DENTAL CLINIC DATABASE SYSTEM

1. INTRODUCTION

The Dentist Clinic Database is designed to manage information related to employees, patients, appointments, treatments, medicines, and operations. The database ensures efficient storage, manipulation, and retrieval of data to support clinical operations and management decision-making. It includes at least 8 tables to organize data about staff (dentists, nurses, employees), patients, appointments, treatments (medical and operational), medicines, and payments. The system supports data analysis through statistical queries to provide insights for management, such as identifying high-performing doctors, popular medicines, and patient payment trends. The design adheres to normalization principles (up to 3NF) to ensure data integrity and scalability.

Why Use SQLite3 for the Dental Clinic Database?

SQLite3 is ideal for the dental clinic's 3NF-normalized database due to:

- **Simplicity**: Serverless, single-file database with minimal setup, perfect for small clinics.
- **Efficiency**: Handles moderate data (e.g., `Patient`, `Appointment`) and supports foreign keys for relationships (e.g., `DentistID` in `Appointment`).
- **Reliability**: ACID-compliant, ensuring data integrity for critical records like `Payment` or `Prescription`.
- **Cost-Effective**: Free and low-resource, reducing costs for hardware and licensing.
- **Portability**: Cross-platform, embedded, and easy to back up or transfer.
- **Scalability**: Suitable for small to medium workloads, with the 3NF design optimizing performance.

