**Database Applications – A2**

# Task 1: CRUD tasks on listingsAndReviews Document Collection

**Statement 1**

**1A**

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**

Compass

{

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

}

Mongo Shell

db.**listingsAndReviews**.find({

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

});

**1C**

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 }

}

])

[**https://www.mongodb.com/docs/manual/reference/operator/aggregation/avg/**](https://www.mongodb.com/docs/manual/reference/operator/aggregation/avg/)

**1D**

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**

Mongo Shell

db.listingsAndReviews.aggregate([

// Stage 1: Match

{

$match: {

"address.market": "Barcelona",

bedrooms: 3

}

},

// Stage 2: Group

{

$group: {

\_id: null,

avgPrice: { $avg: "$price" }

}

},

// Stage 3: AddFields

{

$addFields: {

lowerBound: { $subtract: ["$avgPrice", 5] },

upperBound: { $add: ["$avgPrice", 5] }

}

},

// Stage 4: Lookup

{

$lookup: {

from: "listingsAndReviews",

let: { lower: "$lowerBound", upper: "$upperBound" },

pipeline: [

{

$match: {

$expr: {

$and: [

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

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

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

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

]

}

}

}

],

as: "listingsInRange"

}

},

// Stage 5: Project

{

$project: {

\_id: 0,

averagePrice: "$avgPrice",

listingsInRange: 1

}

}

]);

Mongo Compass

Go to the aggregations tab. Make 5 stages.

**Stage 1 Match**

{ "address.market": "Barcelona", bedrooms: 3 }

***Stage 2 group***

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

***Stage 3 addFields***

{

lowerBound: { $subtract: ["$avgPrice", 5] },

upperBound: {$add: ["$avgPrice", 5] }

}

***Stage 4 lookup***

{

from: "listingsAndReviews",

let: { lower: "$lowerBound", upper: "$upperBound" },

pipeline: [

{

$match: {

$expr: {

$and: [

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

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

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

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

]

}

}

}

],

as: "listingsInRange"

}

***Stage 5 project***

{ \_id: 0, averagePrice: "$avgPrice", listingsInRange: 1}

**1F**

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: "Portugal" // 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": "Portugal"

}

}

}

**Statement 4**

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: {

propertyType: "$property\_type",

bedrooms: "$bedrooms",

market: "$address.market"

},

pipeline: [

{

$match: {

$expr: {

$and: [

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

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

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

]

}

}

},

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

],

as: "similarProperties"

}

Stage 3: Make a set

{ averageDailyRate: {$arrayElemAt: [ "$similarProperties.averageDailyRate", 0 ] } }

Stage 4: Make another set

{ newDailyRate: {

$round: [

{

$multiply: [

{ $subtract: [ "$averageDailyRate", { $multiply: ["$averageDailyRate",0.1] } ] },100] }, 2

]

} }

Stage 5: Merge the collections

{

into: "listingsAndReviews",

whenMatched: "merge",

whenNotMatched: "discard"

}

Mongo Shell Commands

db.listingsAndReviews.aggregate([

// Stage 1: Match the listing "Be Happy in Porto"

{

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

},

// Stage 2: Lookup similar properties in the same market with same property type and bedrooms

{

$lookup: {

from: "listingsAndReviews",

let: {

propertyType: "$property\_type",

bedrooms: "$bedrooms",

market: "$address.market"

},

pipeline: [

{

$match: {

$expr: {

$and: [

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

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

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

]

}

}

},

{

$group: {

\_id: null,

averageDailyRate: { $avg: "$price" } // Assuming price is Decimal128

}

}

],

as: "similarProperties"

}

},

// Stage 3: Set the average daily rate for "Be Happy in Porto"

{

$set: {

averageDailyRate: { $arrayElemAt: [ "$similarProperties.averageDailyRate", 0 ] }

}

},

// Stage 4: Calculate the new daily rate (10% lower than the average)

{

$set: {

newDailyRate: {

$round: [

{

$multiply: [

{ $subtract: [ "$averageDailyRate", { $multiply: [ "$averageDailyRate", 0.1 ] } ] },

100

]

},

2

]

}

}

},

// Stage 5: Merge/Update the listings collection with the new rate

{

$merge: {

into: "listingsAndReviews",

whenMatched: "merge",

whenNotMatched: "discard",

on: "\_id", // Match by \_id to update the correct document

update: {

$set: { price: "$newDailyRate" }

}

}

}

])

**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

References

[**https://www.mongodb.com/docs/manual/reference/operator/aggregation/avg/**](https://www.mongodb.com/docs/manual/reference/operator/aggregation/avg/)

<https://www.mongodb.com/docs/manual/reference/method/db.collection.insert/>

<https://www.w3schools.com/mongodb/mongodb_mongosh_update.php>

<https://www.mongodb.com/docs/manual/reference/operator/update/addToSet/>

1B

<https://studio3t.com/knowledge-base/articles/mongodb-aggregation-framework/>

1C

<https://www.mongodb.com/docs/manual/reference/method/db.collection.insert/>

1E

<https://www.mongodb.com/docs/manual/reference/operator/aggregation/unwind/?utm_source=compass&utm_medium=product>

<https://www.mongodb.com/docs/manual/reference/operator/aggregation/project/?utm_source=compass&utm_medium=product>

1F

<https://www.mongodb.com/docs/manual/reference/operator/aggregation/sort/>