**Database Applications – A2**

# Task 1: CRUD tasks on listingsAndReviews Document Collection

## Statement 1

### 1A: Display 3-bedroom apartments in Barcelona.

Compass

{

accommodates: { $gte: 3 },

property\_type: 'Apartment',

"address.market": "Barcelona"

}

Mongo Shell

db.**listingsAndReviews**.find({

accommodates: { $gte: 3 },

property\_type: 'Apartment',

"address.market": "Barcelona"

});

### 1B: listings in Barcelona that has both Wifi and Cable TV.

Compass

{

"address.market": "Barcelona","amenities":"Wifi","amenities":"Cable TV"

}

Mongo Shell

db.**listingsAndReviews**.find({

"address.market": "Barcelona", "amenities": { $all: ["Wifi", "Cable TV"] }

});

### 1C: Compute and display the average daily rate for Barcelona properties.

Compass

Go to the aggregations tab.

Stage 1: Make “match” stage.

{"address.market": "Barcelona" }

Stage 2: Make “group” stage.

{ id: null, avgPrice: { $avg: "$price" } }

Mongo Shell

db.**listingsAndReviews**.aggregate([

{$match: {"address.market": "Barcelona"} },

{ $group: {\_id: null, // Group all documents together

avgPrice: { $avg: "$price" } } // Calculate the average price

}

])

### 1D: Display the average price for 3-bedroom apartments in Barcelona.

Compass

Go to the aggregations tab. Make 2 stages.

Make a stage “Match”

{ "address.market": "Barcelona",bedrooms: 3,property\_type: "Apartment"}

Make a stage “Group”

{ \_id: null, average\_price: {$avg: "$price" } }

Shell

db.listingsAndReviews.aggregate([

{$match: { "address.market": "Barcelona",bedrooms: 3,property\_type: "Apartment"}},

{ $group: {\_id: null,avgPrice: { $avg: "$price" } } }

])

### 1E: listings within a $10-range from the average price you found in 1D

Mongo Shell

db.listingsAndReviews.aggregate([

// Stage 1: $match

{

$match: {

"address.market": "Barcelona",

"bedrooms": 3,

"property\_type": "Apartment"

}

},

// Stage 2: $group

{

$group: {

\_id: null,

average\_price: { $avg: "$price" }

}

},

// Stage 3: $project

{

$project: {

\_id: 0,

average\_price\_lower\_bound: { $subtract: ["$average\_price", 5] },

average\_price\_upper\_bound: { $add: ["$average\_price", 5] }

}

},

// Stage 4: $lookup

{

$lookup: {

from: "listingsAndReviews",

let: {

lower\_bound: "$average\_price\_lower\_bound",

upper\_bound: "$average\_price\_upper\_bound"

},

pipeline: [

{

$match: {

$expr: {

$and: [

{ $eq: ["$address.market", "Barcelona"] },

{ $eq: ["$bedrooms", 3] },

{ $eq: ["$property\_type", "Apartment"] },

{ $gte: ["$price", "$$lower\_bound"] },

{ $lte: ["$price", "$$upper\_bound"] }

]

}

}

}

],

as: "listings\_in\_range"

}

},

// Stage 5: $unwind

{

$unwind: "$listings\_in\_range"

}

]);

Mongo Compass

Go to the aggregations tab. Make 5 stages.

Stage 1: $match

{ "address.market": "Barcelona", bedrooms: 3, property\_type: "Apartment"}

Stage 2: $Group

{ \_id: null, average\_price: { $avg: "$price"}}

Stage 3: $project

{ \_id: 0,

average\_price\_lower\_bound: { $subtract: ["$average\_price", 5]},

average\_price\_upper\_bound: {$add: ["$average\_price", 5]}

}

Stage 4: $lookup

{ from: "listingsAndReviews",

let: {

lower\_bound: "$average\_price\_lower\_bound",

upper\_bound: "$average\_price\_upper\_bound"

