Stage IV – Elaboration: Database Design

2. Demonstrate that all the relations in the relational schema are normalized to Boyce–Codd normal form (BCNF).

*   ***a. For each table, specify whether it is in BCNF or not, and explain why.***
*   ***b. For each table that is not in BCNF, show the complete process that normalizes it to BCNF.***

Building

First Normal Form (1NF)

- single valued attributes: all attributes are single valued - building name, building type, building year, building square footages
- attribute domain doesn't change: domain for specified attributes will not change
- unique name in every attribute/ column: no repetitive attribute names all are unique
- order of the data doesn't matter: entering and ordering the data within the attributes will not matter

Second Normal Form (2 NF)

- no partial dependency: no partial dependencies within this building database

Third Normal Form (3NF)

- no transitive dependency for non-prime attributes as well as it is in second normal form

BCNF

NG_Entry, EL_Entry

First Normal Form (1NF)

since the functionality of the NG_Entry, EL_Entry, & Entry tables are similar

- single valued attributes: all attributes are unique
- attribute domain doesn't change: attribute domains will not change
- unique name in every attribute/ column: very value within the columns will be unique
- order of the data doesn't matter: order of the data & the display of the data are different

Second Normal Form (2 NF)

- no partial dependency: does not rely on other attributes

Third Normal Form (3NF)

- no transitive dependency for non-prime attributes as well as it is in second normal form

BCNF

SRC:

https://www.geeksforgeeks.org/first-normal-form-1nf/?ref=lbp
https://www.geeksforgeeks.org/second-normal-form-2nf/?ref=lbp
https://www.geeksforgeeks.org/third-normal-form-3nf/?ref=lbp
https://www.geeksforgeeks.org/boyce-codd-normal-form-bcnf/?ref=lbp

3. Define the different views (virtual tables) required. For each view list the data and transaction requirements. Give a few examples of queries, in English, to illustrate.

Table 1 = Building | sqFT | type

To display building square footage by type. We will use aggregate commands in order to obtain summation of area to be used in display. Used for assumption later where sqFt can be used to correlate to energy consumption.

Table 2 = ngEnergyDate ngEnergyMeter ngEnergyTherms, energyCost
Table 3 = elEnergyDate elEnergyMeter elEnergyWatts, energyCost

To display energy usage per meter, by type. To simplify, we could potentially use an aggregate command in order to summate energy usage, or view cost and usage by a specific period of time. If a user wanted to view the energy cost of a specific month, we would utilize the user input when SELECTING information and displaying it via aggregate command.

4. Design a complete set of SQL queries to satisfy the transaction requirements identified in the previous stages, using the relational schema and views defined in tasks 2 and 3 above.

Table 1 :

Obtaining summation of square foot by building type.

```
SELECT
     SUM(buildingSq_ft) AS sqft_by_type
FROM
     Buildings
GROUP BY buildingType;
```

Table 2 & 3:
Obtaining summation of usage per meter by type.
First we obtain the sum of cost of Natural Gas entries.

```
SELECT
     SUM(cost) AS totalCost
FROM
     NG_ENTRY;
```

First we obtain the sum of the cost of Electric entries.

```
SELECT
     SUM(cost) AS totalCost
FROM
     EL_ENTRY;
```

Additionally, if we wanted to view the sum cost of all meters during a specific date. We assume that searchDate as a variable has been used as user input for the specific date that should be used.

```
SELECT SUM(cost)
FROM NG_ENERGY
WHERE NG_ENERGY.date = searchDate
UNION
SELECT SUM(cost)
FROM EL_ENERGY
WHERE EL_ENERGY.date = searchDate
GROUP BY buildingType;
```

Variations of this, such as changing the statement to be specified by year instead, can be used to change the scope of the WHERE statement.