Project Phase 4

Submission Deadline: 20th November 2025, 11:59 PM

1. Introduction and Objectives

This final project phase marks the culmination of your design efforts. The primary objective is to translate the conceptual "Mini World" relational schema, developed in Phase 3, into a tangible, functional, and populated database system.

You will implement your database schema in MySQL, populate it with comprehensive data that brings your Mini World to life, and develop a Python-based application interface to interact with this data. This phase tests your proficiency in database creation, data management, and the practical application of SQL for data retrieval and manipulation.

Core Objectives:

1. **Database Implementation:** To correctly translate your final relational schema (from Phase 3) into a set of CREATE TABLE statements in MySQL, accurately enforcing all primary keys, foreign keys, constraints, and data types.
2. **Data Population:** To populate your database with a substantial and coherent set of legitimate data that reflects your chosen "Mini World" theme. This data must be sufficient to test all functional requirements meaningfully.
3. **Application Development:** To build a robust Python application that serves as an interface to your database.
4. **Functional Requirements:** To implement all functional requirements specified in the project (including the minimum of 5 distinct queries and 3 update operations) using raw SQL commands executed from your Python application.

2. Core Project Components

This project is divided into two primary parts: the database backend (SQL) and the application frontend (Python).

Part A: Database Implementation (SQL)

This part involves creating the database structure and filling it with data. This work should be encapsulated in two separate .sql files for organization and repeatability.

1. Schema Creation (schema.sql)

You must create one SQL script named schema.sql. When executed, this script should:

1. Create your database (e.g., CREATE DATABASE IF NOT EXISTS mini_world_db;).
2. Select your database (USE mini_world_db;).
3. Define and create all tables from your relational schema.
4. Pay meticulous attention to:

- **Data Types:** Select the most appropriate type for each attribute (e.g., INT, VARCHAR(255), TEXT, DATE, DECIMAL(10, 2), ENUM).
- **Primary Keys:** Define a primary key for every table (PRIMARY KEY).
- **Foreign Keys:** Correctly define all foreign key relationships (FOREIGN KEY ... REFERENCES ...).
- **Referential Integrity:** Specify ON DELETE and ON UPDATE actions (e.g., ON DELETE CASCADE, ON DELETE SET NULL, ON UPDATE CASCADE) as logically appropriate for your world's rules.
- **Constraints:** Enforce data integrity using NOT NULL, UNIQUE, and CHECK constraints where applicable.

2. Data Population (populate.sql)

You must create a second SQL script named populate.sql. This script will contain all the INSERT INTO statements necessary to populate your tables.

- **Data Quality:** Data must be realistic and coherent within the context of your Mini World. Avoid simple placeholder data (e.g., 'test1', 'test2').
- **Data Quantity:** You must insert enough data to demonstrate all your application's functional requirements effectively. This includes populating tables to allow for complex queries (e.g., data that will and will not match JOIN conditions) and having data that can be safely updated or deleted.
- **Insertion Order:** Be mindful of foreign key constraints. You must insert data into parent tables (e.g., Worlds) before inserting data into child tables that reference them (e.g., Characters).

Part B: Application Interface (Python)

This part involves building the application that users will interact with.

1. Database Connectivity

Your Python application must connect to your MySQL database using a library such as PyMySQL or mysql-connector-python. Hard-coding credentials directly in the main script is discouraged for production systems, but for this project, you may place them in a configuration function. All database operations (queries, updates) must be executed through this connection.

2. User Interface (UI)

- **Baseline Requirement:** The minimum requirement is a Command Line Interface (CLI). This can be built using simple input() and print() functions in a loop, presenting users with a menu of options that correspond to your functional requirements.
- **Optional Enhancements (No Extra Marks):** Teams may choose to implement a more sophisticated interface, such as a graphical user interface (GUI) with Tkinter or a web application with a micro-framework like Flask. While this can improve usability, it is **strictly optional** and will **not** factor into your grade. The evaluation is based on the database implementation, data quality, and the correctness of the SQL-backed functional requirements, regardless of the interface's complexity.

3. Functional Requirements Implementation

This is the most critical aspect of the application. Your interface must provide access to all functional requirements defined for your project.

- **Minimums:** You must implement *at least* five distinct query (read) operations and three distinct update (write) operations (INSERT, UPDATE, DELETE).
- **SQL Purity:** You **must** write raw SQL queries as strings within your Python code. You are **not** permitted to use an ORM (Object-Relational Mapper) or any high-level library functions that abstract away the SQL query (e.g., pandas.to_sql).

Parameterized Query Example (PyMySQL):

- **INCORRECT (Vulnerable):**

```
# DO NOT DO THIS
world_name = input("Enter world: ")
cursor.execute(f"SELECT * FROM Worlds WHERE world_name = '{world_name}'")
```

- **CORRECT (Secure):**

```
# DO THIS
world_name = input("Enter world: ")
sql_query = "SELECT * FROM Worlds WHERE world_name = %s"
cursor.execute(sql_query, (world_name,))
```

3. Boilerplate Code Examples (Mini World Theme)

Below are simplified boilerplate examples.

