MET CS 669 Database Design and Implementation for Business
Term Project Iteration 6

Name: Katherine Rein

Table of Contents

Project Direction Overview

For the past 2 years I have been involved in a research lab with Dr. Christoph Nolte in the Earth and Environment and Data Science departments. The work I've been doing involves validating a database of land transaction values for conservation work all over the US. One of the goals of this project is to create an economic model to predict land costs. Another reason that this project was taken up, was so that finding land transaction data was easier and more studies could be done using this data. Public data is notoriously awful and so creating a publicly available dataset would do wonders for the research world.

Building off of this work that I have been doing, I would love to use this project to create a way to help with understanding the data we collected better. This could mean lots of different things, but I see this as a way to help merge, visualize, and validate our data. I assume this will simplify the process for the end user. With a simplified process, people that don't know much about coding but know a lot about environmental science and economics can use this data efficiently. This means that the database will primarily be used by scientists and researchers but could be used by anyone wanting to understand this data.

The database will contain all the fields that we validated as a lab group (Appraised Value, Purchase Price, Acreage, Date of Transaction, Protection Type, Government Spending). I am unsure how much identification data I will need but I have multiple fields for that as well (CT ID, Municipality, County, State, Assessor Parcel Numbers). That is where I will start in terms of fields that the database has, however, that could change as I make it more complex. One way I could add data would be including GIS polygons for each transaction. This would allow us to manipulate the data spatially. I am unsure if our tools are set up for GIS data, but it could be a fun extension of the class. I am currently applying to jobs every day that utilize GIS daily to help with their environmental analysis.

I am interested in this topic not only because I want to help make my life easier as a research assistant, but because I want to feel like I've contributed to helping my lab group. While I have validated almost 700 transactions all over the country, I don't feel like I have done much of anything meaningful at the job. Being able to create a tool that will help beyond validation work sounds right up my alley. Hopefully this will be more than just something for me and people can use this for their research to make a difference in the climate crisis.

Use Cases and Fields

(1) Input Data Use Case

Data has already been inputted and we are adding to it or we are starting a brand new database of information.

1. Load in all CT IDs, polygons, and APNs for the whole of the US
2. Ensure data contains at least one of the above 3 identifiers for each row and that each is unique
3. Handle duplicate data (within a data input and with existing data)
    a. Create additional rows or replace rows
    b. Combine Assessor Parcel Numbers into one row based on CT IDs
4. Delineated which column of the inputted data goes in which of our column names

Field	What it Stores	Why it's Needed

Data_Column_Names	These are the column names on the inputted data that we want to use.	This will tell the computer which of the columns to look at in the data inputted to make sure the right data gets inputted in the right spot. This also helps with efficiency as these databases can be very large.
Our_Column_Names	These are the column names on the database that we want to update with the new data.	This will tell the computer exactly which column to line up the values from Data_Column_Names with.
CT_ID	This is the unique identifier number from the Conservation Almanac. This is most often used by the PLACES Lab and is considered the most specific identifier.	This unique identifier helps researchers add more information or update information.
Polygon	This stores the GIS data that tells us spatially where the data is.	This unique identifier helps researchers add more information or update information. It also helps in matching data without a ct_id to a row with a ct_id.
APN	This stores the Assessor Parcel Number identifier. This is an identifier that is most commonly used by the government and specifically the tax assessor when discussing properties.	This unique identifier helps researchers add more information or update information. It also helps in matching data without a ct_id to a row with a ct_id.

(2) Validate Data Use Case

Check the data from the Conservation Almanac to see how far off the data was.

1. Select a variable to validate
2. Load in data from The Trust for Public Land's Conservation Almanac for the selected variable
3. Calculate percent difference for the variable for all rows of our data that have a ct_id
4. Graph (using either a bar chart, a histogram, a scatterplot, a line graph) or map the percent difference to visualize the accuracy of the conservation almanac (and potentially locate input errors

Field	What it Stores	Why it's Needed
Validation_Variable	This stores the variable that we want to investigate in relation to the Conservation Almanac.	This is needed to limit how much data we have to load in and look at if we only need to look at one variable at a time.

Almanac_Data	This is the data from the conservation almanac for the Validation_Variable.	Without it, we would have nothing to compare our data to.
Our_Data	This is our data filtered down to just the Validation_Variable.	Without it, we would have nothing to validate.
Percent_Difference_Validation	This stores how much our data and the conservation almanac data differs.	This is how we can tell how needed our validation was. It can also help locate input errors.

(3) Under/Overpaying Use Case

Identify if each transaction was a deal or not based on the appraised value and the purchase price.

1. Identify all rows in which there is both appraised value and a purchase price
2. Calculate the percent difference between the appraised value and the purchase price
    a. A negative value means the buyers paid less and a positive value means they paid more
3. Graph (using either a bar chart, a histogram, a scatterplot, a line graph) or map the percent difference to visualize the areas where buyers were over or under paying for the land or easements

Field	What it Stores	Why it's Needed
Appraised_Value	This stores the appraised value of the transaction for that polygon, ct_id, or APN.	This will tell us how much the transaction was worth at or around the time of the transaction.
Purchase_Price	This stores the purchase price of the transaction for that polygon, ct_id, or APN.	This will tell us exactly how much was spent on the transaction.
Percent_Difference_Cost	This stores the percent amount that the purchase price is greater than the appraised value.	This will tell us if the buyer had to pay more or less than the Fair Market Value and by how much.

(4) Best Year Use Case

Given a span of years, identify the best year to buy land or an easement in a certain area.

1. Identify a range of years that you would like to study
2. Identify an area of interest that you would like to study (city, state, municipality, county, region, etc.)
3. Normalize the purchase price to 2024 dollars
4. Calculate the Price per acre for each transaction in that area during that time frame
5. Map/Graph (using either a bar chart, a histogram, a scatterplot, a line graph) the data to see if there is a year with the cheapest per acre cost

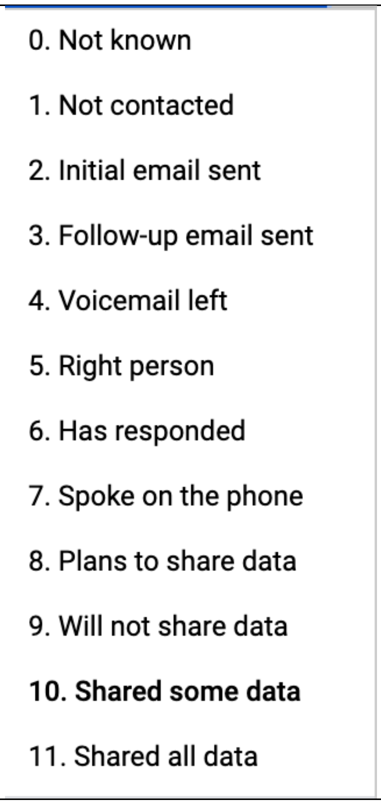
Field	What it Stores	Why it's Needed
Area_of_Interest	This is the city, state, municipality, county, region, etc. that we would like to study.	This sorts our data down to just the area of interest that we want to look at making it more efficient to compute and analyze.
Year_Range	This is range of years that we would like to study.	This sorts our data down to just the time frame that we want to look at making it more efficient to compute and analyze.
Purchase_Price	This stores the purchase price of the transaction for that polygon, ct_id, or APN.	This will tell us exactly how much was spent on the transaction.
Normalized_Purchase_Price	This stores the purchase price of the transaction for that polygon, ct_id, or APN while accounting for inflataion and normalizing it to 2024.	This will tell us how much was spent on the transaction while taking inflation into account so that the data isn't skewed towards older transactions.
Acreage	This stores the acreage of each transaction for that polygon, ct_id, or APN.	This will tell us how much land was bought or had an easement placed on it.
Price_per_Acre	This stores how much was spent per acre for each transaction.	This will create a more equal measure of how much is being spent on the land as transactions are of all different sizes but an acre is one size.

(5) Response Rate Use Case

See how different types of transactions are treated during the validation process in terms of response rate.

1. Filter rows by contact status using a binary method
    a. A transaction either has a response or no response
2. Select a value to test
    a. Ex) Local/State/Federal, Size of Transactions, States
3. Sort transactions into different buckets of said value
4. Graph (using either a bar chart, a histogram, a scatterplot, a line graph) or map to view trends of the response rate

Field	What it Stores	Why it's Needed
Contact_Status	This stores a value from 0 to 11 indicating where the transaction is in regards to contact status.	This is used to help us see where we are on the timeline of completing a transaction which can then be simplified into a binary variable.

