2021

Big Data Architecture & Governance

Northeastern University

Assignment Name:

Class Assignment

1. Contents

2. Assignment

2.1. Metadata

Data export Procedure from Neo4j:

INTRODUCTION/ PLANNING:

- Designed a fully automated process to extract metadata and push to a common location that could be accessed by the data ingestion team.
- Motivation behind using Google Sheets was to avoid back and forth between individual teams and the data ingestion teams.
- If a stand-alone script was used – every team would have to share the data manually with the data ingestion team. This process is eliminated by having the Python process that writes to a Google Sheet.
- If the script is re-run, the process overwrites the old data pushed by a team – effectively ensuring that only one copy of metadata/team is maintained.
- The main steps of the program include setting dB configuration in the python script.
- The python script first creates a Database Session with Neo4j. We use Awesome Procedures on Cypher (APOC) to extract the nodes, labels etc.
- APOC is an add-on library for Neo4j that provides hundreds of procedures and functions adding a lot of useful functionality.
- All the metadata is dynamically written to this Google Sheet and can be accessed here: https://docs.google.com/spreadsheets/d/1rTQeW2DbqpxNK0CBypdsx4ZgK3Kz0E8xk2FgXo37HqQ/edit#gid=61069745
-

LIBRARIES USED:

- Py2neo: connection to neo4j
- Pygsheets: to connect to Google Sheets and write data.
- Pandas – for data wrangling

INPUT PARAMETERS

The Python script takes in the following input parameters:

- Database Name
- Database username
- Database password
- Database URI

- Team Number: Used to write on the appropriate sheet.

Steps to run:

- Install Python3.7+ ; Download Python from here: https://www.python.org/downloads/release/python-370/
- Open a terminal and run - pip install --user pandas py2neo pygsheets
  This installs all the required libraries for the data extraction process to extract the metadata from Neo4j
- In script modify “username”, “pwd” and “team” with <your db username>, <your db password>, <team number> respectively.
- Run the process by executing Python3 extract_metadata.py

Explanation:

The code has 2 main functions: getData() and write2gsheets().

- getData(): This function is responsible for scraping the metadata from neo4j and
- write2gsheets() is used to push the dataframe to Google Sheets.

getData():

- The script tries to create a DB Session.
- It executes an APOC call which will scrape all the label, property, type etc (everything except the relationship)
- The data returned is written in a panda data frame.
- For every unique node label, all the relationships are extracted from the Neo4j database and appended to the dataframe.
- This dataframe returned is passed to write2gsheets() which will use the service_account.json to authenticate the request and write the dataframe on Google Sheets.

write2gsheets():

- The function receives the dataframe containing the metadata.
- To avoid un-authenticated users from programmatically writing to Google Sheet – the process uses the service_account.json file to authenticate access to the sheet.
- The function takes the team number as an input and writes to the corresponding sheet.

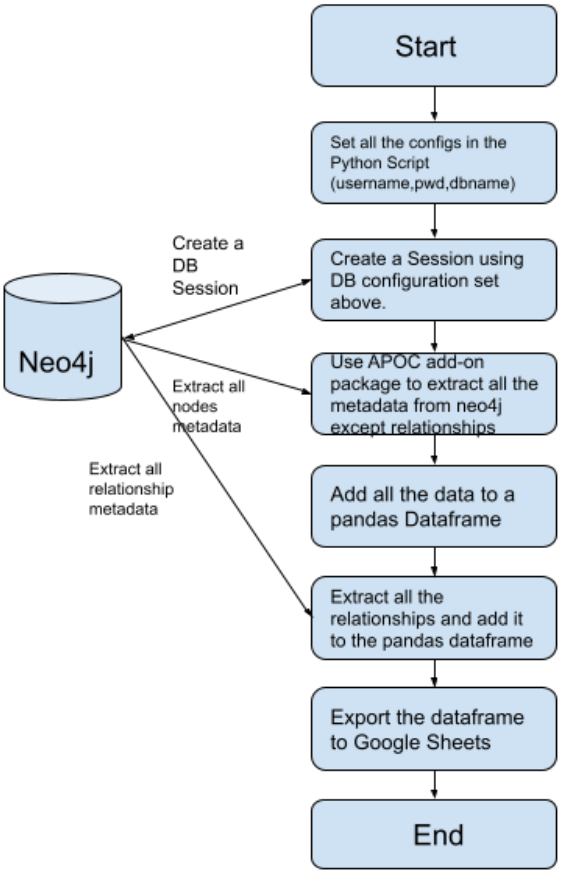
METADATA BEING EXTRACTED:

Field	Description
counts	Count of all nodes
label	Node label
property	Node property
type	Datatype of the property
isIndexed	if the data is indexed or not?
uniqueConstraint	If the property contains a unique value or not?
existenceConstraint	If the property exists for all nodes with a specific label?
team	Team number for easy identification
dbName	Database name
relationships	Relationships with the node

PROCESS FLOW:

- Connect to Neo4j using the connection params supplied in the script.
- Run a APOC query to extract the list of all nodes labels, properties, counts and other metadata.
- Append the results of the query to a panda's data frame.
- For each given node label, extract all relationships and append to the data frame.
- Connect to Google Sheets - authenticate using the service_account.json file. This file allows anyone with the python script to write to Google sheet programmatically.
- Write the extracted metadata to the team's Google Sheet.

DATA EXTRACTION PROCEDURE USING Neo4j FLOWCHART



QUERIES:

Query to extract all metadata:

```
CALL apoc.meta.schema() YIELD value as schemaMap
UNWIND keys(schemaMap) as label
WITH label, schemaMap[label] as data
WHERE data.type = "node"
UNWIND keys(data.properties) as property
WITH label, property, data.properties[property] as propData
RETURN label,
property,
propData.type as type,
propData.indexed as isIndexed,
propData.unique as uniqueConstraint,
propData.existence as existenceConstraint
```

Query to extract all counts of nodes:

```
MATCH (n)
RETURN DISTINCT count(labels(n)) as counts, labels(n) as label;
```

Query to extract all relationships:

```
MATCH (p1:%s)
RETURN apoc.node.relationship.types(p1) AS output;
```

Each node label is dynamically passed to this query to return the list of all relationships.

CODE WALKTHROUGH:

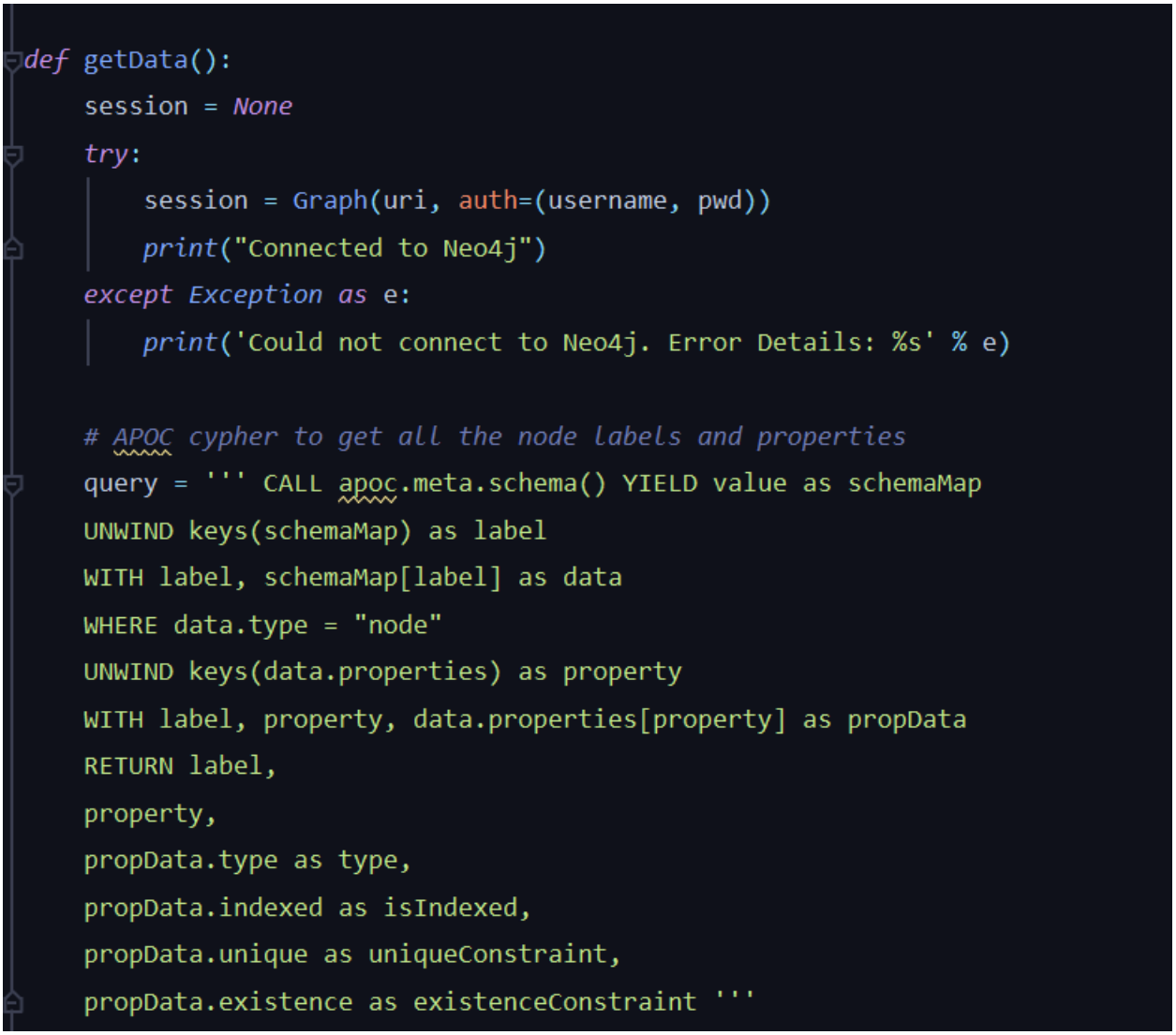
User parameters such as database name, password, uri is given by indivual teams:

```
# Connect to Neo4j - enter your connection params here
uri = "bolt://localhost:7687"
username = "neo4j"
pwd = "123456"

# Enter your team number here - so that the data can be pushed to the appropriate sheet
# Default is 0 - which writes to the Demo Sheet
team = 0

# Enter your DB Name
db_name = 'Movies Database'
```

Execute the APOC query to get all the metadata:



```
def getData():
    session = None
    try:
        session = Graph(uri, auth=(username, pwd))
        print("Connected to Neo4j")
    except Exception as e:
        print('Could not connect to Neo4j. Error Details: %s' % e)

    # APOC cypher to get all the node labels and properties
    query = ''' CALL apoc.meta.schema() YIELD value as schemaMap
    UNWIND keys(schemaMap) as label
    WITH label, schemaMap[label] as data
    WHERE data.type = "node"
    UNWIND keys(data.properties) as property
    WITH label, property, data.properties[property] as propData
    RETURN label,
    property,
    propData.type as type,
    propData.indexed as isIndexed,
    propData.unique as uniqueConstraint,
    propData.existence as existenceConstraint '''
```

Append the results from Neo4j into a dataframe and get the counts for all the node labels and append to the dataframe:

```
    # access the result data
    result = session.run(query).data()

    # convert result into pandas dataframe
    df = DataFrame(result)

    # Get the counts for all Node Labels
    countsQuery = '''
    MATCH (n)
    RETURN DISTINCT count(labels(n)) as counts, labels(n) as label;
    '''
    # access the result data
    result = session.run(countsQuery).data()
    df_counts = DataFrame(result)
    df_counts['label'] = df_counts['label'].str[0]

    # merge both dataframes based on label to get the node counts in the original df
    df = pd.merge(df_counts, df, how='left')

    # Add team number - to make it easy for the data ingestion team
    df["team"] = team

    df["dbName"] = db_name
```

Loop through all the labels to get list of associated relationships:

```
# Loop through all the labels to get list of associated relationships
for i in df.label.unique():
    print("Getting relationships for Node Label: %s" % i)

    relationshipQuery = '''
    MATCH (p1:%s)
    RETURN apoc.node.relationship.types(p1) AS output;
    ''' % i

    # Get results
    result = session.run(relationshipQuery).data()

    # Since a node may have one or more relationships & we want the list of ALL relationships -
    # some data wrangling to find max of length of all values in returned df and choose the one with max length
    relationships = DataFrame(result).loc[DataFrame(result).output.astype(str).map(len).argmax(), 'output']

    # Update the relationships against the node label
    df.loc[df.label == i, 'relationships'] = ','.join(relationships)

print("\nExtracted all metadata. Snapshot:\n")
print(df)
```

Call the write2gsheets function and pass the dataframe as a parameter to allow the function to write to GSheets:

```
def write2gsheets(df):
    # Find the service account file to allow writing to the gsheet
    gc = pygsheets.authorize(service_file='service_account.json')

    # open the google spreadsheet
    sh = gc.open('csye7250-metadata')

    # Write to Sheet
    wks = sh[team]
    wks.set_dataframe(df, (1, 1))

    print ("Data written to Google Sheet - for team %s" %team)
```

METADATA:

DEFINITION:

- Metadata summarizes basic information about data, making finding & working with instances of data easier.
- Metadata can be created manually to be more accurate, or automatically and contain more basic information.

