ISA 414 – Managing Big Data

Lecture 11 – Data Collection

Querying Document-Oriented Databases

Dr. Arthur Carvalho

arthur.carvalho@miamioh.edu

MIAMI UNIVERSITY

Lecture Objectives

- Quick review of Assignment 2

- Learn about document-oriented databases
  - MongoDB
    - How a database can store data in JSON format

Lecture Instructions (Part I)

- Download the notebook "*Lecture 11.ipynb*" available on Canvas
- Open the file "*Lecture 11.ipynb*" with VS Code
- Download the file *cars.csv*
  - Add this file to same folder as "*Lecture 11.ipynb*"

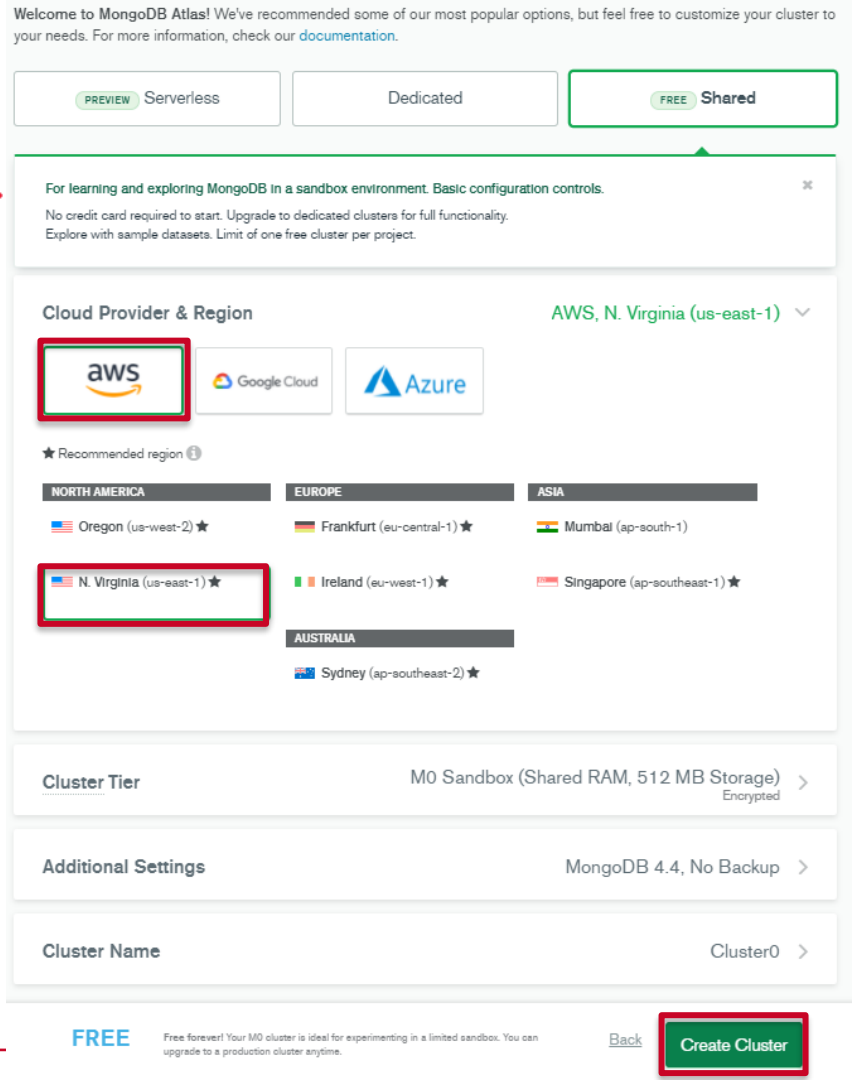
Lecture Instructions (Part II)

- Let's create a MongoDB account on the cloud
  - A MongoDB database in the cloud
    - Go to https://www.mongodb.com/cloud/atlas
      - Click on "Get started now"
      - Fill in the fields
        - Memorize your password and username
      - Verify your email
      - Answer the survey questions
      - Select the Shared (free) option
        - Click on "Create"

