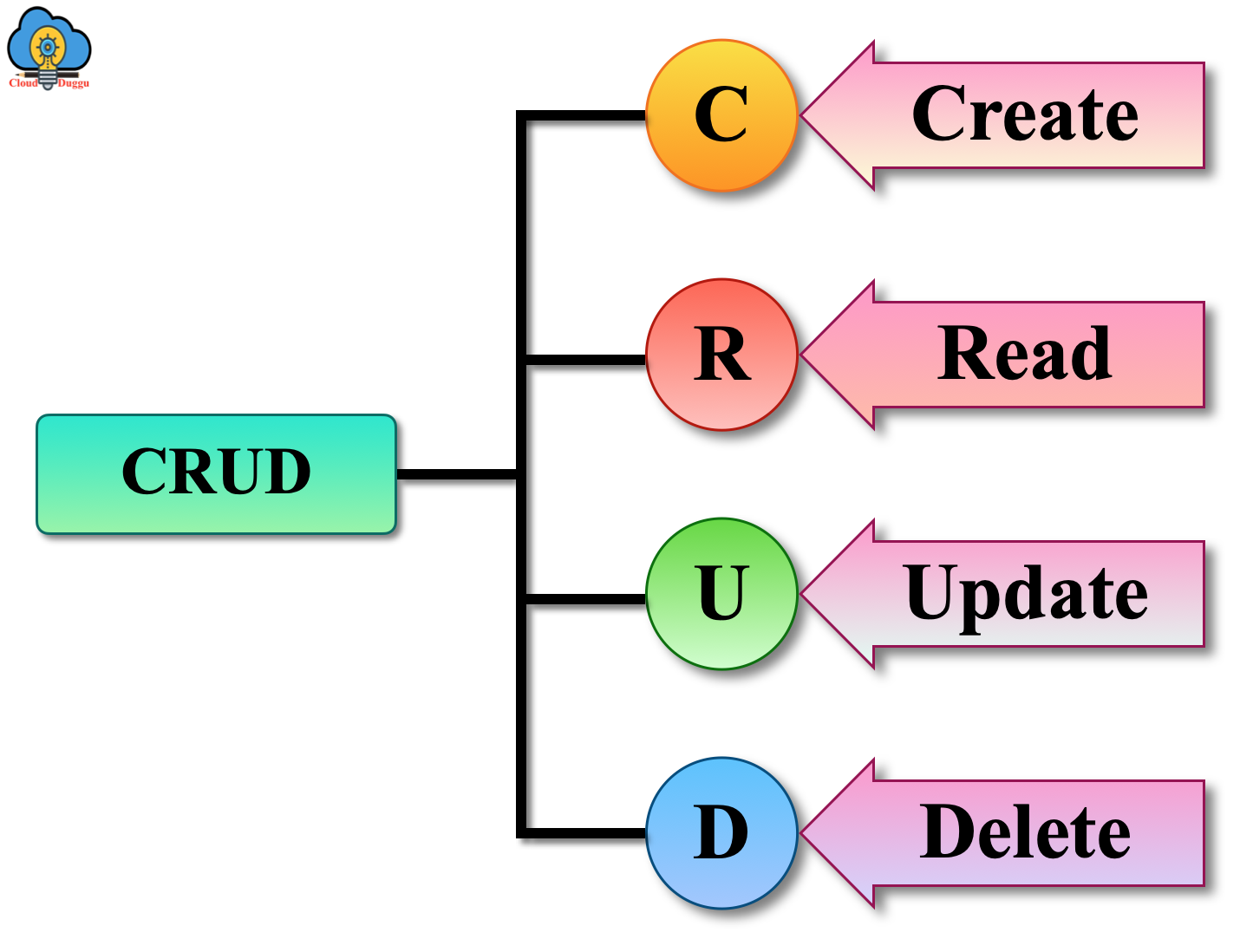
