Taylor Shrode
NoSQL Databases
2/10/2020
MSDS 610

Written Portion

1. Why is NoSQL useful? What are use cases for NoSQL?

   "Big Data NoSQL databases were pioneered by top internet companies like Amazon, Google, LinkedIn and Facebook to overcome the drawbacks of relational databases (NoSQL vs SQL)." NoSQL is an alternative to SQL databases where fixed tables are not required. NoSQL databases have multiple different types of databases including Graph, Key-Value stores, Wide-column, and Document (Types of NoSQL).

   One of the main differences between NoSQL and SQL is the nature of its data and its storage. A major issue when dealing with data is encountering times when a user is unable to process data because it is too complex. NoSQL is the better option for this because of its ease to represent multi-hierarchies using JavaScript Object Notation format (JSON) (NoSQL vs SQL). No SQL databases are also much faster when reading or writing operations to a single entity. NoSQL databases can also store and process data in real-time (NoSQL vs SQL). A solutions architect at Pythanian once said that "most people who choose NoSQL as their primary data storage are trying to solve two main problems: scalability and simplifying the development process. (NoSQL vs SQL)"

   NoSQL is preferable to SQL when personalization, profile management, real-time big data, content management, catalogs, mobile applications, digital communications, or fraud detection is needed (Stephan, 2015).

2. What are some of the different NoSQL databases, and in what cases are they useful?

   As mentioned above, NoSQL databases have multiple different types of databases including Graph, Key-Value stores, Wide-column, and Document (Types of NoSQL). Key-Value databases are the simplest because every item in the database is stored as an attribute name (key) together with its value. These are the most useful when processing a collection of key-value pairs. This NoSQL database is the most scalable and simple (Cielen et al., 2016).

   Wide-Column databases store columns of data together rather than rows. Indexing the data on certain columns significantly improves speed (Cielen et al., 2016). Document databases pair each key with a complex data structure known as a document. This document may contain different key-value pairs or nested documents. MongoDB is a document database. Document Stores databases are the most "natural" database because it's designed to store everyday documents as is. This allows for complex querying (Cielen et al., 2016). Finally, a graph database stores information about networks. (Types

of NoSQL) Although Graph databases are the most complex, but stores relations between entities in an efficient manner (Cielen et al., 2016).

3. What are pros and cons of NoSQL vs SQL?

   NoSQL is more about storing data and less about how to query it (Knight, 2017). NoSQl is open source and offers low-cost training, setup, and development. It offers the ability to scale out and work with unstructured and semi-structured data. However, it is weaker in its consistency (BASE vs ACID) (Foote, 2016).

   On the other hand, SQL is used to perform set operations like union and intersect (Knight, 2017). SQL handles structure data and supports ACID transactional consistency. SQL does not scale out well and is much slower than NoSQL because data is normalized (Foote, 2016).

Technical Portion

NoSQL databases are designed to "cope with the scale and agility challenges that face modern applications" (NoSQL Explained). They encompass a wide variety of database technologies that can handle large volumes of all types of data (NoSQL Explained). One of the leading NoSQL databases is MongoDB. MongoDB offers scalability and the ability to process structured, unstructured, and semi-structured data.

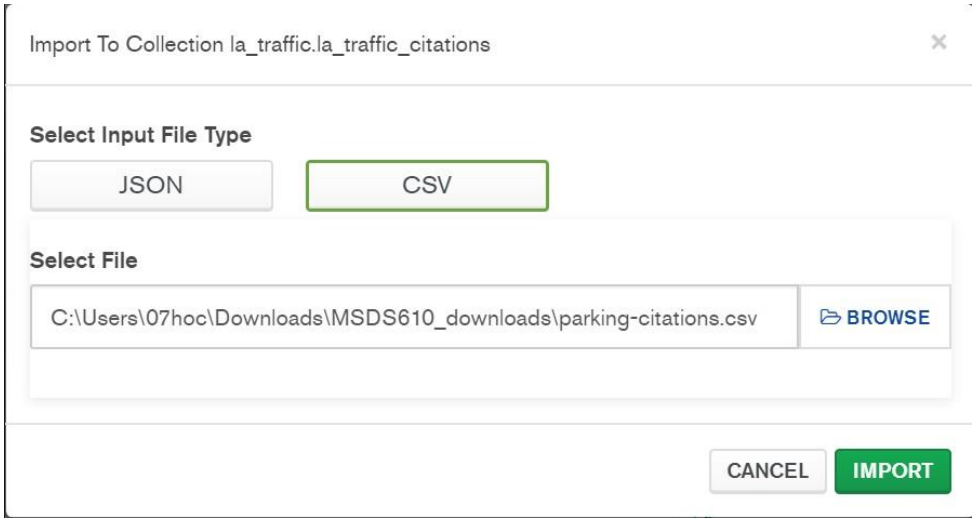
For this assignment, we will be using MongoDB Compass and the Los Angeles Parking Citations dataset. This dataset contains information on parking violations in the city of Los Angeles, CA. The columns of this dataset include the following:

