CLASS 02:

ADD, UPDATE AND DELETE

Add operation:

In MongoDB, the add operation is not a native operation like insert, update, or delete. However, if you're referring to adding a new document to a collection, you typically use the insertOne() or insertMany() methods.

Syntax:

{
$add:
[
<expression1>,
<expression2>,
...
] }

Example:

let's consider a more comprehensive example involving a collection of books. Each document in the books collection contains information about a book, including its title, author, genre, publication year, and availability status.

```
db.employees.find({ "age": { $gt: 25 }, "department": "IT" })
{
  "_id": ObjectId("60c48e0b2e19b51bec7d10aa"),
  "title": "The Great Gatsby",
  "author": "F. Scott Fitzgerald",
  "genre": "Fiction",
  "publication_year": 1925,
  "available": true
}
```

Update :

In MongoDB, the update operations allow you to modify existing documents in a collection. There are several methods for updating documents:

1. updateOne: Updates a single document that matches a specified filter.
2. updateMany: Updates all documents that match a specified filter.
3. replaceOne: Replaces a single document that matches a specified filter with a new document.

Syntax:

db.collection.updateOne(
<filter>,
<update>,
{
upsert: <boolean>, // Optional: If true, creates a new document if no documents match the filter.
writeConcern: <document>, // Optional: The write concern to use for the operation.
collation: <document> // Optional: The collation to use for the operation.
}
)

Example: Using $or in a Find Operation

Let's assume we have a MongoDB collection called employees with the following documents:

```
test> db.stu.updateOne({name:"Alice Smith"},{$set:{gpa:3.8}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 0,
  upsertedCount: 0
}
test>
```

Update One Document

Suppose we want to update the salary of the employee named "Alice" to 55000. We can use the updateOne method:
When you execute the updateOne operation in MongoDB, the output will be a result object indicating the status of the operation. The result object contains several fields that provide information about the update operation. Here is what you can expect:

Example:

Using updateMany

```
test> db.stu.updateMany({gpa:{$lt:3.0}},{$inc:{gpa:0.5}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 261,
  modifiedCount: 261,
  upsertedCount: 0
}
```

Example:

using replaceOne
Replace One Document

Suppose we want to replace the entire document for the employee named "Charlie" with a new document. We can use the replaceOne method:

```
db.employees.replaceOne(
    { "name": "Charlie" },
    { "_id": 3, "name": "Charlie", "age": 36, "department": "Sales", "salary": 58000 }
)
```

Delete :

In MongoDB, delete operations allow you to remove documents from a collection. There are two primary methods for deleting documents:

1. deleteOne: Deletes a single document that matches a specified filter.
2. deleteMany: Deletes all documents that match a specified filter.

Syntax:

Basic syntax of remove() method is as follows –

>db.COLLECTION_NAME.remove(DELLETION_CRITTERIA)

1.Delete a Single Document:

To delete a single document from the users collection where the username field is "john":

```
db.users.deleteOne({ "username": "john" })
```

2.Delete Multiple Documents:

To delete multiple documents from the users collection where the status field is "inactive":

3.Remove a Single Document (Deprecated):

To remove a single document from the users collection where the age field is greater than 50

DOCUMENTS, COLLECTIONS

Documents:

In MongoDB, the data records are stored as BSON documents. Here, BSON stands for binary representation of JSON documents, although BSON contains more data types as compared to JSON. The document is created using field-value pairs or key-value pairs and the value of the field can be of any BSON type

Syntax:

```
{
field1: value1
field2: value2
....
fieldN: valueN
}
```

Collection:

Collections are just like tables in relational databases, they also store data, but in the form of documents. A single database is allowed to store multiple collections.

Syntax:

```
db.collection_name.insertOne({..})
```

CLASS 3

WHERE , CRUD

CRUD Operations:

(Create, Read, Update, and Delete) are the basic set of operations that allow users to interact with the MongoDB server.
As we know, to use MongoDB we need to interact with the MongoDB server to perform certain operations like entering new data into the application, updating data into the application, deleting data from the application, and reading the application data.