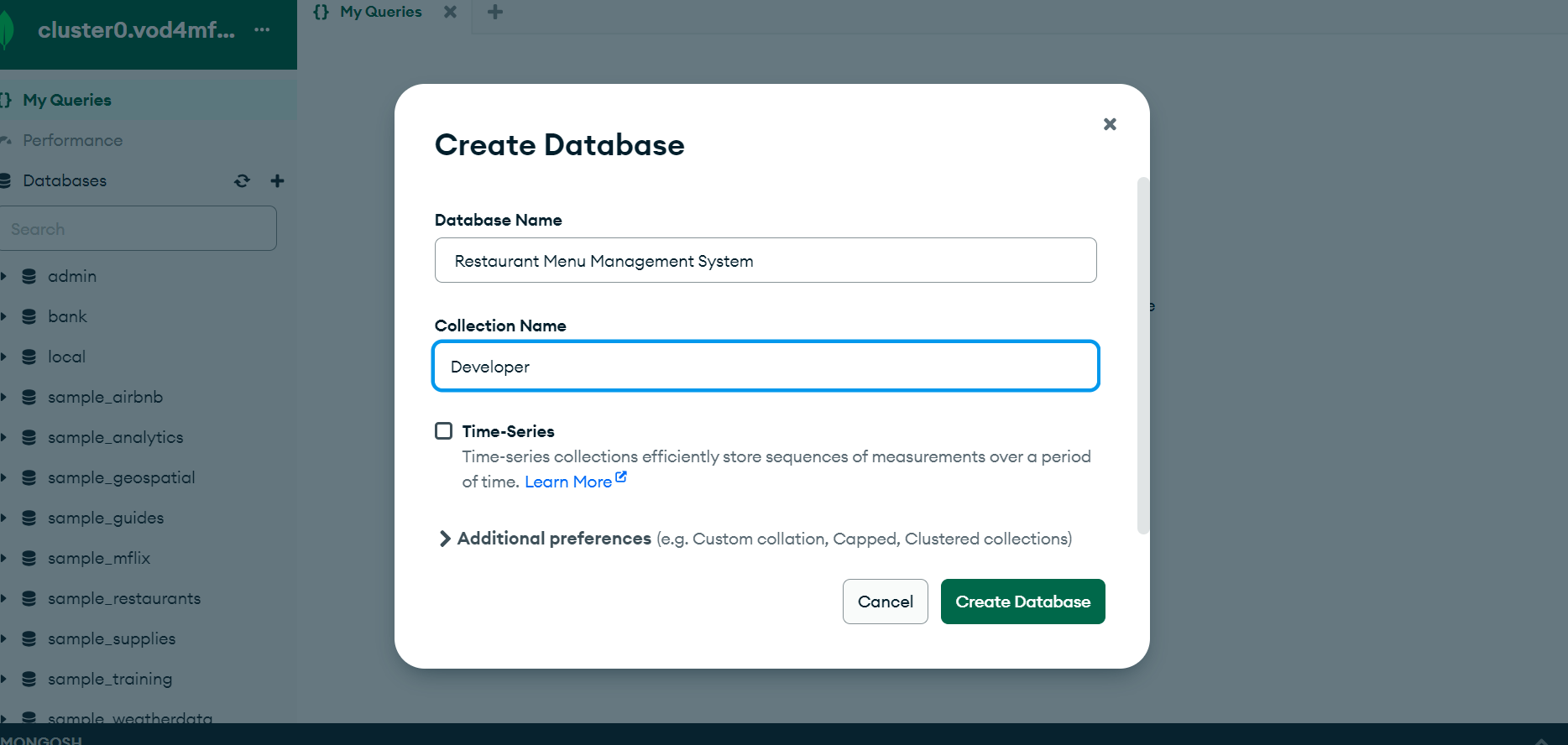
**CRUD OPERATIONS IN MONGO DB:**

