

# Assignment 2: Mongo DB

## <Milestone 1: Written Submission>

Student Name: Keewoo Moon

Student Number: s3831370

### Task 1: CRUD tasks on listingsAndReviews Document Collection

<b>Q1. (a)</b>	<p><b>Mongo Shell:</b> <code>db.listingsAndReviews.find({\$and:[{"address.market":"Barcelona", bedrooms: 3, property_type:"Apartment"}]})</code></p> <p><b>Mongo Compass (On Filter field):</b> {bedrooms: 3, "address.market": "Barcelona", property_type: "Apartment"}</p>
<b>Q1. (b)</b>	<p><b>Mongo Shell:</b> <code>db.listingsAndReviews.find({"address.market": "Barcelona", amenities:{\$all:["Wifi", "Cable TV"]})</code></p> <p><b>Mongo Compass (On Filter field):</b> {"address.market": "Barcelona", "amenities": {"\$all": ["Wifi", "Cable TV"]}}</p>
<b>Q1. (c)</b>	<p><b>Mongo Shell:</b> <code>db.listingsAndReviews.aggregate([{\$group: {_id: null, avgPrice: {\$avg: '\$price'}}}, {\$match: {"address.market": "Barcelona"}}])</code></p> <p><b>Mongo Compass:</b></p> <ul style="list-style-type: none"><li>• Step 1: Click on the "Aggregations" tab located above the document list.</li><li>• Step 2: Add a <b>\$match</b> stage to filter documents with the "address.market" field equal to "Barcelona." (To do this)<ul style="list-style-type: none"><li>○ Click the "+" button below the "Stages" section to add a new stage.</li><li>○ In the stage dropdown, select "<b>\$match</b>"</li><li>○ In the "Field" dropdown, enter like this -&gt; "address.market": "Barcelona"</li></ul></li><li>• Step 3: Click the "Add" button to add the \$match stage to the pipeline.</li><li>• Step 4: Next, add a <b>\$group</b> stage to calculate the average price for Barcelona properties. (To do this)<ul style="list-style-type: none"><li>○ Click the "+" button below the "Stages" section to add a new stage.</li><li>○ In the stage dropdown, select "<b>\$group</b>"</li><li>○ In the "Field" dropdown, enter like this -&gt; <code>_id: null, avgPrice: { \$avg: "\$price" }</code></li></ul></li><li>• Step 5: Click the "Add" button to add the \$group stage to the pipeline.</li></ul>

