PHARMACEUTICAL DATABASE MANAGEMENT SYSTEM

Surya Muthiah Pillai
Computer Science
SUNY Buffalo
New York
suryamut@buffalo.edu

Gowtham Rajasekaran
Computer Science
SUNY Buffalo
New York
grajasek@buffalo.edu

Abstract

Manual pharmacy management is a tedious task. The aspects to be managed in a pharmacy are manifold ranging from maintaining the medicine inventory to storing the purchase and prescription history of the patient. All these real-time and burdensome tasks cannot be managed efficiently by a pharmacy employee using an Excel sheet. A Pharmacy management system is a convenient and secure solution for managing the pharmacy efficiently by implementing a consolidated system that manages the medicine inventory, supplier, manufacturer, billing and prescription information. This system minimizes the employee labor and also decreases the wait time faced by the patient at the pharmacy.

1 Introduction

The scope of the project is to design and implement a system aimed towards the management of pharmaceutical stores. The system enhances the efficiency of management by providing an ease-of-use database system for managing pharmaceutical stores by maintaining details about pharmacies and their employees, customers and their billing details, medicines and their manufacturers and suppliers, prescriptions and also the details of doctors who prescribe the medicines along with their unique license numbers.

There are several hassles faced in the manual maintenance of a pharmaceutical store such as maintaining the medicine inventory details, customer bills, purchase history and prescription details, employee information, manufacturer and supplier details. Our system aims to handle these hassles, by providing a robust and efficient database system which can store all this information and also allow for easy retrieval, insertion, updation and maintaining data consistency at the same time. Since large volumes of pharmaceutical data needs to be maintained across a group of pharmacies, which is also subject to frequent updation and retrieval by the store employees, a database would be much more efficient to maintain the system rather than using an excel file. Using a database system preserves data integrity by enforcing integrity constraints and also eliminates data redundancy.

2 Target users

The Pharmaceutical management system is administered by a central Head/Admin who owns or manages a set of pharmaceutical stores. The Admin employs managers for individual stores who in turn will manage the employees of a store and the store itself. The employees along with the manager of a store will access the database to assist the customers and manage the medicines in the pharmacy. The roles will be explained in detail later with the help of a relational schema on how employees manage the medicines, bills, customers, prescriptions etc. and how the admin takes care of the suppliers and stock management.

3 Dataset

The dataset for this project is created using an online tool. Since, our system is designed and developed from scratch, the availability of requisite scripts to generate our data is bare minimal. So, we have used an online tool (www.mockaroo.com) with appropriate conditions set carefully to match all the constraints for each column in each table to generate our datasets. The datasets (including production dataset) have been provided in a separate folder in csv format for convenient use and access. The datasets are generated specifically to coordinate with corresponding primary keys and foreign keys to satisfy all the constraints. 'On Delete' conditions have been set in proper places to avoid erroneous queries in the future if data is tampered. The sql files to create the table and insert values is also provided along with the report in case of reproduction of the production dataset.

4 Scenario of our System (User/System)

The system is maintained by only The Admin, The Manager and the Employees.

Below is an example scenario of the interaction of various contributors in addition to those who maintain the system.

Admin (The Top Layer):

The admin manages the complete pharmaceutical database system of multiple stores he owns/manages. He is responsible for the system. So, the top layer of our system consists of manufacturers, suppliers and the pharmacies. The manufacturers provide the suppliers with supplies. This is a many-to-many relationship; therefore, the details are maintained separately on which manufacturers supply which suppliers (Manf_Supp_Contract). Suppliers on the other hand provide supplies for the pharmacy. A particular pharmacy can handle only one supplier but a supplier can be the source for multiple pharmacies. This is how the top level of the whole database system works.

Patients/Doctors (The Intermediate/Middle Layer):
Patients and Doctors contribute majorly to the intermediate layer of our system. A patient consults a doctor. Patient-Doctor is a many-to-many relationship as a patient can consult N number of doctors and vice versa. This is kept track using a different entity, prescriptions which is common to both doctors and patients. So, when a doctor prescribes medicines to a patient, a unique prescription is generated which includes the patient's unique SSN and the Doctor's License Number for identification. Then the patient hands over the prescription to an Employee of a pharmacy he visits.
Manager/Employee (Final Layer):
The final layer of the system relies upon the Employees of a pharmacy store. A pharmacy can have one manager and multiple employees where the manager also falls under the category of an employee. The employee now uses the Prescribed Medicines look up table to identify what medicines are allocated for the particular prescription. The employee checks if the medicine is available in the particular pharmacy he is working in, in the Medicine Section of our database system which is stored separately. If it's present, a bill is generated with a unique bill number for the patient. A bill constitutes the most important info of who the patient is and the pharmacy which generated the bill and against what prescription the bill was generated. So only one bill can be generated against a particular prescription. This information needs to be updated carefully which is the responsibility of the store manager Therefore, tampering with this data with a lot of references may lead to loss of information.

