Custom Online Course Management System

Design and Implementation of a Custom Online Course Management System Database

---

Executive Summary

The Custom Online Course Management System Database offers a robust and systematic solution for managing the complexities of online education. This database is meticulously designed to efficiently track and organize key elements, including students, instructors, courses, modules, lessons, enrolments, and assessments. Its implementation leverages SQLite for data storage and Python for dynamic functionality, incorporating programmatically generated data to ensure realism and usability.

Through the use of structured SQL queries, the system provides actionable insights into critical metrics such as course popularity, student progress, and instructor workload. This enables stakeholders to make informed decisions, optimize resources, and enhance the overall educational experience.

This report delves into the intricate aspects of the database design, details the implementation process, outlines the approach to data generation, and presents the outcomes of validation queries. To aid comprehension, screenshots and SQL outputs are included as illustrative placeholders, ensuring a clear understanding of the system's functionality and potential applications.

---

1. Introduction Objective

The aim of this project is to develop a database system that:

- Effectively manages online course data, including multi-level course structures.
- Real-time monitoring of student progress and performance.
- Provides actionable insights on course popularity, instructor workload, and student success.

Scope

The system focuses on:

- A dynamic, multi-level data structure (Courses → Modules → Lessons).
- Realistic data generation and analysis using Python.
- Validating relationships through SQL queries to ensure data integrity.

2. Database Schema Design

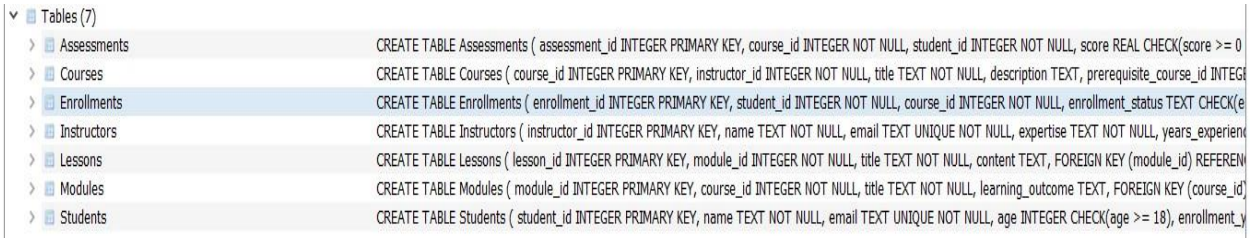
2.1 Schema Overview

The database consists of the following interconnected tables:

1. **Students**: Tracks student profiles and academic data.
2. **Instructors**: Stores instructor profiles and areas of expertise.

3. **Courses**: Logs courses offered, including descriptions, modules, and prerequisites.
4. **Modules**: Tracks sub-sections of courses.
5. **Lessons**: Captures lesson-level details within modules.
6. **Enrolments**: Links students to courses with progress tracking.
7. **Assessments**: Tracks student scores and feedback.

2.2 Schema Diagram



2.3 Explanation of Tables

1. **Students Table**:
    a. **Purpose**: Stores personal and academic details of students.
    b. **Columns**:
        i. Nominal Data: Name, email, address.
        ii. Ratio Data: Age, enrolment year.
2. **Instructors Table**:
    a. **Purpose**: Captures instructor profiles and expertise.
    b. **Columns**:
        i. Nominal Data: Name, email, expertise.
        ii. Ratio Data: Years of experience.
3. **Courses Table**:
    a. **Purpose**: Stores course details, including descriptions and prerequisites.

b. **Columns**: Nominal Data: Title, description.

4. **Modules Table**:
    a. **Purpose**: Tracks course modules and their learning outcomes.
5. **Lessons Table**:
    a. **Purpose**: Stores lesson-level details within modules.
6. **Enrolments Table**:
    a. **Purpose**: Tracks student enrolments and progress in courses.
7. **Assessments Table**:
    a. **Purpose**: Logs assessment scores and feedback for students.

