Designing Database for Students Record Using SQLite Software.

Abstract:

This report offers a thorough examination of a database engineered for the management of student records. It provides detailed insights into the process of generating the database, elucidating the intricacies of its schema design. The schema is meticulously crafted to encompass a wide array of student information, ensuring comprehensive coverage.

Introduction:

The Python script creates a SQLite database named Student Record and defines tables for student, subjects, class, class sessions, practicals, and Examination. It populates these tables with sample data generated using randomization techniques, including student names, ages, GPAs, class names, subject names, examination dates, practical dates, and session details. The script inserts the generated data into the respective tables within the database and commits the changes. This setup enables efficient management of student records within an educational context, facilitating tasks such as tracking student progress, scheduling classes, conducting examinations, and organizing practical sessions. The structured database schema and sample data provided lay the groundwork for further development and testing of educational management systems.

Database Generation:

The Students database was created using Python code to generate random data. It comprises six primary tables: Students, Subjects, Classes, Class Sessions, Practicals, and Examinations, containing a total of 1000 rows distributed across 27 distinct columns. These columns are structured to accommodate data in different formats, including nominal, ordinal, interval, and ratio. The database includes missing values to simulate real-world scenarios where data cleanliness may not be guaranteed. Below, each table is elaborated upon.

Students Table: The structure of the Student table is designed to contain vital information for each student. The attributes include:

StudentID: This is a unique identifier for each student. It has a true zero point and is countable, meaning is a (Ratio, Primary Key)

Name: This attribute pertains to the name of the student. It is classified as nominal because it merely identifies students without any inherent order.

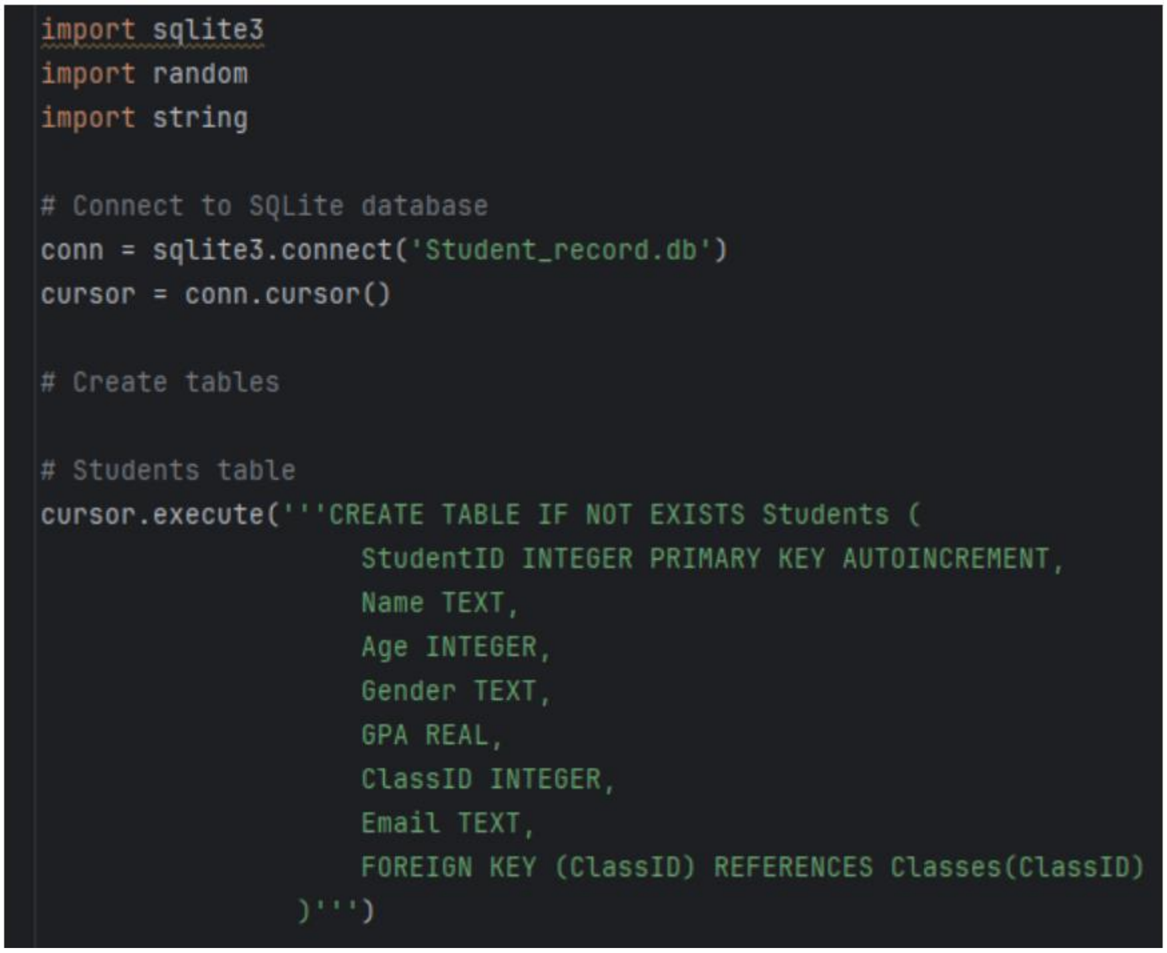
Age: It is quantified in whole numbers and holds a significant zero point (birth), hence categorized as a ratio variable.
Gender: Being a categorical variable, it lacks inherent order, thus classified as nominal.

GPA: Representing Grade Point Average, it is measured on a continuous scale with a meaningful zero point (0.0 GPA) and supports arithmetic operations, making it a Ratio variable.

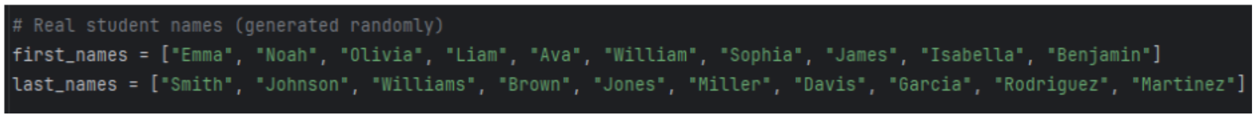
ClassID: This attribute denotes the student's class and shares characteristics with StudentID, ordering classes numerically. Therefore, it is a (Nominal, Foreign Key) variable.

Email: Email addresses serve as unique identifiers, and it falls under nominal data.

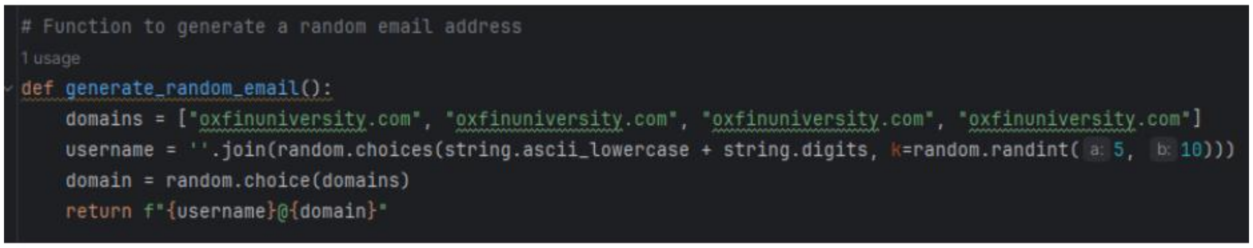
The code snippet and the image below illustrate the data generation process using Python and the dataset for the Students table in SQLite software.



