UNIVERSITY of STIRLING

ITNPBD3 - Relational and Non-Relational Databases Assignment

A Relational Database Design of Newtown Doctor's Surgery

Introduction

The healthcare system has critical and complex data pertaining to patient demographics, diagnoses, and treatments; hence it is crucial to maintain and manage the data in an organised manner. The information will also be beneficial in determining a region's specific health requirements, tracking illness trends, and improve service quality. Therefore, the Newtown medical practice has decided to move the data from a spreadsheet to a more manageable database as it provides accuracy, data integrity, and compliance and is flexible and scalable. The report outlines the relational database design and proposals for the healthcare system at Doctor's Surgery, Newtown. While migrating the data, several fields have been added to improve the system's efficiency and versatility. The spreadsheet's historical data is categorised into various tables during the process, including PatientRecords, DoctorRecords, Appointments, Diagnose, and Prescriptions, to streamline the process for the doctors and staff to maintain and manage the records.

The **use cases** that can be identified from the scenario are,

1. **Electronic Medical Records (EMR)** referred to [5]: The healthcare system may use the data such as diagnosis, current/previous medications, recent conditions, and allergies to track a patient's medical history, diagnose conditions, and prescribe medications.
2. **Pandemic Surveillance System** referred to [6]: The database may help track the illness trends and patterns of the registered patients for early evaluation and surveillance of pandemics.
3. **Appointment Reminder System**: The database system may help provide patients with automatic appointment reminders and send prioritised reminders to the staff, which will aid in lowering the chance of no-shows and improve the patient experience.
4. **Doctor's Schedule & Performance Tracking**: The database can be used to manage doctors' schedules, appointments, and holidays, ensuring that doctors are available to see patients when they are needed. Furthermore, the database can be modified in the future to track a doctor's performance by including patient satisfaction feedback and medical outcomes to assess and enhance their expertise.
5. **GP & Pharmacy Management**: Patient, doctor, and diagnosis information management may aid in delivering prompt medical care to patients by the GP/hospital. In addition, pharmacies may utilise the database to manage prescription information and fill prescriptions precisely and securely.

From the scenario, the following **business rules** can be determined,

1. **The names of the patient and doctor are stored in the tables as FullName** in a single column to prevent the likelihood of data duplication, ensuring data integrity, and facilitate searches, mainly when several tables are related to one another.
2. A join table with the foreign keys PatientID and doctor's GMCNumber as composite primary keys have been used to enforce that **a patient may register with more than one doctor and vice versa**. This ensures a many-to-many relationship.
3. A patient may visit a doctor who is or is not registered with them and is indicated by the field **"IsPatientRegWithDoc,"** which is automatically stored in the database as a Boolean value while booking appointments as a part of patient records management/tracking.

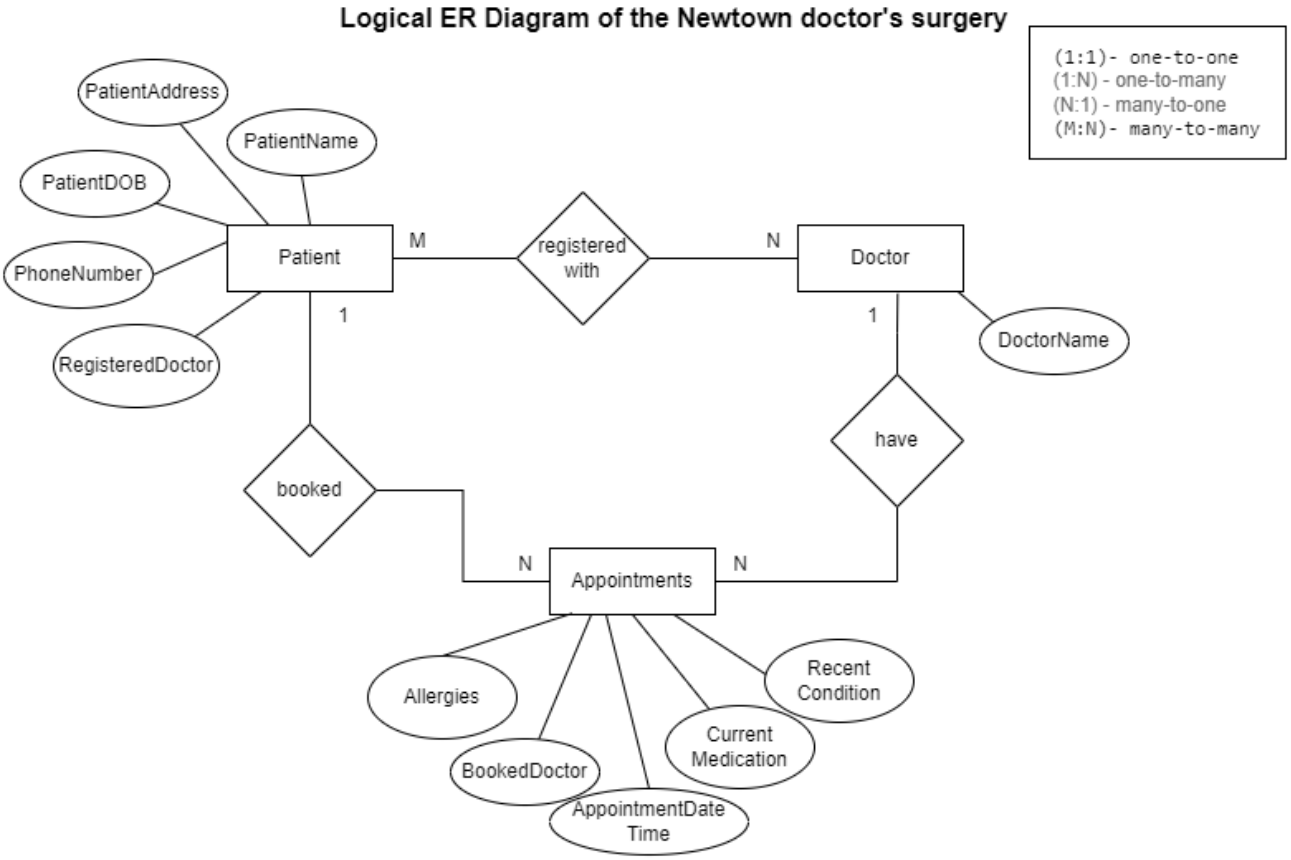
4. **Patients with the same name and contact information** but different birthdates are regarded as two different accounts.
5. **A unique identification has been provided** for each record in the system to ensure that duplicate records are avoided and are identified in the system.
6. The database has been designed to document the **patient's complete address for efficient communication, & care management** and guarantees precise and comprehensive patient identification.
7. **Email addresses are optional for patients and doctors**, with phone numbers as the primary communication means. Moreover, numerous patients can use the same email address and phone number, so it has yet to be considered a unique identification in this scenario.

Logical ER Diagram

The essential representation of the data has been visualised in Chen's diagram below, which gives an abstract view of the basic entities, their attributes and relationships based on the business requirements. Further, in the data modelling stage, it will be decomposed to deliver a schema with a comprehensive database blueprint.

The entities identified initially from the historical data are **Patients, Doctors, and Appointments**. The relationships are defined as,

- A patient can have many appointments; however, one appointment should belong to one patient.
- A doctor can have many patients registered with them and vice-versa.
- A doctor can have many appointments; however, one appointment only belongs to one doctor.

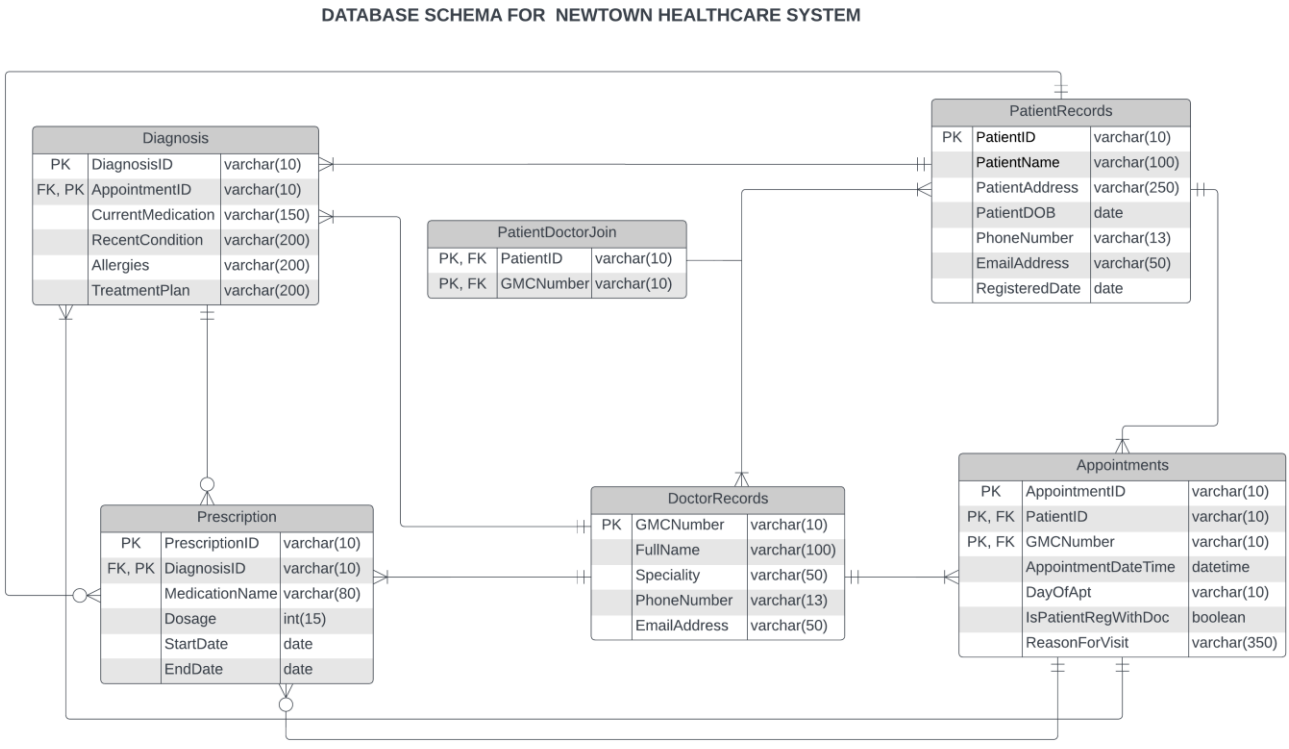


Database Schema

The rules have been defined, and the database schema has been illustrated below with tables, fields, relationships, and keys. These components have been designed in such a way that they can evolve in accordance with business requirements.