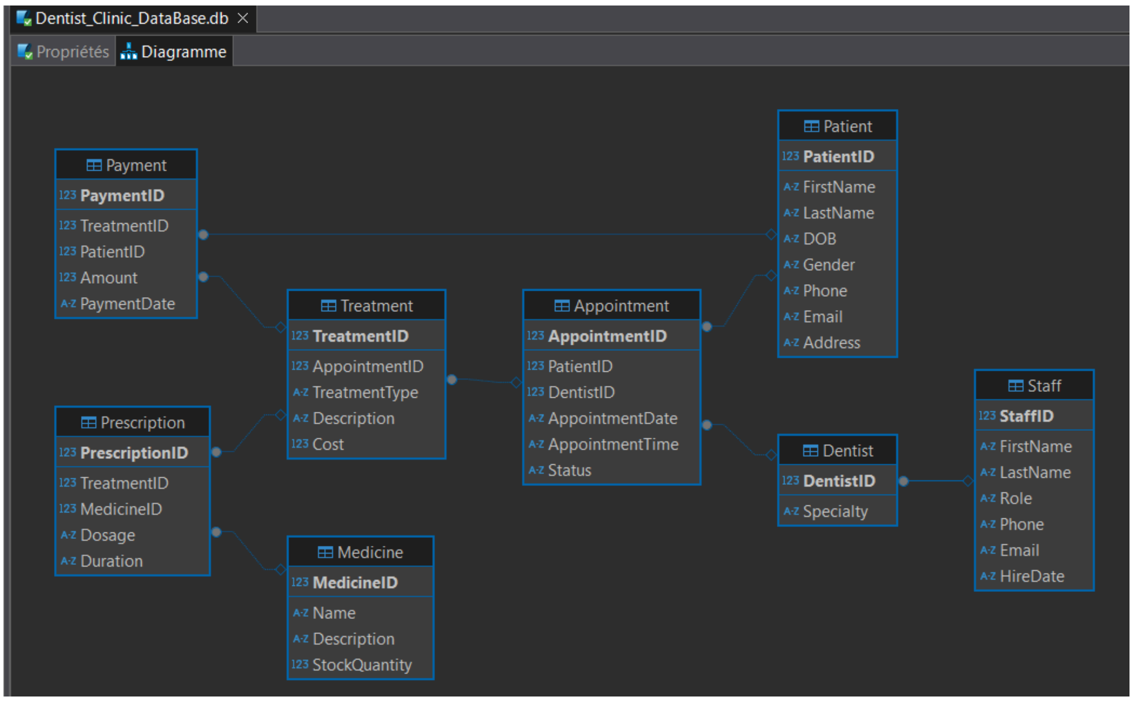
CONNEXION TO THE DATABASE

In [37]:

```
import pandas as pd # import the pandas library
import sqlite3 # import the sqlite3 module
conn = sqlite3.connect("Dentist_Clinic_DataBase.db") # create a connection to the database
cursor = conn.cursor() # create a cursor object to execute SQL commands
cursor.execute("SELECT name FROM sqlite_master WHERE type='table';")
tables = cursor.fetchall()
# show the names of the tables in the database
for table in tables:
    print(table[0])
```

```
Staff
sqlite_sequence
Dentist
Patient
Appointment
Treatment
Medicine
Prescription
Payment
```

2. ER DIAGRAM



3. RELATIOSHIPS

- **Staff-Dentist**: One-to-One (Dentist is a specialized Staff).
- **Dentist-Appointment**: One-to-Many (A dentist can have many appointments, but an appointment is handled by one dentist).
- **Patient-Appointment**: One-to-Many (A patient can have multiple appointments, but each appointment is for one patient).
- **Appointment-Treatment**: One-to-Many (An appointment can have multiple treatments).
- **Treatment-Prescription**: One-to-Many (A treatment can include multiple prescriptions).
- **Medicine-Prescription**: Many-to-Many (A prescription can include multiple medicines, resolved via the Prescription table).
- **Treatment-Payment**: One-to-One (Each treatment has one payment).
- **Patient-Payment**: One-to-Many (A patient can have multiple payments).

4. RELATIONAL MODEL

A. Staff

Attribute	Description
StaffID	Primary Key, unique identifier
FirstName	Staff's first name

Attribute	Description
LastName	Staff's last name
Role	Staff's role (e.g., admin, dentist)
Phone	Contact phone number
Email	Email address
HireDate	Date of hiring

B. Dentist

Attribute	Description
DentistID	Primary Key, unique identifier
Specialty	Dentist's area of expertise

- **Foreign Key**: `DentistID` references `Staff(StaffID)`

C. Patient

Attribute	Description
PatientID	Primary Key, unique identifier
FirstName	Patient's first name
LastName	Patient's last name
DOB	Date of birth
Gender	Patient's gender
Phone	Contact phone number
Email	Email address
Address	Residential address

D. Appointment

Attribute	Description
AppointmentID	Primary Key, unique identifier
PatientID	ID of the patient
DentistID	ID of the dentist
AppointmentDate	Date of the appointment
AppointmentTime	Time of the appointment
Status	Appointment status (e.g., scheduled, completed)

- **Foreign Keys**:
  - `PatientID` references `Patient(PatientID)`
  - `DentistID` references `Dentist(DentistID)`

E. Treatment

Attribute	Description
TreatmentID	Primary Key, unique identifier
AppointmentID	ID of the appointment
TreatmentType	Type of treatment (e.g., cleaning, filling)
Description	Details of the treatment
Cost	Cost of the treatment

- **Foreign Key**: `AppointmentID` references `Appointment(AppointmentID)`

F. Medicine

Attribute	Description
MedicineID	Primary Key, unique identifier
Name	Name of the medicine
Description	Details of the medicine
StockQuantity	Available quantity in stock

G. Prescription

Attribute	Description
PrescriptionID	Primary Key, unique identifier
TreatmentID	ID of the treatment
MedicineID	ID of the medicine
Dosage	Dosage instructions
Duration	Duration of the prescription

- **Foreign Keys**:
  - `TreatmentID` references `Treatment(TreatmentID)`
  - `MedicineID` references `Medicine(MedicineID)`

H. Payment

Attribute	Description
PaymentID	Primary Key, unique identifier
TreatmentID	ID of the treatment
PatientID	ID of the patient
Amount	Payment amount
PaymentDate	Date of the payment

- **Foreign Keys**:
  - `TreatmentID` references `Treatment(TreatmentID)`
  - `PatientID` references `Patient(PatientID)`

6. Normalization (Up to 3NF)

The database is normalized to the Third Normal Form (3NF) to eliminate redundancy and ensure data integrity. Below is the normalization process for the dental clinic database:

First Normal Form (1NF)

All tables have primary keys to uniquely identify records (e.g., StaffID, PatientID). No repeating groups or arrays (e.g., patient contact information is stored as atomic values, not lists). All attributes are atomic (e.g., FirstName and LastName are separate fields, not combined).

