

PROJECT 2 (MONGODB)



Done by

VÍCTOR ORTEGA NÍCOLAS

HÉCTOR RODRÍGUEZ LAMILLA

N/A's

100550953

100550972

TABLE OF CONTENTS

FRONTPAGE	1
DESIGN	2
CODE	3
RESULT EXCERPTS	4
INDEXES	5
EXTENSION	6
APPENDIX	7
AI-USE STATEMENT	8
CODE	9

DESIGN 2.1

Questions about the design

PART A

The main advantage of grouping multiple operations in a single document is that the collection will be more compact, it will present less documents. If there was a document for each one of the operations, there would be a lot of repeated redundant information on every document (such as account id, for example)

However, there is also an important drawback with this method of storing the operations. Having all the operations stored in one document with an array makes it harder (or less comfortable) to work with specific operations. With the operations separated in different documents, it would be much easier.

PART B

Keeping the accounts in a separate collection helps to keep the customer's documents smaller and easier to manage. It also allows to work with accounts independently, without modifying the customer data.

The main disadvantage is that when accounts are stored separately, we sometimes need to combine information from customers and accounts. This makes some queries a bit more complex compared to having everything inside the same document.

PART C

The proposed index is `{"transactions.category": 1}`. Many tasks in the project require grouping or filtering operations by category. Therefore, creating an index on the category field makes these queries faster, as MongoDB won't have to go through all the documents in order to find the operations that match.

CODE 2.2

Insert and update data

A. INSERT A NEW COSTUMER

* `db.accounts.findOne({account_id: 999876});`

`db.accounts.insertOne({account_id: 999876, limit: 10000, products : ["InvestmentStock", "Commodity"]});`

`db.customers.insertOne({username: "rafanadal", name: "Rafael Nadal", address: "Calle de la Palmera, 12", birthdate: ISODate("2003-01-01T00:00:00Z"), email: "rafanadal10@gmail.com", active: true, accounts: [999876], tier_and_details: {}});`

```
> db.accounts.insertOne({account_id: 999876, limit: 10000, products : ["InvestmentStock", "Commodity"]});
< {
  acknowledged: true,
  insertedId: ObjectId('6929cafc96989d79d80d0a1b')
}

> db.customers.insertOne({username: "rafanadal", name: "Rafael Nadal", address: "Calle de la Palmera, 12", birthdate: ISODate("2003-01-01T00:00:00Z"), email: "rafanadal10@gmail.com", active: true, accounts: [999876], tier_and_details: {}});
< {
  acknowledged: true,
  insertedId: ObjectId('6929cb3196989d79d80d0a1c')
}
```

B. UPDATE: ADD PRODUCT TO AN ACCOUNT

* `db.accounts.updateOne({account_id: 999876}, {$push: {products: "Brokerage"}})`

```
> db.accounts.updateOne({account_id: 999876}, {$push: {products: "Brokerage"}});
< {
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

C. UPDATE: INSERT A NEW TRANSACTION

* `db.transactions.updateOne({account_id: 852986}, {$push: {transactions: {date: ISODate("2024-08-01T00:00:00Z"), amount: 100, transaction_code: "buy", symbol: "nflx", price: "20.60", total: "2060"}}});`

`db.transactions.findOne({account_id: 852986});`

```
> db.transactions.updateOne({account_id: 852986}, {$push: {transactions: {date: ISODate("2024-08-01T00:00:00Z"), amount: 100, transaction_code: "buy", symbol: "nflx", price: "20.60", total: "2060"}}});
< {
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

CODE 2.3

Queries for retrieving data

A. SHOW ALL ACCOUNTS OF A SPECIFIC CUSTOMER

* `db.customers.find({name: "Brian Flores"}, {accounts: 1, _id: 0});`

```
> db.customers.find({name: "Brian Flores"}, {accounts: 1, _id: 0});
< {
  accounts: [
    550665,
    321695
  ]
}
```

B. SECTION 2.3 (B) IS ON THE NEXT PAGE

C. SEARCH FOR CUSTOMERS BY PARTIAL NAME

* `db.customers.find({name: {$regex: "Br", $options: "i"}}, {_id: 0, name: 1, accounts: 1});`

```
> db.customers.find({name: {$regex: "Br", $options: "i"}}, {_id: 0, name: 1, accounts: 1});
< {
  name: 'Brad Cardenas',
  accounts: [
    721914,
    817222,
    973067,
    260799,
    87389
  ]
}
{
  name: 'Dr. Angela Brown',
  accounts: [
    571880
  ]
}
```