BUSINESS METADATA:

- Business metadata is data that adds business context to other data. It provides information authored by Businesspeople.
- In our project, as each team is working with public dataset. Business metadata consists of information regarding the databases and their related attributes.
- Domain_name: Name of specific domain/ topic allotted to individual groups/team.
- BusinessTerm_Name: Name of columns/ attributes in the dataset
- Term_Description: Meaning of the Business term and a brief description of it.

TECHNICAL METADATA:

- Technical metadata describes information about technology such as the ownership of the database, physical characteristics of a database, performance tuning (processors, indexing), table name, column name, data type, and relationship between the tables.
- Count: Number of records for the property
- Label: Name of the node/ table in the data model
- Property: Name of the column from the dataset
- Type: Data type of the property/ column
- isIndexed: True/False value on if the data is indexed or not?
- uniqueConstraint: If the property containt a unique value or not?
- existenceConstraint: If the property exists for all nodes with a specific label?
- dbName: Name of the database being used.
- relationships: Relationships linked with the nodes.

DATAMODEL:

- A data model is a visual representation of data elements and the relationships between them. Data models help business and technical resources collaborate in the design of information systems and the databases that power them.

Below are the attached snippets for technical metadata, business metadata and data models for each team:

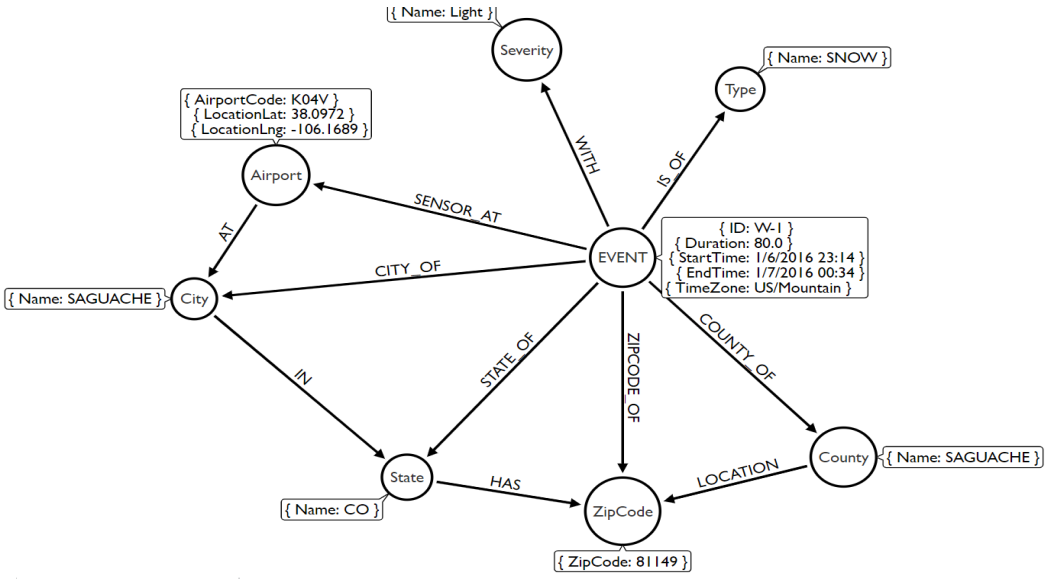
GROUP 1: Weather Report Database

Business Metadata:

	Domain_Name	BusinessTerm_Name	Term_Description
2	WeatherReport	Event ID	Number assigned per Event occured
3	WeatherReport	Event Type	Type of the event like rain , snow
4	WeatherReport	Event Severity	Metrics do describe how severe the event
5	WeatherReport	Event Start Time	Time stamp for date when event has occured
6	WeatherReport	Event End Time	Time stamp for date when event has completed
7	WeatherReport	Time-Zone	The US-Based time zone based on the location of the event
8	WeatherReport	Airport Code	The airport station that a weather event is reported from.
9	WeatherReport	Latitude Location	The latitude in GPS coordinate of airport-based weather station.
10	WeatherReport	Longitude Location	The longitude in GPS coordinate of airport-based weather station.
11	WeatherReport	City	The cityof airport-based weather station.
12	WeatherReport	County	The county of airport-based weather station.
13	WeatherReport	State	The state of airport-based weather station.
14	WeatherReport	Zip Code	The zip code of airport-based weather station.

Technical Metamodel:

	A	B	C	D	E	F	G	H	I	J	K	L
1	counts	label	property	type	isIndexed	uniqueConstrain	existenceConstr	team	dbName	relationships		
2	6274206	EVENT	StartTime	DATE_TIME	FALSE	FALSE	FALSE	1	Weather Database	COUNTY_OF,IS_OF,WITH,SENSOR_AT,CITY_OF,STATE_OF,ZIPCODE_OF		
3	6274206	EVENT	TimeZone	STRING	FALSE	FALSE	FALSE	1	Weather Database	COUNTY_OF,IS_OF,WITH,SENSOR_AT,CITY_OF,STATE_OF,ZIPCODE_OF		
4	6274206	EVENT	Duration	INTEGER	FALSE	FALSE	FALSE	1	Weather Database	COUNTY_OF,IS_OF,WITH,SENSOR_AT,CITY_OF,STATE_OF,ZIPCODE_OF		
5	6274206	EVENT	Id	STRING	TRUE	TRUE	FALSE	1	Weather Database	COUNTY_OF,IS_OF,WITH,SENSOR_AT,CITY_OF,STATE_OF,ZIPCODE_OF		
6	6274206	EVENT	EndTime	DATE_TIME	FALSE	FALSE	FALSE	1	Weather Database	COUNTY_OF,IS_OF,WITH,SENSOR_AT,CITY_OF,STATE_OF,ZIPCODE_OF		
7	7	TYPE	Name	STRING	TRUE	TRUE	FALSE	1	Weather Database	IS_OF		
8	6	SEVERITY	Name	STRING	TRUE	TRUE	FALSE	1	Weather Database	WITH		
9	2071	AIRPORT	AirportCode	STRING	TRUE	TRUE	FALSE	1	Weather Database	SENSOR_AT,AT		
10	2071	AIRPORT	LocationLng	FLOAT	FALSE	FALSE	FALSE	1	Weather Database	SENSOR_AT,AT		
11	2071	AIRPORT	LocationLat	FLOAT	FALSE	FALSE	FALSE	1	Weather Database	SENSOR_AT,AT		
12	1715	CITY	Name	STRING	TRUE	TRUE	FALSE	1	Weather Database	CITY_OF,IN,AT		
13	48	STATE	Name	STRING	TRUE	TRUE	FALSE	1	Weather Database	STATE_OF,IN,HAS		
14	2021	ZIPCODE	ZipCode	INTEGER	TRUE	TRUE	FALSE	1	Weather Database	ZIPCODE_OF,HAS,LOCATION		
15	1100	COUNTY	Name	STRING	TRUE	TRUE	FALSE	1	Weather Database	COUNTY_OF,LOCATION		
16												
17												



Data Model:

GROUP2: … Database

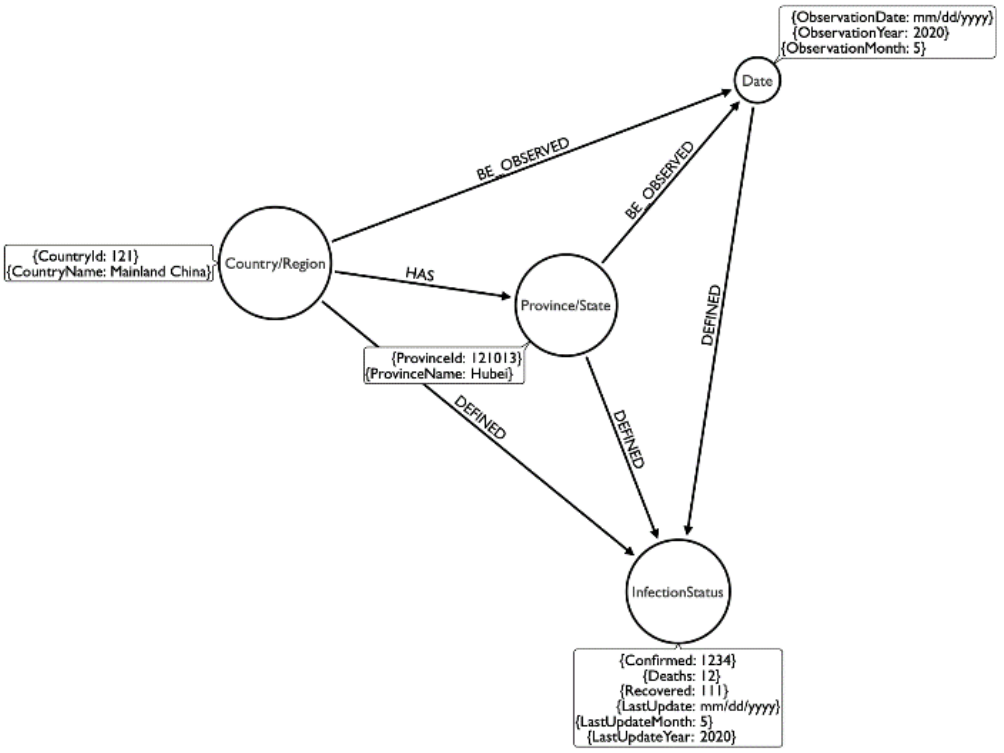
Business Metadata:

	A	B	C
1	**Domain_Name**	**BusinessTerm_Name**	**Term_Description**
2	covid_19_data	Serial Number	Unique number that identifies each row
3	covid_19_data	Date of Observation	The date when the entry was first made
4	covid_19_data	Province or State	State/Province where that entry belongs to
5	covid_19_data	Country or Region	Country/Region that the entry belongs to
6	covid_19_data	Last Update Date	The last date & time when the entries were updated
7	covid_19_data	Number of Confirmed	Number of confirmed covid-19 cases
8	covid_19_data	Number of Deaths	Number of deaths related to covid-19 case in that state
9	covid_19_data	Number of Recovered	Number of people that recovered of covid-19
10	covid_19_data	Year of Observation	Year when the entry was first made
11	covid_19_data	Month of Observation	Month when the entry was first made
12	covid_19_data	Year of Last Update	Year when the entries was last updated
13	covid_19_data	Month of Last Update	Month when the entries was last updated
14	covid_19_data	Country ID number	Unique ID to identify each country in the dataset
15	covid_19_data	Province ID number	Unique ID to identify each Province/State in the dataset

Technical Metamodel:

	A	B	C	D	E	F	G	H	I	J	K
1	counts	label	property	type	isIndexed	uniqueConstrain	existenceConstr	team	dbName	relationships	
2	224	Country	CountryId	STRING	TRUE	TRUE	FALSE	2	COVID19	HAS,BE_OBSERVED	
3	224	Country	CountryName	STRING	FALSE	FALSE	FALSE	2	COVID19	HAS,BE_OBSERVED	
4	966	Province	ProvinceName	STRING	FALSE	FALSE	FALSE	2	COVID19	HAS,BE_OBSERVED,DEFINED	
5	966	Province	ProvinceId	STRING	TRUE	TRUE	FALSE	2	COVID19	HAS,BE_OBSERVED,DEFINED	
6	403	Date	ObservationDate	DATE_TIME	TRUE	TRUE	FALSE	2	COVID19	BE_OBSERVED,DEFINED	
7	236017	InfectionStatus	UpdateTime	DATE_TIME	FALSE	FALSE	FALSE	2	COVID19	DEFINED	
8	236017	InfectionStatus	Recovered	INTEGER	FALSE	FALSE	FALSE	2	COVID19	DEFINED	
9	236017	InfectionStatus	EnteryId	STRING	TRUE	TRUE	FALSE	2	COVID19	DEFINED	
10	236017	InfectionStatus	Deaths	INTEGER	FALSE	FALSE	FALSE	2	COVID19	DEFINED	
11	236017	InfectionStatus	Confirmed	INTEGER	FALSE	FALSE	FALSE	2	COVID19	DEFINED	
12											