},

pipeline: [

{

$match: {

$expr: {

$and: [

{$eq: ["$address.market", "Barcelona"]},

{$eq: ["$bedrooms", 3]},

{$eq: ["$property\_type", "Apartment"]},

{$gte: ["$price", "$$lower\_bound"]},

{$lte: ["$price", "$$upper\_bound"]}

]

}

}

}

],

as: "listings\_in\_range"

}

Stage 5: $unwind

"$listings\_in\_range"

### 1F: Display 10 “least-expensive” tourist destinations for travelers.

Shell

db.**listingsAndReviews**.aggregate([

{ $group: { \_id: "$address.market",average\_price: { $avg: "$price" } } },

{ $sort: { average\_price: 1 } },

{ $limit: 10 }

])

Compass

Go to the aggregarions tab. Make 3 stages.

Stage 1: Make a $group stage.

{ \_id: "$address.market", average\_price: { $avg: "$price" } }

Stage 2: Make a $sort stage.

{ average\_price: 1}

Stage 3: Make a $limit stage.

10

Note: Stage 2 sorts in sorts in ascending order where smaller values come first. -1 would be descending order. I want to have least expensive. If I had -1 then 10 most expensive would be at shown.

## Statement 2

Compass

Step 1: In the document tab filter.

{ "name": "Be Happy in Porto" }

Step 2: After searching, click “Update”. Then enter the following:

{ $addToSet: { amenities: "Netflix" } }

NOTE: The $addToSet operator prevents duplicates. This ensures that "Netflix" is only added to the amenities array if it is not already present

Shell

db.listingsAndReviews.updateOne(

{ name: "Be Happy in Porto" },

{

$addToSet: { amenities: "Netflix" }

}

)

## Statement 3

Shell

db.**listingsAndReviews**.updateOne(

{ name: "Be Happy in Porto" }, // Filter to find the listing by name

{

$push: {

reviews: {

reviewer\_name: "Recent Guest",

date: new Date(), // Use the current date for the review

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.",

market: "Porto" // Assuming the 'market' field represents the location from the address field

}

}

}

)

Mongo Compass

First use the filter enter:

{ "name": "Be Happy in Porto" }

Then click “Update”. Enter the following:

{

"$push": {

"reviews": {

"reviewer\_name": "Recent Guest",

"date": new Date(),

"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.",

"market": "Porto"

}

}

}

## Statement 4

*USE to find owner (Should be 1)*

Query:{ "name": "Be Happy in Porto" ,"price":30}

Project: {name: 1, price:1, "address.market":"1"}

*USE to find in same area (Should be 554).*

Query: {"address.market":"Porto"}

Project: {price:1, "address.market":"1", name: 1}

*USE to find the number of properties in the Porto market with a price of 30 (16)*

Query: { "address.market": "Porto" , price:30}

*Sort: {price:1}*

MongoDB Compass

Go to the aggregations tab. Make 5 stages.

Stage 1: make a $match

{ name: "Be Happy in Porto" }

Stage 2: Make a $lookup

{

from: "listingsAndReviews",

let: {

property\_type: "$property\_type",

bedrooms: "$bedrooms",

market: "$address.market"

},

pipeline: [

{

$match: {

$expr: {

$and: [

{ $eq: [ "$property\_type", "$$property\_type" ]},

{$eq: ["$bedrooms", "$$bedrooms"]},

{ $eq: ["$address.market", "$$market"] }

]

}

}

},

{ $group: { \_id: null, avg\_price: { $avg: "$price" } } }

],

as: "avgPriceData"

}

**Stage 3: Make a $set stage.**

{

new\_price: {

$round: [

{ $multiply: [ {$arrayElemAt: [ "$avgPriceData.avg\_price", 0 ]}, 0.9 ] }, 2

]

}

}

Note: Creates or updates fields in the documents being processed in the pipeline. In this case, it's setting a new field called new\_price.

* We calculate and store the result of the calculated discounted price.
* We round the result to 2 decimal places.
* We multiply 2 numbers, $arrayElemAt and 0.9. This gives the 10% lower value.
* $arrayElemAt. Looks at where we retrieve our values. We look at the array, avgPriceData calculated in stage 2, then the first value, index 0. There are no other values inside the index.

E.g.

