7PAM2005-0105-2023-Data Mining and Discovery

SQL ASSIGNMENT

PATIENT MANAGEMENT DATABASE

Ashika Mungath

Student Id: 22063061

ABSTRACT:

In the current scenario managing patient data is crucial in decision making and providing high quality medical services. Here a synthetic patient management database is created, employing Python scripting and SQL database management techniques. This report also aims to bring forward the different methods in creation of databases with multiple tables which will contain at least 1000 rows and 7 columns along with different data types including nominal, ordinal, interval, and ratio data. We will demonstrate the implementation of various data type restrictions, as well as the significance of primary keys, foreign keys, and composite keys in maintaining data integrity and establishing relationships among tables within a relational database management system (RDBMS).Through creating different datasets on patient information, medical conditions, patient conditions appointments, this project give emphasize to data integrity and privacy considerations. The database schema design enables efficient data retrieval and analysis for the different needs of healthcare stakeholders. Ethical considerations underline the importance of safeguarding patient privacy and confidentiality.

INTRODUCTION:

This report delves into the process of generating synthetic patient data and designing a robust database schema. With a focus on meticulous attention to detail and adherence to ethical guidelines, the aim is to create a database solution that prioritizes patient confidentiality and data integrity. By examining the ethical implications of data management, the goal is to uphold patient trust while advancing healthcare technology. There will be 4 different tables in the database with records being randomly created within sensible ranges including missing values and duplicates. The tables under this Database will be: -

- **Patients** – which contains details of a patients including patient_id, age, name etc…
- **Medical Conditions-** contains various medical conditions
- **Patient Conditions** – contains records of each patient's conditions
- **Appointments-** Contain each patient's appointment details

Synthetic data generation offers a solution to address the challenges of database development while safeguarding patient privacy. By generating simulated data, developers can test systems and analyse trends without compromising sensitive personal information. This approach not only ensures compliance with privacy regulations but also fosters innovation in healthcare technology.

Through practical examples and thoughtful analysis, the report demonstrates the functionality and usefulness of the database, showcasing its potential to enhance clinical decision-making and patient care. By promoting ethical data management practices and technological innovation, the aim is to contribute to the evolution of healthcare systems that prioritize patient well-being and privacy in the digital era.

DATA GENERATION PROCESS:

The data generation includes the utilization of various Python libraries and tools to create synthetic data of patient Management database. Using these various databases such as patients, medical condition, patient condition and appointments are created. Especially Faker library was employed, which provides functionalities to generate a wide array of fake data, including names, addresses, dates, and more.

Table 1:Patients

```
# Initialize Faker to generate random data
fake = Faker()

# Number of patients and medical conditions
n_patients = 1000
n_conditions = 10

# Generate patient data
patients_data = {
    'PatientID': np.arange(1, n_patients + 1),  # Ratio data (unique identifier for patients)
    'Name': [fake.name() for _ in range(n_patients)],  # Nominal data (patient names)
    'Age': np.random.randint(1, 100, size=n_patients),  # Ratio data (age in years)
    'Gender': np.random.choice(['Male', 'Female', 'Other'], size=n_patients),  # Nominal data (gender)
    'BloodType': np.random.choice(['A', 'B', 'AB', 'O', None], size=n_patients),  # Nominal data (blood type)
    'Height_cm': np.random.normal(170, 10, n_patients),  # Interval data (height in centimeters)
    'Weight_kg': np.random.normal(70, 15, n_patients),  # Interval data (weight in kilograms)
    'Postcode': [fake.postcode() for _ in range(n_patients)],  # Nominal data (postcode)
    'PatientKey': [fake.uuid4() for _ in range(n_patients)]  # Nominal data (unique identifier for patients)
}

# Create a DataFrame from the patient data
patients_df = pd.DataFrame(patients_data)

# Create duplicate values in specified columns
duplicate_columns = ['Name', 'Gender', 'BloodType']
for col in duplicate_columns:
    patients_df[col] = patients_df[col].sample(frac=1).reset_index(drop=True)

# Create an ordinal column for health status
health_status_categories = ['Poor', 'Fair', 'Good', 'Very Good', 'Excellent']
patients_df['Health_Status'] = np.random.choice(health_status_categories, size=n_patients)
```

