Stage V(a): Table and Queries (Group 4)

Group Members: Jason Kantner, Gordon Petry, Michael Williams, Casey Lishko, Yanaja Joyner, Len La Rocca

SQL Data Definition Queries

- Database, Tables, Constraints and View Creation Queries
    - Create database called **civstat** where all of the information will be stored
        - `createdb civstat`
    - Access PostgreSQL in the terminal to interact with the civstat database
        - `psql civstat`
    - Create **6 tables** that will be stored within civstat database, called Counties, Categories, Municipalities, Certification, Actions and Complete_Actions (In **create_tables.sql** file):
        - ```
          CREATE TABLE Counties (
          CTYcode SERIAL NOT NULL,
          CTYname VARCHAR(30) NOT NULL,
          PRIMARY KEY (CTYcode)
          );
          ```
        - ```
          CREATE TABLE Categories (
          Ccode SERIAL NOT NULL,
          Cname VARCHAR(50) NOT NULL,
          PRIMARY KEY (Ccode)
          );
          ```
        - ```
          CREATE TABLE Municipalities (
          Mcode SERIAL NOT NULL,
          Mname VARCHAR(30) NOT NULL,
          CTYcode SERIAL NOT NULL,
          PRIMARY KEY (Mcode),
          FOREIGN KEY (CTYcode) REFERENCES Counties
          (CTYcode) ON DELETE CASCADE
          );
          ```
        - ```
          CREATE TABLE Certification (
          Mcode SERIAL NOT NULL,
          Certification_Date DATE NOT NULL,
          Total_Points INT NOT NULL,
          Num_Gold_Stars INT NOT NULL,
          ```

```
Bronze_Silver VARCHAR(10) NOT NULL,
PRIMARY KEY (Mcode),
FOREIGN KEY (Mcode) REFERENCES Municipalities
(Mcode) ON DELETE CASCADE
);
```

- ```
  CREATE TABLE Actions(
  Acode SERIAL NOT NULL UNIQUE,
  Ccode SERIAL NOT NULL,
  Aname VARCHAR(100) NOT NULL,
  Point_Value INT NOT NULL,
  Priority BOOLEAN NOT NULL,
  Required BOOLEAN NOT NULL,
  PRIMARY KEY (Acode, Ccode, Aname),
  FOREIGN KEY (Ccode) REFERENCES Categories (Ccode)
  ON DELETE CASCADE
  );
  ```
- ```
  CREATE TABLE Complete_Actions(
  Acode SERIAL NOT NULL,
  Mcode SERIAL NOT NULL,
  PRIMARY KEY (Acode, Mcode),
  FOREIGN KEY (Mcode) REFERENCES
  Municipalities(Mcode) ON DELETE CASCADE,
  FOREIGN KEY (Acode) REFERENCES Actions (Acode) ON
  DELETE CASCADE
  );
  ```

- The following queries will create **19 views** that will be stored in the civstat databases to serve various uses (In **create_views.sql** file):
  - 1. Show the point total for each category
    - ```
      CREATE VIEW pt_tot_category AS
      SELECT Ccode, SUM(DISTINCT Point_Value)
      FROM Actions
      GROUP BY Ccode;
      ```
  - 2. Total actions per municipality
    - ```
      CREATE VIEW tot_act_mun AS
      SELECT Mcode, COUNT(Acode)
      FROM Complete_Actions
      GROUP BY Mcode;
      ```
  - 3. Total points per municipality
    - ```
      CREATE VIEW tot_pts_mun AS
      ```
      ```

```
SELECT Mcode, Total_Points
FROM Certification;
```

- 4. Total category each municipality contributed to
  - ```
    CREATE VIEW tot_cat_mun AS
    SELECT Mcode, COUNT(DISTINCT Ccode)
    FROM Complete_Actions JOIN Actions
    ON Complete_Actions.Acode = Actions.Acode
    GROUP BY Mcode;
    ```
- 5. Total actions per County
  - ```
    CREATE VIEW tot_act_cty AS
    SELECT CTYcode, COUNT(Acode)
    FROM Municipalities JOIN Complete_Actions
    ON Municipalities.Mcode =
    Complete_Actions.Mcode
    GROUP BY CTYcode;
    ```
- 6. Total points for County
  - ```
    CREATE VIEW tot_pts_cty AS
    SELECT CTYcode, Total_Points
    FROM Municipalities JOIN Certification
    ON Municipalities.Mcode =
    Certification.Mcode
    GROUP BY CTYcode, Total_Points;
    ```
- 7. Total Categories each County contributed to
  - ```
    CREATE VIEW tot_cat_cty AS
    SELECT CTYcode, COUNT(DISTINCT Ccode)
    FROM Municipalities JOIN Complete_Actions
    ON Municipalities.Mcode =
    Complete_Actions.Mcode
    JOIN Actions
    ON Complete_Actions.Acode = Actions.Acode
    GROUP BY CTYcode;
    ```
- 8. Shows municipalities where Num_Gold_Stars is 0
  - ```
    CREATE VIEW no_gold_stars AS
    SELECT Mcode, Num_Gold_Stars
    FROM Certification
    WHERE Num_Gold_Stars = 0;
    ```
- 9. Show municipalities where Num_Gold_Stars is 1 or more
  - ```
    CREATE VIEW many_gold_stars AS
    SELECT Mcode, Num_Gold_Stars
    ```

```
FROM Certification
WHERE Num_Gold_Stars >= 1;
```