- Ticket Number
- Issue Date
- Issue Time
- Meter ID
- Marked Time
- State Plate
- Plate Expiration
- VIN
- Make
- Body Style
- Color
- Location
- Route
- Agency
- Violation Code
- Violation Description
- Fine Amount
- Latitude
- Longitude
- Color Description

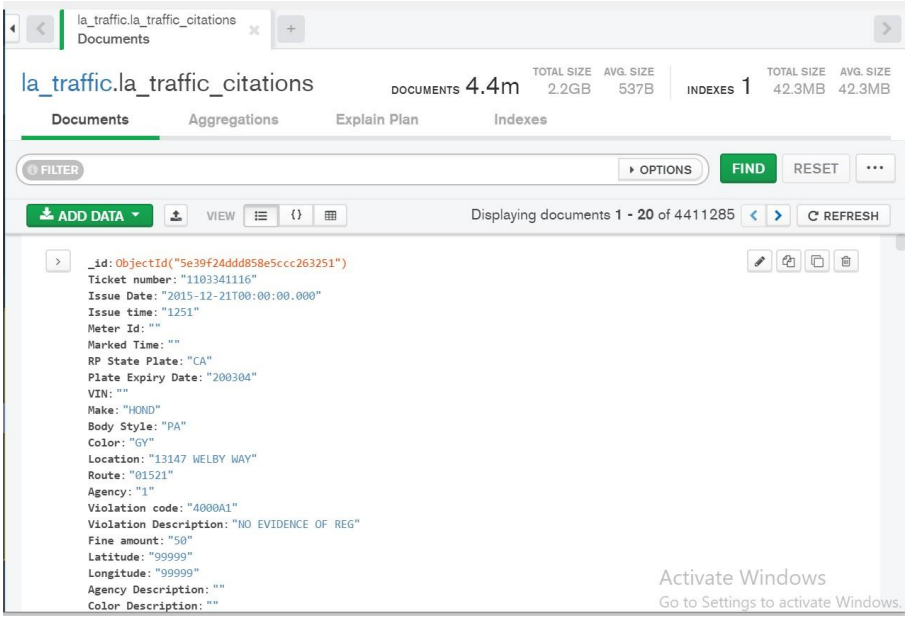
Using the Los Angeles Parking Citations data set, we will be using MongoDB queries to complete tasks such as extracting the most expensive tickets, connecting to the MongoDB using Python, filtering data by state license plates, and creating sorted bar charts in python.

Taylor Shrode
NoSQL Databases
2/10/2020
MSDS 610

To begin, MongoDB and MongoDB Compass (GUI) needs to be installed. We will install the MongoDB Community Edition for Windows here. For the MongoDB installer, the 4.2.3 current version for Windows 64-bit X64 will be downloaded. After completing the MongoDB setup, we can start the MongoDB Compass Client. Then, we can create a new database called **la_traffic** and the Collection within the database we will name **la_traffic_citations**. To import our Los Angeles Parking Citations data from above, we click on the name of the collection, then click on the upper menu "Collection", and then import data. In this case, we will be importing the CSV file.