	 0. Not known 1. Not contacted 2. Initial email sent 3. Follow-up email sent 4. Voicemail left 5. Right person 6. Has responded 7. Spoke on the phone 8. Plans to share data 9. Will not share data **10. Shared some data** 11. Shared all data	
Responded	This stores a binary variable that is 1 if the Contact_Status is a 10 or 11 and stores a 0 if the Contact_Status is from 2-9.	This is important because now we have simplified our understanding of where each transaction is and can do analysis more easily and efficiently.
Value_of_Interest	This stores the column of interest with a limited number of buckets to ease the graphing/mapping process.	This is important because this is the value we want to look at and how response rate varies for it.

(6) City Center Variation Use Case

See how different variables vary as the distance from a city center varies.

1. Identify a value of interest
    a. Ex) Acreage, Price per Acre, Response Rate
2. Load in or calculate the center of each transaction
3. Identify the closest city to each transaction (center)
4. Calculate the distance from the center of the transaction to the closest city
5. Graph the value of interest vs the distance from city center

Field	**What it Stores**	**Why it's Needed**

Value_of_Interest	This stores the column of interest for seeing how city center distance varies for it.	This is important because this is the value we want to look at and how city center distance varies for it.
Center_of_Transaction	This stores the longitude and latitude of the center of each transaction.	This is needed so that we have one point to calculate the distance to the city from.
City_Centers	This stores the longitude and latitude of the center of the closest city to the transaction.	This is needed so that we can calculate Distance_from_City.
Distance_from_City	This stores the distance in meters from the center of the closest city.	This is needed so that we can understand the ways that the Value_of_Interest varies with Distance_from_City.

Structural Database Rules

1. Each researcher may alter many transactions; each transaction may be altered by many researchers.

Both sides are optional plural as multiple researchers can be collaborating on all or a part of this project.

2. Each grantor may transfer many transactions; each transaction may be transferred by many grantors.

Transactions can be owned by many people and they can choose to keep these transactions or transfer them (but they don't have to transfer them). When the transaction is transferred it can be done by the many people (and is typically required to be agreed upon by all of them).

3. Each grantor works with one to many grantees; each grantee works with one to many grantors.

For a grantor to be granting something, there must be a grantee so it is not optional. Like before, multiple grantees and grantors can own or purchase a transaction.

4. Each grantee may purchase many transactions; each transaction may be purchased by many grantees.

Transactions can be owned by many people and each person can own many transactions. However, a person doesn't have to own a transaction.

5. Each researcher may input many validation data; each validation data may be input by one researcher.

Any researcher can input no or multiple validation data but if multiple researchers were to input the same validation datum, then we would have duplicate data and that would be an issue.

6. Each researcher may create many maps; each map may be created by many researchers.

As this is a collaborative project, the maps will be worked on by many people. However, every map that we think of doesn't have to be created. Also, many researchers can make more than one maps.

7. Each researcher may create many graphs; each graph may be created by many researchers.

As this is a collaborative project, the maps will be worked on by many people. However, every map that we think of doesn't have to be created. Also, many researchers can make more than one maps.

8. Each map may depict many transactions; each transaction may be depicted by multiple maps.

Most often we will be depicting more than one transaction. Not every transaction has to be depicted. Oftentimes the transactions will be on more than one of our maps.

9. Each graph may depict many transactions; each transaction may be depicted by multiple graphs.

Most often we will be depicting more than one transaction. Not every transaction has to be depicted. Oftentimes the transactions will be on more than one of our graphs.

10. Each researcher may advise one to many land conservation companies; each land conservation company may be advised by one to many land conservation companies.

The company may have a lot of advising that needs to be done and so they need more than one researcher. Some companies may be small so the researcher can advise more than one company.

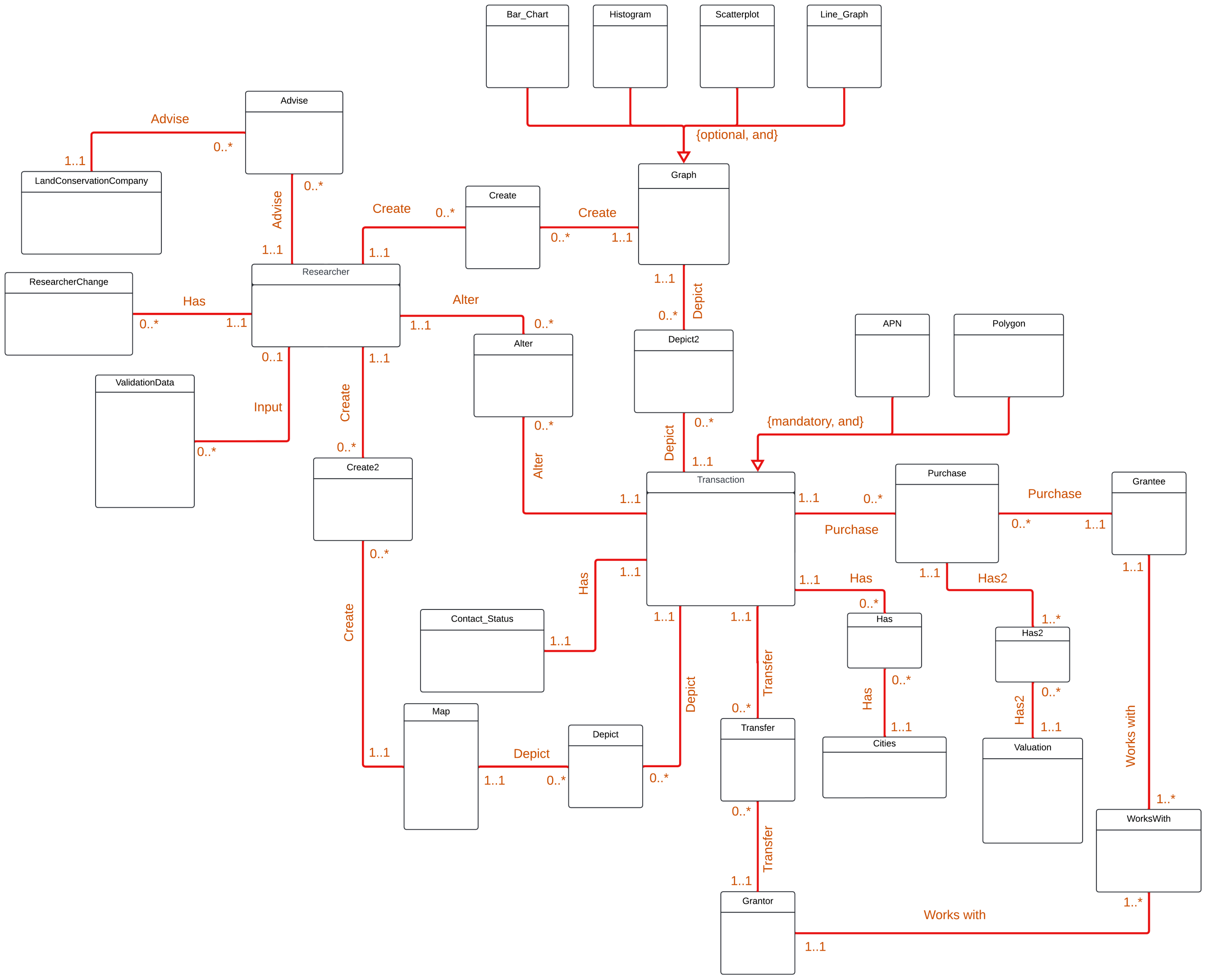
11. A transaction is described by either a ct_id, APN, polygon, or multiple of these.

If we don't have any of these we can't identify the transactions so it has to have at least one. Having multiple makes it easier to identify the transaction and can often get the others from each.