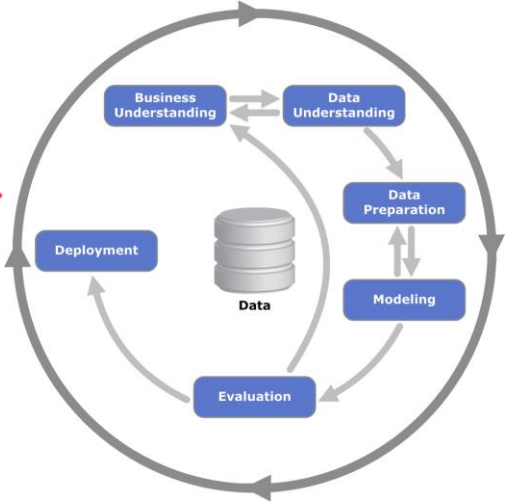
MongoDB

- You should see a screen like this ->
  - Select “AWS” as the cloud provider
  - Select “North America / N. Virginia” as region
  - Click on “Create Cluster”

- You will understand what we are doing here in the future



Data Collection



- What have we learned up to know?
  - Scrap (unstructured) data from the web
    - Retrieve relevant data from HTML files
      - Regular expressions
  - Collect data via APIs
    - Request: REST + HTTP
    - Response: JSON, XML
  - From previous courses: collect data by querying relational databases (SQL)

API

- Bird's-eye perspective of APIs
  - As we use in this course

User

1) user requests service (data) over the web (web service)

Server

3) the server returns some data (usually in JSON or XML)

2) the API processes the request and usually accesses a database to retrieve the underlying data

Database

API

- When coding an API, an organization must define:
  - How to interpret a request (protocol)
  - How to gather data related to the request
  - How to return the collected data (format)

- If the underlying database is relational and the API response is in JSON, then the API must convert tables to the JSON format
  - Question: can we avoid this extra conversion step by having a database that already stores data using the JSON format, as opposed to tables?
    - Yes: document-oriented databases

Document-Oriented Databases

- Recall that **relational databases** store tables that can be in relationships (see ISA 245)
  - One category of databases
- Today, we focus on **document-oriented databases**
  - Another category of databases
  - Store documents (texts)
  - Highly efficient when storing unstructured (textual) data

Document-Oriented Databases

- There are several relational-database implementations
  - Oracle Express, Microsoft SQL Server, MySQL,…
- Likewise, there are several document-oriented databases
  - MongoDB, CouchDB, Solr, …
- We shall focus on MongoDB
  - A **NoSQL database**

MongoDB

- Uses JSON-like flexible schemas (“designs”)
- Supports NoSQL-based queries
  - Often, a query looks for documents containing certain keys and values
- Powerful features
  - Replication, load balancing, quick data retrieval
- Cofounded by **Dwight Merriman**, Miami ’89

MongoDB

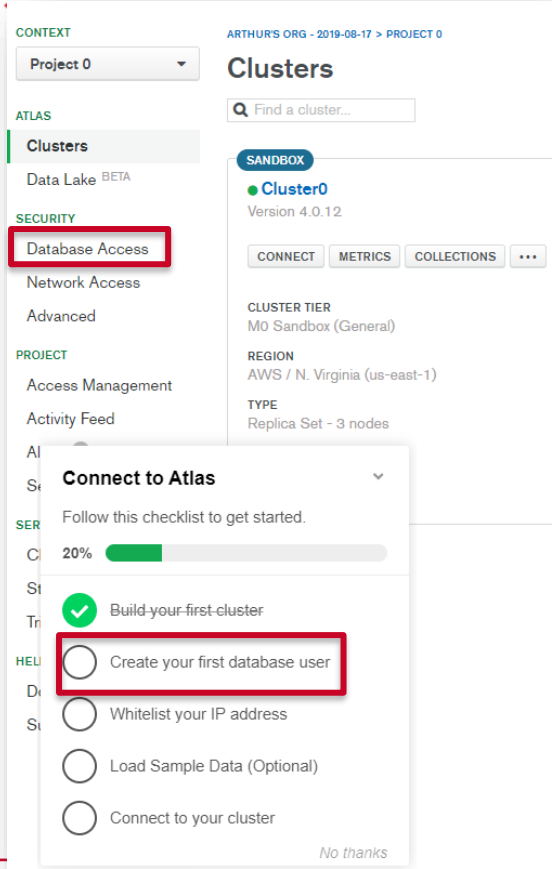
- Let's play with MongoDB
  - When setting up a database, an organization/person has roughly two options
    - Install a database "locally" (on premise)
      - Requires powerful hardware and IT expertise (database administration)
    - Install a database on "someone else's computer" (cloud)
      - Easy process, but one might not know where the data are
  - We will follow the second option so as to have a first contact with cloud storage
    - We further explore this topic in the second part of this course
    - You will see how easy it is to have your own database in the cloud

