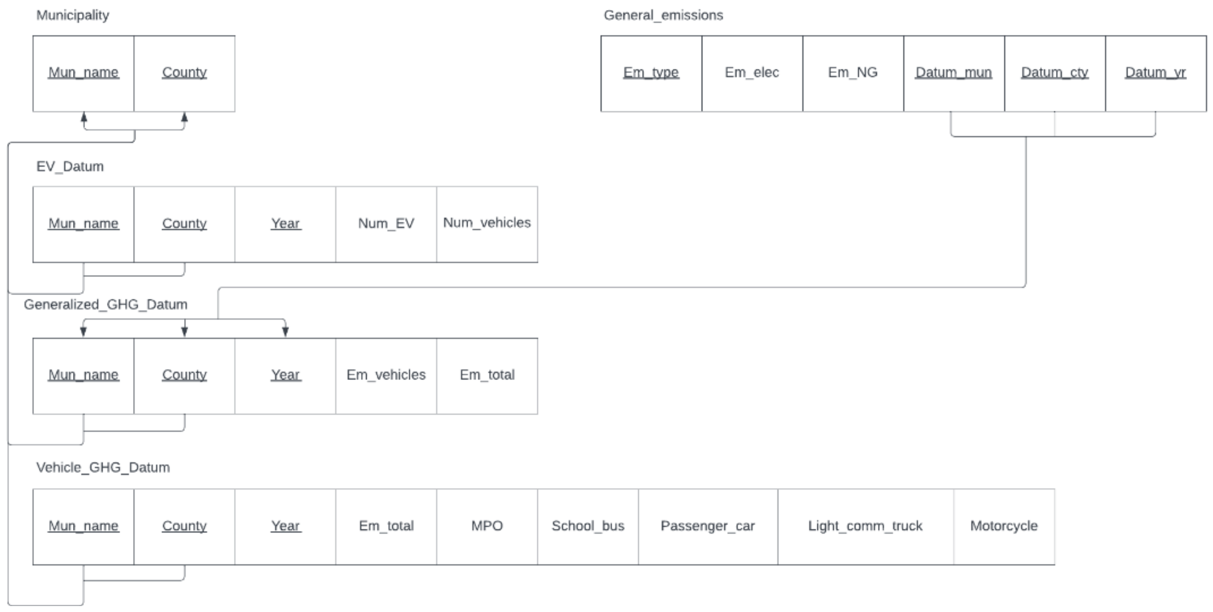
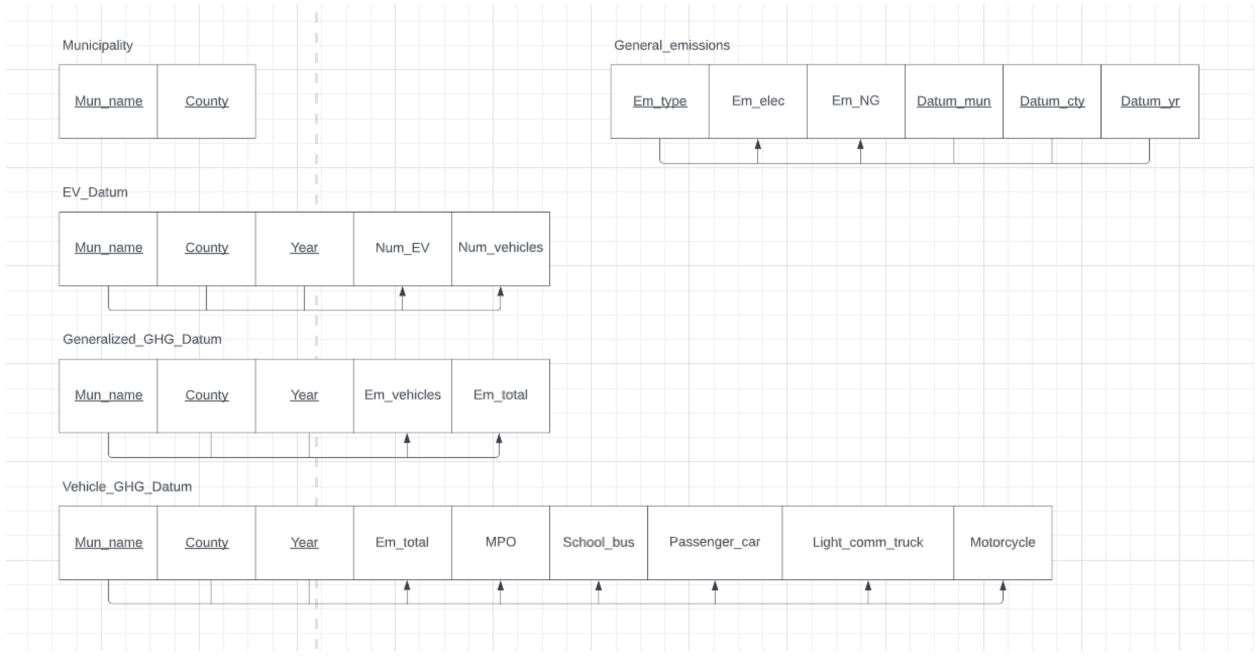
Elliot Topper, Spandana Bondalapati, Shannon Joseph
CSC 315-01
Phase IV - Elaboration: Database Design