With MongoDB CRUD, users may read and write data with extensive capabilities. MongoDB is a distributed database system that is based on documents. The operations CREATE, READ, UPDATE, and DELETE are used to execute on MongoDB documents, including create, update, read, and delete. These operations let us readily interface with the database and carry out a number of different tasks.



**REAL TIME SCENARIO (RESTARUANT MENUE MANAGEMENT):**

**Creating data base:**



**1)Insert** **One():**

**Code:**

db.dishes.insertOne({  
 title: "Indian resturant",  
 dishId: "D001",  
 name: "chicken Pizza",  
 ingredients: ["Flour", "Tomatoes", "Mozzarella Cheese", "Basil"],  
 price: 12.99,  
 category: "Pizza",  
 available: true  
});

A)One document is inserted into a collection using the insertOne function, a MongoDB operation. Here, the group is referred to as dishes.

B)Enclosed in curly brackets {}, the document being added is an entry for a single dish record in a restaurant database.

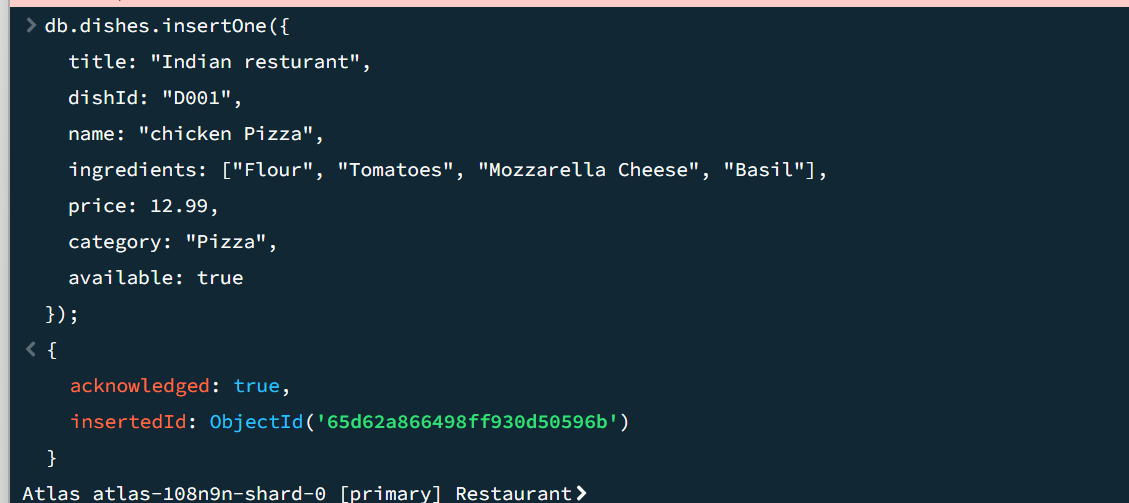
C)Several fields with their corresponding values can be found in the document:

title: A string that contains the dish's title or topic (in this example, "Indian restaurant").

dishId: The dish's unique identification number; in this case, it's "D001".

"Chicken Pizza" is the dish's name in this instance.

ingredients: A list of the ingredients used in the recipe, such as "Margarine," "Flour," "Tomatoes,"



**Insert Many():**

**Method Used:** insertMany - This method is made especially to insert several documents at once into a MongoDB collection.

The papers are going to be added to a collection called dishes, which is the target collection.

**Document Structure:** Two JSON objects are provided by the code, each of which represents a dish and has a number of fields:

**dishId:** A special number assigned to the dish.

**name:** The dish's official name.

**components:** A list of the components that go into making the dish.

**cost:** How much the dish costs.

**category:** The food's appropriate category (e.g., Salad, Pasta).

available A Boolean value that indicates the availability of the dish at the moment.

Inserting a batch: In a single database activity, the code effectively puts both documents into the dishes collection by utilizing insertMany, which can be

**Code** :

db.dishes.insertMany([  
 {  
 dishId: "D002",  
 name: "Caesar Salad",  
 ingredients: ["Romaine Lettuce", "Croutons", "Parmesan Cheese", "Caesar Dressing"],  
 price: 8.99,  
 category: "Salad",  
 available: true  
 },  
 {  
 dishId: "D003",  
 name: "Spaghetti Carbonara",  
 ingredients: ["Spaghetti", "Eggs", "Pancetta", "Parmesan Cheese"],  
 price: 14.99,  
 category: "Pasta",  
 available: true  
 }  
]);



Find():

Code :

db.dishes.find({  
 dishId: "D003",  
});

**Target Collection:** The dishes collection, which most likely has documents representing distinct dish items, is the collection against which the query is run.

**Query Method:** To get all documents that meet the query requirements, the find method is employed. Specifically, it looks for papers with the dishId equal to "D003".

**Criteria for Queries:** This criteria, indicated by { dishId: "D003" }, instructs the query to search for documents with the dishId field exactly equal to "D003".

As a result, the query provides a cursor to the documents that satisfy the requirements. Should there is more than one match, it will return all of them.

