Ministry of Higher Education and Scientific Research
University of Mila
Institute of Mathematics and Computer Science
Department of Computer Science
Master 2 I2A – Big Data
2025/2026

Practical Works 6 - MongoDB Implementation with Python

Objective: Connect to a MongoDB database, seed it with data, and perform CRUD operations using Python.
Choose Your Path:

- **Option A (Cloud - Recommended):** Use Google Colab + MongoDB Atlas. No installation required.
- **Option B (Local):** Install MongoDB Community Server + Python locally.

Part 1: Environment Setup

Option A: Cloud Setup (Google Colab + Atlas)

1. **Create Database:**
    - Go to MongoDB Atlas and sign up (Free Tier).
    - Create a **Cluster** (select "M0 Sandbox" for free).
    - **Network Access:** Go to Security > Network Access > Add IP Address > **Allow Access from Anywhere (0.0.0.0/0)**. (Crucial for Colab access).
    - **Database User:** Create a user (e.g., `student`) and password.
    - **Get Connection String:** Click "Connect" > "Drivers" > Copy the URL (starts with `mongodb+srv://...`).
2. **Open Python Environment:**
    - Go to Google Colab.
    - Create a "New Notebook".
3. **Install Driver:** In the first cell of Colab, run:

```
!pip install pymongo[srv]
```

Option B: Local Setup

1. Ensure MongoDB Community Server is running.
2. Install the driver in your terminal: `pip install pymongo`.

Part 2: establishing the Connection

Create a new Python script (or Colab cell) and use the following code. **Uncomment the connection string that matches your setup.**

```
from pymongo import MongoClient
import datetime
import pprint

# --- CONFIGURATION ---

# OPTION A: CLOUD (Replace <password> with your actual password)
# connection_string = "mongodb+srv://student:<password>@cluster0.mongodb.net/?retryWrites=true&w=majority"

# OPTION B: LOCAL
connection_string = "mongodb://localhost:27017/"

# ---------------------

try:
    client = MongoClient(connection_string)
    db = client['techstore_db']

    # Test command to verify connection
    client.admin.command('ping')
    print(" Connection Successful! Connected to database:", db.name)

except Exception as e:
    print(" Connection failed:", e)
```

Part 3: Database Initialization (Seeding)

Copy the following code to create the collections and insert initial data.

Note: We use `drop()` to ensure we start with a clean state every time we run the script.

```
# --- RESET (Clear old data) ---
db.users.drop()
db.products.drop()
db.orders.drop()

# --- DATA SEEDING ---

# 1. Insert Products
products_data = [
    {
        "_id": "prod_001",
        "name": "Gaming Laptop X1",
        "price": 1500.00,
        "category": "Computers",
        "tags": ["gaming", "high-performance", "rgb"],
        "specs": {"ram": "16GB", "cpu": "i7"},
        "stock": 10
    },
    {
        "_id": "prod_002",
        "name": "Wireless Mouse",
        "price": 25.00,
        "category": "Accessories",
        "tags": ["office", "wireless"],
        "specs": {"dpi": "1600", "battery": "AA"},
        "stock": 50
    },
    {
        "_id": "prod_003",
        "name": "4K Monitor",
        "price": 300.00,
        "category": "Screens",
        "tags": ["office", "4k"],
        "specs": {"resolution": "3840x2160", "refresh_rate": "60Hz"},
        "stock": 0
    }
]
db.products.insert_many(products_data)
print(f"Inserted {len(products_data)} products.")

# 2. Insert Users (Demonstrating Embedded Address)
users_data = [
    {
        "_id": "user_101",
        "username": "ahmed_dev",
        "email": "ahmed@email.com",
        "address": {
            "street": "Cite 500 Logements",
            "city": "Mila",
            "zip": "43000"
        },
        "orders_count": 0
    },
    {
        "_id": "user_102",
        "username": "sara_ds",
        "email": "sara@email.com",
        "address": {
            "street": "Rue de la Liberte",
            "city": "Algiers",
            "zip": "16000"
        },
        "orders_count": 5
    }
]
db.users.insert_many(users_data)
print(f"Inserted {len(users_data)} users.")
```

Part 4: Implementing CRUD Queries

Task: Translate the logic from TD 4 into Python functions.

4.1 Read Operations (Find)

Create a function `find_expensive_computers()` that prints products in the category "Computers" with a price greater than 1000.

Hint:

```
def find_expensive_computers():
    print("\n--- Expensive Computers ---")
    # Remember to use quotes for operators like "$gt"
    query = { "category": "Computers", "price": { "$gt": 1000 } }

    results = db.products.find(query)

    for doc in results:
        pprint.pprint(doc)

# Call the function
find_expensive_computers()
```

4.2 Logical Operators

Create a function `find_mixed_products()` to find products that are EITHER in category "Accessories" OR have a price lower than 50.

4.3 Update Operations

Ahmed (user_101) has moved. Write a script to update his city to "Constantine" and his zip code to "25000".

- Use the `$set` operator.
- Print the user document *after* the update to verify.

4.4 Array Manipulations

The "Gaming Laptop X1" is now on sale.

1. Add a new tag "sale" to its `tags` array (use `$addToSet`).
2. Decrease its price by 100 (use `$inc`).

Part 5: Advanced - The Order Transaction

This task simulates placing an order. You must:

1. Create a new order document in the `orders` collection.
2. Use the **Snapshot Pattern**: Copy the product's *current* name and price into the order.
3. **Reference** the user who bought it.

```
# Structure to implement:
new_order = {
    "order_date": datetime.datetime.now(),
    "user_id": "user_101", # Reference
    "items": [
        {
            "product_id": "prod_001",
            "name": "Gaming Laptop X1", # Snapshot
            "price": 1500.00,           # Snapshot
            "qty": 1
        }
    ],
    "total": 1500.00
}

# TODO: Write the code to insert this into db.orders
# Then print the inserted ID
```

Part 6: Aggregation

Write a Python script that counts how many products exist for each **category** using the Aggregation Framework.

Hint:

```
pipeline = [
    { "$group": { "_id": "$category", "count": { "$sum": 1 } } }
]
# Use db.products.aggregate(pipeline)
```