Example schema.sql (Structure)

```
DROP DATABASE IF EXISTS mini_world_db;
CREATE DATABASE mini_world_db;
USE mini_world_db;
CREATE TABLE Worlds (
    world_id INT AUTO_INCREMENT PRIMARY KEY,
    world_name VARCHAR(100) NOT NULL UNIQUE,
    description TEXT
    -- ... other attributes
);
-- ... etc.
```

Example main_app.py (Python CLI Application Structure)

```
import pymysql
import sys
from getpass import getpass

def get_db_connection(db_user, db_pass, db_host, db_name):
    """Establishes a connection to the MySQL database."""
    try:
        connection = pymysql.connect(
            host=db_host,
            user=db_user,
            password=db_pass,
            database=db_name,
            cursorclass=pymysql.cursors.DictCursor, # Returns results as
dictionaries
            autocommit=False # We will manually commit transactions
        )
        print("Database connection successful.")
        return connection
    except pymysql.Error as e:
        print(f"Error connecting to MySQL Database: {e}", file=sys.stderr)
        return None

def run_example_query(connection):
    """
    EXAMPLE READ FUNCTION: Demonstrates selecting data.
    """
    print("\n--- Running Example Query ---")
    try:
        with connection.cursor() as cursor:
            sql_query = "SELECT * FROM Worlds WHERE world_name = %s"
            search_name = input("Enter a world name to search for:
").strip()

            cursor.execute(sql_query, (search_name,))
            results = cursor.fetchall()

            if not results:
                print(f"No world found with name '{search_name}'.")
            else:
                print("Found:")
                for row in results:
                    print(f"  - ID: {row['world_id']}, Name:
{row['world_name']}")

    except pymysql.Error as e:
        print(f"Error during query: {e}", file=sys.stderr)

def main_cli(connection):
    """The main command-line interface loop."""
    try:
        while True:
            print("\n========== Mini World DB Interface ==========")
            print("1: Run Example Query (READ)")
            print("2: Run Example Update (CREATE)")
            # ... Add options for all your functional requirements ...
            # e.g., "3: Find all characters in a faction"
            # e.g., "4: Update a character's level"
            # e.g., "5: Delete an item"
            # ...
            print("q: Quit")
            print("===========================================")

            choice = input("Enter your choice: ").strip().lower()

            if choice == '1':
                run_example_query(connection)
            elif choice == '2':
                ...
            elif choice == 'q':
                print("Exiting application...")
                break
            else:
                print("Invalid choice. Please try again.")
    finally:
        if connection:
            connection.close()
            print("Database connection closed.")

if __name__ == "__main__":
    DB_HOST = 'localhost'
    DB_NAME = 'mini_world_db'

    print("Please enter your MySQL credentials.")
    DB_USER = input("Username: ").strip()
    DB_PASS = getpass("Password: ")

    db_conn = get_db_connection(DB_USER, DB_PASS, DB_HOST, DB_NAME)

    if db_conn:
        main_cli(db_conn)
    else:
        print("Failed to connect to the database. Application will exit.")
        sys.exit(1)
```

4. Submission Guidelines

Your final submission must be a single .zip file named <team_number>.zip. This file must contain the following items at its root level.

Required File Structure:

```
<team_number>.zip
├── <team_number>.mp4
├── phase3.pdf
├── README.md
└── src/
    ├── schema.sql
    ├── populate.sql
    ├── main_app.py
    └── (any other .py helper files or modules)
```

File-by-File Breakdown:

1. **<team_number>.mp4 (Max 5 Minutes):**
   - This is a screen recording demonstrating your project.
   - You **must** show two windows/terminals simultaneously or switch between them clearly.
     - **Terminal 1:** Running your Python application (python main_app.py).
     - **Terminal 2 (or DB GUI):** Logged into the mysql command-line client.
   - **Demonstration Flow for Updates:** For each update operation (INSERT, UPDATE, DELETE):
     1. In the mysql terminal, run a SELECT query to show the state of the data *before* the operation.
     2. In your Python app terminal, execute the command to perform the update.
     3. Return to the mysql terminal and re-run the *exact same* SELECT query to clearly demonstrate that the data has been changed correctly.
   - **Demonstration Flow for Queries:** Execute all your query functions from the Python app to show they retrieve the correct data.
2. **phase3.pdf:**
   - Your complete report from Phase 3.
3. **README.md:**
   - This is a critical file. It must contain a clear, numbered or bulleted list of all the commands/features your application can run.
   - Provide a small description for each command (e.g., "1. List all characters in a world - Prompts for a world name and shows all characters currently located there.").
   - This list **must** match the order of your video demonstration.
4. **src/ Directory:**
   - A directory named src containing all your source code.
   - **schema.sql:** Your final, executable schema creation script.
   - **populate.sql:** Your final, executable data population script.
   - **main_app.py:** Your main Python application file (or a similarly named entry point).
   - Include any other scripts or modules your main application depends on.

GOOD LUCK WITH PHASE 4!