MAPÚA UNIVERSITY

SCHOOL OF ELECTRICAL, ELECTRONICS, AND COMPUTER ENGINEERING

Experiment 6:
NoSQL Database Models

CPE106L (Software Design Laboratory)

Brian Matthew E. Clemente

Maria Angelica Acantilado

Chalzea Fransen C. Dytianquin

Vance David G. Samia

Group No.: **4**

Section: **FOPI01**

PreLab

Readings, Insights, and Reflection

METIS #1:
Building Cross-Platform Mobile and Web Apps for Engineers and Scientists: An Active Learning Approach. Lingras P.

Chapter 9 [Section 9.1 to 9.6]
The material of this report explores the evolution and practical applications of NoSQL databases, particularly MongoDB, as discussed in Lingras' material. The shift from traditional relational databases to NoSQL models is driven by the need to manage unstructured and semi-structured data more effectively. MongoDB's document-oriented structure offers flexibility and scalability, making it a valuable tool for modern applications with dynamic data requirements.

A key takeaway is the contrast between structured relational databases and schema-less NoSQL models. While relational databases excel in structured data organization, they struggle with evolving and diverse data formats. NoSQL, particularly MongoDB, addresses these limitations by allowing dynamic schemas and JSON-like document storage. However, the lack of strict schema enforcement also introduces challenges, such as potential data inconsistencies and runtime errors, emphasizing the importance of thoughtful database design.

The practical aspects of MongoDB, including setting up a server and saving user data, reinforce its usability in real-world applications. Despite the advantages of NoSQL, the absence of built-in graphical tools in MongoDB can pose a learning curve for beginners. Nevertheless, tools like MongoDB Compass help bridge this gap. Overall, the material highlights the significance of NoSQL databases in modern data management while underscoring the trade-offs between flexibility and structure.

Clemente (Chapter 9)
Lingras' material provides an insightful exploration of NoSQL databases, particularly focusing on MongoDB and its practical applications. The emergence of NoSQL database models (Section 9.1) highlights their significance in handling unstructured and semi-structured data, addressing the limitations of traditional relational databases. The introduction to MongoDB (Section 9.2) serves as a foundational guide, emphasizing its document-based structure and flexibility.

A particularly engaging aspect is the modelling of a NoSQL database (Sections 9.3–9.4), which illustrates how real-world applications, such as the Thyroid App, benefit from schema-less designs. The step-by-step process of launching a MongoDB server (Section 9.5) and saving user data (Section 9.6) provides a hands-on approach, reinforcing the practicality of NoSQL databases in application development.

Overall, these sections effectively bridge theoretical concepts with practical implementation, demonstrating MongoDB's role in modern database management.

Acantilado (Chapter 9)
Lingras' discussion on NoSQL databases, particularly MongoDB, presents a compelling case for their use in handling unstructured and semi-structured data. Section 9.1 introduces the core differences between NoSQL and traditional relational databases, emphasizing the flexibility and scalability that NoSQL models offer. The breakdown of MongoDB's document-based approach in Section 9.2 highlights how data is stored in a way that accommodates evolving application requirements without the constraints of rigid schemas. Sections 9.3 and 9.4 are particularly engaging, as they demonstrate how NoSQL modelling is applied in real-world scenarios, such as the Thyroid App, where a schema-less structure allows for dynamic data storage. Furthermore, the step-by-step guide on setting up a MongoDB server (Section 9.5) and saving user data (Section 9.6) provides a hands-on experience that reinforces the practical benefits of NoSQL databases. By integrating theoretical concepts with real-world applications, these sections offer a well-rounded understanding of how MongoDB serves as a powerful tool in modern database management.

Dytianquin (Chapter 9)
Reflecting on the evolution of database management, the advantages of NoSQL technologies like MongoDB become clear when examining concrete applications. For example, the Thyroid App, as discussed in Sections 9.3–9.4, demonstrates how a flexible, schema-less design can effectively manage evolving, unstructured data compared to traditional relational models. The step-by-step guidance provided in Sections 9.5 and 9.6 on launching a MongoDB server and handling user data offers tangible evidence of how theoretical concepts are translated into practical tools. This practical approach highlights that while moving away from strict relational models simplifies the management of diverse data, it also necessitates rigorous design and testing to prevent inconsistencies and runtime errors. Overall, the transition to NoSQL—as exemplified by the Thyroid App and real-world server configuration—represents a calculated response to the dynamic challenges of modern data management.