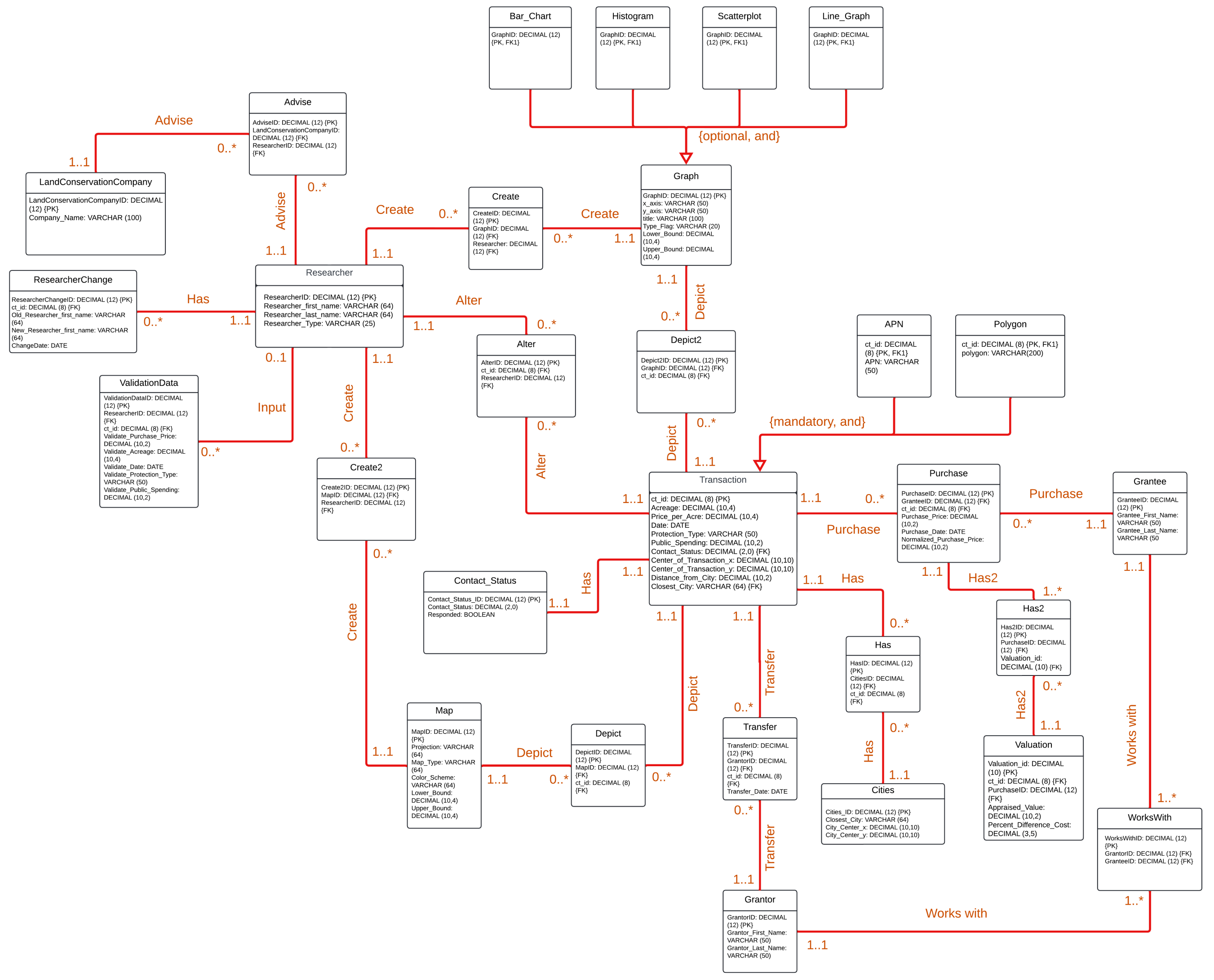
12. A graph is either a bar chart, a histogram, a scatterplot, a line graph, multiple of these, or none of these.

It's possible to have multiple of these elements on one graph (however I rarely see this). There are also graph types that are less common that could be used.

