LAB PROGRAMS

PROGRAM 1

1a. Using MongoDB, create a collection called transactions in database usermanaged (drop if it already exists) and bulk load the data from a json file, transactions.json

b. Upsert the record from the new file called transactions_upsert.json in Mongodb.

DESCRITPION

"Upsert" is a database operation in MongoDB that combines "update" and "insert" operations. The term "upsert" itself is a portmanteau of "update" and "insert".

When you perform an upsert operation in MongoDB, it attempts to update a document based on a specified query criteria. If the query criteria match an existing document, MongoDB will update that document with the specified update operations. However, if there is no document that matches the query criteria, MongoDB will insert a new document with the specified fields and values.

In summary:

- If the document exists, MongoDB performs an update operation.
- If the document does not exist, MongoDB performs an insert operation.
- Upsert operations are particularly useful when you want to update a document if it exists or insert it if it doesn't exist, without needing to explicitly check for the existence of the document beforehand. This can simplify your code and make it more efficient.

Step 1: Launch MongoDB shell and switch to the desired database

Mongo or mongosh

Step 2: Switch to the usermanaged database and drop the transactions collection if it exists

use usermanaged

db.transactions.drop()

Create a transaction.json

```
[
  {
    "transaction_id": 1,
    "user_id": 1001,
    "amount": 50.25,
    "timestamp": "2024-04-01T08:30:00Z",
    "description": "Purchase of groceries"
  },
  {
    "transaction_id": 2,
    "user_id": 1002,
    "amount": 20.75,
    "timestamp": "2024-04-01T12:15:00Z",
    "description": "Payment for utilities"
  },
  {
    "transaction_id": 3,
    "user_id": 1001,
    "amount": 100.0,
    "timestamp": "2024-04-02T10:00:00Z",
    "description": "Salary deposit"
  }
]
```

Step 3: Bulk load data from transactions.json into the transactions collection (Open new CMD Terminal and set the path to transactions.json)

mongoimport.exe --collection=transactions --db=usermanaged --type=json --jsonArray transactions.json

usermanaged> **db.transactions.find()**

OUTPUT:

```
I:\MERN\1pgm>mongoimport.exe --collection=transactions --db=usermanaged --type=json --jsonArray transactions.json
2024-05-14T09:43:38.916+0530    connected to: mongodb://localhost/
2024-05-14T09:43:38.970+0530    3 document(s) imported successfully. 0 document(s) failed to import.
```

```
usermanaged> db.transactions.find()
[
  {
    _id: ObjectId('660cf3cea61cf8591acd546a'),
    transaction_id: 2,
    user_id: 1002,
    amount: 20.75,
    timestamp: '2024-04-01T12:15:00Z',
    description: 'Payment for utilities'
  },
  {
    _id: ObjectId('660cf3cea61cf8591acd546b'),
    transaction_id: 3,
    user_id: 1001,
    amount: 100,
    timestamp: '2024-04-02T10:00:00Z',
    description: 'Salary deposit'
  },
  {
    _id: ObjectId('660cf3cea61cf8591acd546c'),
    transaction_id: 1,
    user_id: 1001,
    amount: 50.25,
    timestamp: '2024-04-01T08:30:00Z',
    description: 'Purchase of groceries'
  }
]
```

Create transactions_upsert.json

```
[
  {
    "transaction_id": 1,
    "user_id": 1001,
    "amount": 50.25,
    "timestamp": "2024-04-03T08:30:00Z",
    "description": "Purchase of electronics"
  },
  {
    "transaction_id": 2,
    "user_id": 1002,
    "amount": 75.50,
    "timestamp": "2024-04-03T12:15:00Z",
    "description": "Payment for rent"
  },
  {
    "transaction_id": 5,
    "user_id": 1010,
    "amount": 1200.0,
    "timestamp": "2024-04-04T10:00:00Z",
    "description": "Investment in stocks"
  }
]
```

Step 4: Upsert records from transactions_upsert.json into the transactions collection

mongoimport --db usermanaged --collection transactions --file transactions_upsert.json --jsonArray --upsertFields transaction_id

OUTPUT:

```
usermanaged> db.transactions.find()
[
  {
    _id: ObjectId('660cf3cea61cf8591acd546a'),
    transaction_id: 2,
    user_id: 1002,
    amount: 75.5,
    timestamp: '2024-04-03T12:15:00Z',
    description: 'Payment for rent'
  },
  {
    _id: ObjectId('660cf3cea61cf8591acd546b'),
    transaction_id: 3,
    user_id: 1001,
    amount: 100,
    timestamp: '2024-04-02T10:00:00Z',
    description: 'Salary deposit'
  },
  {
    _id: ObjectId('660cf3cea61cf8591acd546c'),
    transaction_id: 1,
    user_id: 1001,
    amount: 50.25,
    timestamp: '2024-04-03T08:30:00Z',
    description: 'Purchase of electronics'
  },
  {
    _id: ObjectId('660cf71ec41b1c8ba318da86'),
    transaction_id: 5,
    user_id: 1010,
    amount: 1200,
    timestamp: '2024-04-04T10:00:00Z',
    description: 'Investment in stocks'
  }
]
usermanaged>
```

PROGRAM 2

2**.** Query MongoDB with Conditions: [Create appropriate collection with necessary documents to answer the query]

a. Find any record where Name is Somu

b. Find any record where total payment amount (Payment.Total) is 600.

c. Find any record where price (Transaction.price) is between 300 to 500.

d. Calculate the total transaction amount by adding up Payment.Total in all records.