	<ul style="list-style-type: none"> <li>Step 6: Click the "Run" button to execute the aggregation.</li> </ul>
Q1. (d)	<p><b>Mongo Shell:</b> <code>db.listingsAndReviews.aggregate([{\$match: { "bedrooms": 3, "address.market": "Barcelona" } }, {\$group: { _id: null, avgPrice: { \$avg: "\$price" } } }])</code></p> <p><b>Mongo Compass:</b></p> <ul style="list-style-type: none"> <li>Step 1: Click on the "Aggregations" tab located above the document list.</li> <li>Step 2: Add a <b>\$match</b> stage to filter documents with the "address.market" field equal to "Barcelona." (To do this) <ul style="list-style-type: none"> <li>Click the "+" button below the "Stages" section to add a new stage.</li> <li>In the stage dropdown, select "<b>\$match</b>"</li> <li>In the "Field" dropdown, enter "bedrooms": 3</li> <li>Do it sample as "address.market": "Barcelona"</li> </ul> </li> <li>Step 3: Click the "Add" button to add the \$match stage to the pipeline.</li> <li>Step 4: Next, add a <b>\$group</b> stage to calculate the average price for Barcelona properties. (To do this) <ul style="list-style-type: none"> <li>Click the "+" button below the "Stages" section to add a new stage.</li> <li>In the stage dropdown, select "<b>\$group</b>"</li> <li>In the "Field" dropdown, enter like this -&gt; <code>_id: null, avgPrice: { \$avg: "\$price" }</code></li> </ul> </li> <li>Step 5: Click the "Add" button to add the <b>\$group</b> stage to the pipeline.</li> <li>Step 6: Click the "Run" button to execute the aggregation.</li> </ul>
Q1. (e)	<p><b>Mongo Shell:</b> <code>db.getCollection('listingsAndReviews').aggregate([  { \$match: { "address.market": "Barcelona", "bedrooms": 3 } },  { \$sort: { "price": 1 } },  {  \$group: {  _id: null,  count: { \$sum: 1 },  values: { \$push: "\$price" }  }  },  {  \$addFields: {  middleIndex: { \$divide: [ "\$count", 2 ] },  isEven: { \$eq: [ { \$mod: [ "\$count", 2 ] }, 0 ] }  }  },  {  \$unwind: {  path: "\$values",  includeArrayIndex: "index"  }</code></p>

```

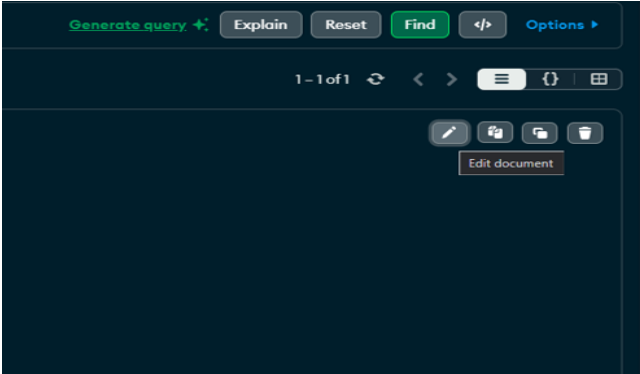
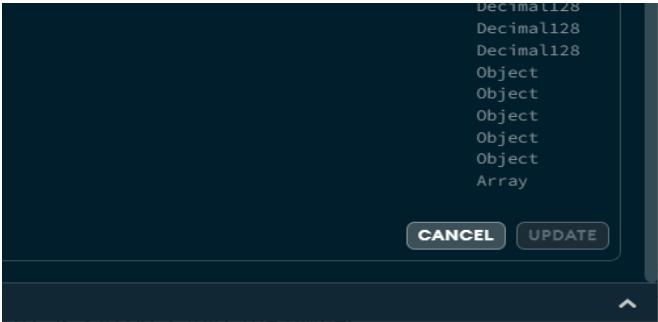
    }
  },
  {
    $match:
    {
      $expr:
      {
        $or:
        [
          { $and: [{ "isEven": true }, { $eq: ["$index", "$middleIndex"] } ] },
          { $and: [{ "isEven": false }, { $gte: ["$index", "$middleIndex"] } ] }
        ]
      }
    }
  },
  {
    $group:
    {
      _id: null,
      medianValues: { $push: "$values" }
    }
  },
  {
    $unwind:
    {
      path: "$medianValues"
    }
  }
}
))

```

#### **Mongo Compass:**

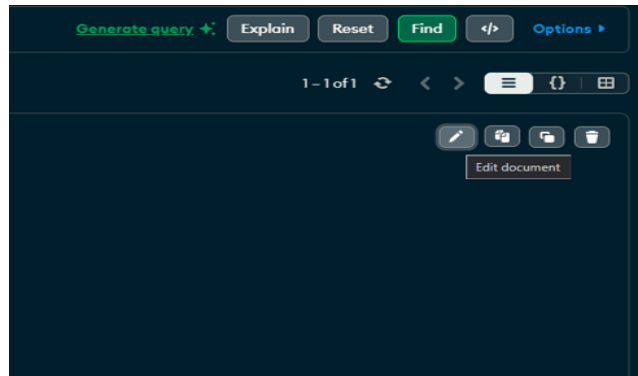
- Step 1: Click on the "Aggregations" tab located above the document list.
- Step 2: **\$match**  
 { "address.market": "Barcelona", "bedrooms": 3 }
- Step 3: **\$sort**  
 { "price": 1 }
- Step 4: **\$group**  
 { "\_id": null, "count": { "\$sum": 1 }, "values": { "\$push": "\$price" } }
- Step 5: **\$addFields**  
 { "middleIndex": { "\$divide": [ "\$count", 2 ] }, "isEven": { "\$eq": [ { "\$mod": [ "\$count", 2 ] }, 0 ] } }
- Step 6: **\$unwind**  
 { "path": "\$values", "includeArrayIndex": "index" }