Perform CRUD Operations in MongoDB:

Now that we know the components of the CRUD operation, let's learn about each individual operation in MongoDB. We will know what each operation does, and the methods to perform these operations in MongoDB.

```
test> const studentData = {
...       "name":"Alice Smith",
...       "age":22,
...       "courses":["Mathematics","Computer Science","English"],
...       "gpa":3.8,
...       "home_city":"New York",
...       "blood_group":"A+",
...       "is_hotel_resident":false
...       };

test> db.stu.insertOne(studentData);
{
  acknowledged: true,
  insertedId: ObjectId('665b529e49389824aecdcdf7')
}
test>
```

INSERTION:

In MongoDB, the `insertOne` method is used to insert a single document into a collection. It takes a single parameter, which is an object representing the document to be inserted. Upon successful insertion, it returns an object with an `acknowledged` field set to true and an `insertedId` field containing the unique identifier (`ObjectId`) assigned to the newly inserted document. This method is useful for adding individual documents to a collection in MongoDB.

```
db> const studentData={
... "name":"Alice Smith",
... "age":22,
... "courses":["Mathematics","Computer Science","English"],
... "gpa":3.8,
... "home_city":"New York",
... "blood_group":"A+",
... "is_hotel_resident":false
... };

db> db.students.insertOne(studentData);
{
  acknowledged: true,
  insertedId: ObjectId('6661da38b0d232162dcdcdf6')
}
db>
```

Where operation:

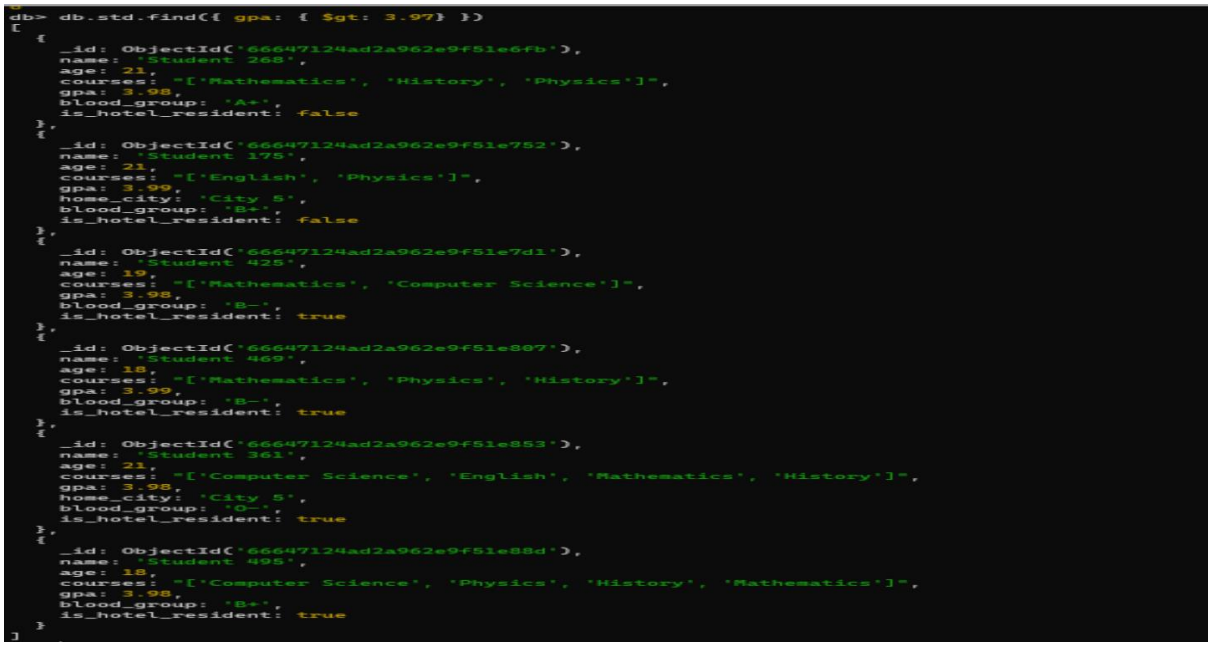
In MongoDB, the $where operator allows you to execute JavaScript code directly within a query. This operator can be used to query documents based on criteria that cannot be expressed using the regular MongoDB query language syntax.