- **Additional fields have been added to the patient entity**, including the registered date of the patients for administrative purposes and the email address as an optional and alternative contact. **The doctor entity** has also been provided with new fields, such as contact information and specialisation, and the **GMC reference(license) number** acts as the primary key.
- Given the **many-to-many relationship** between the patient and doctor entities, 3NF has been achieved by creating a join table with composite primary keys that included foreign keys from both tables.
- The appointments entity has been structured into **three tables – Appointments, Diagnosis and Prescription**- to avoid anomalies. Each has been associated with the primary keys of the patient and doctor and the respective tables.
- The **diagnosis table contains** the patient's medical history, which a doctor can modify according to the diagnosis held within the appointment.
- **The "IsPatientRegWithDoc"** field has been added to the appointment table to help determine whether the patient is seeing a registered doctor, as a patient can visit a doctor who isn't registered with them.

The tables, with fields, keys, data types, and relationships, have been demonstrated using the diagram below,



Furthermore, the assumptions made are,

- The join table depicts how a patient may be registered with one or more doctors and vice versa.
- One patient can have multiple registered doctors, diagnoses, appointments, and (none or many) prescriptions.

- One doctor can have multiple patients, appointments, and diagnoses and write (none or many) prescriptions.
- One appointment can only be associated with one doctor and one patient.
- One diagnosis can only be associated with one appointment and one patient.
- One prescription can only be associated with one diagnosis.
- One diagnosis may contain (none or many) prescriptions during the primary appointment or follow-up.

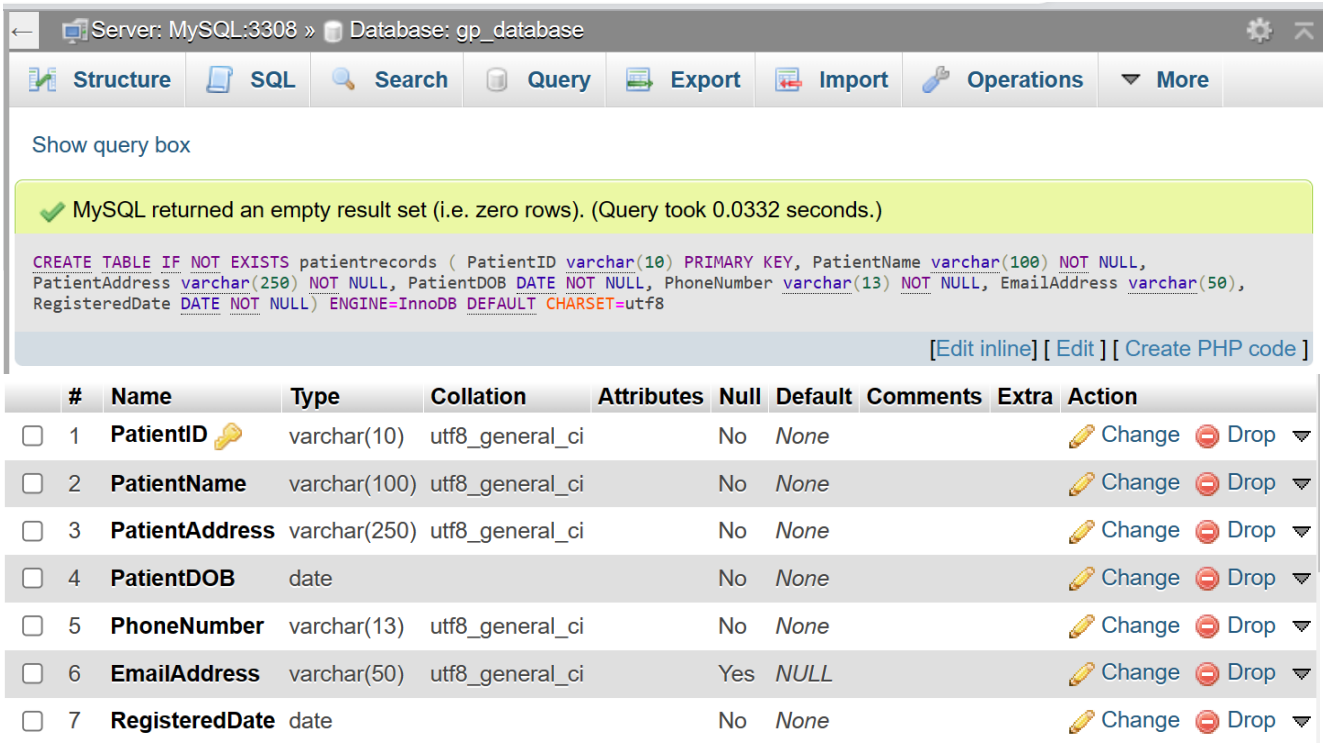
The other entities introduced to the database aim to organise the data and make it more accessible without changing the meaning of the data. Moreover, with this design, each table is intended to have a single atomic value (the primary key), be free of recurring fields that could cause duplication or redundancy, and not be transitively dependent on non-key fields. **The schema and the tables have thus achieved the Third Normal Form(3NF)**.

Create Tables

The database has been created with the name "gp_database", and the queries that have been used to create the tables are the following. The CREATE TABLE query was used to create the following tables, which have data types of varchar() for alphanumeric and special characters, int() for integers, boolean or tinyint() for boolean values, and DATE & DATETIME for entering date and time, respectively. Characters have been set to UFT-8 encoding. The tables now contain FOREIGN KEYs and composite PRIMARY KEYs that were added using the ALTER TABLE query.

PatientRecords Table:

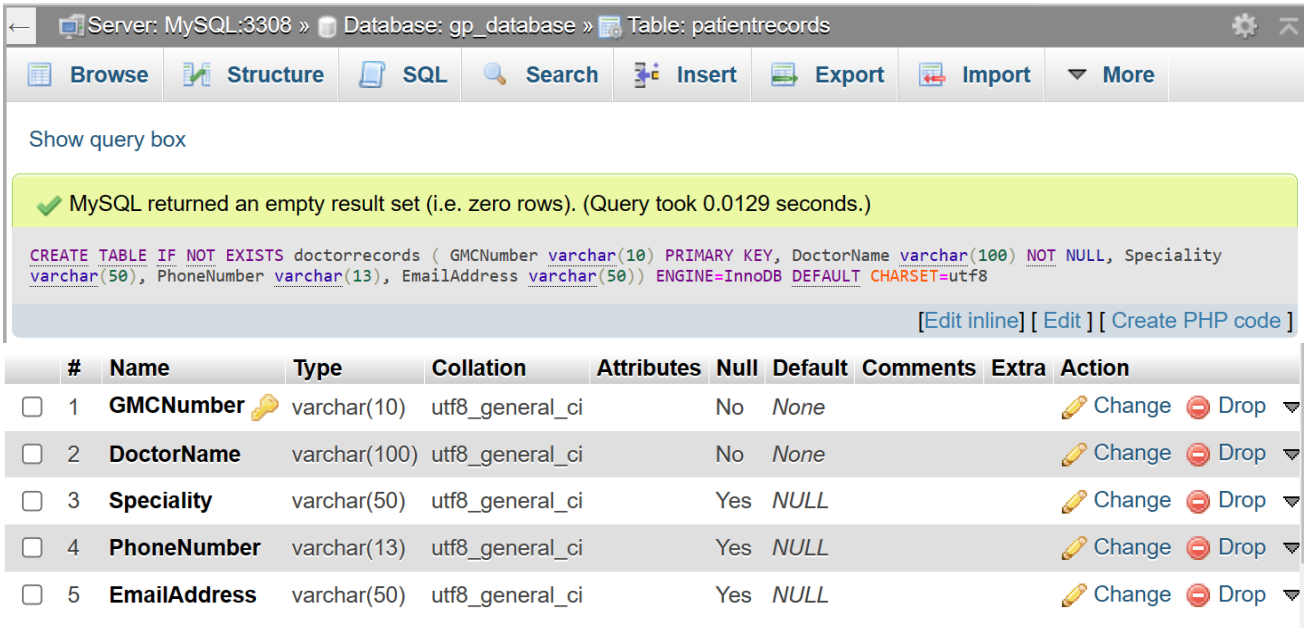
PatientID serves as the PRIMARY KEY in the patient records table, which has been constructed to store essential information regarding patients. Except for EmailAddress, all attributes have been set to NOT NULL since patient data is critical to the healthcare system.