Data Model:



GROUP3: Vehicles Database

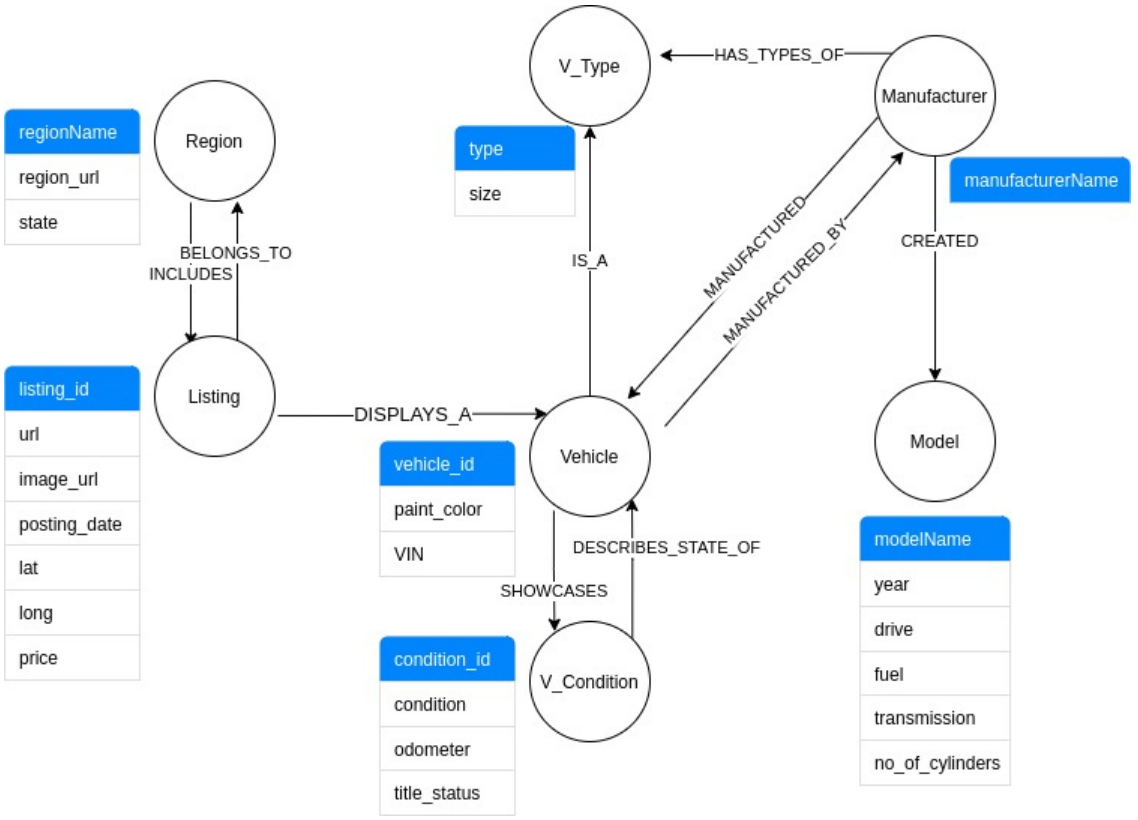
Business Metadata:

	Domain_Name	BusinessTerm_Name	Term_Description
2	Vehicles	Unnamed : 0	Column Number
3	Vehicles	listing_id	Craiglist data entry id
4	Vehicles	url	Post listing url
5	Vehicles	region	craiglist post region
6	Vehicles	region_url	region url
7	Vehicles	price	entry price of the vehicle listed on craiglist
8	Vehicles	year	manufacturing year
9	Vehicles	manufacturer	manufacturer name of the vehicle
10	Vehicles	model	model name of vehicle
11	Vehicles	condition	condition of vehicle
12	Vehicles	cylinders	number of cylinders in the vehicle
13	Vehicles	fuel	Type of fuel the vehicle uses
14	Vehicles	odometer	kilometers the vehicle has been driven
15	Vehicles	title_status	condition of vehicle in terms of quality
16	Vehicles	transmission	gear transmission used by the vehicle
17	Vehicles	VIN	vehicle identity number
18	Vehicles	drive	Type of drive quality the vehicle has
19	Vehicles	size	size of the vehicle
20	Vehicles	type	Category of the vehicle model
21	Vehicles	paint_color	paint color on the vehicle
22	Vehicles	image_url	link to vehicle's image
23	Vehicles	state	state where the craiglist post was made
24	Vehicles	lat	latitude of the location
25	Vehicles	long	longitude of the location
26	Vehicles	posting_date	date when the craiglist post was made
27	Vehicles	vehicle_id	unique identifier for node "vehicle"
28	Vehicles	condition_id	unique identifier for node "condition"

Technical Metamodel:

	A	B	C	D	E	F	G	H	I	J
1	counts	label	property	type	isIndexed	uniqueConstrain	existenceConstr	team	dbName	relationships
2	300	Region	regionName	string	TRUE	TRUE	TRUE	3	projectDB	INCLUDES, BELONGS_TO
3	300	Region	region_url	string	FALSE	FALSE	FALSE	3	projectDB	INCLUDES, BELONGS_TO
4	300	Region	state	string	FALSE	FALSE	FALSE	3	projectDB	INCLUDES, BELONGS_TO
5	458213	Listing	listing_id	integer	TRUE	TRUE	TRUE	3	projectDB	DISPLAYS_A, BELONGS_TO, INCLUDES
6	458213	Listing	URL	string	FALSE	FALSE	FALSE	3	projectDB	DISPLAYS_A, BELONGS_TO, INCLUDES
7	458213	Listing	posting_date	string	FALSE	FALSE	FALSE	3	projectDB	DISPLAYS_A, BELONGS_TO, INCLUDES
8	458213	Listing	price	string	FALSE	FALSE	FALSE	3	projectDB	DISPLAYS_A, BELONGS_TO, INCLUDES
9	458213	Listing	image_url	string	FALSE	FALSE	FALSE	3	projectDB	DISPLAYS_A, BELONGS_TO, INCLUDES
10	458213	Listing	lat	string	FALSE	FALSE	FALSE	3	projectDB	DISPLAYS_A, BELONGS_TO, INCLUDES
11	458213	Listing	long	string	FALSE	FALSE	FALSE	3	projectDB	DISPLAYS_A, BELONGS_TO, INCLUDES
12	458213	Vehicle	vehicle_id	integer	TRUE	TRUE	TRUE	3	projectDB	MANUFACTURED_BY, DISPLAYS_A, MANUFACTURED, IS_A, SHOWCASES, DESCRIBES_STATE_OF
13	458213	Vehicle	paint_color	string	FALSE	FALSE	FALSE	3	projectDB	MANUFACTURED_BY, DISPLAYS_A, MANUFACTURED, IS_A, SHOWCASES, DESCRIBES_STATE_OF
14	458213	Vehicle	VIN	string	FALSE	FALSE	FALSE	3	projectDB	MANUFACTURED_BY, DISPLAYS_A, MANUFACTURED, IS_A, SHOWCASES, DESCRIBES_STATE_OF
15	44	Manufacturer	manufacturerName	string	TRUE	TRUE	TRUE	3	projectDB	MANUFACTURED_BY, MANUFACTURED, CREATED, HAS_TYPES_OF
16	300	Model	modelName	string	TRUE	TRUE	TRUE	3	projectDB	CREATED
17	300	Model	year	string	FALSE	FALSE	FALSE	3	projectDB	CREATED
18	300	Model	fuel	string	FALSE	FALSE	FALSE	3	projectDB	CREATED
19	300	Model	drive	string	FALSE	FALSE	FALSE	3	projectDB	CREATED
20	300	Model	transmission	string	FALSE	FALSE	FALSE	3	projectDB	CREATED
21	300	Model	no_of_cylinders	string	FALSE	FALSE	FALSE	3	projectDB	CREATED
22	14	V_Type	type	string	TRUE	TRUE	TRUE	3	projectDB	HAS_TYPES_OF, IS_A
23	14	V_Type	size	string	FALSE	FALSE	FALSE	3	projectDB	HAS_TYPES_OF, IS_A
24	458213	V_Condition	condition_id	integer	TRUE	TRUE	TRUE	3	projectDB	SHOWCASES, DESCRIBES_STATE_OF
25	458213	V_Condition	odometer	string	FALSE	FALSE	FALSE	3	projectDB	SHOWCASES, DESCRIBES_STATE_OF
26	458213	V_Condition	condition	string	FALSE	FALSE	FALSE	3	projectDB	SHOWCASES, DESCRIBES_STATE_OF
27	458213	V_Condition	title_status	string	FALSE	FALSE	FALSE	3	projectDB	SHOWCASES, DESCRIBES_STATE_OF
28										

Data Model:


V_Type
HAS_TYPES_OF
Manufacturer
regionName
region_url
state
Region
type
size
manufacturerName
BELONGS_TO
INCLUDES
IS_A
MANUFACTURED
MANUFACTURED_BY
CREATED
listing_id
url
image_url
posting_date
lat
long
price
Listing
DISPLAYS_A
Vehicle
Model
vehicle_id
paint_color
VIN
DESCRIBES_STATE_OF
SHOWCASES
modelName
year
drive
fuel
transmission
no_of_cylinders
condition_id
condition
odometer
title_status
V_Condition

GROUP 5: Hotel Review Database

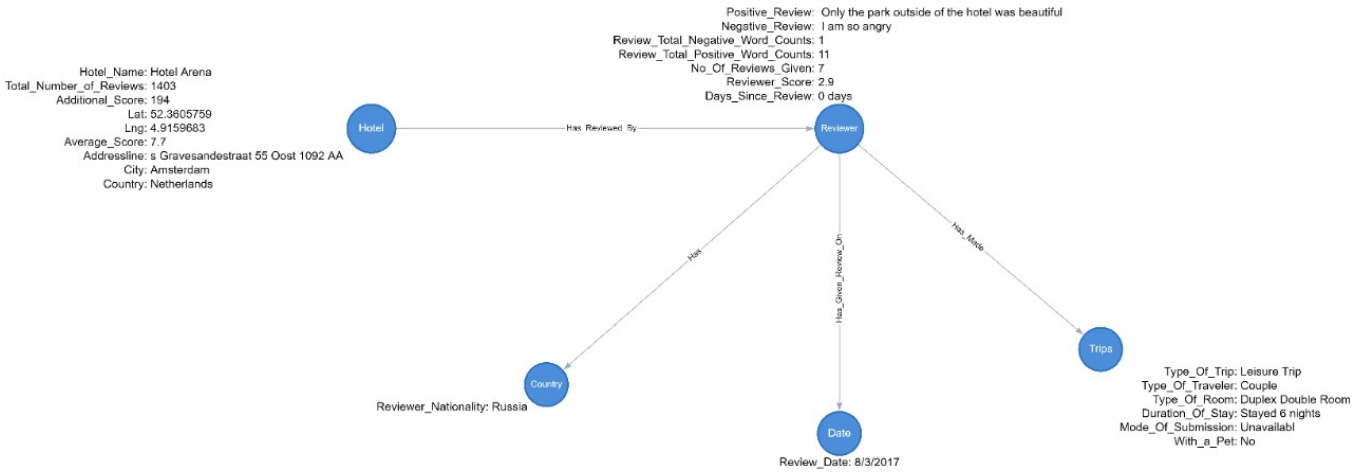
Business Metadata:

Domain_Name	BusinessTerm_Name	Term_Description
HotelReview	Hotel_Address	Address of hotel
HotelReview	Additional_Number_of_Scoring	There are also some guests who just made a scoring on the service rather than a review. This number indicates how many valid scores without review in there
HotelReview	Average_Score	Average Score of the hotel, calculated based on the latest comment in the last year
HotelReview	Hotel_Name	Name of Hotel
HotelReview	Total_Number_of_Reviews	Total number of valid reviews the hotel has
HotelReview	lat	Latitude of the hotel
HotelReview	lng	Longtitude of the hotel
HotelReview	City	City of hotel according to its address
HotelReview	Country	Country of hotel according to its address
HotelReview	Review_Date	Date when reviewer posted the corresponding review
HotelReview	Negative_Review	Negative Review the reviewer gave to the hotel. If the reviewer does not give the negative review, then it should be: 'No Negative'
HotelReview	Review_Total_Negative_Word_Counts	Total number of words in the negative review
HotelReview	Positive_Review	Positive Review the reviewer gave to the hotel. If the reviewer does not give the negative review, then it should be: 'No Positive'
HotelReview	Review_Total_Positive_Word_Counts	Total number of words in the positive review
HotelReview	days_since_review	Duration between the review date and scrape date
HotelReview	Reviewer_Nationality	Nationality of Reviewer
HotelReview	Total_Number_of_Reviews_Reviewer_Has_Given	Number of Reviews the reviewers has given in the past
HotelReview	Reviewer_Score	Score the reviewer has given to the hotel, based on his/her experience
HotelReview	Type_of_Trip	Type of the trip selected by the reviewer
HotelReview	Type_of_Travellers	Type of travellers selected by the reviewer
HotelReview	Type_of_Room	Type of the room selected by the reviewer
HotelReview	Duration_of_Stay	Duration of reviewer's stay night
HotelReview	With_Pet	Is the reviewer with a pet or not? If the reviewer is with a pet, it should be: 'With a pet'; If the reviewer is not with any pet, it should be: 'No'
HotelReview	Mode_of_Submission	Mode the reviewer submits reviews

Technical Metamodel:

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	counts	label	property	type	isIndexed	uniqueConstrain	existenceConstr	team	dbName	relationships			
2	1493	Hotel	Total_Reviews	STRING	FALSE	FALSE	FALSE	5	HotelReview	Is_ReviewedBy			
3	1493	Hotel	Additional_Numl	STRING	FALSE	FALSE	FALSE	5	HotelReview	Is_ReviewedBy			
4	1493	Hotel	HotelID	STRING	TRUE	TRUE	FALSE	5	HotelReview	Is_ReviewedBy			
5	1493	Hotel	lng	STRING	FALSE	FALSE	FALSE	5	HotelReview	Is_ReviewedBy			
6	1493	Hotel	Country	STRING	FALSE	FALSE	FALSE	5	HotelReview	Is_ReviewedBy			
7	1493	Hotel	Hotel_Name	STRING	FALSE	FALSE	FALSE	5	HotelReview	Is_ReviewedBy			
8	1493	Hotel	City	STRING	FALSE	FALSE	FALSE	5	HotelReview	Is_ReviewedBy			
9	1493	Hotel	lat	STRING	FALSE	FALSE	FALSE	5	HotelReview	Is_ReviewedBy			
10	1493	Hotel	Average_Score	STRING	FALSE	FALSE	FALSE	5	HotelReview	Is_ReviewedBy			
11	515738	Reviewer	Positive_Review	STRING	FALSE	FALSE	FALSE	5	HotelReview	Has_Made,Is_ReviewedBy,Belongs_To,Has_Reviewed_On			
12	515738	Reviewer	ReviewerID	STRING	TRUE	TRUE	FALSE	5	HotelReview	Has_Made,Is_ReviewedBy,Belongs_To,Has_Reviewed_On			
13	515738	Reviewer	Days_Since_Re	STRING	FALSE	FALSE	FALSE	5	HotelReview	Has_Made,Is_ReviewedBy,Belongs_To,Has_Reviewed_On			
14	515738	Reviewer	Reviewer_Score	STRING	FALSE	FALSE	FALSE	5	HotelReview	Has_Made,Is_ReviewedBy,Belongs_To,Has_Reviewed_On			
15	515738	Reviewer	Negative_Revie	STRING	FALSE	FALSE	FALSE	5	HotelReview	Has_Made,Is_ReviewedBy,Belongs_To,Has_Reviewed_On			
16	515738	Reviewer	Number_Of_revi	STRING	FALSE	FALSE	FALSE	5	HotelReview	Has_Made,Is_ReviewedBy,Belongs_To,Has_Reviewed_On			
17	227	Country	CountryID	STRING	FALSE	FALSE	FALSE	5	HotelReview	Belongs_To			
18	227	Country	Reviewer_Natio	STRING	FALSE	FALSE	FALSE	5	HotelReview	Belongs_To			
19	731	Date	Review_Date	STRING	FALSE	FALSE	FALSE	5	HotelReview	Has_Reviewed_On			
20	55242	Trip	With_Pet	STRING	FALSE	FALSE	FALSE	5	HotelReview	Has_Made			
21	55242	Trip	TripID	STRING	TRUE	TRUE	FALSE	5	HotelReview	Has_Made			
22	55242	Trip	Reviewer_Score	STRING	FALSE	FALSE	FALSE	5	HotelReview	Has_Made			
23	55242	Trip	Mode_of_Submi	STRING	FALSE	FALSE	FALSE	5	HotelReview	Has_Made			

Data Model:



Group 4: Consumer Complaints Database

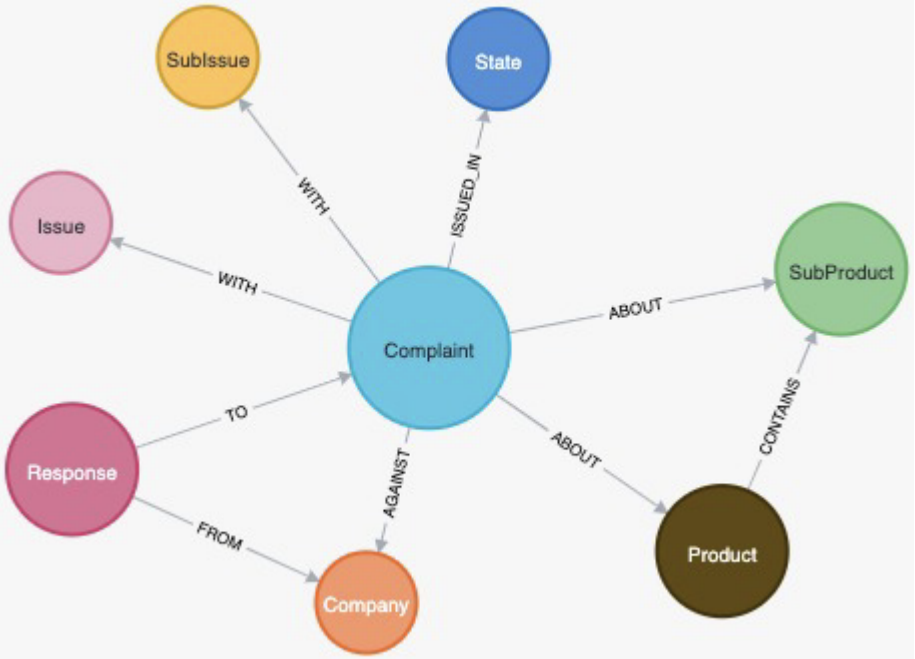
Business Metadata:

	A	B	C
1	**Domain_Name**	**BusinessTerm_Name**	**Term_Description**
2	ConsumerComplaints	Complaint Number	Number generated uniquely for each complaint filed
3	ConsumerComplaints	Complaint Date	Date on which the complaint was received before sending it to the respective company
4	ConsumerComplaints	Company Name	The company addressing the complaint
5	ConsumerComplaints	Response	Response given by the company to the customer
6	ConsumerComplaints	Product	The financial service that the customer had complaint for
7	ConsumerComplaints	Sub-Product	The specific financial service under product
8	ConsumerComplaints	State	The State where the complaint was filed
9	ConsumerComplaints	Zip Code	The zip code where the complaint was filed. Basically more specific location of the complaint
10	ConsumerComplaints	Issue	Problem that the customer was facing in vague terms
11	ConsumerComplaints	Sub-Issue	Problem faced by customer but more exact with respect to the issue

Technical Metamodel:

	A	B	C	D	E	F	G	H	I	J	K	L
1	counts	label	property	type	isIndexed	uniqueConstraint	existenceConstraint	team	dbName	relationships		
2	670598	Complaint	ComplaintId	STRING	TRUE	TRUE	FALSE	4	complaints	WITH,AGAINST,ORIGINATED_FROM,ABOUT		
3	670598	Complaint	DateReceived	STRING	FALSE	FALSE	FALSE	4	complaints	WITH,AGAINST,ORIGINATED_FROM,ABOUT		
4	3933	Company	name	STRING	TRUE	TRUE	FALSE	4	complaints	AGAINST,FROM		
5	8	Response	companyRespor	STRING	TRUE	TRUE	FALSE	4	complaints	FROM		
6	12	Product	ProductName	STRING	TRUE	TRUE	FALSE	4	complaints	ABOUT,CONTAINS		
7	48	SubProduct	SubProductNam	STRING	TRUE	TRUE	FALSE	4	complaints	ABOUT,CONTAINS		
8	63	State	StateName	STRING	TRUE	TRUE	FALSE	4	complaints	ORIGINATED_FROM		
9	27760	Zipcode	Zip	STRING	TRUE	TRUE	FALSE	4	complaints	ORIGINATED_FROM		
10	95	Issue	IssueName	STRING	TRUE	TRUE	FALSE	4	complaints	WITH,IN_CATEGORY		
11	69	SubIssue	SubIssueName	STRING	TRUE	TRUE	FALSE	4	complaints	WITH,IN_CATEGORY		

Data Model:



Group 6: Global Trade Statistics Database:

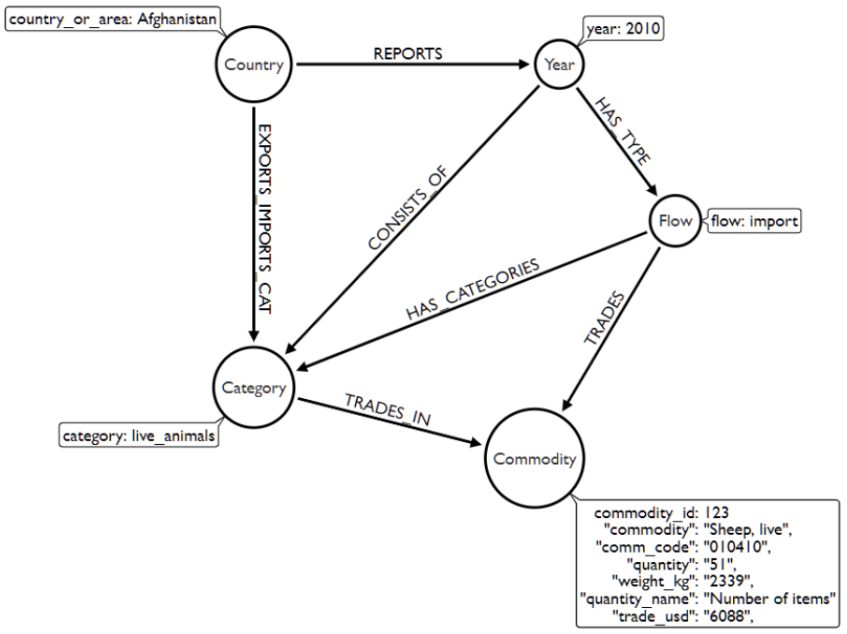
Business Metadata:

	Domain_Name	BusinessTerm_Name	Term_Description
2	GlobalTradeStats	Country	Area or country taking part in exports/imports
3	GlobalTradeStats	Year	Year in which country exported/imported a given commodity
4	GlobalTradeStats	Commodity Code	Unique identifier for a commodity
5	GlobalTradeStats	Commodity	Commodity/Item being exported/imported by a country
6	GlobalTradeStats	Flow	Flow of trade i.e. Export, Import
7	GlobalTradeStats	Trade Value	Value of the trade in US Dollars
8	GlobalTradeStats	Weight	Weight of the commodity in Kilograms
9	GlobalTradeStats	Quantity Name	A description of the quantity measurement type given the type of item (i.e. Number of Items, Weight in KG)
10	GlobalTradeStats	Quantity	Count of the quantity of a given item based on the Quantity Name
11	GlobalTradeStats	Category	Category to identify commodity

Technical Metamodel:

	A	B	C	D	E	F	G	H	I	J
1	counts	label	property	type	isIndexed	uniqueConstrain	existenceConstr	team	dbName	relationships
2	8225871	Commodity	CommodityCode	STRING	FALSE	FALSE	FALSE	6	CommodityDB	TRADES,BELONGS_TO
3	8225871	Commodity	Quantity	STRING	FALSE	FALSE	FALSE	6	CommodityDB	TRADES,BELONGS_TO
4	8225871	Commodity	QuantityName	STRING	FALSE	FALSE	FALSE	6	CommodityDB	TRADES,BELONGS_TO
5	8225871	Commodity	TradeValue	STRING	FALSE	FALSE	FALSE	6	CommodityDB	TRADES,BELONGS_TO
6	8225871	Commodity	CommodityName	STRING	FALSE	FALSE	FALSE	6	CommodityDB	TRADES,BELONGS_TO
7	8225871	Commodity	Weight	STRING	FALSE	FALSE	FALSE	6	CommodityDB	TRADES,BELONGS_TO
8	98	Category	category	STRING	TRUE	TRUE	FALSE	6	CommodityDB	BELONGS_TO,EXPORTS_IMPORTS_CAT,HAS_CATEGORIES
9	209	Country	country	STRING	FALSE	FALSE	FALSE	6	CommodityDB	EXPORTS_IMPORTS_CAT,REPORTS
10	4	Flow	flow	STRING	TRUE	TRUE	FALSE	6	CommodityDB	HAS_TYPE,TRADES,HAS_CATEGORIES
11	29	Year	year	STRING	TRUE	TRUE	FALSE	6	CommodityDB	REPORTS,HAS_TYPE

Data Model:



Group 7: Building Permits Database

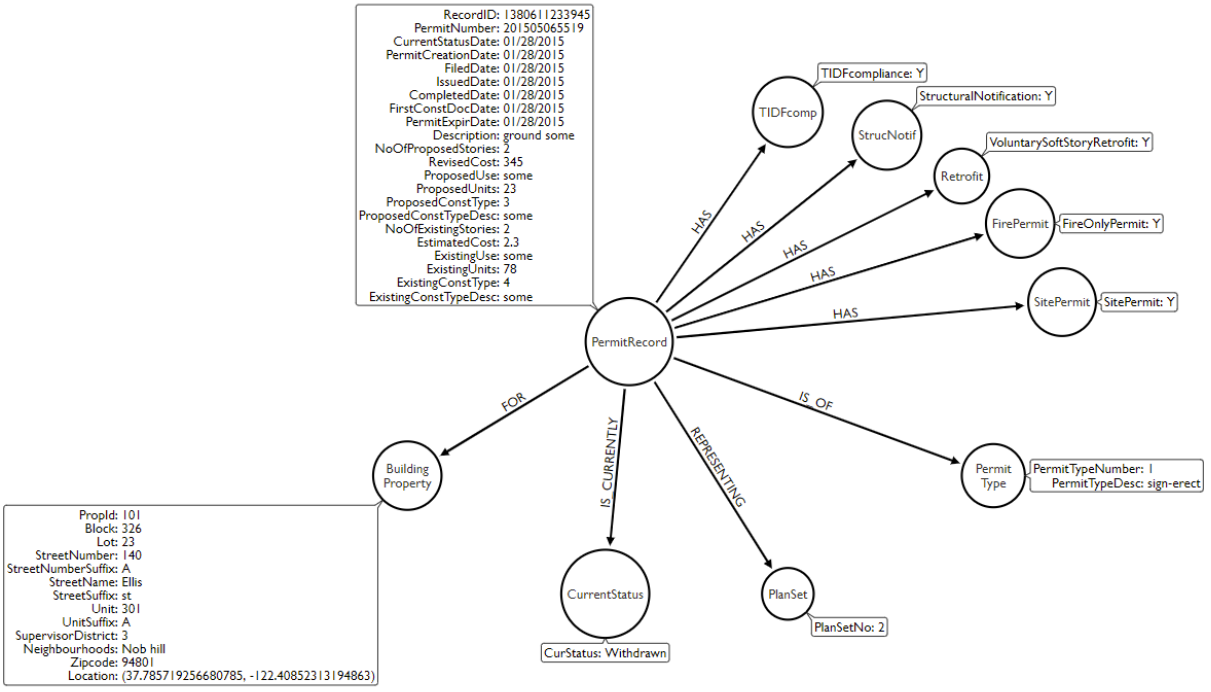
Business Metadata:

	Domain_Name	BusinessTerm_Name	Term_Description
2	BuildingPermits	Permit Number	Number assigned while filing
3	BuildingPermits	Permit Type	Type of the permit represented numerically
4	BuildingPermits	Permit Type Definition	Description of the Permit type, for example new construction, alterations
5	BuildingPermits	Permit Creation Date	Date on which permit created, later than or same as filing date
6	BuildingPermits	Block	Related to address
7	BuildingPermits	Lot	Related to address
8	BuildingPermits	Street Number	Related to address
9	BuildingPermits	Street Number Suffix	Related to address
10	BuildingPermits	Street Name	Related to address
11	BuildingPermits	Street Name Suffix	Related to address
12	BuildingPermits	Unit	Unit of a building
13	BuildingPermits	Unit suffix	Suffix if any, for the unit
14	BuildingPermits	Description	Details about purpose of the permit. Example: reroofing, bathroom renovation
15	BuildingPermits	Current Status	Current status of the permit application
16	BuildingPermits	Current Status Date	Date at which current status was entered
17	BuildingPermits	Filed Date	Filed date for the permit
18	BuildingPermits	Issued Date	Issued date for the permit
19	BuildingPermits	Completed Date	The date on which project was completed, applicable if Current Status = "completed"
20	BuildingPermits	First Construction Document Date	Date on which construction was documented
21	BuildingPermits	Structural Notification	Notification to meet some legal need, given or not
22	BuildingPermits	Number of Existing Stories	Number of existing stories in the building. Not applicable for certain permit types
23	BuildingPermits	Number of Proposed Stories	Number of proposed stories for the construction/alteration
24	BuildingPermits	Voluntary Soft-Story Retrofit	Soft story to meet earth quake regulations
25	BuildingPermits	Fire Only Permit	Fire hazard prevention related permit
26	BuildingPermits	Permit Expiration Date	Expiration date related to issued permit
27	BuildingPermits	Estimated Cost	Initial estimation of the cost of the project
28	BuildingPermits	Revised Cost	Revised estimation of the cost of the project
29	BuildingPermits	Existing Use	Existing use of the building
30	BuildingPermits	Existing Units	Existing number of units
31	BuildingPermits	Proposed Use	Proposed use of the building
32	BuildingPermits	Proposed Units	Proposed number of units
33	BuildingPermits	Plansets	Plan representation indicating the general design intent of the foundation
34	BuildingPermits	TIDF Compliance	TIDF compliant or not, this is a new legal requirement
35	BuildingPermits	Existing Construction Type	Construction type, existing,as categories represented numerically
36	BuildingPermits	Existing Construction Type Description	Description of the construction type, for example, wood or other construction types
37	BuildingPermits	Proposed Construction Type	Construction type, proposed, as categories represented numerically
38	BuildingPermits	Proposed Construction Type Description	Description of the proposed construction type
39	BuildingPermits	Site Permit	Permit for site
40	BuildingPermits	Supervisor District	Supervisor District to which the building location belongs to
41	BuildingPermits	Neighborhoods - Analysis Boundaries	Neighborhood to which the building location belongs to
42	BuildingPermits	Zipcode	Zipcode of building address
43	BuildingPermits	Location	Location in latitude, longitude pair
44	BuildingPermits	Record ID	Some ID, for unique identification
45	BuildingPermits	PropId	Unique Id for identifying a building property

Technical Metamodel:

	A	B	C	D	E	F	G	H	I	J
1	counts	label	property	type	isIndexed	uniqueConstraint	existenceConstraint	team	dbName	relationships
2	198900	PermitRecord	NoOfProposedStories	STRING	FALSE	FALSE	FALSE	7	BuildingPermits	HAS,IS_OF,FOR,IS_CURRENTLY,REPRESENTING
3	198900	PermitRecord	Description	STRING	FALSE	FALSE	FALSE	7	BuildingPermits	HAS,IS_OF,FOR,IS_CURRENTLY,REPRESENTING
4	198900	PermitRecord	ProposedConstTypeDesc	STRING	FALSE	FALSE	FALSE	7	BuildingPermits	HAS,IS_OF,FOR,IS_CURRENTLY,REPRESENTING
5	198900	PermitRecord	ExistingConstType	STRING	FALSE	FALSE	FALSE	7	BuildingPermits	HAS,IS_OF,FOR,IS_CURRENTLY,REPRESENTING
6	198900	PermitRecord	PermitNumber	STRING	FALSE	FALSE	FALSE	7	BuildingPermits	HAS,IS_OF,FOR,IS_CURRENTLY,REPRESENTING
7	198900	PermitRecord	PermitExpirationDate	STRING	FALSE	FALSE	FALSE	7	BuildingPermits	HAS,IS_OF,FOR,IS_CURRENTLY,REPRESENTING
8	198900	PermitRecord	ExistingConstTypeDesc	STRING	FALSE	FALSE	FALSE	7	BuildingPermits	HAS,IS_OF,FOR,IS_CURRENTLY,REPRESENTING
9	198900	PermitRecord	EstimatedCost	STRING	FALSE	FALSE	FALSE	7	BuildingPermits	HAS,IS_OF,FOR,IS_CURRENTLY,REPRESENTING
10	198900	PermitRecord	ProposedConstType	STRING	FALSE	FALSE	FALSE	7	BuildingPermits	HAS,IS_OF,FOR,IS_CURRENTLY,REPRESENTING
11	198900	PermitRecord	ProposedUse	STRING	FALSE	FALSE	FALSE	7	BuildingPermits	HAS,IS_OF,FOR,IS_CURRENTLY,REPRESENTING
12	198900	PermitRecord	CurrentStatusDate	STRING	FALSE	FALSE	FALSE	7	BuildingPermits	HAS,IS_OF,FOR,IS_CURRENTLY,REPRESENTING
13	198900	PermitRecord	ProposedUnits	STRING	FALSE	FALSE	FALSE	7	BuildingPermits	HAS,IS_OF,FOR,IS_CURRENTLY,REPRESENTING
14	198900	PermitRecord	CompletedDate	STRING	FALSE	FALSE	FALSE	7	BuildingPermits	HAS,IS_OF,FOR,IS_CURRENTLY,REPRESENTING
15	198900	PermitRecord	RevisedCost	STRING	FALSE	FALSE	FALSE	7	BuildingPermits	HAS,IS_OF,FOR,IS_CURRENTLY,REPRESENTING
16	198900	PermitRecord	PermitRecordId	STRING	TRUE	TRUE	FALSE	7	BuildingPermits	HAS,IS_OF,FOR,IS_CURRENTLY,REPRESENTING
17	198900	PermitRecord	NoOfExistingStories	STRING	FALSE	FALSE	FALSE	7	BuildingPermits	HAS,IS_OF,FOR,IS_CURRENTLY,REPRESENTING
18	198900	PermitRecord	IssuedDate	STRING	FALSE	FALSE	FALSE	7	BuildingPermits	HAS,IS_OF,FOR,IS_CURRENTLY,REPRESENTING
19	198900	PermitRecord	FirstConstructionDocDate	STRING	FALSE	FALSE	FALSE	7	BuildingPermits	HAS,IS_OF,FOR,IS_CURRENTLY,REPRESENTING
20	198900	PermitRecord	ExistingUnits	STRING	FALSE	FALSE	FALSE	7	BuildingPermits	HAS,IS_OF,FOR,IS_CURRENTLY,REPRESENTING
21	198900	PermitRecord	PermitCreationDate	STRING	FALSE	FALSE	FALSE	7	BuildingPermits	HAS,IS_OF,FOR,IS_CURRENTLY,REPRESENTING
22	198900	PermitRecord	ExistingUse	STRING	FALSE	FALSE	FALSE	7	BuildingPermits	HAS,IS_OF,FOR,IS_CURRENTLY,REPRESENTING
23	198900	PermitRecord	FiledDate	STRING	FALSE	FALSE	FALSE	7	BuildingPermits	HAS,IS_OF,FOR,IS_CURRENTLY,REPRESENTING
24	8	PermitType	PermitTypeDesc	STRING	FALSE	FALSE	FALSE	7	BuildingPermits	IS_OF
25	8	PermitType	PermitTypeNum	STRING	TRUE	TRUE	FALSE	7	BuildingPermits	IS_OF
26	81881	BuildingProperty	PropId	STRING	TRUE	TRUE	FALSE	7	BuildingPermits	FOR
27	81881	BuildingProperty	StreetName	STRING	FALSE	FALSE	FALSE	7	BuildingPermits	FOR
28	81881	BuildingProperty	SupervisorDistrict	STRING	FALSE	FALSE	FALSE	7	BuildingPermits	FOR
29	81881	BuildingProperty	Neighbourhoods	STRING	FALSE	FALSE	FALSE	7	BuildingPermits	FOR
30	81881	BuildingProperty	Unit	STRING	FALSE	FALSE	FALSE	7	BuildingPermits	FOR
31	81881	BuildingProperty	UnitSuffix	STRING	FALSE	FALSE	FALSE	7	BuildingPermits	FOR
32	81881	BuildingProperty	StreetSuffix	STRING	FALSE	FALSE	FALSE	7	BuildingPermits	FOR
33	81881	BuildingProperty	Lot	STRING	FALSE	FALSE	FALSE	7	BuildingPermits	FOR
34	81881	BuildingProperty	Zipcode	STRING	FALSE	FALSE	FALSE	7	BuildingPermits	FOR
35	81881	BuildingProperty	StreetNumber	STRING	FALSE	FALSE	FALSE	7	BuildingPermits	FOR
36	81881	BuildingProperty	Block	STRING	FALSE	FALSE	FALSE	7	BuildingPermits	FOR
37	81881	BuildingProperty	StreetNumberSuffix	STRING	FALSE	FALSE	FALSE	7	BuildingPermits	FOR
38	81881	BuildingProperty	Location	STRING	FALSE	FALSE	FALSE	7	BuildingPermits	FOR
39	14	CurrentStatus	CurStatus	STRING	TRUE	TRUE	FALSE	7	BuildingPermits	IS_CURRENTLY
40	9	PlanSet	PlanSetNum	STRING	TRUE	TRUE	FALSE	7	BuildingPermits	REPRESENTING
41	2	SitePermit	SitePermitCond	STRING	TRUE	TRUE	FALSE	7	BuildingPermits	HAS
42	2	FirePermit	FirePermitCond	STRING	TRUE	TRUE	FALSE	7	BuildingPermits	HAS
43	2	Retrofit	RetrofitCond	STRING	TRUE	TRUE	FALSE	7	BuildingPermits	HAS
44	2	StrucNotif	StrucNotifCond	STRING	TRUE	TRUE	FALSE	7	BuildingPermits	HAS
45	3	TidfComp	TidfCompCond	STRING	TRUE	TRUE	FALSE	7	BuildingPermits	HAS

