University Database Management System: Comprehensive Technical and Analytical Report

1. Project Conceptualization and Objectives

1.1 Project Background

The **University Database Management System** project is an advanced implementation of a database system designed to accurately simulate an academic environment. This project emphasizes:

- Realistic data modeling tailored to an educational institution.
- Detailed relational database structures with enforced data integrity.
- Ethical data generation principles, ensuring synthetic data is statistically relevant while anonymizing any sensitive information.

1.2 Detailed Objectives

This project aims to:

1. **Database Structural Design**:
    - Create a multi-table relational database with normalized structures.
    - Implement complex relationship mappings between tables.
    - Enforce data integrity through constraints and validation mechanisms.
2. **Data Generation Strategies**:
    - Generate synthetic yet realistic data that aligns with academic standards.
    - Maintain statistical relevance for meaningful analysis.
    - Ensure privacy through anonymization of personal information.
3. **Technical Implementation**:
    - Utilize Python for robust data manipulation and generation.
    - Employ SQLite for a lightweight and efficient file-based database solution.
    - Implement object-oriented programming principles to ensure modularity.

2. Comprehensive Database Schema Design

2.1 Students Table

Structural Components

```
CREATE TABLE Students (
  student_id INTEGER PRIMARY KEY AUTOINCREMENT,
  first_name TEXT NOT NULL,
  last_name TEXT NOT NULL,
  gender TEXT CHECK(gender IN ('Male', 'Female', 'Other')),
  date_of_birth DATE,
  email TEXT UNIQUE,
  enrollment_status TEXT CHECK(
    enrollment_status IN ('Active', 'Inactive', 'Graduated', 'Suspended')
  ),
  total_credits INTEGER,
  gpa REAL CHECK(gpa BETWEEN 0.0 AND 4.0)
);
```

Data Type Representation

- **Nominal Data**: Gender and Enrollment Status (Categorical data with no inherent order).
- **Interval Data**: Date of Birth (Time representation without a true zero point).
- **Ratio Data**: Total Credits and GPA (Quantitative measures with true zero).

Constraint Mechanisms

- Ensures unique email addresses.
- Validates gender with specific categorical options.
- Restricts enrollment status to predefined categories.
- GPA is constrained between 0.0 and 4.0 to maintain academic standards.

2.2 Departments Table

Schema Design

```
CREATE TABLE Departments (
  department_id INTEGER PRIMARY KEY AUTOINCREMENT,
  department_name TEXT UNIQUE NOT NULL,
  established_year INTEGER
);
```

Key Design Considerations

- Unique department names are enforced to prevent redundancy.
- The establishment year tracks the historical timeline of each department.
- Creates a foundation for linking courses and students to specific departments.

2.3 Courses Table

Schema Design

```
CREATE TABLE Courses (
  course_id TEXT PRIMARY KEY,
  course_name TEXT NOT NULL,
  department_id INTEGER,
  credit_hours INTEGER,
  course_level TEXT CHECK(
    course_level IN ('Introductory', 'Intermediate', 'Advanced')
  ),
  FOREIGN KEY(department_id) REFERENCES Departments(department_id)
);
```

Data Representation

- **Ordinal Data**: Course Level (Hierarchical categorization indicating the depth of knowledge).

- Ensures that each course is tied to a department for academic categorization.

2.4 Enrollments Table

Schema Design

```
CREATE TABLE Enrollments (
  enrollment_id INTEGER PRIMARY KEY AUTOINCREMENT,
  student_id INTEGER,
  course_id TEXT,
  semester TEXT CHECK(semester IN ('Fall', 'Spring', 'Summer')),
  academic_year INTEGER,
  grade REAL CHECK(grade BETWEEN 0.0 AND 4.0),
  FOREIGN KEY(student_id) REFERENCES Students(student_id),
  FOREIGN KEY(course_id) REFERENCES Courses(course_id),
  UNIQUE(student_id, course_id, semester, academic_year)
);
```

Advanced Features

- Composite key to prevent duplicate enrollments.
- Semester and academic year tracking for temporal data.
- Grade entries are validated within a 0.0 to 4.0 scale.