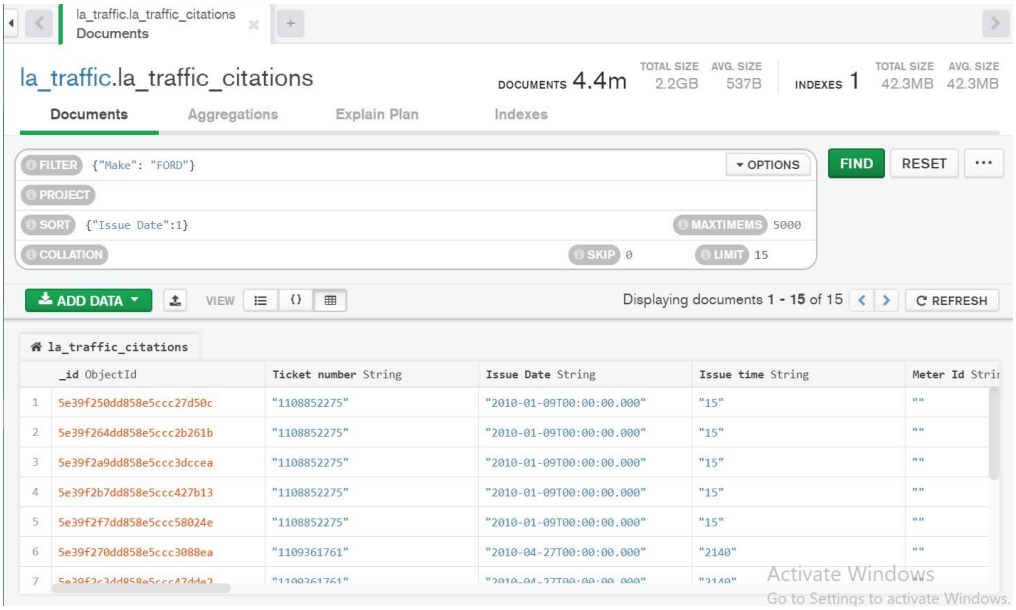


A timeout message did occur, but I was able to load 4.4 million rows of data. As seen below.



Taylor Shrode
NoSQL Databases
2/10/2020
MSDS 610

To test the Query bar, we can filter on the Make of “Ford”, sort in ascending order, and limit to 15 rows returned. As seen below.



To find the amounts and violation descriptions for the top 10 most expensive tickets, the empty “Fine amount” entries need to be filtered out by using the command **{“Fine amount”: {$ne: “”}}.** After this, I would sort “Fine amount” by ascending order and set the “limit” to 10. Unfortunately, I was unable to get this portion of the assignment to work. I believe it was because the entries in “Fine amount” are strings and the command **{“Fine amount”: 1}** does not compare numbers as a string the same way as actual numbers. For example, “100” is less than “2” because 1 is before 2 alphanumerically.

Now, we connect to MongoDB with Python using the Jupyter Notebook that is installed with Anaconda. After launching the Jupyter Notebook, we use the command **conda install pymongo** so we can work with MongoDB using Python.

Taylor Shrode
NoSQL Databases
2/10/2020
MSDS 610

```
conda install pymongo
```

```
Collecting package metadata (current_repodata.json): ...working... done
Solving environment: ...working... done

## Package Plan ##

  environment location: C:\Users\07hoc\Anaconda3

  added / updated specs:
    - pymongo


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    conda-4.8.2                |           py37_0         2.8 MB
    pymongo-3.9.0              |   py37ha925a31_0         1.1 MB
    ------------------------------------------------------------
                                           Total:         4.0 MB

The following NEW packages will be INSTALLED:

  pymongo            pkgs/main/win-64::pymongo-3.9.0-py37ha925a31_0
```

With **pymongo** installed, we can import **MongoClient** which allows us to connect to MongoDB. Next, we create our connection from Python to MongoDB by connecting to the database, **la_traffic**, and to the Collection, **la_traffic_citations**.

In [2]:
```
from pymongo import MongoClient
```

In [3]:
```
#create connection
client = MongoClient()
db = client['la_traffic']
tickets = db['la_traffic_citations']
```

In [5]:
```
type(tickets)
```

Out[5]:
```
pymongo.collection.Collection
```

In [4]:
```
tickets.find_one()
```

The **tickets.find_one()** command allows us to look at an entry located in our Collection

