Designing Database for a Hospital Python and Sqlite Software.

Abstract:

This report focuses on detailed examination of database design process, including the database generation process, the schema of the database, rationale for the selection of specific tables, ethical considerations integral to data generation, and practical examples elucidating query operations, notably joins and the utilization of diverse data types within the database.

Introduction:

The core objective of this project is the development of a SQL database meticulously tailored for hospital applications. Employing the programming language Python, I meticulously crafted a database that not only mirrors real-world scenarios but is also intricately woven with ethical considerations and data privacy policies. The database is structured around three pivotal tables: Patient, Medical, and Treatment. Importantly, this database schema is underpinned by the strategic utilization of primary and foreign keys, facilitating the establishment of relationships among tables, thereby safeguarding data integrity and ensuring efficient relational database management.

Database Generation:

The data for the hospital database was generated randomly using python code and the database consist of four main tables: Patient, Medical, Appointment, and Treatment which consist of 1000 rows across 20 unique columns. These columns are designed to hold data in various formats, such as nominal, ordinal, interval, and ratio. The database incorporates missing values to mirrors a real-world situation where data is not always clean. Each table is discussed below.

Patients Table: The "Patients" table is structured to hold essential information for each patient. The attributes include:

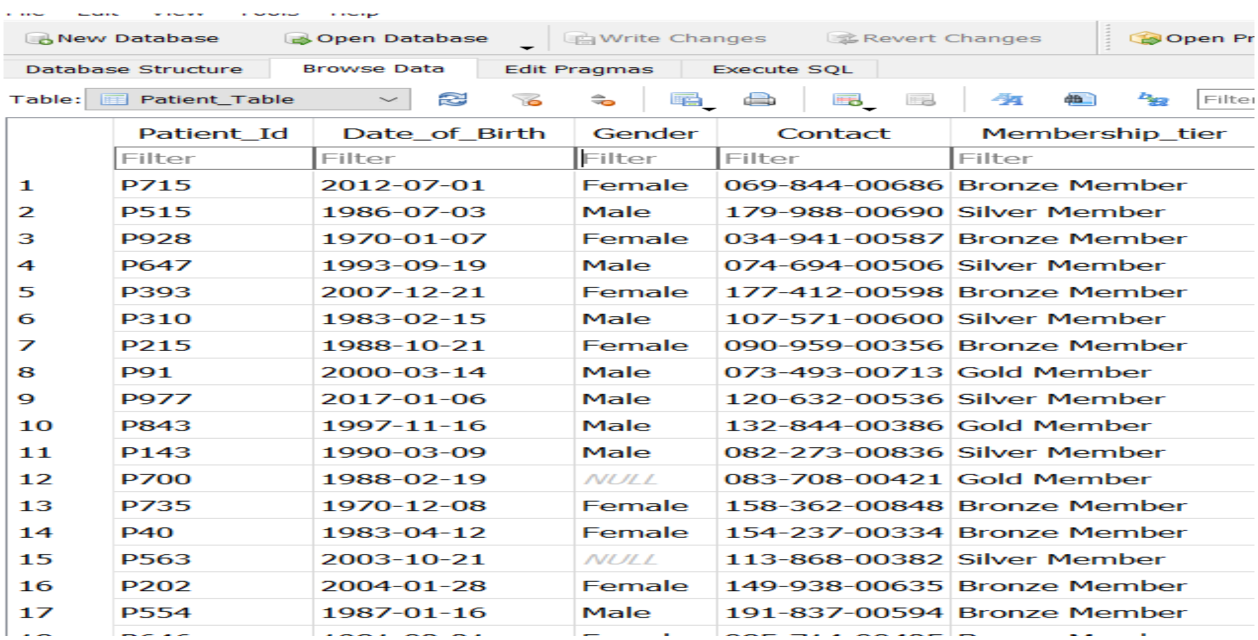
- **Patient ID**: A unique identifier for each patient. This is a nominal data type, meaning it is used to label the patient without providing any quantitative value.
- **Date of Birth**: Represents the birth date of each patient. As an interval data type, it signifies the time between a specific starting point and the birth of the patient, which allows for the calculation of age.
- **Gender**: The gender of the patient. This is also a nominal data type, indicating a category to which the patient belongs without implying any particular order.
- **Contact**: A piece of contact information for the patient, such as a phone number or email address. This is nominal as well since it is another unique identifier that does not have a mathematical operator.