Syntax:

```
db.collection.find({
 $where: function() {
 // JavaScript code to evaluate
 }
});
```

```
db> db.std.find({ home_city:"City 4"} ).count()
27
```

```
db> db.std.find({ gpa: { $gt: 3.97} }).count()
6
db>
```



```
db> db.std.find({ gpa: { $gt: 3.97} })
[
  {
    _id: ObjectId('66647124ad2a962e9f51e6fb'),
    name: 'Student 268',
    age: 21,
    courses: "['Mathematics', 'History', 'Physics']",
    gpa: 3.98,
    blood_group: 'A+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('66647124ad2a962e9f51e752'),
    name: 'Student 175',
    age: 21,
    courses: "['English', 'Physics']",
    gpa: 3.99,
    home_city: 'City 5',
    blood_group: 'B+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('66647124ad2a962e9f51e7d1'),
    name: 'Student 425',
    age: 19,
    courses: "['Mathematics', 'Computer Science']",
    gpa: 3.98,
    blood_group: 'B-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('66647124ad2a962e9f51e807'),
    name: 'Student 469',
    age: 18,
    courses: "['Mathematics', 'Physics', 'History']",
    gpa: 3.99,
    blood_group: 'B-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('66647124ad2a962e9f51e853'),
    name: 'Student 361',
    age: 21,
    courses: "['Computer Science', 'English', 'Mathematics', 'History']",
    gpa: 3.98,
    home_city: 'City 5',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('66647124ad2a962e9f51e88d'),
    name: 'Student 495',
    age: 18,
    courses: "['Computer Science', 'Physics', 'History', 'Mathematics']",
    gpa: 3.98,
    blood_group: 'B+',
    is_hotel_resident: true
  }
]
```

CLASS 4

PROJECTION ,LIMIT ,SELECTORS

Projection:

In MongoDB, projection refers to the operation of selecting only the necessary data fields rather than retrieving the entire document. This can be particularly useful when querying large collections, as it reduces the amount of data transferred over the network and potentially improves query performance.

The projection work based on:

1 indicates that the field should be included name to be returned. 0 indicates that the field should be excluded them from the result.

Example: {
"_id": 1,
"name": "Alice",
"age": 30,
"email": "alice@example.com",
"address": {
"city":
"New York",
"zipcode": "10001"
}
}

Get Selected Attributes:

To select specific fields in a MongoDB query, you use the find() method with a projection document. The projection document specifies which fields to include (using 1) or exclude (using 0).

Syntax:

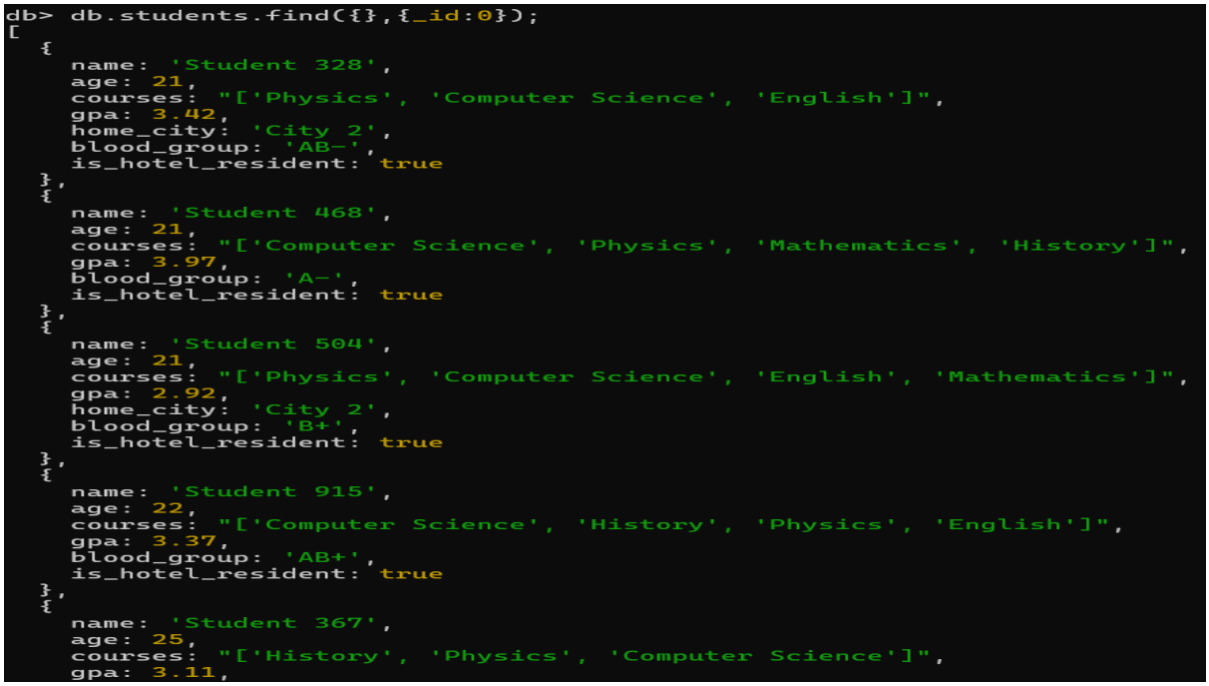
db. students. find ({}, { _id: 0 })

```
b> db.students.find({},{_id:0});

[
  {
    name: 'Student 328',
    age: 21,
    courses: "['Physics', 'Computer Science', 'English']",
    gpa: 3.42,
    home_city: 'City 2',
    blood_group: 'AB-',
    is_hotel_resident: true
  },
  {
    name: 'Student 468',
    age: 21,
    courses: "['Computer Science', 'Physics', 'Mathematics', 'History']",
    gpa: 3.97,
    blood_group: 'A-',
    is_hotel_resident: true
  },
  {
    name: 'Student 504',
    age: 21,
    courses: "['Physics', 'Computer Science', 'English', 'Mathematics']",
    gpa: 2.92,
    home_city: 'City 2',
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    name: 'Student 915',
    age: 22,
    courses: "['Computer Science', 'History', 'Physics', 'English']",
    gpa: 3.37,
    blood_group: 'AB+',
    is_hotel_resident: true
  },
  {
    name: 'Student 367',
    age: 25,
    courses: "['History', 'Physics', 'Computer Science']",
    gpa: 3.11,
```

Ignore attributes:

In MongoDB, ignoring an attribute (or field) in query results is done using projection. By setting the value of a field to 0 in the projection document, you can exclude that field from the returned documents. Db. students. find({},{_id:0})



```
db> db.students.find({},{_id:0});
[
  {
    name: 'Student 328',
    age: 21,
    courses: "['Physics', 'Computer Science', 'English']",
    gpa: 3.42,
    home_city: 'City 2',
    blood_group: 'AB-',
    is_hotel_resident: true
  },
  {
    name: 'Student 468',
    age: 21,
    courses: "['Computer Science', 'Physics', 'Mathematics', 'History']",
    gpa: 3.97,
    blood_group: 'A-',
    is_hotel_resident: true
  },
  {
    name: 'Student 504',
    age: 21,
    courses: "['Physics', 'Computer Science', 'English', 'Mathematics']",
    gpa: 2.92,
    home_city: 'City 2',
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    name: 'Student 915',
    age: 22,
    courses: "['Computer Science', 'History', 'Physics', 'English']",
    gpa: 3.37,
    blood_group: 'AB+',
    is_hotel_resident: true
  },
  {
    name: 'Student 367',
    age: 25,
    courses: "['History', 'Physics', 'Computer Science']",
    gpa: 3.11,
```

Limits:

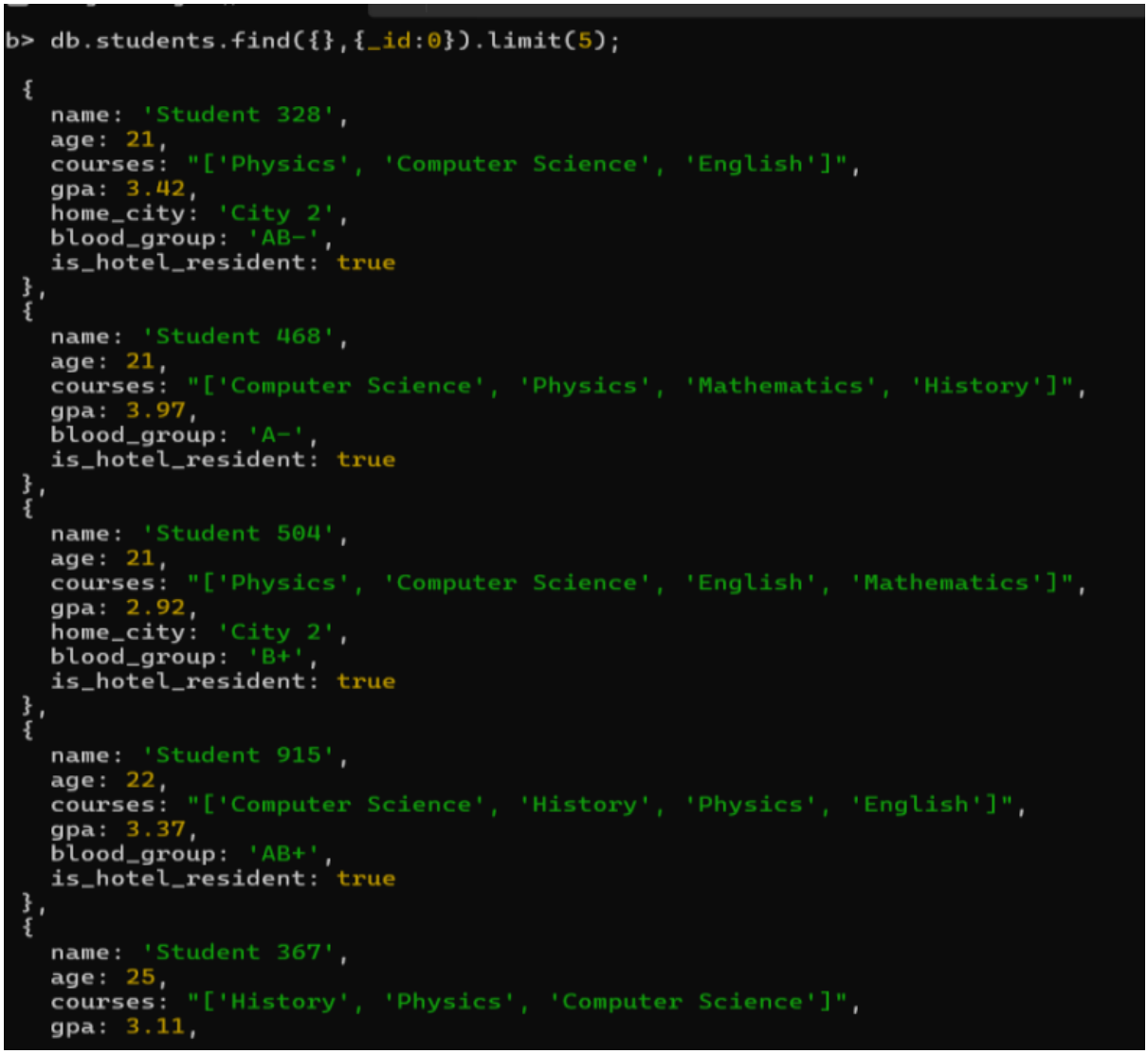
In MongoDB, you can use the limit() method to specify the maximum number of documents to be returned by a query. The syntax for using limit() is simple

Syntax:

db. collection. find({filter},{projection}).limit(number)