Taylor Shrode
NoSQL Databases
2/10/2020
MSDS 610

```
In [4]: tickets.find_one()

Out[4]: {'_id': ObjectId('5e39f24ddd858e5ccc263251'),
         'Ticket number': '1103341116',
         'Issue Date': '2015-12-21T00:00:00.000',
         'Issue time': '1251',
         'Meter Id': '',
         'Marked Time': '',
         'RP State Plate': 'CA',
         'Plate Expiry Date': '200304',
         'VIN': '',
         'Make': 'HOND',
         'Body Style': 'PA',
         'Color': 'GY',
         'Location': '13147 WELBY WAY',
         'Route': '01521',
         'Agency': '1',
         'Violation code': '4000A1',
         'Violation Description': 'NO EVIDENCE OF REG',
         'Fine amount': '50',
         'Latitude': '99999',
         'Longitude': '99999',
         'Agency Description': '',
         'Color Description': '',
         'Body Style Description': ''}
```

To filter data, we can query the data similarly to MongoDB. Since this returns a **pymongo.cursor.Cursor** datatype, we can access it like a Python list. As seen below.

```
In [7]: registration_tickets = tickets.find({'Violation Description': {'$eq': 'NO EVIDENCE OF REG'}}).limit(10)

In [8]: registration_tickets[4]

Out[8]: {'_id': ObjectId('5e39f24ddd858e5ccc263262'),
         'Ticket number': '1107539823',
         'Issue Date': '2015-09-16T00:00:00.000',
         'Issue time': '2120',
         'Meter Id': '',
         'Marked Time': '',
         'RP State Plate': 'CA',
         'Plate Expiry Date': '201502',
         'VIN': '',
         'Make': 'NISS',
         'Body Style': 'PA',
         'Color': '',
         'Location': 'BLAINE/11TH PL',
         'Route': '1FB95',
         'Agency': '1',
         'Violation code': '4000A1',
         'Violation Description': 'NO EVIDENCE OF REG',
         'Fine amount': '50',
         'Latitude': '99999',
         'Longitude': '99999',
         'Agency Description': '',
         'Color Description': '',
         'Body Style Description': ''}
```

By default, MongoDB returns the **_id** column. We can filter it out using **_id : 0**. Below, we will only return the “Make” entry for each datapoint that has a “Violation Description” of ‘NO EVIDENCE OF REG’.

```
In [16]: registration_tickets_makes = tickets.find({'Violation Description':
                                                    {'$eq': 'NO EVIDENCE OF REG'}},
                                                   {'Make': 1, '_id': 0})

In [17]: registration_tickets_makes[0], registration_tickets_makes[1],registration_tickets_makes[3], registration_tickets_makes[4],reg

Out[17]: ({'Make': 'HOND'},
          {'Make': 'GMC'},
          {'Make': 'BMW'},
          {'Make': 'NISS'},
          {'Make': 'NISS'})

In [18]: type(registration_tickets), type(registration_tickets_makes)

Out[18]: (pymongo.cursor.Cursor, pymongo.cursor.Cursor)
```

Taylor Shrode
NoSQL Databases
2/10/2020
MSDS 610

Now, in order to make a barplot, we are going to use pandas and matplotlib.

```
In [19]: import pandas as pd
         import matplotlib.pyplot as plt
         %matplotlib inline
```

After importing pandas and matplotlib, we convert that data from Mongo, which is in a list where each entry is in JSON format. To view the data, the **.head()** command is used.

```
In [32]: registration_tickets_makes = tickets.find({'Violation Description': {'$eq': 'NO EVIDENCE OF REG'}},
                                                  {'Make': 1, '_id': 0})

In [33]: df_1= pd.io.json.json_normalize(registration_tickets_makes)

In [40]: df_1.head()

Out[40]:
```

	Make
0	GMC
1	CHEV
2	BMW
3	NISS
4	NISS

We can also use **.value_counts()** to count the number of each "Make" in our converted data.

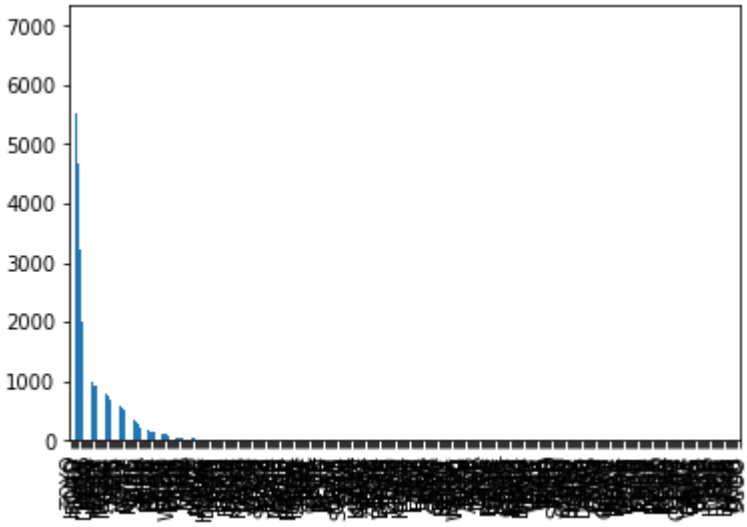
```
In [34]: df_1['Make'].value_counts()

Out[34]: TOYO    7001
         HOND    5586
         FORD    5533
         CHEV    4660
         NISS    3230
                 ...
         AB         1
         BGSR       1
         FMC        1
         OVLD       1
         SCOO       1
         Name: Make, Length: 288, dtype: int64
```