Samia (Chapter 9)
The first part of the chapter traces the evolution of databases over time, starting with the early network and hierarchical models and ending with E.F. Codd in 1970. This background is essential to comprehending why relational databases' shortcomings in managing unstructured data led to the emergence of NoSQL databases. Because of its organized design and focus on normalization, the relational model is best suited for corporate applications; nevertheless, it is less appropriate for contemporary applications that handle heterogeneous and unstructured data. This prepares the way for the advent of NoSQL databases, which provide more scalability and flexibility. The document-oriented structure of MongoDB, a popular NoSQL database, is

highlighted in this chapter. MongoDB employs collections and documents in a format similar to JSON, in contrast to relational databases, which store data in tables. Because of its adaptability, developers can store data without following preset schemas, which makes it perfect for applications whose data needs change over time. Despite their strength, NoSQL databases' unstructured nature has drawbacks. For instance, mistakes that may not be detected until runtime may result from lax schema enforcement. This emphasizes how crucial thorough design and testing are when dealing with NoSQL systems.

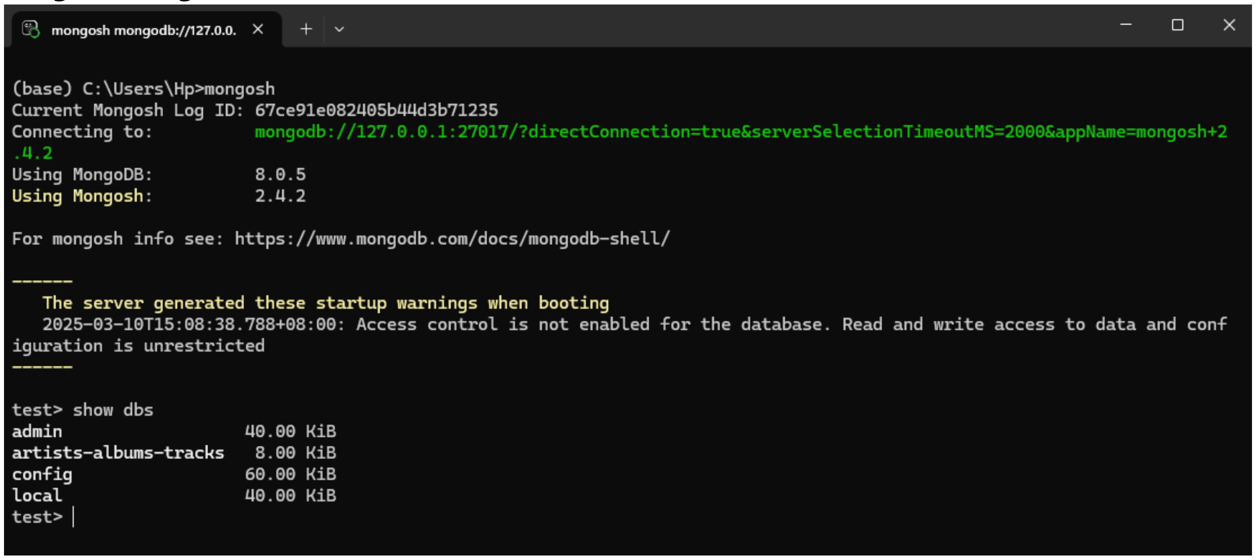
One important lesson is that while NoSQL databases, such as MongoDB, provide unmatched flexibility in managing unstructured data, they do so at the expense of less structure and a higher chance of errors. Although the chapter does not examine current tools like MongoDB Compass, which could lessen the learning curve for novices, the dearth of graphical tools for MongoDB is acknowledged.