Examples:

Staff table: StaffID, FirstName, LastName, Role, Phone, Email, HireDate. Patient table: PatientID, FirstName, LastName, DOB, Gender, Phone, Email, Address.

Second Normal Form (2NF)

All tables are in 1NF. Non-key attributes fully depend on the entire primary key (no partial dependencies). Example: In the Appointment table, AppointmentDate and AppointmentTime depend on the entire AppointmentID, not partially on PatientID or DentistID.

Details:

Ensures that attributes like AppointmentDate are not redundantly tied to subsets of composite keys. Tables like Treatment (TreatmentID, AppointmentID, TreatmentType, Description, Cost) have no composite keys, so 2NF is inherently satisfied.

Third Normal Form (3NF)

All tables are in 2NF. No transitive dependencies (non-key attributes do not depend on other non-key attributes). Examples: In the Patient table, Address depends directly on PatientID, not on Phone or Email. In the Dentist table, Specialty depends on DentistID, not on other attributes of the Staff table (e.g., FirstName or HireDate).

Note: Normalization ensures data integrity and minimizes redundancy, making the database efficient for queries and maintenance.

ALL TABLES

STAFF TABLE

In [38]:

```
query ="SELECT * FROM staff" # SQL query to select all data from the staff table
staff_data = pd.read_sql_query(query, conn) # read the data into a pandas DataFrame
staff_data.head(10) #show the first 10 rows of the DataFrame
```

Out[38]:

	StaffID	FirstName	LastName	Role	Phone	Email	HireDate
0	1	Tom	Green	Dentist	555-6724	john.smith@clinic.com	2024-06-24
1	2	Sarah	Scott	Dentist	555-4664	mary.johnson@clinic.com	2018-08-15
2	3	Karen	Johnson	Dentist	555-8206	alice.brown@clinic.com	2021-09-09
3	4	Mary	Wilson	Dentist	555-6112	bob.taylor@clinic.com	2022-05-06
4	5	Emily	Wilson	Dentist	555-1404	emma.wilson@clinic.com	2024-12-14
5	6	Alice	Allen	Dentist	555-3159	james.davis@clinic.com	2021-02-23
6	7	Michael	Lewis	Dentist	555-1355	sarah.clark@clinic.com	2018-02-17
7	8	Lisa	Clark	Dentist	555-4223	david.lewis@clinic.com	2018-02-13
8	9	Steven	Davis	Dentist	555-8800	lisa.walker@clinic.com	2021-05-19
9	10	Emily	Baker	Dentist	555-5749	michael.hall@clinic.com	2016-03-22

APPOINTMENT TABLE

In [40]:

```
query = "SELECT * FROM appointment" # SQL query to select all data from the appointment table
appointment_data = pd.read_sql_query(query, conn) # read the data into a pandas DataFrame
appointment_data.head(10) # show the first 10 rows of the DataFrame
```

Out[40]:

	AppointmentID	PatientID	DentistID	AppointmentDate	AppointmentTime	Status
0	1	19	7	2024-01-17	13:15:00	Scheduled
1	2	42	8	2024-12-17	11:30:00	Scheduled
2	3	15	17	2024-12-21	17:30:00	Cancelled
3	4	25	2	2025-02-18	10:45:00	Scheduled
4	5	25	1	2024-06-17	14:15:00	Scheduled
5	6	50	26	2024-12-02	10:00:00	Cancelled
6	7	3	27	2024-10-09	16:30:00	Scheduled
7	8	41	10	2024-08-18	11:15:00	Cancelled
8	9	40	11	2025-10-07	13:00:00	Completed
9	10	23	17	2025-02-14	13:45:00	Cancelled