#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	**PatientID**	varchar(10)	utf8_general_ci		No	*None*			Change Drop
2	**PatientName**	varchar(100)	utf8_general_ci		No	*None*			Change Drop
3	**PatientAddress**	varchar(250)	utf8_general_ci		No	*None*			Change Drop
4	**PatientDOB**	date			No	*None*			Change Drop
5	**PhoneNumber**	varchar(13)	utf8_general_ci		No	*None*			Change Drop
6	**EmailAddress**	varchar(50)	utf8_general_ci		Yes	*NULL*			Change Drop
7	**RegisteredDate**	date			No	*None*			Change Drop

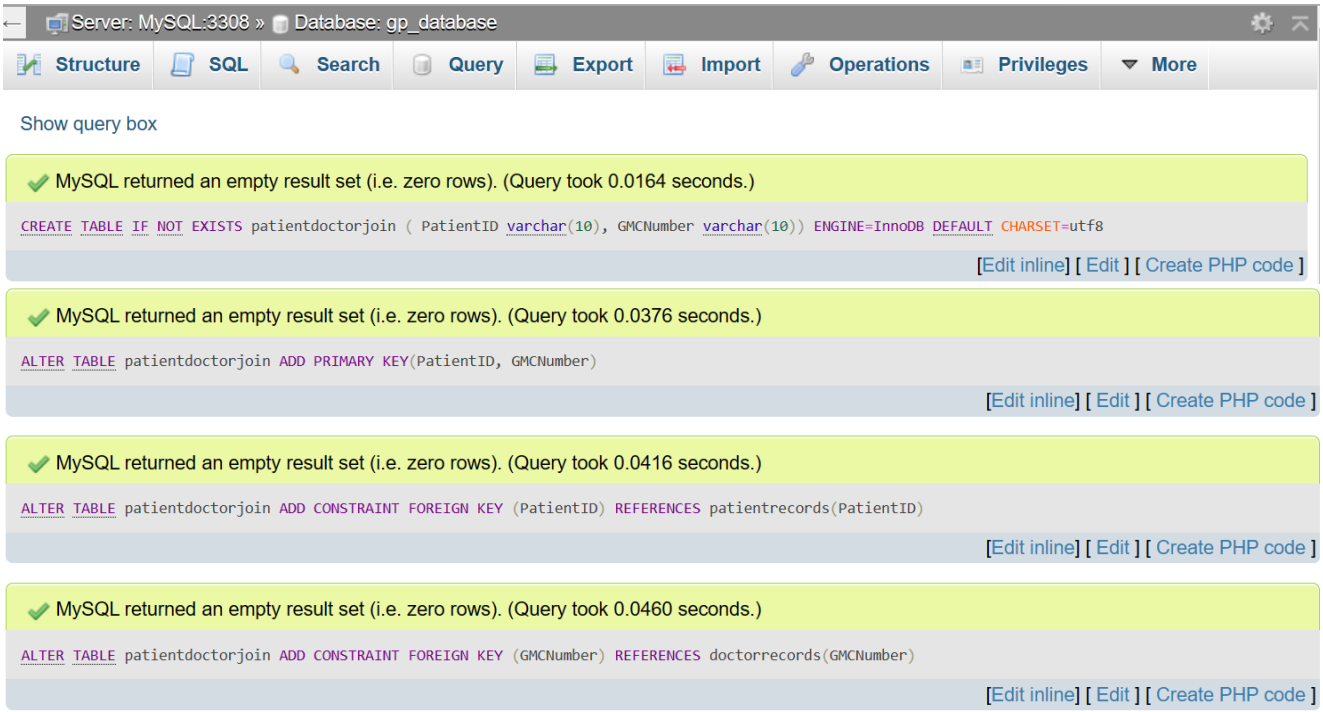
DoctorRecords Table:

The doctor records have been designed to hold the information of GPs in the surgery, much as the patient records. The primary key in this table is the GMC Reference (license) number. Furthermore, additional data like contact and specialisation information has been obtained to ensure compliance and patient care coordination.



PatientDoctorJoin table:

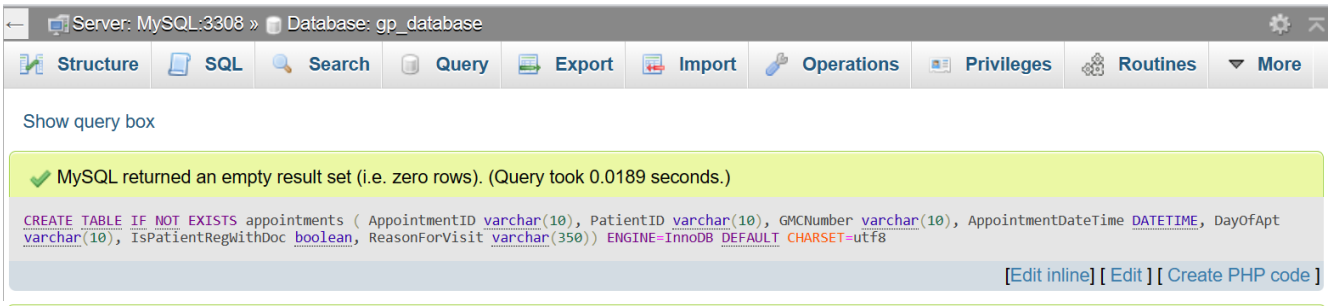
In compliance with the business rule that each doctor may have one or more patients registered to them and each patient may have one or more registered doctors, a joint table of the patient and doctor has been developed to avoid anomalies in this case. The PatientID and GMCNumber are the composite keys which are the foreign keys from PatientRecords and DoctorRecords table.

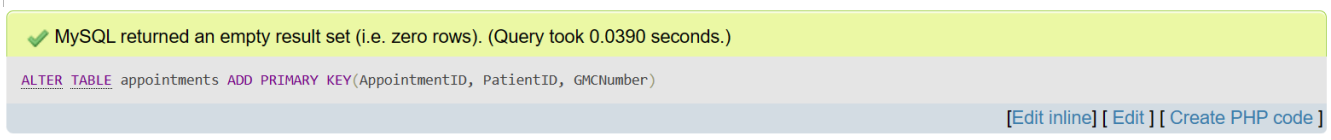


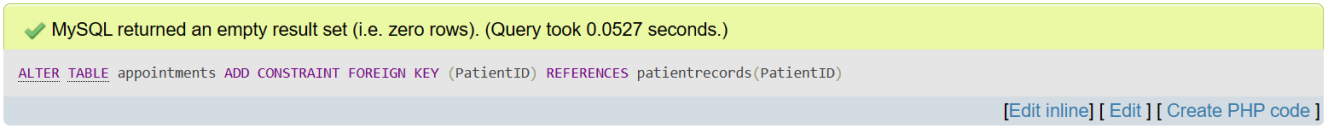
#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	**PatientID**	varchar(10)	utf8_general_ci		No	*None*			Change Drop More
2	**GMCNumber**	varchar(10)	utf8_general_ci		No	*None*			Change Drop More

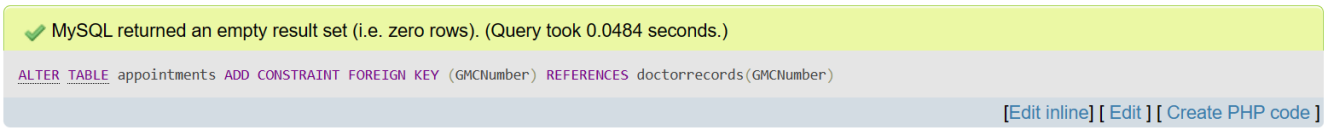
Appointments table:

The appointments table, which comprises fields for recording the date, day, and time of the appointment and the purpose for a visit, has been designed to update the details of an appointment booked by the patient and to facilitate managing patient visits. Along with the AppointmentID, the foreign keys PatientID and GMCNumber are added as a composite primary key.





