Database II
Semester 2025-III
Practice Guide — PostgreSQL Web Services with FastAPI

Eng. Carlos Andrés Sierra, M.Sc.
Full-time Adjunct Professor
Computer Engineering
School of Engineering
Universidad Distrital Francisco José de Caldas

Welcome to this practice for *Database II*. In this session, you will design and implement a set of web services using FastAPI and Python to interact with a PostgreSQL database. You will apply object-oriented programming principles, use environment variables for configuration, and expose endpoints for adding and extracting data.

Practice Steps:

1. **Database Schema Setup:**

   - Create the following tables in PostgreSQL:

```
CREATE TYPE continent_enum AS ENUM ('Africa', 'America', 'Asia',
'Europe', 'Oceania');

CREATE TABLE IF NOT EXISTS country (
    Code VARCHAR(3) PRIMARY KEY,
    Name VARCHAR(52),
    Continent continent_enum,
    Region VARCHAR(26),
    Population INT,
    LifeExpectancy FLOAT(3, 1),
    ...
);
```

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 TABLE IF NOT EXISTS city (
    id SERIAL PRIMARY KEY,
    name VARCHAR(30) NOT NULL,
    CountryCode CHAR(3) REFERENCES country(Code),
    District VARCHAR(20),
    population INT DEFAULT 0
);

CREATE TABLE IF NOT EXISTS countryLanguage (
    CountryCode CHAR(3) REFERENCES country(Code),
    Language VARCHAR(30),
    IsOfficial CHAR(1) CHECK (IsOfficial IN ('T', 'F')),
    Percentage FLOAT(4, 1),
    PRIMARY KEY (CountryCode, Language)
);
```

- Insert some default data into all tables (at least 10 countries, 2 cities per country, and 5 languages).

2. **Environment Configuration:**

   - Use the `python-dotenv` library and a `.env` file to store your database connection parameters (host, port, user, password, dbname).
   - Example `.env` file:

     ```
     DB_HOST=localhost
     DB_PORT=5432
     DB_USER=test_user
     DB_PASSWORD=P4$$w0rd
     DB_NAME=world_db
     ```

     – In PostgreSQL, as a superuser, run the following commands to create the database and user:

       ```
       CREATE DATABASE world_db;
       CREATE USER test_user WITH PASSWORD 'P4$$w0rd';
       GRANT ALL PRIVILEGES ON DATABASE world_db TO test_user;
       ```

     – Then connect to `world_db` as `test_user` to create tables and insert data.

3. **Object-Oriented Design:**

   - Define a Python interface `DBConnector` (using ABC) with methods for connecting, closing, and basic CRUD operations.
   - Implement a concrete class `PostgresConnector` that inherits from `DBConnector` and provides the actual PostgreSQL logic.

4. **FastAPI Web Services:**

   - Create a FastAPI application with endpoints to:

(a) Add a new country (including its language).
(b) Add a new city and relate it to a country.
(c) Extract all city names of Africa where the city has between 100,000 and 1,000,000 people and the language is Spanish or French.
(d) For the continent America, for every country, extract how many cities it has and the average population of the cities.
(e) For each continent, extract the name of the country with the highest life expectation, considering only countries with English as the official language.
(f) For each continent, list the top 3 most populous cities using a common table expression (CTE).
(g) For each language, show the total number of cities in countries where that language is official, using GROUP BY and a CTE.

- Ensure all endpoints use the connector object for database access.

5. **Testing and Documentation:**

- Use SwaggerUI (provided by `FastAPI`) to test your endpoints. Then use `Postman`.
- Document your code and provide a short README explaining how to run your API and how to set up the database.

Python Code Examples:
1. .env file example

```
DB_HOST=localhost
DB_PORT=5432
DB_USER=test_user
DB_PASSWORD=P4$$w0rd
DB_NAME=world_db
```

2. DBConnector Interface and Implementation