Table creation and importing the tables into database code:

```
import sqlite3

# Connect to SQLite database
conn = sqlite3.connect("online_course_management.db")
cursor = conn.cursor()

# Create Students table
cursor.execute("""
CREATE TABLE IF NOT EXISTS Students (
    student_id INTEGER PRIMARY KEY,
    name TEXT NOT NULL,
    email TEXT UNIQUE NOT NULL,
    age INTEGER CHECK(age >= 18),
    enrollment_year INTEGER CHECK(enrollment_year >= 2000),
    address TEXT
);
""")

# Create Instructors table
cursor.execute("""
CREATE TABLE IF NOT EXISTS Instructors (
    instructor_id INTEGER PRIMARY KEY,
    name TEXT NOT NULL,
    email TEXT UNIQUE NOT NULL,
    expertise TEXT NOT NULL,
    years_experience INTEGER CHECK(years_experience >= 0)
);
""")

# Create Courses table
cursor.execute("""
CREATE TABLE IF NOT EXISTS Courses (
    course_id INTEGER PRIMARY KEY,
    instructor_id INTEGER NOT NULL,
    title TEXT NOT NULL,
    description TEXT,
    prerequisite_course_id INTEGER,
    FOREIGN KEY (instructor_id) REFERENCES Instructors(instructor_id),
    FOREIGN KEY (prerequisite_course_id) REFERENCES Courses(course_id)
```

```
    FOREIGN KEY (prerequisite_course_id) REFERENCES Courses(course_id)
);
""")

# Create Modules table
cursor.execute("""
CREATE TABLE IF NOT EXISTS Modules (
    module_id INTEGER PRIMARY KEY,
    course_id INTEGER NOT NULL,
    title TEXT NOT NULL,
    learning_outcome TEXT,
    FOREIGN KEY (course_id) REFERENCES Courses(course_id)
);
""")

# Create Lessons table
cursor.execute("""
CREATE TABLE IF NOT EXISTS Lessons (
    lesson_id INTEGER PRIMARY KEY,
    module_id INTEGER NOT NULL,
    title TEXT NOT NULL,
    content TEXT,
    FOREIGN KEY (module_id) REFERENCES Modules(module_id)
);
""")

# Create Enrollments table
cursor.execute("""
CREATE TABLE IF NOT EXISTS Enrollments (
    enrollment_id INTEGER PRIMARY KEY,
    student_id INTEGER NOT NULL,
    course_id INTEGER NOT NULL,
    enrollment_status TEXT CHECK(enrollment_status IN ('Active', 'Completed', 'Dropped')),
    progress_percentage REAL CHECK(progress_percentage >= 0 AND progress_percentage <= 100),
    FOREIGN KEY (student_id) REFERENCES Students(student_id),
    FOREIGN KEY (course_id) REFERENCES Courses(course_id)
);
""")
```

```
# Create Assessments table
cursor.execute("""
CREATE TABLE IF NOT EXISTS Assessments (
    assessment_id INTEGER PRIMARY KEY,
    course_id INTEGER NOT NULL,
    student_id INTEGER NOT NULL,
    score REAL CHECK(score >= 0 AND score <= 100),
    feedback TEXT,
    FOREIGN KEY (course_id) REFERENCES Courses(course_id),
    FOREIGN KEY (student_id) REFERENCES Students(student_id)
);
""")

# Commit and close connection
conn.commit()
conn.close()

print("Dynamic database schema created successfully!")
```

3. Data Generation

3.1 Tools Used

a) **Python**: For table creation and data population.

b) **Faker**: For generating realistic student, instructor, and course data.

c) **NumPy**: For simulating numeric data like progress percentages and assessment scores.