Get first 5 document:

db. Students .find({},{_id:0}).limit(5);



```
b> db.students.find({},{_id:0}).limit(5);

 {
   name: 'Student 328',
   age: 21,
   courses: "['Physics', 'Computer Science', 'English']",
   gpa: 3.42,
   home_city: 'City 2',
   blood_group: 'AB-',
   is_hotel_resident: true
 },
 {
   name: 'Student 468',
   age: 21,
   courses: "['Computer Science', 'Physics', 'Mathematics', 'History']",
   gpa: 3.97,
   blood_group: 'A-',
   is_hotel_resident: true
 },
 {
   name: 'Student 504',
   age: 21,
   courses: "['Physics', 'Computer Science', 'English', 'Mathematics']",
   gpa: 2.92,
   home_city: 'City 2',
   blood_group: 'B+',
   is_hotel_resident: true
 },
 {
   name: 'Student 915',
   age: 22,
   courses: "['Computer Science', 'History', 'Physics', 'English']",
   gpa: 3.37,
   blood_group: 'AB+',
   is_hotel_resident: true
 },
 {
   name: 'Student 367',
   age: 25,
   courses: "['History', 'Physics', 'Computer Science']",
   gpa: 3.11,
```

Limiting Results:

Limiting results in MongoDB is an important technique for controlling the amount of data returned by a query. This can help in optimizing performance, managing large datasets, and implementing pagination.

Syntax:

db. collection. find().limit()

Example :db. students. find({ gpa :{$gt:3.5} },{_id:0}).limit(2);

```
db> db.students.find({gpa:{$gt:3.5}},{_id:0}).limit(2);
[
  {
    name: 'Student 468',
    age: 21,
    courses: "['Computer Science', 'Physics', 'Mathematics', 'History']",
    gpa: 3.97,
    blood_group: 'A-',
    is_hotel_resident: true
  },
  {
    name: 'Student 969',
    age: 24,
    courses: "['History', 'Mathematics', 'Physics', 'English']",
    gpa: 3.71,
    blood_group: 'B+',
    is_hotel_resident: true
  }
]
db>
```

Selectors:

- AND operation
- OR operation
- Comparison between gt and lt:

Grater than:

The $gt operator in MongoDB is used to specify a query condition where the field value must be greater than a specified value. It stands for "greater than."

Lesser than:

the less than operation is performed using the $lt operator. This operator allows you to query documents where a specific field's value is less than a given value.

Greater Than ($gt):

• **Functionality:** The $gt operator selects documents where the value of a specified field is greater than (but not equal to) a given value.

• **Example:** db. collection. find({ field: { $gt: value } }) selects documents where the field value is greater than value.

• **Use Cases:** Filtering documents with values exceeding a certain threshold, selecting the latest entries based on a timestamp, or retrieving documents with numerical values above a specific limit.

Less Than ($lt):

• **Functionality:** The $lt operator selects documents where the value of a specified field is less than (but not equal to) a given value.

• **Example:** db. collection. find({ field: { $lt: value } }) selects documents where the field value is less than value.

• **Use Cases**: Filtering documents with values below a certain threshold, selecting older entries based on a timestamp, or retrieving documents with numerical values below a specific limit.

Ex: db. students. find({age:{$gt:20} });

```
db> db.students.find({age:{$gt:20}});
[
  {
    _id: ObjectId('665609bde575a33e43e69479'),
    name: 'Student 328',
    age: 21,
    courses: "['Physics', 'Computer Science', 'English']",
    gpa: 3.42,
    home_city: 'City 2',
    blood_group: 'AB-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665609bde575a33e43e6947c'),
    name: 'Student 468',
    age: 21,
    courses: "['Computer Science', 'Physics', 'Mathematics', 'History']",
    gpa: 3.97,
    blood_group: 'A-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665609bde575a33e43e6947d'),
    name: 'Student 504',
    age: 21,
    courses: "['Physics', 'Computer Science', 'English', 'Mathematics']",
    gpa: 2.92,
    home_city: 'City 2',
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665609bde575a33e43e6947f'),
    name: 'Student 915',
    age: 22,
    courses: "['Computer Science', 'History', 'Physics', 'English']",
    gpa: 3.37,
    blood_group: 'AB+',
    is_hotel_resident: true
  },
  {
```