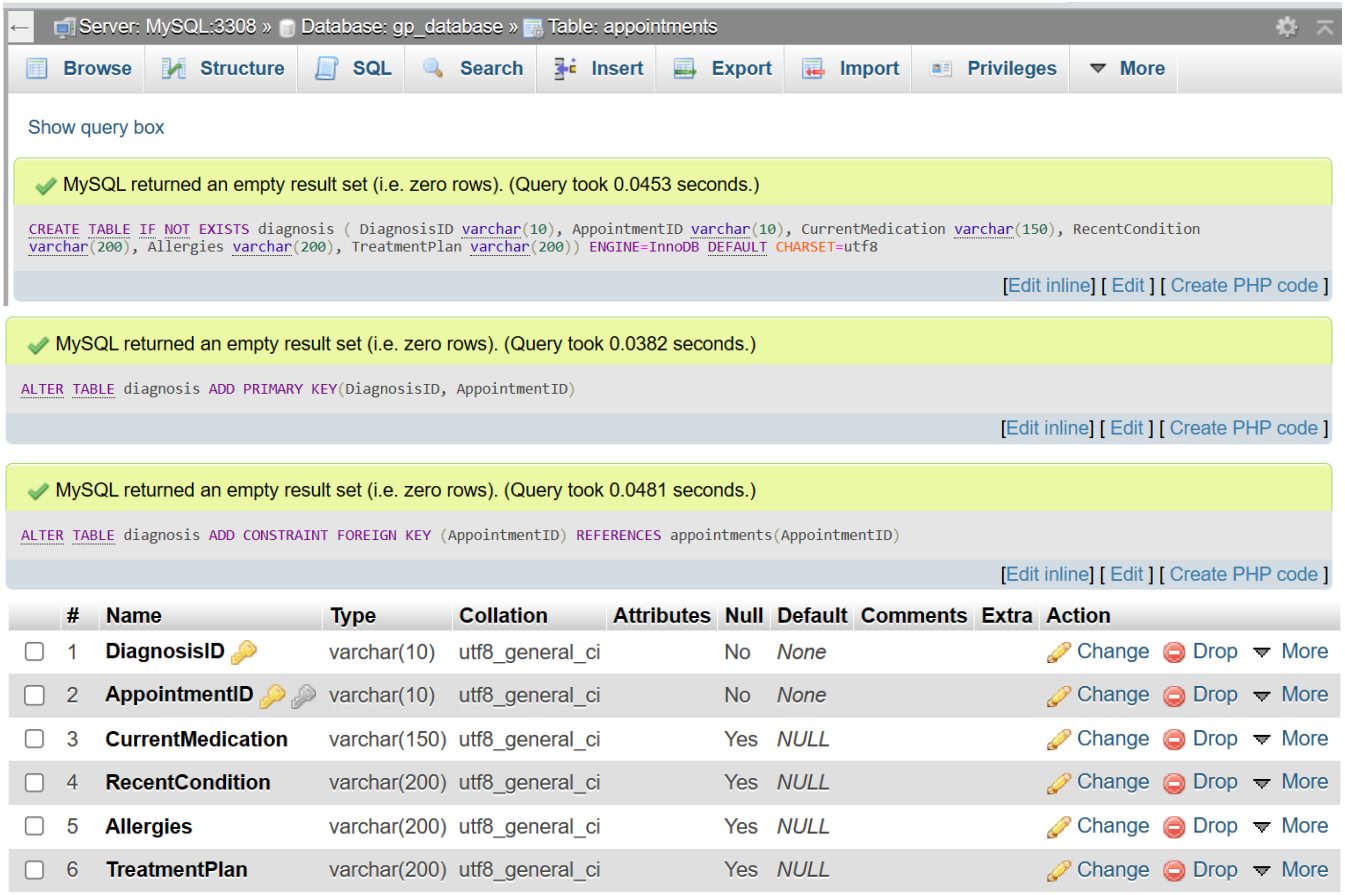




#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	**AppointmentID**	varchar(10)	utf8_general_ci		No	*None*			Change Drop More
2	**PatientID**	varchar(10)	utf8_general_ci		No	*None*			Change Drop More
3	**GMCNumber**	varchar(10)	utf8_general_ci		No	*None*			Change Drop More
4	**AppointmentDateTime**	datetime			Yes	*NULL*			Change Drop More
5	**DayOfApt**	varchar(10)	utf8_general_ci		Yes	*NULL*			Change Drop More
6	**IsPatientRegWithDoc**	tinyint(1)			Yes	*NULL*			Change Drop More
7	**ReasonForVisit**	varchar(350)	utf8_general_ci		Yes	*NULL*			Change Drop More

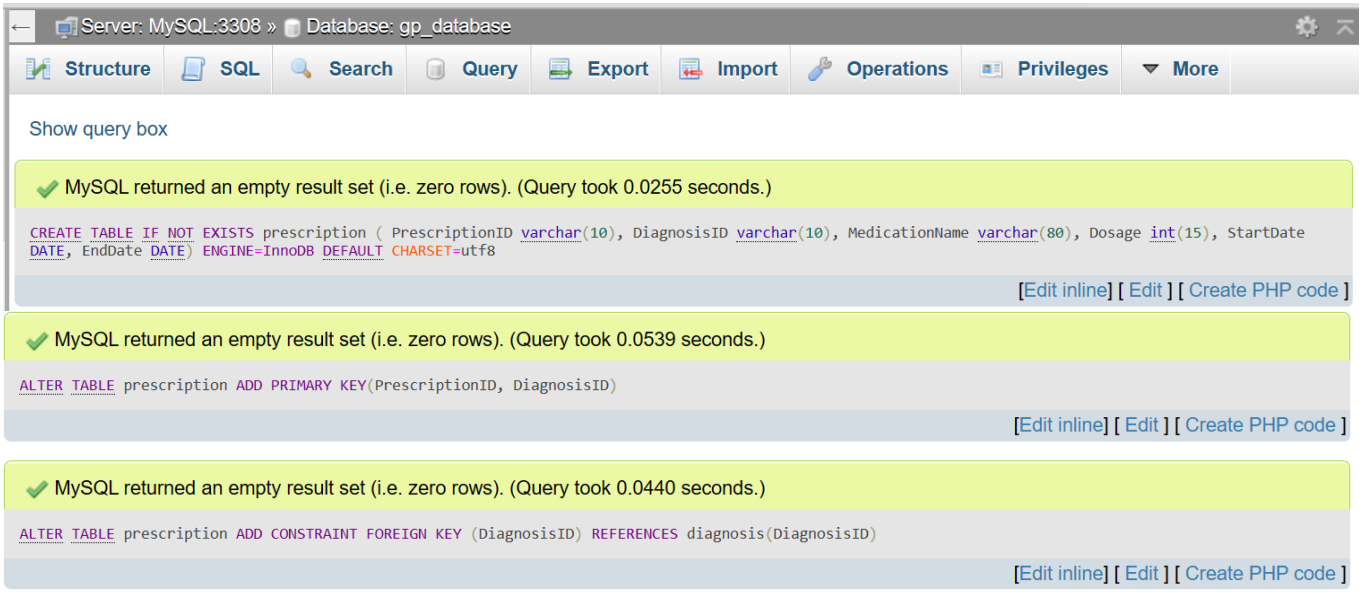
Diagnosis table:

The diagnosis table intends to keep track of patient diagnosis data following the appointment. The doctor may modify this section, and it has a composite primary key of DiagnosisID and AppointmentID (the foreign key from the Appointments table). The table now includes a field called TreatmentPlan, which also aids with patients' appointment history. The fields can be empty or NULL, depending on the treatment the doctor has suggested to each patient.



Prescription table:

The prescription table provides information about the prescriptions based on the appointment and the diagnosis. This section may be modified by the doctor during the appointments. It has PrescriptionID and DiagnosisID as composite primary keys, where DiagnosisID is a foreign key from the Diagnosis table.

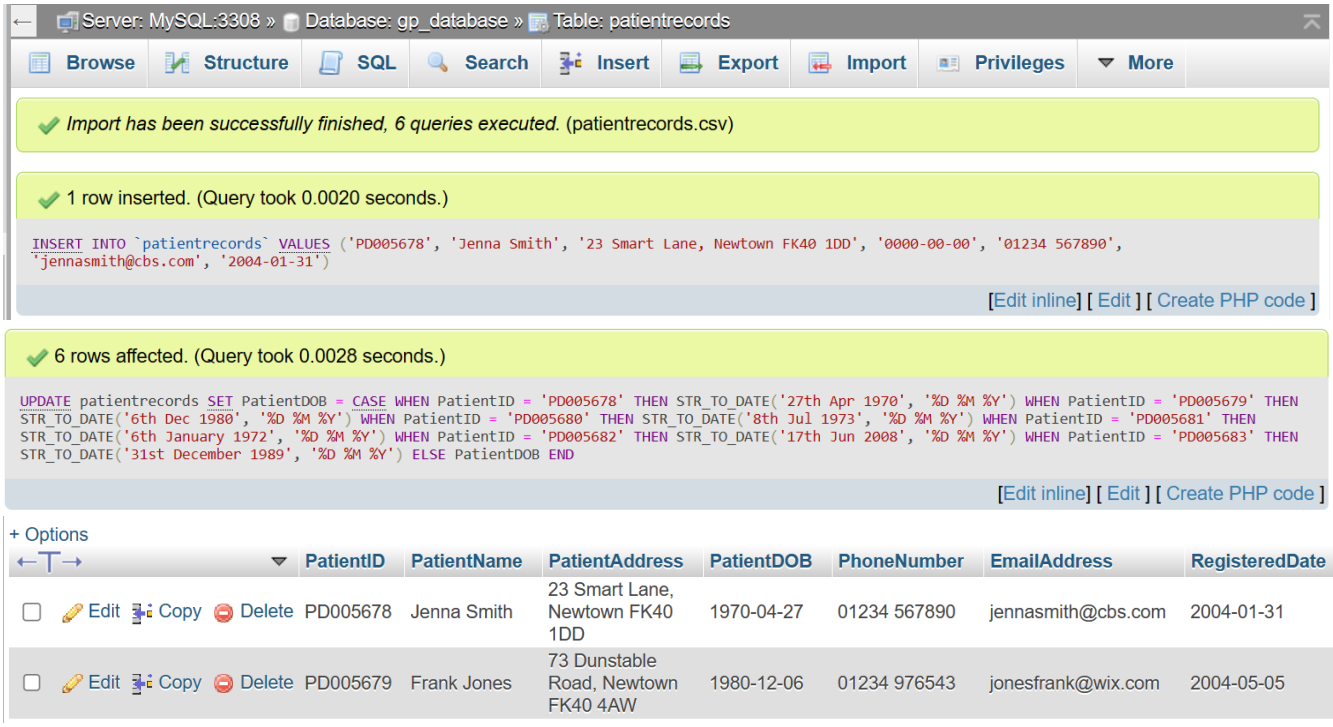


#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	**PrescriptionID**	varchar(10)	utf8_general_ci		No	*None*			Change Drop More
2	**DiagnosisID**	varchar(10)	utf8_general_ci		No	*None*			Change Drop More
3	**MedicationName**	varchar(80)	utf8_general_ci		Yes	*NULL*			Change Drop More
4	**Dosage**	int(15)			Yes	*NULL*			Change Drop More
5	**StartDate**	date			Yes	*NULL*			Change Drop More
6	**EndDate**	date			Yes	*NULL*			Change Drop More

Insert the Data

In the PatientRecords table:

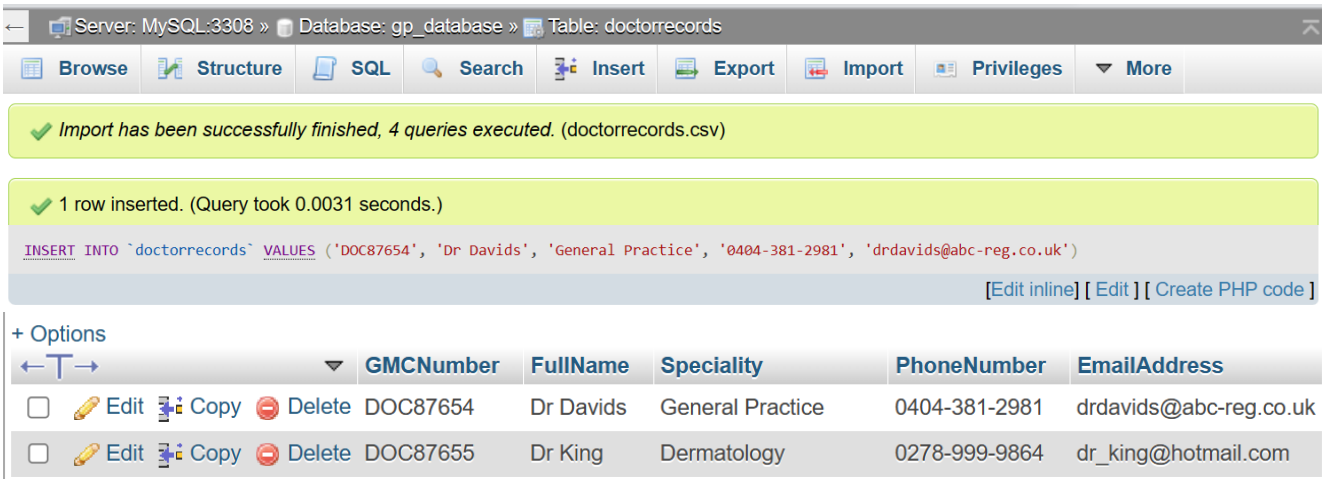
Only information pertinent to the patient records table was pulled out of the data from the Excel spreadsheet and modified. Given the relevance of the information and the wrong date format, the birth date field was subsequently extracted from the spreadsheet and processed independently to prevent any errors or missing data in the database. With the STR_TO_DATE method, which obtains the date in the format %D%M%Y and converts it to the default format in MYSQL, the Birthdate of the patients has been updated given the PatientID, using the UPDATE query with CASE WHEN-THEN statements.