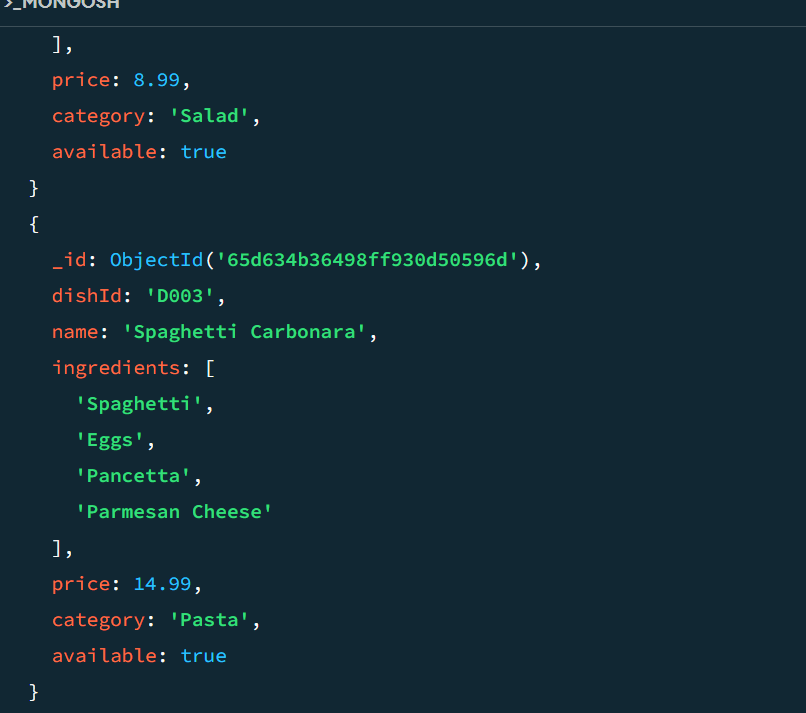


**FindMany():**

Code : db.dishes.find({

dishId: { $in: ["D003", "D002"] }

});



Using either "D003" or "D002" in the dishId column, this MongoDB query returns documents from the dishes collection. For a brief summary, see this:

Assumed to include documents that represent individual dishes, the operation's target collection is the dishes collection.

Find documents that match the given criteria by using the query method.

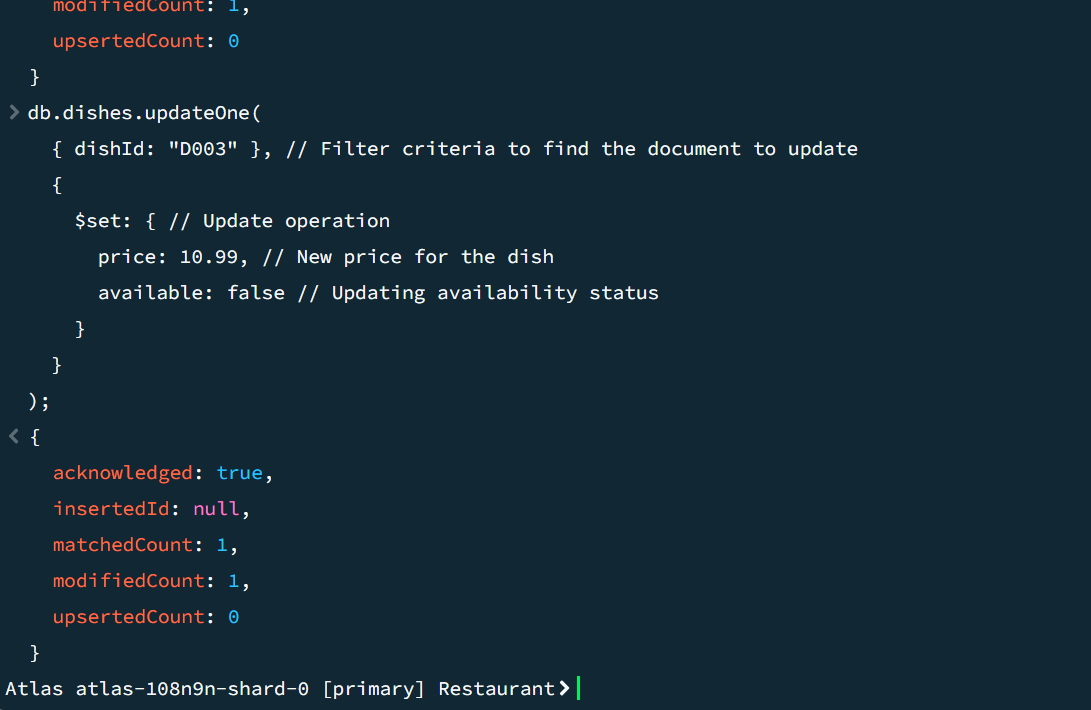
**Query Criteria:** use the $in operator in the query object { dishId: { $in: ["D003", "D002"] } } to tell MongoDB to choose documents whose value for dishId is present in the supplied array.