```
# db_connector.py
from abc import ABC, abstractmethod
import psycopg2
from dotenv import load_dotenv
import os

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

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

```
    @abstractmethod
    def execute(self, query, params=None):
        pass

class PostgresConnector(DBConnector):
    def __init__(self):
        load_dotenv()
        self.conn = None

    def connect(self):
        self.conn = psycopg2.connect(
            host=os.getenv("DB_HOST"),
            port=os.getenv("DB_PORT"),
            user=os.getenv("DB_USER"),
            password=os.getenv("DB_PASSWORD"),
            dbname=os.getenv("DB_NAME")
        )

    def close(self):
        if self.conn:
            self.conn.close()

    def execute(self, query, params=None):
        with self.conn.cursor() as cur:
            cur.execute(query, params)
            if query.strip().lower().startswith("select"):
                return cur.fetchall()
            self.conn.commit()
```

3. FastAPI Example Endpoints

```
# main.py
from fastapi import FastAPI
from db_connector import PostgresConnector

app = FastAPI()
db = PostgresConnector()
db.connect()

@app.post("/country/")
def add_country(code: str, name: str, continent: str, population: int, life_expectancy: flo
    db.execute("INSERT INTO country (Code, Name, Continent, Population, LifeExpectancy)
                VALUES (%s, %s, %s, %s, %s)",
               (code, name, continent, population, life_expectancy))
    db.execute("INSERT INTO countryLanguage (CountryCode, Language, IsOfficial)
```

```
                VALUES (%s, %s, %s)",
               (code, language, is_official))
    return {"message": "Country and language added"}

@app.post("/city/")
def add_city(name: str, country_code: str, population: int):
    db.execute("INSERT INTO city (name, CountryCode, population)
                VALUES (%s, %s, %s)", (name, country_code, population))
    return {"message": "City added"}

@app.get("/african_cities_spanish_french/")
def get_african_cities():
    query = """
        SELECT city.name
        FROM city
        JOIN country ON city.CountryCode = country.Code
        JOIN countryLanguage ON country.Code = countryLanguage.CountryCode
        WHERE country.Continent = 'Africa'
          AND city.population BETWEEN 100000 AND 1000000
          AND countryLanguage.Language IN ('Spanish', 'French')
    """
    result = db.execute(query)
    return [r[0] for r in result]

@app.get("/america_country_city_stats/")
def get_america_country_city_stats():
    query = """
        SELECT country.Name, COUNT(city.id), AVG(city.population)
        FROM country
        LEFT JOIN city ON country.Code = city.CountryCode
        WHERE country.Continent = 'America'
        GROUP BY country.Name
    """
    result = db.execute(query)
    return [{"country": r[0], "num_cities": r[1],
            "avg_population": r[2]} for r in result]

@app.get("/continent_highest_life_expectancy/")
def get_highest_life_expectancy():
    query = """
        SELECT country.Continent, country.Name
        FROM country
        JOIN countryLanguage ON country.Code = countryLanguage.CountryCode
        WHERE countryLanguage.Language = 'English' AND countryLanguage.IsOfficial = 'T'
        AND country.LifeExpectancy IS NOT NULL
```

```
        AND country.LifeExpectancy = (
            SELECT MAX(c2.LifeExpectancy)
            FROM country c2
            JOIN countryLanguage cl2 ON c2.Code = cl2.CountryCode
            WHERE cl2.Language = 'English' AND cl2.IsOfficial = 'T'
                AND c2.Continent = country.Continent
        )
    """
    result = db.execute(query)
    return [{"continent": r[0], "country": r[1]} for r in result]
```

Notes:

- All code and documentation must be in **English**.
- Never hardcode credentials in your code; always use environment variables.
- Use `Poetry` to list all dependencies (`fastapi`, `psycopg2`, `python-dotenv`, etc.).
- Test all endpoints using SwaggerUI or Postman.

This practice will help you gain hands-on experience in building RESTful APIs, using OOP for database access, and managing configuration securely. Good luck!