	PatientID	PatientName	PatientAddress	PatientDOB	PhoneNumber	EmailAddress	RegisteredDate
Edit Copy Delete	PD005678	Jenna Smith	23 Smart Lane, Newtown FK40 1DD	1970-04-27	01234 567890	jennasmith@cbs.com	2004-01-31
Edit Copy Delete	PD005679	Frank Jones	73 Dunstable Road, Newtown FK40 4AW	1980-12-06	01234 976543	jonesfrank@wix.com	2004-05-05

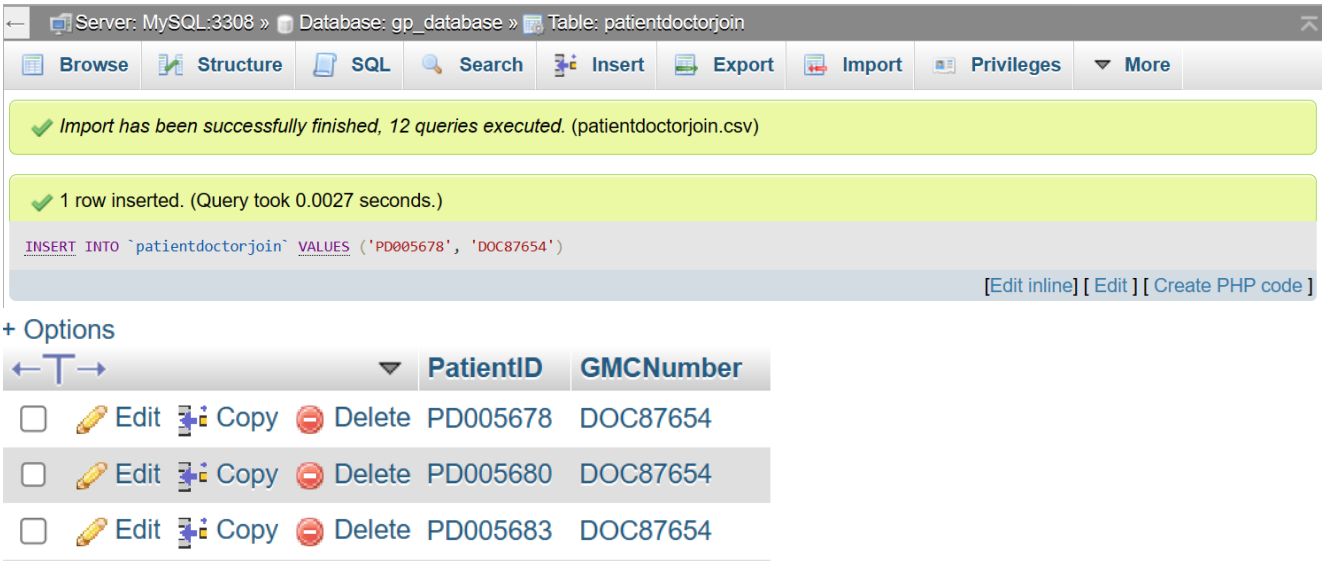
In the DoctorRecords table:

All the records have been imported in the form of a CSV file. The historical spreadsheet was used to extract the doctors' names, and the general records were used to gather additional data.



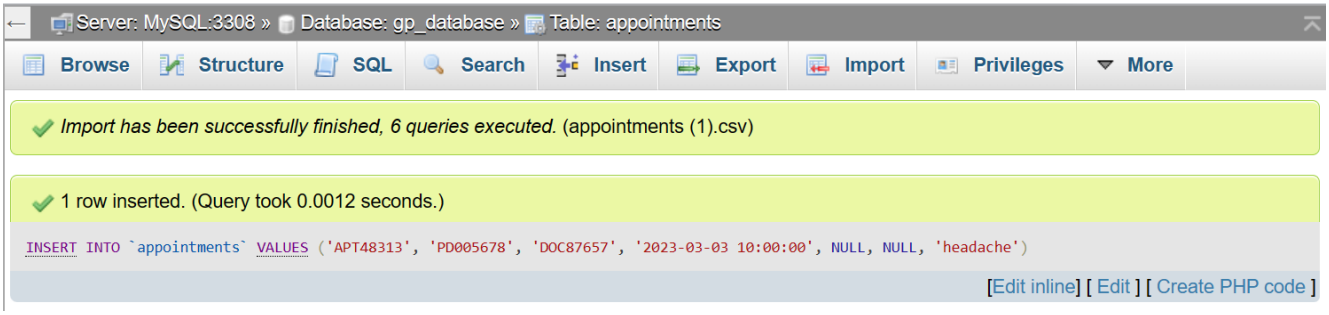
In the PatientDoctorJoin table:

Data has been prepared and imported as a CSV file with the PatientID and their appropriate GMCNumber.



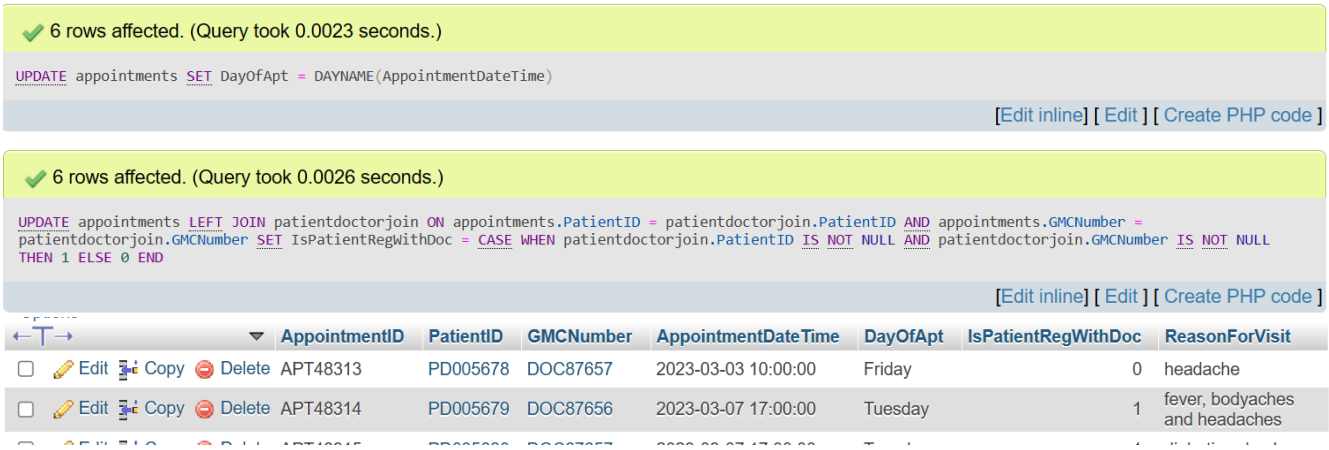
In the Appointments table:

The data has been imported from the CSV file extracted from the historic spreadsheet.



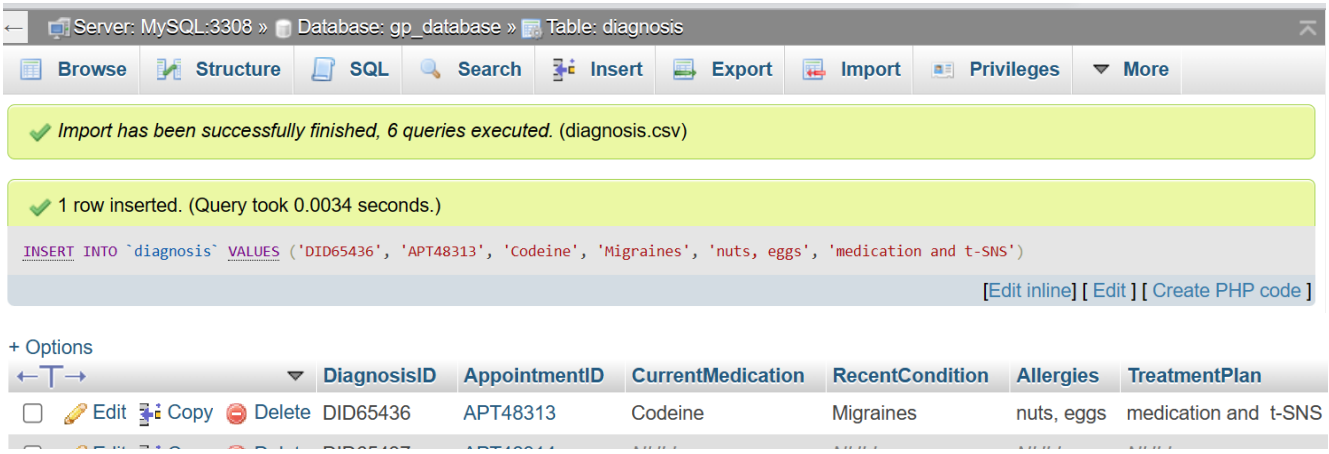
The DAYNAME() function, referred from [2], automatically produces the name of the weekday based on the AppointmentDateTime field and has been used to update the DayOfAppointment column.