Conceptual Entity-Relationship Diagram



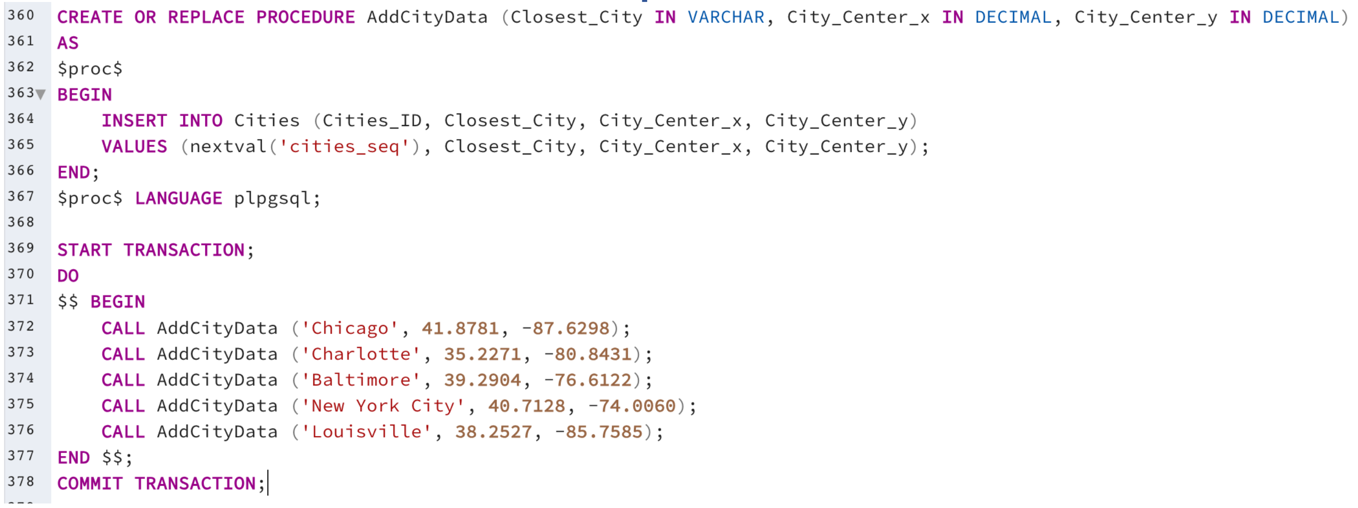
Full DBMS Physical ERD


Bar_Chart
GraphID: DECIMAL (12) {PK, FK1}
Histogram
GraphID: DECIMAL (12) {PK, FK1}
Scatterplot
GraphID: DECIMAL (12) {PK, FK1}
Line_Graph
GraphID: DECIMAL (12) {PK, FK1}
{optional, and}
Graph
GraphID: DECIMAL (12) {PK}
x_axis: VARCHAR (50)
y_axis: VARCHAR (50)
title: VARCHAR (100)
Type_Flag: VARCHAR (20)
Lower_Bound: DECIMAL (10,4)
Upper_Bound: DECIMAL (10,4)
Advise
AdviseID: DECIMAL (12) {PK}
LandConservationCompanyID: DECIMAL (12) {FK}
ResearcherID: DECIMAL (12) {FK}
LandConservationCompany
LandConservationCompanyID: DECIMAL (12) {PK}
Company_Name: VARCHAR (100)
Create
CreateID: DECIMAL (12) {PK}
GraphID: DECIMAL (12) {FK}
Researcher: DECIMAL (12) {FK}
Researcher
ResearcherID: DECIMAL (12) {PK}
Researcher_first_name: VARCHAR (64)
Researcher_last_name: VARCHAR (64)
Researcher_Type: VARCHAR (25)
ResearcherChange
ResearcherChangeID: DECIMAL (12) {PK}
ct_id: DECIMAL (8) {FK}
Old_Researcher_first_name: VARCHAR (64)
New_Researcher_first_name: VARCHAR (64)
ChangeDate: DATE
ValidationData
ValidationDataID: DECIMAL (12) {PK}
ResearcherID: DECIMAL (12) {FK}
ct_id: DECIMAL (8) {FK}
Validate_Purchase_Price: DECIMAL (10,2)
Validate_Acreage: DECIMAL (10,4)
Validate_Date: DATE
Validate_Protection_Type: VARCHAR (50)
Validate_Public_Spending: DECIMAL (10,2)
Alter
AlterID: DECIMAL (12) {PK}
ct_id: DECIMAL (8) {FK}
ResearcherID: DECIMAL (12) {FK}
Depict2
Depict2ID: DECIMAL (12) {PK}
GraphID: DECIMAL (12) {FK}
ct_id: DECIMAL (8) {FK}
APN
ct_id: DECIMAL (8) {PK, FK1}
APN: VARCHAR (50)
Polygon
ct_id: DECIMAL (8) {PK, FK1}
polygon: VARCHAR(200)
{mandatory, and}
Create2
Create2ID: DECIMAL (12) {PK}
MapID: DECIMAL (12) {FK}
ResearcherID: DECIMAL (12) {FK}
Transaction
ct_id: DECIMAL (8) {PK}
Acreage: DECIMAL (10,4)
Price_per_Acre: DECIMAL (10,4)
Date: DATE
Protection_Type: VARCHAR (50)
Public_Spending: DECIMAL (10,2)
Contact_Status: DECIMAL (2,0) {FK}
Center_of_Transaction_x: DECIMAL (10,10)
Center_of_Transaction_y: DECIMAL (10,10)
Distance_from_City: DECIMAL (10,2)
Closest_City: VARCHAR (64) {FK}
Purchase
PurchaseID: DECIMAL (12) {PK}
GranteeID: DECIMAL (12) {FK}
ct_id: DECIMAL (8) {FK}
Purchase_Price: DECIMAL (10,2)
Purchase_Date: DATE
Normalized_Purchase_Price: DECIMAL (10,2)
Grantee
GranteeID: DECIMAL (12) {PK}
Grantee_First_Name: VARCHAR (50)
Grantee_Last_Name: VARCHAR (50
Contact_Status
Contact_Status_ID: DECIMAL (12) {PK}
Contact_Status: DECIMAL (2,0)
Responded: BOOLEAN
Has
HasID: DECIMAL (12) {PK}
CitiesID: DECIMAL (12) {FK}
ct_id: DECIMAL (8) {FK}
Has2
Has2ID: DECIMAL (12) {PK}
PurchaseID: DECIMAL (12) {FK}
Valuation_id: DECIMAL (10) {FK}
Map
MapID: DECIMAL (12) {PK}
Projection: VARCHAR (64)
Map_Type: VARCHAR (64)
Color_Scheme: VARCHAR (64)
Lower_Bound: DECIMAL (10,4)
Upper_Bound: DECIMAL (10,4)
Depict
DepictID: DECIMAL (12) {PK}
MapID: DECIMAL (12) {FK}
ct_id: DECIMAL (8) {FK}
Transfer
TransferID: DECIMAL (12) {PK}
GrantorID: DECIMAL (12) {FK}
ct_id: DECIMAL (8) {FK}
Transfer_Date: DATE
Cities
Cities_ID: DECIMAL (12) {PK}
Closest_City: VARCHAR (64)
City_Center_x: DECIMAL (10,10)
City_Center_y: DECIMAL (10,10)
Valuation
Valuation_id: DECIMAL (10) {PK}
ct_id: DECIMAL (8) {FK}
PurchaseID: DECIMAL (12) {FK}
Appraised_Value: DECIMAL (10,2)
Percent_Difference_Cost: DECIMAL (3,5)
WorksWith
WorksWithID: DECIMAL (12) {PK}
GrantorID: DECIMAL (12) {FK}
GranteeID: DECIMAL (12) {FK}
Grantor
GrantorID: DECIMAL (12) {PK}
Grantor_First_Name: VARCHAR (50)
Grantor_Last_Name: VARCHAR (50)
Advise
Create
Has
Alter
Input
Depict
Purchase
Has2
Transfer
Works with
1..1
0..*
0..1
1..*

This follows 1NF as each transaction has a unique id so we don't need to rely on row order for any information, everything has its own datatype with no mixing, each table has a primary key, and there is no way for repeating groups. The lack of repeating groups was fixed by having everything stem from the ct_id which is completely unique (hard coded that way).

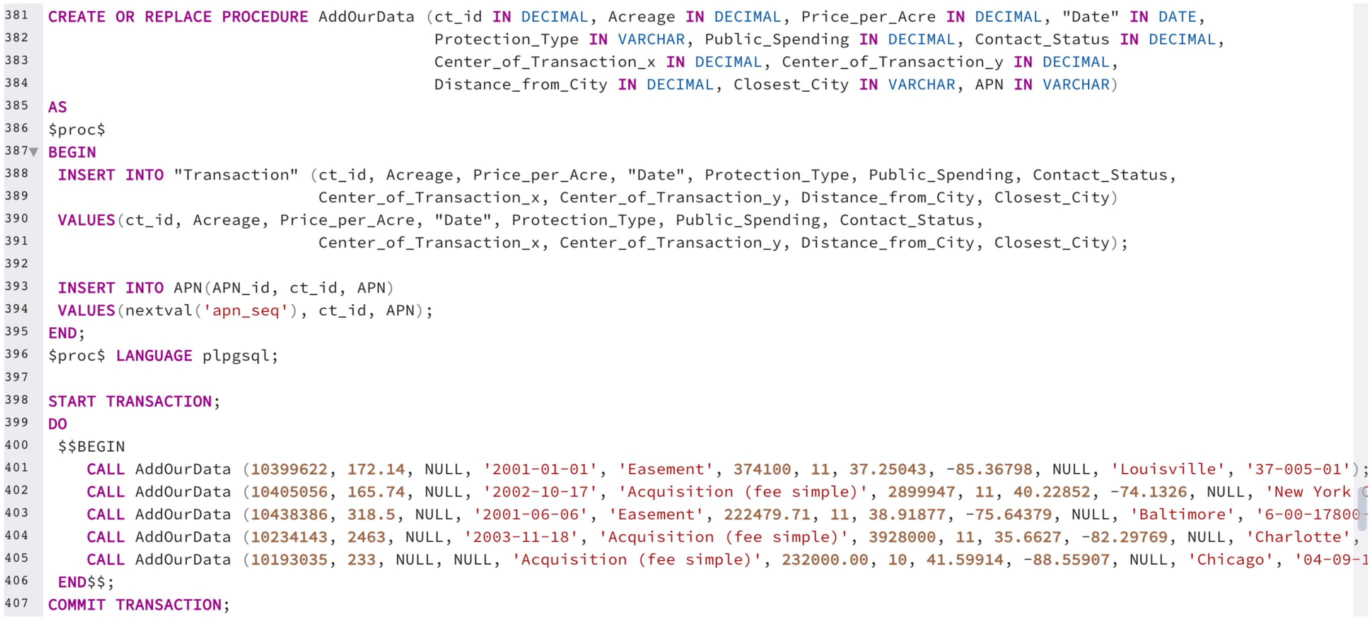
This follows 2NF because each non-key attribute in the table is dependent on the entire primary key. This follows 3NF/BCNF because each attribute in the table is dependent on the key, the whole key, and nothing but the key. These were achieved by splitting some of the attributes of the transaction table into side tables that reference attributes of transaction.

Stored Procedure Execution and Explanations



```
CREATE OR REPLACE PROCEDURE AddCityData (Closest_City IN VARCHAR, City_Center_x IN DECIMAL, City_Center_y IN DECIMAL)
AS
$proc$
BEGIN
    INSERT INTO Cities (Cities_ID, Closest_City, City_Center_x, City_Center_y)
    VALUES (nextval('cities_seq'), Closest_City, City_Center_x, City_Center_y);
END;
$proc$ LANGUAGE plpgsql;

START TRANSACTION;
DO
$$ BEGIN
    CALL AddCityData ('Chicago', 41.8781, -87.6298);
    CALL AddCityData ('Charlotte', 35.2271, -80.8431);
    CALL AddCityData ('Baltimore', 39.2904, -76.6122);
    CALL AddCityData ('New York City', 40.7128, -74.0060);
    CALL AddCityData ('Louisville', 38.2527, -85.7585);
END $$;
COMMIT TRANSACTION;
```

This adds city data manually while utilizing the cities_seq. It takes the name of the city and the lat/long coordinates of the city.

