MongoDB for DiscountMate

T3 2025 – Rebekah De Losa, SIT764

Introduction

MongoDB is currently used in this project to house the data that feeds into DiscountMate. As a NoSQL Database Management system, MongoDB offers flexibility to data storage that does not rely on structures. While it is not inherently built for relational databases, for best practice and performance of the DiscountMate project we have opted to build our MongoDB database in a way that supports relationships across collections.

Structure of the DiscountMate MongoDB

Organise your access

To get access to the DiscountMate project in MongoDB for the first time, you will need to create an account with MongoDB using your Deakin Email address. You may then contact your company mentor to have them grant access to you.

> ***Link to create an account:***
>
> https://account.mongodb.com/account/register

Databases

Once access is provided, within the DiscountMate MongoDB there are many databases to connect with. The various databases represent different iterations of data collection strategies over the years that DiscountMate has been active. You will need to connect with your team to understand which database(s) are currently in use.

*Note: At end of T3 2025, DiscountMate is using the **DiscountMate_DB** database (**Figure 1**).*

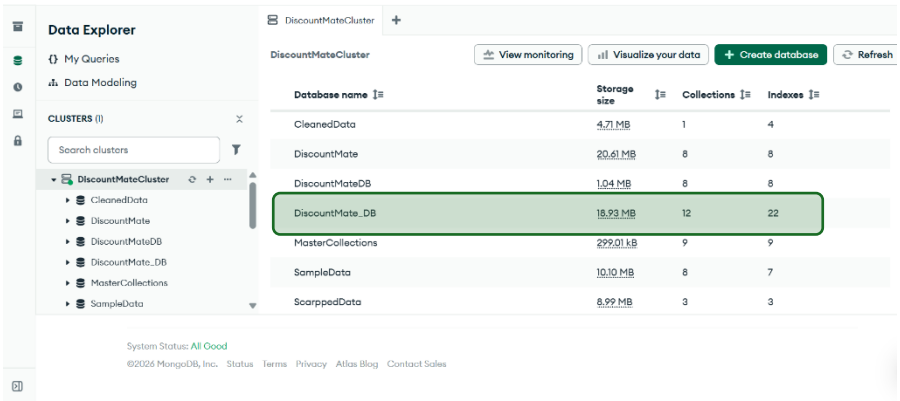

Figure 1. MongoDB database in use at end of T3 2025.

Collections

Within a database in MongoDB, there can be many collections. Collections can be used to house data with shared characteristics. Depending on the set up of your database, there may be a schema to be aware of in order to understand how the collections connect to each other.

Collections can also be thought of as **tables** as they might in a relational database system. It is likely that in a given database, there are connections between each of the collections.

*Note: Please refer to the DiscountMate schema image to view the structure of relationships between collections and their associated data fields (**Figure 2**).*