This continues with all the names that contain “Br”, but due to space we only included two

CODE 2.3

Queries for retrieving data

B. LIST THE LAST 10 TRANSACTION SORTED BY DATE DESCENDING (1º SOLUTION)

* `db.transactions.find({}, {_id: 0, account_id: 1, bucket_end_date: 1, transactions: 1}).sort({bucket_end_date: -1}).limit(10);`

```
< {
  account_id: 278866,
  bucket_end_date: 2017-01-09T00:00:00.000Z
}
{
  account_id: 599752,
  bucket_end_date: 2017-01-09T00:00:00.000Z
}
{
  account_id: 78388,
  bucket_end_date: 2017-01-09T00:00:00.000Z
}
{
  account_id: 996263,
  bucket_end_date: 2017-01-09T00:00:00.000Z
}
{
  account_id: 674364,
  bucket_end_date: 2017-01-09T00:00:00.000Z
}
```

```
{
  account_id: 503933,
  bucket_end_date: 2017-01-09T00:00:00.000Z
}
{
  account_id: 918103,
  bucket_end_date: 2017-01-09T00:00:00.000Z
}
{
  account_id: 673123,
  bucket_end_date: 2017-01-09T00:00:00.000Z
}
{
  account_id: 581161,
  bucket_end_date: 2017-01-09T00:00:00.000Z
}
{
  account_id: 987988,
  bucket_end_date: 2017-01-09T00:00:00.000Z
}
```

With this solution we get back the 10 latest transaction documents. Each document represents all the transactions made by an account in given a time period. The drawbacks of this solution are that we are not getting the last 10 transactions per se and it is also possible that one account has a later bucket_end_date than another account, while the other account still contains more recent transactions inside its bucket. To solve this problems we created a solution based on aggregation pipeline (see this solution on the next page)

CODE 2.3

Queries for retrieving data

B. LIST THE LAST 10 TRANSACTION SORTED BY DATE DESCENDING (2º SOLUTION)

* `db.transactions.aggregate([{$unwind: "$transactions"}, {$sort: {"transactions.date": -1}}, {$limit: 10}, {$project: {_id: 0, account_id: 1, date: "$transactions.date", amount: "$transactions.amount", transaction_code: "$transactions.transaction_code", symbol: "$transactions.symbol", total: "$transactions.total"}}]);`

```
{
  account_id: 852986,
  date: 2024-08-01T00:00:00.000Z,
  amount: 100,
  transaction_code: 'buy',
  symbol: 'nflx',
  total: '2060'
}
{
  account_id: 261248,
  date: 2017-01-09T00:00:00.000Z,
  amount: 8345,
  transaction_code: 'buy',
  symbol: 'fb',
  total: '1044809.243070348178790140992'
}
{
  account_id: 673123,
  date: 2017-01-09T00:00:00.000Z,
  amount: 2290,
  transaction_code: 'sell',
  symbol: 'fb',
  total: '284440.7188750756974116029596'
}
{
  account_id: 225602,
  date: 2017-01-09T00:00:00.000Z,
  amount: 4105,
  transaction_code: 'buy',
  symbol: 'goog',
  total: '3306993.985752043170691649721'
}
{
  account_id: 946196,
  date: 2017-01-09T00:00:00.000Z,
  amount: 1731,
  transaction_code: 'sell',
  symbol: 'goog',
  total: '1400511.570582843473061904696'
}
```

```
{
  account_id: 350036,
  date: 2017-01-09T00:00:00.000Z,
  amount: 2825,
  transaction_code: 'sell',
  symbol: 'goog',
  total: '2283007.717086967628006277664'
}
{
  account_id: 830030,
  date: 2017-01-09T00:00:00.000Z,
  amount: 2845,
  transaction_code: 'buy',
  symbol: 'team',
  total: '70359.00004840941581818469786'
}
{
  account_id: 156715,
  date: 2017-01-09T00:00:00.000Z,
  amount: 8941,
  transaction_code: 'buy',
  symbol: 'znga',
  total: '24034.02443241277721286408564'
}
{
  account_id: 776263,
  date: 2017-01-09T00:00:00.000Z,
  amount: 5097,
  transaction_code: 'sell',
  symbol: 'goog',
  total: '4093379.981822397120481582533'
}
{
  account_id: 645476,
  date: 2017-01-09T00:00:00.000Z,
  amount: 7827,
  transaction_code: 'buy',
  symbol: 'goog',
  total: '6321722.601813343698836433759'
}
```