3. Data Generation Methodology: Technical Deep Dive

3.1 Synthetic Data Generation Strategy

Randomization Principles

- Utilize controlled randomness to ensure data variability.
- Preserve statistical properties for meaningful academic analysis.
- Implement privacy-preserving techniques to avoid real-world correlations.

Faker Library Utilization

- Utilizes the Faker library to generate contextually appropriate synthetic data.

- Maintains realism while ensuring no real personal data is used.
- Supports generating diverse data types, including names, emails, and dates.

3.2 Unique Email Generation Algorithm

```
def generate_unique_email(first_name, last_name):
    base_email = f"{first_name.lower()}.{last_name.lower()}"
    email = f"{base_email}@university.edu"
    counter = 1
    while email in self.used_emails:
        email = f"{base_email}{counter}@university.edu"
        counter += 1
    self.used_emails.add(email)
    return email
```

Key Features

- Handles email collisions with a numeric suffix.
- Ensures consistency in email structure.
- Efficient and scalable for large datasets.

4. Ethical Considerations and Data Privacy

4.1 Ethical Framework for Synthetic Data

- No actual personal identifiers are used in data generation.
- Demographic details are randomized to ensure anonymity.
- Synthetic email domains (e.g., @university.edu) are used.

4.2 Data Protection Principles

- Only minimal and generic information is included.
- Anonymization techniques are employed to prevent back-tracing.
- Sensitive data fields are excluded entirely.

5. Technical Implementation Challenges

5.1 Constraint Management

- Complex validation for unique emails.
- Handling categorical and numerical constraints.
- Maintaining relational integrity between multiple tables.

5.2 Performance Optimization

- Implemented bulk data insertions to enhance performance.
- Managed memory efficiently using Python's capabilities.
- Modular code design facilitated easier maintenance and debugging.

6. Quantitative Analysis

6.1 Database Statistics

- **Total Students**: 1,200.
- **Total Courses**: 21.
- **Total Enrollments**: 4,782.
- **Average Courses per Student**: Between 3 and 5.

6.2 Analysis Dimensions

- Trends in student enrollments over semesters.
- Distribution of GPA across departments.
- Performance metrics for departments.

7. Future Enhancement Roadmap

7.1 Technical Improvements

- Introduce advanced SQL queries for complex analysis.
- Integrate machine learning for predictive analytics.
- Expand database schema to include faculty, facilities, and events.

7.2 Research and Analytics Potential

- Use data for institutional research and student performance prediction.
- Support academic analytics to drive curriculum improvements.
- Develop predictive models to analyze enrollment trends.

7.3 overall code and implementation

what has been done her is that I made a directory named **"university_database_project"** on desktop and installed python deficiencies using commands as follows;