- **PatientID (Primary Key)**: A sequential range of numbers starting from 1, representing unique identifiers for each patient.
- **Nominal Data**: 'Gender' and 'Blood Type' represent nominal data, as they consist of categories without any inherent order. Gender is randomly chosen from ['Male', 'Female', 'Other'] for each patient and Blood Type is randomly chosen from ['A', 'B', 'AB', 'O', None] for each patient.
- **Ordinal Data:** Ordinal data is created through the *condition_data* array which indicates the condition of each patient in one of the following ['Poor', 'Fair', 'Good', 'Very Good', 'Excellent']
- **Interval Data**: 'Age' can be considered interval data, as it represents a range of values with equal intervals. Similarly, 'Height_cm' and 'Weight_kg' also represent interval data. Age is generated by random integers between 1 and 100, representing the age of each patient. Heights

are normally distributed with a mean of 170 cm and standard deviation of 10 cm.Weights are normally distributed with a mean of 70 kg and standard deviation of 15 kg.

- **Ratio Data**: 'Age', 'Height_cm', and 'Weight_kg' can also be considered ratio data, as they have a true zero point and meaningful ratios between values.
- **Postcode**: Fake postcodes generated using the Faker library.
- **Patient Key**: Unique UUIDs generated for each patient using the Faker library.
- **Duplicate Columns**: The columns 'Name', 'Gender', and 'Blood Type' are shuffled randomly to create duplicates.

Table 2: Medical Conditions

```
# Define medical conditions
medical_conditions = [
    "Diabetes", "Hypertension", "Obesity", "Asthma", "Arthritis",
    "Cancer", "Heart Disease", "Depression", "Migraine", "Allergy"
]

# Generate medical conditions data
conditions_df = pd.DataFrame({
    'ConditionID': np.arange(1, n_conditions + 1),
    'Condition': np.random.choice(medical_conditions, size=n_conditions)
})
```

- List 'medical conditions' defines various health issues like Diabetes, Hypertension, Cancer, etc
- **Condition ID Generation**: ‘Conditions_df' DataFrame structures medical condition data with unique IDs and random selection of conditions. 'ConditionID' serves as the primary key, uniquely identifying each medical condition entry
- 'Condition' column is populated randomly from 'medical_conditions', representing diverse health issues in the dataset.

Table 3: Patient Conditions

```
# Generate patient conditions data
patient_condition_data = {
    'PatientID': np.random.choice(patients_df['PatientID'], size=n_patients),
    'ConditionID': np.random.choice(conditions_df['ConditionID'], size=n_patients)
}
patient_conditions_df = pd.DataFrame(patient_condition_data)
```

- **Condition ID Generation**: For the 'PatientID' column, it randomly selects patient IDs from the 'patients_df' DataFrame using NumPy's random.choice() function similarly 'ConditionID' column, it randomly selects condition IDs from the 'conditions_df' DataFrame using random.choice().
- The dictionary patient_condition_data is used to create a pandas DataFrame named patient_conditions_df, organizing the patient-condition associations.

- A composite key, potentially formed by combining 'PatientID' and 'ConditionID', can serve as the primary key in the patient_conditions_df, ensuring uniqueness in patient-condition associations.
- 'PatientID' and 'ConditionID' serve as foreign keys, respectively referencing the 'PatientID' column in the 'patients_df' DataFrame and the 'ConditionID' column in the 'conditions_df' DataFrame, establishing associations between patient conditions and specific patients as well as medical conditions.

Table 4: Appointments

```
# Generate appointment data
n_appointments = 2000  # Assuming multiple appointments per patient
appointment_data = {
    'AppointmentID': np.arange(1, n_appointments + 1),
    'PatientID': np.random.choice(patients_df['PatientID'], size=n_appointments),
    'Date': [fake.date_this_year() for _ in range(n_appointments)],
    'Time': [fake.time() for _ in range(n_appointments)]
}
appointments_df = pd.DataFrame(appointment_data)
```

- **Appointment ID Generation**: AppointmentID' is generated using NumPy's arange() function, ensuring that each appointment has a unique identifier ranging from 1 to the total number of appointments (n_appointments).
- **PatientID:**'PatientID' is randomly selected from the 'PatientID' column of the 'patients_df' DataFrame. This random selection links each appointment to a specific patient, ensuring diversity in patient appointments.
- 'Date' is generated using the fake.date_this_year() function from the Faker library.
- 'Time' is generated using the fake. Time() function from the Faker library.
- Randomly generated date within the current year, and a random time value representing the appointment time ensures the creation of a diverse dataset reflecting various patient appointments across different dates and times.