With this solution, the pipeline flattens the transactions array inside each transaction's document and sorts them by their date, allowing us to retrieve exactly the last 10 transactions

CODE 2.4

Aggregations pipelines

A. TOP-5 MERCHANTS OR CATEGORIES BY TOTAL VOLUME (AMOUNT) GLOBALLY

```
* db.transactions.aggregate([
  { $unwind: "$transactions" },
  {
    $group: {
      _id: "$transactions.symbol",
      totalAmount: { $sum: "$transactions.amount" }
    }
  },
  { $sort: { totalAmount: -1 } },
  { $limit: 5 }
])
```

```
> db.transactions.aggregate([
  { $unwind: "$transactions" },
  {
    $group: {
      _id: "$transactions.symbol",
      totalAmount: { $sum: "$transactions.amount" }
    }
  },
  { $sort: { totalAmount: -1 } },
  { $limit: 5 }
])
{
  _id: 'adbe',
  totalAmount: 27463715
}
{
  _id: 'ebay',
  totalAmount: 27232371
}
{
  _id: 'crm',
  totalAmount: 27099929
}
{
  _id: 'goog',
  totalAmount: 27029894
}
{
  _id: 'nvda',
  totalAmount: 26108705
}
```

B. SHARED LIMIT: LIST THE ACCOUNTS THAT HAVE THE SAME LIMIT IN COMMON, COUNT THEM AND SHOW THEIR_IDS

```
* db.accounts.aggregate([
  {
    $group: {
      _id: "$limit",
      accounts: { $push: "$_id" },
      count: { $sum: 1 }
    }
  }
])
```

```
> db.accounts.aggregate([
  {
    $group: {
      _id: "$limit",
      accounts: { $push: "$_id" },
      count: { $sum: 1 }
    }
  }
])
{
  _id: 8000,
  accounts: [
    ObjectId('5ca4bbc7a2dd94ee581624d5'),
    ObjectId('5ca4bbc7a2dd94ee58162540'),
    ObjectId('5ca4bbc7a2dd94ee58162587'),
    ObjectId('5ca4bbc7a2dd94ee5816264b'),
    ObjectId('5ca4bbc7a2dd94ee5816288f'),
    ObjectId('5ca4bbc7a2dd94ee58162a31')
  ],
  count: 6
}
{
  _id: 10000,
  accounts: [
    ObjectId('5ca4bbc7a2dd94ee5816238d'),
    ObjectId('5ca4bbc7a2dd94ee5816238e'),
    ObjectId('5ca4bbc7a2dd94ee5816238f'),
    ObjectId('5ca4bbc7a2dd94ee58162390'),
    ObjectId('5ca4bbc7a2dd94ee58162391'),
    ObjectId('5ca4bbc7a2dd94ee58162393'),
    ObjectId('5ca4bbc7a2dd94ee58162394'),
    ObjectId('5ca4bbc7a2dd94ee58162395')
  ],
  count: 8
}
```

The output keeps going with the rest of the limits and their accounts, but the screenshot only shows a few rows because there is not enough space to display the full result

CODE 2.4

Aggregations pipelines

C. TRANSACTIONS ENRICHED WITH THE CUSTOMER NAME FOR EACH OPERATION (JOIN VIA \$LOOKUP)

```
* db.transactions.aggregate([
  { $unwind: "$transactions" },
  {
    $lookup: {
      from: "customers",
      localField: "account_id",
      foreignField: "accounts",
      as: "customer_info"
    }
  },
  { $unwind: "$customer_info" },
  {
    $project: {
      _id: 0,
      account_id: 1,
      date: "$transactions.date",
      amount: "$transactions.amount",
      customerName: "$customer_info.name"
    }
  }
])
```