- 10. Show municipalities where Bronze_Silver is Bronze
  - ```
    CREATE VIEW bronze_stars AS
    SELECT Mcode, Bronze_Silver
    FROM Certification
    WHERE Bronze_Silver = 'Bronze';
    ```
- 11. Show municipalities where Bronze_Silver is Silver
  - ```
    CREATE VIEW silver_stars AS
    SELECT Mcode, Bronze_Silver
    FROM Certification
    WHERE Bronze_Silver = 'Silver';
    ```
- 12. Show Actions where Priority is False
  - ```
    CREATE VIEW no_priority AS
    SELECT Acode, Aname, Ccode
    FROM Actions
    WHERE Priority IS NOT TRUE;
    ```
- 13. Show Actions where Priority is True
  - ```
    CREATE VIEW yes_priority AS
    SELECT Acode, Aname, Ccode
    FROM Actions
    WHERE Priority IS TRUE;
    ```
- 14. Show Actions where Required is False
  - ```
    CREATE VIEW no_required AS
    SELECT Acode, Aname, Ccode
    FROM Actions
    WHERE Required IS NOT TRUE;
    ```
- 15. Shows where Required is True
  - ```
    CREATE VIEW yes_required AS
    SELECT Acode, Aname, Ccode
    FROM Actions
    WHERE Required IS TRUE;
    ```
- 16. Show Actions where Point_Value is <= 10
  - ```
    CREATE VIEW small_pt_act AS
    SELECT Acode, Aname, Point_Value
    FROM Actions
    WHERE Point_Value <= 10;
    ```
- 17. Create a view where Point_Value is > 10
  - ```
    CREATE VIEW large_pt_act AS
    ```

```
SELECT Acode, Aname, Point_Value
FROM Actions
WHERE Point_Value > 10;
```

- 18. Shows the total points that were earned from each action
  - ```
    CREATE VIEW pt_tot_act AS
    SELECT Acode, SUM(Point_Value)
    FROM Actions
    GROUP BY Acode;
    ```
- 19. Shows the actions that are not completed by any municipality.
  - CREATE VIEW not_completed_actions AS
    SELECT A.Acode, A.Aname
    FROM Actions AS A LEFT JOIN Complete_Actions AS CA
    ON A.Acode = CA.Acode;

- Note: the **drop_tables.sql** and **drop_views.sql** files were used during the testing process

- Data Retrieval and Formatting Queries. Utilized through Wepscraper to retrieve necessary data. (Data is stored in the **StgV(a)_Tables_Group4 file**)
  - # Python 3 Web Scraper
  -
  - # Ensure both beautifulsoup and requests are installed:
  - # pip install beautifulsoup4
  - # pip install requests
  -
  - import requests
  - from bs4 import BeautifulSoup
  -
  - map_page = requests.get("https://www.sustainablejersey.com/certification/search-participating-municipalities-approved-actions/")
  - map_data = map_page.content
  - soup = BeautifulSoup(map_data, "html.parser")
  -
  - # lists all municipalities, listing them in the order of:
  - # Municipality, County, Certification
  - all = []
  -
  - # finds all the tr tags in soup (the webpage) containing community=row as its class
  - for tr_tags in soup.find_all(attrs={"class": "community-row"}):
    ```

```
            #  finds all the td values in a given tr tag (there should be 3)
            for td_tags in tr_tags.find_all("td"):
            # Appends the values to all and removes whitespace
            all.append(" ".join(td_tags.text.split()))
    # print(all)
    # total number should come out to 1365 with 455 municipalities
    # print(len(all))

    municipalities = []
    counties = []
    certifications = []
    i = 0
    for item in all:
            if i % 3 == 0:
            municipalities.append(item)
            if i % 3 == 1:
            counties.append(item)
            if i % 3 == 2:
            certifications.append(item)
            i = i + 1

    # print(municipalities)
    # print(len(municipalities))

    # Converts counties to a dictionary to remove duplicates
    counties = list(dict.fromkeys(counties))
    # print(counties)
    # print(len(counties))

    # Removes duplicates from certifications to show all certification values
    certifications = list(dict.fromkeys(certifications))
    # print(certifications)
    # print(len(certifications))




    # Start of Actions page
```

```
actions_page = requests.get("https://www.sustainablejersey.com/actions/")
actions_data = actions_page.content
soup2 = BeautifulSoup(actions_data, "html.parser")