PostLab

Programming Problems

This post-lab activity focuses on the implementation of NoSQL database models, with a particular emphasis on utilizing MongoDB and MongoDB Compass for efficient data management, scalability, and flexibility. The tasks involve applying commonly used NoSQL data models, such as documents, key-value, column-family, and graph databases, to store and retrieve structured and unstructured data effectively. Additionally, the lab explores the process of exporting SQL database records as JSON files and importing them into MongoDB to facilitate seamless data migration between relational and NoSQL systems. Data validation techniques are integrated to ensure consistency and integrity across different database operations. Each programming task was distributed among group members, with contributions documented within the code files to maintain clarity and organization. The development process emphasized structured data modeling techniques, enabling a systematic approach to designing efficient and scalable NoSQL schemas. MongoDB Compass was utilized for database visualization and management, ensuring a user-friendly interface for querying and analyzing stored data. Version control tools were used to track changes, ensuring seamless collaboration and database integrity.

Programming Problem

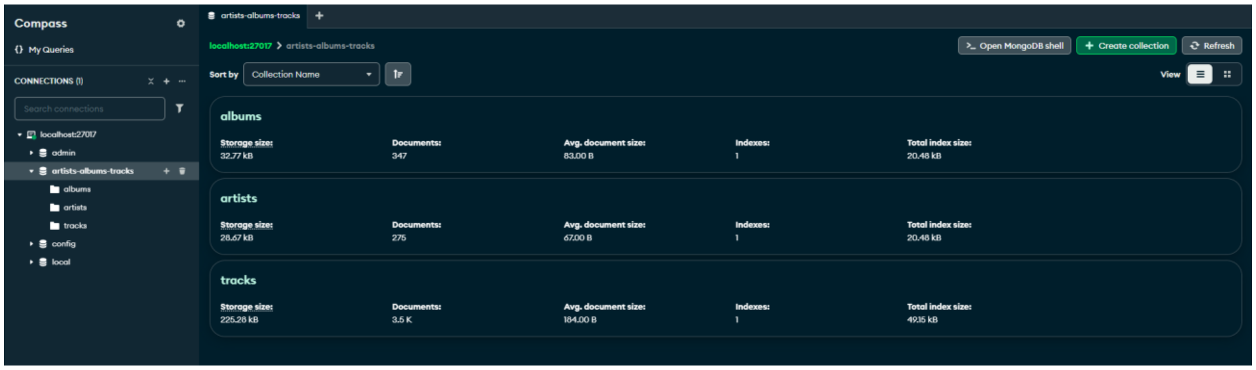


```
(base) C:\Users\Hp>mongosh
Current Mongosh Log ID: 67ce91e082405b44d3b71235
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2
.4.2
Using MongoDB:          8.0.5
Using Mongosh:          2.4.2

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

------
   The server generated these startup warnings when booting
   2025-03-10T15:08:38.788+08:00: Access control is not enabled for the database. Read and write access to data and conf
iguration is unrestricted
------

test> show dbs
admin                  40.00 KiB
artists-albums-tracks   8.00 KiB
config                 60.00 KiB
local                  40.00 KiB
test> |
```

Figure 1.1 Screenshot of Mongo Shell executed in the Anaconda environment to test Mongo Local Connectivity and Available Databases

Figure 1.1 illustrates the execution of MongoDB Shell within the Anaconda environment, demonstrating the successful connection to a locally hosted MongoDB instance. The command executed in the shell verifies that the MongoDB server is running and accessible from the

Anaconda environment. Additionally, the output displays a list of available databases, confirming proper configuration and database recognition, which is crucial in ensuring that MongoDB is correctly installed and can be interacted with before proceeding with database operations such as inserting, querying, and aggregating data.



albums

Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
32.77 kB	347	83.00 B	1	20.48 kB

artists

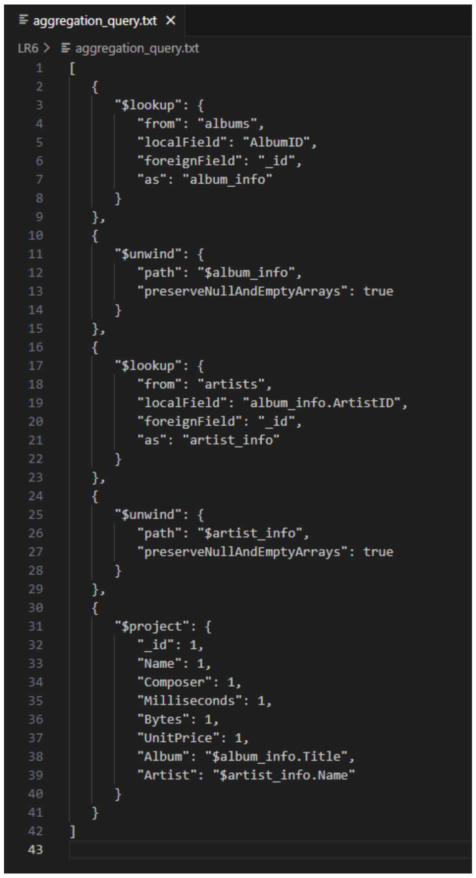
Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
28.67 kB	275	67.00 B	1	20.48 kB