*Let’s say the average price of similar properties is* ***$50****. Here’s how the calculation proceeds in Stage 3:*

* ***Average Price****: $arrayElemAt: [ "$avgPriceData.avg\_price", 0 ] returns* ***50****.*
* ***10% Discount****: $multiply: [ 50, 0.9 ] results in* ***45****.*
* ***Rounding****: $round: [ 45, 2 ] results in* ***45.00****.*
* ***Set New Price****: The new\_price is set to* ***45.00****.*

**Stage 4: $project**

{

\_id: 1,

price: "$new\_price", // Only updating the price field

name: 1,

property\_type: 1,

bedrooms: 1,

address: 1,

// Add other necessary fields here

}

**Note:** Ensures that all the fields in the document except price are retained, so you only update price while keeping the rest intact.

**Stage 5: Merge**

{ into: "listingsAndReviews", on: "\_id", whenMatched: "merge", whenNotMatched: "discard" }

Note: merging just the price field back into the document, preventing the deletion of other fields.

**~~Stage 4: $set~~**

~~//This is for $Set~~

~~{~~

~~price: "$new\_price"~~

~~}~~

Note: I found that while this did run in the aggregation stage, it wasn’t able to update the values in documents.

~~Stage 4: Update~~

~~{~~

~~\_id: 1, // Keep the \_id field for the update operation~~

~~new\_price: 1 // Keep the new price to apply it in the update~~

~~}~~

~~Then go filter.~~

~~Filter:{ name: "Be Happy in Porto" }~~

~~Click update.~~

~~{~~

~~$set: { price: newPrice } //if the new price is 27, enter 27~~

~~}~~

Don’t do update. Otherwise you have to manually insert the listing id and new price value

*EXPLANATION*

*Stage 2: Here we look at other properties that are similar to the one in stage 1. We however add on to that by calulcating the average price.*

* *Let are the matching conditions we want to pass onto pipeline.*
* *In the pipeline, operations are performed on the listingsAndReviews collection. We match the attributes from $property\_type( outside of pipeline)* to attributes inside pipeline, $$*property\_type.*
* *Expressions is to allow the usage of $eq to compare the fields.*
* *We then group by calculating the average price of filtered properties.*
* *The result is stored as avgPriceData.*

*Stage 3:* A ***new attribute*** *called new\_price is created, which is calculated as 90% of the average price of similar properties.At this point, the price attribute is untouched.*

Stage 4: During the merge operation, the original price is **updated** to the value of new\_price, effectively setting the price to 10% lower than the average daily rate.

Shell

db.listingsAndReviews.aggregate([

// Stage 1: Match the specific listing

{

$match: {

name: "Be Happy in Porto"

}

},

// Stage 2: Lookup to find similar properties and calculate the average price

{

$lookup: {

from: "listingsAndReviews",

let: {

property\_type: "$property\_type",

bedrooms: "$bedrooms",

market: "$address.market"

},

pipeline: [

{

$match: {

$expr: {

$and: [

{ $eq: ["$property\_type", "$$property\_type"] },

{ $eq: ["$bedrooms", "$$bedrooms"] },

{ $eq: ["$address.market", "$$market"] },

{ $ne: ["$price", null] } // Ensure price is not null

]

}

}

},

{

$group: {

\_id: null,

avg\_price: { $avg: "$price" } // Calculate the average price

}

}

],

as: "avgPriceData" // Output array

}

},

// Stage 3: Set new price based on average

{

$set: {

new\_price: {

$round: [

{

$multiply: [

{ $arrayElemAt: ["$avgPriceData.avg\_price", 0] }, // Get average price

0.9 // Set new price to 10% lower

]

},

2 // Round to 2 decimal places

]

}

}

},

// Stage 4: Merge to update the original listing's price

{

$merge: {

into: { db: "sample\_airbnb", coll: "listingsAndReviews" },

on: "\_id",

whenMatched: [

{

$set: { price: "$new\_price" // Update the price

}

}

],

whenNotMatched: "discard" // Discard if no match found

}

}

]);

## Statement 5 (Checked)

Compass

First search for the review.