The next page explains the above scenario with the help of a relational schema which illustrates how our system works on the database level with the detailed explanation of individual relations and their relationship constraints.

5 Relation Schema (ER Diagram)

The relational schema for our system is visualized in detail in Figure 1.

5.1 Key attributes of Relations

The Primary keys and Foreign keys for the corresponding relations are given in Table 1.

Entity	**Primary Key**	**Foreign key**
Manufacturer	Manf_ID	
Supplier	Supplier_ID	
Manf_Supp_Contract		Manf_ID, Supplier_ID
Pharmacy	Pharmacy_ID	Supplier_ID
Employee	Emp_ID	Pharmacy_ID
Medicine	Medicine_ID	Manf_ID
pharmacy_medicines		Pharmacy_ID, Medicine_ID
Doctor	License_Number	
Patient	SSN	
Prescription	Presc_ID	SSN, License_Number
Bill	Bill_Number	Pharmacy_ID, Presc_ID
Prescribed_Medicines		Presc_ID, Medicine_ID

Table 1: Key Attributes of Relations

6 Detailed Description of Relations and their Functional Dependencies

6.1 Manufacturer (Manf_ID, Manf, Phone, Email)

Primary Key: Manf_ID
The Manufacturer Id is a unique identifier provided to each manufacturer for retrieving information about the medicine manufacturers. There can be multiple manufacturers with the same name, so we need to use a primary key like Manf_ID to uniquely identify a manufacturer record.

Attributes:

- Manf_ID: Unique Id for uniquely referencing the manufacturer information. (INTEGER)
- Manf: Name of the manufacturing company. (VARCHAR)
- Phone: Phone number of the manufacturer. (VARCHAR)
- Email: Email address of the manufacturer. (VARCHAR)

Functional dependencies:
In the Manufacturer relation, Manufacturer ID could uniquely determine the manufacturer's name, phone and email values.

- Manf_ID -> Manf, Phone, Email
- Manf_ID, Manf -> Phone, Email

This relation is already in BCNF (Boyce-Codd Normal Form) because,

- The FDs are non-trivial.
- Manf_ID, (Manf_ID, Manf) -> super keys in the relation.