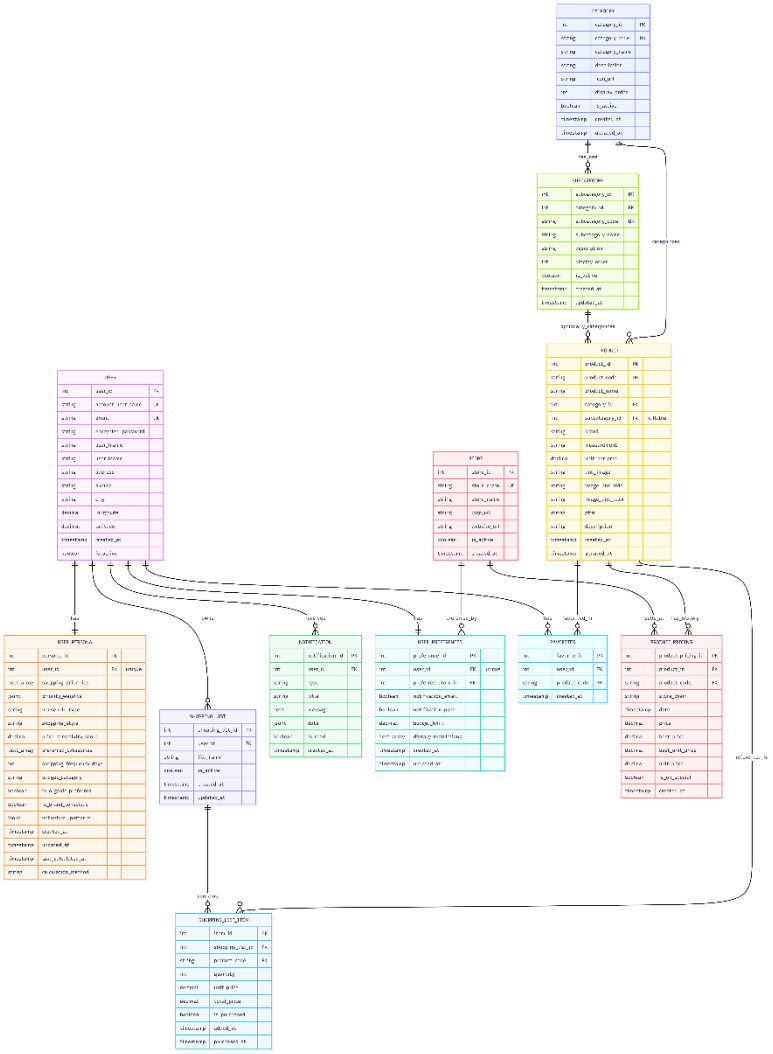

Figure 2. DiscountMate schema in use at end of T3 2025.

Documents

Inside each collection there are documents that contain each instance of data. These can also be thought of as *records* or *rows*. As an example, for each product in DiscountMate there is a document represented in the 'Products' collection.

Fields

Each document contains a number of fields that are common to every document in the collection. These fields represent attributes for each instance of data added.

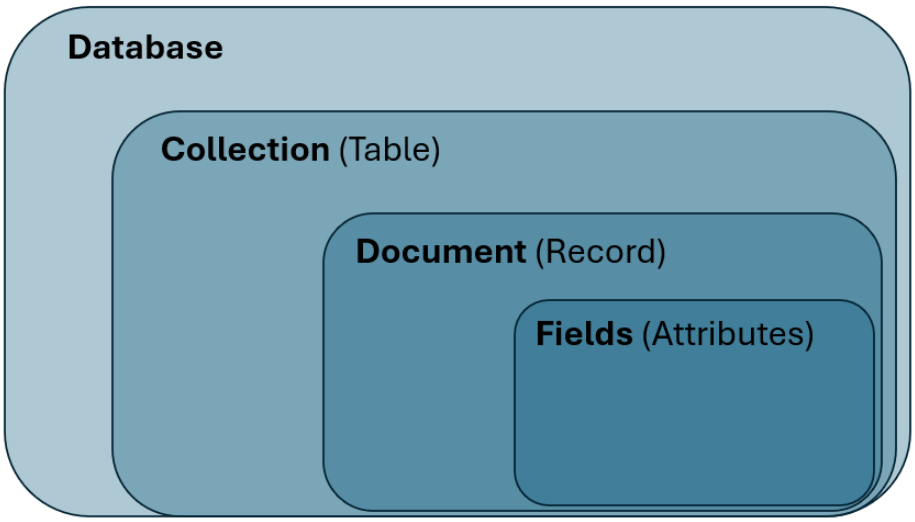

Figure 3. Structure of databases in MongoDB.

Connecting to MongoDB using python

Once you have access to MongoDB you may want to run scripts that create, update or extract data from your database. To do this with python the following steps are required.

1. **Prepare a *.env* file with the DiscountMate connection string. Ensure this file is saved within your working environment.**
   a. *Find the connection string for your team – this can be found by clicking 'connect now' from the DiscountMate project homepage.*

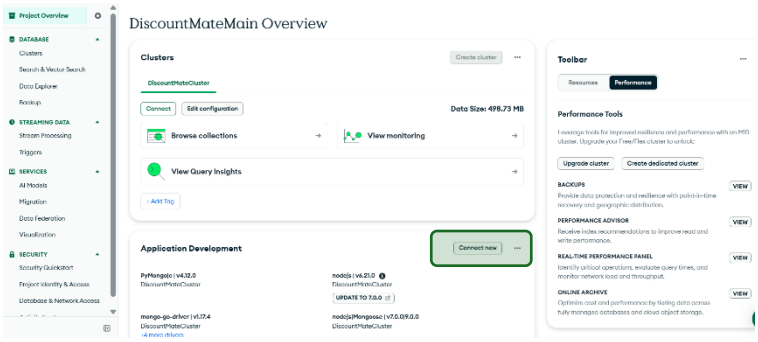

Figure 4. Retrieving connection string.

```
mongodb+srv://<db_username>:<db_password>@<connection>/?appName=<cluster_name>
```

Figure 5. Example connection string.

   b. *DiscountMate currently use a common database user profile to fill in the connection string. You can get the username and password by navigating to **Security (left hand pane) > Database & Network access**. In this list, one user has the password visible in the description. This is the username and password to input in the connection string.*

