7 a. Find all listings with listing url, name, address, host picture url in the listings And Reviews

Sample Document Structure

Here's an example structure of a document in the `listingsAndReviews` collection:

```
> db.listingsAndReviews.insertMany( [ {
"listing_id": "10006546",
"listing_url": "https://www.example.com/rooms/10006546",
"name": "Beautiful Beach House",
"address": {
  "street": "123 Beach St",
  "suburb": "Santa Monica",
  "state": "CA",
  "country": "United States"
  },
"host": {
  "host_id": "12345678",
  "host_picture_url": "https://www.example.com/host12345678.jpg"
},}]);
```

Query

To find all listings with the required fields, you can use the following query:

```
> db.listingsAndReviews.find( {  },
{  _id: 0,
  listing_url: 1,
  name: 1,
  "address.street": 1,
```

"address.suburb": 1,

"address.state": 1,

"address.country": 1,

"host.host_picture_url": 1

})

Example Output:


```
  {
    "listing_url": "https://www.example.com/rooms/10006546",
    "name": "Beautiful Beach House",
    "address": {
      "street": "123 Beach St",
      "suburb": "Santa Monica",
      "state": "CA",
      "country": "United States"
    },
    "host": {
      "host_picture_url": "https://www.example.com/host12345678.jpg"
    }
  },
  // other listings...
]
```


7.b). Using E-commerce collection write a query to display reviews summary

Here's an example of a document in the **`ecommerce` collection**:

> db.ecommerce.insertMany([{

"product_id": "1001",

"name": "Smartphone",

"category": "Electronics",

"price": 299.99,

"reviews":

[{

"reviewer_name": "Alice",

"review_date": "2024-01-10",

```
    "rating": 5,
    "comment": "Excellent phone with great battery life!"
  },
  {
    "reviewer_name": "Bob",
    "review_date": "2024-02-12",
    "rating": 4,
    "comment": "Good value for money."
  } ] } ] )
```

Query

```
db.ecommerce.aggregate([
 {  $unwind: "$reviews"                    // Unwind the reviews array to work with individual review documents
  },
 {
   $group: {
    _id: "$product_id",
    productName: { $first: "$name" },
    averageRating: { $avg: "$reviews.rating" },
    totalReviews: { $sum: 1 }
   }
 },
 {   $project: {
    _id: 0,
    product_id: "$_id",
    productName: 1,
```

```
    averageRating: 1,
    totalReviews: 1
  } } ] )
```

Example output:

```
[ {
  "product_id": "1001",
  "productName": "Smartphone",
  "averageRating": 4.5,
  "totalReviews": 2
 } ]
```

8)a. Demonstrate creation of different types of indexes on collection (unique, sparse, compound and multikey indexes)

Let's assume we have a **collection named `products`** with the following sample data:

```
db.products.insertMany([
 { product_id: 1, name: "Laptop", category: "Electronics", price: 899.99, tags: ["computer", "portable"] },
 { product_id: 2, name: "Smartphone", category: "Electronics", price: 599.99, tags: ["phone", "portable"] },
 { product_id: 3, name: "Coffee Maker", category: "Home Appliances", price: 49.99, tags: ["kitchen", "coffee"] },
 { product_id: 4, name: "Blender", category: "Home Appliances", price: 29.99, tags: ["kitchen", "blender"] }
])
```

Creating Different Types of Indexes

1. Unique Index

A unique index ensures that the indexed field does not contain duplicate values

// Create a unique index on the product_id field

> db.products.createIndex({ product_id: 1 }, { unique: true })

2. Sparse Index

// Create a sparse index on the price field

> db.products.createIndex({ price: 1 }, { sparse: true })

3. Compound Index

// Create a compound index on the category and price fields

> db.products.createIndex({ category: 1, price: -1 })

4. Multikey Index

// Create a multikey index on the tags field

> db.products.createIndex({ tags: 1 })

4. Multikey Index

// Create a multikey index on the tags field

> db.products.createIndex({ tags: 1 })

Verifying Indexes

// Get the list of indexes on the products collection

> db.products.getIndexes()

8) b. Demonstrate optimization of queries using indexes.