tracks

Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
225.28 kB	3.5 K	184.00 B	1	49.15 kB

Figure 1.2 artists-albums-tracks Database created in MongoDB Compass along with the Albums, Artists, and Tracks Collections

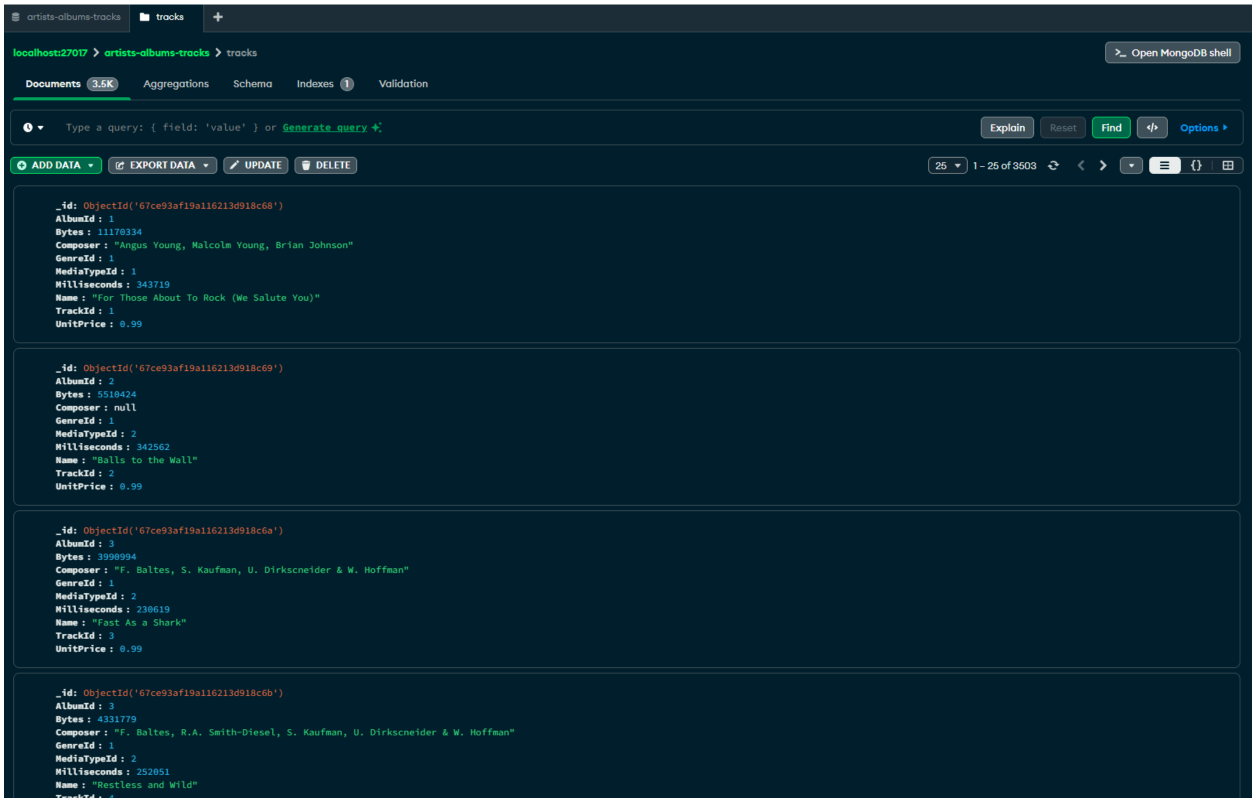
Figure 1.2 showcases the successful creation of the `artists-albums-tracks` database in MongoDB Compass. The screenshot highlights the database structure, including the albums, artists, and tracks collections, which store the respective entities. This database organization follows a relational structure within MongoDB, where tracks are linked to albums, and albums are associated with artists. The clear separation of collections ensures efficient data management and facilitates aggregation queries for retrieving related information.



```
[
    {
        "$lookup": {
            "from": "albums",
            "localField": "AlbumID",
            "foreignField": "_id",
            "as": "album_info"
        }
    },
    {
        "$unwind": {
            "path": "$album_info",
            "preserveNullAndEmptyArrays": true
        }
    },
    {
        "$lookup": {
            "from": "artists",
            "localField": "album_info.ArtistID",
            "foreignField": "_id",
            "as": "artist_info"
        }
    },
    {
        "$unwind": {
            "path": "$artist_info",
            "preserveNullAndEmptyArrays": true
        }
    },
    {
        "$project": {
            "_id": 1,
            "Name": 1,
            "Composer": 1,
            "Milliseconds": 1,
            "Bytes": 1,
            "UnitPrice": 1,
            "Album": "$album_info.Title",
            "Artist": "$artist_info.Name"
        }
    }
]
```