	<ul style="list-style-type: none"> <li>• Step 7: <b>\$match</b>  <pre>{   "\$expr": {     "\$or": [       { "\$and": [{ "isEven": true }, { "\$eq": ["\$index", "\$middleIndex"] } ] },       { "\$and": [{ "isEven": false }, { "\$gte": ["\$index", "\$middleIndex"] } ] }     ]   } }</pre> </li> <li>• Step 8: <b>\$group</b>  <pre>{ "_id": null, "medianValues": { "\$push": "\$values" } }</pre> </li> <li>• Step 9: <b>\$unwind</b>  <pre>{ "path": "\$medianValues" }</pre> </li> </ul>
<b>Q1. (f)</b>	<p><b>Mongo Shell:</b> <code>db.listingsAndReviews.aggregate([ { \$group: { _id: "\$address.market", avgPrice: { \$avg: "\$price" } } }, { \$sort: { avgPrice: 1 } }, { \$limit: 10 } ])</code></p> <p><b>Mongo Compass:</b></p> <ul style="list-style-type: none"> <li>• Step 1: Click on the "Aggregations" tab located above the document list.</li> <li>• Step 2: Add a <b>\$group</b> stage to filter documents – compute the average price, by the city.        (To do this)       <ul style="list-style-type: none"> <li>○ Click the "+" button below the "Stages" section to add a new stage.</li> <li>○ In the stage dropdown, select "<b>\$group</b>"</li> <li>○ In the "Field" dropdown, enter like this -&gt; <code>_id: "\$address.market", avgPrice: { \$avg: "\$price" }</code></li> </ul> </li> <li>• Step 3: Click the "Add" button to add the <b>\$group</b> stage to the pipeline.</li> <li>• Step 4: Next, add a <b>\$sort</b> stage to sort by average price.        (To do this)       <ul style="list-style-type: none"> <li>○ Click the "+" button below the "Stages" section to add a new stage.</li> <li>○ In the stage dropdown, select "<b>\$sort</b>"</li> <li>○ In the "Field" dropdown, enter like this -&gt; <code>avgPrice: 1</code></li> </ul> </li> <li>• Step 5: Click the "Add" button to add the <b>\$sort</b> stage to the pipeline.</li> <li>• Step 6: Next, add a <b>\$limit</b> stage to display first 10 documents.        (To do this)       <ul style="list-style-type: none"> <li>○ Click the "+" button below the "Stages" section to add a new stage.</li> <li>○ In the stage dropdown, select "<b>\$limit</b>"</li> <li>○ In the "Field" dropdown, enter like this -&gt; <code>10</code></li> </ul> </li> <li>• Step 7: Click the "Run" button to execute the aggregation.</li> </ul>

<p><b>Q2.</b></p>	<p><b>Mongo Shell:</b> <code>db.listingsAndReviews.updateOne({ name: "Be Happy in Porto" }, { \$push: { amenities: "Netflix" } })</code></p> <p><b>Mongo Compass (On Filter field):</b></p> <ul style="list-style-type: none"> <li>Step 1: enter <code>{name: "Be Happy in Porto"}</code> on Filter field of Mongo Compass to filter name is "Be Happy in Porto"</li> <li>Step 2: click <b>"Edit document"</b> button in the top right side. This is a screenshot to show that.</li></ul>  <ul style="list-style-type: none"> <li>Step 3: extend amenities array and <b>add item</b> just after 31. "Netflix"</li> <li>Step 4: click <b>"update"</b> button in the bottom of right corner.</li></ul> 
<p><b>Q3.</b></p>	<p><b>Mongo Shell:</b> <code>db.listingsAndReviews.updateOne(</code>  <code>{ name: "Be Happy in Porto" },</code>  <code>{</code>  <code>  \$push: {</code>  <code>    reviews: {</code>  <code>      _id: "510014317",</code>  <code>      date: new Date(),</code>  <code>      listing_id: "10083468",</code>  <code>      reviewer_id: "72603902",</code>  <code>      reviewer_name: "Paul2023",</code>  <code>      comments: "This holiday accommodation did not meet my expectation. Being in</code>  <code>      Portugal, I wanted to watch bull-fighting from the balcony. But, neither balcony nor bull-</code>  <code>      fighting nearby are there."</code>  <code>    }</code>  <code>  }</code>  <code>}</code>  <code>)</code></p>