In addition, if the doctor (the GMCNumber in the table) for whom the patient has made the appointment is registered with the patient or not, **the "IsPatientRegWithDoc"** function will automatically return TRUE or FALSE (1 or 0) using SET & CASE WHEN statements & to search the table the LEFT JOIN is used as it joins the appointments table with patientdoctorjoin table, which means all the rows in the appointments will be taken into account even if there isn't a matching row in patientdoctorjoin. [3] has been referred for CASE WHEN statements.



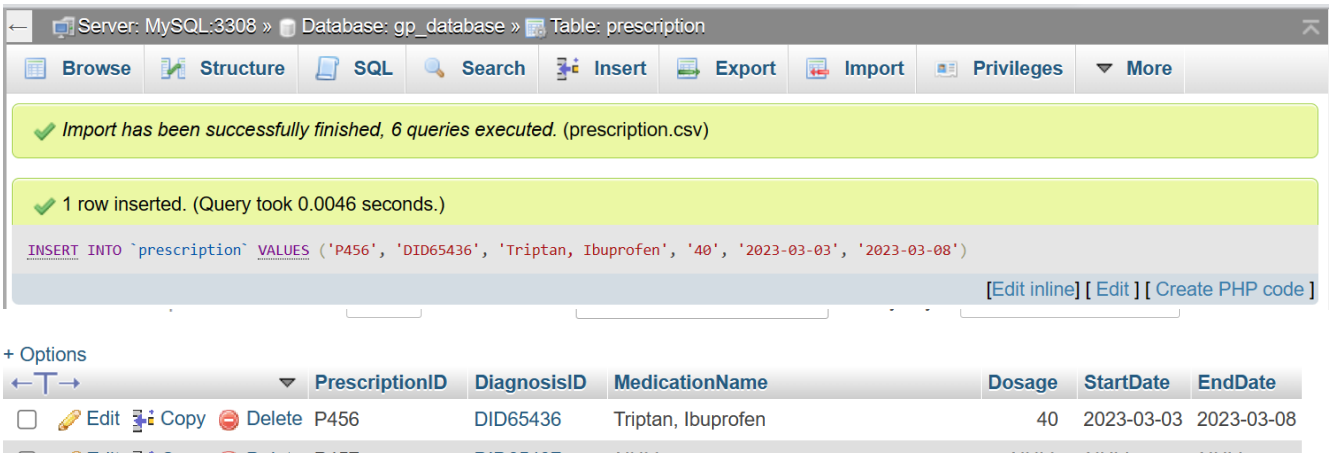
In the Diagnosis table:

The data has been imported to the table using a CSV file, which was extracted from the historical spreadsheet with an additional column treatment plan. The table will help doctors and staff manage and maintain the patient's medical history and can be modified by the doctor during the appointment.



In the Prescription table:

The data has been imported using a CSV file, and if the diagnosis does not have any values in a record, the prescription can be set to null/not null based on the doctor's analysis. The PrescriptionID and DiagnosisID (from the diagnosis table) act as the composite primary key for the table.



Suggestions

In future work, a **UI application for web-based forms** using JavaScript or HTML could be developed, where patients or staff (on behalf of patients) could book appointments using appointment forms, and doctors could modify or create a patient's history form with diagnosis and other information that would be stored in the database.
Furthermore, using HTML referred [7], modifications to the design, such as "**checkboxes for multiple selection and radio button for single selection**", can be created in the UI for fields such as "allergies" and so on, where the values selected by the individual will be passed to the server or database.

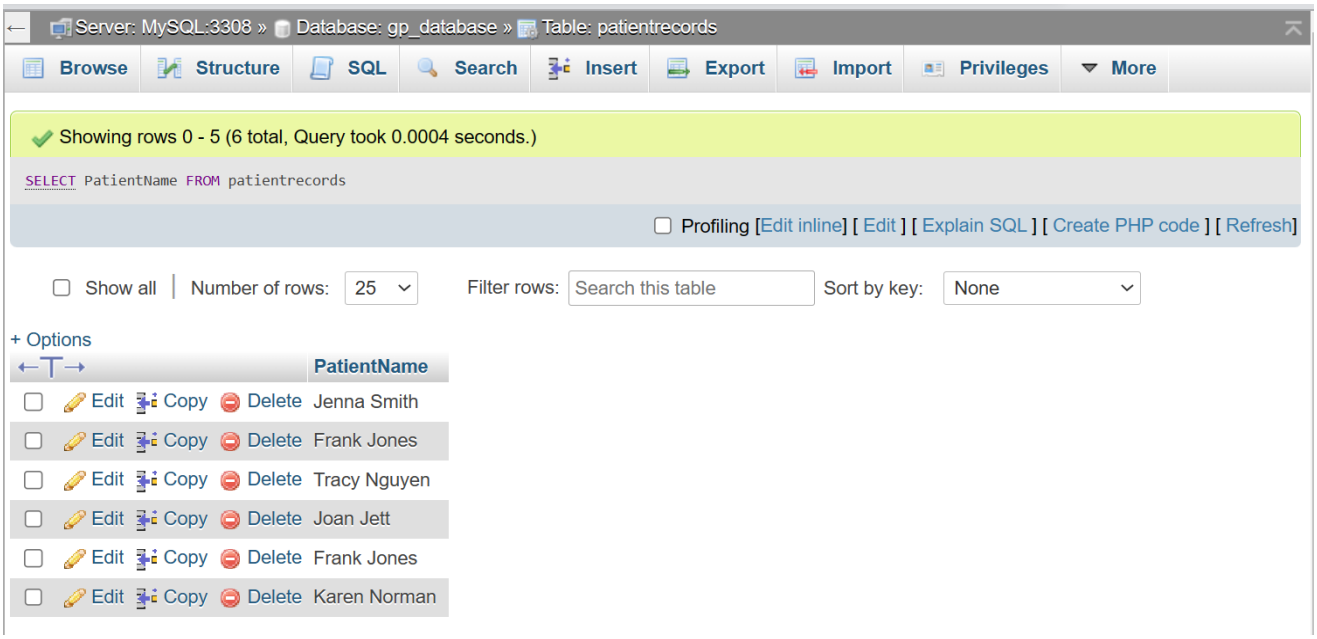
```
<html>
	<body>
		<label for="allergies"> <strong>Allergies: </strong></label><br>
		<input type="checkbox" id="allergy1" name="allergies" value="Nuts">
		<label for="allergy1">Nuts</label><br>
		<input type="checkbox" id="allergy2" name="allergies" value="Eggs">
		<label for="allergy2">Eggs</label><br>
		<input type="checkbox" id="allergy2" name="allergies" value="Shellfish">
		<label for="allergy2">Shellfish</label><br>
		<input type="checkbox" id="allergy3" name="allergies" value="Penicillin">
		<label for="allergy3">Penicillin</label><br>
	</body>
</html>
```

'Input type' is used to place the checkboxes in front of each label, and 'label' is used to write the names of allergies.

SQL Queries

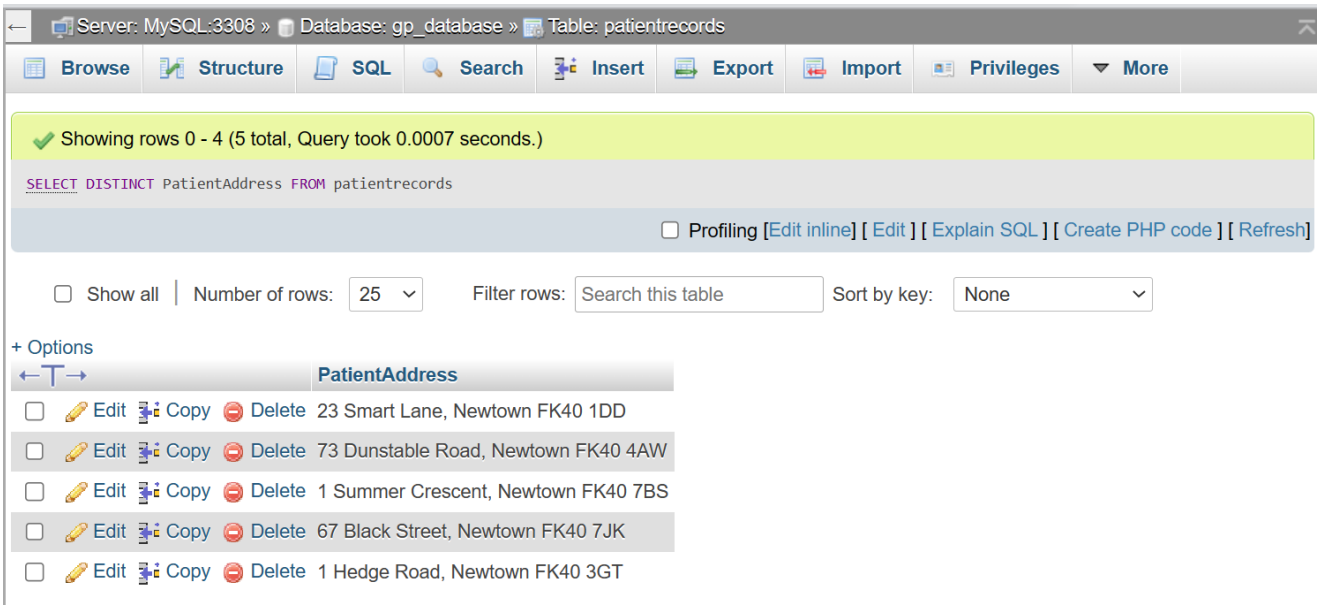
Return a list of all patient names.

The SELECT statement will list all the patients from the table patientrecords.