- **Membership Tier**: The level of membership assigned to the patient. This is an ordinal data type, which suggests a ranking or order to the membership levels, such as 'Bronze', 'Silver', 'Gold', etc.

The code snippet and the image below illustrate shows the data generated process using python and the dataset for the patient's table in Sqlite Software

```
# importing necessary libraries

import numpy as np
import pandas as pd

# stating the number of observations to be created
n= 1000

#Patient_Id: Nominal Data Type

Patient_number= np.random.choice(range(1,1001), 1000, replace=False)

Patient_Id= [f'P{str(i).zfill(2)}' for i in Patient_number ]

# Date of Birth: Interval data type
birth_year = np.random.randint(1967, 2023, n)
birth_month = np.random.randint(1, 13, n)
birth_day = np.random.randint(1, 29, n)
Date_of_Birth = [f'{birth_year[i]}-{str(birth_month[i]).zfill(2)}-{str(birth_day[i]).zfill(2)}' for i in range(n)]


# Gender : # Nominal Data Type
Gender_Type= ['Male', 'Female']
Gender = np.random.choice(Gender_Type, n, p=[0.4,0.6])


# Contact_Number : Nominal Data Type
first_number= np.random.randint(0,200,n)
second_number = np.random.randint(111,999,n)
third_number = np.random.randint(333,899,n)
Contact=[f'{str(first_number[i]).zfill(3)}-{str(second_number[i]).zfill(3)}-{str(third_number[i]).zfill(5)}' for i in
range(n)]



# Membership Tier: Ordinal Data
membership_tier= ['Gold Member','Silver Member','Bronze Member']
membership_tier=np.random.choice(membership_tier,1000, p=[0.33,0.36,0.31])
```

```
# Creating a dataframe for the first table- Patients Table:
Patient_Table = pd.DataFrame({'Patient_Id':Patient_Id, 'Date_of_Birth':Date_of_Birth, 'Gender':Gender, 'Contact':Contact, 'Membership_tier':membership_tier})
```



	Patient_Id	Date_of_Birth	Gender	Contact	Membership_tier
1	P715	2012-07-01	Female	069-844-00686	Bronze Member
2	P515	1986-07-03	Male	179-988-00690	Silver Member
3	P928	1970-01-07	Female	034-941-00587	Bronze Member
4	P647	1993-09-19	Male	074-694-00506	Silver Member
5	P393	2007-12-21	Female	177-412-00598	Bronze Member
6	P310	1983-02-15	Male	107-571-00600	Silver Member
7	P215	1988-10-21	Female	090-959-00356	Bronze Member
8	P91	2000-03-14	Male	073-493-00713	Gold Member
9	P977	2017-01-06	Male	120-632-00536	Silver Member
10	P843	1997-11-16	Male	132-844-00386	Gold Member
11	P143	1990-03-09	Male	082-273-00836	Silver Member
12	P700	1988-02-19	NULL	083-708-00421	Gold Member
13	P735	1970-12-08	Female	158-362-00848	Bronze Member
14	P40	1983-04-12	Female	154-237-00334	Bronze Member
15	P563	2003-10-21	NULL	113-868-00382	Silver Member
16	P202	2004-01-28	Female	149-938-00635	Bronze Member
17	P554	1987-01-16	Male	191-837-00594	Bronze Member

Medical Table The "Medical" table serves as a repository for detailed patient medical records, enabling healthcare providers to maintain a comprehensive history of patient visits, diagnoses, and test results:

- **Record ID**: Unique identifier for each record (Nominal, Primary Key).
- **Patient ID**: Links records to specific patients (Nominal, Foreign Key).
- **Doctor Name**: Name of the attending physician (Nominal).
- **Visit Date**: Date of the medical visit (Interval).
- **Lab Result**: Test results (Nominal).
- **Diagnosis**: Medical diagnosis (Nominal).
- **Number of Appointments**: Count of appointments (Ratio)