The output keeps going with the rest of the transactions, but due to space we only included a few of them

```
< {
  account_id: 443178,
  date: 2003-09-09T00:00:00.000Z,
  amount: 7514,
  customerName: 'Michael Johnson'
}
{
  account_id: 443178,
  date: 2016-06-14T00:00:00.000Z,
  amount: 9240,
  customerName: 'Michael Johnson'
}
{
  account_id: 443178,
  date: 2002-12-04T00:00:00.000Z,
  amount: 2824,
  customerName: 'Michael Johnson'
}
{
  account_id: 443178,
  date: 2014-07-14T00:00:00.000Z,
  amount: 7418,
  customerName: 'Michael Johnson'
}
{
  account_id: 443178,
  date: 2011-10-28T00:00:00.000Z,
  amount: 5638,
  customerName: 'Michael Johnson'
}
{
  account_id: 443178,
  date: 2010-12-20T00:00:00.000Z,
  amount: 6774,
  customerName: 'Michael Johnson'
}
```

CODE 2.4

Aggregations pipelines

D. INCOME VS. EXPENSES PER CUSTOMER IN A QUARTER: SUM OF POSITIVE AMOUNTS (INFLOWS), NEGATIVE AMOUNTS (OUTFLOWS), AND TRANSACTION COUNT

```
* db.transactions.aggregate([
  { $unwind: "$transactions" },
  {
    $match: {
      "transactions.date": {
        $gte: ISODate("2017-01-01"),
        $lt: ISODate("2017-04-01")
      }
    }
  },
  {
    $lookup: {
      from: "customers",
      localField: "account_id",
      foreignField: "accounts",
      as: "customer_info"
    }
  },
  { $unwind: "$customer_info" },
  {
    $group: {
      _id: "$customer_info.name",
      totalIncome: {
        $sum: {
          $cond: [
            { $eq: ["$transactions.transaction_code", "sell"] },
            "$transactions.amount",
            0
          ]
        }
      },
      totalExpenses: {
        $sum: {
          $cond: [
            { $eq: ["$transactions.transaction_code", "buy"] },
            "$transactions.amount",
            0
          ]
        }
      },
      transactionCount: { $sum: 1 }
    }
  },
  { $sort: { _id: 1 } }
])
```

The output keeps going with the rest of the transactions in that quarter, but due to space we only included a few of them

```
< {
  _id: 'Adam Anderson',
  totalIncome: 0,
  totalExpenses: 4057,
  transactionCount: 1
}
{
  _id: 'Adam Miller',
  totalIncome: 0,
  totalExpenses: 683,
  transactionCount: 1
}
{
  _id: 'Alex Turner',
  totalIncome: 0,
  totalExpenses: 1128,
  transactionCount: 1
}
{
  _id: 'Alexis Miller',
  totalIncome: 0,
  totalExpenses: 2596,
  transactionCount: 1
}
{
  _id: 'Alvin Larson',
  totalIncome: 0,
  totalExpenses: 5752,
  transactionCount: 2
}
{
```

INDEXES AND PERFORMANCE

2.5

A. CREATE THREE INDEXES

```
* db.customers.createIndex({name: 1});  
* db.accounts.createIndex({account_id: 1});  
* db.transactions.createIndex({account_id: 1});
```

For the `account_id` of the transactions we used the ascending order because it is the most common in these cases. Then the best sort method is also ascending, so MongoDB can use the index and avoid sorting in memory

B. COMMENTS ABOUT HOW INDEXES AFFECT SOME QUERIES AND AGGREGATIONS

The queries and aggregations that benefit from the indexes are the ones that need to go through the customer names, like in 2.3a or 2.3c. The same happens with anything using `account_id`, either in accounts or transactions, like 2.4c or 2.4d

C. CODE AND COMMENTS ABOUT THE PERFORMANCE OF THE INDEXES

```
db.customers.dropIndex("name_1");  
db.customers.find({name: "James Sanchez"}).explain("executionStats");  
db.customers.createIndex({name: 1});  
db.customers.find({ name: "James Sanchez" }).explain("executionStats");
```

Without the index the query takes around 13 ms and goes through 501 documents, basically the whole collection. MongoDB asks every document if the name James Sanchez is there, so it needs a full COLLSCAN. With the index the time drops to 1 ms, it only touches 1 document and uses an IXSCAN. MongoDB checks the index for the name and, once it finds it, it jumps straight to that document