Figure 1.3 Aggregation Query used in MongoDB Compass

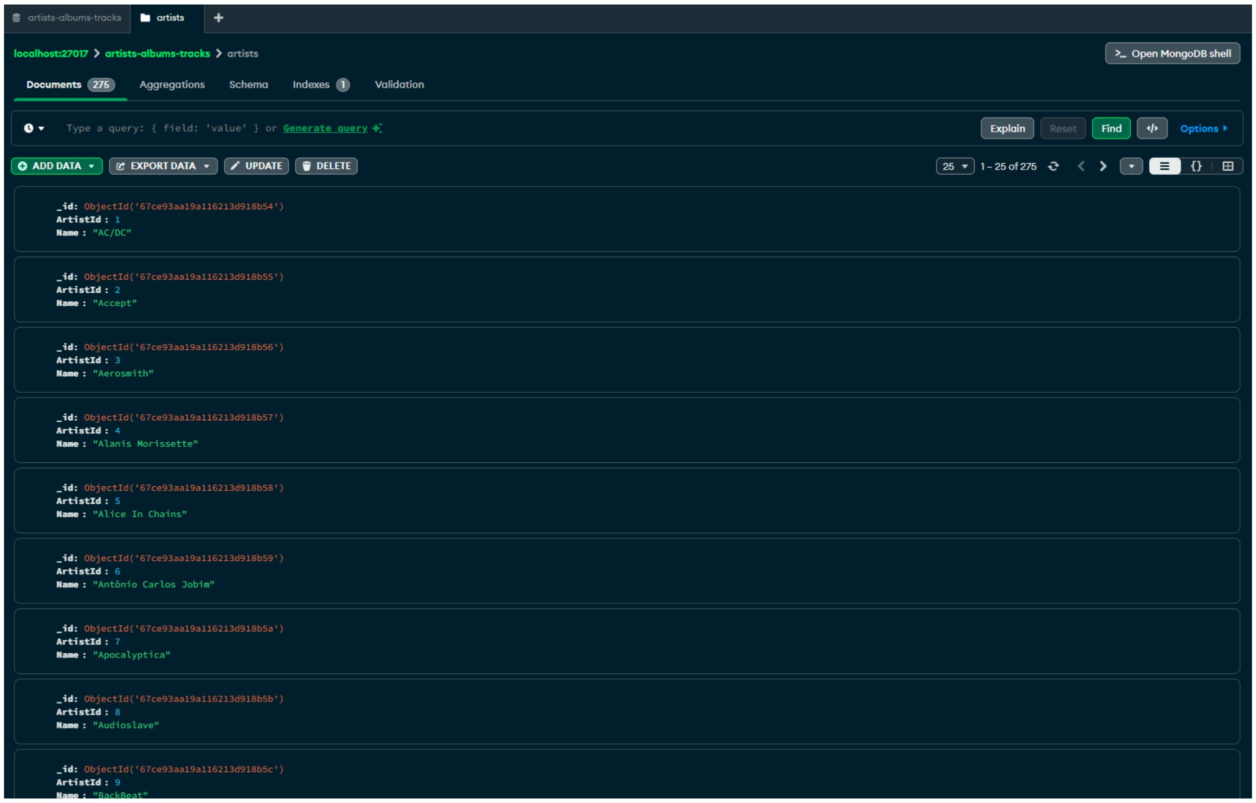
Figure 1.3 displays the aggregation query utilized in MongoDB Compass, joining the tracks, albums, and artists collections. The output presents track details along with their corresponding album and artist information in a structured format.



```
_id: ObjectId('67ce93af19a116213d918c68')
AlbumId : 1
Bytes : 11170334
Composer : "Angus Young, Malcolm Young, Brian Johnson"
GenreId : 1
MediaTypeId : 1
Milliseconds : 343719
Name : "For Those About To Rock (We Salute You)"
TrackId : 1
UnitPrice : 0.99

_id: ObjectId('67ce93af19a116213d918c69')
AlbumId : 2
Bytes : 5510424
Composer : null
GenreId : 1
MediaTypeId : 2
Milliseconds : 342562
Name : "Balls to the Wall"
TrackId : 2
UnitPrice : 0.99

_id: ObjectId('67ce93af19a116213d918c6a')
AlbumId : 3
Bytes : 3990994
Composer : "F. Baltes, S. Kaufman, U. Dirkscneider & W. Hoffman"
GenreId : 1
MediaTypeId : 2
Milliseconds : 230619
Name : "Fast As a Shark"
TrackId : 3
UnitPrice : 0.99

_id: ObjectId('67ce93af19a116213d918c6b')
AlbumId : 3
Bytes : 4331779
Composer : "F. Baltes, R.A. Smith-Diesel, S. Kaufman, U. Dirkscneider & W. Hoffman"
GenreId : 1
MediaTypeId : 2
Milliseconds : 252051
Name : "Restless and Wild"
```