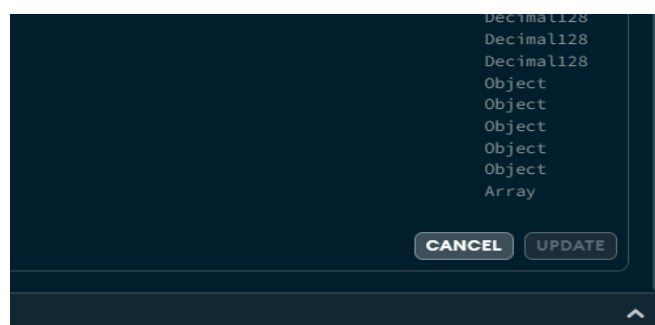
### Mongo Compass:

- Step 1: enter `{name: "Be Happy in Porto"}` on Filter field of Mongo Compass to filter name is "Be Happy in Porto"
- Step 2: click **"Edit document"** button in the top right side. This is a screenshot to show that.



- Step 3: extend reviews array and **add item** just after 178 like this ->  

```
{
  "_id": "510014317",
  "date": {
    "$date": "2023-10-11T15:47:59.893Z"
  },
  "listing_id": "10083468",
  "reviewer_id": "72603902",
  "reviewer_name": "Paul2023",
  "comments": "This holiday accommodation did not meet my expectation. Being in Portugal, I wanted to watch bull-fighting from the balcony. But, neither balcony nor bull-fighting nearby are there."
}
```
- Step 4: click **"update"** button in the bottom of right corner.

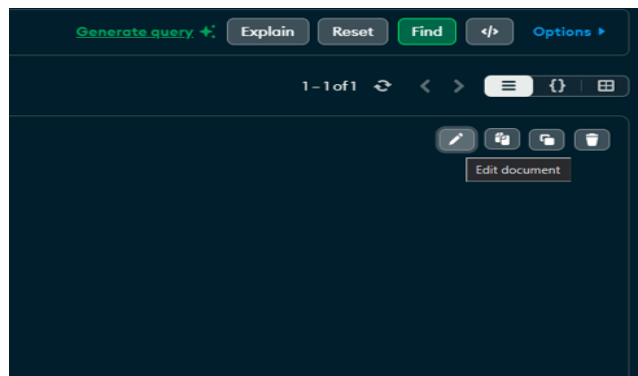


**Q4.** **Mongo Shell:** `db.listingsAndReviews.updateOne(`  
    `{ name: "Be Happy in Porto" },`  
    `{ $set: { "price": 40.00 } }`  
    `)`

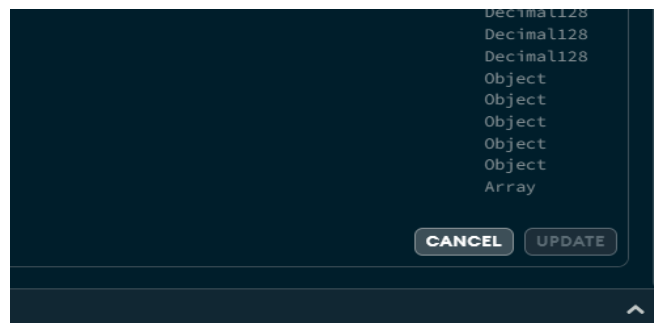
### Mongo Compass (On Filter field):

- Step 1: enter `{name: "Be Happy in Porto"}` on Filter field of Mongo Compass to filter name is "Be Happy in Porto"

- Step 2: click “**Edit document**” button in the top right side. This is a screenshot to show that.



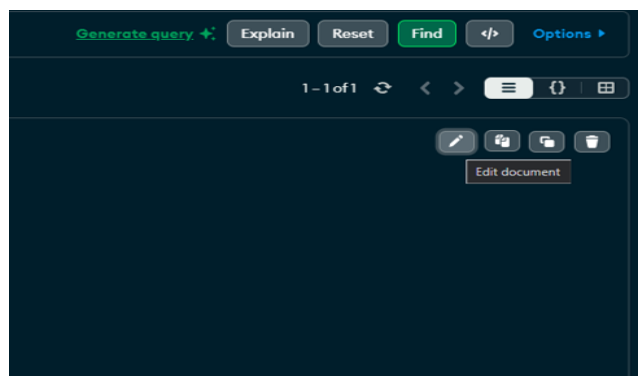
- Step 3: find price row in the document and **change value** from 30 to 40
- Step 4: click “**update**” button in the bottom of right corner.