MongoDB

- We have already created a MongoDB account (Lecture Instructions)
- Each one of us now has one entire database at our disposal
  - We are now database administrators
  - Stored and running on computers owned by Amazon somewhere in the East Coast and managed by MongoDB (the company)
    - No need to deal with complex issues such as replication, load balance, *etc*.
  - Clearly, you could have installed MongoDB on your own machine
    - Choosing between cloud services (*e.g.,* running MongoDB on AWS) and in-house services (*e.g.*, running MongoDB on a company's computer) is a common question IT/IS managers often face

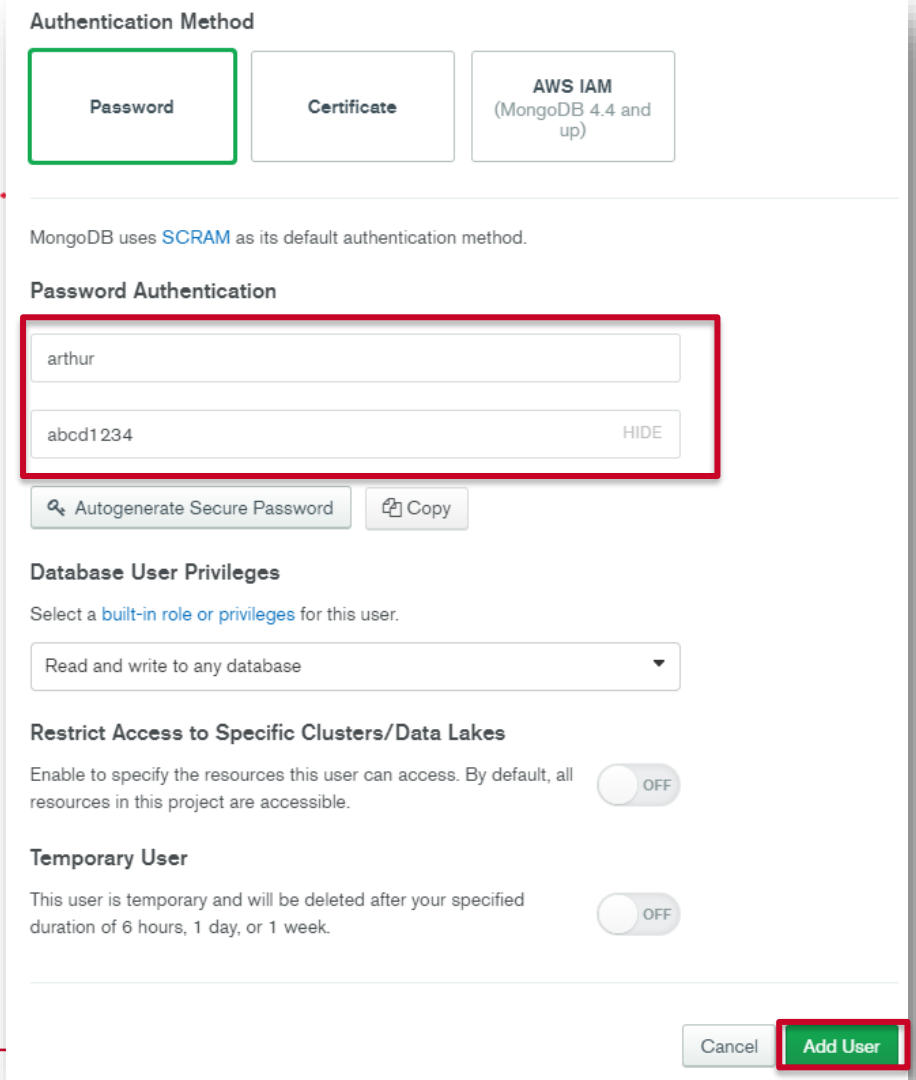
MongoDB

- Next step: let's create a database user
  - Click on "*Database Access*" -> "*Add New Database User*"
  - Select a username and password
    - For the sake of illustration, my username is *arthur* my password *is abcd1234*
      - See next slide
    - Keep in mind that you have two usernames and passwords now
      - One is the administrator of the database system
      - The other is a user of the database