Figure 1.4 tracks.json File Data Imported into Tracks Collection

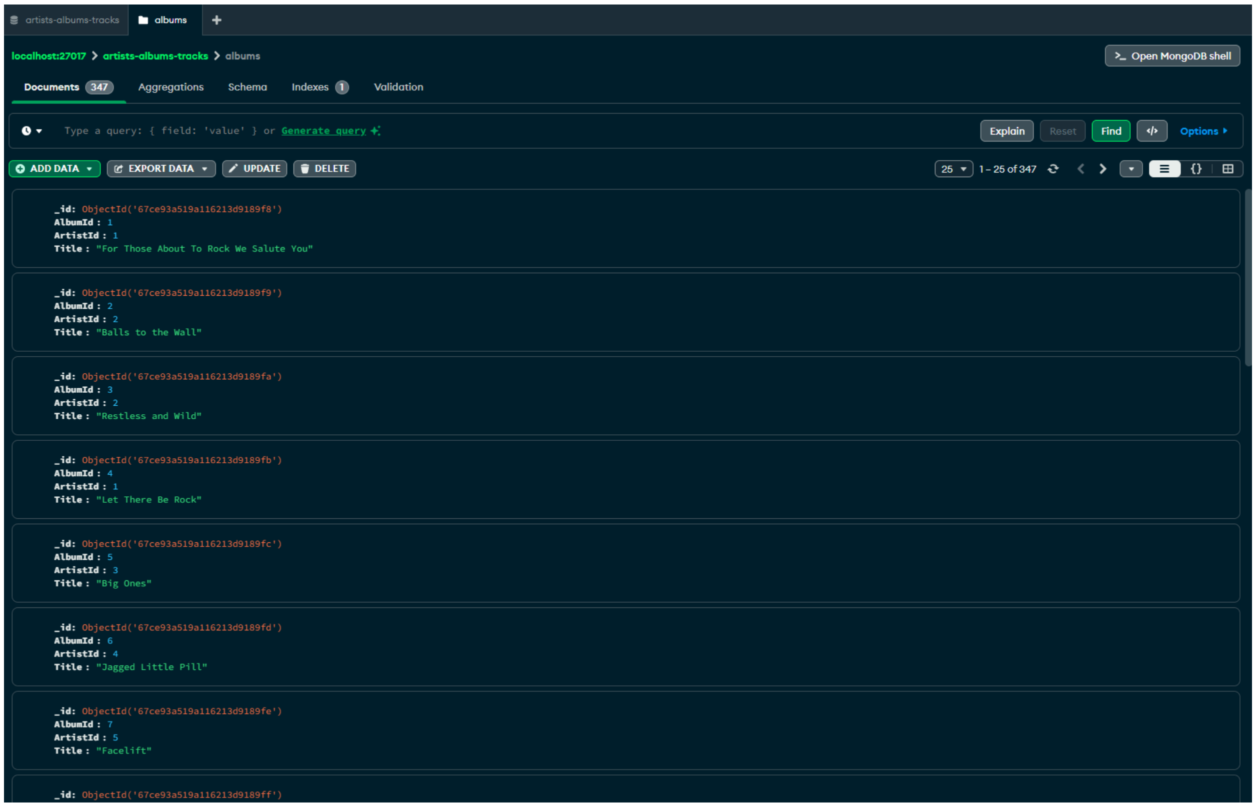
Figure 1.4 displays the successful import of the `tracks.json` file into the tracks collection within MongoDB Compass. The imported data includes key attributes such as `TrackID`, `AlbumID`, `GenreID`, `Composer`, `Milliseconds`, `Bytes`, and `UnitPrice`. This collection serves as the foundation for storing track-related details, which can be queried and linked to albums and artists through aggregation operations. The correct import ensures that all track records are available for further processing and retrieval.



```
_id: ObjectId('67ce93aa19a116213d918b54')
ArtistId : 1
Name : "AC/DC"

_id: ObjectId('67ce93aa19a116213d918b55')
ArtistId : 2
Name : "Accept"

_id: ObjectId('67ce93aa19a116213d918b56')
ArtistId : 3
Name : "Aerosmith"

_id: ObjectId('67ce93aa19a116213d918b57')
ArtistId : 4
Name : "Alanis Morissette"

_id: ObjectId('67ce93aa19a116213d918b58')
ArtistId : 5
Name : "Alice In Chains"

_id: ObjectId('67ce93aa19a116213d918b59')
ArtistId : 6
Name : "Antônio Carlos Jobim"

_id: ObjectId('67ce93aa19a116213d918b5a')
ArtistId : 7
Name : "Apocalyptica"

_id: ObjectId('67ce93aa19a116213d918b5b')
ArtistId : 8
Name : "Audioslave"

_id: ObjectId('67ce93aa19a116213d918b5c')
ArtistId : 9
Name : "BackBeat"
```