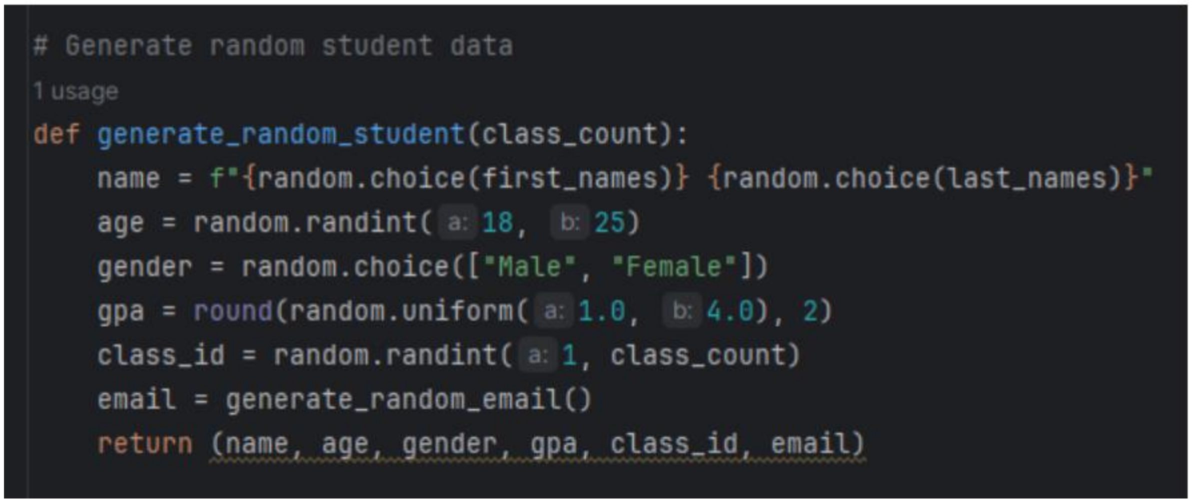
```
import sqlite3
import random
import string

# Connect to SQLite database
conn = sqlite3.connect('Student_record.db')
cursor = conn.cursor()

# Create tables

# Students table
cursor.execute('''CREATE TABLE IF NOT EXISTS Students (
                    StudentID INTEGER PRIMARY KEY AUTOINCREMENT,
                    Name TEXT,
                    Age INTEGER,
                    Gender TEXT,
                    GPA REAL,
                    ClassID INTEGER,
                    Email TEXT,
                    FOREIGN KEY (ClassID) REFERENCES Classes(ClassID)
                )''')
```



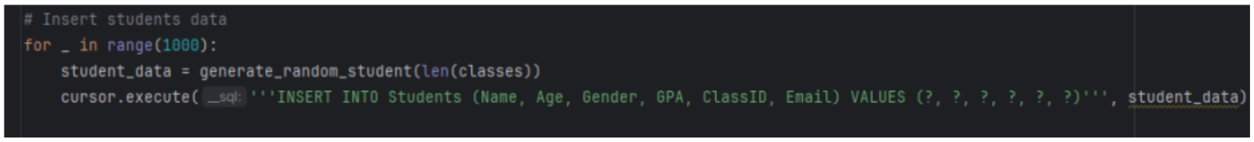
```
# Real student names (generated randomly)
first_names = ["Emma", "Noah", "Olivia", "Liam", "Ava", "William", "Sophia", "James", "Isabella", "Benjamin"]
last_names = ["Smith", "Johnson", "Williams", "Brown", "Jones", "Miller", "Davis", "Garcia", "Rodriguez", "Martinez"]
```



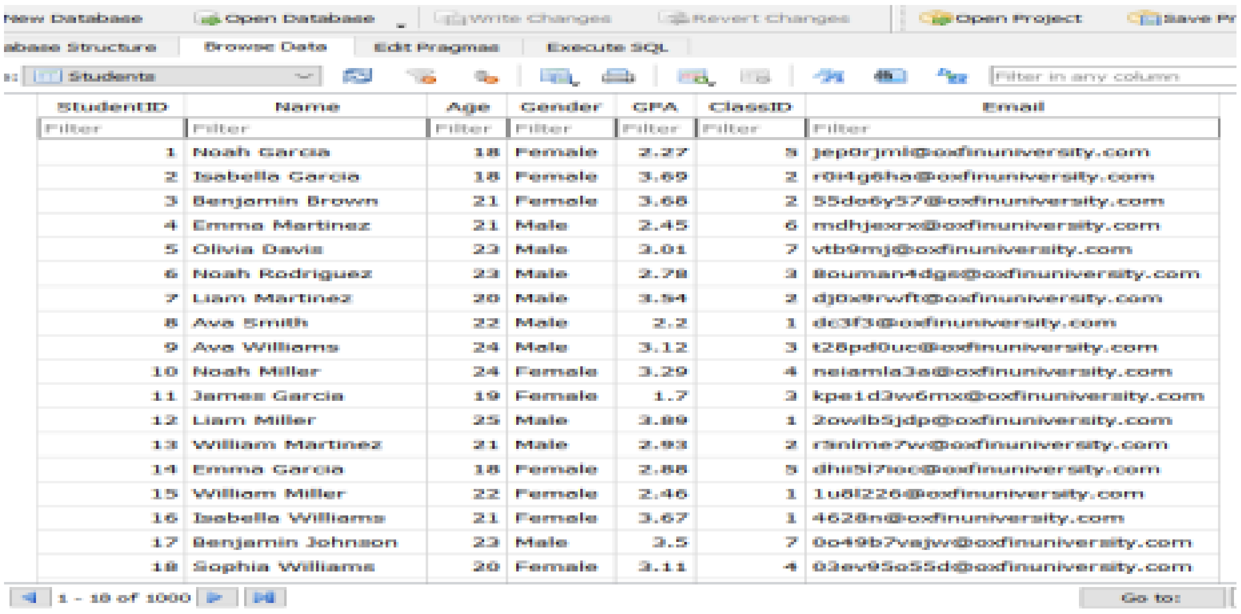
```
# Function to generate a random email address
1 usage
def generate_random_email():
    domains = ["oxfinuniversity.com", "oxfinuniversity.com", "oxfinuniversity.com", "oxfinuniversity.com"]
    username = ''.join(random.choices(string.ascii_lowercase + string.digits, k=random.randint( a: 5,  b: 10)))
    domain = random.choice(domains)
    return f"{username}@{domain}"
```



```
# Generate random student data
1 usage
def generate_random_student(class_count):
    name = f"{random.choice(first_names)} {random.choice(last_names)}"
    age = random.randint( a: 18,  b: 25)
    gender = random.choice(["Male", "Female"])
    gpa = round(random.uniform( a: 1.0,  b: 4.0), 2)
    class_id = random.randint( a: 1, class_count)
    email = generate_random_email()
    return (name, age, gender, gpa, class_id, email)
```



```
# Insert students data
for _ in range(1000):
    student_data = generate_random_student(len(classes))
    cursor.execute( _sql: '''INSERT INTO Students (Name, Age, Gender, GPA, ClassID, Email) VALUES (?, ?, ?, ?, ?, ?)''', student_data)
```



StudentID	Name	Age	Gender	GPA	ClassID	Email
1	Noah Garcia	18	Female	2.27	5	jep0rjmi@oxfinuniversity.com
2	Isabella Garcia	18	Female	3.69	2	r0i4g6ha@oxfinuniversity.com
3	Benjamin Brown	21	Female	3.68	2	55do6y57@oxfinuniversity.com
4	Emma Martinez	21	Male	2.45	6	mdhjexrx@oxfinuniversity.com
5	Olivia Davis	23	Male	3.01	7	vtb9mj@oxfinuniversity.com
6	Noah Rodriguez	23	Male	2.78	3	8ouman4dgs@oxfinuniversity.com
7	Liam Martinez	20	Male	3.54	2	dj0x9rwft@oxfinuniversity.com
8	Ava Smith	22	Male	2.2	1	dc3f3@oxfinuniversity.com
9	Ava Williams	24	Male	3.12	3	t28pd0uc@oxfinuniversity.com
10	Noah Miller	24	Female	3.29	4	neiamla3a@oxfinuniversity.com
11	James Garcia	19	Female	1.7	3	kpe1d3w6mx@oxfinuniversity.com
12	Liam Miller	25	Male	3.89	1	2owib5jdp@oxfinuniversity.com
13	William Martinez	21	Male	2.93	2	r5nime7w@oxfinuniversity.com
14	Emma Garcia	18	Female	2.88	5	dhi5l7ioc@oxfinuniversity.com
15	William Miller	22	Female	2.46	1	1u8l226@oxfinuniversity.com
16	Isabella Williams	21	Female	3.67	1	4628n@oxfinuniversity.com
17	Benjamin Johnson	23	Male	3.5	7	0o49b7vajw@oxfinuniversity.com
18	Sophia Williams	20	Female	3.11	4	03ev95o55d@oxfinuniversity.com