The code snippet and the image show the data generated process for the medical table using python and the dataset for the medical table in SQLite

```
# Record_ID: Nominal Number
```

```
record_number= np.random.choice(range(1001,3001), n, replace=False)

Record_id= [f'R{str(i).zfill(3)}' for i in record_number]

# Lab_Result : Ordinal Data Type

Lab_test= ['Mild', 'Moderate', 'Severe']
Lab_Result= np.random.choice(Lab_test, n, p=[0.3,0.34,0.36])

# Visit_Date : Interval Data Type
Visit_year = np.random.randint(1990, 2023, n)
Visit_month = np.random.randint(1, 13, n)
Visit_day = np.random.randint(1, 29, n)
Visit_Date = [f'{Visit_year[i]}-{str(Visit_month[i]).zfill(2)}-{str(Visit_day[i]).zfill(2)}' for i in range(n)]

# Number_of_appiontment  Ratio Data Type

Number_of_Appointment = np.random.randint(2,20, n)

# Doctor_Name : Nominal Data Type

Doctor= ["Falope", "Johnson", "Oni", "Ojo", "Olabode", "Garcia", "Daniel"]

Doctor =np.random.choice(Doctor, n, p=[0.1,0.1,0.1,0.2,0.1,0.2,0.2])


Doctor_Name= [f'Dr. {np.random.choice(Doctor)}' for i in range (n)]

# Diagnosis : Nominal Table
Diagnosis= ['Common Cold', 'Allergy', 'High Blood Pressure', 'Skin disorders','bipolar disorder']
Diagnosis=np.random.choice(Diagnosis, n, p=[0.2,0.25,0.25,0.15,0.15])


# Heart_Rate (bpm) : Ratio Data Type
Heart_Rate_bpm = np.random.randint(70,91,100)


# Creating a dataframe for the Second table- Medical Table:

Medical_Table = pd.DataFrame({"Record_Id":Record_id,'Patient_Id':Patient_Id,
'Doctor_Name':Doctor_Name, 'Visit_Date':Visit_Date,
                              'Lab_Result':Lab_Result,'Diagnosis':Diagnosis,
'Number_of_Appointment':Number_of_Appointment})
```

Database Structure | Browse Data | Edit Pragmas | Execute SQL

Table: Medical_Table | Filter in any column

	Record_Id	Patient_Id	Doctor_Name	Visit_Date	Lab_Result	Diagnosis	Number_of_Appointment
	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	R2127	P681	Dr. Falope	1998-01-18	Severe	Common Cold	13
2	R1728	P283	Dr. Falope	2021-07-08	Severe	bipolar disorder	17
3	R2320	P994	Dr. Oni	2001-01-04	Moderate	High Blood Pressure	7
4	R2078	P234	Dr. Ojo	1995-11-26	Severe	Common Cold	3
5	R2673	P348	Dr. Ojo	2006-01-17	Moderate	Allergy	18
6	R1401	P189	Dr. Daniel	2019-02-17	Mild	bipolar disorder	5
7	R2134	P620	Dr. Ojo	2022-09-10	Severe	High Blood Pressure	11
8	R1673	P207	Dr. Ojo	2018-06-27	Mild	bipolar disorder	2
9	R2789	P208	Dr. Ojo	2022-03-22	Mild	NULL	16
10	R1920	P571	Dr. Oni	2001-06-10	Severe	High Blood Pressure	5
11	R2060	P696	Dr. Ojo	2002-04-11	Mild	Common Cold	7
12	R2846	P901	Dr. Ojo	2019-03-01	Severe	High Blood Pressure	12
13	R2386	P734	Dr. Ojo	2009-06-18	Moderate	High Blood Pressure	19
14	R2880	P858	Dr. Ojo	1997-08-25	Severe	Allergy	12
15	R1416	P840	Dr. Ojo	1995-04-20	Moderate	High Blood Pressure	3
16	R2809	P549	Dr. Ojo	1990-10-15	Severe	Allergy	7
17	R2698	P855	Dr. Oni	2019-02-01	Severe	High Blood Pressure	11

Treatment Table: This table records essential information about patient treatments, facilitating efficient tracking and analysis of medical interventions

- **Treatment ID**: A unique identifier for each treatment (Nominal).
- **Patient ID**: Links treatments to specific patients (Nominal, Foreign Key).
- **Attending Doctor**: Name of the doctor administering the treatment (Nominal).
- **Treatment Date**: Date when the treatment was administered (Interval).
- **Treatment Type**: Type or category of the treatment (Nominal).
- **Treatment Duration**: Duration or length of the treatment (Interval).
- **Treatment Cost:** Cost associated with the treatment (Ratio).
- **Treatment Outcome:** Outcome or result of the treatment (Ordinal).

