# Syllabus

- Mongo DB Data types, CRUD operations: Mongo DB create operations, Mongo DB read operation, Mongo DB update operations, Mongo DB delete operations, mongo DB bulk write operations

# Mongo DB CRUD Operations

- CRUD operations describe the conventions of a user-interface that let users view, search, and modify parts of the database.
- MongoDB documents are modified by connecting to a server, querying the proper documents, and then changing the setting properties before sending the data back to the database to be updated. CRUD is data-oriented, and it's standardized according to HTTP action verbs.

# Mongo DB CRUD Operations

- When it comes to the individual CRUD operations:
- The Create operation is used to insert new documents in the MongoDB database.
- The Read operation is used to query a document in the database.
- The Update operation is used to modify existing documents in the database.
- The Delete operation is used to remove documents in the database

# Create Operations

- For MongoDB CRUD, if the specified collection doesn't exist, the create operation will create the collection when it's executed. Create operations in MongoDB target a single collection, not multiple collections. Insert operations in MongoDB are atomic on a single document level.
- MongoDB provides two different create operations that you can use to insert documents into a collection:
  - db.collection.insertOne()
  - db.collection.insertMany()

# Insert one()

- As the namesake, insertOne() allows you to insert one document into the collection.
- For this example, we're going to work with a collection called RecordsDB. We can insert a single entry into our collection by calling the insertOne() method on RecordsDB.
- We then provide the information we want to insert in the form of key-value pairs, establishing the schema.

# Insert one()

- db.RecordsDB.insertOne({ name: "Marsh", age: "6 years", species: "Dog", ownerAddress: "380 W. FirAve", chipped: true })

# Insert one()

- If the create operation is successful, a new document is created. The function will return an object where “acknowledged” is “true” and “insertID” is the newly created “ObjectId.”
- db.RecordsDB.insertOne({ ... name: "Marsh", ... age: "6 years", ... species: "Dog", ... ownerAddress: "380 W. Fir Ave", ... chipped: true ... }) { "acknowledged" : true, "insertedId“ ObjectId("5fd989674e6b9ceb8665c57d") }

# insertMany()

- It’s possible to insert multiple items at one time by calling the *insertMany()* method on the desired collection. In this case, we pass multiple items into our chosen collection (*RecordsDB*) and separate them by commas. Within the parentheses, we use brackets to indicate that we are passing in a list of multiple entries. This is commonly referred to as a nested method.
- db.RecordsDB.insertMany([{ name: "Marsh", age: "6 years", species: "Dog", ownerAddress: "380 W. Fir Ave", chipped: true}, {name: "Kitana", age: "4 years", species: "Cat", ownerAddress: "521 E. Cortland", chipped: true}])

# insertMany()

- db.RecordsDB.insertMany([{ name: "Marsh", age: "6 years", species: "Dog", ownerAddress: "380 W. Fir Ave", chipped: true}, {name: "Kitana", age: "4 years", species: "Cat", ownerAddress: "521 E. Cortland", chipped: true}]) { "acknowledged" : true, "insertedIds" : [ ObjectId("5fd98ea9ce6e8850d88270b4"), ObjectId("5fd98ea9ce6e8850d88270b5") ] }

# Read Operations

- The read operations allow you to supply special query filters and criteria that let you specify which documents you want. The MongoDB documentation contains more information on the available query filters. Query modifiers may also be used to change how many results are returned.
- MongoDB has two methods of reading documents from a collection:
  - db.collection.find()
  - db.collection.findOne()

```
db.users.find(
    { age: { $gt: 18 } },
    { name: 1, address: 1 }
).limit(5)
```

← collection
← query criteria
← projection
← cursor modifier

# Read Operations

- find()
- In order to get all the documents from a collection, we can simply use the *find()* method on our chosen collection. Executing just the *find()* method with no arguments will return all records currently in the collection.
- db.RecordsDB.find()
- { "_id" : ObjectId("5fd98ea9ce6e8850d88270b5"), "name" : "Kitana", "age" : "4 years", "species" : "Cat", "ownerAddress" : "521 E. Cortland", "chipped" : true } { "_id" : ObjectId("5fd993a2ce6e8850d88270b7"), "name" : "Marsh", "age" : "6 years", "species" : "Dog", "ownerAddress" : "380 W. Fir Ave", "chipped" : true } { "_id" : ObjectId("5fd993f3ce6e8850d88270b8"), "name" : "Loo", "age" : "3 years", "species" : "Dog", "ownerAddress" : "380 W. Fir Ave", "chipped" : true }