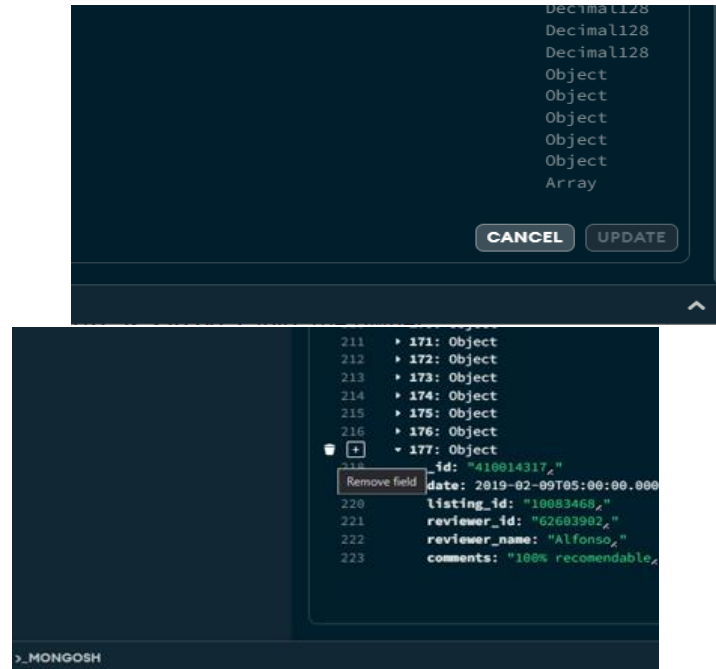
**Q5.** **Mongo Shell:** `db.listingsAndReviews.updateOne(`  
     `{ name: "Be Happy in Porto" },`  
     `{ $pull: { "reviews": { _id: "510014317" } } }`  
   `)`

**Mongo Compass (On Filter field):**

- Step 1: enter `{name: "Be Happy in Porto"}` on Filter field of Mongo Compass to filter name is “Be Happy in Porto”
- Step 2: click “**Edit document**” button in the top right side. This is a screenshot to show that.



- Step 3: extend reviews array and **delete item** -> 178: Object. This is a screenshot to show that.
- Step 4: click “**update**” button in the bottom of right corner.





## Task 2: Extend the AirBnB database

### Q1. Designing an extended database for AirBnB

**Approach:** Embedded

**Additional fields:** bookings

- Within listingsAndReviews collection (existing)
- bookings (New): This new field will store booking-related information. It will be embedded within each listing document in the listingsAndReviews existing collection.

**Fields in the bookings (New):**

- booking\_id: unique booking ID
- checkInDate: The pre-defined check-in date for the listing.
- checkOutDate: The pre-defined check-out date for the listing.
- details: An array of booking documents for each listing. Each booking document should include:
  - clientName: Name of the client making the booking
  - emailAddress: Client's email address
  - daytimePhoneNumber: Client's daytime phone number
  - mobileNumber: Client's mobile number
  - postalAddress: Client's postal address
  - homeAddress: Client's home address
  - depositPaid: Amount paid as a deposit at the time of booking
  - balanceDue: Amount remaining to be paid
  - balanceDueDate: The due date for the balance payment
  - numOfGuests: Number of guests for this booking
  - guests: An array of guest documents, each containing:
    - guestName: Name of the guest
    - guestAge: Age of the guest

Fields example:

```
{
  { _id: "10006546" },
  {$push: {
    bookings: {
      "booking_id": "1",
      "checkInDate": "2023-11-01",
      "checkOutDate": "2023-11-10",
      "details": {
        "clientName": "John Doe",
        "emailAddress": "johndoe@example.com",
        "daytimePhoneNumber": "(03)53497498",
        "mobileNumber": "0468987456",
        "postalAddress": "55 Clifton Street Narrung Victoria",
        "homeAddress": "55 Clifton Street Narrung Victoria",
        "depositPaid": 200.00,
        "balanceDue": 800.00,
        "balanceDueDate": "2023-10-01",
        "numOfGuests": 3,
        "guests": [
```