Data Model:



Group 8: US permit Visas Database

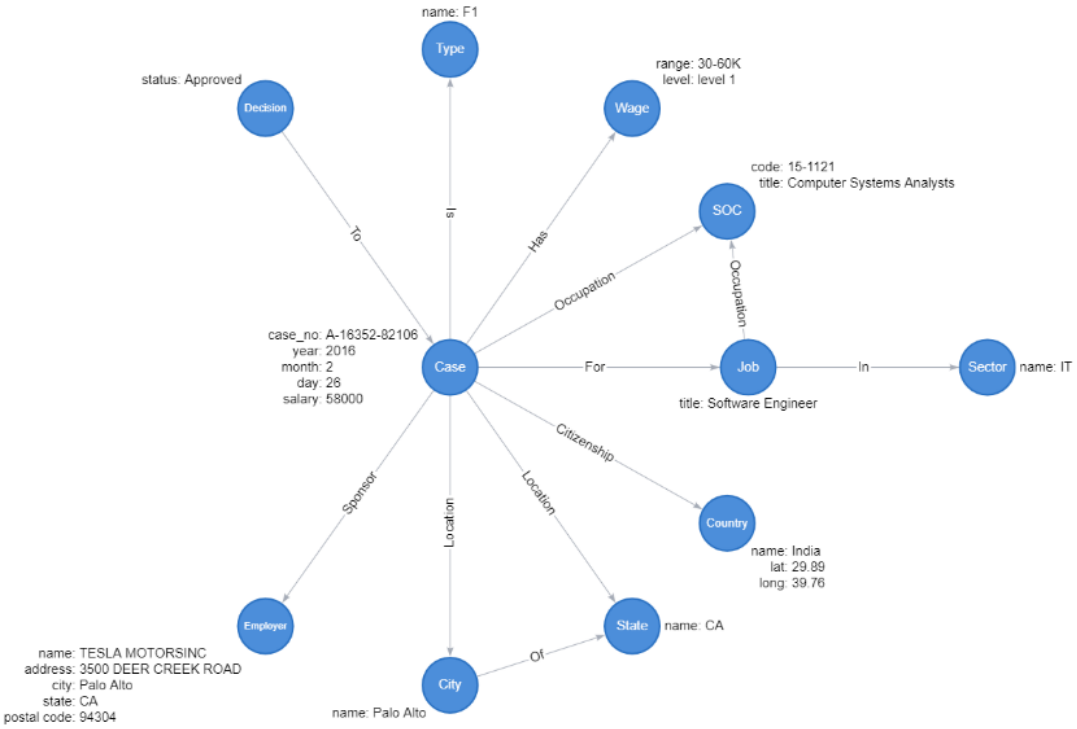
Business Metadata:

	Domain_Name	BusinessTerm_Name	Term_Description
2	USPermVisas	Case Number	Unique identifier assigned to each application submitted for processing
3	USPermVisas	Application Year	Year during which the application was submitted for processing
4	USPermVisas	Application Month	Month during which the application was submitted for processing
5	USPermVisas	Application Day	Day during which the application was submitted for processing
6	USPermVisas	Application Decision	Status associated with the last significant event or decision. Example: Apprpoved or Denied
7	USPermVisas	Country Of Citizenship	Country of citizenship of the foreign worker being sponsored
8	USPermVisas	Country Latitude	Derived Column showing Latitude Coordinates for each country
9	USPermVisas	Country Longitude	Derived Column showing Longitude Coordinates for each country
10	USPermVisas	Visa Type	Class of immigration visa the foreign worker held. Example: F1, H1-B
11	USPermVisas	SOC Code	Standard Occupational code associated with the job being requested for permanent labor certification
12	USPermVisas	SOC Title	Standard Occupational Code Title of the permanent job.
13	USPermVisas	Job City	City in which the foreign worker is expected to be employed
14	USPermVisas	Job State	State in which the foreign worker is expected to be employed
15	USPermVisas	Employer Name	Name of employer requesting permanent labor certification
16	USPermVisas	Employer Address	Address information of the employer requesting permanent labor certification.
17	USPermVisas	Employer City	City name of the employer requesting permanent labor certification.
18	USPermVisas	Employer State	State name of the employer requesting permanent labor certification.
19	USPermVisas	Employer Postal Code	Postal Code of the employer requesting permanent labor certification.
20	USPermVisas	Wage Amount	Prevailing wage for the job being requested for permanent labor certification
21	USPermVisas	Wage Level	Level of the prevailing wage determination. Valid values include “Level I,” “Level II,” “Level III,” and “Level IV”
22	USPermVisas	Wage Range	Derived Column where Wage Amount is split into different Ranges
23	USPermVisas	Job Title	Industry title associated with the North American Industrial Classification System (NAICS) code
24	USPermVisas	Job Sector	Major economic sector associated with the NAICS code of the employer.

Technical Metamodel:

	A	B	C	D	E	F	G	H	I	J
1	counts	label	property	type	isIndexed	uniqueConstrain	existenceConstraint	team	dbName	relationships
2	373025	Case	case_no	STRING	TRUE	TRUE	FALSE	8	USPermVisas	Has,Citizenship,To,Is,Occupation,Sponsor,For,Location
3	373025	Case	month	INTEGER	FALSE	FALSE	FALSE	8	USPermVisas	Has,Citizenship,To,Is,Occupation,Sponsor,For,Location
4	373025	Case	salary	INTEGER	FALSE	FALSE	FALSE	8	USPermVisas	Has,Citizenship,To,Is,Occupation,Sponsor,For,Location
5	373025	Case	day	INTEGER	FALSE	FALSE	FALSE	8	USPermVisas	Has,Citizenship,To,Is,Occupation,Sponsor,For,Location
6	373025	Case	year	INTEGER	FALSE	FALSE	FALSE	8	USPermVisas	Has,Citizenship,To,Is,Occupation,Sponsor,For,Location
7	203	Country	name	STRING	TRUE	TRUE	FALSE	8	USPermVisas	Citizenship
8	3	Decision	status	STRING	TRUE	TRUE	FALSE	8	USPermVisas	To
9	58	Type	name	STRING	TRUE	TRUE	FALSE	8	USPermVisas	Is
10	1368	SOC	title	STRING	FALSE	FALSE	FALSE	8	USPermVisas	Occupation
11	1368	SOC	code	STRING	TRUE	TRUE	FALSE	8	USPermVisas	Occupation
12	70846	Employer	name	STRING	TRUE	TRUE	FALSE	8	USPermVisas	Sponsor
13	70846	Employer	address	STRING	FALSE	FALSE	FALSE	8	USPermVisas	Sponsor
14	70846	Employer	state	STRING	FALSE	FALSE	FALSE	8	USPermVisas	Sponsor
15	70846	Employer	postal_code	STRING	FALSE	FALSE	FALSE	8	USPermVisas	Sponsor
16	70846	Employer	city	STRING	FALSE	FALSE	FALSE	8	USPermVisas	Sponsor
17	45754	Job	title	STRING	TRUE	TRUE	FALSE	8	USPermVisas	In,For
18	6833	City	name	STRING	TRUE	TRUE	FALSE	8	USPermVisas	Location,Of
19	58	State	name	STRING	TRUE	TRUE	FALSE	8	USPermVisas	Location,Of
20	10	Wage	range	STRING	TRUE	TRUE	FALSE	8	USPermVisas	Has
21	10	Wage	level	STRING	FALSE	FALSE	FALSE	8	USPermVisas	Has
22	18	Sector	name	STRING	TRUE	TRUE	FALSE	8	USPermVisas	In

Data Model:



2.2. Database

Database Creation Procedure:

The Database team started by gathering and analyzing the business requirements. DB team then collaborated with Ingestion, front end team and other teams to understand the proposed system and obtain and document the data and functional requirements. This helped us achieve the detailed requirements provided by the teams. These meetings with different teams also helped us established requirements involved agreement among, all the users as to what data they want to store along with an agreement as to the meaning and interpretation of the data elements. E.g.: Technical and Business meta data

Data analysis was done with the help of data requirements and then we produced a conceptual data model. The aim of analysis was to obtain a detailed description of the data that will suit user requirements so that both high- and low-level properties of data and their use are dealt with. Based on the requirements the team drafted a conceptual data model which contained information about what data a database should contain and the constraints the data must satisfy.

Conceptual data model was based on the analysis and focused on the question, “What is required by each team?”

Based on the conceptual design we drafted a relational representation of the conceptual data model in ER studio because of its convenience and ease of use. We could easily generate SQL scripts based on the model. This design was our 1st input to the logical design process. We decided the entities and relationships and defined Primary Key, Foreign keys. The output of this stage was a detailed relational specification, the logical schema, of all the tables and constraints needed to satisfy the description of the data in the conceptual data model.