# lists all categories
categories = []
# Lists all actions under each category. New categories are specified to make it easier to seperate categories from actions
actions = []
# finds all the tr tags in soup2 (the webpage) containing community=row as its class
for li_tags in soup2.find_all(attrs={"class": "action-category"}):
        #  finds all the td values in a given tr tag (there should be 3)
        i = 0
        for h3_tags in li_tags.find_all("h3"):
        # Appends the values to all and removes whitespace
        categories.append(" ".join(h3_tags.text.split()))
        actions.append("NEW_CATEGORY")
        actions.append(" ".join(h3_tags.text.split()))
        for sub_cat in li_tags.find_all(attrs={"class": "action-subcategory--header"}):
        actions.append("NEW_SUB_CAT")
        actions.append(" ".join(sub_cat.text.split()))
        # Actions are all printed in order
        for h4_tags in li_tags.find_all("h4"):
                actions.append(" ".join(h4_tags.text.split()))
        # Points are all printed in order
        for div_tags in li_tags.find_all(attrs={"class": "action--points"}):
                actions.append(" ".join(div_tags.text.split()))
        for req_tags in li_tags.find_all(attrs={"class": "action--req"}):
                actions.append(" ".join(req_tags.text.split()))

# print(categories)

# The category is first in the list, followed by a sub category (specified by NEW_SUB_CAT),
# followed by all of the actions, followed by their corresponding point values, followed by
# the items that are priority or required.
```

    - print(actions)
- Table Population Queries
    - There are **6 SQL files** submitted that contained the insert queries that inserted the data into the table. These files are called **insert_county.sql**, **insert_category.sql**, **insert_municipality.sql**, **insert_certification.sql**, **insert_action.sql**, **insert_complete_action.sql**.
    - In the VM, the files go through the psql -f statement to store the data into the civstat database.

SQL Data Manipulation Queries

- SQL Data Definition Queries
    - These queries are stored in the **manipulation_queries.sql** file that was submitted
    - These queries are an example of manipulation queries that will be utilized in the database. More may be added later in the process and it will be updated appropriately.

- Jason Kantner's Virtual Machine is where the database is stored
    - The schema is currently present in the VM
    - To access the files in the VM, it is located below:
        - Documents → JKantner → Final_Project → Stage_V
    - The command history of the above queries running in the VM terminal is stored in the **cmd_history_stgV.txt** file.

Stage IV: Design (Group 4)

Group Members: Jason Kantner, Gordon Petry, Michael Williams, Casey Lishko, Yanaja Joyner, Len La Rocca

Boyce-Codd Normal Form (BCNF) for Relations

- **Conditions that must be Satisfied for a relation to be in BCNF:**
    - **1. Must be in 3NF**
    - **2. For any dependency A → B, A should be a super key**
- Counties
    - Attributes: *CTYcode, CTYname*
    - The County relation is normalized to Boyce-Codd Normal Form.

        - This relation is normalized to BCNF because it satisfies both conditions. It satisfies the conditions for all the required forms:
            - 1NF: Values are atomic, each value in a given column is of the same type, each column has a unique name and the order of insertion does not matter.
            - 2NF: Satisfies 1NF and has no partial dependencies. There is only one prime attribute, so partial dependency is not an issue.
            - 3NF: Satisfies 2NF and has no transitive dependency. The non-prime attribute *CTYname* is not dependent on another non-prime attribute.
        - Since all of these conditions are satisfied, that means that condition 1 for BCNF is also satisfied. Condition 2 is satisfied because there are no violations. The prime attribute of the relation, *CTYcode,* is not dependent on a non-prime attribute.
- Municipalities
    - Attributes: *Mcode, Mname, CTYcode, Certification_Date, Total_Points, Certified Num_Gold_Stars, Bronze_Silver*