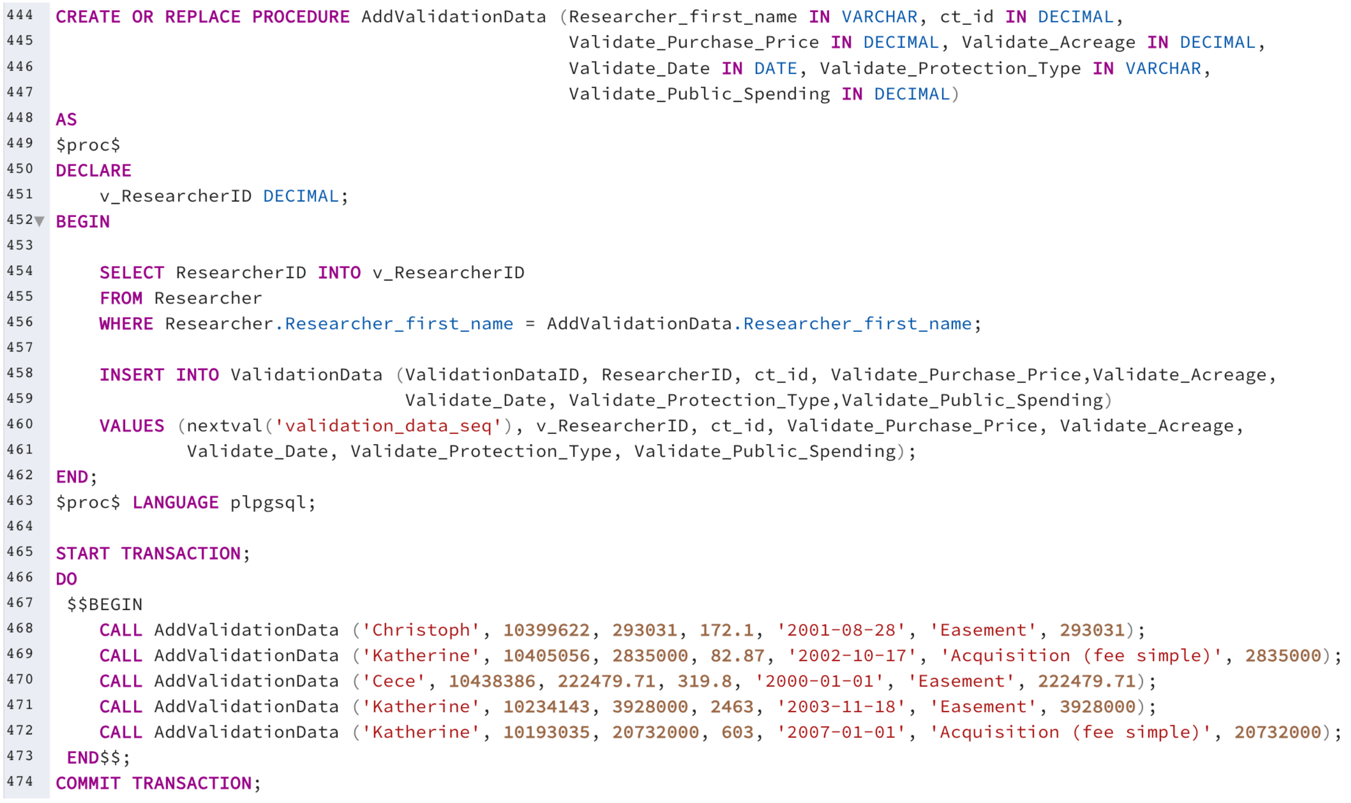


```
CREATE OR REPLACE PROCEDURE AddOurData (ct_id IN DECIMAL, Acreage IN DECIMAL, Price_per_Acre IN DECIMAL, "Date" IN DATE,
                                        Protection_Type IN VARCHAR, Public_Spending IN DECIMAL, Contact_Status IN DECIMAL,
                                        Center_of_Transaction_x IN DECIMAL, Center_of_Transaction_y IN DECIMAL,
                                        Distance_from_City IN DECIMAL, Closest_City IN VARCHAR, APN IN VARCHAR)
AS
$proc$
BEGIN
 INSERT INTO "Transaction" (ct_id, Acreage, Price_per_Acre, "Date", Protection_Type, Public_Spending, Contact_Status,
                           Center_of_Transaction_x, Center_of_Transaction_y, Distance_from_City, Closest_City)
 VALUES(ct_id, Acreage, Price_per_Acre, "Date", Protection_Type, Public_Spending, Contact_Status,
                           Center_of_Transaction_x, Center_of_Transaction_y, Distance_from_City, Closest_City);

 INSERT INTO APN(APN_id, ct_id, APN)
 VALUES(nextval('apn_seq'), ct_id, APN);
END;
$proc$ LANGUAGE plpgsql;

START TRANSACTION;
DO
 $$BEGIN
    CALL AddOurData (10399622, 172.14, NULL, '2001-01-01', 'Easement', 374100, 11, 37.25043, -85.36798, NULL, 'Louisville', '37-005-01');
    CALL AddOurData (10405056, 165.74, NULL, '2002-10-17', 'Acquisition (fee simple)', 2899947, 11, 40.22852, -74.1326, NULL, 'New York C
    CALL AddOurData (10438386, 318.5, NULL, '2001-06-06', 'Easement', 222479.71, 11, 38.91877, -75.64379, NULL, 'Baltimore', '6-00-17800-
    CALL AddOurData (10234143, 2463, NULL, '2003-11-18', 'Acquisition (fee simple)', 3928000, 11, 35.6627, -82.29769, NULL, 'Charlotte',
    CALL AddOurData (10193035, 233, NULL, NULL, 'Acquisition (fee simple)', 232000.00, 10, 41.59914, -88.55907, NULL, 'Chicago', '04-09-1
 END$$;
COMMIT TRANSACTION;
```

This adds our researched data manually. There is only the ability to add one APN at the moment but more can be added on later on. The APN is not required but it can be added (same with the city).

```
INSERT INTO Researcher (ResearcherID, Researcher_first_name, Researcher_last_name, Researcher_Type)
VALUES (nextval('researcher_seq'),'Katherine','Rein','Undergraduate'),
       (nextval('researcher_seq'),'Julianne', NULL, 'Undergraduate'),
       (nextval('researcher_seq'),'Cece', NULL, 'Undergraduate'),
       (nextval('researcher_seq'),'Ella', NULL, 'Undergraduate'),
       (nextval('researcher_seq'),'Christoph','Nolte','PI');

CREATE OR REPLACE PROCEDURE AddResearcherToTransaction (Researcher_first_name IN VARCHAR, ct_id IN DECIMAL)
AS
$proc$
DECLARE
    v_ResearcherID DECIMAL;
BEGIN

    SELECT ResearcherID INTO v_ResearcherID
    FROM Researcher
    WHERE Researcher.Researcher_first_name = AddResearcherToTransaction.Researcher_first_name;

    INSERT INTO "Alter" (AlterID, ct_id, ResearcherID)
    VALUES(nextval('alter_seq'), ct_id, v_ResearcherID);
END;
$proc$ LANGUAGE plpgsql;

START TRANSACTION;
DO
 $$BEGIN
    CALL AddResearcherToTransaction ('Christoph', 10399622);
    CALL AddResearcherToTransaction ('Katherine', 10405056);
    CALL AddResearcherToTransaction ('Cece', 10438386);
    CALL AddResearcherToTransaction ('Katherine', 10234143);
    CALL AddResearcherToTransaction ('Katherine', 10193035);
 END$$;
COMMIT TRANSACTION;
```

This correlates a researcher to a transaction. This could be done within the addourdata procedure but that felt like too much information to require.



```sql
CREATE OR REPLACE PROCEDURE AddValidationData (Researcher_first_name IN VARCHAR, ct_id IN DECIMAL,
                                               Validate_Purchase_Price IN DECIMAL, Validate_Acreage IN DECIMAL,
                                               Validate_Date IN DATE, Validate_Protection_Type IN VARCHAR,
                                               Validate_Public_Spending IN DECIMAL)
AS
$proc$
DECLARE
    v_ResearcherID DECIMAL;
BEGIN

    SELECT ResearcherID INTO v_ResearcherID
    FROM Researcher
    WHERE Researcher.Researcher_first_name = AddValidationData.Researcher_first_name;

    INSERT INTO ValidationData (ValidationDataID, ResearcherID, ct_id, Validate_Purchase_Price,Validate_Acreage,
                                Validate_Date, Validate_Protection_Type,Validate_Public_Spending)
    VALUES (nextval('validation_data_seq'), v_ResearcherID, ct_id, Validate_Purchase_Price, Validate_Acreage,
            Validate_Date, Validate_Protection_Type, Validate_Public_Spending);
END;
$proc$ LANGUAGE plpgsql;

START TRANSACTION;
DO
 $$BEGIN
    CALL AddValidationData ('Christoph', 10399622, 293031, 172.1, '2001-08-28', 'Easement', 293031);
    CALL AddValidationData ('Katherine', 10405056, 2835000, 82.87, '2002-10-17', 'Acquisition (fee simple)', 2835000);
    CALL AddValidationData ('Cece', 10438386, 222479.71, 319.8, '2000-01-01', 'Easement', 222479.71);
    CALL AddValidationData ('Katherine', 10234143, 3928000, 2463, '2003-11-18', 'Easement', 3928000);
    CALL AddValidationData ('Katherine', 10193035, 20732000, 603, '2007-01-01', 'Acquisition (fee simple)', 20732000);
 END$$;
COMMIT TRANSACTION;
```

This is the data that we are wanting to check the accuracy of. We are able to add in the all of the data and the researcher here.