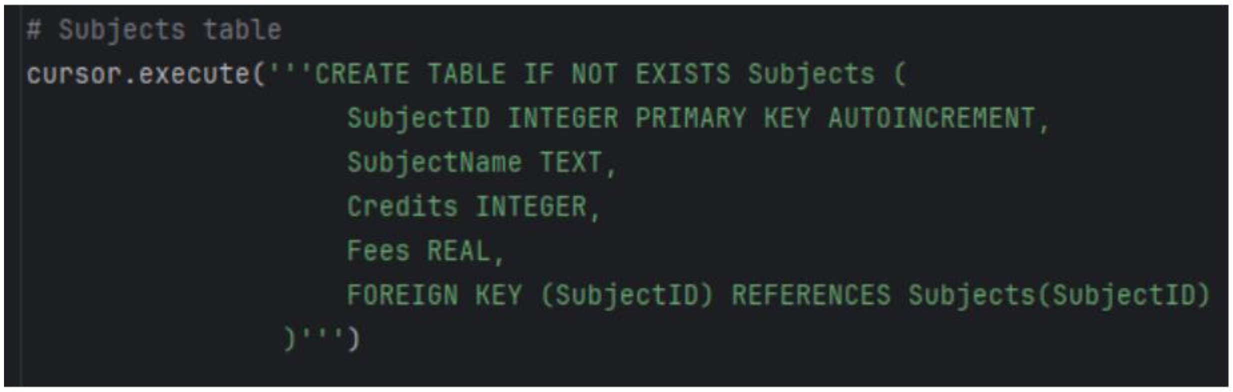
<u>**Subjects Table:**</u> The Subjects Table is structured to hold essential details for every student. Included attributes are:

SubjectID: This is Unique identifier for each subject. It is a (Ratio, Primary Key) it serves as a label or identifier for each subject.

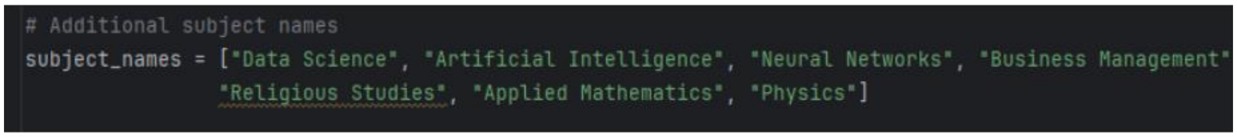
SubjectName: The is a nominal, representing the names of subjects. There's no inherent order. **Credits:** This is a ratio because they have a true zero point.

Fees: Fees are typically ratio data since they have a true zero point.

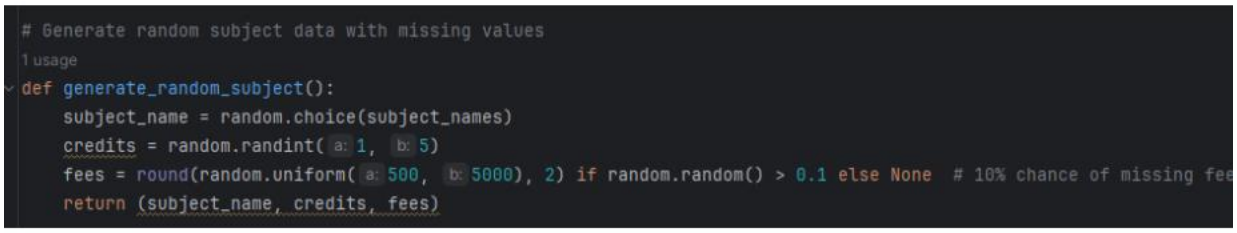
The provided code snippet, along with the accompanying image, illustrates the data generation process for the "subject" table utilizing Python and the dataset intended for the "subject" table in SQLite.



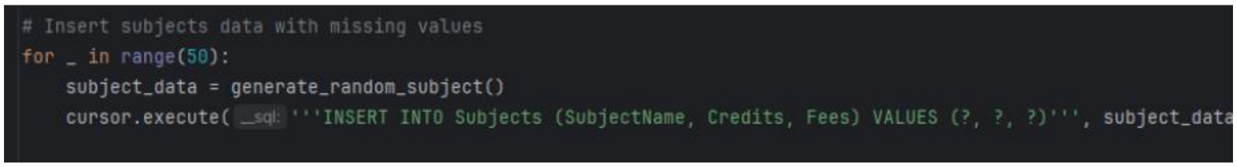
```
# Subjects table
cursor.execute('''CREATE TABLE IF NOT EXISTS Subjects (
                     SubjectID INTEGER PRIMARY KEY AUTOINCREMENT,
                     SubjectName TEXT,
                     Credits INTEGER,
                     Fees REAL,
                     FOREIGN KEY (SubjectID) REFERENCES Subjects(SubjectID)
                 )''')
```



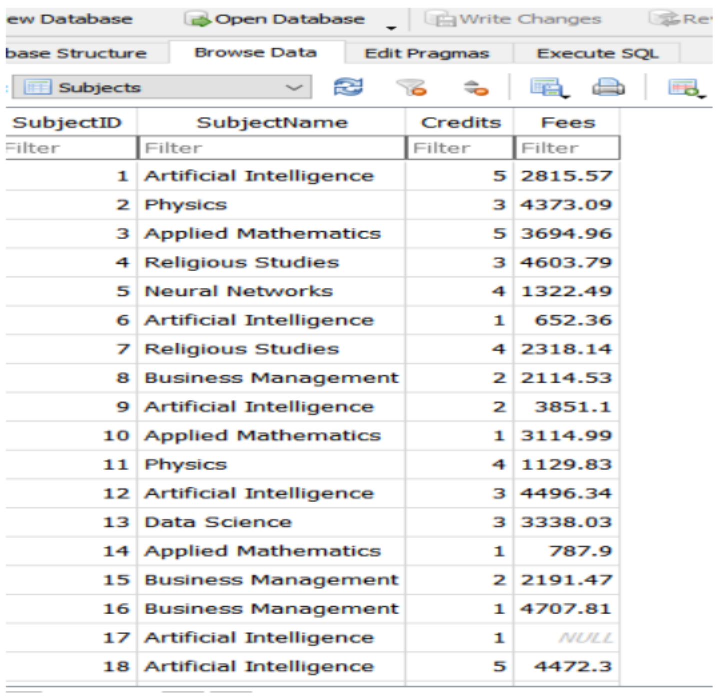
```
# Additional subject names
subject_names = ["Data Science", "Artificial Intelligence", "Neural Networks", "Business Management"
                 "Religious Studies", "Applied Mathematics", "Physics"]
```



```
# Generate random subject data with missing values
1 usage
def generate_random_subject():
    subject_name = random.choice(subject_names)
    credits = random.randint( a: 1,  b: 5)
    fees = round(random.uniform( a: 500,  b: 5000), 2) if random.random() > 0.1 else None  # 10% chance of missing fee
    return (subject_name, credits, fees)
```



```
# Insert subjects data with missing values
for _ in range(50):
    subject_data = generate_random_subject()
    cursor.execute( _sql: '''INSERT INTO Subjects (SubjectName, Credits, Fees) VALUES (?, ?, ?)''', subject_data
```



SubjectID	SubjectName	Credits	Fees
Filter	Filter	Filter	Filter
1	Artificial Intelligence	5	2815.57
2	Physics	3	4373.09
3	Applied Mathematics	5	3694.96
4	Religious Studies	3	4603.79
5	Neural Networks	4	1322.49
6	Artificial Intelligence	1	652.36
7	Religious Studies	4	2318.14
8	Business Management	2	2114.53
9	Artificial Intelligence	2	3851.1
10	Applied Mathematics	1	3114.99
11	Physics	4	1129.83
12	Artificial Intelligence	3	4496.34
13	Data Science	3	3338.03
14	Applied Mathematics	1	787.9
15	Business Management	2	2191.47
16	Business Management	1	4707.81
17	Artificial Intelligence	1	NULL
18	Artificial Intelligence	5	4472.3

Incorporating missing value, foreign keys, and compound keys into the database design accurately mirrors real world-scenarios.