{

"reviews.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."

}

Next update the comment with the specified comment.

{

"$pull": {

"reviews": {

"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."

}

}

}

NOTE: The $pull operator removes the entire review object where the comment matches the given string.

Shell

db.listingsAndReviews.updateOne(

{ "reviews.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." },

{

$pull: {

reviews: {

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."

}

}

}

);

*TO CHECK, use this command: {*

*"reviews.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."*

*}*

# Task 2: Extend the AirBnB database

## Task 2.1 Embedded Approach

How I accommodate extending the database in a embedded approach is by first looking at the sample data. I can see that it provides a structure for a what booking will contain. I then query into the relevant documents, { name: "Be Happy in Porto" } and { name: "Be Happy in the Heart Of Montreal DT 1MIN to Metro" }. Only 2 documents are needed as bookings 1 and 2 share 1 document. No additional collections are needed and I will only need to use listingsAndReviews for an embedded approach. This means that booking information is contained in each document. I just need to insert a new object, “bookings” which will contain all the sample data.

## Task 2.2 Embedded Approach

*Add the first booking in the sample bookings file to your extended database.*

I just use the sample data but enclose it in shell by using updateOne and including the relevant filter for the booking.

**Bookings 1**

db.listingsAndReviews.updateOne(

{ name: "Be Happy in Porto" }, // Filter by listing name

{

$push: { // Add the new booking to the bookings array

bookings: {

booking\_id: 12345,

arrival\_date: ISODate("2025-03-11T14:00:00.000+10:00"),

departure\_date: ISODate("2025-03-13T10:00:00.000+10:00"),

client: {

name: "Brodie Mackrell",

email: "BrodieMackrell@rhyta.com",

daytime\_phone: "(03) 5352 6218",

mobile: "0453 526 218",

postal\_address: "48 Mills Street DALYSTON VIC 3992",

home\_address: "48 Mills Street DALYSTON VIC 3992"

},

deposit\_paid: 10,

balance\_due: 20,

balance\_due\_date: ISODate("2025-03-01T00:00:00.000+10:00"),

number\_of\_guests: 2,

guests: [

{ name: "Brodie", age: 44 },

{ name: "John", age: 44 }

]

}

}

}

);

**Booking 2**

db.listingsAndReviews.updateOne(

{ name: "Be Happy in Porto" }, // Same listing name for second booking

{

$push: { // Add the second booking to the bookings array

bookings: {

booking\_id: 12347,

arrival\_date: ISODate("2025-11-11T14:00:00.000+11:00"),

departure\_date: ISODate("2025-11-14T10:00:00.000+11:00"),

client: {

name: "Alexander Kevin",

email: "AlexanderKevin@teleworm.us",

daytime\_phone: "(08) 9038 8742",

mobile: "0490 388 742",

postal\_address: "81 Farrar Parade CARNAMAH WA 6517",

home\_address: "81 Farrar Parade CARNAMAH WA 6517"

},

deposit\_paid: 30,

balance\_due: 0,

balance\_due\_date: null,

number\_of\_guests: 3,

guests: [

{ name: "Alexander", age: 54 },

{ name: "Alexa", age: 50 },

{ name: "Siri", age: 15 }

]

}

}

}

);

Booking 3

db.listingsAndReviews.updateOne(

{ name: "Be Happy in the Heart Of Montreal DT 1MIN to Metro" }, // Filter by different listing name

{

$push: { // Add the booking to the bookings array

bookings: {

booking\_id: 12346,

arrival\_date: ISODate("2025-04-11T14:00:00.000+10:00"),

departure\_date: ISODate("2025-04-12T10:00:00.000+10:00"),

client: {

name: "Brodie Mackrell",

email: "BrodieMackrell@rhyta.com",

daytime\_phone: "(03) 5352 6218",

mobile: "0453 526 218",

postal\_address: "48 Mills Street DALYSTON VIC 3992",

home\_address: "48 Mills Street DALYSTON VIC 3992"