EXTENSION WITH ONE COLLECTION

2.6

* MINIMAL SCHEMA

```
{ "_id": ObjectId,  
  "merchant_code": "UNQ-STR",  
  "name": "Canonical Merchant Name",  
  "category": "food | travel | shopping | health |  
education",  
  "country": "ES | FR | GB | US | IT" }
```

* A. INSERTION

```
db.merchants.insertMany([  
  {  
    merchant_code: "M001",  
    name: "Mercadona",  
    category: "food",  
    country: "ES"  
  },  
  {  
    merchant_code: "M002",  
    name: "Air France",  
    category: "travel",  
    country: "FR"  
  },  
  {  
    merchant_code: "M003",  
    name: "Burberry",  
    category: "shopping",  
    country: "GB"  
  },  
  {  
    merchant_code: "M004",  
    name: "Johns Hopkins Hospital",  
    category: "health",  
    country: "US"  
  },  
  {  
    merchant_code: "M005",  
    name: "American University of Rome",  
    category: "education",  
    country: "IT"  
  },  
  {  
    merchant_code: "M006",  
    name: "Zara",  
    category: "shopping",  
    country: "ES"  
  },  
])
```

```
{  
  merchant_code: "M007",  
  name: "Le Ratatouille",  
  category: "food",  
  country: "FR"  
},  
{  
  merchant_code: "M008",  
  name: "Oxford University",  
  category: "education",  
  country: "GB"  
},  
{  
  merchant_code: "M009",  
  name: "American Airlines",  
  category: "travel",  
  country: "US"  
},  
{  
  merchant_code: "M010",  
  name: "Farmacia Termini",  
  category: "health",  
  country: "IT"  
},  
{  
  merchant_code: "M011",  
  name: "UC3M",  
  category: "education",  
  country: "ES"  
},  
{  
  merchant_code: "M012",  
  name: "Hôpital de la Pitié-Salpêtrière",  
  category: "health",  
  country: "FR"  
},  
}
```

```
{  
  merchant_code: "M013",  
  name: "British Airways",  
  category: "travel",  
  country: "GB"  
},  
{  
  merchant_code: "M014",  
  name: "Costco",  
  category: "food",  
  country: "US"  
},  
{  
  merchant_code: "M015",  
  name: "Gucci",  
  category: "shopping",  
  country: "IT"  
},  
])
```

* B. CREATE AN INDEX FOR FIELDS MERCHANT_CODE

```
db.merchants.createIndex({ merchant_code: 1 })
```

* C. AGGREGATION PIPELINES FOR MERCHANTS

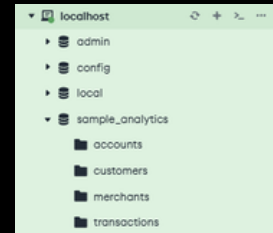
```
db.transactions.aggregate([
  { $unwind: "$transactions" },
  {
    $lookup: {
      from: "merchants",
      localField: "transactions.merchant",
      foreignField: "name",
      as: "merchant_info"
    }
  }
])
```

```
db.transactions.aggregate([
  { $unwind: "$transactions" },
  {
    $lookup: {
      from: "merchants",
      localField: "transactions.merchant",
      foreignField: "name",
      as: "merchant_info"
    }
  },
  { $unwind: "$merchant_info" },
  {
    $group: {
      _id: "$merchant_info.country",
      totalAmount: { $sum: "$transactions.amount" },
      count: { $sum: 1 }
    }
  }
])
```

APPENDIX

how we loaded the dataset

We loaded the dataset in a local MongoDB instance (localhost: 27017). Then we used MongoDB Compass to import the JSON files into the sample_analytics database. After importing, the three collections (accounts, customers, and transactions) appeared as shown in the screenshot. The fourth collection is the one we had to had on the point 2.6



AI-USE STATEMENT

* DECLARATION

We, the undersigned students, hereby declare the following regarding the use of Generative AI tools (e.g., ChatGPT, Copilot, etc.) in the preparation of this assignment: We did use Generative AI tools, specifically: ChatGPT.