Then the code saves patient data, medical conditions, patient conditions, and appointments into CSV files and an SQLite database. It defines tables in SQLite for each data type and provides a function to read data from the database into Pandas DataFrames. Tables include patient details, medical conditions, patient-condition associations, and appointments, ensuring data integrity and relational structure.

DATABASE SCHEMA:

A database schema serves as a plan detailing the logical arrangement of a database, including tables, columns, relationships, and rules for storing and accessing data. It acts as a framework dictating how information is structured and managed within the database environment.

Table: **Patients**

Column Name	Data Type	Constraints
PatientId	INTEGER	PRIMARY KEY
Name	TEXT	NOT NULL
Age	INTEGER	CHECK(AGE>0 AND AGE<150
Gender	TEXT	CHECK(GENDER IN['MALE', 'FEMALE', 'OTHER'])
Blood Type	TEXT	CHECK (BLOOD TYPE IN['A', 'B', 'AB', 'O', NULL])
Height_Cm	REAL	CHECK(HEIGHT IN CM>0)
Weight_Kg	REAL	CHECK(WEIGHT IN CM>0)
Postcode	TEXT	NONE
Patient key	TEXT	UNIQUE

Table: **Medical Conditions**

Column Name	Data Type	Constraints
ConditionID	INTEGER	PRIMARY KEY
Condition	TEXT	NOT NULL,UNIQUE

Table: **Patient Conditions**

Column Name	Data Type	Constraints
PatientID	INTEGER	FOREIGN KEY (PATIENTS)
ConditionID	INTEGER	FOREIGN KEY (MEDICAL CONDITIONS)