Now, we can use the convenience of pandas to create our barplot using the command below.

```
In [36]: ax = df_1['Make'].value_counts().plot.bar()
```

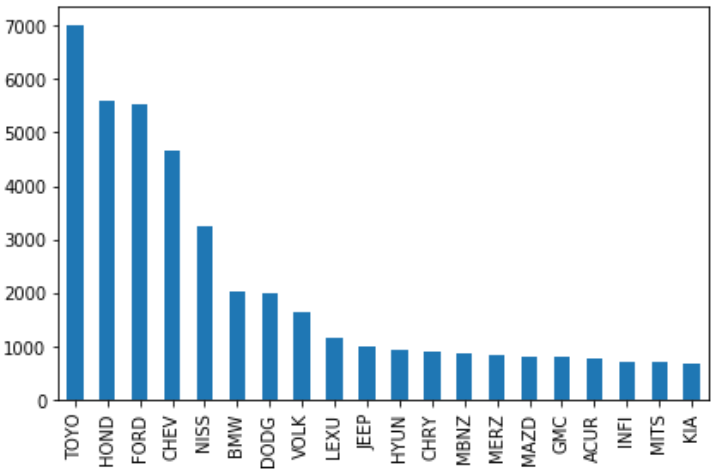
The result of that command is below.

Taylor Shrode
NoSQL Databases
2/10/2020
MSDS 610



Notice how "messy" the barplot is. We can fix this by using the **tight_layout()** command, and limit the number of "Makes" in the barplot.

```
In [37]: ax = df_1['Make'].value_counts()[:20].plot.bar()
         plt.tight_layout()
```



From this, we can easily see that Toyotas have the most parking violations labeled as "NO EVIDENCE OF REG". We can also filter the data to see which state license plates, excluding California, have the most parking citations.

```
In [45]: license_not_CA = tickets.find({'RP State Plate': {'$ne': 'CA'}}).limit(10)

In [46]: license_not_CA[0]

Out[46]: {'_id': ObjectId('5e39f24ddd858e5ccc263265'),
 'Ticket number': '1107780822',
 'Issue Date': '2015-12-22T00:00:00.000',
 'Issue time': '1105',
 'Meter Id': '',
 'Marked Time': '',
 'RP State Plate': 'FL',
 'Plate Expiry Date': '201611',
 'VIN': '',
 'Make': 'FORD',
 'Body Style': 'PA',
 'Color': 'WH',
 'Location': '',
 'Route': '2A1',
 'Agency': '1',
 'Violation code': '8069B',
 'Violation Description': 'NO PARKING',
 'Fine amount': '73',
 'Latitude': '99999',
 'Longitude': '99999',
 'Agency Description': '',
 'Color Description': '',
 'Body Style Description': ''}
```

Similar to above, we need to convert the data.

```
In [51]: license_not_CA = tickets.find({'RP State Plate': {'$ne': 'CA'}})

In [52]: df_3 = pd.io.json.json_normalize(license_not_CA)

In [53]: df_3.head()

Out[53]:
```

