Database II
Semester 2025-I
Class Exercise Guide — Multi-Database Integration

Eng. Carlos Andrés Sierra, M.Sc.
Computer Engineering
Universidad Distrital Francisco José de Caldas

Welcome to the practical guide for *Database II*. This session focuses on integrating relational and NoSQL databases, applying SOLID principles, and exposing CRUD operations through a modern API.

Exercise Steps:

1. **Environment Validation:**
   - Verify that both PostgreSQL and MongoDB are installed on your machine.
   - Validate credentials for connecting to each database (test with `psql` and `mongosh`).
2. **PostgreSQL Schema Creation:**
   - Create a user named `test_user` with password.

     ```
     CREATE USER test_user WITH PASSWORD 'P4$$w0rd';
     ```

   - Create a PostgreSQL database named `students_db`.

     ```
     CREATE DATABASE students_db;
     ```

   - Login to the database as `test_user`.

     ```
     psql -h localhost -U test_user -d students_db
     ```

   - Create a table `career` with fields: `code` (PK), `name`, `credits`.

Carlos Andrés Sierra, Computer Engineer, M.Sc. in Computer Engineering, Titular Professor at Universidad Distrital Francisco José de Caldas.

Any comment or concern regarding this exercise can be sent to Carlos A. Sierra at: *cavirguezs@udistrital.edu.co*.

- Create a table `students` with fields: `code` (PK), `name`, `address`, `phone`, `career` (FK to `career.code`).

3. **MongoDB Collection:**

- Create a MongoDB user named `test_mongo`.

```
use admin
db.createUser({
    user: "test_mongo",
    pwd: "P4$$w0rd",
    roles: [{ role: "readWrite", db: "enrollments_db" }]
});
```

- Create a collection named `enrollments`.

```
use enrollments_db;
db.createCollection("enrollments");
db.enrollments.find({}); // Check if the collection is created
```

- Login to the database as `test_mongo`.

```
mongosh -u test_mongo
```

4. **Database Interface Design (SOLID):**

- Design a Python interface `DBConnection` that defines the basic CRUD operations and connection methods.
- Implement two classes, `MongoConnection` and `PostgresConnection`, that inherit from `DBConnection` and provide concrete implementations for MongoDB and PostgreSQL, respectively.
- Ensure your design follows SOLID principles, especially the Open/Closed and Liskov Substitution principles.
- *Example interface:*

```
from abc import ABC, abstractmethod

class DBConnection(ABC):
    @abstractmethod
    def connect(self):
        pass

    @abstractmethod
    def close(self):
        pass
```

```
@abstractmethod
def create(self, data):
    pass

@abstractmethod
def read(self, query):
    pass

@abstractmethod
def update(self, query, data):
    pass

@abstractmethod
def delete(self, query):
    pass
```

- Both `MongoConnection` and `PostgresConnection` must inherit from `DBConnection` and implement all abstract methods.

5. **API Development with FastAPI:**
   - Build a FastAPI application exposing endpoints to add, update, delete, and retrieve information from both databases.
   - **Logging Requirement:** Before processing any API request, log the request details (endpoint, method, parameters, timestamp) to a log file or logging system. This log should record every API call before the request is sent to the specific database.
   - Endpoints should interact with both databases and combine responses as needed.
   - The specific information to retrieve (fields, filters, combined queries, etc.) should be proposed and justified by the students, encouraging analysis and design decisions.
6. **Testing:**
   - Use Postman or SwaggerUI to test all endpoints.
   - Observe and document how each database is affected by the operations.

Notes:

- All code and documentation must be in **English**.
- Follow best practices for security (never hardcode credentials in code).
- Cite any references or libraries used.

This exercise will help you gain hands-on experience in integrating heterogeneous databases, designing robust interfaces, and exposing data through modern APIs. Good luck!