* IF AI TOOLS WERE USED, PLEASE DESCRIBE HOW:

We specifically asked the AI not to solve the exercises directly, as we know that when AI solves large exercises it usually performs mistakes. Also, we didn't want to use AI to solve anything for us, we wanted to use it as a helpful tool for consulting and brainstorming. Furthermore, we did not want the AI to write for us. Every text is written entirely by us, human students. We will include here the prompts we used to get help from ChatGPT:

----- PROMPT 1 -----

I'm a Data Science and Engineering student working on a NoSQL project with MongoDB and Compass. I'll send you the full project and the labs from class. I don't want you to solve the project or write the code for me. I just want help step by step. I want to follow the same functions and style my professor uses in class. If there's another possible function or approach, explain it clearly, but my main goal is to stick to what we learned. I only need help understanding concepts and clarifying doubts. I'll write everything myself

----- PROMPT 2 -----

Hi, I have to do a database assignment, I'll send you the PDF of the task and I need you to explain it to me in Spanish in detail and with examples, so I can really understand what they're asking me to do and how to do it. I don't want you to do the assignment for me, but to do it little by little with your help

All parts of the project were done by us. We used ChatGPT only as a tool to understand better certain concepts and clarify specific doubts about MongoDB. It was useful for discussing our design choices and verify that our reasoning was coherent, but every query, explanation, and decision in the final version of the project comes from our own work and understanding.

We confirm that all final decisions, analysis, and conclusions presented in this work are our own responsibility, and we are fully accountable for the content delivered.

CODE

use sample_analytics;

```
// -----  
// 2.2  
// -----
```

```
// 2.2. a
```

```
db.accounts.findOne({account_id: 999876});
```

```
db.accounts.insertOne({account_id: 999876, limit: 10000, products : ["InvestmentStock", "Commodity"]});
```

```
db.customers.insertOne({username: "rafanadal", name: "Rafael Nadal", address: "Calle de la Palmera, 12",  
birthdate: ISODate("2003-01-01T00:00:00Z"), email: "rafanadal10@gmail.com", active: true, accounts:  
[999876], tier_and_details: {}});
```

```
// 2.2. b
```

```
db.accounts.updateOne({account_id: 999876}, {$push: {products: "Brokerage"}});
```

```
// 2.2.c
```

```
db.transactions.updateOne({account_id: 852986}, {$push: {transactions: {date: ISODate("2024-08-  
01T00:00:00Z"), amount: 100, transaction_code: "buy", symbol: "nflx", price: "20.60", total: "2060"}}});
```

```
db.transactions.findOne({account_id: 852986});
```

```
// -----  
// 2.3  
// -----
```

```
// 2.3. a
```

```
db.customers.find({name: "Brian Flores"}, {accounts: 1, _id: 0});
```

```
// 2.3. b
```

```
db.transactions.find({}, {_id: 0, account_id: 1, bucket_end_date: 1, transactions: 1}).sort({ bucket_end_date:  
-1}).limit(10);
```

```
db.transactions.aggregate([{$unwind: "$transactions"}, {$sort: {"transactions.date": -1}}, {$limit: 10},  
{$project: {_id: 0, account_id: 1, date: "$transactions.date", amount:  
"$transactions.amount", transaction_code: "$transactions.transaction_code", symbol:  
"$transactions.symbol", total: "$transactions.total"}}]);
```

```
// 2.3. c
```

```
db.customers.find({name: {$regex: "Br", $options: "i"}, {_id: 0, name: 1, accounts: 1});
```

```
// -----  
// 2.4  
// -----  
  
// 2.4. a  
db.transactions.aggregate([  
  { $unwind: "$transactions" },  
  {  
    $group: {  
      _id: "$transactions.symbol",  
      totalAmount: { $sum: "$transactions.amount" }  
    }  
  },  
  { $sort: { totalAmount: -1 } },  
  { $limit: 5 }  
])  
  
// 2.4. b  
db.accounts.aggregate([  
  {  
    $group: {  
      _id: "$limit",  
      accounts: { $push: "$_id" },  
      count: { $sum: 1 }  
    }  
  }  
])  
  
// 2.4. c  
db.transactions.aggregate([  
  { $unwind: "$transactions" },  
  {  
    $lookup: {  
      from: "customers",  
      localField: "account_id",  
      foreignField: "accounts",  
      as: "customer_info"  
    }  
  },  
  { $unwind: "$customer_info" },  
  {  
    $project: {  
      _id: 0,  
      account_id: 1,  
      date: "$transactions.date",  
      amount: "$transactions.amount",  
      customerName: "$customer_info.name"  
    }  
  }  
])
```

```

// 2.4. d
/db.transactions.aggregate([
  { $unwind: "$transactions" },
  {
    $match: {
      "transactions.date": {
        $gte: ISODate("2017-01-01"),
        $lt: ISODate("2017-04-01")
      }
    }
  },
  {
    $lookup: {
      from: "customers",
      localField: "account_id",
      foreignField: "accounts",
      as: "customer_info"
    }
  },
  { $unwind: "$customer_info" },
  {
    $group: {
      _id: "$customer_info.name",
      totalIncome: {
        $sum: {
          $cond: [
            { $eq: ["$transactions.transaction_code", "sell"] },
            "$transactions.amount",
            0
          ]
        }
      },
      totalExpenses: {
        $sum: {
          $cond: [
            { $eq: ["$transactions.transaction_code", "buy"] },
            "$transactions.amount",
            0
          ]
        }
      },
      transactionCount: { $sum: 1 }
    }
  },
  { $sort: { _id: 1 } }
])