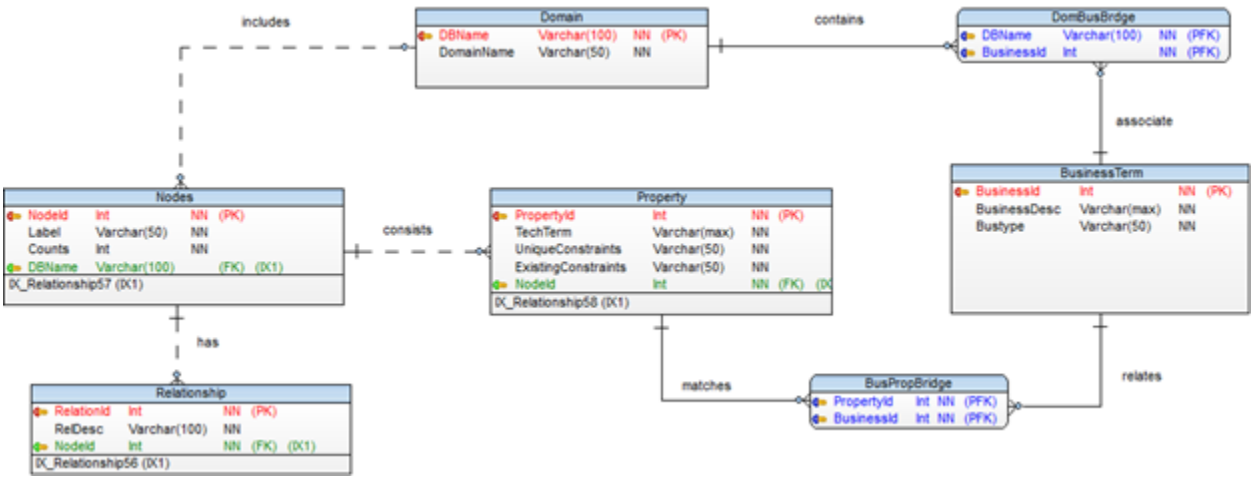
This logical model was then reviewed by all the teams. We did some alterations in our initial design based on the feedbacks received from the teams.

This logical schema was then implemented in SQL server DBMS as everyone in the team had detailed knowledge of the specific features and facilities that SSMS had to offer. SSMS was also faster and easily accessible than the other available options (MySQL).

The DB model consists of 7 tables. The “Domain” table has 2 columns and is used to retrieve each team’s DB name and domain. Domain consists of nodes and has zero/one to many relationships with the “Node” table. Node table has 4 columns, and it is used to define/create main junction in graph DB. It has several properties and relationships with other nodes. To define relationships with other nodes, a table called “Relationship” is created with 3 columns and has zero/one to one relationship with “Nodes” table. “Property” table is related to “Nodes” with zero/one to one cardinality. It consists of 5 columns and serves to create the properties of a node. We can extract the business terms from every team’s DB. A table called “BusinessTerm” is created and contains 3 columns. It has to be connected to “Domain” table in order to extract the DBname. That is why a bridge table called “DomBusBridge” has been created and has PK of both domain and businessterm tables. “BusinessTerm” is associated with the “property” table. A bridge table called “BusPropBridge” is created with the primary keys. Each property should be properly assigned against its business terms.

Normalization is one of the factors which we considered while designing the DB model. The DB model has been redesigned a few times to achieve its desired aim.

Database Design Model:



Entities:

1. Domain
2. DomBusBrdge
3. BusinessTerm
4. BusPropBridge
5. Property
6. Nodes
7. Relationship

SQL Scripts to create tables and define relationships:

```
4   Model: Microsoft SQL Server 2014
5   Database: MS SQL Server 2014
6   */
7
8
9   -- Create tables section -------------------------------------------------
10  -- Table Domain
11  CREATE TABLE [Domain]
12  (
13   [DBName] Varchar(100) NOT NULL,
14   [DomainName] Varchar(50) NOT NULL
15  )
16  go
17
18  -- Add keys for table Domain
19
20  ALTER TABLE [Domain] ADD CONSTRAINT [PK_Domain] PRIMARY KEY ([DBName])
21  go
22
23  -- Table BusinessTerm
24
25  CREATE TABLE [BusinessTerm]
26  (
27   [BusinessId] Int IDENTITY(100,1) NOT NULL,
28   [BusinessDesc] Varchar(max) NOT NULL,
29   [Bustype] Varchar(50) NOT NULL
30  )
31  go
32
33  -- Add keys for table BusinessTerm
34
35  ALTER TABLE [BusinessTerm] ADD CONSTRAINT [PK_BusinessTerm] PRIMARY KEY ([BusinessId])
36  go
37
```

```
-- Table Nodes

CREATE TABLE [Nodes]
(
 [NodeId] Int IDENTITY(1,1) NOT NULL,
 [Label] Varchar(50) NOT NULL,
 [Counts] Int NOT NULL,
 [DBName] Varchar(100) NOT NULL
)
go

-- Add keys for table Nodes

ALTER TABLE [Nodes] ADD CONSTRAINT [PK_Nodes] PRIMARY KEY ([NodeId],[DBName])
go

-- Table Property

CREATE TABLE [Property]
(
 [PropertyId] Int IDENTITY(10,1) NOT NULL,
 [TechTerm] Varchar(max) NOT NULL,
 [UniqueConstraints] Varchar(50) NOT NULL,
 [ExistingConstraints] Varchar(50) NOT NULL,
 [NodeId] Int NOT NULL,
 [DBName] Varchar(100) NOT NULL
)
go

-- Add keys for table Property

ALTER TABLE [Property] ADD CONSTRAINT [PK_Property] PRIMARY KEY ([PropertyId],[NodeId],[DBName])
go

-- Table Relationship

CREATE TABLE [Relationship]
(
 [ChildNode] Varchar(50) NOT NULL,
 [RelDesc] Varchar(100) NOT NULL,
 [NodeId] Int NOT NULL,
 [DBName] Varchar(100) NOT NULL
)
go

-- Add keys for table Relationship

ALTER TABLE [Relationship] ADD CONSTRAINT [PK_Relationship] PRIMARY KEY ([ChildNode],[NodeId],[DBName])
go

-- Table DomBusBrdge

CREATE TABLE [DomBusBrdge]
(
 [DBName] Varchar(100) NOT NULL,
 [BusinessId] Int NOT NULL
)
go

-- Add keys for table DomBusBrdge

ALTER TABLE [DomBusBrdge] ADD CONSTRAINT [PK_DomBusBrdge] PRIMARY KEY ([DBName],[BusinessId])
go
```

```
-- Table BusPropBridge

CREATE TABLE [BusPropBridge]
(
 [PropertyId] Int NOT NULL,
 [NodeId] Int NOT NULL,
 [BusinessId] Int NOT NULL,
 [DBName] Varchar(100) NOT NULL
)
go

-- Add keys for table BusPropBridge

ALTER TABLE [BusPropBridge] ADD CONSTRAINT [PK_BusPropBridge] PRIMARY KEY ([PropertyId],[NodeId],[BusinessId],[DBName])
go

-- Create foreign keys (relationships) section -------------------------------------------------


ALTER TABLE [Relationship] ADD CONSTRAINT [links] FOREIGN KEY ([NodeId], [DBName]) REFERENCES [Nodes] ([NodeId], [DBName])
ON UPDATE NO ACTION ON DELETE NO ACTION
go



ALTER TABLE [DomBusBrdge] ADD CONSTRAINT [contains] FOREIGN KEY ([DBName]) REFERENCES [Domain] ([DBName])
ON UPDATE NO ACTION ON DELETE NO ACTION
go



ALTER TABLE [DomBusBrdge] ADD CONSTRAINT [associate] FOREIGN KEY ([BusinessId]) REFERENCES [BusinessTerm] ([BusinessId])
ON UPDATE NO ACTION ON DELETE NO ACTION
go


ALTER TABLE [Property] ADD CONSTRAINT [Relationship52] FOREIGN KEY ([NodeId], [DBName]) REFERENCES [Nodes] ([NodeId], [DBName])
ON UPDATE NO ACTION ON DELETE NO ACTION
go



ALTER TABLE [BusPropBridge] ADD CONSTRAINT [Relationship53] FOREIGN KEY ([PropertyId], [NodeId], [DBName])
REFERENCES [Property] ([PropertyId], [NodeId], [DBName])
ON UPDATE NO ACTION ON DELETE NO ACTION
go


ALTER TABLE [BusPropBridge] ADD CONSTRAINT [Relationship54] FOREIGN KEY ([BusinessId]) REFERENCES [BusinessTerm] ([BusinessId])
ON UPDATE NO ACTION ON DELETE NO ACTION
go



ALTER TABLE [Nodes] ADD CONSTRAINT [exists] FOREIGN KEY ([DBName]) REFERENCES [Domain] ([DBName])
ON UPDATE NO ACTION ON DELETE NO ACTION
go
```

2.3. Integration

Here we describe how the data is integrated into the database.

To perform data integration the integration team have followed the following steps.

1. Data from each individual team was gathered both for business and technical metadata.

2. Business data is filled by the teams manually in a google sheet explaining all the terms they have and their description
3. Technical data is imported to a google sheet by each team using the script as part of Data extraction
4. Once all the data is in the sheets, we used google google_spreadsheet, google-auth-oauthlib, pygsheets, py2neo, pandas libraries in python, and we got the data into pandas data frames, one for business metadata and one for technical metadata.

Here is the code used to import the data.

```
import pandas as pd
from googleapiclient.discovery import build
from google_auth_oauthlib.flow import InstalledAppFlow,Flow
from google.auth.transport.requests import Request
import os
import pickle

SCOPES = ['https://www.googleapis.com/auth/spreadsheets']

# here enter the id of your google sheet
SAMPLE_SPREADSHEET_ID_input = '18YdO9N_KmMV57-l_2utxQ-i5AH2A66XCEnDV1Lyb_zs'
SAMPLE_RANGE_NAME = 'Final!A1:D'

def main():
    global values_input, service
    creds = None
    if os.path.exists('token.pickle'):
        with open('token.pickle', 'rb') as token:
            creds = pickle.load(token)
    if not creds or not creds.valid:
        if creds and creds.expired and creds.refresh_token:
            creds.refresh(Request())
        else:
            flow = InstalledAppFlow.from_client_secrets_file(
                'credentials.json', SCOPES) # here enter the name of your downloaded JSON file
            creds = flow.run_local_server(port=0)
        with open('token.pickle', 'wb') as token:
            pickle.dump(creds, token)

    service = build('sheets', 'v4', credentials=creds)

    # Call the Sheets API
    sheet = service.spreadsheets()
    result_input = sheet.values().get(spreadsheetId=SAMPLE_SPREADSHEET_ID_input,
                                      range=SAMPLE_RANGE_NAME).execute()
    values_input = result_input.get('values', [])

    if not values_input and not values_expansion:
        print('No data found.')

main()

df=pd.DataFrame(values_input[1:], columns=values_input[0])
```

Here we connect to the google sheets API using the token and credentials file. Once that is done we import the data into a dataframe.

	DomainName	DBName	BusinessTerm_Name	Term_Description
0	Team1	WeatherReport	Event ID	Number assigned per Event occured
1	Team1	WeatherReport	Event Type	Type of the event like rain , snow
2	Team1	WeatherReport	Event Severity	Metrics do describe how severe the event
3	Team1	WeatherReport	Event Start Time	Time stamp for date when event has occured
4	Team1	WeatherReport	Event End Time	Time stamp for date when event has completed
...	...	...	...	...
110	Team8	USPermVisas	Wage Amount	Prevailing wage for the job being requested fo...
111	Team8	USPermVisas	Wage Level	Level of the prevailing wage determination. V...
112	Team8	USPermVisas	Wage Range	Derived Column where Wage Amount is split into...
113	Team8	USPermVisas	Job Title	Industry title associated with the North Ameri...
114	Team8	USPermVisas	Job Sector	Major economic sector associated with the NAIC...

115 rows × 4 columns

The following steps were followed for the business and technical metadata tables.

Populate data into Business metadata tables:
1. For each table, we have taken the needed columns from the data frame and has put it in a data frame
2. By iterating each row of the data frame, the columns in the table are populated. This is done using pyodbc to connect to MSSQL server from python.

Technical metadata tables:
1. A similar approach is followed for technical metadata also, the data was extracted from the csv files and converted into data frame.
2. SQL queries template were written to insert specific data into the tables in MSSQL

```
#For using mssql
import pyodbc

connection = pyodbc.connect('DRIVER={SQL Server Native Client 11.0};SERVER=PV-DESKTOP\SQLSERVERDEV19;DATABASE=Project_
Metadata;UID=pv2612;PWD=classproject')
```

One the connection is successful, we extract the tables

```
('Project_Metadata', 'dbo', 'Domain', 'BASE TABLE')
('Project_Metadata', 'dbo', 'BusinessTerm', 'BASE TABLE')
('Project_Metadata', 'dbo', 'Nodes', 'BASE TABLE')
('Project_Metadata', 'dbo', 'Property', 'BASE TABLE')
('Project_Metadata', 'dbo', 'Relationship', 'BASE TABLE')
('Project_Metadata', 'dbo', 'DomBusBrdge', 'BASE TABLE')
('Project_Metadata', 'dbo', 'Entity25', 'BASE TABLE')
```

From that we extract the domain table.

```
df_sub = df[['DBName','DomainName']]
domList = df_sub.drop_duplicates().reset_index(drop=True)
domList
```

	DBName	DomainName
0	WeatherReport	Team1
1	HotelReview	Team5
2	GlobalTradeStats	Team6
3	BuildingPermits	Team7
4	USPermVisas	Team8

The business terms table is then compiled

```
df_bus = df[['BusinessTerm_Name','Term_Description']]
BusList = df_bus.drop_duplicates().reset_index(drop=True)
BusList
```

	BusinessTerm_Name	Term_Description
0	Event ID	Number assigned per Event occured
1	Event Type	Type of the event like rain , snow
2	Event Severity	Metrics do describe how severe the event
3	Event Start Time	Time stamp for date when event has occured
4	Event End Time	Time stamp for date when event has completed
...	...	...
110	Wage Amount	Prevailing wage for the job being requested fo...
111	Wage Level	Level of the prevailing wage determination. V...
112	Wage Range	Derived Column where Wage Amount is split into...
113	Job Title	Industry title associated with the North Ameri...
114	Job Sector	Major economic sector associated with the NAIC...