DENTIST TABLE

In [41]:

```
query = "SELECT * FROM dentist" # SQL query to select all data from the dentist table
dentist_table = pd.read_sql_query(query, conn) # read the data into a pandas DataFrame
dentist_table.head(10) # show the first 10 rows of the DataFrame
```

Out[41]:

	DentistID	Specialty
0	1	Periodontics
1	2	Endodontics
2	3	Orthodontics
3	4	Prosthodontics
4	5	Endodontics
5	6	Orthodontics
6	7	Endodontics
7	8	Periodontics
8	9	Endodontics
9	10	Endodontics

TREATMENT TABLE

In [42]:

```
query = "SELECT * FROM Treatment" # SQL query to select all data from the treatment table
treatment_table = pd.read_sql_query(query, conn) # read the data into a pandas DataFrame
treatment_table.head(10) # show the first 10 rows of the DataFrame
```

Out[42]:

	TreatmentID	AppointmentID	TreatmentType	Description	Cost
0	1	1	Medical	Filling	418.54
1	2	2	Medical	Filling	195.95
2	3	3	Medical	Filling	268.36
3	4	4	Medical	Braces Adjustment	370.04
4	5	5	Operational	Root Canal	769.76
5	6	6	Operational	Filling	672.31
6	7	7	Medical	Root Canal	566.27
7	8	8	Operational	Teeth Cleaning	242.85
8	9	9	Operational	Teeth Cleaning	641.86
9	10	10	Operational	Teeth Cleaning	464.76

PRESCRIPTION TABLE

In [43]:

```
query = "SELECT * FROM Prescription" # SQL query to select all data from the precscription table
prescription_table = pd.read_sql_query(query, conn) # read the data into a pandas DataFrame
prescription_table.head(10) # show the first 10 rows of the
```

Out[43]:

	PrescriptionID	TreatmentID	MedicineID	Dosage	Duration
0	1	1	32	495mg twice daily	13 days
1	2	37	15	260mg thrice daily	9 days
2	3	42	17	170mg thrice daily	12 days
3	4	29	33	289mg thrice daily	5 days
4	5	6	9	463mg once daily	4 days
5	6	15	9	156mg once daily	6 days
6	7	41	27	312mg once daily	12 days
7	8	18	43	404mg once daily	13 days
8	9	42	1	388mg twice daily	8 days
9	10	3	13	460mg once daily	8 days

PAYMENT TABLE

In [44]:

```
query = "SELECT * FROM Payment" # SQL query to select all data from the payment table
payment_table = pd.read_sql_query(query, conn) # read the data into a pandas DataFrame
payment_table.head(10) # show the first 10 rows of the DataFrame
```

Out[44]:

	PaymentID	TreatmentID	PatientID	Amount	PaymentDate
0	1	1	19	281.63	2025-10-31
1	2	2	42	513.60	2025-05-12
2	3	3	15	143.93	2025-03-21
3	4	4	25	410.24	2024-11-01
4	5	5	25	134.06	2025-08-11
5	6	6	50	207.07	2024-04-05
6	7	7	3	311.71	2024-10-08
7	8	8	41	965.86	2025-07-15
8	9	9	40	937.19	2025-08-14
9	10	10	23	381.54	2025-05-30

PATIENT TABLE

In [45]:

```
query = "SELECT * FROM Patient" # SQL query to select all data from the patient table
patient_table = pd.read_sql_query(query, conn) # read the data into a pandas DataFrame
patient_table.head(10) # show the first 10 rows of the DataFrame
```

Out[45]:

	PatientID	FirstName	LastName	DOB	Gender	Phone	Email	Address
0	1	Michael	Allen	1972-05-03	M	555-4259	john.smith@patient.com	789 Pine Rd
1	2	John	Brown	1979-06-21	Other	555-7855	mary.johnson@patient.com	456 Oak Ave
2	3	David	Brown	1967-12-24	Other	555-5476	alice.brown@patient.com	789 Pine Rd
3	4	Sarah	Lewis	2003-07-10	M	555-4359	bob.taylor@patient.com	456 Oak Ave
4	5	Susan	Taylor	1988-02-20	Other	555-1668	emma.wilson@patient.com	456 Oak Ave
5	6	Karen	Taylor	1963-02-23	Other	555-7135	james.davis@patient.com	456 Oak Ave
6	7	Lisa	Davis	1998-09-09	Other	555-2845	sarah.clark@patient.com	202 Birch Ln
7	8	Emma	Lewis	1995-02-09	M	555-4733	david.lewis@patient.com	789 Pine Rd
8	9	Lisa	Green	1975-08-08	F	555-4497	lisa.walker@patient.com	456 Oak Ave
9	10	Mark	Lewis	1984-06-13	M	555-1973	michael.hall@patient.com	789 Pine Rd

MEDCINE TABLE

In [46]:

```
query = "SELECT * FROM medicine" # SQL query to select all data from the medicine table
medecine_table = pd.read_sql_query(query, conn) # read the data into a pandas DataFrame
medecine_table.head(10) # show the first 10 rows of the DataFrame
```

Out[46]:

	MedicineID	Name	Description	StockQuantity
0	1	Lidocaine 1	Lidocaine for dental use	115
1	2	Fluoride Gel 2	Paracetamol for dental use	162
2	3	Ibuprofen 3	Amoxicillin for dental use	172
3	4	Ibuprofen 4	Amoxicillin for dental use	19
4	5	Fluoride Gel 5	Paracetamol for dental use	177
5	6	Amoxicillin 6	Lidocaine for dental use	67
6	7	Amoxicillin 7	Paracetamol for dental use	21
7	8	Lidocaine 8	Ibuprofen for dental use	59
8	9	Ibuprofen 9	Lidocaine for dental use	158
9	10	Paracetamol 10	Ibuprofen for dental use	71

16 SQL Queries for Statistical Information

1. Count of Staff by Role

In [47]:

```
query = """
SELECT Role, COUNT(*) as StaffCount
FROM Staff
GROUP BY Role;"""
staff_data = pd.read_sql_query(query, conn)
staff_data # show the first 10 rows of the DataFrame
```

Out[47]:

	Role	StaffCount
0	Dentist	30
1	Employee	10
2	Nurse	10

2. Number of Dentists by Specialty

In [48]:

```
query = """
SELECT Specialty, COUNT(*) as DentistCount
FROM Dentist
GROUP BY Specialty;"""
dentist_data = pd.read_sql_query(query, conn)
dentist_data
```

Out[48]:

	Specialty	DentistCount
0	Endodontics	10
1	General Dentistry	1
2	Orthodontics	9
3	Periodontics	5
4	Prosthodontics	5

3. Total Number of Patients by Gender

In [49]:

```
query = """
SELECT Gender, COUNT(*) as PatientCount
FROM Patient
GROUP BY Gender;"""
patient_data = pd.read_sql_query(query, conn)
patient_data
```

Out[49]:

	Gender	PatientCount
0	F	11
1	M	16
2	Other	23

4. Total Appointments by Status

In [50]:

```
query = """
SELECT Status, COUNT(*) as AppointmentCount
FROM Appointment
GROUP BY Status;"""
status_data = pd.read_sql_query(query, conn)
status_data
```

Out[50]:

	Status	AppointmentCount
0	Cancelled	19
1	Completed	13
2	Scheduled	18

5. Average Cost of Treatments by Type

In [51]:

```
query = """
SELECT TreatmentType, ROUND(AVG(Cost), 2) as AvgCost
FROM Treatment
GROUP BY TreatmentType;"""
treatment_data = pd.read_sql_query(query, conn)
treatment_data
```