	<pre> {   "guestName": "Sarah Smith",   "guestAge": 30 }, {   "guestName": "Kyle Walker",   "guestAge": 35 }, {   "guestName": "Jack Taylor",   "guestAge": 28 } ] } } } } </pre>
Q2.	<p>The chosen data model for the extended Airbnb database is the <b>embedded data model</b>. Because this data model offers several advantages in the context of this application. First of all, this model is characterized by storing booking information directly within the listing documents. The primary advantages of this model include its <b>simplicity, faster read operations, strong data integrity</b>, and <b>suitability for a database of moderate size</b>.</p> <p>Secondly, in the embedded model, booking details are <b>seamlessly integrated into the listing documents</b>, simplifying the overall database structure. This streamlined design aligns with the inherent "one listing, one or more bookings" relationship, making the system easier to manage and query. As a result, it allows for faster read operations, as booking information is readily available when querying a listing, thus reducing the need for complex joins or additional queries.</p> <p>Furthermore, the embedded model ensures <b>data integrity</b> by establishing a strong connection between a listing and its associated bookings. Any changes made to a listing are automatically reflected in its associated bookings, reducing the risk of data inconsistencies. This model is particularly well-suited for applications where document sizes remain within database limits and where database scaling requirements are not overly demanding.</p> <p>On the other hand, the <b>referenced data model</b> was not chosen for this application due to its <b>complexities, additional query requirements, potential data consistency challenges, increased development effort</b>, and <b>scalability concerns</b>, particularly in larger databases. The referenced model would have involved creating a separate "bookings" collection and using references or foreign keys to link bookings to listings, introducing unnecessary complexities and potential performance bottlenecks. In summary, the embedded data model provides a straightforward and efficient approach to managing booking information within the extended Airbnb database.</p>

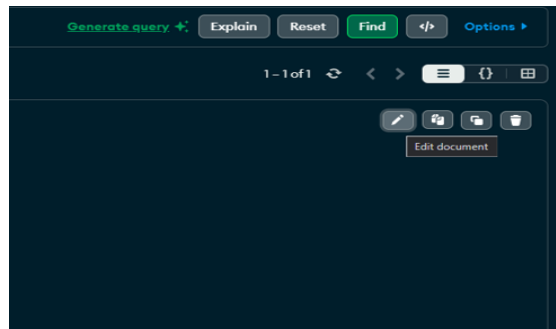
**Q3.**  
**(a)**

```
Mongo Shell: db.listingsAndReviews.updateOne({ _id: "10006546" },
{
  $push: {
    bookings: {
      "booking_id": "1",
      "checkInDate": ISODate("2023-11-01T00:00:00.000+00:00"),
      "checkOutDate": ISODate("2023-11-10T00:00:00.000+00:00"),
      "details": {
        "clientName": "John Doe",
        "emailAddress": "johndoe@example.com",
        "daytimePhoneNumber": "(03)53497498",
        "mobileNumber": "0468987456",
        "postalAddress": "55 Clifton Street Narrung Victoria",
        "homeAddress": "55 Clifton Street Narrung Victoria",
        "depositPaid": 200.00,
        "balanceDue": 800.00,
        "balanceDueDate": ISODate("2023-10-01T00:00:00.000+00:00"),
        "numOfGuests": 3,
        "guests": [
          {
            "guestName": "Sarah Smith",
            "guestAge": 30
          },
          {
            "guestName": "Kyle Walker",
            "guestAge": 35
          },
          {
            "guestName": "Jack Taylor",
            "guestAge": 28
          }
        ]
      }
    }
  }
})
```

**Mongo Compass:**

- Step 1: enter { \_id: " 10006546" } on Filter field of Mongo Compass to filter document id is "10006546"