Initial relations:



Initial relations with FDs (as no further normalization is needed, these are also the final relations):



Normalization to BCNF

a. 1NF: Relations are already normalized to 1NF because there are no multivalued attributes or nested relations.

b. 2NF: There are no partial dependencies so the relations are already normalized to 2NF.

c. 3NF: There are no transitive dependencies so the relations are already normalized to 3NF.

d. BCNF: In BCNF, there should be no dependencies on non prime attributes.

   Generalized_GHG_Datum: Mun_name, County, Year → Em_vehicles, Em_total
   - This relation is already in BCNF, as the candidate key (Mun_name, County, Year) determines all other attributes.

   Vehicle_GHG_Datum: Mun_name, County, Year → Em_total, MPO, School_bus, Percentage_car, light_comm_truck, motorcycle
   - This relation is already in BCNF, as the candidate key (Mun_name, County, Year) determines all other attributes.

   EV_Datum: Mun_name, County, Year → num_EV, num_vehicles
   - This relation is already in BCNF, as the candidate key (Mun_name, County, Year) determines all other attributes.

   General_emissions: Em_type, Datum_mun, Datum_cty, Datum_yr → Em_elec, Em_NG
   - This relation is already in BCNF, as the candidate keys determine all the other attributes.

   Municipality: Mun_name → County
   - This relation is already in BCNF, as the candidate key (Mun_name) determines all other attributes.

2. Creation of views

   a. Because of the way data is queried in our project, the use of views prevents undesired data manipulation by the user, as well as simplifying the queries required in the front-end web application such that calculations are automatically computed as needed.

   b. The first view is Mun_EV_Percentage, which simply retrieves the percentages required for a given municipality. Its definition is as follows:
   ```
   CREATE VIEW Mun_EV_Percentage AS SELECT m.Mun_name,
   m.County, e.Year, ((e.Num_EV * 1.0) / e.Num_vehicles) AS
   EV_percentage FROM Municipality AS m NATURAL JOIN EV_Datum
   AS e;
   ```
   This view contains for each data point in the EV_Datum relation, a percentage of people that have EVs at the given time in the location. This makes it easier to

query for the percentage. This requires only data from the EV_Datum relation and is read-only, making it easy to implement with no transactional issues.

c. The second view is Mun_EV_Average:

```
CREATE VIEW Mun_EV_Average AS SELECT m.Mun_name, m.County,
((SELECT EV_percentage FROM Mun_EV_Percentage WHERE Mun_name
= m.Mun_name AND County = m.County ORDER BY Year DESC LIMIT
1) - (SELECT EV_percentage FROM Mun_EV_Percentage WHERE
Mun_name = m.Mun_name AND County = m.County ORDER BY Year
ASC LIMIT 1)) / (MAX(Year) - MIN(Year)) AS Avg_EV_change
FROM Municipality AS m NATURAL JOIN Mun_EV_Percentage GROUP
BY Mun_name, County;
```

This view combines the percentage data stored in Mun_EV_Percentage for a given municipality by giving the average annual change in EV ownership percentage between the years stored. It assumes that, as is true for the data we have been provided, all statistics are measured over exactly two years, and only averages the first and last measured data. Like the previous view, this view requires only data from the EV_Datum relation and is read-only, making it easy to implement with no transactional issues.

d.

```
CREATE VIEW County_EV_Average AS SELECT County,
AVG(Avg_EV_change) AS County_avg FROM Mun_EV_Average GROUP
BY County;
```

This view, County_EV_Average, stores the average EV data for each county. It has the same assumptions and limitations as the view it uses, Mun_EV_Average.