Question Identification and Explanations

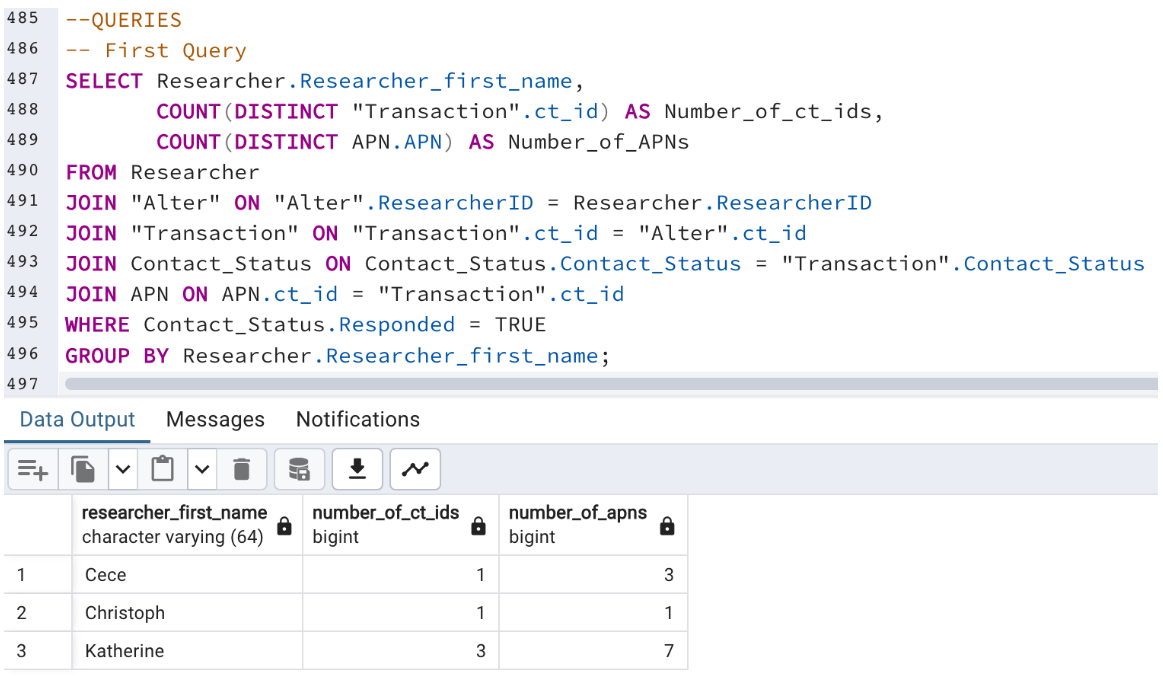
First Query: How many transactions and APNs has each researcher inputted that have at least a response?

Second Query: How many transactions have multiple APNs?

Third Query: Which transactions have over $200,000 of public spending?

Query Executions and Explanations

First Query:

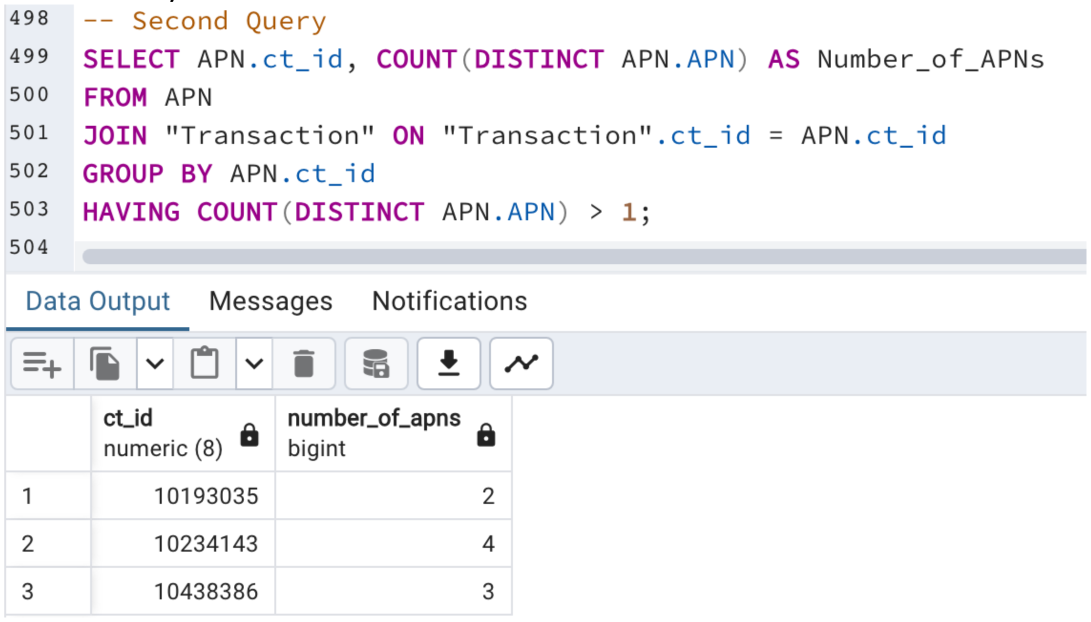


```
--QUERIES
-- First Query
SELECT Researcher.Researcher_first_name,
       COUNT(DISTINCT "Transaction".ct_id) AS Number_of_ct_ids,
       COUNT(DISTINCT APN.APN) AS Number_of_APNs
FROM Researcher
JOIN "Alter" ON "Alter".ResearcherID = Researcher.ResearcherID
JOIN "Transaction" ON "Transaction".ct_id = "Alter".ct_id
JOIN Contact_Status ON Contact_Status.Contact_Status = "Transaction".Contact_Status
JOIN APN ON APN.ct_id = "Transaction".ct_id
WHERE Contact_Status.Responded = TRUE
GROUP BY Researcher.Researcher_first_name;
```

	researcher_first_name character varying (64)	number_of_ct_ids bigint	number_of_apns bigint
1	Cece	1	3
2	Christoph	1	1
3	Katherine	3	7

This counts the number of APNs and ct_ids where the responded column is true. It then groups these all by the Researchers first name so that we can see each researchers progress.