Figure 1.5 artists.json File Data Imported into Artists Collection

Figure 1.5 illustrates the import process of the `artists.json` file into the artists collection. The screenshot confirms that artist data, including `ArtistID` and `Name`, has been successfully added to the database. This collection plays a crucial role in maintaining artist information and establishing relationships with albums. The imported data allows for efficient querying and retrieval of artist-related details, ensuring proper integration within the database.



```
_id: ObjectId('67ce93a519a116213d9189f8')
AlbumId : 1
ArtistId : 1
Title : "For Those About To Rock We Salute You"

_id: ObjectId('67ce93a519a116213d9189f9')
AlbumId : 2
ArtistId : 2
Title : "Balls to the Wall"

_id: ObjectId('67ce93a519a116213d9189fa')
AlbumId : 3
ArtistId : 2
Title : "Restless and Wild"

_id: ObjectId('67ce93a519a116213d9189fb')
AlbumId : 4
ArtistId : 1
Title : "Let There Be Rock"

_id: ObjectId('67ce93a519a116213d9189fc')
AlbumId : 5
ArtistId : 3
Title : "Big Ones"

_id: ObjectId('67ce93a519a116213d9189fd')
AlbumId : 6
ArtistId : 4
Title : "Jagged Little Pill"

_id: ObjectId('67ce93a519a116213d9189fe')
AlbumId : 7
ArtistId : 5
Title : "Facelift"

_id: ObjectId('67ce93a519a116213d9189ff')
```

Figure 1.6 albums.json File Data Imported into Albums Collection

Figure 1.6 presents the successful import of the `albums.json` file into the albums collection. This collection contains essential attributes such as `AlbumID`, `ArtistID`, and `Title`, which establish connections between albums and their respective artists. The imported data enables seamless aggregation queries that link albums to both their tracks and artists. Proper data import ensures the integrity of the database and allows for efficient data retrieval within MongoDB Compass.