SQL ASSIGNMENT – COLLEGE DATABASE

Abstract:

The college database outlined in this report comprises seven tables: Students, Courses, Instructors, Departments, Enrollments, Grades, and Classrooms. Each table is meticulously designed to store specific information related to students, courses, instructors, departments, enrollments, grades, and classrooms. The schema utilises primary keys, foreign keys, and appropriate constraints to ensure data integrity and reduce redundancy.

Introduction:

In the modern era of educational institutions, managing vast amounts of student and course-related data efficiently is paramount. This necessitates the design of a robust database schema that not only organises data logically but also adheres to ethical standards regarding data privacy. In this report, we will outline the database schema for a college system, discuss the justification for separate tables, and address ethical considerations surrounding data privacy. Additionally, we will provide sample queries showcasing the versatility and utility of the database schema.

Database Generation:

Students table:

- student_id: Primary Key for uniquely identifying each student.
- name: Full name of the student.
- age: Age of the student.
- gender: Gender of the student.
- department_id: Foreign Key referencing the department the student belongs to.

	student_id	name	age	gender	department_id
	Filter	Filter	Fil...	Filter	Filter
1	1	Amber Gonzales	18	Male	5
2	2	Ethan Huffman	23	Male	4
3	3	Adam Johnston	25	Female	5
4	4	Ashley Kane	24	Female	5
5	5	Pamela Johnson	18	Male	4
6	6	Christine Jarvis	24	Female	3
7	7	Christopher Christian	19	Male	4
8	8	Katelyn Rodriguez	19	Female	4
9	9	Renee Lucero	20	Female	2
10	10	Kathryn Cunningham	25	Male	3

Courses Table:

- course_id: Primary Key for uniquely identifying each course.
- course_name: Name of the course.
- credits: Number of credits associated with the course.

	course_id	course_name	credits
	Filter	Filter	Filter
1	1	Scientist, physiological	4
2	2	Chiropodist	2
3	3	Higher education careers adviser	3
4	4	Building control surveyor	1
5	5	Contractor	4
6	6	Curator	3
7	7	Secondary school teacher	3
8	8	Development worker, international aid	1
9	9	Electronics engineer	1
10	10	Naval architect	1

Instructors Table:

- instructor_id: Primary Key for uniquely identifying each instructor.
- name: Full name of the instructor.
- department_id: Foreign Key referencing the department the instructor belongs to.

	instructor_id	name	department_id
	Filter	Filter	Filter
1	1	Margaret Coleman	2
2	2	Robert Williams	1
3	3	Michelle Harmon	5
4	4	Tammy Perez	5
5	5	Bradley Lester	5
6	6	Donald Henry	1
7	7	Teresa Clark	5
8	8	Virginia Moore	2
9	9	Jennifer Stephens	5
10	10	Brittany Coleman	3

Departments Table:

- department_id: Primary Key for uniquely identifying each department.
- department_name: Name of the department.

	department_id	department_name
	Filter	Filter
1	1	Computer Science
2	2	Mathematics
3	3	Physics
4	4	Biology
5	5	Chemistry

Enrollments Table:

- enrollment_id: Primary Key for uniquely identifying each enrollment.
- student_id: Foreign Key referencing the student enrolled.
- course_id: Foreign Key referencing the course enrolled.
- enrollment_date: Date when the enrollment was made.

	enrollment_id	student_id	course_id	enrollment_date
	Filter	Filter	Filter	Filter
1	1	1	3	2024-01-14
2	2	1	1	2024-02-25
3	3	2	7	2024-02-24
4	4	2	19	2024-02-17
5	5	2	7	2024-02-28
6	6	2	4	2024-01-10
7	7	3	14	2024-02-06
8	8	3	2	2024-01-28
9	9	3	20	2024-02-21
10	10	3	17	2024-02-26

Grades Table:

- grade_id: Primary Key for uniquely identifying each grade.
- enrollment_id: Foreign Key referencing the enrollment the grade is associated with.
- grade: Grade obtained by the student in the course.