The code snippet and the image below show the data generated process for the Treatment table using python and the dataset for the Treatment table in Sqlite

```
# Treatment_Id - Nominal Data
treatment_session_number = np.random.choice(range(7001, 9001), n, replace=False)
Treatment_Session_ID = [f'TX{str(i).zfill(4)}' for i in treatment_session_number]

Attending_Doctor = np.random.choice(Doctor_Name, n, replace=True) # Nominal Data

treatment_year = np.random.randint(2020, 2024, n) # Interval Data Type
treatment_month = np.random.randint(1, 13, n)
treatment_day = np.random.randint(1, 29, n)
Treatment_Date = [f'{treatment_year[i]}-{str(treatment_month[i]).zfill(2)}-{str(treatment_day[i]).zfill(2)}' for
i in range(n)]

Treatment_Types = ['Physical Therapy', 'Chemotherapy', 'Radiation Therapy', 'Surgery', 'Medication']
Treatment_Type = np.random.choice(Treatment_Types, n) # Nominal Data TYpe

Treatment_Duration = np.random.choice([30, 45, 60, 90, 120], n, p=[0.2, 0.2, 0.2, 0.2, 0.2])  # Nominal
Treatment_Cost = np.random.choice([200, 400, 600, 800, 1000], n, p=[0.2, 0.2, 0.2, 0.2, 0.2]) # Ratio
Treatment_Outcome = np.random.choice(['Improved', 'Unchanged', 'Worsened'], n, p=[0.5, 0.3, 0.2]) #
Ordinal

# Creating the DataFrame for the 4th Table
treatment_sessions_data = {
    'Treatment_Id': Treatment_Session_ID,
    'Patient_Id': Patient_Id,
    'Attending_Doctor': Attending_Doctor,
    'Treatment_Date': Treatment_Date,
    'Treatment_Type': Treatment_Type,
    'Treatment_Duration': Treatment_Duration,
    'Treatment_Cost': Treatment_Cost,
    'Treatment_Outcome': Treatment_Outcome
}

# Convert the dictionary to a DataFrame
Treatment_Table = pd.DataFrame(treatment_sessions_data)
```

Table: Treatment_Table

	Treatment_Id	Patient_Id	Attending_Doctor	Treatment_Date	Treatment_Type	Treatment_Duration	Treatment_Cost	Treatment_Outcome
1	TX7465	P681	Dr. Daniel	2022-09-02	Chemotherapy	90	1000	Worsened
2	TX7890	P283	Dr. Olabode	2022-12-17	Chemotherapy	60	800	Improved
3	TX8751	P994	Dr. Ojo	2022-01-14	Chemotherapy	30	1000	Improved
4	TX8601	P234	Dr. Garcia	2020-01-19	Physical Therapy	60	600	Improved
5	TX8009	P348	Dr. Garcia	2022-05-20	Surgery	45	200	Unchanged
6	TX7402	P189	Dr. Olabode	2020-08-07	Chemotherapy	60	400	Unchanged
7	TX7043	P620	Dr. Oni	2023-08-22	Chemotherapy	120	1000	Improved
8	TX8378	P207	Dr. Ojo	2023-12-09	Chemotherapy	90	800	Unchanged
9	TX7913	P208	Dr. Johnson	2021-03-01	Medication	90	1000	Unchanged
10	TX7412	P571	Dr. Olabode	2023-01-15	Medication	45	400	Improved
11	TX7011	P696	Dr. Oni	2021-02-03	Surgery	30	400	Improved
12	TX7747	P901	Dr. Oni	2021-02-07	Radiation Therapy	120	800	Unchanged
13	TX8711	P734	Dr. Ojo	2022-05-15	Chemotherapy	30	200	Worsened
14	TX8698	P858	Dr. Ojo	2021-05-23	Surgery	60	800	Worsened
15	TX8216	P840	Dr. Daniel	2023-11-18	Physical Therapy	120	1000	Improved
16	TX8389	P549	Dr. Garcia	2022-06-14	Medication	30	200	Improved
17	TX8590	P855	Dr. Johnson	2022-01-17	Medication	30	200	Improved

Reflecting Real-World Scenarios With Missing Values And Foreign Keys And Compound Keys

In order to accurately mirror real-world situations, I deliberately introduced missing values into each table within the database. Furthermore, I established primary keys as well as foreign keys, the latter serving as references to other tables in the database. These keys were appropriately set as indices within their respective tables.