        - Foreign Keys: *CTYcode*
    - The Municipalities relation is not normalized to Boyce-Codd Normal Form
        - This relation is not normalized to BCNF because it violates the first condition. This relation is not in 3NF because there are transitive dependencies present. The non-prime attributes *Certification_Date, Total_Points, Num_Gold_Stars, Bronze_Silver* are all dependent on the non-prime attribute, *Certified*. However, condition 2 is satisfied because no prime attributes (*Mcode*) are dependent on non-prime attributes.
    - To normalize this relation into BCNF, it must be decomposed into two relations as listed below
        - **Municipalities:** *Mcode, Mname, CTYcode*
            - Foreign Keys: *CTYcode*
        - **Certification:** *Mcode, Certification_Date, Total_Points, Num_Gold_Stars, Bronze_Silver*
            - Foreign Key: *Mcode*
    - With both of these relations, there are no longer any transitive dependencies because the non-prime attributes of each relation are only dependent on their respective prime attributes. Also, we decided to cut the *Certified* attribute because all of the municipalities we are storing are certified. Although we did not have time to implement, we would have liked to have non-certified municipalities as well. With this change, condition 1 is now satisfied along with condition 2, meaning that these relations are both normalized in BCNF.

- Actions
    - Attributes: *Acode, Mcode, Ccode, Aname, Point_Value, Priority, Required*
        - *Acode, Mcode* and *Ccode* form a composite key
        - Foreign Keys: *Mcode* and *Ccode*
    - The Actions relation is not normalized to Boyce-Codd Normal Form
        - This relation violates condition 1 because it is not in 2NF, meaning it cannot be in 3NF. This relation is not in 2NF because there are partial dependencies
            - *Acode -> Aname*
        - To normalize this relation into BCNF, it must be decomposed into two relations as listed below:
            - **Actions:** *Acode, Ccode, Aname, Point_Value, Priority, Required*
            - **Completed Actions:** *Acode, Mcode,*
    - With these new relations, there are no longer any partial dependencise because each non-prime attribute depends on each prime attribute in the relation. There is also no transitive dependency which fulfils condition one. Finally, the prime attributes are not derived from non-prime attributes thus fulfilling condition 2, making this new relation normalized in BCNF.
- Categories
    - Attributes: *Ccode, Cname*
    - The Categories relation is normalized to Boyce-Codd Normal Form
        - This relation is normalized to BCNF because it satisfies both conditions. It satisfies the conditions for all the required forms:
            - 1NF: Values are atomic, each value in a given column is of the same type, each column has a unique name and the order of insertion does not matter
            - 2NF: Satisfies 1NF and has no partial dependencies. There are no partial dependencies because there is only one prime attribute.
            - 3NF: Satisfies 2NF and has no transitive dependency. The non-prime attribute *Cname* is not dependent on another non-prime attribute.
        - Since all of these conditions are satisfied, that makes condition 1 of BCNF satisfied as well. Condition 2 is satisfied because no prime attributes are dependent on non-prime attributes. The only prime attribute of this relation, *Ccode* is not dependent on a non-prime attribute.

Views

- Define the different views required. For each view list the data and transaction requirements. Give a few examples of queries, in English, to illustrate.
- What do we want users to be able to display based on the data in the database? (Views Required)
- Each will be put in the format of (X axis) v (Y axis) for a bar graph, where a pie chart the X axis will correlate to the colored item, and the total will correlate to the size of each section.
    - 1 Specific Municipality
        - Categories v Point total
        - Categories v # of gold actions
        - Categories v # of silver actions
        - Categories v Total actions
    - N Municipalities