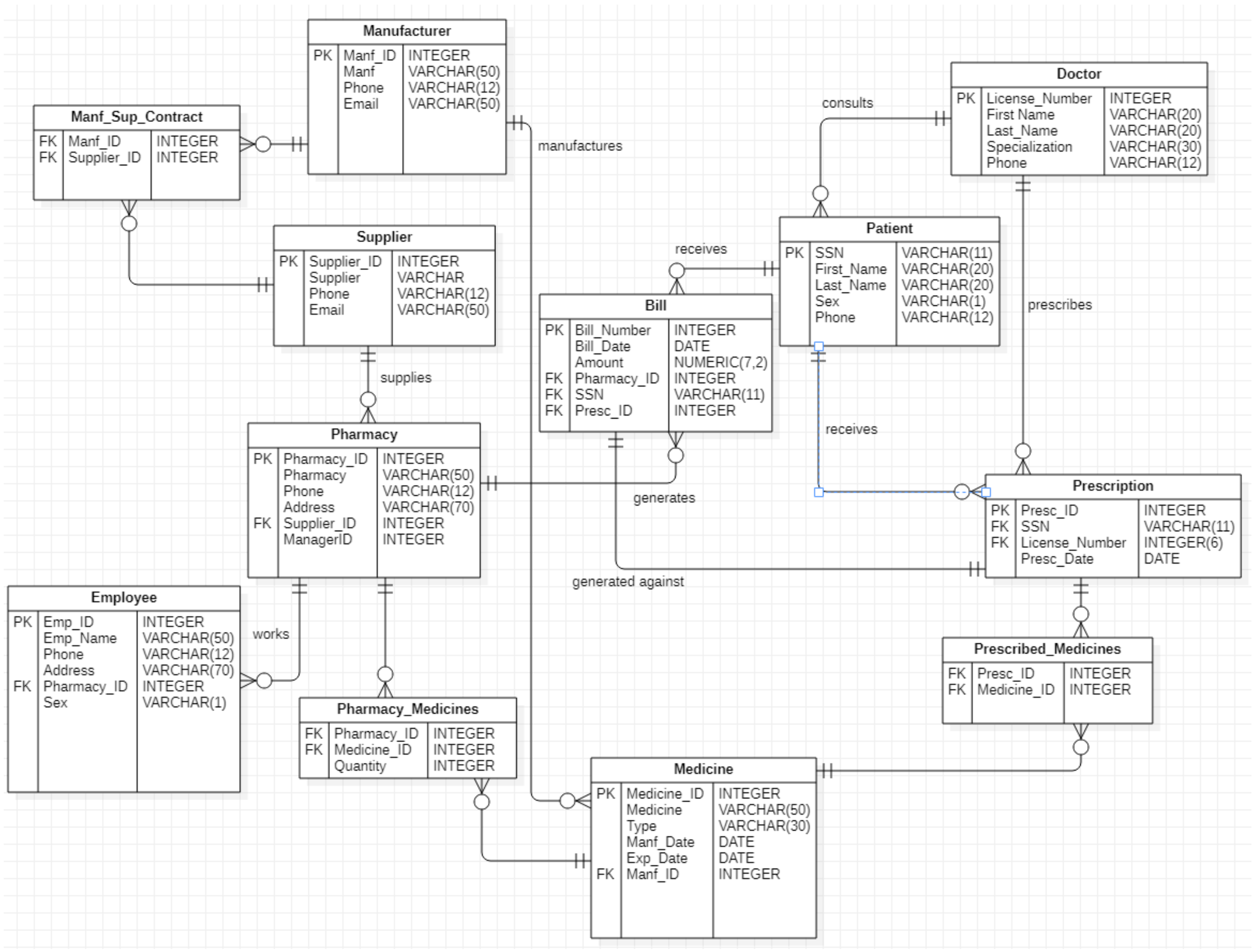


Figure 1: Relational Schema

6.2 Supplier(Supplier_ID, Supplier, Phone, Email)

Primary Key: Supplier_ID

Supplier_ID is a primary key because there may be many suppliers with the same name working with each pharmacy. We associate the supplier details with a unique Id to uniquely reference them. The supplier key is referenced in the pharmacy relation.

Attributes:

- ➢ Supplier_ID:Unique Id for uniquely referencing the supplier information. (INTEGER)
- ➢ Supplier: Name of the supplier. (VARCHAR)
- ➢ Phone: Phone number of the supplier. (VARCHAR)
- ➢ Email: Email address of the supplier. (VARCHAR)

Functional dependencies:

In the Supplier relation, Supplier ID could uniquely determine the supplier's name, phone and email values.

- Supplier_ID -> Supplier, Phone, Email
- Supplier_ID, Supplier -> Phone, Email

This relation is already in BCNF (Boyce-Codd Normal Form) because,

- The FDs are non-trivial
- Supplier_ID, (Supplier_ID, Supplier) -> super keys in the relation.

6.3 Manf_Supp_Contract(Manf_ID, Supplier_ID)

Foreign keys: Manf_ID, Supplier_ID

This relation is used for mapping the manufacturer ID to the supplier ID, the main purpose of the relation is to map multiple suppliers to multiple manufacturers. A supplier can be contracted to multiple medicine manufacturers to get medicines, also at the same time a manufacturer also can provide medicines to multiple suppliers. So, in order to solve the many to many relationship dependencies, we create this relation with two foreign keys mapped to each other.

Attributes:

- ➢ Manf_ID: Unique Id for uniquely referencing the manufacturer information. (INTEGER)
- ➢ Supplier_ID: Unique Id for uniquely referencing the supplier information. (INTEGER)

Foreign key deletion handling: ON DELETE CASCADE

Functional dependencies:

Both the attributes in the Manf_Supp_Contract relation are not functionally dependent on each other, this relation as the main purpose of the relation is to map multiple suppliers to multiple manufacturers.

The relation has no functional dependencies, so the only candidate key is (Manf_ID,Supplier_ID) this implies the relation is both in 3NF and BCNF.

6.4 Pharmacy (Pharmacy_ID, Pharmacy, Phone, Address, ManagerID, Supplier_ID)

Primary key: Pharmacy_ID

The pharmacy Id is an unique identifier provided to each pharmacy store for maintaining its information. The Pharmacy_ID will be referenced by the Bill and the Pharmacy_medicines relation for bill generation and medicine availability purposes.

Foreign Key: Supplier_ID

Each pharmacy has a dedicated supplier supplying medicines to them, we reference this key from the supplier table to identify the supplier information for that pharmacy.

The Pharmacy relation holds the information about each pharmacy store. There are multiple pharmacy stores under a single management. The name, phone number, address, supplier Id, pharmacy Id and the manager Id of each pharmaceutical store is stored in this relation. Each store has a manager and the manager's employee Id is stored in this table.

Attributes:

- ➢ Pharmacy_ID: Unique Id for uniquely referencing the pharmacy information. (INTEGER)
- ➢ Pharmacy: Name of the pharmacy. (VARCHAR)
- ➢ Phone: Phone number of the pharmacy. (VARCHAR)
- ➢ Address: Address of the pharmacy. (VARCHAR)
- ➢ ManagerID: Employee ID of the manager managing that particular pharmacy. (INTEGER)
- ➢ Supplier_ID: Unique Id for uniquely referencing the supplier information. (INTEGER)

Foreign key deletion handling: ON DELETE CASCADE

Functional dependencies:

- Pharmacy_ID -> Pharmacy, Phone, Address, ManagerID, Supplier_ID.
- Pharmacy_ID,Pharmacy -> Phone, Address, ManagerID, Supplier_ID.