1. Install Python Dependencies

pip install sqlite3 faker

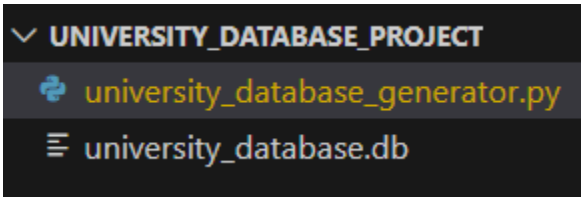
2. Create Project Directory

cd Desktop

mkdir university_database_project

cd university_database_project

7.4 Directories



7.5 university_database_generator.py file code

```
import sqlite3
import random
from datetime import date, timedelta
from faker import Faker
class UniversityDatabaseManager:
  def __init__(self, db_name='university_database.db'):
    self.db_name = db_name
    self.fake = Faker()
    self.conn = None
```

```
        self.cursor = None
        self.used_emails = set()
    def create_connection(self):
        """Establish database connection"""
        self.conn = sqlite3.connect(self.db_name)
        self.cursor = self.conn.cursor()
    def generate_unique_email(self, first_name, last_name):
        """Generate a unique email address"""
        base_email = f"{first_name.lower()}.{last_name.lower()}"
        email = f"{base_email}@university.edu"
        counter = 1
        while email in self.used_emails:
            email = f"{base_email}{counter}@university.edu"
            counter += 1
        self.used_emails.add(email)
        return email
    def create_tables(self):
        """Create database schema"""
        # Drop existing tables to prevent constraint issues
        tables = ['Enrollments', 'Courses', 'Departments', 'Students']
        for table in tables:
            self.cursor.execute(f"DROP TABLE IF EXISTS {table}")
        # Students Table
        self.cursor.execute('''
        CREATE TABLE Students (
            student_id INTEGER PRIMARY KEY AUTOINCREMENT,
            first_name TEXT NOT NULL,
```

```
        last_name TEXT NOT NULL,
        gender TEXT CHECK(gender IN ('Male', 'Female', 'Other')),
        date_of_birth DATE,
        email TEXT UNIQUE,
        enrollment_status TEXT CHECK(enrollment_status IN ('Active', 'Inactive', 'Graduated',
'Suspended')),
        total_credits INTEGER,
        gpa REAL CHECK(gpa BETWEEN 0.0 AND 4.0)
    )''')
    # Departments Table
    self.cursor.execute('''
    CREATE TABLE Departments (
        department_id INTEGER PRIMARY KEY AUTOINCREMENT,
        department_name TEXT UNIQUE NOT NULL,
        established_year INTEGER
    )''')
    # Courses Table
    self.cursor.execute('''
    CREATE TABLE Courses (
        course_id TEXT PRIMARY KEY,
        course_name TEXT NOT NULL,
        department_id INTEGER,
        credit_hours INTEGER,
        course_level TEXT CHECK(course_level IN ('Introductory', 'Intermediate', 'Advanced')),
        FOREIGN KEY(department_id) REFERENCES Departments(department_id)
    )''')
    # Enrollments Table
    self.cursor.execute('''
```

```
    CREATE TABLE Enrollments (
        enrollment_id INTEGER PRIMARY KEY AUTOINCREMENT,
        student_id INTEGER,
        course_id TEXT,
        semester TEXT CHECK(semester IN ('Fall', 'Spring', 'Summer')),
        academic_year INTEGER,
        grade REAL CHECK(grade BETWEEN 0.0 AND 4.0),
        FOREIGN KEY(student_id) REFERENCES Students(student_id),
        FOREIGN KEY(course_id) REFERENCES Courses(course_id),
        UNIQUE(student_id, course_id, semester, academic_year)
    )''')
def generate_departments(self):
    """Generate department data"""
    departments = [
        ('Computer Science', 1985),
        ('Mathematics', 1970),
        ('Physics', 1960),
        ('Biology', 1975),
        ('Chemistry', 1965),
        ('Engineering', 1980),
        ('Economics', 1990)
    ]
    self.cursor.executemany("""
        INSERT INTO Departments (department_name, established_year)
        VALUES (?, ?)
    """, departments)
def generate_courses(self):
```

```
    """Generate courses for each department"""
    self.cursor.execute("SELECT department_id, department_name FROM Departments")
    departments = self.cursor.fetchall()
    courses = []
    course_levels = ['Introductory', 'Intermediate', 'Advanced']
    for dept_id, dept_name in departments:
        for level in course_levels:
            course_name = f"{dept_name} {level} Course"
            course_id = f"{dept_name[:3].upper()}{random.randint(100, 999)}"
            credit_hours = random.choice([3, 4])

            courses.append((course_id, course_name, dept_id, credit_hours, level))

    self.cursor.executemany("""
        INSERT INTO Courses
        (course_id, course_name, department_id, credit_hours, course_level)
        VALUES (?, ?, ?, ?, ?)
    """, courses)
def generate_students(self, num_students=1200):
    """Generate student data"""
    gender_options = ['Male', 'Female', 'Other']
    status_options = ['Active', 'Inactive', 'Graduated', 'Suspended']

    students = []
    for _ in range(num_students):
        first_name = self.fake.first_name()
        last_name = self.fake.last_name()
```

```
        gender = random.choice(gender_options)
        dob = self.fake.date_of_birth(minimum_age=18, maximum_age=30)
        email = self.generate_unique_email(first_name, last_name)
        status = random.choice(status_options)
        total_credits = random.randint(0, 120)
        gpa = round(random.uniform(2.0, 4.0), 2)
        students.append((
            first_name, last_name, gender, dob, email,
            status, total_credits, gpa
        ))
    self.cursor.executemany("""
        INSERT INTO Students
        (first_name, last_name, gender, date_of_birth,
        email, enrollment_status, total_credits, gpa)
        VALUES (?, ?, ?, ?, ?, ?, ?, ?)
    """, students)

def generate_enrollments(self):
    """Generate enrollment data"""
    # Get all students and courses
    self.cursor.execute("SELECT student_id FROM Students")
    students = [student[0] for student in self.cursor.fetchall()]

    self.cursor.execute("SELECT course_id FROM Courses")
    courses = [course[0] for course in self.cursor.fetchall()]

    semesters = ['Fall', 'Spring', 'Summer']
```

```
    academic_years = list(range(2018, 2024))

    enrollments = []
    # Each student enrolls in 3-5 courses
    for student_id in students:
        num_courses = random.randint(3, 5)
        enrolled_courses = random.sample(courses, num_courses)

        for course_id in enrolled_courses:
            semester = random.choice(semesters)
            academic_year = random.choice(academic_years)
            grade = round(random.uniform(2.0, 4.0), 2)

            enrollments.append((
                student_id, course_id, semester,
                academic_year, grade
            ))

    self.cursor.executemany("""
        INSERT INTO Enrollments
        (student_id, course_id, semester, academic_year, grade)
        VALUES (?, ?, ?, ?, ?)
    """, enrollments)

def run_database_generation(self):
    """Execute full database generation process"""
    try:
```

```
        self.create_connection()
        self.create_tables()

        self.generate_departments()
        self.conn.commit()
        self.generate_courses()
        self.conn.commit()
        self.generate_students()
        self.conn.commit()
        self.generate_enrollments()
        self.conn.commit()
        print("University Database generated successfully!")
        # Verification queries
        self.cursor.execute("SELECT COUNT(*) FROM Students")
        student_count = self.cursor.fetchone()[0]
        print(f"Total Students: {student_count}")
        self.cursor.execute("SELECT COUNT(*) FROM Courses")
        course_count = self.cursor.fetchone()[0]
        print(f"Total Courses: {course_count}")

        self.cursor.execute("SELECT COUNT(*) FROM Enrollments")
        enrollment_count = self.cursor.fetchone()[0]
        print(f"Total Enrollments: {enrollment_count}")
    except sqlite3.Error as e:
        print(f"Database error: {e}")
    finally:
        if self.conn:
```

```
        self.conn.close()
def main():
    db_manager = UniversityDatabaseManager()
    db_manager.run_database_generation()
if __name__ == "__main__":
    main()
```