        - Municipalities v Point total
        - Municipalities v Number of gold actions
        - Municipalities v Number of silver actions
        - Municipalities v Total actions
        - Municipalities v categories contributed to
    - All Municipalities
        - Municipalities v Point total
        - Municipalities v Number of gold actions
        - Municipalities v Number of silver actions
        - Municipalities v Total actions
        - Municipalities v categories contributed to
    - 1 Specific County
        - Municipalities v Point total
        - Municipalities v # of gold actions
        - Municipalities v # of silver actions
        - Municipalities v Total actions
        - Municipalities v # of categories contributed to
    - N Counties
        - Counties v Point total
        - Counties v Number of gold actions
        - Counties v Number of silver actions
        - Counties v Total actions
        - Counties v categories contributed to
    - All Counties
        - Counties v Point total
        - Counties v Number of gold actions

        - Counties v Number of silver actions
        - Counties v Total actions
        - Counties v categories contributed to

Queries

- Design a complete set of queries to satisfy the transaction requirements identified in the previous stages.

These queries are written out as a mix of sql and english. Parentheses are used to make joins easier to read

- 1 specific municipality
    - Categories v Point Total
        - X Axis: Cname of desired Categories
        - Y Axis: SUM of Point_Value FROM ((Municipality INNER JOIN Action ON Mcode WHERE Mname = “User Input”) INNER JOIN Category ON Ccode) WHERE Cname = “User chosen category”
          (Must be done for every category)
    - Categories v # of Gold Actions
        - X Axis: Cname of desired Categories
        - Y Axis: COUNT Acode FROM (((Municipality INNER JOIN Action ON Mcode WHERE Mname = “User chosen”) INNER JOIN Gold Star Standards ON Acode) INNER JOIN Category at Ccode) WHERE Cname = “User chosen category”
          (Must be done for every category)
    - Categories v # of Silver Actions
        - X Axis: Cname of desired Categories
        - Y Axis: COUNT Acode FROM (((Municipality INNER JOIN Action at Mcode WHERE Mname = “User Chosen”) INNER JOIN Regular Actions ON Acode WHERE Silver? = True) INNER JOIN Category ON Ccode) WHERE Cname = “User chosen category”
          (Must be done for every category)
    - Categories v # of total Actions
        - X Axis: Cname of desired Categories
        - Y Axis: COUNT Acode FROM ((Municipality INNER JOIN Action ON Mcode WHERE Mname = “User Chosen”) INNER JOIN Category ON Ccode) WHERE Cname = “User chosen category”
          (Must be done for every category)

- N municipalities
    - Municipalities v Point total
        - X: SELECT Mname FROM Municipality WHERE Mname = *user input* AND Mname = *user input 2* AND … Mname = *user input N*
        - Y: SELECT Total_Point FROM (Municipality INNER JOIN Participating Municipality ON Mcode = Mcode) WHERE Mname = *user input* AND Mname = *user input 2* AND … Mname = *user input N*
    - Municipalities v Number of gold actions
        - X: SELECT Mname FROM Municipality WHERE Mname = *user input* AND Mname = *user input 2* AND … Mname = *user input N*
        - Y: SELECT count(*) FROM ((Municipality INNER JOIN Action ON Mcode = Mcode) INNER JOIN Gold Star Standards ON Acode = Acode) WHERE Mname = *user input* AND Mname = *user input 2* AND … Mname = *user input N*
    - Municipalities v Number of silver actions
        - X: SELECT Mname FROM Municipality WHERE Mname = *user input* AND Mname = *user input 2* AND … Mname = *user input N*
        - Y: SELECT count(*) FROM (Municipality INNER JOIN Action ON Mcode = Mcode INNER JOIN Regular Actions ON Acode = Acode) WHERE Silver? = ‘True’ AND Mname = *user input* AND Mname = *user input 2* AND … Mname = *user input N*
    - Municipalities v Total actions
        - X: SELECT Mname FROM Municipality WHERE Mname = *user input*
        - Y: SELECT count(*) FROM (Municipality INNER JOIN Action ON Mcode = Mcode) WHERE Mname = *user input* AND Mname = *user input 2* AND … Mname = *user input N*
