         # FROM real_estate_metadata
         '''
         Note: a foreign key column in a table points to a column with unique values in another table a
         '''
```

```
Out[19]: '\nNote: a foreign key column in a table points to a column with unique values in another table
```

0.2 Question 3: Entity Resolution

0.2.1 Question 3a

There is a lot of mess in this dataset related to entity names. As a start, have a look at all of the distinct values in the `units` field of the `metadata` table. What do you notice about these values? Are there any duplicates?

It looks like sql is detecting the differences in capitalization as distinct values, for example it looks like there are many different versions of KWH as kWh or kWH which would be an issue for us if these are meant to represent the same value.

```
In [28]: # %%sql
         # SELECT DISTINCT units
         # FROM metadata
         # LIMIT 50
```

0.2.2 Question 3b

Sometimes, entity resolution is as simple as a text transformation. For example, how many unique `units` values are there, and how many would there be if we ignored case (upper vs. lower case)? Your output should be a table with one row and two columns; the first column should contain the number of unique `units` values, and the second column should contain the number of unique `units` values if we ignored case. The two columns can have arbitrary names.

```
In [29]: %%sql result_3b <<
         SELECT COUNT(DISTINCT units), COUNT(DISTINCT (upper(units)))
         FROM metadata
```

```
   postgresql://jovyan@127.0.0.1:5432/template1
 * postgresql://jovyan@127.0.0.1:5432/ucb_buildings
1 rows affected.
Returning data to local variable result_3b
```

0.2.3 Question 3c

Arguably we shouldn't care about these alternative unit labels, *as long as each sensor class uses a single value of `units` for all its sensor ids.* After all, maybe the capitalization means something to somebody!

Write a SQL query that returns single row with one column of value `true` if the condition in italics above holds, or a single row with one column of value `false` otherwise. Please do not hard code this query - we reserve the right to penalize your score if you do so.

each sensor class uses a single value of units for all its sensor ids

```
In [32]: # %%sql
         # SELECT class, COUNT(DISTINCT units)
         # FROM metadata
         # GROUP BY class
         # LIMIT 100
```

```
In [33]: %%sql result_3c <<
         with cte as (
         SELECT class, COUNT(DISTINCT units)
         FROM metadata
         GROUP BY class)

         SELECT CASE WHEN MAX(count) > 1 THEN FALSE ELSE TRUE END as case
         FROM cte
```

```
   postgresql://jovyan@127.0.0.1:5432/template1
 * postgresql://jovyan@127.0.0.1:5432/ucb_buildings
1 rows affected.
Returning data to local variable result_3c
```

0.2.4 Question 3d

Moving on, have a look at the `real_estate_metadata` table—starting with the distinct values in the `location` field! What do you notice about these values?

```
In [37]: %%sql
         SELECT DISTINCT(location)
         FROM real_estate_metadata
```

```
   postgresql://jovyan@127.0.0.1:5432/template1
 * postgresql://jovyan@127.0.0.1:5432/ucb_buildings
15 rows affected.
```

```
Out[37]: [('LOS ANGELES',),
          ('FRANCISC O',),
          ('SYSTEMWI DE',),
          ('SAN DSAIENG O',),
          ('FRANCISC SOAN',),
          ('SANTA CRUZ',),
          ('AG FIELD STAT',),
          ('IRVINE',),
          ('RIVERSIDE',),
          ('BERKELEY',),
          ('SANTA BARBARA',),
          ('DAVIS',),
          ('FRANCISC OSAN',),
          ('MERCED',),
          ('SAN DIEGO',)]
```

There appears to be spelling errors and inconsistencies in the distinct location types as well as repeated locations with different spellings.