e. There is another view created to find the average GHG emissions data, stored in Mun_GHG_Average. It is defined as follows:

```
CREATE VIEW Mun_GHG_Average AS SELECT m.Mun_name, m.County,
((SELECT Em_vehicles FROM Generalized_GHG_Datum WHERE
Mun_name = m.Mun_name AND County = m.County ORDER BY Year
DESC LIMIT 1) - (SELECT Em_vehicles FROM
Generalized_GHG_Datum WHERE Mun_name = m.Mun_name AND County
= m.County ORDER BY Year ASC LIMIT 1) / (MAX(Year) -
MIN(Year)) AS Avg_em_vehicle_change, (MAX(Em_total) -
MIN(Em_total) / (MAX(Year) - MIN(Year)) AS Avg_em_change
FROM Municipality AS m NATURAL JOIN Generalized_GHG_Datum
GROUP BY Mun_name, County;
```

This view combines the percentage data stored in Generalized_GHG_Datum for a given municipality by giving the average annual change in GHG emission data between the years stored, both in total and for vehicles. All of the assumptions are made as for the two previous views, only for the Generalized_GHG_Datum relation. There are several transaction requirements: first of all, the transaction must make sure that the Generalized_GHG_Datum table is present and that the user or role creating the view has permission to access the tables created in the view. The transaction must also ensure that the SELECT statement used to create the view doesn't violate any constraints or rules defined in the database. The view must also be named correctly and not conflict with any other views in the database and must also be created atomically, which means it should be fully created and not incomplete.

f. `CREATE VIEW County_GHG_Average AS SELECT County, AVG(Avg_em_vehicle_change) AS County_vehicle_avg, AVG(Avg_em_change) AS County_total_avg FROM Mun_GHG_Average GROUP BY County;`
   The County_GHG_Average view stores the average GHG emissions data for each county. Furthermore, for the transaction requirements, it must ensure that the Mun_GHG_Average view used in the SELECT statement in the County_GHG_Average view is available and accessible. Also, for the transaction requirements it must ensure that the SELECT statement used to create a view doesn't violate constraints that the database had defined. The transaction must also ensure that the needed permissions are granted to the user.

3. Queries
   a. The queries using the views are:
      i. `SELECT Mun_name, County, (Avg_EV_change / (SELECT MAX(Avg_EV_change) FROM Mun_EV_Average WHERE County = <county-name>)) AS normalized FROM Mun_EV_Average WHERE County = <county-name> ORDER BY Avg_EV_change DESC;`
         This selects the normalized values for EV average changes in a county, ordered so that ranking is automatically provided from highest to lowest EV usage (by percentage). It will return normalized values in the range -1 to 1, which can be dealt with on the frontend.
      ii. `SELECT County, (County_avg / (SELECT MAX(County_avg) FROM County_EV_Average)) AS normalized FROM County_EV_Average ORDER BY County_avg DESC;`
         This serves the same purpose as the previous query, but compares averages for each county as opposed to each municipality in a county.
      iii. `SELECT Mun_name, County, (Avg_em_vehicle_change / (SELECT MAX(Avg_em_vehicle_change) FROM Mun_GHG_Average WHERE County = <county-name>)) AS`

```
normalized FROM Mun_GHG_Average WHERE County =
<county-name> ORDER BY Avg_em_vehicle_change DESC;
```

This selects the normalized values for vehicle GHG changes in a county, also from -1 to 1.

iv.
```
SELECT Mun_name, County, (Avg_em_change  / (SELECT
MAX(Avg_em_change) FROM Mun_GHG_Average WHERE County =
<county-name>)) AS normalized FROM Mun_GHG_Average
WHERE County = <county-name> ORDER BY Avg_em_change
DESC;
```

This is similar to query 3, but selects the normalized total GHG change for a municipality, not just for vehicles.

v.
```
SELECT County, (County_vehicle_avg / (SELECT
MAX(County_vehicle_avg) FROM County_GHG_Average)) AS
normalized FROM County_GHG_Average ORDER BY
County_vehicle_avg DESC;
```

vi.
```
SELECT County, (County_total_avg / (SELECT
MAX(County_total_avg) FROM County_GHG_Average)) AS
normalized FROM County_GHG_Average ORDER BY
County_total_avg DESC;
```

Both queries 5 and 6 are versions of queries 3 and 4 that have been generalized to county averages using County_GHG_Average, as opposed to Mun_GHG_Average. Otherwise, they are the same.