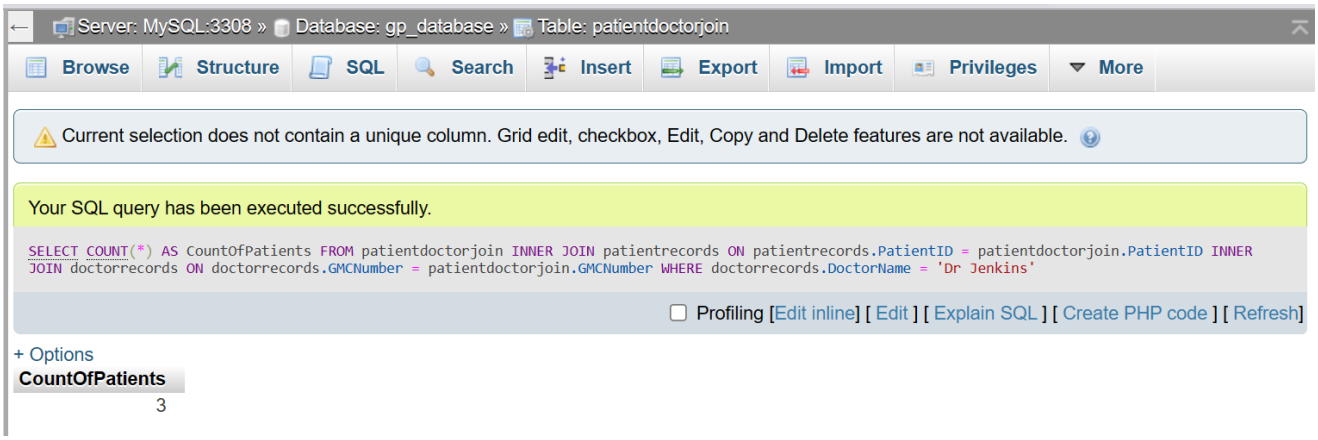
Return a list of all patient addresses, showing each address only once.

The SELECT DISTINCT statement will list all the unique patient addresses from the table patientrecords.



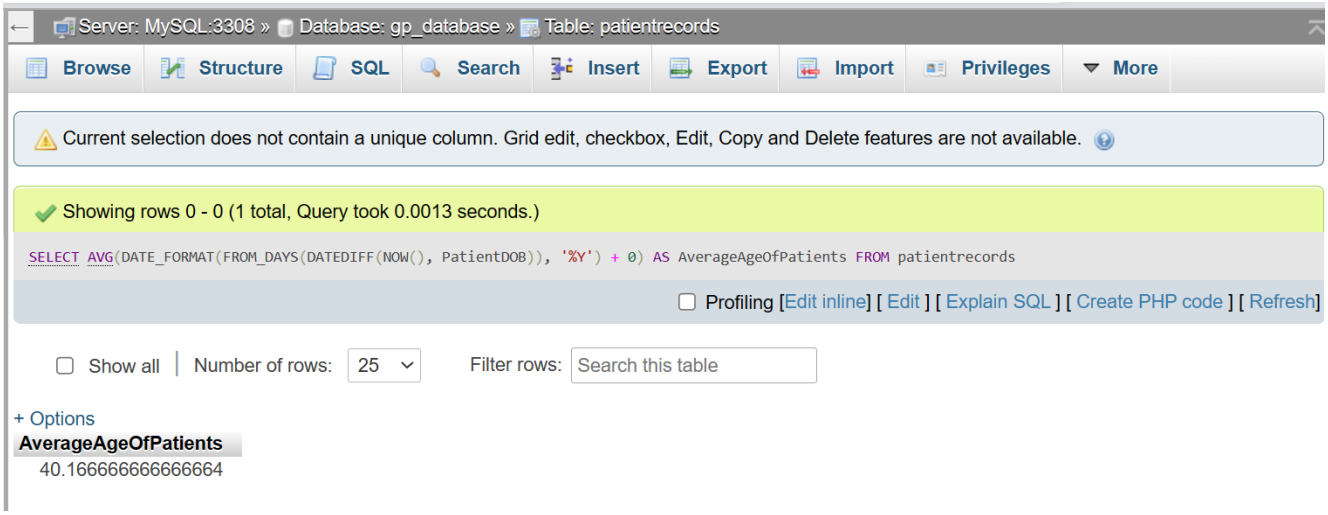
Write a query to count how many patients have Dr Jenkins as one of their registered doctors.

The COUNT() function will provide the total number of patients, but the INNER JOIN joins the patient and doctor tables based on common columns and picks any entries that match both tables, and the WHERE statement has been used to filter the records based on DoctorName matching Dr Jenkins.



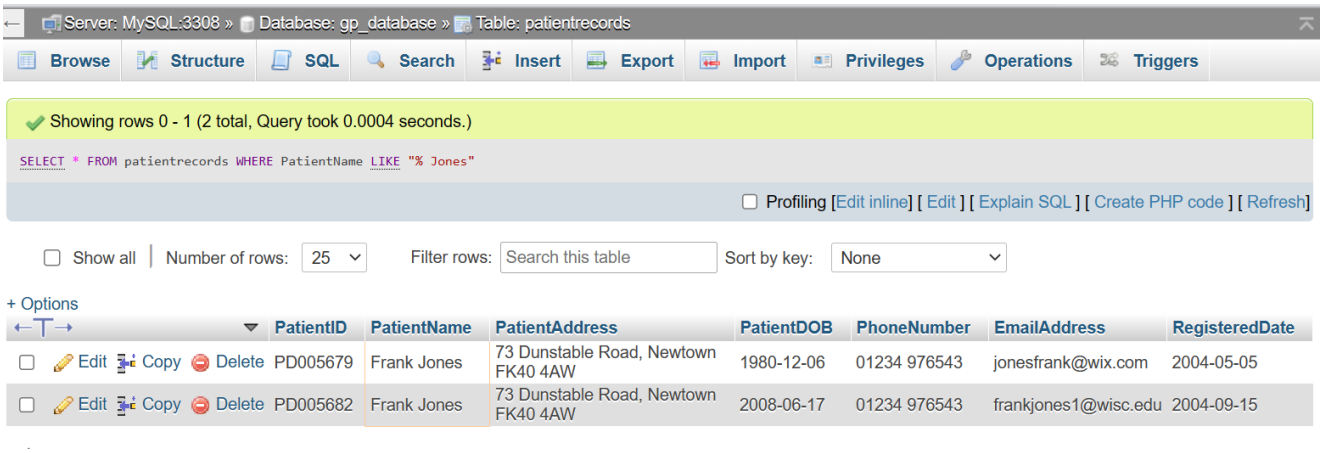
Calculate the average age of all patients.

The average age of all the patients is - 40.17. The difference between the date of birth and the current date was calculated with DATEDIFF(NOW(), PatientDOB), yielding the number of days, which was then converted to a date value with the FROM DAYS() function. Finally, DATE FORMAT() with Y% is used to find the year, which gives the age of each patient; a "+0" is added at the end to convert it to a numeric function, and the age average is calculated using the AVG() function. [1] has been referred to find the age of the patients.



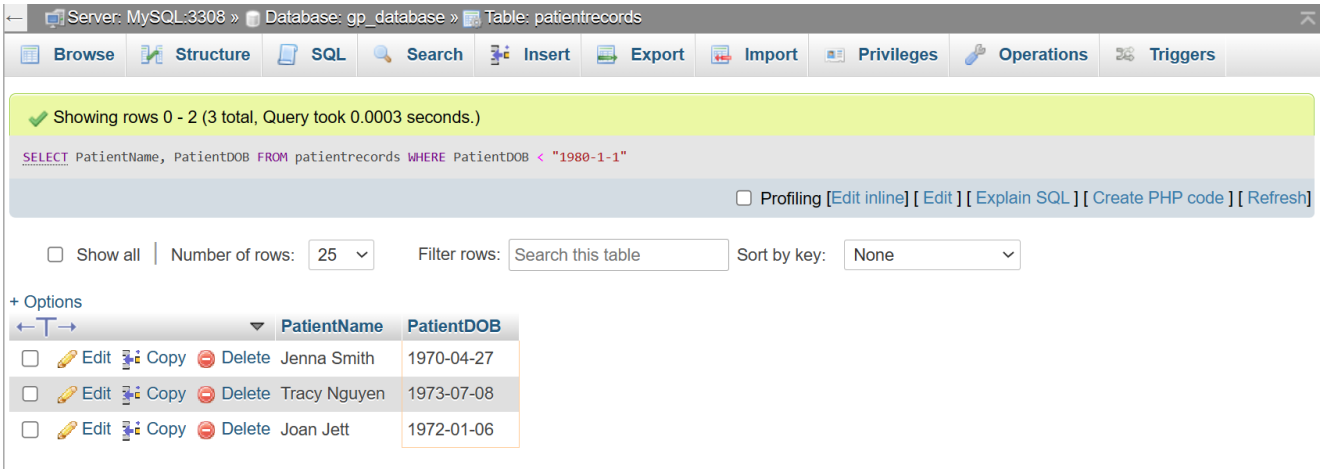
Return all the patients whose last name is ‘Jones’.

The wildcard character % is used to search and retrieve all the patient information from the patientrecords table for strings that end in "Jones".