# Find one()

- In order to get one document that satisfies the search criteria, we can simply use the *findOne()* method on our chosen collection. If multiple documents satisfy the query, this method returns the first document according to the natural order which reflects the order of documents on the disk. If no documents satisfy the search criteria, the function returns null. The function takes the following form of syntax.
- db.{collection}.findOne({query}, {projection})

# Find one()

- { "_id" : ObjectId("5fd98ea9ce6e8850d88270b5"), "name" : "Kitana", "age" : "8 years", "species" : "Cat", "ownerAddress" : "521 E. Cortland", "chipped" : true } { "_id" : ObjectId("5fd993a2ce6e8850d88270b7"), "name" : "Marsh", "age" : "6 years", "species" : "Dog", "ownerAddress" : "380 W. Fir Ave", "chipped" : true } { "_id" : ObjectId("5fd993f3ce6e8850d88270b8"), "name" : "Loo", "age" : "3 years", "species" : "Dog", "ownerAddress" : "380 W. Fir Ave", "chipped" : true }

# Find one()

- db.RecordsDB.find({"age":"8 years"})
- { "_id" : ObjectId("5fd98ea9ce6e8850d88270b5"), "name" : "Kitana", "age" : "8 years", "species" : "Cat", "ownerAddress" : "521 E. Cortland", "chipped" : true }

# Update Operations

- Like create operations, update operations operate on a single collection, and they are atomic at a single document level. An update operation takes filters and criteria to select the documents you want to update.
- You should be careful when updating documents, as updates are permanent and can't be rolled back. This applies to delete operations as well.
- For MongoDB CRUD, there are three different methods of updating documents:
  - db.collection.updateOne()
  - db.collection.updateMany()
  - db.collection.replaceOne()

# Update one()

- We can update a currently existing record and change a single document with an update operation. To do this, we use the *updateOne()* method on a chosen collection, which here is “RecordsDB.” To update a document, we provide the method with two arguments: an update filter and an update action.
- The update filter defines which items we want to update, and the update action defines how to update those items. We first pass in the update filter. Then, we use the “$set” key and provide the fields we want to update as a value. This method will update the first record that matches the provided filter.

# Update one()

- db.RecordsDB.updateOne({name: "Marsh"}, {$set:{ownerAddress: "451 W. Coffee St. A204"}})
- { "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
- { "_id" : ObjectId("5fd993a2ce6e8850d88270b7"), "name" : "Marsh", "age" : "6 years", "species" : "Dog", "ownerAddress" : "451 W. Coffee St. A204", "chipped" : true }

```
db.users.updateMany(                          <-- collection
   { age: { $lt: 18 } },                      <-- update filter
   { $set: { status: "reject" } }             <-- update action
)
```

# Update Many()

- updateMany() allows us to update multiple items by passing in a list of items, just as we did when inserting multiple items. This update operation uses the same syntax for updating a single document.
- db.RecordsDB.updateMany({species:"Dog"}, {$set: {age: "5"}})
- { "acknowledged" : true, "matchedCount" : 3, "modifiedCount" : 3 }

> db.RecordsDB.find()

- { "_id" : ObjectId("5fd98ea9ce6e8850d88270b5"), "name" : "Kitana", "age" : "4 years", "species" : "Cat", "ownerAddress" : "521 E. Cortland", "chipped" : true }
- { "_id" : ObjectId("5fd993a2ce6e8850d88270b7"), "name" : "Marsh", "age" : "5", "species" : "Dog", "ownerAddress" : "451 W. Coffee St. A204", "chipped" : true }
- { "_id" : ObjectId("5fd993f3ce6e8850d88270b8"), "name" : "Loo", "age" : "5", "species" : "Dog", "ownerAddress" : "380 W. Fir Ave", "chipped" : true }
- { "_id" : ObjectId("5fd994efce6e8850d88270ba"), "name" : "Kevin", "age" : "5", "species" : "Dog", "ownerAddress" : "900 W. Wood Way", "chipped" : true }