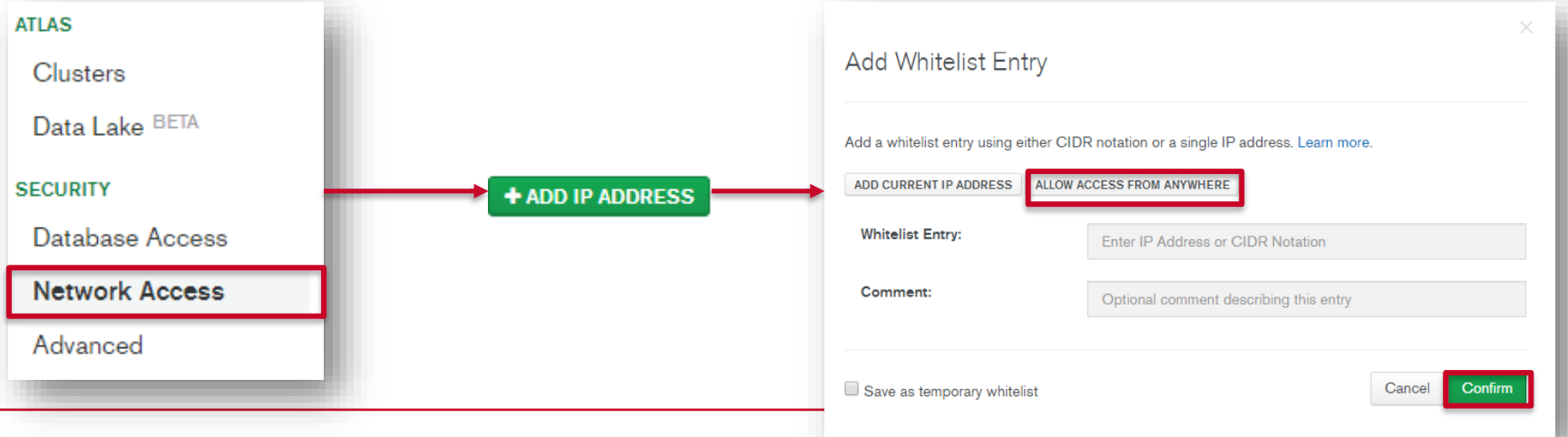
MongoDB

- Creating a database user called *arthur*



MongoDB

- As a security feature, MongoDB (company) requires the database administrator to specify which IP addresses can access the database
  - Click on "*Network Access*" -> "*Add IP Address*" -> "*ALLOW ACCESS FROM ANYWHERE*" -> "*Confirm*"
  - This make take a few seconds