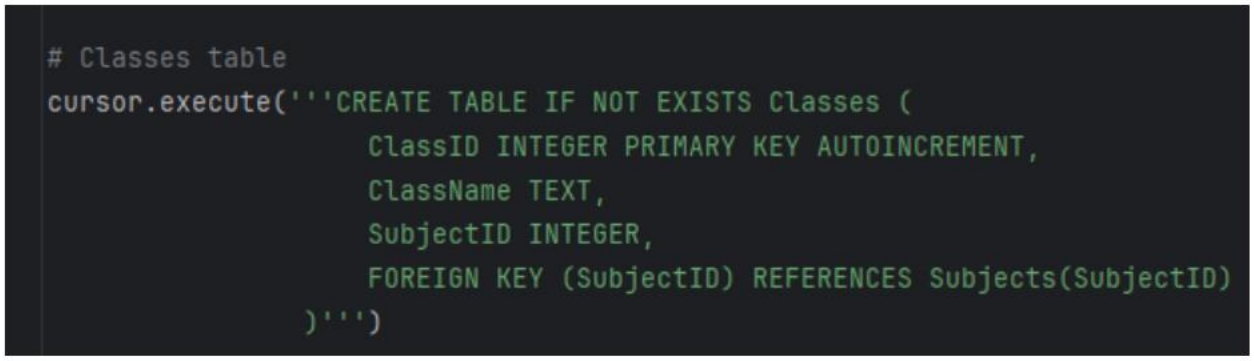
In an effort to faithfully replicate real-world scenarios, I purposely introduced missing values into subject table within the database. Additionally, I implemented primary keys, the latter of which act as references to other tables in the database. These keys were properly designated as indices within their corresponding tables. The above code snippet illustrates the incorporation of missing values and the establishment of primary keys within the tables.

Classes Table: The Classes table is designed to capture fundamental details pertaining to classes, facilitating the efficient tracking and analysis of class-related information.

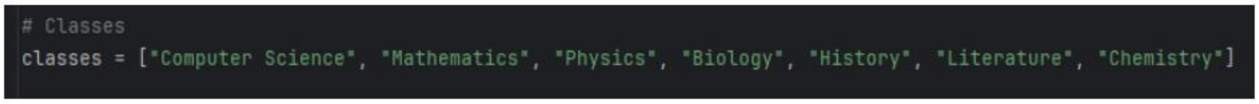
ClassID: This attribute serves as a unique identifier for each class. It has a zero point (no class can have an ID of 0) and possesses the properties of identity, magnitude, and equal intervals. Meaning is a (Ratio, Primary Key)

ClassName: This attribute represents the name or label of the class. It only provides a name or label without any inherent order or numerical significance. The data here are categories without any inherent numerical, meaning is a Nominal

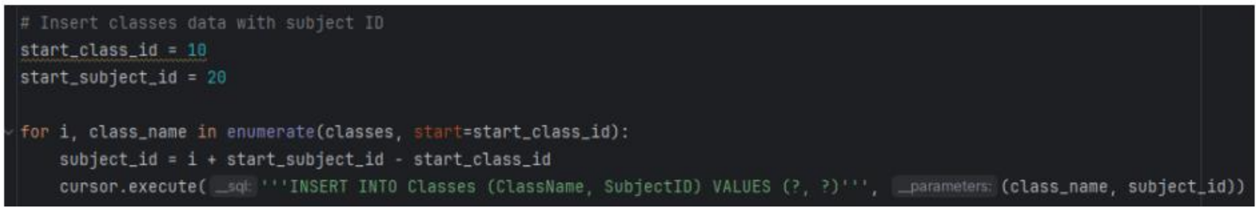
SubjectID: This attribute refers to the ID of the subject associated with the class. This is an (Ratio, Foreign Key) and it represents ordered categories.
The provided code snippet, in conjunction with the accompanying image, demonstrates the process of generating data for the class table using Python and the dataset designated for the class table in SQLite.



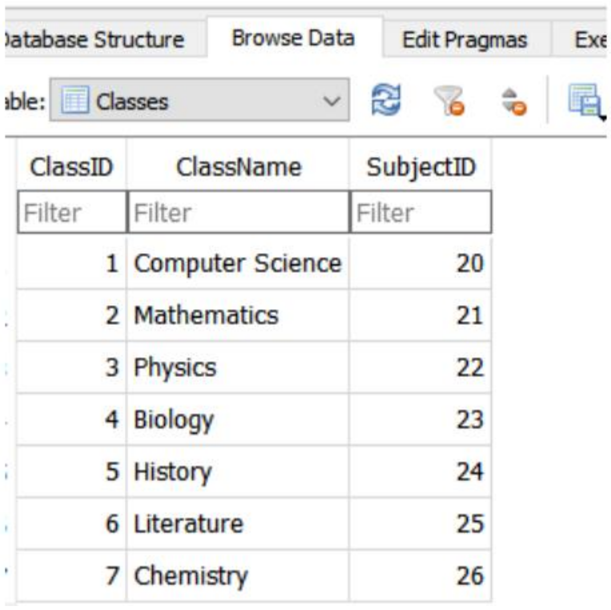
```
# Classes table
cursor.execute('''CREATE TABLE IF NOT EXISTS Classes (
                    ClassID INTEGER PRIMARY KEY AUTOINCREMENT,
                    ClassName TEXT,
                    SubjectID INTEGER,
                    FOREIGN KEY (SubjectID) REFERENCES Subjects(SubjectID)
                )''')
```



```
# Classes
classes = ["Computer Science", "Mathematics", "Physics", "Biology", "History", "Literature", "Chemistry"]
```



```
# Insert classes data with subject ID
start_class_id = 10
start_subject_id = 20

for i, class_name in enumerate(classes, start=start_class_id):
    subject_id = i + start_subject_id - start_class_id
    cursor.execute( _sql: '''INSERT INTO Classes (ClassName, SubjectID) VALUES (?, ?)''', _parameters: (class_name, subject_id))
```



ClassID	ClassName	SubjectID
Filter	Filter	Filter
1	Computer Science	20
2	Mathematics	21
3	Physics	22
4	Biology	23
5	History	24
6	Literature	25
7	Chemistry	26

ClassSesstions Table: The classsesstion table, crafted for the capture of foundational classsesstion particulars, is intended to streamline the tracking and analysis of class-related data.

SubjectID: serves as a unique identifier for each subject. (Ratio, Foreign Key)
SessionID: serves as a unique identifier for each session (Ratio, Primary Key)

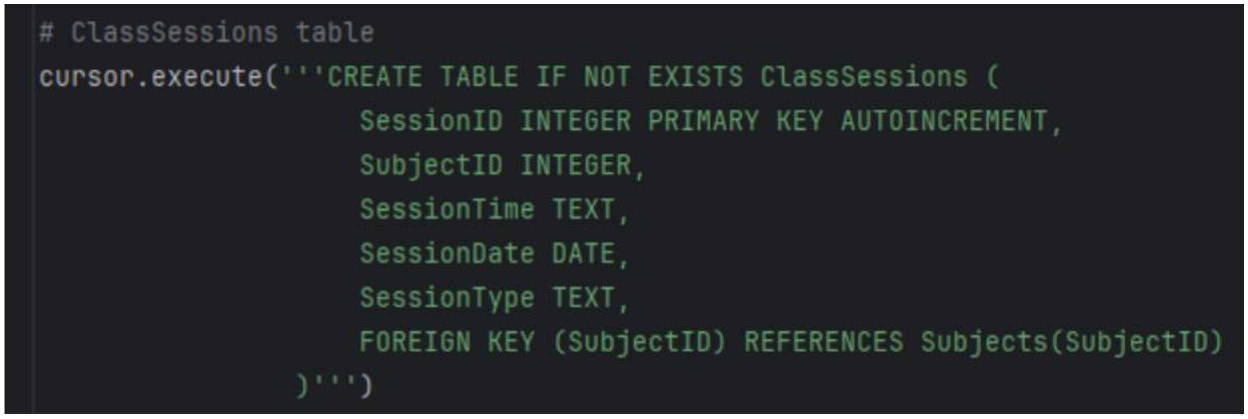
SessionType: This column represents different categories of session types such as lecture, lab and that is Nomial.