    - Municipalities v categories contributed to
        - X: SELECT Mname FROM Municipality WHERE Mname = *user input* AND Mname = *user input 2* AND … Mname = *user input N*
        - Y: SELECT count(Cname) FROM (Municipality INNER JOIN Action ON Mcode = Mcode INNER JOIN Category ON Ccode = Ccode) WHERE Mname = *user input* AND Mname = *user input 2* AND … Mname = *user input N*

- ALL Municipalities
    - Municipalities v Point total
        - X: SELECT Mname FROM Municipality
        - Y: SELECT Total_Point FROM (Municipality INNER JOIN Participating Municipalities ON Mcode = Mcode)

    - Municipalities v Number of gold actions
        - X: SELECT Mname FROM Municipality
        - Y: SELECT count(*) FROM (Municipality INNER JOIN Action ON Mcode = Mcode INNER JOIN Gold Star Standards ON Acode = Acode)
    - Municipalities v Number of silver actions
        - X: SELECT Mname FROM Municipality
        - Y: SELECT count(*) FROM (Municipality INNER JOIN Action ON Mcode = Mcode INNER JOIN Regular Actions ON Acode = Acode) WHERE Silver? = 'True'
    - Municipalities v Total actions
        - X: SELECT Mname FROM Municipality
        - Y: SELECT count(*) FROM (Municipality INNER JOIN Action ON Mcode = Mcode)
    - Municipalities v categories contributed to
        - X: SELECT Mname FROM Municipality
        - Y: SELECT count(Cname) FROM (Municipality INNER JOIN Action ON Mcode = Mcode INNER JOIN Category ON Ccode = Ccode)

- 1 Specific County
    - Municipality v Point Total
        - X Axis: Mname's FROM (Municipalities INNER JOIN County ON CTYcode)
        - Y Axis: SELECT Total_Point FROM ((Municipalities INNER JOIN County ON CTYcode) INNER JOIN Participating Municipalities ON Mcode)
    - Municipality v # of gold actions
        - X Axis: Mname's FROM (Municipalities INNER JOIN County ON CTYcode)
        - Y Axis: SELECT #_of_Gold_Stars FROM ((Municipalities INNER JOIN County ON CTYcode) INNER JOIN Participating Municipalities ON Mcode)
    - Municipalities v # of silver actions
        - X Axis: Mname's FROM (Municipalities INNER JOIN County ON CTYcode)
        - Y Axis: SELECT #_of_Silver_Stars FROM ((Municipalities INNER JOIN County ON CTYcode) INNER JOIN Participating Municipalities ON Mcode)
    - Municipalities v Total Categories

        - X Axis: Mname's FROM (Municipalities INNER JOIN County ON CTYcode)
        - Y Axis: COUNT Ccode FROM ((Municipalities INNER JOIN County ON CTYcode) INNER JOIN Action ON Acode)
          (Must be done for every Municipality)
    - Municipalities v # of categories contributed to
        - X Axis: Mname's FROM (Municipalities INNER JOIN County ON CTYcode)
        - Y Axis: COUNT Ccode FROM (Municipalities INNER JOIN County ON CTYcode) INNER JOIN (Action INNER JOIN Category ON Ccode) ON Mcode

- N Counties and ALL Counties
    - Counties v Point total
        - X Axis: CTYName's
        - Y Axis: SUM Total_Point FROM ((Municipalities INNER JOIN County ON CTYcode) INNER JOIN Participating Municipalities ON Mcode)
          (Do the above query for selected counties)
    - Counties v Number of gold actions
        - X Axis: CTYName's
        - Y Axis: SUM #_of_Gold_Stars FROM ((Municipalities INNER JOIN County ON CTYcode) INNER JOIN Participating Municipalities ON Mcode)
          (Do the above query for selected counties)
    - Counties v Number of silver actions
        - X Axis: CTYName's
        - Y Axis: SUM #_of_Silver_Stars FROM ((Municipalities INNER JOIN County ON CTYcode) INNER JOIN Participating Municipalities ON Mcode)
          (Do the above query for selected counties)
    - Counties v Total actions
        - X Axis: CTYName's
        - Y Axis: COUNT Acode FROM ((Municipalities INNER JOIN County ON CTYcode) INNER JOIN Action ON Acode)
          (Do above query for selected counties)
    - Counties v categories contributed to
        - X Axis: CTYName's

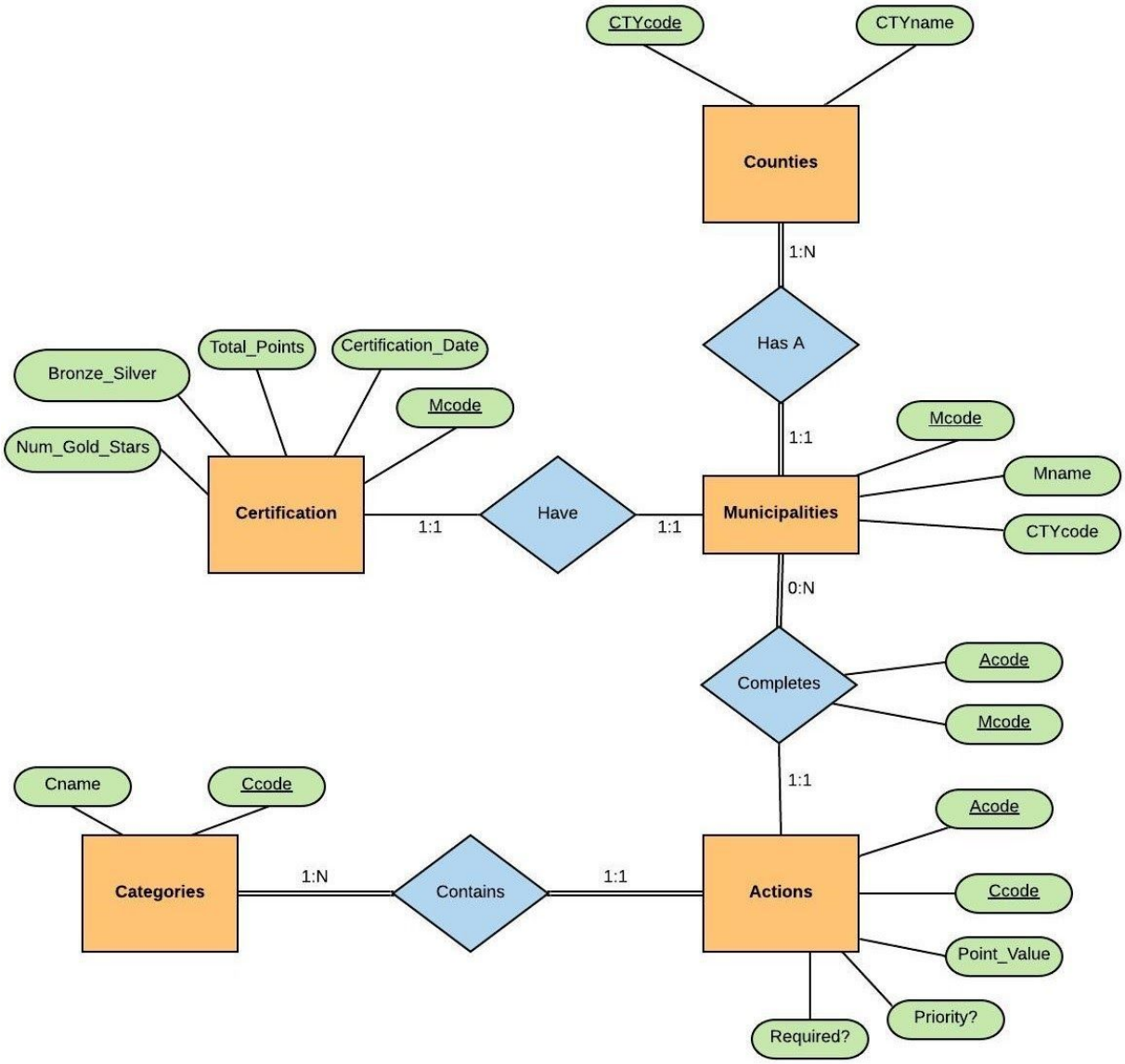
- Y Axis: COUNT Ccode FROM (Municipalities INNER JOIN County ON CTYcode) INNER JOIN (Action INNER JOIN Category ON Ccode) ON Mcode
  (Do above query for selected counties)

Stage III: Data Model (Group 4)