```

```
// -----  
// 2.5  
// -----  
  
// 2.5. a  
// For the name of the customers  
db.customers.createIndex({name: 1});  
  
// For the account_id of the accounts  
db.accounts.createIndex({account_id: 1});  
  
// For the account_id of the transactions  
db.transactions.createIndex({account_id: 1});  
  
// 2.5. c  
db.customers.dropIndex("name_1");  
  
db.customers.find({name: "James Sanchez"}).explain("executionStats");  
  
db.customers.createIndex({name: 1});  
  
db.customers.find({ name: "James Sanchez" }).explain("executionStats");  
  
// -----  
// 2.6  
// -----  
  
// 2.6. a  
db.merchants.insertMany([  
  {  
    merchant_code: "M001",  
    name: "Mercadona",  
    category: "food",  
    country: "ES"  
  },  
  {  
    merchant_code: "M002",  
    name: "Air France",  
    category: "travel",  
    country: "FR"  
  },  
  {  
    merchant_code: "M003",  
    name: "Burberry",  
    category: "shopping",  
    country: "GB"  
  },  
])
```

```
{
  merchant_code: "M004",
  name: "Johns Hopkins Hospital",
  category: "health",
  country: "US"
},
{
  merchant_code: "M005",
  name: "American University of Rome",
  category: "education",
  country: "IT"
},
{
  merchant_code: "M006",
  name: "Zara",
  category: "shopping",
  country: "ES"
},
{
  merchant_code: "M007",
  name: "Le Ratatouille",
  category: "food",
  country: "FR"
},
{
  merchant_code: "M008",
  name: "Oxford University",
  category: "education",
  country: "GB"
},
{
  merchant_code: "M009",
  name: "American Airlines",
  category: "travel",
  country: "US"
},
{
  merchant_code: "M010",
  name: "Farmacia Termini",
  category: "health",
  country: "IT"
},
{
  merchant_code: "M011",
  name: "UC3M",
  category: "education",
  country: "ES"
},
{
  merchant_code: "M012",
  name: "Hôpital de la Pitié-Salpêtrière",
  category: "health",
  country: "FR"
},
{
  merchant_code: "M013",
  name: "British Airways",
  category: "travel",
  country: "GB"
},
{
  merchant_code: "M014",
  name: "Costco",
  category: "food",
  country: "US"
},
{
  merchant_code: "M015",
  name: "Gucci",
  category: "shopping",
  country: "IT"
},
});
```

```
// 2.6. b
db.merchants.createIndex({ merchant_code: 1 });
```

```
// 2.6. c
db.transactions.aggregate([
  { $unwind: "$transactions" },
  {
    $lookup: {
      from: "merchants",
      localField: "transactions.merchant",
      foreignField: "name",
      as: "merchant_info"
    }
  }
])
```

```
db.transactions.aggregate([
  { $unwind: "$transactions" },
  {
    $lookup: {
      from: "merchants",
      localField: "transactions.merchant",
      foreignField: "name",
      as: "merchant_info"
    }
  },
  { $unwind: "$merchant_info" },
  {
    $group: {
      _id: "$merchant_info.country",
      totalAmount: { $sum: "$transactions.amount" },
      count: { $sum: 1 }
    }
  }
])
```