This relation is already in BCNF (Boyce Codd Normal Form) because,

- The FDs are non-trivial
- Pharmacy_ID, (Pharmacy_ID,Pharmacy) -> super keys in the relation.

6.5 Employee(Emp_ID, Emp_Name, Phone, Address, Pharmacy_ID, Sex)

Primary key: Emp_ID

Each employee has a unique employee id 'Emp_ID' attributed to them for retrieving the information about the Employees as there can be multiple employees with the same name.

Foreign key: Pharmacy_ID

Each employee works for a particular pharmacy, we reference the Pharmacy_ID from the Pharmacy relation to map the employees to the Pharmacy_ID.

The employee relation will contain the information of both the employees and the pharmacy manager. The employees and the

manager will access the database to assist the customers and manage the medicines in the pharmacy.
Attributes:

- ➢ Emp_Id: Unique Identifier given to Each employee (INTEGER)
- ➢ Emp_Name: Name of the employee. (VARCHAR)
- ➢ Phone: phone number of the employee. (VARCHAR)
- ➢ Address: Address of the employee. (VARCHAR)
- ➢ Pharmacy_ID: Pharmacy Id of the store that the employee works for. (INTEGER)
- ➢ Sex: Gender of the employee. (VARCHAR)

Foreign key deletion handling: ON DELETE CASCADE

Functional dependencies:

- Emp_ID -> Emp_Name, Phone, Address, Pharmacy_ID, Sex
- Emp_ID, Emp_Name -> Phone, Address, Pharmacy_ID, Sex

This relation is already in BCNF (Boyce Codd Normal Form) because,

- The FDs are non-trivial
- Empr_ID, (Emp_ID, Emp_Name) -> super keys in the relation.

6.6 Medicine(Medicine_ID, Medicine, Manf_Date, Exp_Date, Manf_ID)

Primary Key:Medicine_ID
The Medicine Id is a Unique Id provided to a medicine for retrieving details of the particular medicine. We use this as a primary key because this ID can uniquely identify each medicine record.
Foreign Key: Manf_ID
We use the Manufacturer Id as the foreign key. Each medicine has a manufacturer and the details of the manufacturer are stored in a separate table, we match each Medicine_ID to the Manf_ID to retrieve information about the manufacturer of a medicine in case of restocking medicines.
The medicine relation stores the details of all the medicines such as medicine name, its manufacture and expiration date and the manufacturer Id
Attributes:

- ➢ Medicine_ID: A unique ID for retrieving medicine information. (INTEGER)
- ➢ Medicine: Name of the medicine. (VARCHAR)
- ➢ Manf_Date: Date when the medicine was manufactured. (DATE)
- ➢ Exp_Date: Date when the medicine will expire. (DATE)
- ➢ Manf_ID: Unique Id for uniquely referencing the manufacturer information. (INTEGER)

Foreign key deletion handling: ON DELETE CASCADE

Functional dependencies

- Medicine_ID -> Medicine, Manf_Date, Exp_Date, Manf_ID
- Medicine_ID, Medicine -> Manf_Date, Exp_Date, Manf_ID

This relation is already in BCNF (Boyce Codd Normal Form) because,

- The FDs are non-trivial
- Medicine_ID, (Medicine_ID, Medicine) -> super keys in the relation.

6.7 Pharmacy_medicines (Pharmacy_ID, Medicine_ID, Quantity)

Foreign Keys: Pharmacy_ID, Medicine_ID
The purpose of the relation is to handle the many to many relationship dependencies between the pharmacy and medicine relations. We map the foreign keys Pharmacy_ID to the Medicine_ID to retrieve information about the medicines available in that particular pharmacy associated with the Pharmacy_ID. We also store the information of the quantity of the medicine associated with the Medicine_ID.
Attributes:

- ➢ Pharmacy_ID: Unique Id for uniquely referencing the pharmacy information. (INTEGER)
- ➢ Medicine_ID: A unique ID for retrieving medicine information. (INTEGER)
- ➢ Quantity: Quantity of that particular medicine corresponding to the Pharmacy_ID (INTEGER)

(DEFAULT)=0
The default value of quantity is set to 0 if values are not inserted.
Foreign key deletion handling: ON DELETE CASCADE

Function dependencies:

- Pharmacy_ID, Medicine_ID -> Quantity

This relation is already in BCNF (Boyce Codd Normal Form) because,

- The FDs are non-trivial.
- (Pharmacy_ID, Medicine_ID) is the super key in the relation.