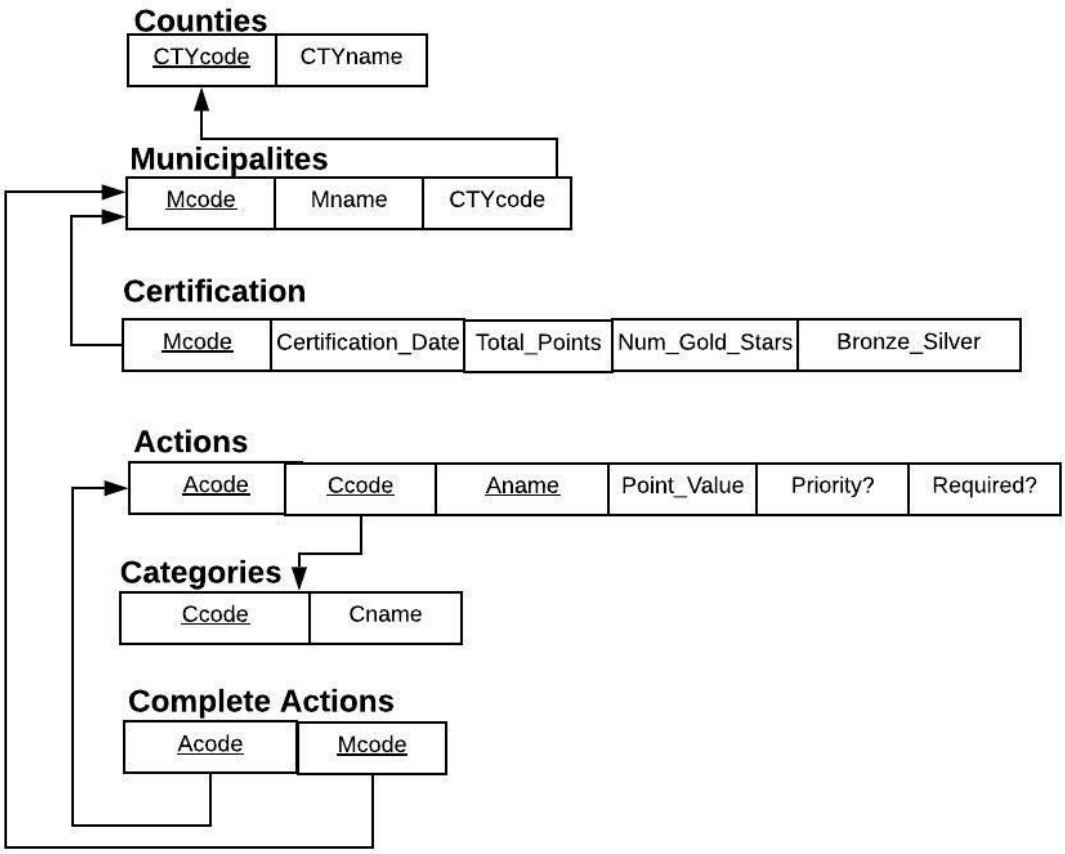
Group Members: Jason Kantner, Gordon Petry, Michael Williams, Casey Lishko, Yanaja Joyner, Len La Rocca

Link to LucidChart:
https://www.lucidchart.com/documents/edit/897a80eb-bf1d-41a9-9c7d-81e4dfc2bb1c/0_0?shared=true

ER Diagram



Relational Schema



Additional Information

- Initial Database Size (approximate number of records)
    - Our initial database hold **844** records
        - Counties: 21
        - Municipalities: 455
        - Certification: 204
        - Actions: 152
        - Categories: 18
        - Complete Actions: 63
- Types and average number of searches
    - Our initial estimate of queries that will be used in a typical user session is around **10 queries**

- We got this rough estimate because we are planning to have multiple drop-down menus where users can choose many variables to filter the data to their liking. The menus we currently have are below…
    - Graph Type
    - Municipality / Location
    - Sustainability Point Amount
    - Gold Star?
    - Silver Star?
    - Bronze Star?
    - Certifications
    - Certification Date Range
    - Actions Taken
    - Categories of Entries
- *Note: We are likely going to implement a way for users to generate up to 2 charts at once so they can compare data. This would bring the potential query usage per session up to around* ***20 queries***

Stage II: Project Proposal and Specifications (Group 4)

Group Members: Jason Kantner, Gordon Petry, Michael Williams, Casey Lishko, Yanaja Joyner, Len La Rocca

Link to GitHub Wiki: https://github.com/CSC-315/cab-civicstats-group-4

Problem Statement

For this project, the main problem that we want to address and find a solution for is the lack of easily accessible/understandable sustainability data on the CivicStory website. Currently, the CivicStory website is a hub for relevant news articles relating to sustainability, however direct information on sustainability efforts based on geographic location is not accessible to the common user.

Objective of the Module

The objective of this module is to create an addition to the CivicStory website that will allow normal users to access detailed data on sustainability in their area. Another objective which we hope to accomplish is to ensure that this data is easily understandable to the common user and it is tailored to what information they want to know. It is our hope that this tool will keep the public well informed about sustainability in their own area.

Desired End Product / Part Developed for This Class

In general, the product we plan to design can be used by many different websites and is not only limited to CivicStory. The module that will be designed will be a page where the user can choose from many different variables relating to data in a connected database. Once the user specifies the data they want to see, the data will then be presented as a bar graph on the user's screen. After all of the selections are complete, queries will pull data from the connecting database in PostgreSQL and the desired data will be visualized onto the screen of the user. For this class, the data in the database will be relating to sustainability in the New Jersey area. This makes the planned module a great tool to be utilized on the CivicStory website because it strongly relates to the purpose of the website and the goals it wants to accomplish.

Importance of this Module and How it Addresses the Problem

Right now CivicStory is mostly a stream of news articles relating to sustainability efforts. Although informative, these do not offer the user a look at raw data that relates to the issue. Allowing the user to search for specific data gives them a true understanding of the issues revolving around sustainability and what can be done to improve it in the future. Also, they can compare and contrast data from different times or places to get the "whole picture" of where sustainability is, where it comes from, and where it needs to go.

Problem Domain Research Plan and Data Obtainment