	grade_id	enrollment_id	grade
	Filter	Filter	Filter
1	1	1	F
2	2	2	D
3	3	3	F
4	4	4	A
5	5	5	C
6	6	6	A
7	7	7	C
8	8	8	A
9	9	9	C
10	10	10	B

Classrooms Table:

- classroom_id: Primary Key for uniquely identifying each classroom.
- room_number: Number of the classroom.
- building_name: Name of the building where the classroom is located.

	classroom_id	room_number	building_name
	Filter	Filter	Filter
1	1	1392	Craigburgh
2	2	2252	New Ashley
3	3	67023	Janetshire
4	4	58395	Davisland
5	5	3610	Shepardton
6	6	00175	South Patrickshire
7	7	05669	Richardfurt
8	8	347	Johnville
9	9	7301	North Catherine
10	10	4320	East Kathrynmouth

Data Type Representation:

Nominal Data:
Nominal data represents categories without any inherent order or ranking. In our database, the gender column in the Students table is an example of nominal data.

Ordinal Data:
Ordinal data represents categories with a defined order or ranking. In our database, the credits column in the Courses table represents ordinal data.

Interval Data:
Interval data represents numerical values where the difference between values is meaningful, but there is no true zero point. In our database, the age column in the Students table represents interval data.

Ratio Data:
Ratio data represents numerical values where the difference between values is meaningful, and there is a true zero point. In our database, the enrollment_date column in the Enrollments table represents ratio data.

Code:

```
import sqlite3
from faker import Faker
import random
import os

# Create SQLite database
db_path = 'college_database.db'

#Remove the existing database file if it exists
if os.path.exists(db_path):
    os.remove(db_path)

conn = sqlite3.connect(db_path)
cursor = conn.cursor()

# Create tables
cursor.execute('''CREATE TABLE Students (
                    student_id INTEGER PRIMARY KEY,
                    name TEXT,
                    age INTEGER,
                    gender TEXT,
                    department_id INTEGER,
                    FOREIGN KEY(department_id) REFERENCES Departments(department_id)
                    )''')

cursor.execute('''CREATE TABLE Courses (
                    course_id INTEGER PRIMARY KEY,
                    course_name TEXT,
                    credits INTEGER
                    )''')

cursor.execute('''CREATE TABLE Instructors (
```

```
                    instructor_id INTEGER PRIMARY KEY,
                    name TEXT,
                    department_id INTEGER,
                    FOREIGN KEY(department_id) REFERENCES Departments(department_id)
                    )''')

cursor.execute('''CREATE TABLE Departments (
                    department_id INTEGER PRIMARY KEY,
                    department_name TEXT
                    )''')

cursor.execute('''CREATE TABLE Enrollments (
                    enrollment_id INTEGER PRIMARY KEY,
                    student_id INTEGER,
                    course_id INTEGER,
                    enrollment_date TEXT,
                    FOREIGN KEY(student_id) REFERENCES Students(student_id),
                    FOREIGN KEY(course_id) REFERENCES Courses(course_id)
                    )''')

cursor.execute('''CREATE TABLE Grades (
                    grade_id INTEGER PRIMARY KEY,
                    enrollment_id INTEGER,
                    grade TEXT,
                    FOREIGN KEY(enrollment_id) REFERENCES Enrollments(enrollment_id)
                    )''')

cursor.execute('''CREATE TABLE Classrooms (
                    classroom_id INTEGER PRIMARY KEY,
                    room_number TEXT,
                    building_name TEXT
                    )''')
```

```
# Generate data using Faker
fake = Faker()

# Generate departments
departments = ['Computer Science', 'Mathematics', 'Physics', 'Biology', 'Chemistry']
for department in departments:
    cursor.execute("INSERT INTO Departments (department_name) VALUES (?)", (department,))
conn.commit()

# Generate students
for _ in range(1000):
    name = fake.name()
    age = random.randint(18, 25)
    gender = random.choice(['Male', 'Female'])
    department_id = random.randint(1, len(departments))
    cursor.execute("INSERT INTO Students (name, age, gender, department_id) VALUES (?, ?, ?, ?)", (name, age, gender,
                                                                                                    department_id))
conn.commit()

# Generate courses
for _ in range(20):
    course_name = fake.job()
    credits = random.randint(1, 5)
    cursor.execute("INSERT INTO Courses (course_name, credits) VALUES (?, ?)", (course_name, credits))
conn.commit()

# Generate instructors
for _ in range(20):
    name = fake.name()
    department_id = random.randint(1, len(departments))
    cursor.execute("INSERT INTO Instructors (name, department_id) VALUES (?, ?)", (name, department_id))
conn.commit()
```

```python

# Generate enrollments
for student_id in range(1, 1001):
    for _ in range(random.randint(1, 5)):
        course_id = random.randint(1, 20)
        enrollment_date = fake.date_this_year()
        cursor.execute("INSERT INTO Enrollments (student_id, course_id, enrollment_date) VALUES (?, ?, ?)",
                       (student_id, course_id, enrollment_date))
conn.commit()

# Generate grades
for enrollment_id in range(1, 5001):
    grade = random.choice(['A', 'B', 'C', 'D', 'F'])
    cursor.execute("INSERT INTO Grades (enrollment_id, grade) VALUES (?, ?)", (enrollment_id, grade))
conn.commit()

# Generate classrooms
for _ in range(10):
    room_number = fake.building_number()
    building_name = fake.city()
    cursor.execute("INSERT INTO Classrooms (room_number, building_name) VALUES (?, ?)", (room_number, building_name))
conn.commit()

# Close the connection
conn.close()

print("Database and data generation completed successfully.")