6.8 Doctor(License_Number, First_Name, Last_Name, Specialization, Phone)

Primary_Key: License_Number
License_Number is the Unique ID mapped to a Doctor for retrieving their information and for evaluating the legitimacy of the Doctor mentioned in the prescription provided by the patient. The License number is made the primary key as there can be multiple doctors with the same names and we also reference the License_number in the prescription relation.
The Doctor relation has a list of Doctors and their information such as doctor’s Name, Specialization and Phone number.
Attributes:

- License_Number: A unique number associated with a doctor for medical practices. (INTEGER)
- First_Name: First name of the doctor. (VARCHAR)
- Last_Name: Last name of the doctor. (VARCHAR)
- Specialization: Specialization field of the doctor (VARCHAR)
- Phone: Phone number of the doctor (VARCHAR)

Functional dependencies

- License_Number -> First_Name, Last_Name, Specialization, Phone

This relation is already in BCNF (Boyce Codd Normal Form) because,

- The FDs are non-trivial
- License_Number - super key in the relation.

6.9 Patient(SSN, First_Name, Last_Name, Sex, Phone)

Primary key: SSN

SSN is the primary key for the Patient relation, An SSN is a unique ID provided to each US citizen, which can be used to uniquely retrieve the patient records as there can be multiple patients with the same name.

Patient relation stores the Personal information of the patient such as Name, Sex and Phone.

Attributes:

- SSN: Social security number of the patient. (VARCHAR)
- First_Name: First name of the Patient. (VARCHAR)
- Last_Name: Last name of the Patient. (VARCHAR)
- Sex: Gender of the patient. (VARCHAR)
- Phone: Phone number of the patient. (VARCHAR)

Functional dependencies

- SSN -> First_Name, Last_Name, Sex, Phone
- SSN, First_Name, Last_Name -> Sex, Phone

This relation is already in BCNF (Boyce Codd Normal Form) because,

- The FDs are non-trivial
- SSN, (SSN, First_Name, Last_Name) -> super keys in the relation.

6.10 Prescription(Presc_ID,SSN, License_Number, Presc_Date)

Primary Key:Presc_ID

The prescription ID is the primary key of the prescription relation, it is used to retrieve the prescription details of a patient. The Presc_ID is the primary key because a patient may have multiple prescriptions with the same doctor on the same day, so we tag the fields with a Presc_ID to uniquely identify the record. The Presc_ID will be referenced in the Bill relation for billing purposes.

Foreign Keys: SSN, License_Number

We use the SSN and License_Number from the Patient and Doctor relation for storing the prescription details of the patient along with the doctor's license number

The Prescription relation stores the details of the prescription along with the patient's SSN and the Doctor's License number. Given a patient's SSN, we can retrieve all the prescriptions the patient received along with the details for the Doctor who prescribed it.

Attributes:

- Presc_ID: A unique ID for retrieving prescription information. (INTEGER)
- SSN: Social security number of the patient. (VARCHAR)
- License_Number: A unique number associated with a doctor for medical practices. (INTEGER)
- Presc_Date: Date of prescription. (DATE)

Foreign key deletion handling: ON DELETE CASCADE

Functional dependencies

- Presc_ID -> SSN, License_Number, Presc_Date

This relation is already in BCNF (Boyce Codd Normal Form) because,

- The FDs are non-trivial
- Presc_ID -> super key in the relation.

6.11 Bill (Bill_Number, Bill_Date, Amount, Pharmacy_ID, SSN, Presc_ID)

Primary key: Bill_Number

A unique Id is provided for each bill generated which helps in keeping track of the bill information which serves as a primary key. This bill uniquely identifies each record in the relation table.

Foreign_Key: Pharmacy_ID, SSN, Presc_ID

We reference the Pharmacy_ID, SSN, Presc_ID from the pharmacy, patient and prescription tables respectively. Using these references, we can find out the patient's information, previous purchase history of the patient, the prescription information associated with the bill and also from which pharmacy the medicines were purchased from.

Attributes:

- Bill_Number: The Number associated with each bill for uniquely identifying billing records (INTEGER)
- Bill_Date: The Date when the medicines were purchased. (DATE)
- Amount: Total Cost of the bill. (NUMERIC)
- Pharmacy_ID: Pharmacy Id of the store that the medicines were purchased from. (INTEGER)
- SSN: Social Security Number of the patient. (VARCHAR)
- Presc_ID: The prescription ID for matching the bill to the medicines. (INTEGER)

Foreign key deletion handling: ON DELETE CASCADE

In the Bill relation, a Bill number can be used to uniquely determine the values of the date of billing, total amount, Id of the pharmacy where the bill was generated, SSN of the patient and the prescription ID used for purchasing the medicines. And a prescription ID may also uniquely determine a patient's SSN.