},

deposit\_paid: 20,

balance\_due: 59,

balance\_due\_date: ISODate("2025-04-01T00:00:00.000+10:00"),

number\_of\_guests: 3,

guests: [

{ name: "Brodie", age: 44 },

{ name: "John", age: 44 },

{ name: "John Jr", age: 10 }

]

}

}

}

);

**Remove bookings for referenced approach**

//Booking 1

db.listingsAndReviews.updateOne(

{ name: "Be Happy in Porto" }, // Filter by listing name

{

$pull: { // Remove the booking with the specified booking\_id

bookings: { booking\_id: 12345 }

}

}

);

//Booking 2

db.listingsAndReviews.updateOne(

{ name: "Be Happy in Porto" }, // Same listing name for the second booking

{

$pull: { // Remove the booking with the specified booking\_id

bookings: { booking\_id: 12347 }

}

}

);

**REFRESH then delete the bookings field**

db.listingsAndReviews.updateOne(

{ name: "Be Happy in Porto", bookings: { $size: 0 } }, // Filter by listing name and check if bookings is empty

{

$unset: { bookings: "" } // Remove the bookings field if it's empty

}

);

//Booking 3

db.listingsAndReviews.updateOne(

{ name: "Be Happy in the Heart Of Montreal DT 1MIN to Metro" }, // Filter by the listing name

{

$pull: { // Remove the booking with the specified booking\_id

bookings: { booking\_id: 12346 }

}

}

);

**Refresh then unset**

db.listingsAndReviews.updateOne(

{ name: "Be Happy in the Heart Of Montreal DT 1MIN to Metro", bookings: { $size: 0 } }, // Filter by listing name and check if bookings is empty

{

$unset: { bookings: "" } // Remove the bookings field if it's empty

}

);

*Display the client name, email address, booking start date, and booking end date for all bookings made so far for the listing named “Be Happy in Porto”.*

Query: { name: "Be Happy in Porto"}

Project: {"bookings.client.name":1,"bookings.client.email":1,"bookings.arrival\_date":1,"bookings.departture\_date":1}

## Task 2.1: Referenced Approach

I would create the collection, Client. This is to store information documents about the clients. This is because a client can be involved in many bookings. In this case it is Brodie, as he is involved in booking 1 to 3. We create this document to avoid redundancy by normalising the fields similar to relational model.

* In our case we are only focusing on 3 documents which will not affect performance.

I would create new collection in for Bookings. This is to store the 3 bookings found in the sample data. This would be called upon by using the just inserting the documents booking id into the documents from listingAndReviews. This is to avoid redundancy by normalising the fields similar to relational model.

* With embedding, querying can be easier.
* We use embedding as many bookings(child documents) can correspond to one property (parent documents).
* While a referenced approach can lead to redundancy, and performance loss, we are only focusing on 3 documents so issues would be minimal.

## Task 2.2: Referenced approach

*Add the first booking in the sample bookings file to your extended database.*

Only after creating the Client documents and getting their ID’s.

/\*\*

This is booking 1

\*\*/

{

"\_id": {

"$oid": "670238f3563825de0da55f89"

},

"booking\_id": "12345",

"arrival\_date": {

"$date": "2025-03-11T04:00:00.000Z"

},

"departure\_date": {

"$date": "2025-03-13T00:00:00.000Z"

},

"listing\_id": "10083468",

"client\_id": {

"$oid": "670232f5563825de0da55f81"

},

"deposit\_paid": 10,

"balance\_due": 20,

"balance\_due\_date": {

"$date": "2025-02-28T14:00:00.000Z"

},

"number\_of\_guests": 2,

"guests": [

{

"name": "Brodie",

"age": "44"

},

{

"name": "John",

"age": "44"

}

]

}

// Insert Booking 2

{

"\_id": {

"$oid": "67023d00563825de0da55f8c"

},

"booking\_id": 12347,

"arrival\_date": {

"$date": "2025-11-11T03:00:00.000Z"

},

"departure\_date": {

"$date": "2025-11-13T23:00:00.000Z"

},

"listing\_id": "10083468",

"client\_id": {

"$oid": "67023660563825de0da55f83"