Out[51]:

	TreatmentType	AvgCost
0	Medical	445.91
1	Operational	504.16

6. Total Revenue from Treatments

In [52]:

```
query = """
SELECT ROUND(SUM(Cost), 2) as TotalRevenue
FROM Treatment;"""
status_data = pd.read_sql_query(query, conn)
status_data
```

Out[52]:

	TotalRevenue
0	23809.96

7. Medicine Stock Levels

In [ ]:

```
query = """
SELECT Name, StockQuantity
FROM Medicine
WHERE StockQuantity < 20
ORDER BY StockQuantity ASC;"""
medecine_data = pd.read_sql_query(query, conn)
medecine_data
```

Out[ ]:

	Name	StockQuantity
0	Fluoride Gel 40	11
1	Lidocaine 50	11
2	Amoxicillin 29	12
3	Ibuprofen 44	12
4	Ibuprofen 4	19

8. Total Prescriptions by Medicine

In [54]:

```
query = """
SELECT m.Name, COUNT(p.PrescriptionID) as PrescriptionCount
```

```
FROM Medicine m
LEFT JOIN Prescription p ON m.MedicineID = p.MedicineID
GROUP BY m.MedicineID, m.Name
ORDER BY PrescriptionCount DESC;"""
medecine_prescription_data = pd.read_sql_query(query, conn)
medecine_prescription_data
```

Out[54]:

	Name	PrescriptionCount
0	Fluoride Gel 2	3
1	Ibuprofen 9	3
2	Ibuprofen 19	3
3	Lidocaine 43	3
4	Lidocaine 1	2
5	Ibuprofen 13	2
6	Amoxicillin 14	2
7	Paracetamol 17	2
8	Lidocaine 27	2
9	Amoxicillin 33	2
10	Fluoride Gel 34	2
11	Amoxicillin 35	2
12	Ibuprofen 48	2
13	Ibuprofen 3	1
14	Ibuprofen 4	1
15	Fluoride Gel 5	1
16	Amoxicillin 7	1
17	Lidocaine 8	1
18	Fluoride Gel 15	1
19	Amoxicillin 21	1
20	Amoxicillin 24	1
21	Amoxicillin 26	1
22	Amoxicillin 29	1
23	Ibuprofen 32	1
24	Lidocaine 36	1
25	Fluoride Gel 37	1
26	Lidocaine 38	1
27	Amoxicillin 39	1
28	Paracetamol 41	1
29	Fluoride Gel 42	1
30	Ibuprofen 44	1
31	Amoxicillin 45	1
32	Fluoride Gel 49	1
33	Amoxicillin 6	0
34	Paracetamol 10	0
35	Paracetamol 11	0
36	Paracetamol 12	0
37	Fluoride Gel 16	0
38	Fluoride Gel 18	0
39	Lidocaine 20	0
40	Lidocaine 22	0
41	Ibuprofen 23	0
42	Ibuprofen 25	0
43	Lidocaine 28	0
44	Ibuprofen 30	0
45	Lidocaine 31	0
46	Fluoride Gel 40	0
47	Paracetamol 46	0
48	Ibuprofen 47	0
49	Lidocaine 50	0

9. Total Payments by Patient

In [55]:

```
query = """
SELECT p.FirstName, p.LastName, ROUND(SUM(py.Amount), 2) as TotalPaid
FROM Patient p
JOIN Payment py ON p.PatientID = py.PatientID
GROUP BY p.PatientID, p.FirstName, p.LastName
ORDER BY TotalPaid DESC;"""
medecine_patient_data = pd.read_sql_query(query, conn)
medecine_patient_data
```

Out[55]:

	FirstName	LastName	TotalPaid
0	David	Taylor	1641.84
1	Karen	Brown	1604.39
2	David	King	1568.85
3	John	Johnson	1550.05
4	Chris	Young	1265.17
5	Karen	Smith	1200.40
6	Alice	Hall	1168.83
7	David	Brown	1086.45
8	Lisa	Green	1062.04
9	Susan	Carter	1004.28
10	Alice	Carter	1001.38
11	Emma	Lewis	1000.29
12	Steven	Hall	996.25
13	Susan	Taylor	996.20
14	Lisa	Clark	990.76
15	Mark	Wright	965.86
16	Paul	Taylor	952.15
17	Alice	Lewis	849.22
18	Tom	Smith	822.65
19	Mary	Brown	710.25
20	Sarah	Lewis	689.08
21	John	Smith	678.85
22	Bob	Davis	619.58
23	Emma	Wilson	579.34
24	Michael	Young	570.81
25	Alice	Green	478.24
26	Tom	Taylor	381.54
27	Paul	Adams	257.73
28	Bob	King	246.97
29	Anna	Lewis	230.86
30	Karen	Taylor	224.12
31	Chris	Taylor	210.95
32	James	Hall	143.93
33	Emily	Hall	115.75

10. Average Appointment Duration by Treatment Type

In [56]:

```
query = """
SELECT t.TreatmentType, ROUND(AVG(t.Cost), 2) as AvgTreatmentCost
FROM Treatment t
JOIN Appointment a ON t.AppointmentID = a.AppointmentID
GROUP BY t.TreatmentType;"""
treatment_appoint_data = pd.read_sql_query(query, conn)
treatment_appoint_data
```

Out[56]:

	TreatmentType	AvgTreatmentCost
0	Medical	445.91
1	Operational	504.16

11. Prescriptions per Treatment Type

In [57]:

```
query = """
SELECT t.TreatmentType, COUNT(p.PrescriptionID) as PrescriptionCount
FROM Treatment t
LEFT JOIN Prescription p ON t.TreatmentID = p.TreatmentID
GROUP BY t.TreatmentType;"""
treatment_presc_data = pd.read_sql_query(query, conn)
treatment_presc_data
```

Out[57]:

	TreatmentType	PrescriptionCount
0	Medical	27
1	Operational	23

12. Number of Dentists

In [58]:

```
query = """
SELECT COUNT(*) as TotalDentists
FROM Dentist;""" # SQL query to count the number of staff members by role
denstist = pd.read_sql_query(query, conn)
denstist
```

Out[58]:

	TotalDentists
0	30

13. Total Number of Patients

In [59]:

```
query = """
SELECT COUNT(*) as TotalPatients
FROM Patient;""" # SQL query to count the number of staff members by role
patients = pd.read_sql_query(query, conn)
patients
```

Out[59]:

	TotalPatients
0	50

14. Number of Male Patients

In [60]:

```
query = """
SELECT gender,COUNT(*) as MalePatients
FROM Patient
WHERE Gender = 'M';""" # SQL query to count the number of staff members by role
patients_male = pd.read_sql_query(query, conn)
patients_male
```

Out[60]:

	Gender	MalePatients
0	M	16

15. Number of Scheduled Appointments

In [61]:
```
query = """
SELECT status,COUNT(*) as ScheduledAppointments
FROM Appointment
WHERE Status = 'Scheduled';""" # SQL query to count the number of staff members by role
appointment= pd.read_sql_query(query, conn)
appointment
```

Out[61]:

	Status	ScheduledAppointments
0	Scheduled	18

16. Total Cost of All Treatments

In [62]:
```
query = """
SELECT ROUND(SUM(Cost), 2) as TotalTreatmentCost
FROM Treatment;
""" # SQL query to count the number of staff members by role
cost= pd.read_sql_query(query, conn)
cost
```

Out[62]:

	TotalTreatmentCost
0	23809.96

githup source code link : https://github.com/TimotheeNkwar/Database-Systems/blob/main/DataBase_Project/Dentist_Clinic_DB_Project.ipynb

By TIMOTHEE NKWAR and CLED NGOY