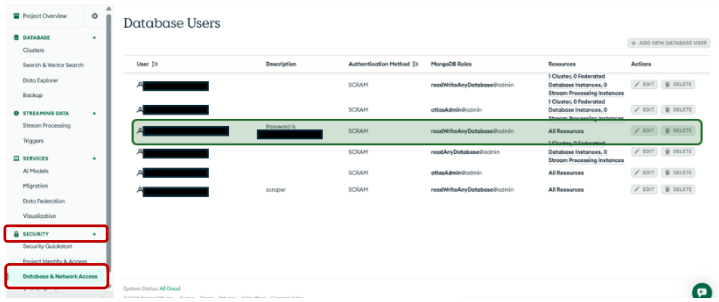

Figure 6. Where to find the username and password for your connection string in MongoDB.

c. *Open a blank text file.*

d. *In the text file, create a 'MONGO_URI' variable and assign the connection string from step 1a to it. Ensure you add the actual username and password retrieved in step 1b. The cluster name and connection will be already visible in your copied connection string from step 1a.*

Figure 7. Example .env file. <db_username> and <db_password> must be replaced with the retrieved username and password (Step 1b).

e. *Save the file as '.env' in your local python working environment.*

Figure 8. Saving the .env file.

2. Ensure *pymongo* and *dotenv* python packages are installed.

```
1.  pip install pymongo
2.  pip install python-dotenv
```

Figure 9. Using pip to install required packages.

Once the above steps are followed, in each script that you connect to MongoDB you will need to set up your client and specify the database you want to interact with. You can use the code in **Figure 10** to do this.

```
1. # import the required packages
2. from pymongo import MongoClient
3. from dotenv import load_dotenv
4.
5. # Load the .env file you created
6. load_dotenv()
7.
8. # Retrieve the mongo URI from your .env file.
9. mongo_uri = os.getenv('MONGO_URI') # Ensure syntax is correct for your instance
```

```
10.
11. # Create your Mongo client
12. client = MongoClient(mongo_uri)
13.
14. # Set up you
15. database = # Specify the name of your database here
16.
17. # Load database
18. db = client[database]
```

Figure 10. Connecting to your mongo database with python

Downloading from MongoDB

Downloading data from MongoDB may be required during the course of your project. Before you extract data ensure that you understand the schema (if any) for the collection you are extracting. Understanding the schema and the relationships between collections will ensure that you can extract data into a flat format easily.

The following is an example of how you can extract the contents of a collection into a pandas dataframe

1. **Ensure the client connection is set up** (see **Figure 10**).
2. **Define a data extraction function and run it.**

```
 1. import pandas as pd
 2.
 3. def get_data(table, database):
 4.     """Extracts data from a MongoDB collection.
 5.
 6.     Args:
 7.         table: (str) the name of the collection to target.
 8.         database: variable that connects to your MongoDB database.
 9.
10.     Returns:
11.         A dataframe with each document in the collection as a row, each field as a column.
12.     """
13.     return pd.DataFrame(database[table].find({}))
14.
15. products_table = get_data('products', db)
```

Figure 11. Example of a data extraction function to pull a collection from a MongoDB database.

3. **If required, depending on your database schema, you can use joins to create a single master table.**

```
1. products_table = get_data('products', db)                        # Collection 1
2. product_pricings_table = get_data('product_pricings', db)        # Collection 2
3.
4. joined_data = pd.merge(products_table,
5.                         product_pricings_table,
6.                         how='left',
7.                         left_on='_id',                           # Primary key
8.                         right_on='product_id')                   # Foreign key
```

Figure 12. Example of joining two collections together in a notebook.

Example of making changes to MongoDB collection

Please note: *Only upload to your database if you are certain of your script and the repercussions in the project workflow. Data uploaded with errors can cause downstream issues that may be difficult to fix. Testing with a local version must be done before applying in production.*

```
def create_data(database, tablename, newDocuments):
    """Upload new documents to a mongo collection.

    Args:
        tablename: (str) the name of the collection to target.
        database: variable that connects to your MongoDB database.
        newDocuments: a list of dictionaries, each item on the list representing a new record, each key-value pairing representing a field-data value pairing
    Returns:
        A final dataframe with all documents in the collection as a row, each field as a column.
    """
    collection = database[tablename]
    for doc in newDocuments:
        collection.insert_one(doc)
    table = pd.DataFrame(collection.find({})) # re-import the table with updated documents
    return table
```

Figure 13. Example of a function that writes new documents to a collection within your database.

For the above function to work correctly, once your processed data is ready to be uploaded it needs to be converted into the right format prior. The required format would be a list of dictionaries.

```
# Transform pre-processed dataframe into a list of dictionaries
upload_docs = df.to_dict(orient='records') # A dictionary per row.

# Upload the data and return the full completed collection as a dataframe
final_table = create_data(db, 'products', upload_docs)

```

Figure 14. Example of transforming a dataframe of processed data into a list of dictionaries.