},

"deposit\_paid": 30,

"balance\_due": 0,

"balance\_due\_date": "",

"number\_of\_guests": 3,

"guests": [

{

"name": "Alexander",

"age": "54"

},

{

"name": "Alexa",

"age": "50"

},

{

"name": "Siri",

"age": "15"

}

]

}

/\*\*

This is booking document 3

\*\*/

{

"\_id": {

"$oid": "67023f6f563825de0da55f90"

},

"booking\_id": 12346,

"arrival\_date": {

"$date": "2025-04-11T04:00:00.000Z"

},

"departure\_date": {

"$date": "2025-04-12T00:00:00.000Z"

},

"listing\_id": "25937713",

"client\_id": {

"$oid": "670232f5563825de0da55f81"

},

"deposit\_paid": 20,

"balance\_due": 59,

"balance\_due\_date": {

"$date": "2025-03-31T14:00:00.000Z"

},

"number\_of\_guests": 3,

"guests": [

{

"name": "Brodie",

"age": "44"

},

{

"name": "John",

"age": "44"

},

{

"name": "John Jr",

"age": "10"

}

]

}

*Display the client name, email address, booking start date, and booking end date for all bookings made so far for the listing named “Be Happy in Porto”.*

**Stage 1: Match**

{ name: "Be Happy in Porto" }

**Stage 2: lookup**

{

from: "bookings",

localField: "\_id", // The \_id field in the listingsAndReviews collection

foreignField: "listing\_id", // The listing\_id field in the bookings collection

as: "booking\_info" // Output array containing the matching booking documents

}

**Stage 3: lookup**

{

from: "clients",

localField: "booking\_info.client\_id", // The client\_id field in bookings

foreignField: "\_id", // The \_id field in clients

as: "client\_info" // Output array with matching clients

}

**Stage 4: project**

{

\_id: 0, // Exclude \_id from the output

"client\_info.name": 1, // Include client name

"client\_info.email": 1, // Include client email

"booking\_info.arrival\_date": 1, // Include booking arrival date

"booking\_info.departure\_date": 1 // Include booking departure date

}

# References

GeeksforGeeks (2024) *MongoDB Cheat Sheet,* GeeksforGeeks, accessed 6 October 2024. <https://www.geeksforgeeks.org/mongodb-cheat-sheet/>

MongoDB (2024) *db.collection.insert(),* accessed 6 October 2024. <https://www.mongodb.com/docs/manual/reference/method/db.collection.insert/>

w3schools (2024) *MongoDB mongosh Update,* accessed 6 October 2024. <https://www.w3schools.com/mongodb/mongodb_mongosh_update.php>

MongoDB (2024) *$addToSet,* MongoDB, accessed 6 October 2024. <https://www.mongodb.com/docs/manual/reference/operator/update/addToSet/>

1B

3T (2024) *The Beginner’s Guide to MongoDB Aggregation (With Exercise), 3T*, accessed 6 October 2024. <https://studio3t.com/knowledge-base/articles/mongodb-aggregation-framework/>

1C

MongoDB (2024) *$insert (aggregation),* MongoDB, accessed 6 October 2024. <https://www.mongodb.com/docs/manual/reference/method/db.collection.insert/>

1D

MongoDB (2024) *$avg (aggregation),* MongoDB, accessed 6 October 2024. [**https://www.mongodb.com/docs/manual/reference/operator/aggregation/avg/**](https://www.mongodb.com/docs/manual/reference/operator/aggregation/avg/)

1E

MongoDB (2024) *$unwind (aggregation),* MongoDB, accessed 6 October 2024. <https://www.mongodb.com/docs/manual/reference/operator/aggregation/unwind/?utm_source=compass&utm_medium=product>

MongoDB (2024) *$project (aggregation),* MongoDB, accessed 6 October 2024. <https://www.mongodb.com/docs/manual/reference/operator/aggregation/project/?utm_source=compass&utm_medium=product>

1F

MongoDB (2024) *$sort (aggregation),* MongoDB, accessed 6 October 2024. <https://www.mongodb.com/docs/manual/reference/operator/aggregation/sort/>