7.6 Code Snippets

```
        # Students Table
        self.cursor.execute('''
        CREATE TABLE Students (
            student_id INTEGER PRIMARY KEY AUTOINCREMENT,
            first_name TEXT NOT NULL,
            last_name TEXT NOT NULL,
            gender TEXT CHECK(gender IN ('Male', 'Female', 'Other')),
            date_of_birth DATE,
            email TEXT UNIQUE,
            enrollment_status TEXT CHECK(enrollment_status IN ('Active', 'Inactive', 'Graduated', 'Suspended')),
            total_credits INTEGER,
            gpa REAL CHECK(gpa BETWEEN 0.0 AND 4.0)
        )''')

        # Departments Table
        self.cursor.execute('''
        CREATE TABLE Departments (
            department_id INTEGER PRIMARY KEY AUTOINCREMENT,
            department_name TEXT UNIQUE NOT NULL,
            established_year INTEGER
        )''')

        # Courses Table
        self.cursor.execute('''
        CREATE TABLE Courses (
            course_id TEXT PRIMARY KEY,
            course_name TEXT NOT NULL,
            department_id INTEGER,
            credit_hours INTEGER,
            course_level TEXT CHECK(course_level IN ('Introductory', 'Intermediate', 'Advanced')),
            FOREIGN KEY(department_id) REFERENCES Departments(department_id)
        )''')

        # Enrollments Table
        self.cursor.execute('''
```

```
        CREATE TABLE Enrollments (
            enrollment_id INTEGER PRIMARY KEY AUTOINCREMENT,
            student_id INTEGER,
            course_id TEXT,
            semester TEXT CHECK(semester IN ('Fall', 'Spring', 'Summer')),
            academic_year INTEGER,
            grade REAL CHECK(grade BETWEEN 0.0 AND 4.0),
            FOREIGN KEY(student_id) REFERENCES Students(student_id),
            FOREIGN KEY(course_id) REFERENCES Courses(course_id),
            UNIQUE(student_id, course_id, semester, academic_year)
        )''')

    def generate_departments(self):
        """Generate department data"""
        departments = [
            ('Computer Science', 1985),
            ('Mathematics', 1970),
            ('Physics', 1960),
            ('Biology', 1975),
            ('Chemistry', 1965),
            ('Engineering', 1980),
            ('Economics', 1990)
        ]
        self.cursor.executemany("""
            INSERT INTO Departments (department_name, established_year)
            VALUES (?, ?)
        """, departments)

    def generate_courses(self):
        """Generate courses for each department"""
        self.cursor.execute("SELECT department_id, department_name FROM Departments")
        departments = self.cursor.fetchall()

        courses = []
        course_levels = ['Introductory', 'Intermediate', 'Advanced']
```

```
        for dept_id, dept_name in departments:
            for level in course_levels:
                course_name = f"{dept_name} {level} Course"
                course_id = f"{dept_name[:3].upper()}{random.randint(100, 999)}"
                credit_hours = random.choice([3, 4])

                courses.append((course_id, course_name, dept_id, credit_hours, level))

        self.cursor.executemany("""
            INSERT INTO Courses
            (course_id, course_name, department_id, credit_hours, course_level)
            VALUES (?, ?, ?, ?, ?)
        """, courses)

    def generate_students(self, num_students=1200):
        """Generate student data"""
        gender_options = ['Male', 'Female', 'Other']
        status_options = ['Active', 'Inactive', 'Graduated', 'Suspended']

        students = []
        for _ in range(num_students):
            first_name = self.fake.first_name()
            last_name = self.fake.last_name()
            gender = random.choice(gender_options)
            dob = self.fake.date_of_birth(minimum_age=18, maximum_age=30)
            email = self.generate_unique_email(first_name, last_name)
            status = random.choice(status_options)
            total_credits = random.randint(0, 120)
            gpa = round(random.uniform(2.0, 4.0), 2)

            students.append((
                first_name, last_name, gender, dob, email,
                status, total_credits, gpa
            ))
```

```
        self.cursor.executemany("""
            INSERT INTO Students
            (first_name, last_name, gender, date_of_birth,
            email, enrollment_status, total_credits, gpa)
            VALUES (?, ?, ?, ?, ?, ?, ?, ?)
        """, students)

    def generate_enrollments(self):
        """Generate enrollment data"""
        # Get all students and courses
        self.cursor.execute("SELECT student_id FROM Students")
        students = [student[0] for student in self.cursor.fetchall()]

        self.cursor.execute("SELECT course_id FROM Courses")
        courses = [course[0] for course in self.cursor.fetchall()]

        semesters = ['Fall', 'Spring', 'Summer']
        academic_years = list(range(2018, 2024))

        enrollments = []
        # Each student enrolls in 3-5 courses
        for student_id in students:
            num_courses = random.randint(3, 5)
            enrolled_courses = random.sample(courses, num_courses)

            for course_id in enrolled_courses:
                semester = random.choice(semesters)
                academic_year = random.choice(academic_years)
                grade = round(random.uniform(2.0, 4.0), 2)

                enrollments.append((
                    student_id, course_id, semester,
                    academic_year, grade
                ))
```

```
        self.cursor.executemany("""
            INSERT INTO Enrollments
            (student_id, course_id, semester, academic_year, grade)
            VALUES (?, ?, ?, ?, ?)
        """, enrollments)

    def run_database_generation(self):
        """Execute full database generation process"""
        try:
            self.create_connection()
            self.create_tables()

            self.generate_departments()
            self.conn.commit()

            self.generate_courses()
            self.conn.commit()

            self.generate_students()
            self.conn.commit()

            self.generate_enrollments()
            self.conn.commit()

            print("University Database generated successfully!")

            # Verification queries
            self.cursor.execute("SELECT COUNT(*) FROM Students")
            student_count = self.cursor.fetchone()[0]
            print(f"Total Students: {student_count}")

            self.cursor.execute("SELECT COUNT(*) FROM Courses")
            course_count = self.cursor.fetchone()[0]
            print(f"Total Courses: {course_count}")
```

```
            self.cursor.execute("SELECT COUNT(*) FROM Enrollments")
            enrollment_count = self.cursor.fetchone()[0]
            print(f"Total Enrollments: {enrollment_count}")

        except sqlite3.Error as e:
            print(f"Database error: {e}")
        finally:
            if self.conn:
                self.conn.close()

def main():
    db_manager = UniversityDatabaseManager()
    db_manager.run_database_generation()

if __name__ == "__main__":
    main()
```