	_id	Ticket number	Issue Date	Issue time	Meter Id	Marked Time	RP State Plate	Plate Expiry Date	VIN	Make	...	Route	Agency	Violation code	Violation Description
0	5e39f24ddd858e5ccc26327c	1109455266	2015-12-26T00:00:00.000	1815			NY			TOYO	...		36	8603	PK IN PROH AREA
1	5e39f24ddd858e5ccc263286	1109704934	2015-12-28T00:00:00.000	847			OR	200316		SUBA	...		54	8069BS	NO PARK/STREET CLEAN
2	5e39f24ddd858e5ccc2632a6	1111884093	2015-12-12T00:00:00.000	938			WA	201505		VOLK	...	1A4	11	4000A1	NO EVIDENCE OF REG
3	5e39f24ddd858e5ccc2632aa	1111884130	2015-12-19T00:00:00.000	1200			NJ	201212		SUBA	...	1A4	11	4000A1	NO EVIDENCE OF REG
4	5e39f24ddd858e5ccc2632ae	1111927526	2015-12-21T00:00:00.000	1000			CO	201403		CHRY	...	1A4	11	002	4000A

5 rows × 23 columns

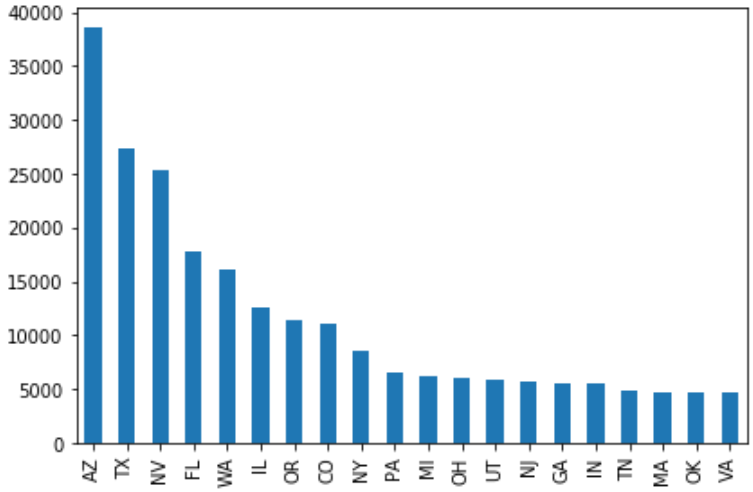
Now, we can view the count of parking citations for each state license plate.

```
In [54]: df_3['RP State Plate'].value_counts()

Out[54]: AZ    38577
         TX    27410
         NV    25308
         FL    17816
         WA    16040
               ...
         PE       11
         YU        9
         PR        7
         NW        6
         NF        3
         Name: RP State Plate, Length: 75, dtype: int64
```

Taylor Shrode
NoSQL Databases
2/10/2020
MSDS 610

Now, creating our barplot with a tight layout,

```
In [58]: ax = df_3['RP State Plate'].value_counts()[:20].plot.bar()
         plt.tight_layout()
```



We can see that Arizona has the most parking citations in Los Angeles. Finally, we close our connection to the database.

```
In [59]: client.close()
```

As we have seen above, we were able to import a CSV into MongoDB, use the MongoDB shell to query and subset data, connect to the MongoDB from Python, and query data from the MongoDB to Python and plot the results.

Taylor Shrode
NoSQL Databases
2/10/2020
MSDS 610

Resources

Cielen, D., Meysman, A. D. B., & Ali, M. (2016, March 5). *Nosql database types—Dzone database*. Dzone.Com. https://dzone.com/articles/nosql-database-types-1

Foote, K. D. (2016, December 21). A review of different database types: Relational versus non-relational. *DATAVERSITY*. https://www.dataversity.net/review-pros-cons-different-databases-relational-versus-non-relational/

Knight, M. (2017a, August 21). What is structured query language (Sql)? *DATAVERSITY*. https://www.dataversity.net/structured-query-language-sql/

Knight, M. (2017b, September 4). What is NoSQL? *DATAVERSITY*. https://www.dataversity.net/what-is-nosql/

Nosql databases explained. (n.d.). MongoDB. Retrieved February 9, 2020, from https://www.mongodb.com/nosql-explained

Nosql vs sql- 4 reasons why nosql is better for big data applications. (2015, March 19). https://www.dezyre.com/article/nosql-vs-sql-4-reasons-why-nosql-is-better-for-big-data-applications/86

Stephan, T. (2015, October 30). *10 use cases where NoSQL will outperform SQL*. Network World. https://www.networkworld.com/article/2999856/10-use-cases-where-nosql-will-outperform-sql.html

Types of nosql database management systems. (n.d.). MongoDB. Retrieved February 11, 2020, from https://www.mongodb.com/scale/types-of-nosql-database-management-system