The code snippet below shows the code infusion of missing values and the setting of foreign and composite keys in the tables

```
# Setting null values into the data

# Randomly select 101 indices to set to NaN in the Patient_Table
random_indices = np.random.choice(Patient_Table.index, 101, replace=False)

# Set the corresponding values in the "Gender", columns to NaN
Patient_Table.loc[random_indices, 'Gender'] = np.nan



# Randomly select 76 indices to set to NaN in the Medical_Table
random_indices2 = np.random.choice(Medical_Table.index, 98, replace=False)
```

```
# Set the corresponding values in the "Diagnosis" column to NaN
Medical_Table.loc[random_indices2, 'Diagnosis'] = np.nan
```

```
# Setting Primary and Compound Keys to the Table.

# Setting Primary Key for the Patients Table

Patient_Table.set_index('Patient_Id', inplace=True)

# Setting Primary and Compound Keys for the Medical Table

Medical_Table.set_index(['Record_Id','Patient_Id'], inplace=True)

# Setting Primary and Compound Keys for the Treatment Table
Treatment_Table.set_index(['Treatment_Id','Patient_Id'], inplace=True)
```

Database Schema

The multi-table database comprises three main tables: Medical, Patient, and Treatment. In each of these tables, I strategically employed primary, foreign, and compound keys to establish relationships between them. Ensuring data integrity, I implemented constraints on specific columns in each table.

Patient Table:

- **Primary Key:** Patient ID
    - **Description:** The Patient ID serves as the primary key for the Patient Table, acting as a reference point for other tables in the database.
- **Constraints:** Unique and not null constraints are enforced on the Patient ID to ensure data accuracy and completeness.

Medical Table:

- **Primary Key:** Record ID
    - **Description:** The Record ID functions as the primary key for the Medical Table, uniquely identifying each medical record.

- **Constraints:** Unique and not null constraints are applied to the Record ID to maintain data accuracy and completeness.
- **Foreign Key:** Patient ID
    - **Description:** The Patient ID serves as a foreign key in the Medical Table, referencing the Patient Table and the Treatment Table. This establishes a relationship between medical records, patients, and treatments.
- **Constraints:** Unique and not null constraints are applied to the Patient ID, ensuring data integrity.

Treatment Table:

- **Primary Key:** Treatment ID
    - **Description:** The Treatment ID acts as the primary key for the Treatment Table, uniquely identifying each treatment record.
- **Constraints:** Unique and not null constraints are enforced on the Treatment ID to ensure data accuracy and completeness.
- **Foreign Key:** Patient ID
    - **Description:** The Patient ID serves as a foreign key in the Treatment Table, referencing both the Patient Table and the Medical Table. This establishes relationships between treatments, patients, and medical records.
- **Constraints:** Unique and not null constraints are applied to the Patient ID to maintain data integrity.
- **Additional Constraint:** The "Attending Doctor" column features a not null constraint, ensuring that this information is consistently provided, further contributing to data integrity.

The screenshot below illustrates the database schema, displaying foreign and compound keys, as well as the constraints assigned to each table.