```

Database Schema:

Name	Type	Schema
Classrooms		CREATE TABLE Classrooms (classroom_id INTEGER PRIMARY KEY, room_number TEXT, building_name TEXT)
classroom_id	INTEGER	"classroom_id" INTEGER
room_number	TEXT	"room_number" TEXT
building_name	TEXT	"building_name" TEXT
Courses		CREATE TABLE Courses (course_id INTEGER PRIMARY KEY, course_name TEXT, credits INTEGER)
course_id	INTEGER	"course_id" INTEGER
course_name	TEXT	"course_name" TEXT
credits	INTEGER	"credits" INTEGER
Departments		CREATE TABLE Departments (department_id INTEGER PRIMARY KEY, department_name TEXT)
department_id	INTEGER	"department_id" INTEGER
department_name	TEXT	"department_name" TEXT
Enrollments		CREATE TABLE Enrollments (enrollment_id INTEGER PRIMARY KEY, student_id INTEGER, course_id INTEGER, enrollment_date TEXT, FOREIGN KEY(student_id) REFERENCES Students(student_id), FOREIGN KEY(course_id) REFERENCES Courses(course_id))
enrollment_id	INTEGER	"enrollment_id" INTEGER
student_id	INTEGER	"student_id" INTEGER
course_id	INTEGER	"course_id" INTEGER
enrollment_date	TEXT	"enrollment_date" TEXT
Grades		CREATE TABLE Grades (grade_id INTEGER PRIMARY KEY, enrollment_id INTEGER, grade

Name	Type	Schema
		TEXT, FOREIGN KEY(enrollment_id) REFERENCES Enrollments(enrollment_id))
grade_id	INTEGER	"grade_id" INTEGER
enrollment_id	INTEGER	"enrollment_id" INTEGER
grade	TEXT	"grade" TEXT
Instructors		CREATE TABLE Instructors (instructor_id INTEGER PRIMARY KEY, name TEXT, department_id INTEGER, FOREIGN KEY(department_id) REFERENCES Departments(department_id))
instructor_id	INTEGER	"instructor_id" INTEGER
name	TEXT	"name" TEXT
department_id	INTEGER	"department_id" INTEGER
Students		CREATE TABLE Students (student_id INTEGER PRIMARY KEY, name TEXT, age INTEGER, gender TEXT, department_id INTEGER, FOREIGN KEY(department_id) REFERENCES Departments(department_id))
student_id	INTEGER	"student_id" INTEGER
name	TEXT	"name" TEXT
age	INTEGER	"age" INTEGER
gender	TEXT	"gender" TEXT
department_id	INTEGER	"department_id" INTEGER

Justification for Separate Tables:

- Separate tables help maintain data integrity by organising data into logical units. By storing student information in a separate Students table, we can ensure that each student's data is uniquely identified by their student_id, avoiding duplicate records.
- Normalising the database by splitting it into separate tables reduces redundancy. Instead of storing course information alongside student

information, we create a separate course table. This prevents repetition of course details for each student and allows for easy updates to course information without affecting student records.