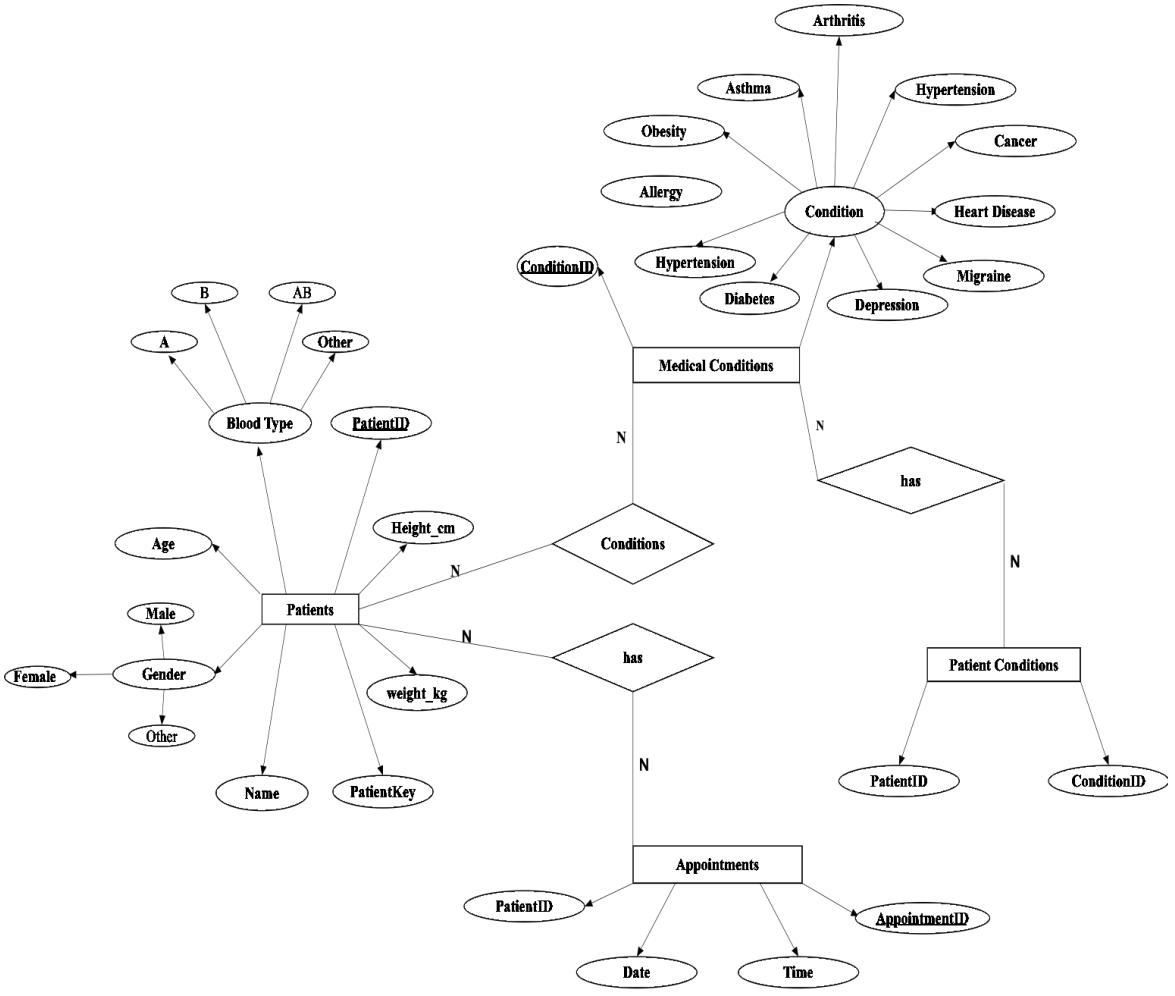
Table**: Appointments**

Column Name	Data Type	Constraints
AppointmentID	INTEGER	PRIMARY KEY
PatientID	INTEGER	FOREIGN KEY (PATIENTS)
Date	TEXT	NOT NULL
Time	TEXT	NOT NULL

Compound Key:

The Combination of Patient_ID and Condition_ID can be used as compound key for retrieving information.

ER diagram:



REPORT JUSTIFICATION:

In the given scenario, the database aims to manage patient information, medical conditions, patient appointments, and their associations.

CHOICE OF TABLES:

1. Patients Table (‘patients’):

   This table is crucial for storing comprehensive patient information, including demographic details such as name, age, gender, and contact information. Additional data such as blood type, height, weight, and a unique patient key can facilitate accurate patient identification and medical history tracking.

2. Medical Conditions Table (`medical_conditions’):
   This table is necessary for maintaining a standardised list of medical conditions. Each medical condition is assigned a unique identifier (`ConditionID`), enabling efficient referencing and data consistency across the database.

3. Patient Conditions Table (`patient_conditions’):
   The patient conditions table facilitates the establishment of relationships between patients and their respective medical conditions. By employing a compound primary key (`PatientID`, `ConditionID`), the table ensures that each patient can be associated with multiple medical conditions, supporting comprehensive healthcare management.

4. Appointments Table (`appointments’):
   The appointments table plays a crucial role in managing patient schedules and healthcare appointments. By storing appointment details such as date, time, and patient ID, healthcare providers can efficiently track patient visits and manage appointment scheduling.

ETHICS AND DATA PRIVACY CONSIDERATIONS:

1. Data Security: Storing patient data, including personally identifiable information (PII) such as names, ages, genders, and medical conditions, requires stringent security measures. The code snippet saves patient data to CSV files and an SQLite database. It's essential to ensure that appropriate security measures are implemented to protect this sensitive information from unauthorized access, both during storage and transmission.

2. Informed Consent: The code generates synthetic patient data using Faker, which includes personal attributes and medical conditions. While this data is synthetic and not tied to real individuals, it's crucial to consider informed consent principles. In a real-world scenario, patients must consent to the collection, storage, and use of their data for specific purposes, such as medical research or healthcare management.

3. Data integrity and Accuracy:Ethical considerations encompass data integrity and accuracy, assured through primary keys and foreign keys, ensuring reliable data representation and linkage across different tables.