To research the problem domain, we will look over the CivicStory website to see what type of data would be relevant to put in the database connected to the website. Not every piece of sustainability data we find should be put in the database. For example, we need to have a limited geographic location so it remains relevant to those that will access the CivicStory website. Once we research the problem domain in more depth, it will then be much easier to figure out what kind of data we will implement into our database. We obtained our data from the Sustainable Jersey website. This website has a large amount of data regarding sustainability in New Jersey divided by counties. We will collect this data and store it in our PostgreSQL database.

Similar Systems and How our System Differs

Search bars and user interfaces that allow for specific search terms do exist in many forms (Ex. PAWS class search tool). However, the addition that we plan to make ours stand out from others in that our search tool creates a data visualization based on the information provided by the user on what data they want to view.

Other Possible Applications for this System

The search system has wide applications on many databases. It is a robust search system that visualizes the data, which is useful in any database. Possibly, instead of typing in the search

criteria, the user can point out on a map where they want to search, or use a slider to determine what time period they want to look at. However, the data visualization tool will always be used after the search.

Performance

The system we plan to design will ensure efficiency between the many programs that are being utilized. Our goal is to reduce lag time as much as possible between the Database Management System and the web interface. This way, when the user prompts to create a data visualization, the task will be completed in a reasonable amount of time. Performance and efficiency should always be a top priority when creating a new web tool to use. This will be accomplished by taking time and planning out our code before going into the programs right away. We will also continue to test our system throughout the development process to see what areas we need to improve.

Security

Regarding the security of our system, we don't plan to allow users to directly alter the data within the PostgreSQL database. This access will be limited to only administrators, who are allowed to do so. For the users, they will only be able to select the parameters in which they want to query the data and view the visualization that it provides. So, the users are only limited to viewing the web interface side of this module.

Backup and Recovery

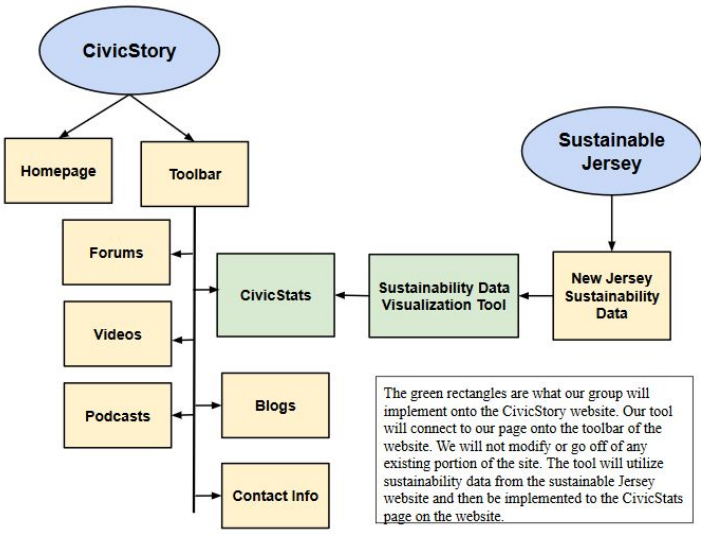
Backup and recovery are an important part when implementing a tool that utilizes data. For our project, we plan to have a copy of the database that will be connected to the web interface. This means that if anything happens to the database connected to the website, we will have a copy of the data just in case it needs to be utilized. This copy database will also be within PostgreSQL which makes it readily available to replace the original database if need be.

Technology and Database Concepts to Learn

For this project, there are a few database concepts and technologies that our group will need to learn to be able to accomplish our objective. First, our team will need to learn how to utilize the Database Management System PostgreSQL. It is within PostgreSQL where our database containing the sustainability data will be located. We will also utilize HTML to create the web interface side of this project. This will create the frontend of the project where the user will view the data visualization tool. To implement this data visualization tool, we will also learn how to use data visualization software. Data visualization software is abundant and is also vital to properly creating our module. Finally, Python will be learned and utilized to bridge the gap between the web interface and the Database Management System. Python is very important to

learn because without it, we would not be able to make the connection between the front and back end portions of this module.

Diagrammatic Representation



Quad Chart