Name	Type	Schema
Tables (3)		
Medical_Table		CREATE TABLE "Medical_Table" ("Record_Id" TEXT NOT NULL, "Patient_Id" TEXT NOT NULL UNIQUE, "Doctor_Name" TEXT, "Visit_Date" TEXT, "Lab_Result" TEXT, "Diagnosis" TEXT, "Number_of
Record_Id	TEXT	"Record_Id" TEXT NOT NULL
Patient_Id	TEXT	"Patient_Id" TEXT NOT NULL UNIQUE
Doctor_Name	TEXT	"Doctor_Name" TEXT
Visit_Date	TEXT	"Visit_Date" TEXT
Lab_Result	TEXT	"Lab_Result" TEXT
Diagnosis	TEXT	"Diagnosis" TEXT
Number_of_Appointment	INTEGER	"Number_of_Appointment" INTEGER
Patient_Table		CREATE TABLE "Patient_Table" ("Patient_Id" TEXT NOT NULL UNIQUE, "Date_of_Birth" TEXT, "Gender" TEXT, "Contact" TEXT, "Membership_tier" TEXT, PRIMARY KEY("Patient_Id"))
Patient_Id	TEXT	"Patient_Id" TEXT NOT NULL UNIQUE
Date_of_Birth	TEXT	"Date_of_Birth" TEXT
Gender	TEXT	"Gender" TEXT
Contact	TEXT	"Contact" TEXT
Membership_tier	TEXT	"Membership_tier" TEXT
Treatment_Table		CREATE TABLE "Treatment_Table" ("Treatment_Id" TEXT NOT NULL, "Patient_Id" TEXT NOT NULL UNIQUE, "Attending_Doctor" TEXT NOT NULL, "Treatment_Date" TEXT, "Treatment_Type" TEX
Treatment_Id	TEXT	"Treatment_Id" TEXT NOT NULL
Patient_Id	TEXT	"Patient_Id" TEXT NOT NULL UNIQUE
Attending_Doctor	TEXT	"Attending_Doctor" TEXT NOT NULL
Treatment_Date	TEXT	"Treatment_Date" TEXT
Treatment_Type	TEXT	"Treatment_Type" TEXT
Treatment_Duration	INTEGER	"Treatment_Duration" INTEGER
Treatment_Cost	INTEGER	"Treatment_Cost" INTEGER
Treatment_Outcome	TEXT	"Treatment_Outcome" TEXT
Indices (0)		
Views (0)		
Triggers (0)		

Table Choice Justification

These three tables, when used together, create a comprehensive healthcare dataset that covers patient demographics, medical records, and treatment information. This structured approach to database design allows for efficient data analysis and reporting. The justification for each table is discussed below

Patients Table:

- **Justification**: The "Patients Table" is the foundational table that stores patient demographics and contact details. It contains critical patient information such as patient IDs, date of birth, gender, contact information, and membership tiers. This table enables the identification and differentiation of individual patients across the entire healthcare system. It provides essential context for the other tables and allows for personalized healthcare delivery, communication with patients, and management of membership-related benefits.

Medical Table:

- **Justification**: The "Medical Table" serves as a repository for storing detailed medical records for patients. It includes essential information such as the doctor the patient saw, the visit date, lab results, diagnosis, and the number of appointments associated with each medical visit. This table is invaluable for tracking and maintaining a comprehensive medical history for each patient, enabling healthcare providers to understand the patient's health status over time, make informed medical decisions, and provide continuity of care.

Treatment Table:

- **Justification:** The "Treatment Table" records vital information about the medical treatments administered to patients. It includes details like treatment IDs, patient information, attending doctors, treatment dates, treatment types, durations, costs, and treatment outcomes. This table is essential for monitoring and analyzing the effectiveness of different treatments and their associated costs. It provides insights into the patient's treatment journey and helps healthcare providers make informed decisions regarding future treatments.

Ethics and Data Privacy Considerations

In the development of the 3 data tables, namely the "Medical_Table," "Treatment_Table," and "Patients Table," a paramount concern has been to uphold rigorous ethical standards and data privacy principles as obtainable in the real world. This commitment aligns with the imperative to protect individual privacy and comply with data protection regulations, such as the General Data Protection Regulation (GDPR). Below, we delve into the ethical considerations and privacy safeguards integrated into the design and generation of these tables.

- **Data Anonymization and Privacy Preservation:**
  One of the foundational principles guiding the data generation process of this database was the avoidance of collecting personal information. Instead, I adopted a meticulous approach that ensured data was generated without compromising individuals' privacy. The data collected in this database was anonymized and did not include sensitive personal identifiers, such as names and addresses except personal identifiers such as contact number which serve as a means to communicate with patients when necessary. This deliberate approach served to mitigate the risk of data breaches and protect the confidentiality of patients

- **Use of Random Identifiers:**
  In line with data privacy best practices, patient identifiers, such as 'Patient_Id' were intentionally designed to be random identifiers. These identifiers were used exclusively for identification and tracking purposes within the dataset. They are not directly linked to real individuals, ensuring that patient identities remain undisclosed

Query Sample: Demonstrating Joins and Data Types

In this section of this report, I will present a code snippet and snapshot queries outputs exemplifying the utilization of joins and showcasing various data types within the database. The query addresses the following questions:

1) Retrieve the Patient Id, gender, Membership tier, date of birth , number of appointments of the top 10 patients with the highest treatment cost-?