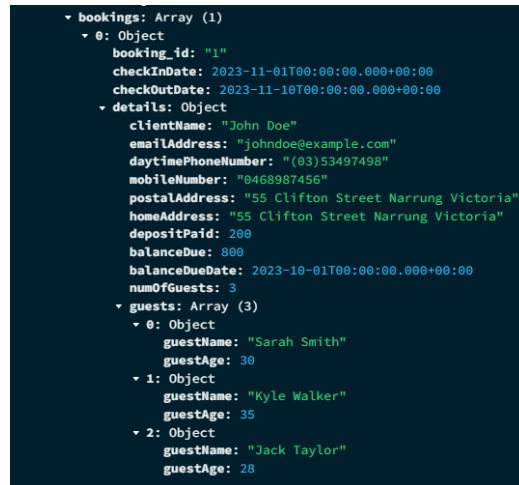
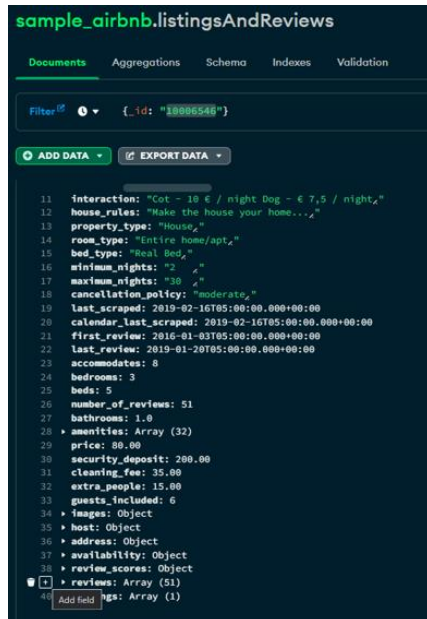
- Step 2: click “**Edit document**” button in the top right side. This is a screenshot to show that.



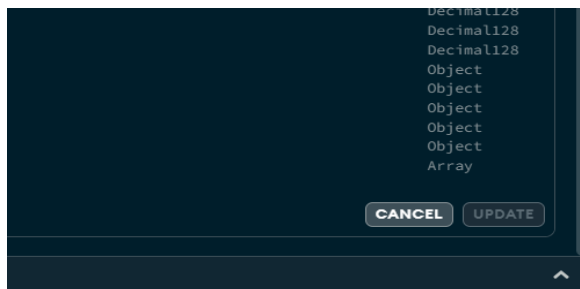
- Step 3: **add field** after reviews (row number 39), like this JSON query -> bookings: {

```
"booking_id": "1",
"checkInDate": ISODate("2023-11-01T00:00:00.000+00:00"),
"checkOutDate": ISODate("2023-11-10T00:00:00.000+00:00"),
"details": {
  "clientName": "John Doe",
  "emailAddress": "johndoe@example.com",
  "daytimePhoneNumber": "(03)53497498",
  "mobileNumber": "0468987456",
  "postalAddress": "55 Clifton Street Narrung Victoria",
  "homeAddress": "55 Clifton Street Narrung Victoria",
  "depositPaid": 200.00,
  "balanceDue": 800.00,
  "balanceDueDate": "2023-10-01",
  "numOfGuests": 3,
  "guests": [
    {
      "guestName": "Sarah Smith",
      "guestAge": 30
    },
    {
      "guestName": "Kyle Walker",
      "guestAge": 35
    },
    {
      "guestName": "Jack Taylor",
      "guestAge": 28
    }
  ]
}
```

This is a screenshot to show that.



- Step 4: click “update” button in the bottom right corner.



**Q3.**  
**(b)**

```
Mongo Shell: db.listingsAndReviews.updateOne(
  { "name": "Ribeira Charming Duplex" },
  {
    $pull: {
      "bookings": {
        "checkInDate": ISODate("2023-11-01T00:00:00.000+00:00")
      }
    }
  }
)
```

// depends Date type

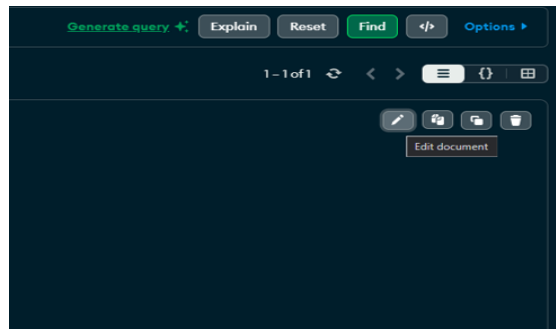
// “checkInDate”: “2023-11-01”

// "checkInDate": new ISODate("2023-11-01T00:00:00.000+00:00")