SesstionTime: This column represents time intervals (Interval)

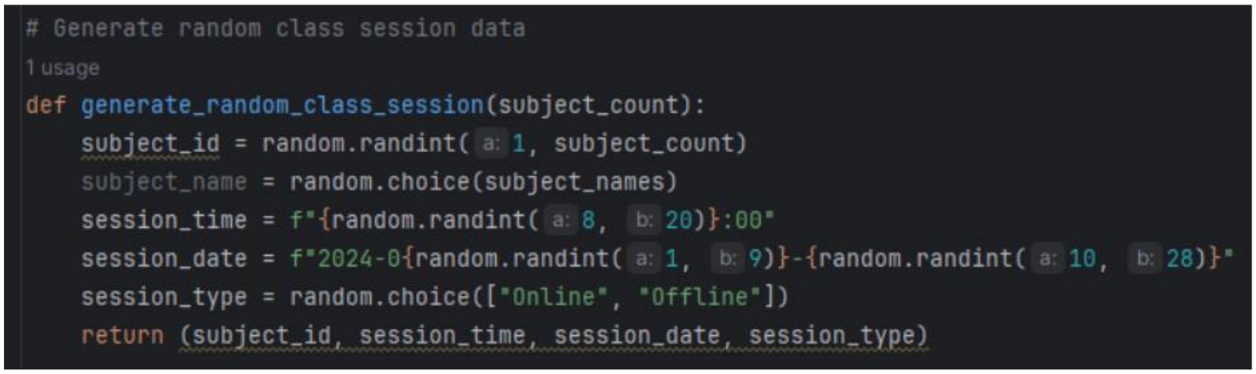
SessionDate: This column represents points in time and has a true zero point. (Ratio)

The code snippet provided, along with the accompanying image, showcases the process of generating data for the ClassSession table using Python and the dataset intended for the ClassSession table in

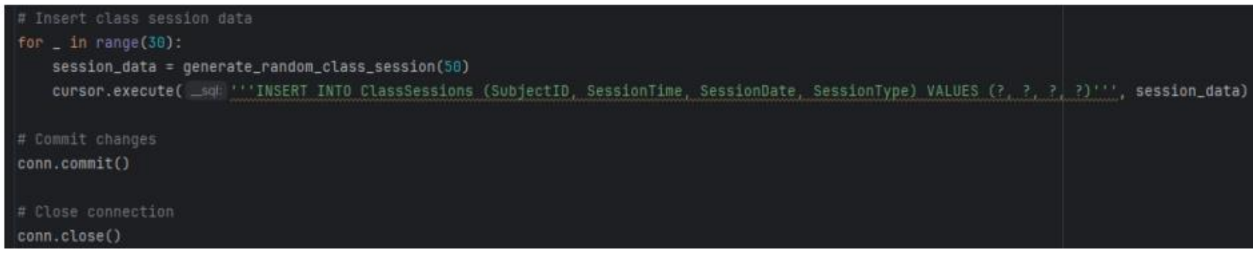
SQLite.



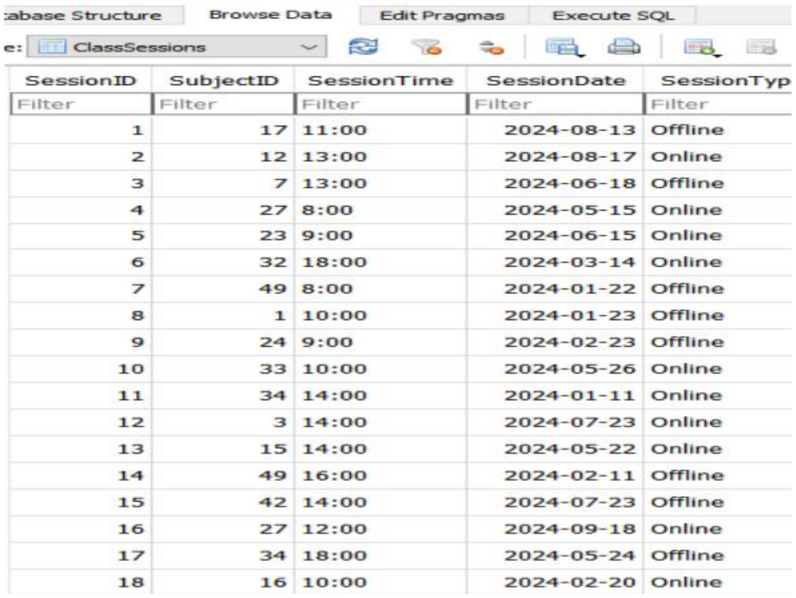
```
# ClassSessions table
cursor.execute('''CREATE TABLE IF NOT EXISTS ClassSessions (
                    SessionID INTEGER PRIMARY KEY AUTOINCREMENT,
                    SubjectID INTEGER,
                    SessionTime TEXT,
                    SessionDate DATE,
                    SessionType TEXT,
                    FOREIGN KEY (SubjectID) REFERENCES Subjects(SubjectID)
                )''')
```



```
# Generate random class session data
1 usage
def generate_random_class_session(subject_count):
    subject_id = random.randint( a: 1, subject_count)
    subject_name = random.choice(subject_names)
    session_time = f"{random.randint( a: 8,  b: 20)}:00"
    session_date = f"2024-0{random.randint( a: 1,  b: 9)}-{random.randint( a: 10,  b: 28)}"
    session_type = random.choice(["Online", "Offline"])
    return (subject_id, session_time, session_date, session_type)
```



```
# Insert class session data
for _ in range(30):
    session_data = generate_random_class_session(50)
    cursor.execute( _sql: '''INSERT INTO ClassSessions (SubjectID, SessionTime, SessionDate, SessionType) VALUES (?, ?, ?, ?)''', session_data)

# Commit changes
conn.commit()

# Close connection
conn.close()
```



SessionID	SubjectID	SessionTime	SessionDate	SessionTyp
Filter	Filter	Filter	Filter	Filter
1	17	11:00	2024-08-13	Offline
2	12	13:00	2024-08-17	Online
3	7	13:00	2024-06-18	Offline
4	27	8:00	2024-05-15	Online
5	23	9:00	2024-06-15	Online
6	32	18:00	2024-03-14	Online
7	49	8:00	2024-01-22	Offline
8	1	10:00	2024-01-23	Offline
9	24	9:00	2024-02-23	Offline
10	33	10:00	2024-05-26	Online
11	34	14:00	2024-01-11	Online
12	3	14:00	2024-07-23	Online
13	15	14:00	2024-05-22	Online
14	49	16:00	2024-02-11	Offline
15	42	14:00	2024-07-23	Offline
16	27	12:00	2024-09-18	Online
17	34	18:00	2024-05-24	Offline
18	16	10:00	2024-02-20	Online

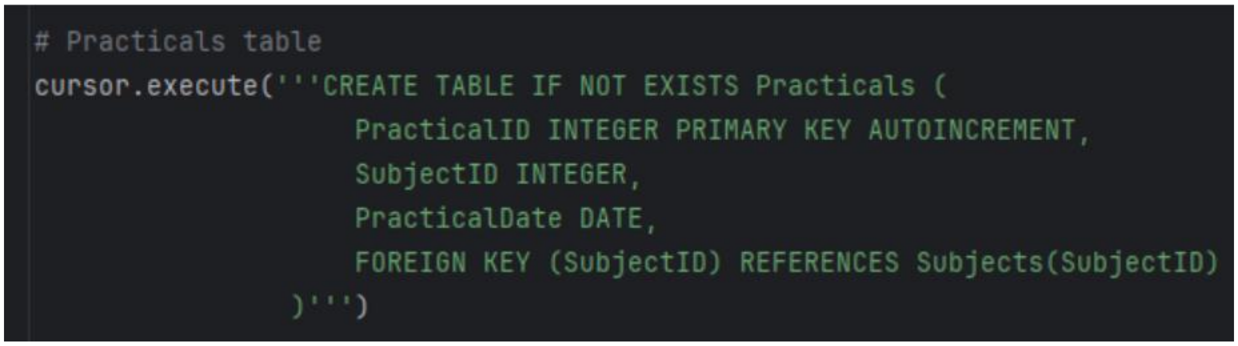
<u>Practicals Table:</u> The practical table is formulated to gather essential details concerning practical sessions, enabling the efficient tracking and analysis of class-related information.