2. What is total treatment cost by each membership tier and which tier tops the list?

The code snippets and the images below establish connections between the Patients Table, Medical and the Treatment Table, providing valuable insights into data relationships through joins. Furthermore, it highlights the diverse data types present within the hospital Database, including nominal, ratio, and ordinal data types as seen below

```
- Retrive the Patient Id, gender, Membership tier, and date of birth and the number of appointment of the top 10 patients with the highest treatment cost-

SELECT P.Patient_Id, P.Gender, P. Membership_tier, P.Date_of_Birth, M.Number_of_Appointment, T.Treatment_Cost
FROM Patient_Table AS P
INNER JOIN Medical_Table AS M
ON P.Patient_Id = M.Patient_Id
INNER JOIN Treatment_Table AS T
ON P.Patient_Id = T.Patient_Id
ORDER BY 6 DESC
LIMIT 10
```

SQL 1

```
 - Retrive the Patient Id, gender, Membership tier, and date of birth and the number of appointment of the top 10 patients with the highest treatment cost-

SELECT P.Patient_Id, P.Gender, P. Membership_tier, P.Date_of_Birth, M.Number_of_Appointment, T.Treatment_Cost
FROM Patient_Table AS P
INNER JOIN Medical_Table AS M
ON P.Patient_Id = M.Patient_Id
INNER JOIN Treatment_Table AS T
ON P.Patient_Id = T.Patient_Id
ORDER BY 6 DESC
```

	Patient_Id	Gender	Membership_tier	Date_of_Birth	Number_of_Appointment	Treatment_Cost
1	P515	Male	Silver Member	1986-07-03	7	1000
2	P647	Male	Silver Member	1993-09-19	5	1000
3	P91	Male	Gold Member	2000-03-14	11	1000
4	P700	NULL	Gold Member	1988-02-19	16	1000
5	P614	Male	Bronze Member	2005-11-26	15	1000
6	P963	Female	Gold Member	1972-07-05	11	1000

```
Execution finished without errors.
Result: 1000 rows returned in 56ms
At line 3:
SELECT P.Patient_Id, P.Gender, P. Membership_tier, P.Date_of_Birth, M.Number_of_Appointment, T.Treatment_Cost
FROM Patient_Table AS P
INNER JOIN Medical_Table AS M
ON P.Patient_Id = M.Patient_Id
INNER JOIN Treatment_Table AS T
ON P.Patient_Id = T.Patient_Id
ORDER BY 6 DESC
```

```
- What is total treatment cost by each membership tier and which tier tops the list -

SELECT P. Membership_tier, SUM(T.Treatment_Cost) as TOTAL_COST
FROM Patient_Table AS P
INNER JOIN Treatment_Table AS T
ON P.Patient_Id = T.Patient_Id
GROUP BY 1
ORDER BY 2 DESC
```

SQL 1

```
- What is total treatment cost by each membership tier and which tier tops the list -

SELECT  P. Membership_tier,  SUM(T.Treatment_Cost) as TOTAL_COST
FROM Patient_Table AS P
INNER JOIN Treatment_Table AS T
ON P.Patient_Id = T.Patient_Id
GROUP BY 1
ORDER BY 2 DESC
```

	Membership_tier	TOTAL_COST
1	Silver Member	223400
2	Gold Member	201000
3	Bronze Member	180000

```
Execution finished without errors.
Result: 3 rows returned in 44ms
At line 13:
SELECT  P. Membership_tier,  SUM(T.Treatment_Cost) as TOTAL_COST
FROM Patient_Table AS P
INNER JOIN Treatment_Table AS T
ON P.Patient_Id = T.Patient_Id
GROUP BY 1
ORDER BY 2 DESC
```

Conclusion:

In conclusion, the development of a SQL database tailored for hospital use has been successfully achieved, featuring three vital tables: Patient, Medical, and Treatment. The process involved generating random data through Python, ensuring ethical standards and data privacy compliance. Each table's design serves distinct healthcare purposes while maintaining data integrity through primary and foreign keys, along with column constraints. Ethical considerations, such as data anonymization and the use of random identifiers, have been embedded to safeguard individual privacy. The resulting database provides a solid foundation for patient care, comprehensive medical record management, and insightful treatment analysis, making it a robust and ethical solution for healthcare data management in a hospital context.