- Separate tables facilitate efficient querying by allowing us to focus on specific aspects of the data without needing to scan unnecessary information. If we want to retrieve a list of students enrolled in a particular course, we can simply join the Students and Enrollments tables based on the common student_id, rather than scanning through all student records.

Ethical Considerations:

- It's crucial to safeguard sensitive student information, such as personal details and academic records, from unauthorised access or misuse. Access controls and encryption techniques should be implemented to prevent data breaches.
- Encrypting the enrollment_date column in the Enrollments table ensures that sensitive enrollment dates are protected from unauthorised viewing, even if the database is compromised.

Example Queries:

1. **Select all students and their enrolled courses:**

```
1  SELECT s.name AS student_name, c.course_name
2  FROM Students s
3  JOIN Enrollments e ON s.student_id = e.student_id
4  JOIN Courses c ON e.course_id = c.course_id;
5
```

	student_name	course_name
1	Amber Gonzales	Higher education careers adviser
2	Amber Gonzales	Scientist, physiological
3	Ethan Huffman	Secondary school teacher
4	Ethan Huffman	Social researcher
5	Ethan Huffman	Secondary school teacher
6	Ethan Huffman	Building control surveyor
7	Adam Johnston	Archivist
8	Adam Johnston	Chiropodist
9	Adam Johnston	Water quality scientist
10	Adam Johnston	Pilot, airline
11	Adam Johnston	Engineer, maintenance (IT)
12	Ashley Kane	Building control surveyor
13	Ashley Kane	Curator
14	Ashley Kane	Building control surveyor
15	Ashley Kane	Scientist, water quality
16	Ashley Kane	Scientist, water quality
17	Pamela Johnson	Social researcher
18	Pamela Johnson	Secondary school teacher
19	Pamela Johnson	Pilot, airline
20	Christine Jarvis	Electronics engineer
21	Christine Jarvis	Archivist
22	Christine Jarvis	Curator
23	Christopher Christian	Equality and diversity officer
24	Christopher Christian	Engineer, maintenance (IT)

2. **Find the average age of students in each department:**

```
1  SELECT d.department_name, AVG(s.age) AS avg_age
2  FROM Students s
3  JOIN Departments d ON s.department_id = d.department_id
4  GROUP BY d.department_name;
5
```

	department_name	avg_age
1	Biology	21.6344086021505
2	Chemistry	21.7015706806283
3	Computer Science	21.0975609756098
4	Mathematics	21.5570776255708
5	Physics	21.4673366834171

3. **List all courses with the number of students enrolled in each course.**

```
1  SELECT c.course_name, COUNT(e.student_id) AS num_students_enrolled
2  FROM Courses c
3  LEFT JOIN Enrollments e ON c.course_id = e.course_id
4  GROUP BY c.course_name;
5
```

	course_name	num_students_enrolled
1	Archivist	154
2	Building control surveyor	144
3	Chiropodist	156
4	Contractor	152
5	Curator	140
6	Development worker, international aid	153
7	Electronics engineer	165
8	Engineer, communications	147
9	Engineer, maintenance (IT)	158
10	Equality and diversity officer	160
11	Higher education careers adviser	144
12	Materials engineer	157
13	Naval architect	148
14	Pilot, airline	149
15	Scientist, physiological	145
16	Scientist, water quality	173
17	Secondary school teacher	146
18	Social researcher	131
19	Water quality scientist	151
20	Youth worker	143

Conclusion:

The creation of a robust database for a college system necessitates thoughtful planning, meticulous design, and adherence to ethical principles. By adopting a modular approach with separate tables, we ensure streamlined data management, efficient querying, and enhanced data privacy.

Name: Sindhu Kavya Alahari
Student ID: 22070331