MongoDB

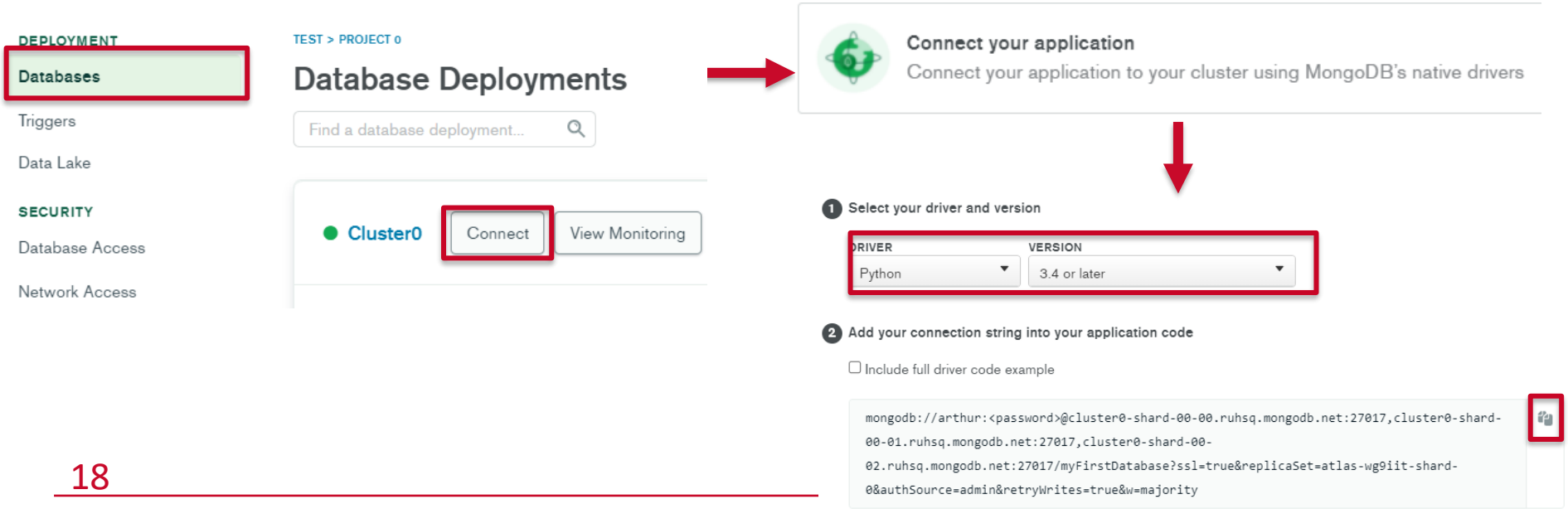
- We now have a database in the cloud and a database user
  - Let’s access and play with the created database from Python
  - Install the required packages (pymongo)
    - Go to the Terminal and type pip install 'pymongo[snappy,gssapi,srv,tls]'
  - We shall use the cars data set

```
import pandas as pd


pd.read_csv("cars.csv")
```

MongoDB

- Let's create a connection to our MongoDB database
  - On cloud.mongodb.com, go to *Clusters* → *Connect* → "*Connect your application*"
  - Select *Driver* = "*Python*" and *Version* = "*3.4 or later*"

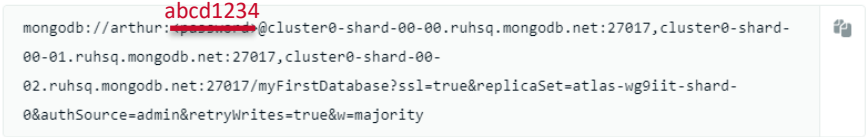


MongoDB

- Let's create a connection to our MongoDB database
  - Now, paste that URL into your code

```
import pymongo
client = pymongo.MongoClient("PASTE THE COPIED STRING HERE")
```

  - **Remember to replace <password> with the database password you created before**
    - For example, my password is *abcd1234*
    - Remember to remove the < > symbols



MongoDB

- Let's create a database called *ISA414*

```
db = client.ISA414
```

- Within that database, let's create a collection *cars*

```
collection = db.cars
```

MongoDB

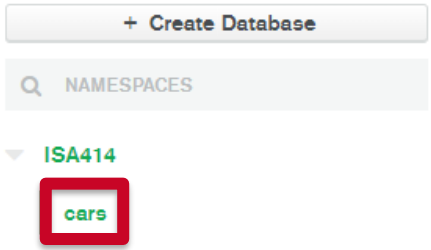
- Let's send our cars data (df) to our database
  - Recall that df is a data frame
  - Let's transform it into a dictionary (JSON)

```
data_json = df.to_dict("records")
```

  - Inserting the data into the collection

```
collection.insert_many(data_json)
```

- Now, go to *Databases* -> "*Browse Collections*"



Yay! We were able to send the "cars" data set from Python to our MongoDB database in the cloud