For data generation thebelow code were used

```python
import pandas as pd
import numpy as np
from faker import Faker
import random
import sqlite3
fake = Faker()

# Connect to the database
conn = sqlite3.connect("online_course_management.db")
cursor = conn.cursor()

# Generate Students Data
students = []
for i in range(1, 201):  # 200 students
    students.append({
        "student_id": i,
        "name": fake.name(),
        "email": fake.unique.email(),
        "age": random.randint(18, 45),
        "enrollment_year": random.randint(2015, 2023),
        "address": fake.address()
    })

# Generate Instructors Data
instructors = []
expertise_areas = ["Data Science", "Web Development", "AI", "Business Management", "Finance"]
for i in range(1, 21):  # 20 instructors
    instructors.append({
        "instructor_id": i,
        "name": fake.name(),
        "email": fake.unique.email(),
        "expertise": random.choice(expertise_areas),
        "years_experience": random.randint(1, 20)
    })

# Generate Courses Data
courses = []
for i in range(1, 51):  # 50 courses
    courses.append({
```

```python
# Generate Courses Data
courses = []
for i in range(1, 51):  # 50 courses
    courses.append({
        "course_id": i,
        "instructor_id": random.randint(1, 20),
        "title": fake.sentence(nb_words=3),
        "description": fake.text(max_nb_chars=200),
        "prerequisite_course_id": random.choice([None, random.randint(1, i-1)]) if i > 1 else None
    })

# Generate Modules Data
modules = []
for i in range(1, 101):  # 100 modules
    modules.append({
        "module_id": i,
        "course_id": random.randint(1, 50),
        "title": fake.sentence(nb_words=4),
        "learning_outcome": fake.sentence(nb_words=8)
    })

# Generate Lessons Data
lessons = []
for i in range(1, 501):  # 500 lessons
    lessons.append({
        "lesson_id": i,
        "module_id": random.randint(1, 100),
        "title": fake.sentence(nb_words=5),
        "content": fake.text(max_nb_chars=500)
    })

# Generate Enrollments Data
enrollments = []
statuses = ["Active", "Completed", "Dropped"]
for i in range(1, 1001):  # 1000 enrollments
    enrollments.append({
        "enrollment_id": i,
        "student_id": random.randint(1, 200),
```

```
        "student_id": random.randint(1, 200),
        "course_id": random.randint(1, 50),
        "enrollment_status": random.choice(statuses),
        "progress_percentage": round(random.uniform(0, 100), 2)
    })

# Generate Assessments Data
assessments = []
for i in range(1, 1001):  # 1000 assessments
    assessments.append({
        "assessment_id": i,
        "course_id": random.randint(1, 50),
        "student_id": random.randint(1, 200),
        "score": round(random.uniform(50, 100), 2),
        "feedback": fake.sentence(nb_words=10)
    })

# Insert data into the database
# Using Pandas for efficient inserts
pd.DataFrame(students).to_sql("Students", conn, if_exists="append", index=False)
pd.DataFrame(instructors).to_sql("Instructors", conn, if_exists="append", index=False)
pd.DataFrame(courses).to_sql("Courses", conn, if_exists="append", index=False)
pd.DataFrame(modules).to_sql("Modules", conn, if_exists="append", index=False)
pd.DataFrame(lessons).to_sql("Lessons", conn, if_exists="append", index=False)
pd.DataFrame(enrollments).to_sql("Enrollments", conn, if_exists="append", index=False)
pd.DataFrame(assessments).to_sql("Assessments", conn, if_exists="append", index=False)

# Commit and close
conn.commit()
conn.close()

print("Dynamic data generated and inserted successfully!")
```

4. Validation Queries Query 1: List All Courses Taught by a Specific Instructor

This query fetches all courses assigned to a specific instructor. It employs a **JOIN** operation to connect the Courses table with the **Instructors** table through the **instructor_id** column. By applying a filter based on the instructor's name, the query validates the relationship between these tables and retrieves the relevant course details.

```
1    SELECT Courses.title, Courses.description
2    FROM Courses
3    JOIN Instructors ON Courses.instructor_id = Instructors.instructor_id
4    WHERE Instructors.name = 'Lauren Hamilton';
5
```