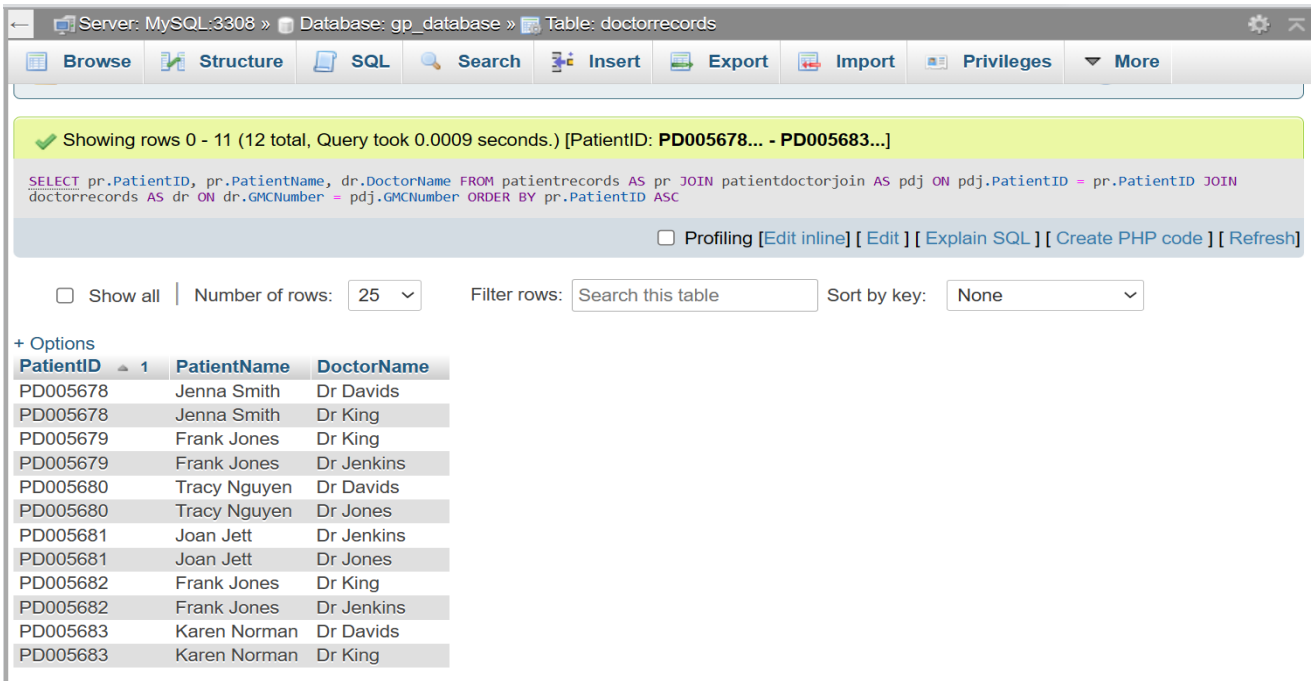
Find the names of the patients that were born before 1st January 1980

For date comparison, a simple "less than" operator is used, which checks if the patient's DateOfBirth is less than "1980-1-1."



List all the patients' names along with their registered doctors' names.

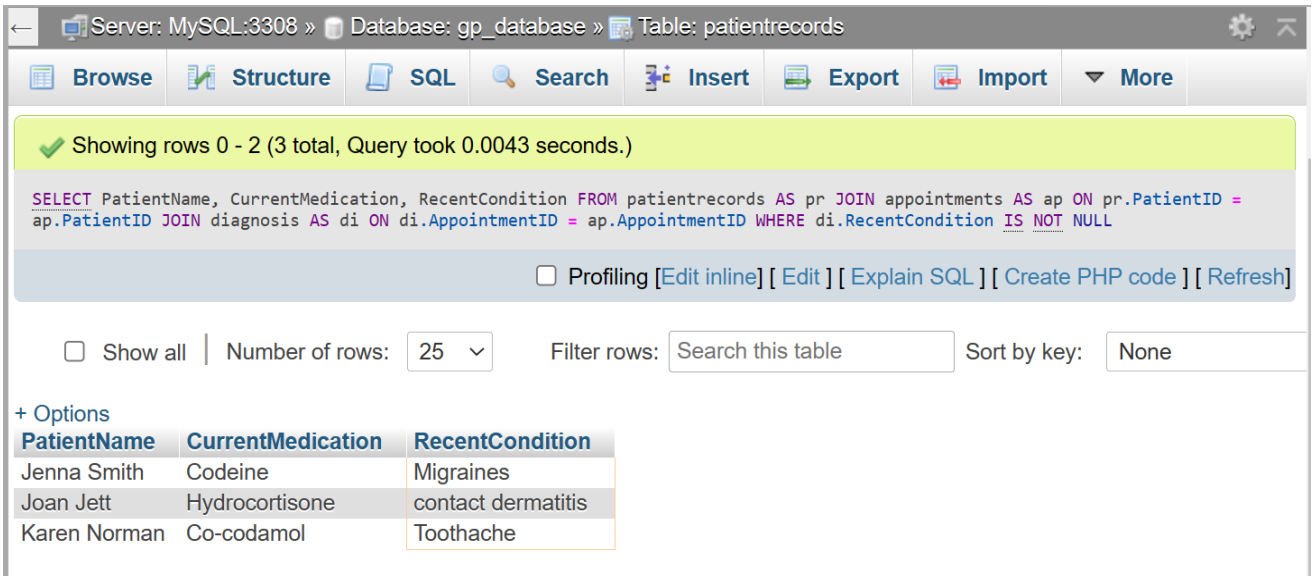
The JOIN statement will join the patient table with the patientdoctorjoin and doctorrecords tables and return the PatientID, PatientName, and DoctorName in ascending order, ordered by PatientID.



PatientID	PatientName	DoctorName
PD005678	Jenna Smith	Dr Davids
PD005678	Jenna Smith	Dr King
PD005679	Frank Jones	Dr King
PD005679	Frank Jones	Dr Jenkins
PD005680	Tracy Nguyen	Dr Davids
PD005680	Tracy Nguyen	Dr Jones
PD005681	Joan Jett	Dr Jenkins
PD005681	Joan Jett	Dr Jones
PD005682	Frank Jones	Dr King
PD005682	Frank Jones	Dr Jenkins
PD005683	Karen Norman	Dr Davids
PD005683	Karen Norman	Dr King

List all the patients who are currently taking medication. Give the name of the patient, their current medication, and the recent condition they are taking the medication for.

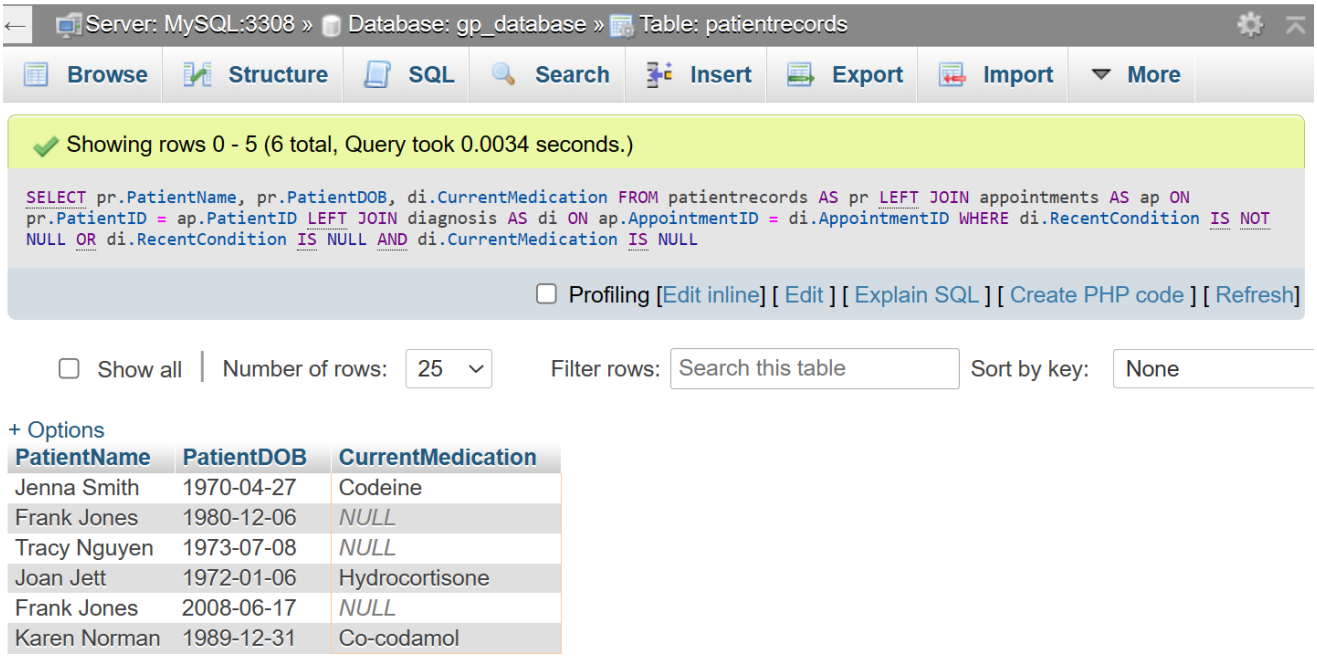
The JOIN statement connects the patient table, appointments, and diagnosis, and the WHERE statement specifies that the RecentCondition field should not be NULL, which means it will not retrieve records with NULL values in RecentCondition.



PatientName	CurrentMedication	RecentCondition
Jenna Smith	Codeine	Migraines
Joan Jett	Hydrocortisone	contact dermatitis
Karen Norman	Co-codamol	Toothache

List all patients, giving their name and date of birth, and, if the patient has had a recent condition, provide the medication they are taking. Otherwise, if the patient has had no recent condition, return null in the current medication field.

The LEFT JOIN was used to retrieve matching records from the patientrecords table even if they were not present in other tables, and the WHERE condition specifies the condition when the patient's name, DOB, and medication are required. The OR condition states that if the 'RecentCondition' is NULL, then the CurrentMedication should also be NULL. [4] has been referred to filter the NULL and IS NOT NULL values.



References

[1] https://www.scaler.com/topics/how-to-calculate-age-from-date-of-birth-in-sql/

[2] https://www.w3schools.com/sql/func_mysql_dayofweek.asp

[3] https://www.w3schools.com/mysql/mysql_case.asp

[4] https://www.w3schools.com/sql/sql_null_values.asp

[5] https://www.wellbeingsoftware.com/resources/what-is-emr/

[6] R. H. Mulholland *et al.*, "Cohort Profile: Early Pandemic Evaluation and Enhanced Surveillance of COVID-19 (EAVE II) Database," vol. 50, no. 4, pp. 1064–1074, 2021, doi: 10.1093/ije/dyab028. [Online]. Available: https://doi.org/10.1093/ije/dyab028

[7] https://www.w3schools.com/tags/att_input_type_checkbox.asp