115 rows × 2 columns

We then insert the business metadata in to the mssql database.

```
query = "Insert into BusinessTerm (BusType, BusinessDesc) values(?,?)";
for row in BusList.iterrows():
    val = (row[1][0], row[1][1]);
    cursor.execute(query,val);
connection.commit()
```

Upon executing the cursor query we get this output.

```
100 Number assigned per Event occured Event ID
101 Type of the event like rain , snow Event Type
102 Metrics do describe how severe the event  Event Severity
103 Time stamp for date when event has occured Event Start Time
104 Time stamp for date when event has completed Event End Time
105 The US-Based time zone based on the location of the event Time-Zone
106 The airport station that a weather event is reported from. Airport Code
107 The latitude in GPS coordinate of airport-based weather station. Latitude Location
108 The longitude in GPS coordinate of airport-based weather station. Longitude Location
109 The cityof airport-based weather station. City
110 The county of airport-based weather station. County
111 The state of airport-based weather station. State
112 The zip code of airport-based weather station. Zip Code
113 The duration in minutes  Duration
114 Address of hotel Hotel_Address
115 There are also some guests who just made a scoring on the service rather than a review. This number indicates how
many valid scores without review in there Additional_Number_of_Scoring
116 Average Score of the hotel, calculated based on the latest comment in the last year Average_Score
117 Name of Hotel Hotel_Name
118 Total number of valid reviews the hotel has Total_Number_of_Reviews
```

We then insert the domain terms relationship table.

```
query = "Insert into DomBusBrdge (DBName, BusinessId) values(?,?)";
for row in df.iterrows():
    domainName = row[1][0]
    busTermName = row[1][2]
    domId = get_key(domainName,domain)
    busId = get_key(busTermName,busterms)
    val = (domId,busId)
    cursor.execute(query,val)
connection.commit()
```

We then extract the data from the google sheet to link it with nodes and relationship.

This dataframe shows the data from all teams.

	counts	label	property	type	isIndexed	uniqueConstraint	existenceConstraint	team	dbName	rel
0	6274206	EVENT	StartTime	DATE_TIME	FALSE	FALSE	FALSE	1	Weather Database	COUNTY_OF,IS_OF,WITH,SENSOR_AT,CITY_OF,
1	6274206	EVENT	TimeZone	STRING	FALSE	FALSE	FALSE	1	Weather Database	COUNTY_OF,IS_OF,WITH,SENSOR_AT,CITY_OF,
2	6274206	EVENT	Duration	INTEGER	FALSE	FALSE	FALSE	1	Weather Database	COUNTY_OF,IS_OF,WITH,SENSOR_AT,CITY_OF,
3	6274206	EVENT	Id	STRING	TRUE	TRUE	FALSE	1	Weather Database	COUNTY_OF,IS_OF,WITH,SENSOR_AT,CITY_OF,
4	6274206	EVENT	EndTime	DATE_TIME	FALSE	FALSE	FALSE	1	Weather Database	COUNTY_OF,IS_OF,WITH,SENSOR_AT,CITY_OF,
...	...	...	...	...	...	...	...	...	...	
1130	6833	City	name	STRING	TRUE	TRUE	FALSE	8	USPermVisas	L
1131	58	State	name	STRING	TRUE	TRUE	FALSE	8	USPermVisas	L
1132	10	Wage	range	STRING	TRUE	TRUE	FALSE	8	USPermVisas	
1133	10	Wage	level	STRING	FALSE	FALSE	FALSE	8	USPermVisas	
1134	18	Sector	name	STRING	TRUE	TRUE	FALSE	8	USPermVisas	

1135 rows × 10 columns

We then insert the nodes table.

```
df_node = df[['counts','label','dbName']]
nodeList = df_node.drop_duplicates().reset_index(drop=True)
nodeList
```

	counts	label	dbName
0	6274206	EVENT	Weather Database
1	7	TYPE	Weather Database
2	6	SEVERITY	Weather Database
3	2071	AIRPORT	Weather Database
4	1715	CITY	Weather Database
5	48	STATE	Weather Database
6	2021	ZIPCODE	Weather Database
7	1100	COUNTY	Weather Database
8	224	Country	COVID19
9	966	Province	COVID19
10	403	Date	COVID19
11	236017	InfectionStatus	COVID19

We follow similar steps to insert data into properties table.

We execute the following code to insert the data into properties table.

```
# Send data to sql server Property table
query = "Insert into Nodes (PropertyId, TechTerm, UniqueConstraints, ExistingConstraints, NodeId) values(?,?,?,?,?)";
startPropId = 100
for row in propList.iterrows():
    startPropId = startPropId+1
    val = (startPropId, row[1][0], row[1][1], row[1][2], get_key(row[1][3],nodeDict));
    cursor.execute(query,val);
connection.commit()
```

We then execute the query to insert the relationship table.

```
# Send data to sql server Relationship table
query = "Insert into Relationship (RelationId, RelDesc, NodeId) values(?,?,?)";
startRelId = 1000
for row in relList.iterrows():
    startRelId = startRelId+1
    val = (startRelId, row[1][1], get_key(row[1][0],nodeDict));
    cursor.execute(query,val);
connection.commit()
```

2.4. User Interface

For user interface the team has built a front-end application to browse and navigate the technical and business metadata.

Details of the web application:

The application is built using **ReactJS** for front end. React is an open-source, front end, JavaScript library for building user interfaces or UI components.

The backend is built using **Spring Boot**. The Spring Framework is an application framework and inversion of control container for the Java platform

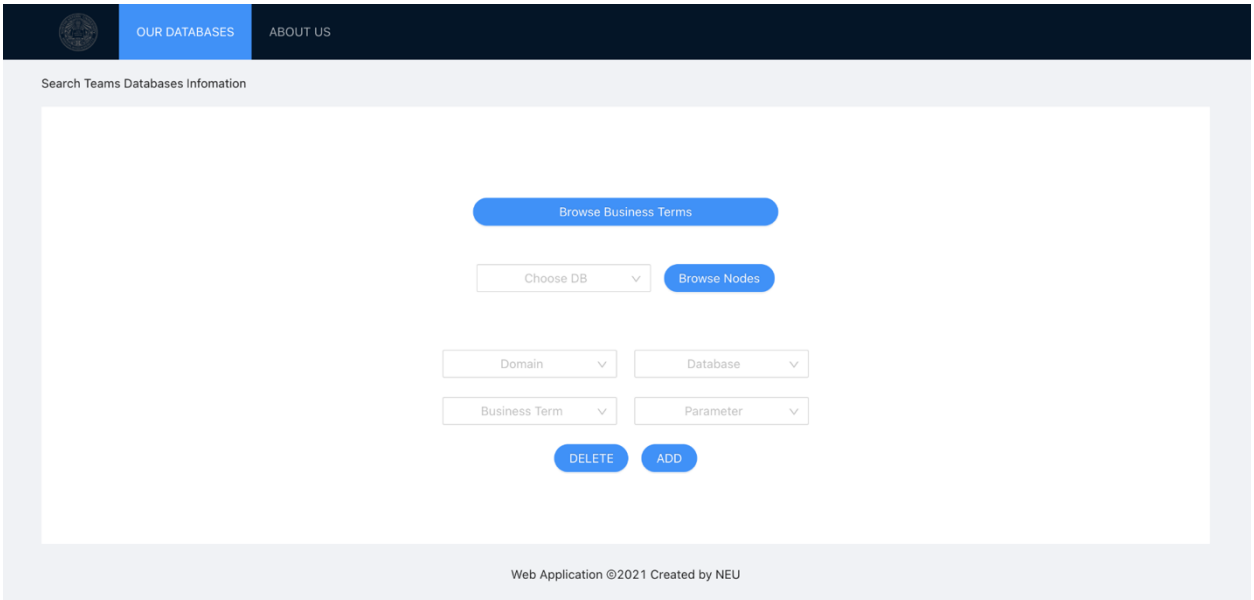
IDE used for application building is IntelliJ for the Spring-Boot backend webapp and Visual Studio Code for the ReactJS side of the application.

Testing was done manually by checking all parts of the web app and reporting bugs to the respective teams.

The application is hosted and need the following IP address to access it.

http://142.173.12.1:8000

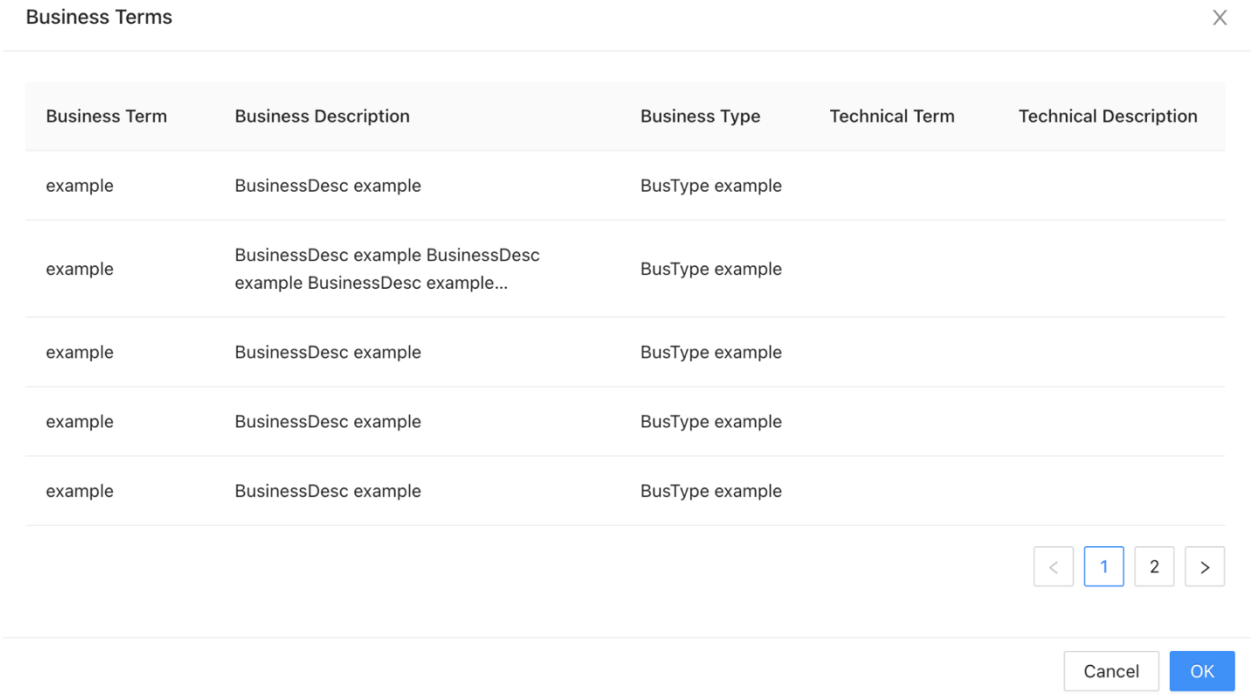
When accessing the URL, we get to the application.



This is the front end application.

To access the metadata we have to use the dropdown menus to select specific data.

The ‘browse business metadata’ gives us the milti page view of all the business metadata with the associated technical terms.



Choose MSSQL to connect the database. Then click on browse nodes to get data on all nodes and properties.

Using the application, we can also browse the business terms and technical terms.

User can also add or delete various parameters such as different terms.