DESCRIPTION

In a typical business scenario, **"transaction price"** and **"payment total"** are related but represent different aspects of a transaction:

1. **Transaction Price:** This refers to the price or cost associated with a particular transaction. It's the amount of money that is being charged or exchanged for a product or service. For example, in retail setting, if you buy a product for Rs.50, then Rs.50 would be the transaction price.

2. **Payment Total:** This represents the total amount of money paid for a transaction. It includes not only the transaction price but also any additional charges, taxes, or discounts that might be applicable to the transaction. For instance, if you buy a product for Rs.50 but also pay Rs.5 in taxes, then the payment total would be Rs.55.

```
//Create a collection named “transaction”
db.createCollection("transactions")
```

```
// Insert sample documents into the transactions collection

db.transactions.insertMany([
  {
   "_id": 1,
   "Name": "Somu",
   "Payment": { "Total": 500 }
  },
  {
   "_id": 2,
   "Name": "Ravi",
   "Payment": { "Total": 600 }
  },
```

```
  {
   "_id": 3,
   "Name": "Somu",
   "Payment": { "Total": 700 }
  },
  {
   "_id": 4,
   "Name": "John",
   "Payment": { "Total": 400 }
  },
  {
   "_id": 5,
   "Name": "David",
   "Payment": { "Total": 800 }
  },
  {
   "_id": 6,
   "Name": "Somu",
   "Payment": { "Total": 600 }
  },
  {
   "_id": 7,
   "Name": "Alex",
   "Payment": { "Total": 450 }
  },
  {
   "_id": 8,
   "Name": "Chris",
   "Transaction": { "price": 350 }
  },
  {
   "_id": 9,
   "Name": "Emma",
   "Transaction": { "price": 400 }
  },
  {
   "_id": 10,
   "Name": "Sophia",
   "Transaction": { "price": 500 }
  },
  {
   "_id": 11,
   "Name": "Olivia",
   "Transaction": { "price": 600 }
  }
]);
```

a. **db.transactions.find({ "Name": "Somu" });**

OUTPUT:

```
pg2> db.transactions.find({ "Name": "Somu" });
[
  { _id: 1, Name: 'Somu', Payment: { Total: 500 } },
  { _id: 3, Name: 'Somu', Payment: { Total: 700 } },
  { _id: 6, Name: 'Somu', Payment: { Total: 600 } }
]
pg2>
```

b. **db.transactions.find({ "Payment.Total": 600 });**

OUTPUT:

```
pg2> db.transactions.find({ "Payment.Total": 600 });
[
  { _id: 2, Name: 'Ravi', Payment: { Total: 600 } },
  { _id: 6, Name: 'Somu', Payment: { Total: 600 } }
]
pg2>
```

c. **db.transactions.find({ "Transaction.price": { $gte: 300, $lte: 500 } });**

OUTPUT:

```
pg2> db.transactions.find({ "Transaction.price": { $gte: 300, $lte: 500 } });
[
  { _id: 8, Name: 'Chris', Transaction: { price: 350 } },
  { _id: 9, Name: 'Emma', Transaction: { price: 400 } },
  { _id: 10, Name: 'Sophia', Transaction: { price: 500 } }
]
pg2>
```

d. db.transactions.aggregate([
{
$group: {
_id: null,
totalAmount: { $sum: "$Payment.Total" }
}
}
]);

OUTPUT:

```
pg2> db.transactions.aggregate([
...   {
...     $group: {
...       _id: null,
...       totalAmount: { $sum: "$Payment.Total" }
...     }
...   }
... ]);
[ { _id: null, totalAmount: 4050 } ]
pg2>
```

- **db.transactions.find():** This is a MongoDB query to retrieve documents from the transactions collection.
- **{ "Transaction.price": { $gte: 300, $lte: 500 } }:** This is the query filter criteria. It specifies the condition that documents should meet to be included in the result.
- **"Transaction.price":** This is the field to filter on. It specifies that we're interested in the price field nested within the Transaction subdocument.
- **{ $gte: 300, $lte: 500 }:** This is a range condition using MongoDB query operators.
- **$gte:** Stands for "greater than or equal to". It specifies that the price field must be greater than or equal to 300.
- **$lte:** Stands for "less than or equal to". It specifies that the price field must be less than or equal to 500.
- Combining these, the filter condition means that we want to find documents where the price field within the Transaction subdocument falls within the range from 300 to 500 (inclusive).