Horse Riding Center

Final Report

Denis Belocur - Sizhe Wu - Yashwant Bist

[**Introduction 3**](#_poqiwju0txga)

[**Database Design 4**](#_k7sat6e2xd4u)

[**Data Insertion 5**](#_ijt69jqzhqi5)

[**CRUD Tasks 6**](#_dofzetgmz8r8)

[Data Viewing Example 6](#_ry0xjgdvrrnx)

[Data Viewing (Date) Example 6](#_7rc8qyji436q)

[Data Aggregation Example 7](#_pt3bv46ggsa0)

[Data Update Example 9](#_amaoj8pcoac)

[**Conclusion 10**](#_2g41v6yeos8j)

## 

## 

## Introduction

The crux of this project was a demonstration of our ability to design and work with a non relational database. Specifically, MongoDB. The particular theme we chose was a horse riding center, aiming to represent information about specific horses the center keeps, specific customers that have used the center, and records of all the transactions between the center and it’s customers.

We created sample data on which we test the database, including a random data generator which is included as well.

We created several examples of how one would go about viewing and aggregating the data, as well as further examples of data insertion, data modification, and data deletion. All of these are bundled in a series of scripts which will be detailed in later sections.

## 

## 

## Database Design

Above you can see an approximation of the database schema we used. It’s an ‘approximation’ because of the modular nature of a non relational database like MongoDB, seeing as new rows or tables can be added almost arbitrarily.

Some interesting interactions can be noticed here as well. For one, Customers and Rentals can be related to each other through either the contactEmail or the contactPhonenum, or not at all! There really isn’t much of a problem with the contact information attached to the rental being a dangling reference here. Due to the non-relational nature of this database, none of that is an issue.

## Data Insertion

We do our sample data insertion using scripts. Below is an example of one of our scripts for this purpose.

let db = connect('mongodb://localhost/dbs311\_a2');

db.horses.insertMany(

[

{

name: "Shadow",

colour: "Black",

height: 60,

speed: 7,

dateborn: new Date("2007-03-27"),

standard\_priceperhour: 29.99

},

{

name: "Apples",

colour: "Orange",

height: 55,

speed: 6,

dateborn: new Date("2012-06-12"),

standard\_priceperhour: 27.99

}

]

);

We load these javascript scripts into MongoSH, from where we connect to the DB (here hosted locally), and insert data into the tables that are defined in the previous section. Similar sample data exists for each of the three tables, and then our sample data is also complimented by a random sample data generation module. The module randomly combies various pieces of data that are predefined for that particular data set. For example, we would have an array of sample first names and sample last names to populate the customers table with.

## 

## CRUD Tasks

We have a series of javascript scripts that perform various CRUD tasks on the sample data.

### Data Viewing Example

let db = connect('mongodb://localhost/dbs311\_a2');

try {

console.log(db.horses.find({"colour": "Black"}).pretty());