**Mongo Compass:**

- Step 1: enter {name: “Ribeira Charming Duplex”} on Filter field of Mongo Compass to filter the document name is “Ribeira Charming Duplex”

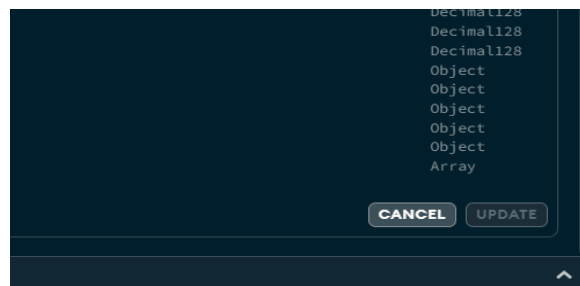
- Step 2: click “**Edit document**” button in the top right side. This is a screenshot to show that.



- Step 3: In line number 40, extend bookings array and **delete field** -> 0: Object. This is a screenshot to show that.



- Step 4: click “**update**” button in the bottom of right corner.



## Bonus Question

### Step 1: Define Check-in and Check-out dates

```
// Step 1)
// Define the check-in and check-out dates for the new booking
const checkInDate = new ISODate("2023-11-01T00:00:00.000+00:00");
const checkOutDate = new ISODate("2023-11-10T00:00:00.000+00:00");
```

In this step, the check-in and check-out dates for the new booking are defined as ISODate objects, specifying the exact date and time.

### Step 2: Check if Requested Dates and Vacant

```
// Step 2)
// Check if the requested dates are vacant
const isVacant = db.listingsAndReviews.findOne({
  "_id": "10006546",
  "bookings": {
    $not: {
      $elemMatch: {
        $or: [
          {
            "checkInDate": { $lt: checkOutDate },
            "checkOutDate": { $gt: checkInDate }
          }
        ]
      }
    }
  }
});
```

In this step, a query is made to check if the requested check-in and check-out dates are vacant for a specific property listing. The query searches for the property by its "\_id" and checks if there are no overlapping bookings using the \$elemMatch operator. Overlapping bookings are identified using the \$or condition, ensuring that either the check-in or check-out dates fall within an existing booking.

### Step 3: Proceed with Inserting the New Booking

```
// Step 3)
if (!isVacant) {
  // Dates are vacant, proceed to insert the new booking
  db.listingsAndReviews.updateOne(
    { "_id": "10006546" },
    {
      $set: {
        bookings: {
          "booking_id": "1",
          "booking_info": {
```

```

        "checkInDate": checkInDate,
        "checkOutDate": checkOutDate,
        "depositPaid": 200.00,
        "balanceDue": 800.00,
        "balanceDueDate": ISODate("2023-10-01T00:00:00.000+00:00"),
    },
    "client_details": {
        "clientName": "John Doe",
        "emailAddress": "johndoe@example.com",
        "daytimePhoneNumber": "(03)53497498",
        "mobileNumber": "0468987456",
        "postalAddress": "55 Clifton Street Narrung Victoria",
        "homeAddress": "55 Clifton Street Narrung Victoria",
        "guests": [
            {
                "guestName": "Sarah Smith",
                "guestAge": 30,
            },
            {
                "guestName": "Kyle Walker",
                "guestAge": 35,
            },
            {
                "guestName": "Jack Taylor",
                "guestAge": 28,
            },
        ]
    },
    bookingCalendar: {
        "2023-11-01": false,
        "2023-11-02": false,
        "2023-11-03": false,
        "2023-11-04": false,
        "2023-11-05": false,
        "2023-11-06": false,
        "2023-11-07": false,
        "2023-11-08": false,
        "2023-11-09": false
    }
}
};
}

```

If the requested dates are vacant (no overlap with existing bookings), the code proceeds to insert the new booking details into the property listing using the updateOne method. This includes client information, booking details, and updating the booking calendar to mark the specified dates as "occupied."



**Step 4: Provide Feedback if Dates are Not Vacant**

```
//Step 4)
else {
    // Dates are not vacant, provide appropriate feedback to the client
    print("Sorry, these dates were booked already");
}
```

If the requested dates are not vacant (overlap with existing bookings), the code prints a message indicating that the dates are already booked, providing appropriate feedback to the client.