MongoDB

- Let’s take a look at our database
  - Click on the **collection** named “cars”
  - Note that:
    - Each observation in our data frame becomes a **document**
    - The database is a **collection of documents**


```
_id: ObjectId("60f09819adfb83b8cf2c684b")
name: "Mazda RX4"
mpg: 21
cyl: 6
disp: 160
hp: 110
drat: 3.9
wt: 2.62
qsec: 16.46
vs: 0
am: 1
gear: 4
carb: 4
```

```
_id: ObjectId("60f09819adfb83b8cf2c684c")
name: "Mazda RX4 Wag"
mpg: 21
cyl: 6
disp: 160
hp: 110
drat: 3.9
wt: 2.875
qsec: 17.02
vs: 0
am: 1
gear: 4
carb: 4
```

MongoDB

- Let's take a look at our database

name	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4

```
{
  "_id": ObjectID(
          "5d574bd121590000470023e2"),
  "name": "Mazda RX4"
  "mpg": 21,
  "cyl": 6,
  "disp": 160,
  "hp": 110,
  "drat": 3.9,
  "wt": 2.62,
  "qsec": 16.46,
  "vs": 0,
  "am": 1,
  "gear": 4,
  "carb": 4
}
```

  - Each document has an "id"
    - In our case, it was automatically created
  - Each table's attribute-value becomes a key-value pair

MongoDB

- We are now in a position to understand important facts about MongoDB
  - Based on the concept of *key-value pairs*, crucial NoSQL database concept
  - Document-oriented database
    - Document is often represented as a JSON file
  - Important distinction
    - Relational databases store data in separate *tables* that are defined *ex ante* by the database designer
    - Document-oriented databases store all the data in documents, and one document might have different fields (no need to be predefined)

MongoDB

- Example: relational vs document-oriented database
  - Skip is a Full Professor of ISA
  - Arthur is an Assistant Professor of ISA

ID	Position	Name	Department
1	Full Professor	Skip	ISA
2	Assistant Professor	Arthur	ISA

```
{
   "ID": 1,
   "Position": "Full Professor",
   "Name": "Skip",
   "Department": "ISA"
}
{
   "ID": 2,
   "Position": "Assistant Professor",
   "Name": "Arthur",
   "Department": "ISA“
}
```

MongoDB

- Example: relational vs document-oriented database
  - What if we want to add a second “position” to a table
    - *E.g.*, Skip is also the ISA chair
  - Relational database: redesign the original table
    - Side effect: extra missing values

ID	Position	Position 2	Name	Department
1	Full Professor	Chair	Skip	ISA
2	Assistant Professor		Arthur	ISA

MongoDB

- Example: relational vs document-oriented database
  - Missing values often consume storage space in relational databases
  - For example, suppose the tables are stored as CSV files
    - The addition of a new column to a table means that we need one extra comma per row (extra storage cost = 1 byte)

      2, Assistant Professor,,Arthur,ISA

  - Suppose the underlying table has 1 billion rows
    - Total storage space required by commas: 1 billion bytes = 1 GB
    - That is, adding a new column with no data to a table might drastically increase the required storage space
      - Not much flexibility here