PracticalID: identifier for each practical. (Ratio, Primary Key)

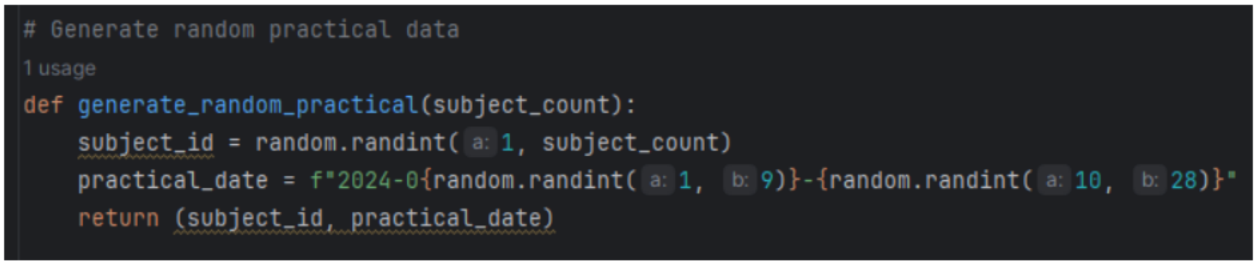
SubjectID: serves as a unique identifier for each subject. (Ratio, Foreign

Key) PracticalData: provides date for the date. (Interval)

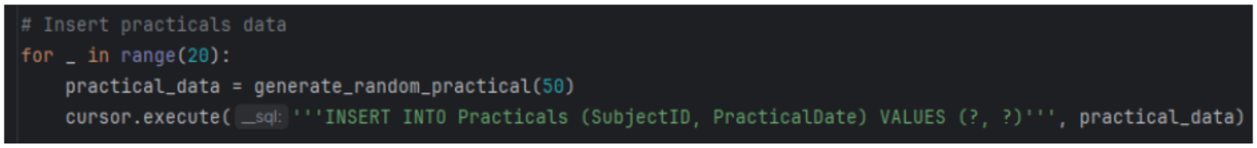
The provided code snippet illustrates the creation of a table named Practicals in a SQLite database using Python's SQLite



```
# Practicals table
cursor.execute('''CREATE TABLE IF NOT EXISTS Practicals (
                    PracticalID INTEGER PRIMARY KEY AUTOINCREMENT,
                    SubjectID INTEGER,
                    PracticalDate DATE,
                    FOREIGN KEY (SubjectID) REFERENCES Subjects(SubjectID)
                )''')
```



```
# Generate random practical data
1 usage
def generate_random_practical(subject_count):
    subject_id = random.randint(1, subject_count)
    practical_date = f"2024-0{random.randint(1, 9)}-{random.randint(10, 28)}"
    return (subject_id, practical_date)
```



```
# Insert practicals data
for _ in range(20):
    practical_data = generate_random_practical(50)
    cursor.execute('''INSERT INTO Practicals (SubjectID, PracticalDate) VALUES (?, ?)''', practical_data)
```

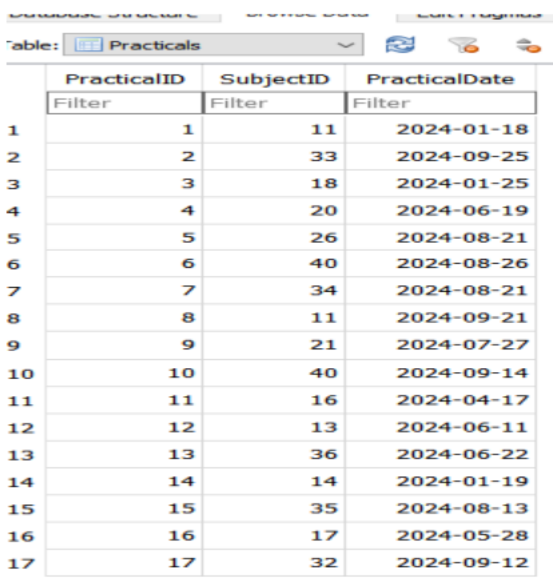


Table: Practicals

	PracticalID	SubjectID	PracticalDate
	Filter	Filter	Filter
1	1	11	2024-01-18
2	2	33	2024-09-25
3	3	18	2024-01-25
4	4	20	2024-06-19
5	5	26	2024-08-21
6	6	40	2024-08-26
7	7	34	2024-08-21
8	8	11	2024-09-21
9	9	21	2024-07-27
10	10	40	2024-09-14
11	11	16	2024-04-17
12	12	13	2024-06-11
13	13	36	2024-06-22
14	14	14	2024-01-19
15	15	35	2024-08-13
16	16	17	2024-05-28
17	17	32	2024-09-12

<u>Practicals Table:</u> The Examinations table is designed to collect vital information regarding examinations, facilitating the effective tracking and analysis of exam-related data.

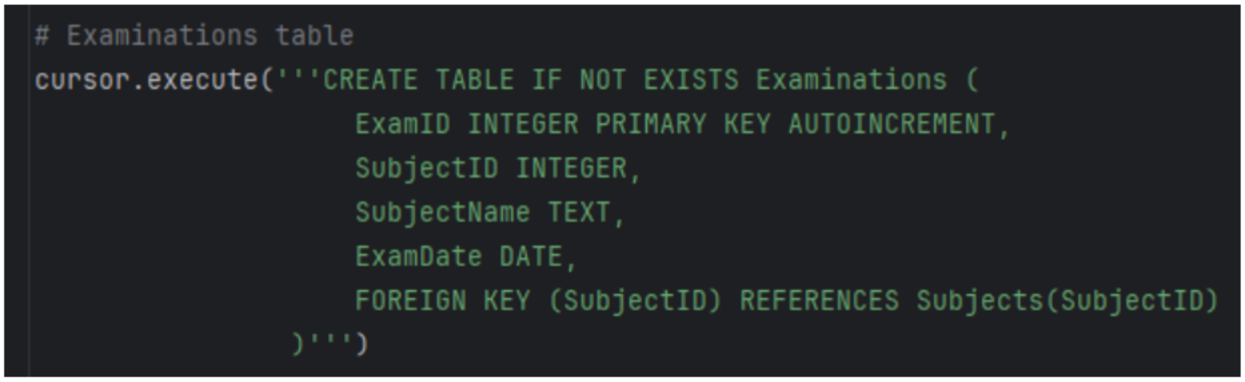
ExamID: This represents a unique identifier for each examination. (Ratio, Primary Key)

SubjectID: This represents the identifier for the subject of the examination. (Ratio, Foreign

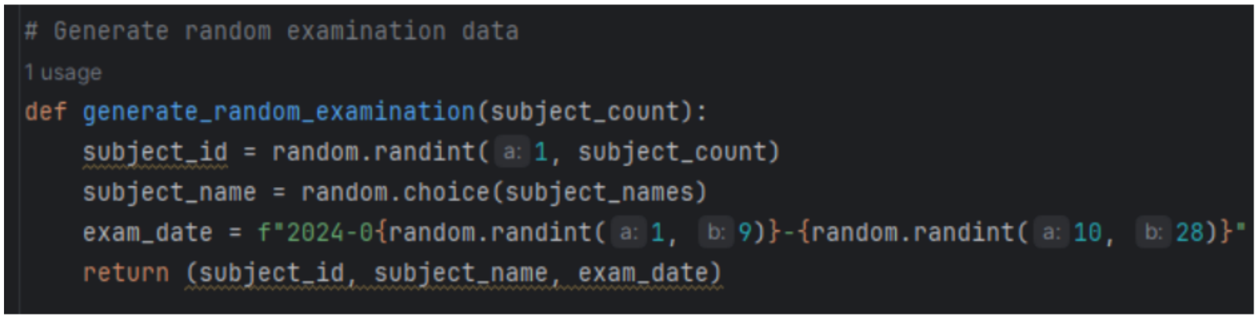
Key) **SubjectName:** This is the name or title of the subject. (Nominal)

ExamDate: This is the date of the examination. (interval)

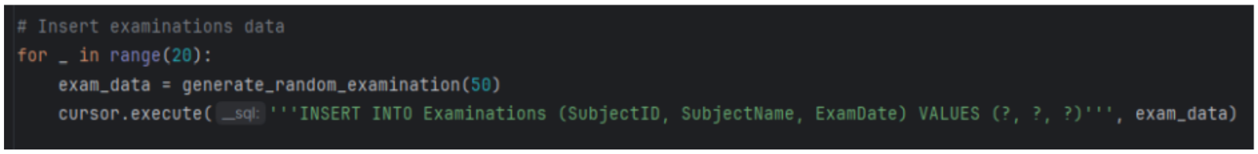
The provided code snippet, along with the accompanying image, demonstrates the procedure of generating data for the examination table using Python and the dataset designated for the examination table in SQLite.