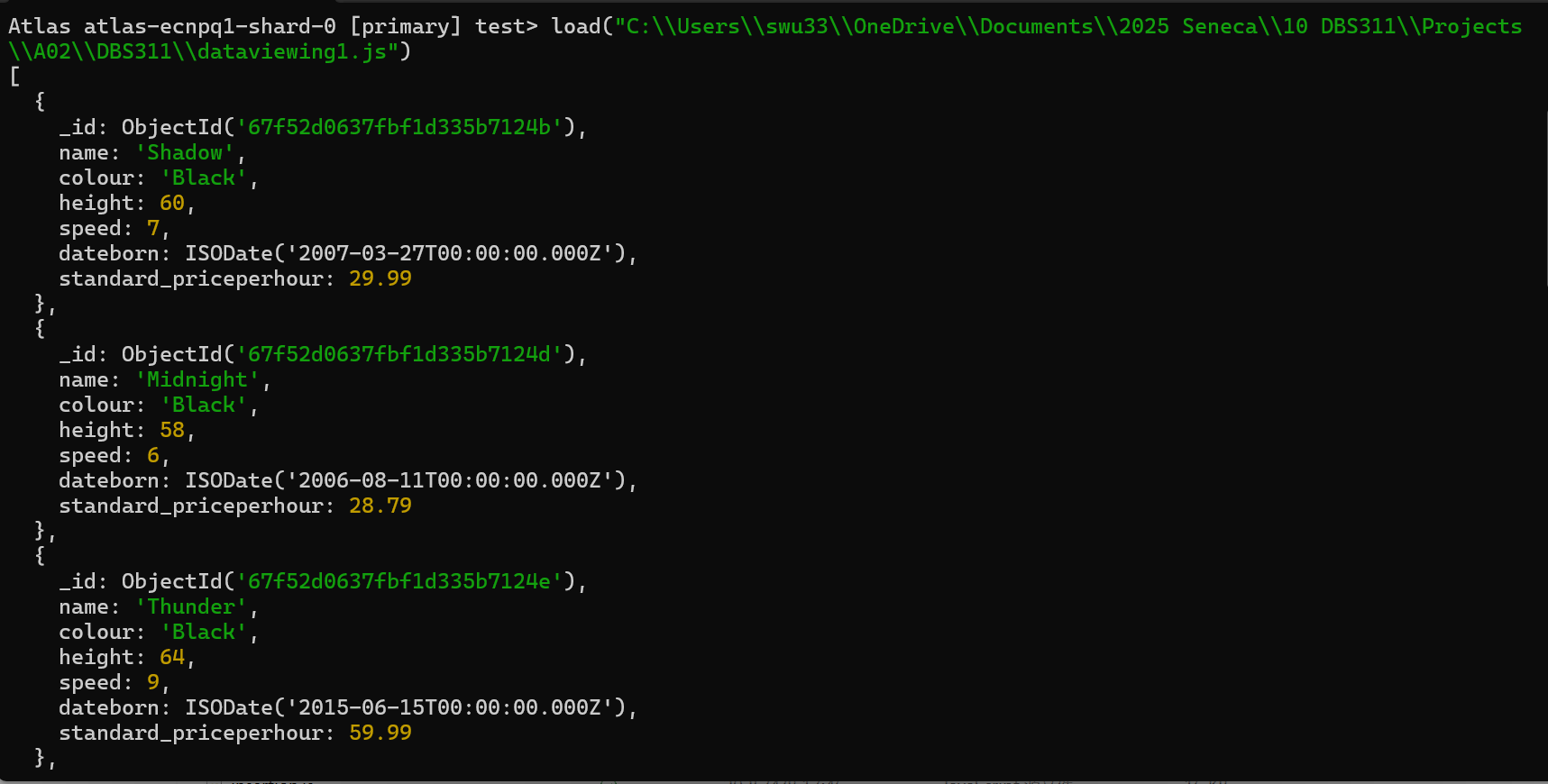
} catch (error) {

console.log(error);

}

This is a fairly simple example of finding data according to a set condition. Here we find all the horses whose colour is black. This is equivalent to the OracleSQL statement SELECT \* FROM horses WHERE colour IS ‘Black’;.

Sample Output:



### Data Viewing (Date) Example

console.log(db.rentals.find({

"date\_rented" : { // summer of '24

$gte : new Date("2024-06-01T00:00:00Z"),

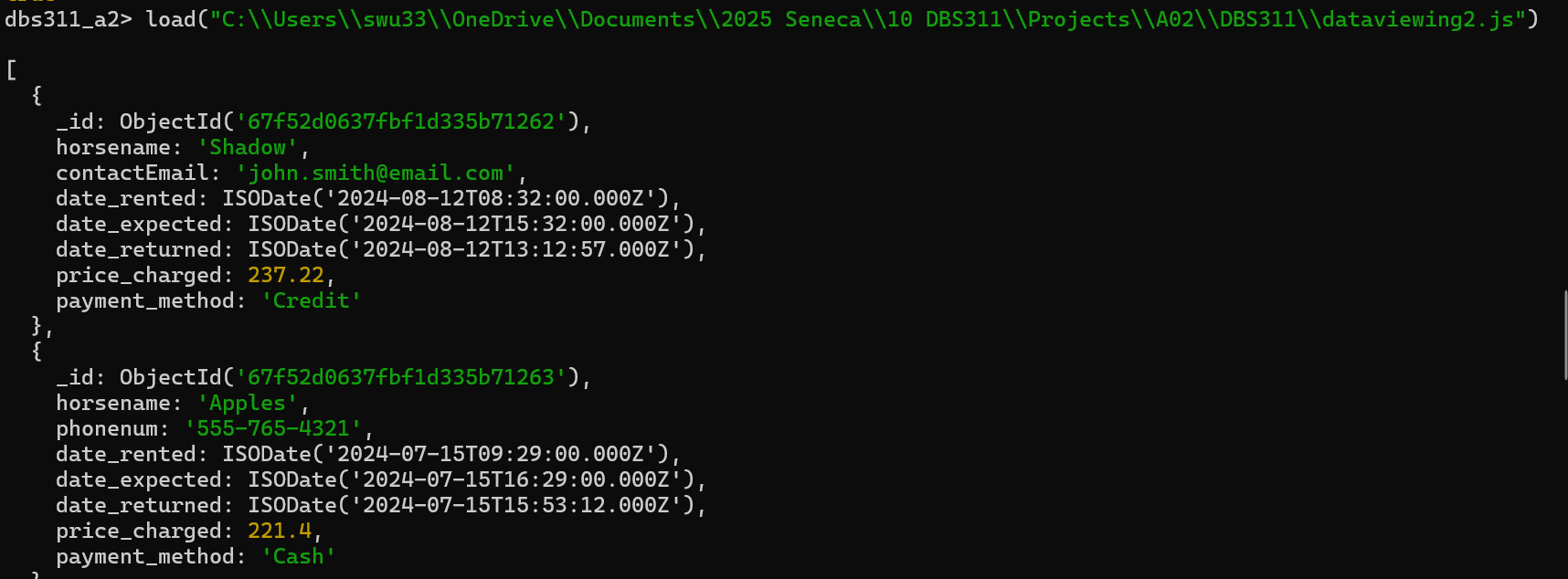
$lte : new Date("2024-08-31T23:59:59Z")

}

}).pretty());

Here is included an example of sorting by date, and an example of some of the operators included in MongoDB (Without the connection code; I think we’ve gotten the idea by now). We print to the screen all the rentals that took place in the summer of 2024. Rather than assigning the “date\_rented” filter a single value, we assign it an object with the $gte and $lte filters set to the target values. There are many more filters available, for a range of purposes. This is equivalent to the OracleSQL statement SELECT \* FROM rentals WHERE date\_rented BETWEEN TO\_DATE( '2024-06-01 00:00:00', 'yyyy-mm-dd hh24:mi:ss' ) AND TO\_DATE('2024-08-31 23:59:59’, 'yyyy-mm-dd hh24:mi:ss');

Sample Output:



### Data Aggregation Example

console.log(db.horses.aggregate(

[

{

$group: {

\_id: "$speed",

average\_price\_per\_hour: { $avg:

"$standard\_priceperhour" },

count: { $sum: 1 }

}

},

{

$sort: { \_id: 1 }

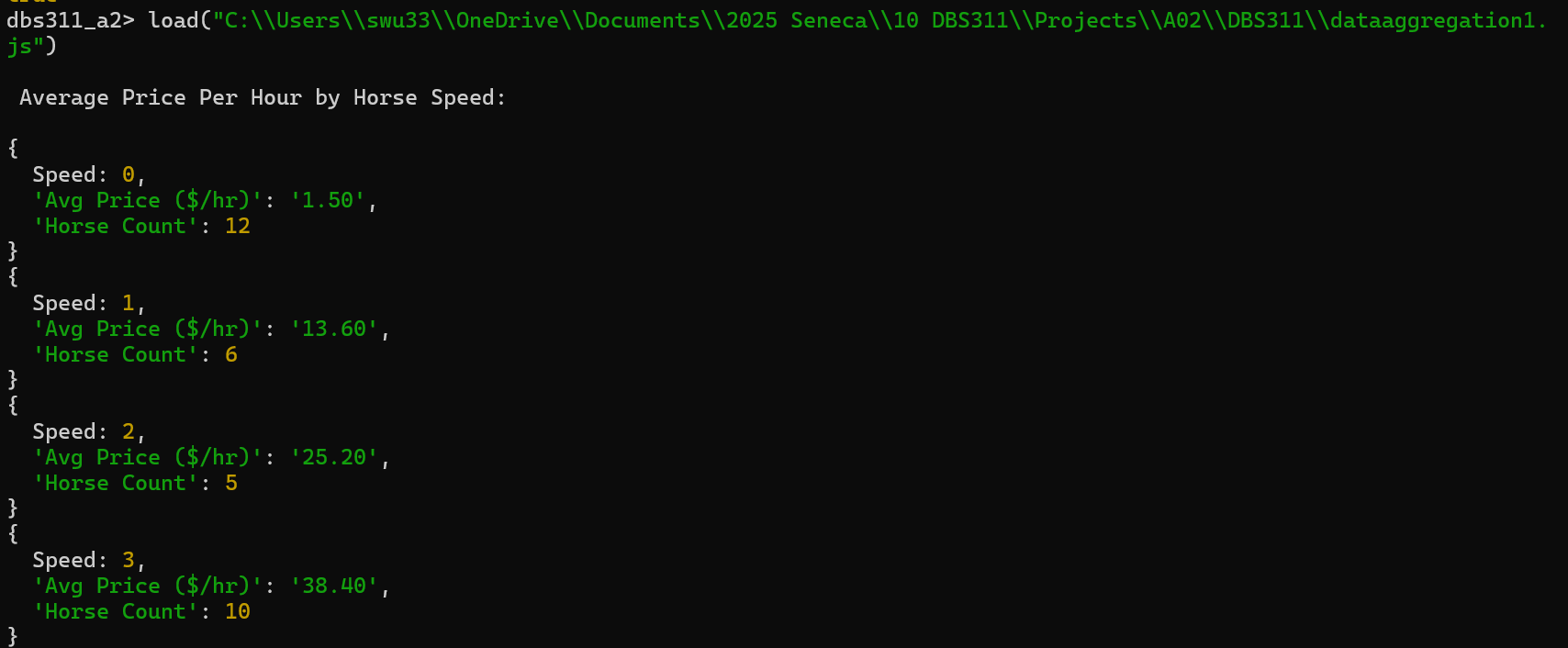
}

]

).pretty());

Here we see an example of the aggregate() function, a more complicated version of the find() function. It has three components: filtering, grouping, and sorting. Filtering can be seen in the examples of the find() function, so we’ll talk about the other two here. The $group section creates a temporary table, here indexed by the speed value of the horse collection, and then also including the average of all the hourly prices of all the speeds, and another field including the number of horses with that speed. From there, the $sort section simply sorts the output by the id (which is set to the various speeds that the horses can have). This is equivalent to the OracleSQL statement SELECT speed, AVG(standard\_priceperhour), COUNT(\*) FROM horses GROUP BY speed ORDER BY speed ASC;

Sample Output:



### Data Update Example

db.horses.updateMany(

{

"speed" : {$gte : 7}

},

{

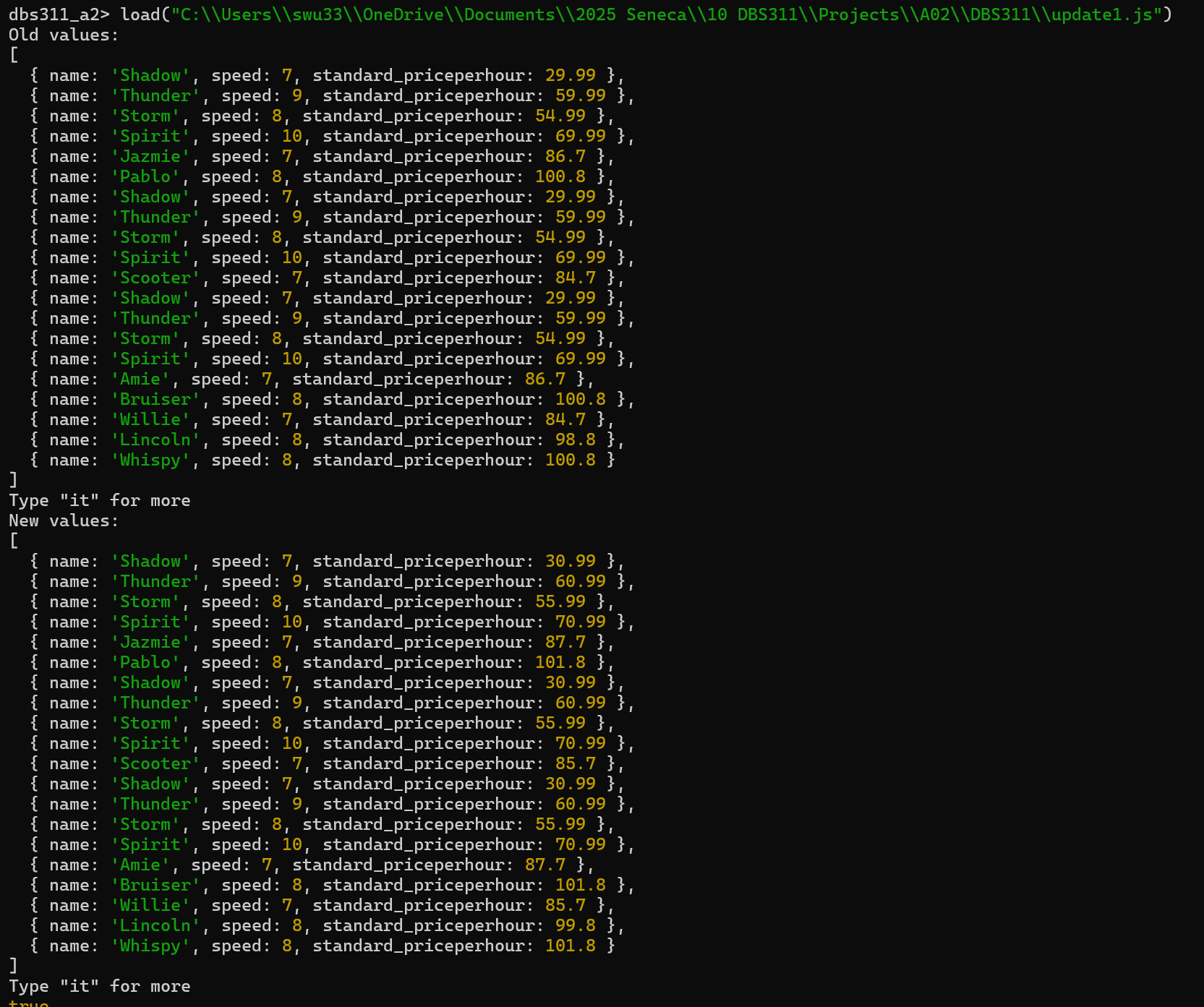
$inc : {"standard\_priceperhour" : 1}

}

);

This is a simple example of data updating. The updateOne() and updateMany() functions consist mainly of two parts: a filter and a modification rule. Here, the filter selects all the horses whose speed is higher than 7, and the modification rule increments all of those horses hourly prices by one. The equivalent OracleSQL command is UPDATE horses SET standard\_priceperhour = standard\_priceperhour + 1 WHERE speed > 7;

Sample Output:



## Conclusion

To conclude, we created a fully functional example non-relational database in MongoDB, complete with a sample dataset and CRUD tasks. We successfully represented a hypothetical horse riding center’s database, and all the functionality associated with it. Working on this assignment gave us a much deeper understanding of how MongoDB handles data, especially compared to SQL databases. One thing that stood out was how MongoDB doesn’t enforce strict schemas or foreign key constraints. While that offers flexibility, it also means you need to manage consistency and data relationships more carefully on your own.

We successfully implemented a working data model for a horse riding center, complete with random data generation and fully functional CRUD operations. Seeing how data could be queried, aggregated, and updated in a non-relational environment was eye-opening, and it gave us a much clearer picture of when and why NoSQL databases like MongoDB are used in the real world.

Overall, this project was a great introduction to MongoDB, and a valuable step toward becoming more versatile with different types of database systems.