**Outcome:** If both identifiers are present in the collection, the command may retrieve several documents by returning a cursor to the documents with dishId values of "D003" or "D002".

**Update one ():**

**Code :**

db.dishes.updateOne(  
 { dishId: "D003" }, // Filter criteria to find the document to update  
 {  
 $set: { // Update operation  
 price: 10.99, // New price for the dish  
 available: false // Updating availability status  
 }  
 }  
);



**Collection:** The MongoDB database's dish collection is the focus of the procedure.

**Filter Criteria:** The document that requires updating can be found using the filter { dishId: "D003"}. It indicates which document should be chosen for the update process and has a dishId value of "D003".

**Update Operation:** To make changes to particular fields in the chosen document, use the $set operator. This action merely modifies the designated fields; the document as a whole is not replaced.

**Updateable fields:**

**price:** 10.99: This modifies the corresponding document's price field to 10.99.

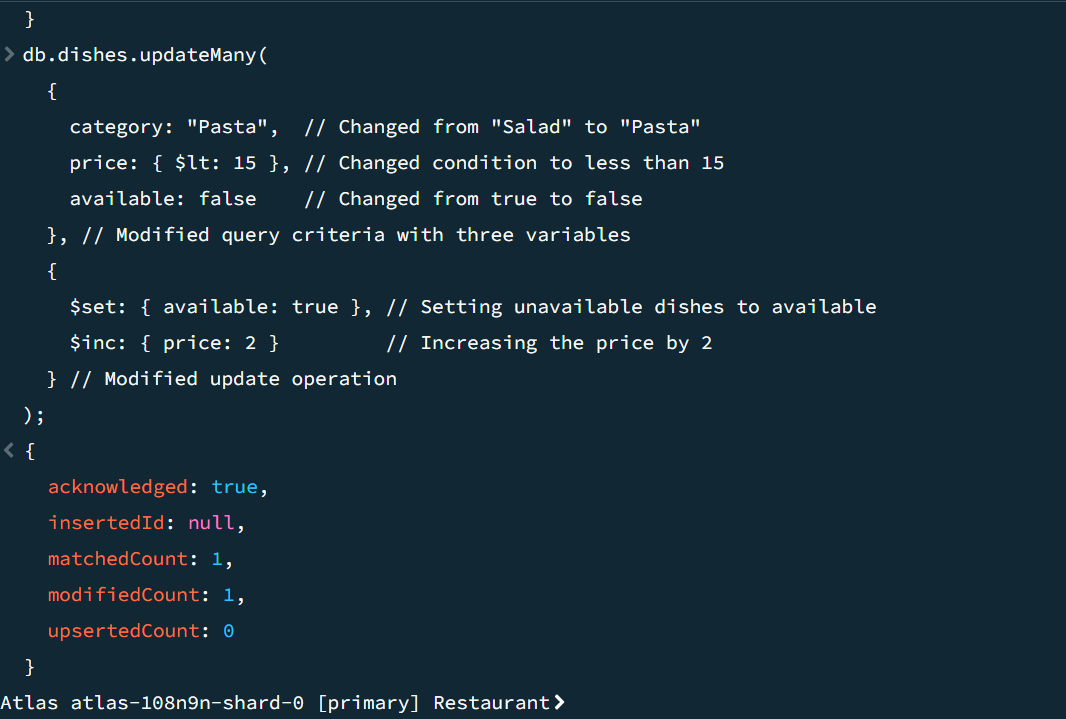
**accessible:** false: This indicates that the dish is no longer available by changing the availability field to false.

To sum up, this command looks for the document with dishId "D003" in the dishes collection and modifies its

**Update Many():**

**Code :**

db.dishes.updateMany(  
 {   
 category: "Pasta", // Changed from "Salad" to "Pasta"  
 price: { $lt: 15 }, // Changed condition to less than 15  
 available: false // Changed from true to false  
 }, // Modified query criteria with three variables  
 {   
 $set: { available: true }, // Setting unavailable dishes to available  
 $inc: { price: 2 } // Increasing the price by 2  
 } // Modified update operation  
);



Several documents in the dishes collection are updated by this MongoDB command. This is a quick rundown of what it can do:

The command is intended to work on the dishes collection, which is presumed to contain data regarding different dishes.