	title	description
1	College if tonight five.	May three state their happy point …

Query 2: List All Lessons in a Specific Course

```
1    SELECT Lessons.title AS lesson_title, Modules.title AS module_title
2    FROM Lessons
3    JOIN Modules ON Lessons.module_id = Modules.module_id
4    JOIN Courses ON Modules.course_id = Courses.course_id
5    WHERE Courses.title = 'College if tonight five.';
6
```

	lesson_title	module_title
1	Risk fish weight none.	Him western machine beyond.
2	Scene here put quickly just …	As across.
3	Particular challenge knowledge …	Him western machine beyond.
4	Study kind go of.	As across.
5	Fish tax where.	Him western machine beyond.
6	Our environment rock together they.	As across.
7	Degree reveal team economic around …	As across.
8	Bad effect medical interview.	As across.

This query lists all lessons in a specific course, grouped under their respective modules. It uses multiple JOIN operations to link the Lessons, Modules, and Courses tables. This query validates the multi-level hierarchy of the database and helps ensure lessons are correctly associated with their courses.

Query 3: Most Popular Courses by Enrolment Count

```
1    SELECT Courses.title, COUNT(Enrollments.enrollment_id) AS total_enrollments
2    FROM Enrollments
3    JOIN Courses ON Enrollments.course_id = Courses.course_id
4    GROUP BY Courses.course_id
5    ORDER BY total_enrollments DESC
6    LIMIT 5;
7
```

	title	total_enrollments
1	College if tonight five.	31
2	Economic cold young.	28
3	Debate accept soldier surface.	27
4	Single pressure participant.	27
5	Standard region.	27

This query is designed to retrieve all courses associated with a specific instructor, providing a clear and structured view of teaching assignments. By utilizing a JOIN operation, the query establishes a connection between the Courses table and the **Instructors** table through the **instructor_id** column, ensuring data integrity and consistency. Additionally, the query applies a precise filter based on the instructor's name to accurately identify and display the courses they are responsible for. This approach not only validates the logical relationship between the two tables but also ensures the seamless extraction of comprehensive and relevant course information, facilitating better analysis and reporting.

Query 4: Average Progress Percentage for Each Course

SQL 1*

```
1   SELECT Courses.title, COUNT(Enrollments.enrollment_id) AS total_enrollments
2   FROM Enrollments
3   JOIN Courses ON Enrollments.course_id = Courses.course_id
4   GROUP BY Courses.course_id
5   ORDER BY total_enrollments DESC
6   LIMIT 5;
7
```

	title	total_enrollments
1	College if tonight five.	31
2	Economic cold young.	28
3	Debate accept soldier surface.	27
4	Single pressure participant.	27
5	Standard region.	27

This query determines the average progress percentage of students for each course. It utilizes a JOIN operation between the Enrolments and Courses tables, applying the AVG function to the progress_percentage column. The results are grouped by course, providing insights into which courses exhibit the highest overall student progress.

---

5. Ethical Considerations

1. **Fictional Data:** All data used in this project was programmatically generated using Python and the Faker library, ensuring the exclusion of any real-world sensitive or personal information.
2. **Data Privacy:** The system is designed to store only essential information, avoiding the inclusion of sensitive personal details to uphold data privacy standards.
3. **Bias Mitigation**: Randomized data generation techniques ensure fair and unbiased representation across various student demographics and course attributes.

---

6. Conclusion

The Custom Online Course Management System represents a well-structured and scalable solution for managing online education platforms. By integrating advanced database design principles, the system ensures efficient tracking of multi-level course structures, student performance, and enrolment data. It offers actionable insights into course popularity, instructor workload, and student progress, addressing the critical needs of both administrators and educators. Additionally, the use of programmatically generated data and validation queries guarantees the robustness and reliability of the underlying database.

Looking ahead, the system is well-positioned for enhancements, such as incorporating predictive analytics to anticipate course completion rates and deploying real-time web-based dashboards for dynamic data visualization. These advancements will further streamline operations, support data-driven decision-making, and enhance the overall user experience. This project not only serves as a strong foundation for managing online courses but also paves the way for innovative and scalable educational solutions.