Sample Data

Make sure the '**`products` collection'** contains the sample data:

> db.products.insertMany([

{ product_id: 1, name: "Laptop", category: "Electronics", price: 899.99, tags: ["computer", "portable"] },

{ product_id: 2, name: "Smartphone", category: "Electronics", price: 599.99, tags: ["phone", "portable"] },

{ product_id: 3, name: "Coffee Maker", category: "Home Appliances", price: 49.99, tags: ["kitchen", "coffee"] },

{ product_id: 4, name: "Blender", category: "Home Appliances", price: 29.99, tags: ["kitchen", "blender"] },

{ product_id: 5, name: "Tablet", category: "Electronics", price: 299.99, tags: ["tablet", "portable"] }])

Creating Indexes

// Create indexes

> db.products.createIndex({ category: 1 })

> db.products.createIndex({ price: 1 })

> db.products.createIndex({ tags: 1 })

> db.products.createIndex({ name: "text" })

Query Performance Comparison

Query 1: Find products by category

Without Index:

// Drop the index if exists

> db.products.dropIndex({ category: 1 })

Output:

// Explain the query

> db.products.find({ category: "Electronics" }).explain("executionStats")

With Index:

// Create the index again

> db.products.createIndex({ category: 1 })

// Explain the query

> db.products.find({ category: "Electronics" }).explain("executionStats")

Query 2: Find products within a price range

Without Index:

```
// Drop the index if exists
> db.products.dropIndex({ price: 1 })
Output:
{nIndexwas: 5,ok:1}

// Explain the query
> db.products.find({ price: { $lt: 300 }
}).explain("executionStats")
```

With Index:

```
// Create the index again
> db.products.createIndex({ price: 1 })

// Explain the query
> db.products.find({ price: { $lt: 300 }
}).explain("executionStats")
```

9 a. Develop a query to demonstrate Text search using catalog data collection for a given word

Here's an example of a document in the **`catalogData`** collection:

> db.catalogdata.insertOne([

{

"_id": ObjectId("60b47193d242f45dd5cfa6a7"),

"product_id": 1,

"name": "Laptop",

"description": "Powerful laptop for gaming and productivity tasks.",

"category": "Electronics"

}])

Text Search Query

> db.catalogData.find({ $text: { $search: "catagory" } })

Example Output

[{

"_id": ObjectId("60b47193d242f45dd5cfa6a7"),

"product_id": 1,

"name": "Laptop",

"description": "Powerful laptop for gaming and productivity tasks.",

"category": "Electronics"

},

{

"_id": ObjectId("60b47193d242f45dd5cfa6b2"),

"product_id": 10,

"name": "Gaming Mouse",

"description": "High-performance gaming mouse with customizable buttons.",

"category": "Accessories"

},

// Other matching documents...

]

b. Develop queries to illustrate excluding documents with certain words and phrases

Example 1: Exclude Documents with a Specific Word

db.catalogData.find({ $text: { $search: "outdoor", $not: true } })

This query will return documents where the `description` field does not contain the word "outdoor".

Example 2: Exclude Documents with a Specific Phrase

db.catalogData.find({ $text: { $search: "\"heavy duty\"", $not: true } })

This query will return documents where the `description` field does not contain the phrase "heavy duty".

Example 3: Exclude Documents with Multiple Words

db.catalogData.find({ $text: { $search: "waterproof -shockproof -dustproof", $not: true } })

In this example, the – (minus sign) operator is used to exclude documents containing the words "shockproof" and "dustproof" from the search results.

10) Develop an aggregation pipeline to illustrate Text search on Catalog data collection.

QUERY:

```
> db.catalogdata.aggregate( [
 {
   $match: {
     $text: {
       $search: "laptop" // Specify the search term
     } }
 },
 {    $project: {
     _id: 0,
     product_id: 1,
     name: 1,
     description: 1,
     score: { $meta: "textScore" } // Include the text score for each document
   }  },
 {
   $sort: {
     score: { $meta: "textScore" } // Sort the documents by text score
   } } ])
```

Example Output:


```
[ {
   "product_id": 1,
   "name": "Powerful Laptop",
   "description": "This laptop is great for gaming and productivity tasks.",
   "score": 2.5
 },
 {
   "product_id": 2,
   "name": "Ultra-thin Laptop",
   "description": "An ultra-thin laptop ideal for business users.",
   "score": 1.8
 }, ]
```