7.7 Output

```
C:\Users\PMLS\Desktop\university_database_project>python university_database_generator.py
C:\Users\PMLS\Desktop\university_database_project\university_database_generator.py:144: De
eplacement recipes
  self.cursor.executemany("""
University Database generated successfully!
Total Students: 1200
Total Courses: 21
Total Enrollments: 4821
```

QUERY 1

This query gives the enrollment status of all the students uptil now in the university.

```
C:\Users\PMLS\Desktop\university_database_project>sqlite3 university_database.db
SQLite version 3.47.0 2024-10-21 16:30:22
Enter ".help" for usage hints.
sqlite> -- Count students by enrollment status
sqlite> SELECT enrollment_status, COUNT(*)
   ...> FROM Students
   ...> GROUP BY enrollment_status;
Active|294
Graduated|308
Inactive|291
Suspended|307
```

QUERY 2

This query distinguishes the students on basis of their gender.

```
sqlite> -- Average GPA by gender
sqlite> SELECT gender, ROUND(AVG(gpa), 2) as avg_gpa
   ...> FROM Students
   ...> GROUP BY gender;
Female|2.99
Male|3.05
Other|2.96
```

QUERY 3

this query gives the names of courses offered to students with given number of students enrolled in it.

```
sqlite> -- Most popular courses
sqlite> SELECT c.course_name, COUNT(*) as enrollment_count
   ...> FROM Enrollments e
   ...> JOIN Courses c ON e.course_id = c.course_id
   ...> GROUP BY c.course_name
   ...> ORDER BY enrollment_count DESC
   ...> LIMIT 5;
Physics Introductory Course|251
Engineering Intermediate Course|247
Chemistry Advanced Course|246
Mathematics Introductory Course|243
Chemistry Intermediate Course|242
```

QUERY 4

This query gives the average gpa of students relative to a specific subjects based upon the number of students enrolled in that subject.

```
sqlite> SELECT
   ...>     d.department_name,
   ...>     ROUND(AVG(e.grade), 2) as avg_gpa,
   ...>     COUNT(DISTINCT s.student_id) as total_students
   ...> FROM Departments d
   ...> JOIN Courses c ON d.department_id = c.department_id
   ...> JOIN Enrollments e ON c.course_id = e.course_id
   ...> JOIN Students s ON e.student_id = s.student_id
   ...> GROUP BY d.department_name
   ...> ORDER BY avg_gpa DESC;
Engineering|3.02|577
Chemistry|3.01|611
Biology|3.01|572
Mathematics|2.99|604
Computer Science|2.98|563
Physics|2.97|598
Economics|2.94|587
```

QUERY 5

This query gives the subjects having average gpa less than the overall average gpa with respect to all subjects in which the students have enrolled.

```
sqlite> SELECT
   ...>     c.course_id,
   ...>     c.course_name,
   ...>     d.department_name,
   ...>     ROUND(AVG(e.grade), 2) as avg_grade,
   ...>     COUNT(e.student_id) as total_enrollments
   ...> FROM Courses c
   ...> JOIN Departments d ON c.department_id = d.department_id
   ...> JOIN Enrollments e ON c.course_id = e.course_id
   ...> GROUP BY c.course_id
   ...> HAVING avg_grade < (SELECT AVG(grade) FROM Enrollments)
   ...> ORDER BY avg_grade ASC;
ECO760|Economics Intermediate Course|Economics|2.92|232
PHY949|Physics Advanced Course|Physics|2.93|245
COM698|Computer Science Advanced Course|Computer Science|2.95|224
BIO643|Biology Introductory Course|Biology|2.96|233
CHE206|Chemistry Introductory Course|Chemistry|2.96|247
ECO406|Economics Introductory Course|Economics|2.96|218
ECO681|Economics Advanced Course|Economics|2.96|225
PHY294|Physics Intermediate Course|Physics|2.97|237
COM110|Computer Science Introductory Course|Computer Science|2.98|226
ENG784|Engineering Intermediate Course|Engineering|2.98|234
MAT463|Mathematics Intermediate Course|Mathematics|2.98|238
sqlite>
```