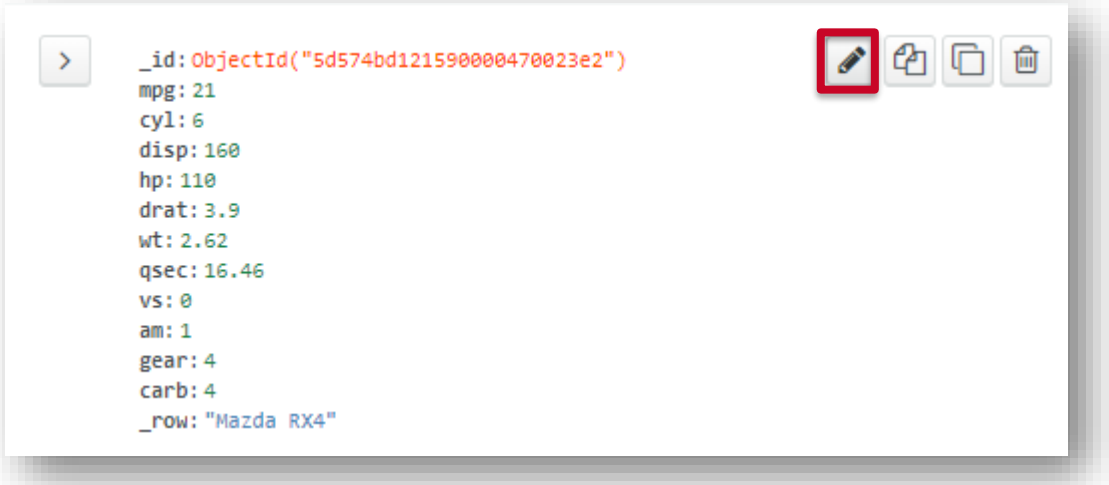
MongoDB

- Example: relational database vs MongoDB
  - What if we want to add a second “position” to a document?
    - *E.g.*, Skip is also the ISA chair
  - MongoDB: simply add a new key-value pair to the first document

```
{
   "ID": 1,
   "Position": "Full Professor",
   "Position 2": "Chair",
   "Name": "Skip",
   "Department": "ISA"
}
{
   "ID": 2,
   "Position": "Assistant Professor",
   "Name": "Arthur",
   "Department": "ISA“
}
```

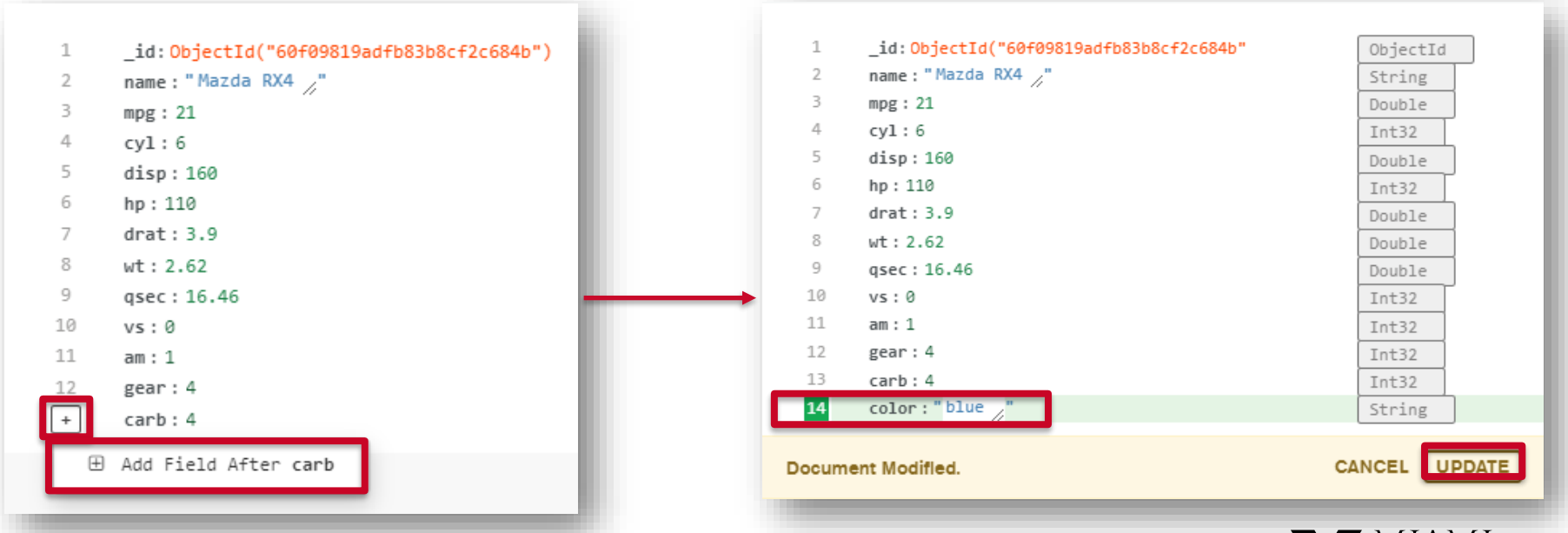
MongoDB

- Illustrating the previous point
  - Let's add a key-value pair to the first car



MongoDB

- Illustrating the previous point



MIAMI UNIVERSITY

MongoDB

- The previously-discussed flexibility makes document-oriented databases very attractive to store textual/unstructured data
  - No need to design a database
  - For example: one can have Facebook posts and tweets inside the same collection of documents
    - Tweet-like documents can have fields such as the number of likes, shares, retweets, *etc.*
    - Facebook-like documents can have fields such as the number of likes, smiley faces, sad faces, hearts, replies, shares, *etc*.