Functional dependencies

- Bill_Number -> Bill_Date, Amount, Pharmacy_ID, SSN, Presc_ID
- Presc_ID -> SSN

For the dependency, Bill_Number -> Bill_Date, Amount, Pharmacy_ID, SSN, Presc_ID, the FD is not trivial and the attribute Bill_Number is a super key for the relation Bill, but for the non-trivial FD, Presc_ID -> SSN, Presc_ID is not a super key in the relation. This implies that the relation is not in BCNF. We will not be trying to solving this by decomposing the tables for now as it will be later discussed in detail in Section 6.3, Errors faced during Normalization.

6.12 Prescribed_Medicines (Presc_ID, Medicine_ID)

Foreign Keys: Presc_ID, Medicine_ID

The primary purpose of this relation is to handle the many to many relationship dependencies between the Prescription and Medicine relation. Here we map Each of the Medicines that the patient bought with the prescription ID, so we can retrieve all the details of the medicines purchased by a patient. A prescription can have multiple medicines, and a medicine can also be present in multiple prescriptions.

Attributes:

Presc_ID: A unique ID for retrieving prescription information. (INTEGER)

Medicine_ID: A unique ID for retrieving medicine information. (INTEGER)

Foreign key deletion handling: ON DELETE CASCADE

Functional dependencies:

Both the attributes in the Prescribed_Medicines relation are not functionally dependent on each other, this relation as the main purpose of the relation is to map multiple prescriptions to multiple medicines.

The relation has no functional dependencies, so the only candidate key is (Presc_ID, Medicine_ID) this implies the relation is both in 3NF and BCNF.

7 Normalization to BCNF

Upon looking up the FDs in relations from Section 6.1 till 6.12, it can be seen that all Functional Dependencies are in BCNF.

The aggregate simplified final FDs in BCNF are:

- **Manufacturer:** Manf_ID -> Manf, Phone, Email
- **Supplier:** Supplier_ID -> Supplier, Phone, Email
- **Manf_Supp_Contract:** NA
- **Pharmacy:** Pharmacy_ID -> Pharmacy, Phone, Address, ManagerID, Supplier_ID
- **Employee:** Emp_ID -> Emp_Name, Phone, Address, Pharmacy_ID, Sex
- **Medicine:** Medicine_ID -> Medicine, Manf_Date, Exp_Date, Manf_ID
- **Pharmacy_medicines:** Pharmacy_ID, Medicine_ID -> Quantity
- **Doctor:** License_Number -> First_Name, Last_Name, Specialization, Phone
- **Patient:** SSN -> First_Name, Last_Name, Sex, Phone
- **Prescription:** Presc_ID -> SSN, License_Number, Presc_Date
- **Bill:** Bill_Number -> Bill_Date, Amount, Pharmacy_ID, SSN, Presc_ID (There is one issue in this Functional Dependency which is addressed in the upcoming section)
- **Prescribed_Medicines:** NA

7.1 Errors faced during Normalization

Normalization was done in a very careful manner as the dependencies were ambiguous in certain relations. It turned into quite the problem that we had to drop one column from the relation Bill (this query is included at the end of the DELETE.sql file which needs to be run at last – discussed in detail in Section 8). According to the above normalized form, Bill_number can uniquely identify any given tuple. At the same time, Presc_ID in the relation can also uniquely identify the attribute SSN i.e., Presc_ID->SSN as there is only one prescription generated against a bill. So according to BCNF rules, the relation is supposed to be split into 2 in which the SSN is removed from the relation Bill and added to a newly created relation containing the attributes SSN and Presc_ID. But we already have a relation that does the job, i.e., the relation Prescription where the Functional Dependency is already satisfied. Therefore, the column SSN is alone dropped from the relation Bill satisfying our BCNF conditions finally. The FD for Bill alone now changes to the following:

- **Bill:** Bill_Number -> Bill_Date, Amount, Pharmacy_ID, Presc_ID

8 Implementation and Execution:

The SQL data to replicate or reproduce this dataset and our database system has been provided along with the report. The order of execution is explained in simple terms as below.

1. **Create a database named in your choice** *(Eg: Pharmacy)*
2. **Create Tables:**

CREATE.sql - the order is maintained so the file can be executed as it is.

3. **Sample Dataset:**

Load_sample.sql - Running this file will load a sample dataset from the production dataset into the system.