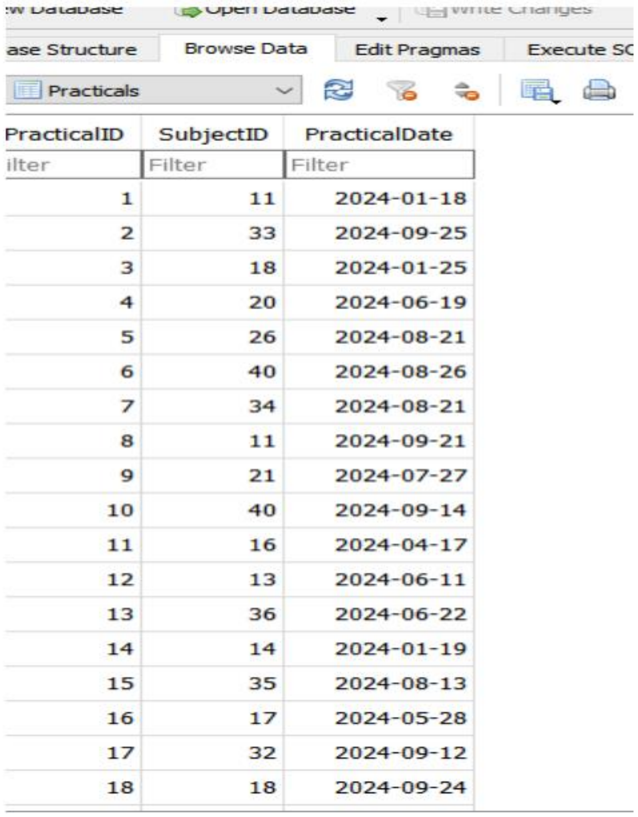
```
# Examinations table
cursor.execute('''CREATE TABLE IF NOT EXISTS Examinations (
                    ExamID INTEGER PRIMARY KEY AUTOINCREMENT,
                    SubjectID INTEGER,
                    SubjectName TEXT,
                    ExamDate DATE,
                    FOREIGN KEY (SubjectID) REFERENCES Subjects(SubjectID)
                )''')
```



```
# Generate random examination data
1 usage
def generate_random_examination(subject_count):
    subject_id = random.randint( a: 1, subject_count)
    subject_name = random.choice(subject_names)
    exam_date = f"2024-0{random.randint( a: 1,  b: 9)}-{random.randint( a: 10,  b: 28)}"
    return (subject_id, subject_name, exam_date)
```



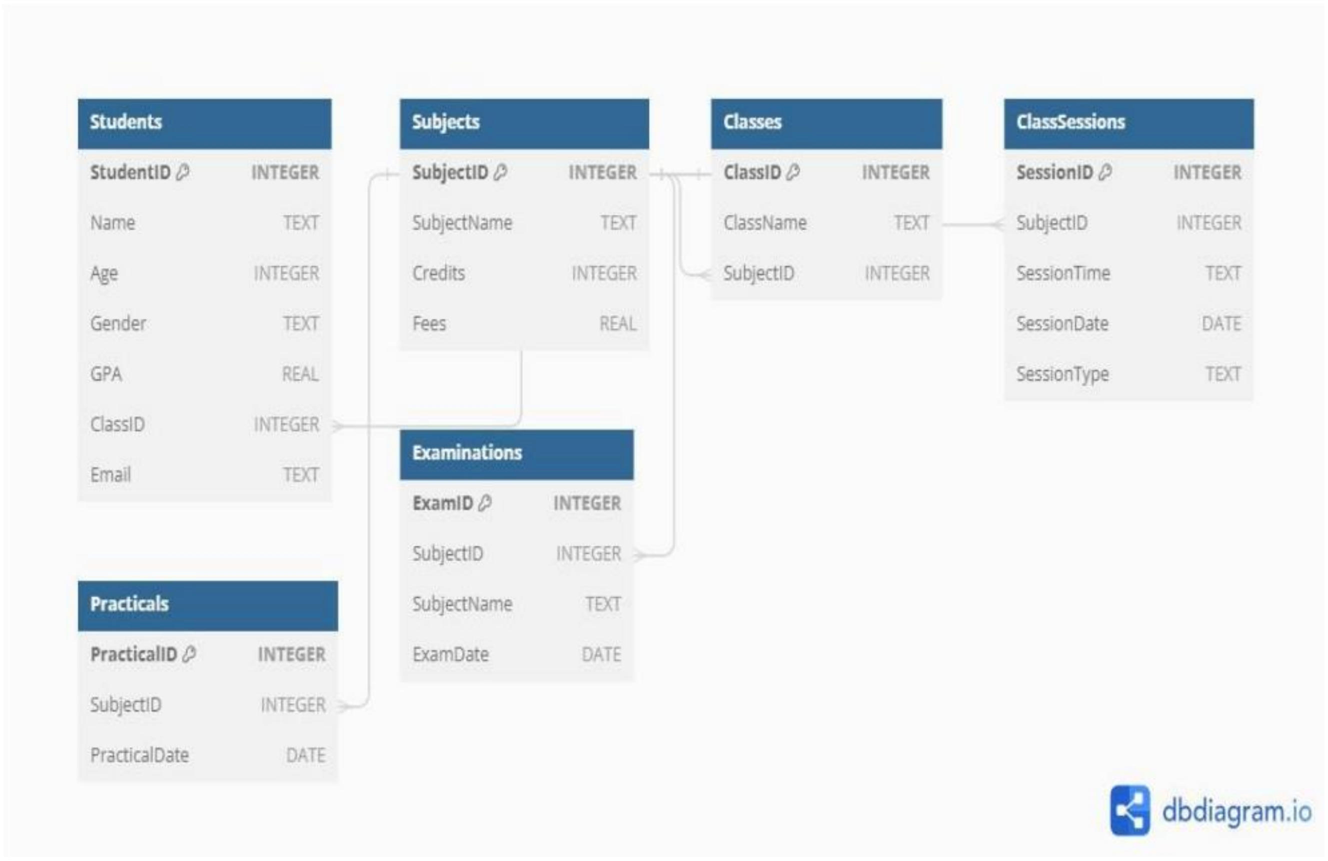
```
# Insert examinations data
for _ in range(20):
    exam_data = generate_random_examination(50)
    cursor.execute( _sql: '''INSERT INTO Examinations (SubjectID, SubjectName, ExamDate) VALUES (?, ?, ?)''', exam_data)
```



PracticalID	SubjectID	PracticalDate
ilter	Filter	Filter
1	11	2024-01-18
2	33	2024-09-25
3	18	2024-01-25
4	20	2024-06-19
5	26	2024-08-21
6	40	2024-08-26
7	34	2024-08-21
8	11	2024-09-21
9	21	2024-07-27
10	40	2024-09-14
11	16	2024-04-17
12	13	2024-06-11
13	36	2024-06-22
14	14	2024-01-19
15	35	2024-08-13
16	17	2024-05-28
17	32	2024-09-12
18	18	2024-09-24

Database Schema:

Within the multi-table database, there are six primary tables: Student, Subject, Classes, ClasseSession, Practical, and Examination. I've strategically utilized primary, foreign, and compound keys to forge connections between them. Additionally, I've enforced constraints on particular columns in each table to uphold data integrity.