4. Sensitive Attributes Handling: Medical conditions, along with other sensitive attributes like gender and blood type, require special consideration regarding privacy and confidentiality. Access controls should be implemented to restrict access to sensitive data only to authorized personnel who need it for legitimate purposes.

5.Missing data:Addressing missing data ethically involves distinguishing intentional omissions from null values within the data record, ensuring transparency and clarity regarding the completeness of information.

6. Ethical Use of Data: Data collected from patients should be used ethically and responsibly, with a focus on improving healthcare outcomes, medical research, or patient care. Any use of patient data should prioritize patient well-being and adhere to ethical guidelines and principles.

DATABASE QUERIES AND OUTPUTS

• Query 1: Retrieve patient Information with health status

SELECT PatientID,Name,Gender,Health_Status FROM patients;

	PatientID	Name	Gender	Health_Status
1	1	Jay Carter	Other	Fair
2	2	Joy Velez	Male	Fair
3	3	Andrew Blevins	Male	Poor
4	4	Melissa Marshall	Other	Very Good
5	5	Christine Kelley	Female	Very Good
6	6	Danielle Smith	Female	Good
7	7	Mark Chavez	Female	Fair
8	8	Jennifer Lee	Male	Good
9	9	Christopher Bush	Male	Excellent
10	10	Stephen Olsen	Other	Fair

• Query 2: Retrieve appointments after a specific date

SELECT * FROM appointments WHERE Date > '2024-03-01';

	AppointmentID	PatientID	Date	Time
1	238	566	2024-03-02	10:15:27
2	399	186	2024-03-02	09:25:30
3	418	813	2024-03-02	02:26:11
4	438	755	2024-03-02	18:12:30
5	444	443	2024-03-02	02:53:45
6	485	82	2024-03-02	12:06:12
7	519	169	2024-03-02	14:37:15
8	534	771	2024-03-02	03:56:33
9	555	99	2024-03-02	01:31:34
10	650	711	2024-03-02	06:06:49

• Query 3: Retrieve patient conditions along with condition names

SELECT p.Name, pc.PatientID, m.Condition FROM patients p

JOIN patient_conditions pc ON p.PatientID = pc.PatientID

JOIN medical_conditions m ON pc.ConditionID = m.ConditionID;

	Name	PatientID	Condition
1	Jay Carter	1	Diabetes
2	Andrew Blevins	3	Depression
3	Andrew Blevins	3	Cancer
4	Andrew Blevins	3	Heart Disease
5	Andrew Blevins	3	Heart Disease
6	Melissa Marshall	4	Depression
7	Christine Kelley	5	Allergy
8	Christine Kelley	5	Diabetes
9	Christine Kelley	5	Heart Disease
10	Danielle Smith	6	Allergy

• Query 4: Count appointments per patient

SELECT PatientID, COUNT(*) AS TotalAppointments

FROM appointments GROUP BY PatientID;

	PatientID	TotalAppointments
1	1	2
2	2	1
3	3	2
4	4	5
5	5	1
6	6	4
7	7	2
8	8	4
9	9	5
10	10	5

• Query 5: Return the number of patients with diabetics

SELECT COUNT(Name) FROM patients WHERE PatientID IN (SELECT PatientID FROM patient_conditions WHERE ConditionID = (SELECT ConditionID FROM medical_conditions WHERE Condition = 'Diabetes'));

	COUNT(Name)
1	101

• Query 6: Find the average age of patients by gender

WITH AvgAgeByGender AS (SELECT Gender, AVG(Age) AS AvgAge FROM patients GROUP BY Gender) SELECT * FROM AvgAgeByGender;

	Gender	AvgAge
1	Female	49.6442577030812
2	Male	49.5451713395639
3	Other	50.6211180124224

CONCLUSION

The script generates synthetic patient data, medical conditions, and appointments, storing them in CSV files, and creates a SQLite database named 'patients_database.db'. It establishes tables for patients, medical conditions, patient conditions, and appointments, enforcing data integrity through primary keys, foreign keys, and check constraints. Finally, it commits changes and closes the database connection.

GitHub Link:https://github.com/ASHIKAMOHAN/creating-database.git