4. **Production Dataset:**

The below sql files are required to be run in the same order as below to reproduce our production dataset:

1) Manufacturer.sql
2) Supplier.sql
3) Manf_Supp_Contract.sql
4) Pharmacy.sql
5) Employee.sql
6) Medicine.sql
7) Pharmacy_Medicines.sql
8) Doctor.sql
9) Patient.sql
10) Prescription.sql
11) Bill.sql
12) Prescribed_Medicines.sql

(either you can run the above-mentioned files or just run the **LOAD.sql** only for one time)

5. **DELETE.sql:** Since our dataset is randomly generated, same names can be matched with different primary key ids. So, in order to keep the ID and name unique, we delete the duplicate names which correspond to different ids. This file also contains a drop column (SSN) for relation Bill, the reason for which was discussed in Section 7.1. This file needs to be run irrespective of the whether it is sample or production dataset after running the above files in order.

9 Problems faced while using the production dataset

- We faced the problem of similar names matched up against different unique ids in a table. For example: Since our data is randomly generated, the same company name got generated twice against two different primary key ids i.e., a single company has two identities which is wrong.
- So, in order to change the above dataset, we ran a query that removes duplicates. So, before the creation of whole data, we ran the delete query each time we created a table which resulted in ambiguity errors as we had generated sql queries to insert data in the upcoming tables with foreign keys references which had no primary keys.
- The problem was corrected by creating the whole dataset first and then running the delete query which removed the duplicates alone. And since we had 'on delete cascade' condition in place of every foreign key references, the delete query ran smoothly without any data pointing problems.
- Generating SSN was one of the difficult tasks when created with a random tool set. As matching it with the bills and prescription needed those exact SSNs of Patients in the patients table, recreating that data for prescriptions and Bills was quite challenging.
- One more thing that had to be done was altering Pharmacy table. There is an attribute named ManagerID which points to one of Emp_ID in Employee relation. The foreign key constraint cannot be instantiated as there is a foreign key reference for each employee's pharmacy, he/she is working, referencing to Pharmacy table. So, in order to tackle this issue, we created a trigger which checks on each update or insert happening on Pharmacy table that whether the new value contained in ManagerID matches with any of the entries in the Employee table (also discussed in Section 10.3).

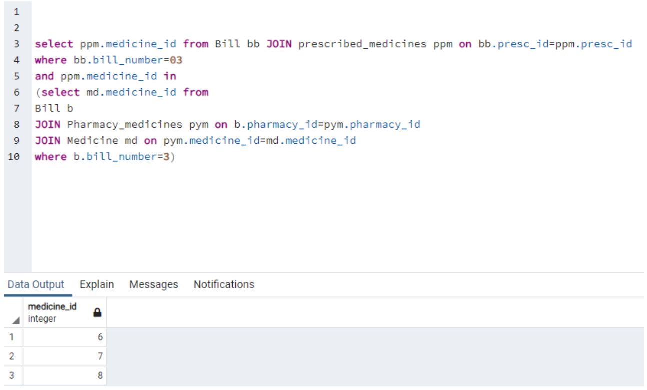
10 Examples of Advanced Queries for this System

10.1 Example 1(Query of Higher-Degree): The first example is used to help analyze how our system works and how the multi-user system needs to be handled in very careful manner in order to prevent any discrepancies. (This example was discussed in Milestone1)

One example of querying our dataset is described below to better understand how multi-connected and complex our database system is.

Q: Find the medicines bought against the bill_number 03:

This also explains a wrong entry of a pharmacist in the bill which needs to be examined and queried properly.



```
select ppm.medicine_id from Bill bb JOIN prescribed_medicines ppm on bb.presc_id=ppm.presc_id
where bb.bill_number=03
and ppm.medicine_id in
(select md.medicine_id from
Bill b
JOIN Pharmacy_medicines pym on b.pharmacy_id=pym.pharmacy_id
JOIN Medicine md on pym.medicine_id=md.medicine_id
where b.bill_number=3)
```

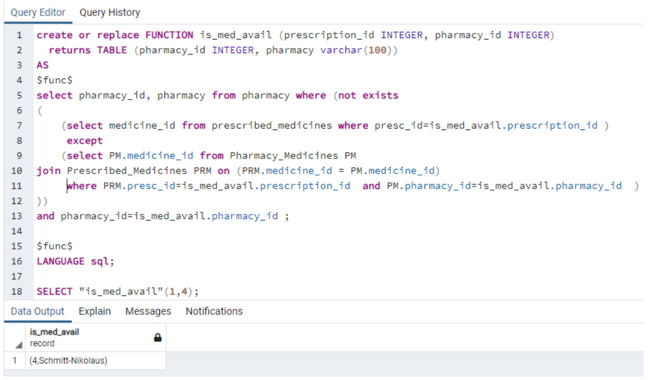
Data Output Explain Messages Notifications

	medicine_id integer
1	6
2	7
3	8

The bill03 has been generated against a prescription (Presc_ID=03) which has 4 medicines (Medicine_ID=1,6,7,8) to be prescribed which will be billed automatically but the pharmacy has only 3 medicines of the prescription (6,7,8) available if looked up in the pharmacy_medicines table. So, the query above is structured in such a way that it retrieves only the correct available medicines in the bill. So instead of billing for the available medicines, the entry in prescribed medicines has led to the billing of an unavailable medicine which needs to be checked before entering by the pharmacist in this case.