Students	
StudentID	INTEGER
Name	TEXT
Age	INTEGER
Gender	TEXT
GPA	REAL
ClassID	INTEGER
Email	TEXT

Subjects	
SubjectID	INTEGER
SubjectName	TEXT
Credits	INTEGER
Fees	REAL

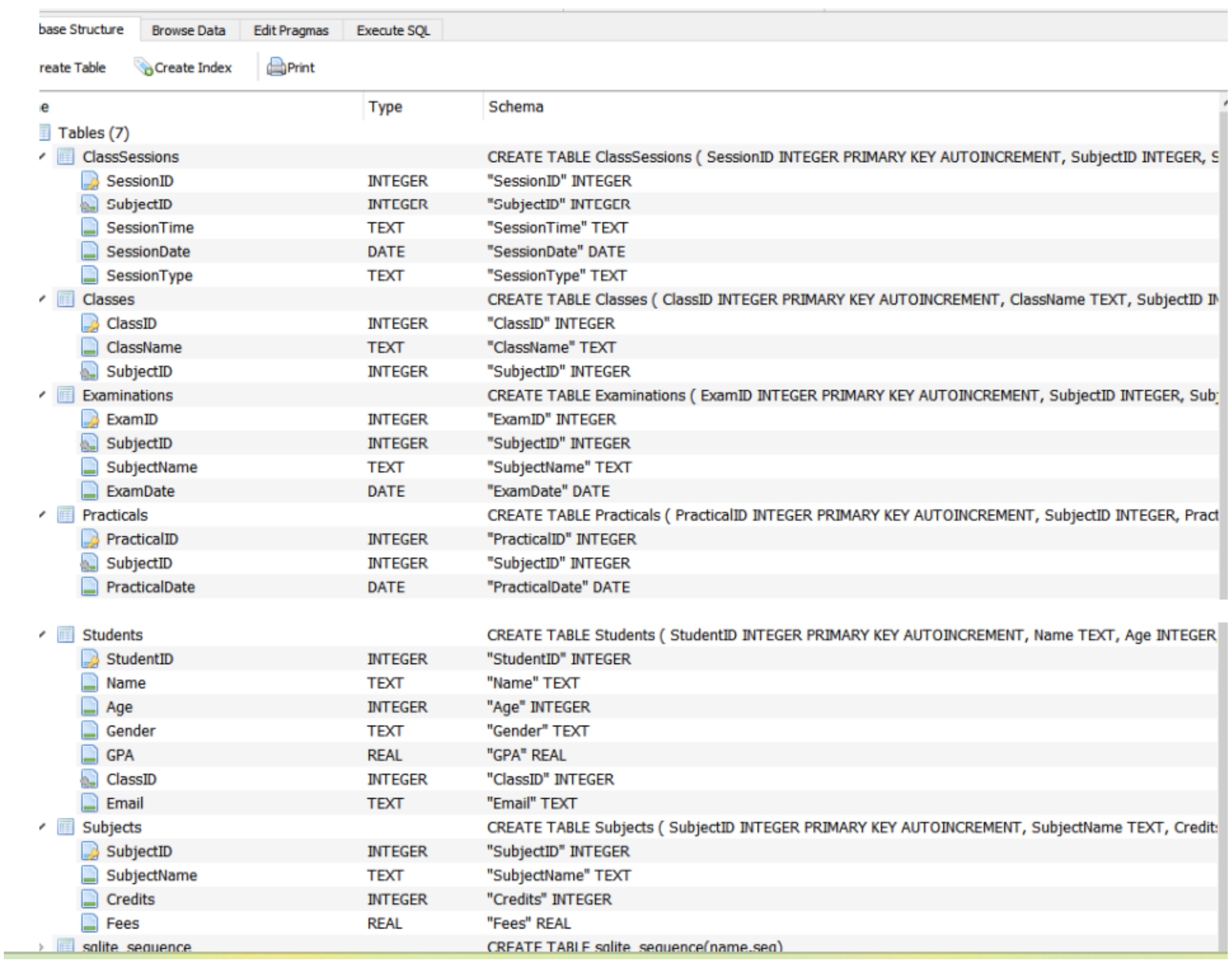
Classes	
ClassID	INTEGER
ClassName	TEXT
SubjectID	INTEGER

ClassSessions	
SessionID	INTEGER
SubjectID	INTEGER
SessionTime	TEXT
SessionDate	DATE
SessionType	TEXT

Examinations	
ExamID	INTEGER
SubjectID	INTEGER
SubjectName	TEXT
ExamDate	DATE

Practicals	
PracticalID	INTEGER
SubjectID	INTEGER
PracticalDate	DATE

The database schema depicted in the screenshot below showcases foreign and compound keys, along with the constraints allocated to each table.



e	Type	Schema
Tables (7)		
ClassSessions		CREATE TABLE ClassSessions (SessionID INTEGER PRIMARY KEY AUTOINCREMENT, SubjectID INTEGER, S
SessionID	INTEGER	"SessionID" INTEGER
SubjectID	INTEGER	"SubjectID" INTEGER
SessionTime	TEXT	"SessionTime" TEXT
SessionDate	DATE	"SessionDate" DATE
SessionType	TEXT	"SessionType" TEXT
Classes		CREATE TABLE Classes (ClassID INTEGER PRIMARY KEY AUTOINCREMENT, ClassName TEXT, SubjectID I
ClassID	INTEGER	"ClassID" INTEGER
ClassName	TEXT	"ClassName" TEXT
SubjectID	INTEGER	"SubjectID" INTEGER
Examinations		CREATE TABLE Examinations (ExamID INTEGER PRIMARY KEY AUTOINCREMENT, SubjectID INTEGER, Sub
ExamID	INTEGER	"ExamID" INTEGER
SubjectID	INTEGER	"SubjectID" INTEGER
SubjectName	TEXT	"SubjectName" TEXT
ExamDate	DATE	"ExamDate" DATE
Practicals		CREATE TABLE Practicals (PracticalID INTEGER PRIMARY KEY AUTOINCREMENT, SubjectID INTEGER, Pract
PracticalID	INTEGER	"PracticalID" INTEGER
SubjectID	INTEGER	"SubjectID" INTEGER
PracticalDate	DATE	"PracticalDate" DATE
Students		CREATE TABLE Students (StudentID INTEGER PRIMARY KEY AUTOINCREMENT, Name TEXT, Age INTEGER
StudentID	INTEGER	"StudentID" INTEGER
Name	TEXT	"Name" TEXT
Age	INTEGER	"Age" INTEGER
Gender	TEXT	"Gender" TEXT
GPA	REAL	"GPA" REAL
ClassID	INTEGER	"ClassID" INTEGER
Email	TEXT	"Email" TEXT
Subjects		CREATE TABLE Subjects (SubjectID INTEGER PRIMARY KEY AUTOINCREMENT, SubjectName TEXT, Credit
SubjectID	INTEGER	"SubjectID" INTEGER
SubjectName	TEXT	"SubjectName" TEXT
Credits	INTEGER	"Credits" INTEGER
Fees	REAL	"Fees" REAL
sqlite_sequence		CREATE TABLE sqlite_sequence(name,seq)

Table Choice Justification

Together, these six tables compile a thorough dataset of student records, encompassing diverse facets of student demographics. This methodical approach to database design enables streamlined data analysis and reporting. Refer below for the rationale behind table's inclusion.

Student Table: Stores demographic details of students, facilitating individual student record management.

Subject Table: Catalogs information about academic subjects offered, aiding in subject management and organization.

Classes Table: Records details about available classes or courses, enabling effective class enrollment and scheduling.
ClassSession Table: Tracks sessions or meetings associated with classes, aiding in session organization and scheduling.

Practical Table: Documents practical or laboratory sessions related to subjects or classes, facilitating practical session management

Examination Table: Manages and tracks examination-related information such as schedules and grades, supporting examination administration and record-keeping.

Ethics and Data Privacy Considerations

Throughout the creation of the six data tables - the Student Table, Subject Table, Classes Table, ClassSession Table, Practical Table, and Examination Table - a primary focus has been on maintaining stringent ethical standards and adhering to data privacy principles reflective of real world practices. This dedication is in alignment with the crucial objective of safeguarding individual privacy and adhering to data protection regulations, including but not limited to the General Data Protection Regulation (GDPR).

Conclusion

In summary, the development of a SQL database tailored for student record use has been successfully accomplished, comprising six essential tables: Student, Subject, Classes, ClassSession, Practical, and Examination. The process entailed generating random data through Python, ensuring adherence to ethical standards and compliance with data privacy regulations. Each table's design fulfills distinct student-related purposes while upholding data integrity through primary and foreign keys, as well as column constraints. Ethical considerations, such as data anonymization and the implementation of random identifiers, have been integrated to safeguard individual privacy. The resulting database establishes a solid foundation for student records, presenting a robust and ethical solution for student data management within a student records context.