Second Query:

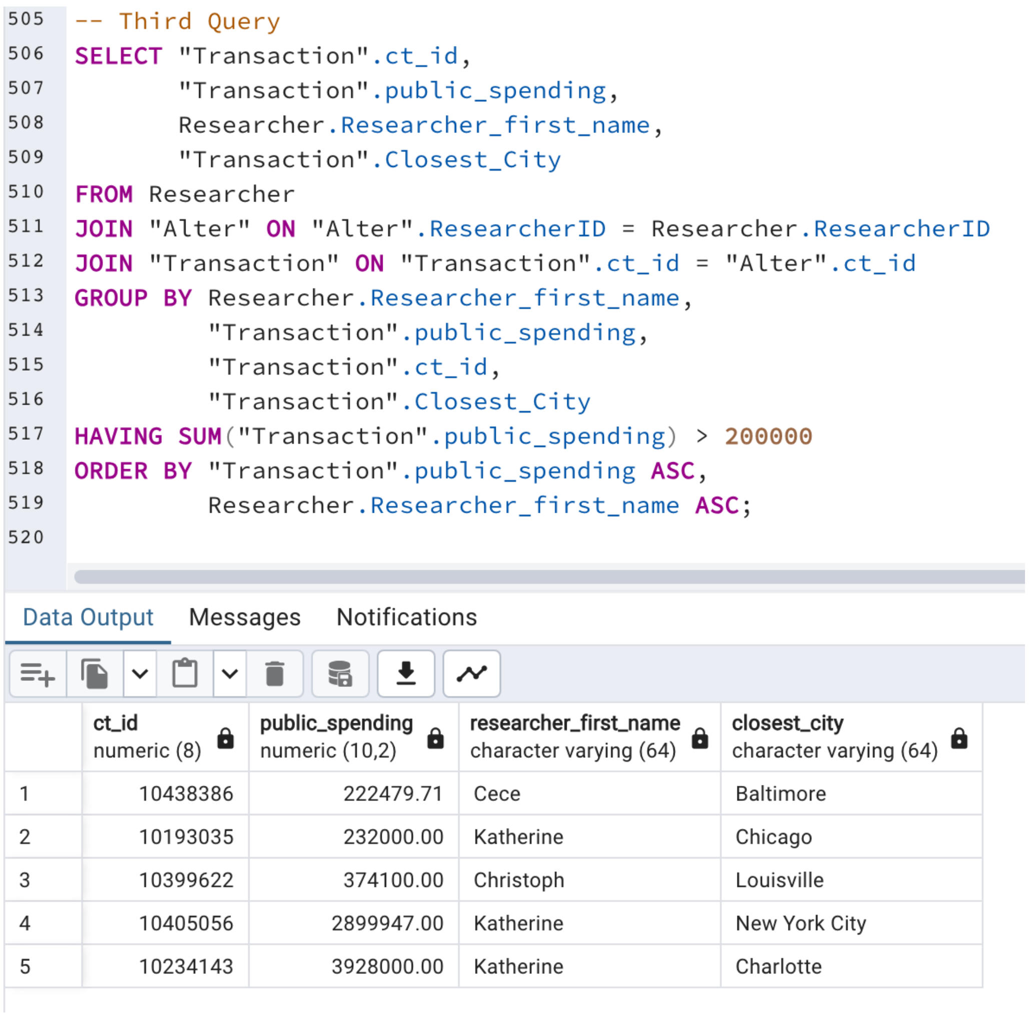


```
-- Second Query
SELECT APN.ct_id, COUNT(DISTINCT APN.APN) AS Number_of_APNs
FROM APN
JOIN "Transaction" ON "Transaction".ct_id = APN.ct_id
GROUP BY APN.ct_id
HAVING COUNT(DISTINCT APN.APN) > 1;
```

	ct_id numeric (8)	number_of_apns bigint
1	10193035	2
2	10234143	4
3	10438386	3

This tells us which ct_ids have multiple and how many APNs each has. It does so utilizing a having clause.

Third Query:



```
-- Third Query
SELECT "Transaction".ct_id,
       "Transaction".public_spending,
       Researcher.Researcher_first_name,
       "Transaction".Closest_City
FROM Researcher
JOIN "Alter" ON "Alter".ResearcherID = Researcher.ResearcherID
JOIN "Transaction" ON "Transaction".ct_id = "Alter".ct_id
GROUP BY Researcher.Researcher_first_name,
         "Transaction".public_spending,
         "Transaction".ct_id,
         "Transaction".Closest_City
HAVING SUM("Transaction".public_spending) > 200000
ORDER BY "Transaction".public_spending ASC,
         Researcher.Researcher_first_name ASC;
```

	ct_id numeric (8)	public_spending numeric (10,2)	researcher_first_name character varying (64)	closest_city character varying (64)
1	10438386	222479.71	Cece	Baltimore
2	10193035	232000.00	Katherine	Chicago
3	10399622	374100.00	Christoph	Louisville
4	10405056	2899947.00	Katherine	New York City
5	10234143	3928000.00	Katherine	Charlotte

This looks for which ct_ids have a public spending of more than $200,000. It then orders the data by spending and researcher first name.

Index Identification and Creations

Primary Keys:
LandConservationCompany.LandConservationCompanyID
Researcher.ResearcherID
Contact_Status.Contact_Status_ID
Cities.Cities_ID
Transaction.ct_id
Map.MapID
Graph.GraphID
Grantee.GranteeID
Grantor.GrantorID
Advise.AdviseID
ValidationData.ValidationDataID
Create2.Create2ID
Depict.DepictID
Alter.AlterID

Depict2.Depict2ID
Create.CreateID
Bar_Chart.GraphID
Histogram.GraphID
Scatterplot.GraphID
Line_Graph.GraphID
APN.APN_id
Polygon.ct_id
Purchase.PurchaseID
Valuation.Valuation_id
WorksWith.WorksWithID
Transfer.TransferID

Foreign Keys:

Column	Unique?	Description
Transaction.Contact_Status	Not unique	The foreign key in Transaction referencing Contact_Status. Not unique because multiple transactions can have the same contact status.
Transaction.Closest_City	Not unique	The foreign key in Transaction referencing Cities. Not unique because multiple transactions can be associated with the same city.
Advise.LandConservationCompanyID	Not unique	The foreign key in Advise referencing LandConservationCompany. Not unique because multiple advises can be related to the same company.
Advise.ResearcherID	Not unique	The foreign key in Advise referencing Researcher. Not unique because multiple advises can be associated with the same researcher.
ValidationData.ResearcherID	Not unique	The foreign key in ValidationData referencing Researcher. Not unique because multiple validation data entries can be associated with the same researcher.
ValidationData.ct_id	Unique	The foreign key in ValidationData referencing Transaction. Unique because only one validation data entry can be related to the same transaction.
Create2.MapID	Not unique	The foreign key in Create2 referencing Map. Not unique because multiple create entries can be associated with the same map.
Create2.ResearcherID	Not unique	The foreign key in Create2 referencing Researcher. Not unique because multiple create entries can be associated with the same researcher.

Depict.MapID	Not unique	The foreign key in Depict referencing Map. Not unique because multiple depicts can be associated with the same map.
Depict.ct_id	Not unique	The foreign key in Depict referencing Transaction. Not unique because multiple depicts can be related to the same transaction.
Alter.ct_id	Not unique	The foreign key in Alter referencing Transaction. Not unique because multiple alterations can be related to the same transaction.
Alter.ResearcherID	Not unique	The foreign key in Alter referencing Researcher. Not unique because multiple alterations can be associated with the same researcher.
Depict2.GraphID	Not unique	The foreign key in Depict2 referencing Graph. Not unique because multiple depict2 entries can be associated with the same graph.
Depict2.ct_id	Not unique	The foreign key in Depict2 referencing Transaction. Not unique because multiple depict2 entries can be related to the same transaction.
Create.GraphID	Not unique	The foreign key in Create referencing Graph. Not unique because multiple create entries can be associated with the same graph.
Create.ResearcherID	Not unique	The foreign key in Create referencing Researcher. Not unique because multiple create entries can be associated with the same researcher.
Bar_Chart.GraphID	Unique	The foreign key in Bar_Chart referencing Graph. Unique because each bar chart is associated with a specific graph.
Histogram.GraphID	Unique	The foreign key in Histogram referencing Graph. Unique because each histogram is associated with a specific graph.
Scatterplot.GraphID	Unique	The foreign key in Scatterplot referencing Graph. Unique because each scatterplot is associated with a specific graph.
Line_Graph.GraphID	Unique	The foreign key in Line_Graph referencing Graph. Unique because each line graph is associated with a specific graph.
APN.ct_id	Not unique	The foreign key in APN referencing Transaction. Not unique because multiple APNs can be related to the same transaction.
Polygon.ct_id	Unique	The foreign key in Polygon referencing Transaction. Unique because each polygon is associated with a specific transaction.
Purchase.GranteeID	Not unique	The foreign key in Purchase referencing Grantee. Not unique because multiple purchases can be associated with the same grantee.