**Criteria for Queries:** It lays out three requirements for choosing documents:

Only documents with the category field set to "Pasta" are included in this category.

**price:** { $lt: 15}: Only records with a price field value < 15.

available documents only whose availability field is set to false (false).

Two update actions are carried out by the command on all papers that meet the aforementioned requirements.

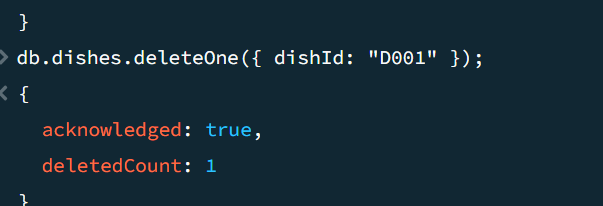
$set: { available: true }: Makes pasta meals that were previously unavailable available by setting the available field to true.

$inc: { price: 2 }: Raises the cost of every corresponding dish by two.

**Delete one ():**

Code:

db.dishes.deleteOne({ dishId: "D001" });



**Collection Target:** The command is directed at the dishes collection, which is likely to contain documents that correspond to specific dish items.

The first document that satisfies the given query criteria is deleted using the deleteOne function.

**{ dishId: "D001" } Query Criteria: -** This indicates the deletion condition. The deletion of a document will only be considered for those whose dishId equals "D001".

**Operation Impact:** If the specified condition is met, the command will remove a maximum of one document. Only the first document that MongoDB comes across that meets the requirements will be removed if several do.

**Use Case:** When a certain dish is no longer available and needs to be taken out of the collection, this function is helpful for eliminating certain entries from a database.

**Delete many()**:

Code :

// Find documents with name 'chicken pizza'

db.dishes.find({ name: 'chicken pizza' })

// Deleting many documents based on a filter

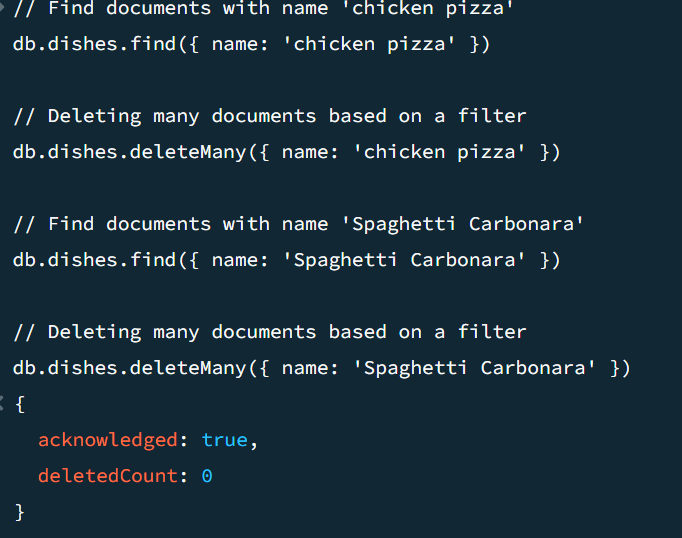
db.dishes.deleteMany({ name: 'chicken pizza' })

// Find documents with name 'Spaghetti Carbonara'

db.dishes.find({ name: 'Spaghetti Carbonara' })

// Deleting many documents based on a filter

db.dishes.deleteMany({ name: 'Spaghetti Carbonara' })



Searching for "chicken pizza": The code uses the find() function to look for documents in the dishes collection with the name "chicken pizza."

**'Chicken pizza' document deletion:** Next, it uses the delete Many() function to remove several documents with the name 'chicken pizza' from the dishes collection.

**'Spaghetti Carbonara' query**: Similarly, the code uses the find() method to look for documents in the dishes collection with the name "Spaghetti Carbonara."

**'Spaghetti Carbonara' document deletion**: The deleteMany() method is used to remove several documents with the name 'Spaghetti Carbonara' from the dishes collection.

**Separate Operations:** To ensure code clarity and modularity, the tasks of searching and deleting documents are carried out independently.

**Sorting according to Name:** A filter object is used by the find() and deleteMany() functions.