MongoDB

- Querying
  - A database is rather useless if one cannot easily retrieve data from it
  - How to query document-oriented databases?
    - SQL cannot really be used (NoSQL)
  - Idea: search for key-value pairs
    - Our query is actually a JSON “file”
    - Result: JSON “file”

MongoDB

- Querying examples
  - Finding all the cars (documents) containing the key (variable) *mpg* associated with the value 21

```
result = collection.find({"mpg":21})
for document in result:
    print(document)
```

```
{
  "mpg":21
}
```

  - Finding all the cars (documents) containing the *key:value* pairs *mpg:21* **and** *color:blue*

```
result = collection.find({"mpg":21, "color":"blue"})
for document in result:
    print(document)
```

```
{
  "mpg":21,
  "color":"blue"
}
```

MongoDB

- MongoDB has several operators that can be used in queries
  - Finding all the cars (documents) where the variable *mpg* (key) is greater than ($gt) 20 (value)
    - The value associated with *mpg* is "$gt":20

```
result = collection.find({"mpg":{"$gt":20}})
for document in result:
    print(document)
```

```
{
  "mpg":{
        "$gt":20
  }
}
```

  - Finding all the cars (documents) where the variable *mpg* (key) is greater than ($gt) 20, but less than ($lt) 25

```
result = collection.find({"mpg":{"$gt":20, "$lt":25}})
for document in result:
    print(document)
```

```
{
  "mpg":{
        "$gt":20,
        "$lt":25
  }
}
```

MongoDB

- One can also perform update and delete operations with MongoDB
  - Examples:
    - Updating the mpg to 100 for cars named Mazda RX4

```
myquery   = {"name":"Mazda RX4"}
newvalues = {"$set":{"mpg":100}}

collection.update_many(myquery, newvalues)
```

MongoDB

- One can also perform update and delete operations with MongoDB
  - Examples:
    - Deleting all 6-cylinder cars

```
collection.delete_many({"cyl":6})
```

    - Deleting all documents

```
collection.delete_many({})
```

MongoDB

- We have barely scratched the surface of what can be done with MongoDB and document-oriented databases
- When should one use MongoDB?
  - Textual data (documents)
  - No predefined database model
  - When the database is used primarily to support APIs
- MongoDB vs Relational Databases
  - Detailed technical analysis https://www.mongodb.com/compare/mongodb-mysql

"*Using MongoDB removes the complex object-relational mapping (ORM) layer that translates objects in code to relational tables. MongoDB's flexible data model also means that your database schema can evolve with business requirements.*"

Homework #6

- Suppose you work for FAA - Federal Aviation Administration
  - Job: database administrator
  - Current tasks:
    1. Migrate FAA's table-based database to a document-oriented database
    2. Provide data to FAA decision makers

Homework #6

- Preliminaries
  - Task 1: load the original data
    - *flights.csv*: available on Canvas (Assignment -> Homework 6)
  - Task 2: create a new collection called *flights* inside the *ISA414* database
  - Task 3: populate the flights collection

Homework #6

- Queries: submit the solutions on Canvas by the end of tomorrow

1. Add the new key-value pair "*delayed":"true"* to the documents containing delayed flights (dep_delay > 0)
   - Hint #1: this is an update statement
   - Hint #2: the “query” part concerns finding the documents where dep_delay >0
   - Hint #3: the “update” part is where you set "*delayed":"true"*
2. Return all the data about delayed flights (*i.e., delayed:true*)
3. Return only the carrier names for the delayed flights (i.e*., delayed:true*)
   - Hint: look at the section “*Specify the Fields to Return*” in the document at https://docs.mongodb.com/manual/reference/method/db.collection.find/

Summary

- We learned about document-oriented databases (MongoDB)
- References
  - List of query commands
    - https://docs.mongodb.com/manual/reference/operator/query/
  - List of aggregate commands
    - https://docs.mongodb.com/manual/reference/operator/aggregation/
  - List of update commands
    - https://docs.mongodb.com/manual/reference/operator/update
- Next lecture: supervised learning