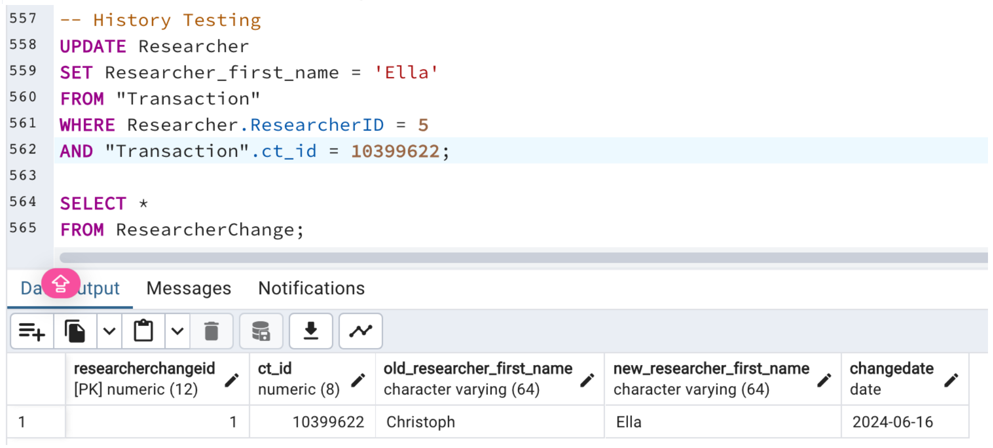
Purchase.ct_id	Unique	The foreign key in Purchase referencing Transaction. Unique because only one purchases can be related to the same transaction.
Valuation.ct_id	Not unique	The foreign key in Valuation referencing Transaction. Not unique because multiple valuations can be related to the same transaction.
Valuation.PurchaseID	Not unique	The foreign key in Valuation referencing Purchase. Not unique because multiple valuations can be related to the same purchase.
WorksWith.GrantorID	Not unique	The foreign key in WorksWith referencing Grantor. Not unique because multiple works with entries can be associated with the same grantor.
WorksWith.GranteeID	Not unique	The foreign key in WorksWith referencing Grantee. Not unique because multiple works with entries can be associated with the same grantee.
Transfer.GrantorID	Not unique	The foreign key in Transfer referencing Grantor. Not unique because multiple transfers can be associated with the same grantor.
Transfer.ct_id	Not unique	The foreign key in Transfer referencing Transaction. Not unique because multiple transfers can be related to the same transaction.

Query:
"Transaction".public_spending: This could be useful when trying to identify certain information about trasactions that are in a certain price range.
“Transaction".Acreage: This could be useful when trying to identify certain information about trasactions that are of a certain size.
"Transaction"."Date": This could be useful when trying to identify certain information about trasactions that are in a certain time frame.

History Table Demonstration

The ResearcherChange table is used if we need to reallocate some of the transactions to a different researcher. This occurs when certain researchers leave, can work more hours, or need to switch up what they are focusing on.

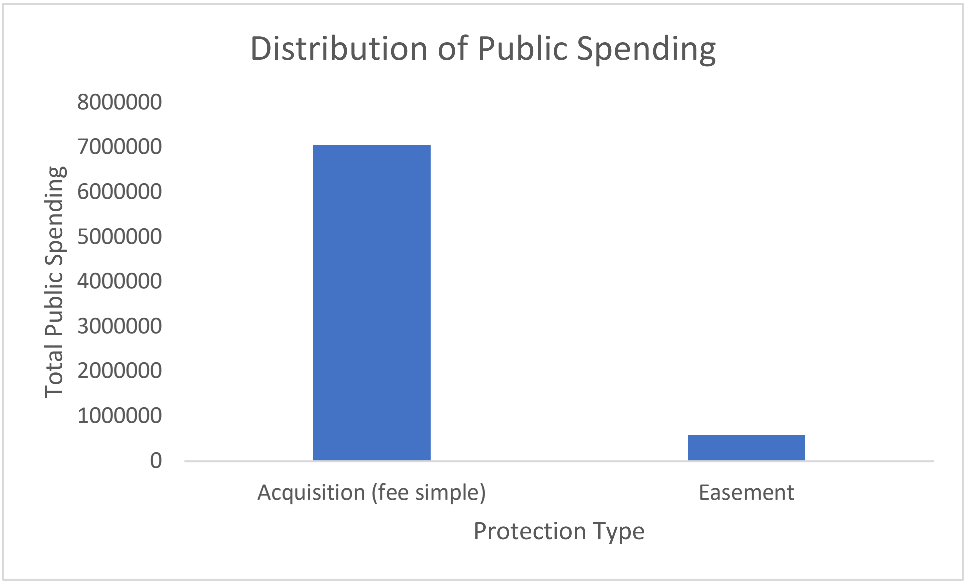
```
CREATE OR REPLACE FUNCTION ResearcherChangeFunction()
RETURNS TRIGGER LANGUAGE plpgsql
AS $trigfunc$
BEGIN
    INSERT INTO ResearcherChange(ResearcherChangeID, ct_id, Old_Researcher_first_name, New_Researcher_first_name, ChangeDate)
    SELECT nextval('researcher_change_seq'),
           alt.ct_id,
           OLD.Researcher_first_name,
           NEW.Researcher_first_name,
           current_date
    FROM "Alter" alt
    WHERE alt.ResearcherID = OLD.ResearcherID;

    RETURN NEW;
END;
$trigfunc$;

CREATE TRIGGER ResearcherChangeTrigger
BEFORE UPDATE OF Researcher_first_name ON Researcher
FOR EACH ROW
EXECUTE PROCEDURE ResearcherChangeFunction();
```



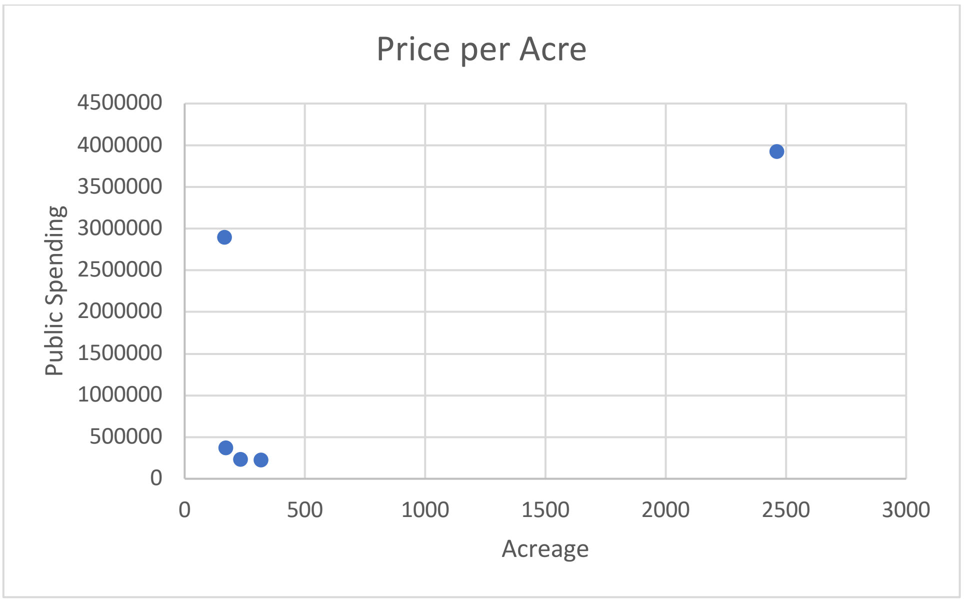
```
-- History Testing
UPDATE Researcher
SET Researcher_first_name = 'Ella'
FROM "Transaction"
WHERE Researcher.ResearcherID = 5
AND "Transaction".ct_id = 10399622;

SELECT *
FROM ResearcherChange;
```

	researcherchangeid [PK] numeric (12)	ct_id numeric (8)	old_researcher_first_name character varying (64)	new_researcher_first_name character varying (64)	changedate date
1	1	10399622	Christoph	Ella	2024-06-16

Data Visualizations



To determine how companies and governments tend to spend their money when it comes to conservation work. This will help us understand where most of the money already goes when we want to talk to people about increasing funds.



When trying to predict how much a transaction might cost, it could be nice to be able to look at the trends of transactions of different sizes. Looking at the history of how much transactions cost would benefit our predictions.

Summary and Reflection

This database will be used to merge, visualize, compute, and analyze the data that is collected in the PLACES Lab at Boston University more efficiently. It will need to be able to merge and clean data very quickly and efficiently as that is the most common fault of working with data inputted by humans. Public data is notoriously dirty. There will also be a lot of working with GIS or spatial data. While possible to have use cases without a spatial component to the database, it would be nice to be able to include that.

I don’t have any new questions at the moment.