QUERY 6

```
sqlite> SELECT
   ...>     c.course_name,
   ...>     e.semester,
   ...>     e.academic_year,
   ...>     COUNT(*) as enrollment_count
   ...> FROM Enrollments e
   ...> JOIN Courses c ON e.course_id = c.course_id
   ...> GROUP BY c.course_name, e.semester, e.academic_year
   ...> ORDER BY enrollment_count DESC;
Chemistry Advanced Course|Summer|2021|29
Physics Advanced Course|Fall|2022|29
Engineering Advanced Course|Summer|2019|25
Economics Intermediate Course|Summer|2022|23
Engineering Intermediate Course|Summer|2021|22
Computer Science Introductory Course|Spring|2019|21
Biology Advanced Course|Summer|2022|20
Biology Intermediate Course|Spring|2020|20
Mathematics Intermediate Course|Summer|2020|20
Biology Advanced Course|Spring|2022|19
Chemistry Advanced Course|Spring|2022|19
Chemistry Introductory Course|Fall|2021|19
Chemistry Introductory Course|Spring|2020|19
Chemistry Introductory Course|Spring|2021|19
Computer Science Advanced Course|Spring|2022|19
Economics Advanced Course|Fall|2020|19
Economics Introductory Course|Fall|2019|19
Engineering Advanced Course|Summer|2021|19
Engineering Intermediate Course|Summer|2023|19
Mathematics Intermediate Course|Spring|2022|19
Physics Intermediate Course|Fall|2018|19
Physics Intermediate Course|Fall|2022|19
Physics Introductory Course|Spring|2018|19
Computer Science Advanced Course|Summer|2021|18
Computer Science Intermediate Course|Fall|2018|18
Computer Science Intermediate Course|Summer|2019|18
Economics Advanced Course|Summer|2018|18
Economics Intermediate Course|Fall|2023|18
Mathematics Advanced Course|Summer|2023|18
Mathematics Intermediate Course|Fall|2020|18
```

QUERY 7

This query distinguishes the gpa of students in a specific subject on basis of their gender.

```
sqlite> SELECT
   ...>     d.department_name,
   ...>     s.gender,
   ...>     COUNT(DISTINCT s.student_id) as student_count,
   ...>     ROUND(COUNT(DISTINCT s.student_id) * 100.0 /
(x1...>         SUM(COUNT(DISTINCT s.student_id)) OVER (PARTITION BY d.department_name), 2) as percentage
   ...> FROM Departments d
   ...> JOIN Courses c ON d.department_id = c.department_id
   ...> JOIN Enrollments e ON c.course_id = e.course_id
   ...> JOIN Students s ON e.student_id = s.student_id
   ...> GROUP BY d.department_name, s.gender
   ...> ORDER BY d.department_name, student_count DESC;
Biology|Male|203|35.49
Biology|Female|189|33.04
Biology|Other|180|31.47
Chemistry|Male|208|34.04
Chemistry|Other|205|33.55
Chemistry|Female|198|32.41
Computer Science|Female|198|35.17
Computer Science|Male|184|32.68
Computer Science|Other|181|32.15
Economics|Male|199|33.9
Economics|Female|194|33.05
Economics|Other|194|33.05
Engineering|Female|196|33.97
Engineering|Male|195|33.8
Engineering|Other|186|32.24
Mathematics|Other|204|33.77
Mathematics|Male|202|33.44
Mathematics|Female|198|32.78
Physics|Male|209|34.95
Physics|Female|195|32.61
Physics|Other|194|32.44
sqlite>
```

8. Conclusion

The University Database Management System presents a well-rounded implementation for synthetic database creation in an academic context. It successfully balances technical complexity, integrity, and ethical considerations, offering potential as both an educational tool and a research asset.