# Replace one()

- The replaceOne() method is used to replace a single document in the specified collection. replaceOne() replaces the entire document, meaning fields in the old document not contained in the new will be lost.
- db.RecordsDB.replaceOne({name: "Kevin"}, {name: "Maki"})
- { "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }

> db.RecordsDB.find()

- { "_id" : ObjectId("5fd98ea9ce6e8850d88270b5"), "name" : "Kitana", "age" : "4 years", "species" : "Cat", "ownerAddress" : "521 E. Cortland", "chipped" : true }
- { "_id" : ObjectId("5fd993a2ce6e8850d88270b7"), "name" : "Marsh", "age" : "5", "species" : "Dog", "ownerAddress" : "451 W. Coffee St. A204", "chipped" : true }
- { "_id" : ObjectId("5fd993f3ce6e8850d88270b8"), "name" : "Loo", "age" : "5", "species" : "Dog", "ownerAddress" : "380 W. Fir Ave", "chipped" : true }
- { "_id" : ObjectId("5fd994efce6e8850d88270ba"), "name" : "Maki" }

# Delete Operations

- Delete operations operate on a single collection, like update and create operations. Delete operations are also atomic for a single document. You can provide delete operations with filters and criteria in order to specify which documents you would like to delete from a collection. The filter options rely on the same syntax that read operations utilize.
- MongoDB has two different methods of deleting records from a collection:

- db.collection.deleteOne()
- db.collection.deleteMany()

# Delete Operations

```
db.users.deleteMany(
   { status: "reject" }
)
```


# Delete one()

- deleteOne() is used to remove a document from a specified collection on the MongoDB server. A filter criteria is used to specify the item to delete. It deletes the first record that matches the provided filter.
- db.RecordsDB.deleteOne({name:"Maki"})
- { "acknowledged" : true, "deletedCount" : 1 }

> db.RecordsDB.find()

{ "_id" : ObjectId("5fd98ea9ce6e8850d88270b5"), "name" : "Kitana", "age" : "4 years", "species" : "Cat", "ownerAddress" : "521 E. Cortland", "chipped" : true }

{ "_id" : ObjectId("5fd993a2ce6e8850d88270b7"), "name" : "Marsh", "age" : "5", "species" : "Dog", "ownerAddress" : "451 W. Coffee St. A204", "chipped" : true }

{ "_id" : ObjectId("5fd993f3ce6e8850d88270b8"), "name" : "Loo", "age" : "5", "species" : "Dog", "ownerAddress" : "380 W. Fir Ave", "chipped" : true }

# Delete Many()

- deleteMany() is a method used to delete multiple documents from a desired collection with a single delete operation. A list is passed into the method and the individual items are defined with filter criteria as in deleteOne().
- db.RecordsDB.deleteMany({species:"Dog"})
- db.RecordsDB.deleteMany({species:"Dog"})
- > db.RecordsDB.find()
- { "_id" : ObjectId("5fd98ea9ce6e8850d88270b5"), "name" : "Kitana", "age" : "4 years", "species" : "Cat", "ownerAddress" : "521 E. Cortland", "chipped" : true }

# Mongo DB Bulk Write

- MongoDB provides a way to perform multiple writing operations. By using the MongoDB bulk write, we can perform bulk insert, bulk update and bulk remove operations in one go.
- These operations can be ordered or unordered.
- If the `ordered` option is set to true and if any error occurs while performing bulk write, in that case, MongoDB will not process the remaining write operations and will return an error.

# Mongo DB Bulk Write

```
try {
   db.pizzas.bulkWrite( [
      { insertOne: { document: { _id: 3, type: "beef", size:
"medium", price: 6 } } },
      { insertOne: { document: { _id: 4, type: "sausage", size:
"large", price: 10 } } },
      { updateOne: {
         filter: { type: "cheese" },
         update: { $set: { price: 8 } }
      } },
      { deleteOne: { filter: { type: "pepperoni"} } },
      { replaceOne: {
         filter: { type: "vegan" },
         replacement: { type: "tofu", size: "small", price: 4 }
      } }
   ] )
} catch( error ) {
   print( error )
}
```