10.2 Example 2 (Functions): This example illustrates how handy a sql function in our system can help a user to find if the prescribed medicines are available in a particular pharmacy or not. This was created to address the above issue of billing unavailable medicines and convenience of the buyers to check if the medicines the patient is trying to buy is available in a given pharmacy.

Q: Given the prescription id (Presc_id) and pharmacy id (Pharmacy_ID), it outputs the pharmacy_id back along with the pharmacy name if the medicines are available.



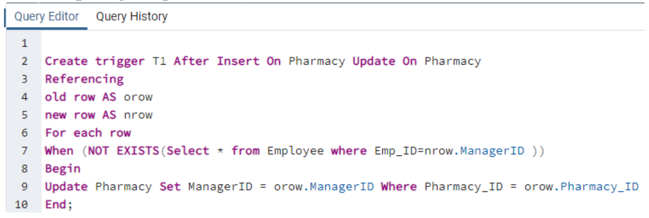
```
create or replace FUNCTION is_med_avail (prescription_id INTEGER, pharmacy_id INTEGER)
  returns TABLE (pharmacy_id INTEGER, pharmacy varchar(100))
AS
$func$
select pharmacy_id, pharmacy from pharmacy where (not exists
(
    (select medicine_id from prescribed_medicines where presc_id=is_med_avail.prescription_id )
     except
    (select PM.medicine_id from Pharmacy_Medicines PM
join Prescribed_Medicines PRM on (PRM.medicine_id = PM.medicine_id)
     where PRM.presc_id=is_med_avail.prescription_id  and PM.pharmacy_id=is_med_avail.pharmacy_id  )
))
and pharmacy_id=is_med_avail.pharmacy_id ;

$func$
LANGUAGE sql;

SELECT "is_med_avail"(1,4);
```

```
is_med_avail
record
1  (4,Schmitt-Nikolaus)
```

10.3 Example 3 (Triggers):

We create this trigger to prevent any discrepancy happening from an entry in Pharmacy table.

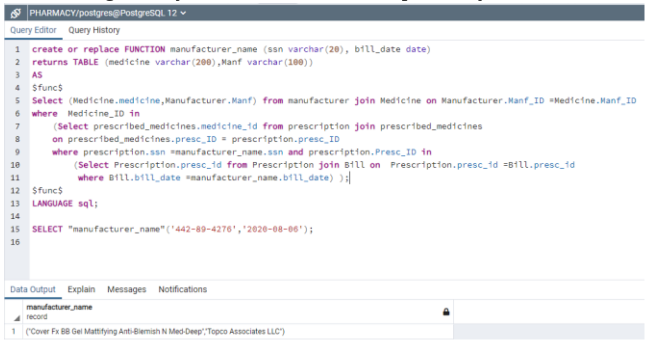
This problem was addressed in the last point (5th point) in Section 8 where problems faced during the production dataset were discussed. Since we have not enforced a foreign key reference in Pharmacy table against managerID which is supposed to point to Emp_ID in Employee table, we have solved this by creating a trigger which gets triggered on each insertion or updation in the Pharmacy Table and checks whether the ManagerID inputted is present in the Employee relation against any value of Emp_ID. (Note: This trigger needs to be implemented as a trigger function in case postgresql is used).



```

Create trigger T1 After Insert On Pharmacy Update On Pharmacy
Referencing
old row AS orow
new row AS nrow
For each row
When (NOT EXISTS(Select * from Employee where Emp_ID=nrow.ManagerID ))
Begin
Update Pharmacy Set ManagerID = orow.ManagerID Where Pharmacy_ID = orow.Pharmacy_ID
End;
```

10.4 Example 4 (Functions):

This example illustrates how the following function can help find the list of manufacturers of medicines prescribed for a particular patient on a certain date which are given by SSN and bill_date respectively.



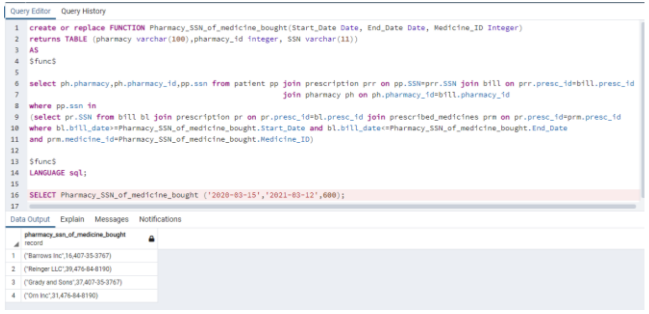
PHARMACY/postgres@PostgreSQL 12

```
create or replace FUNCTION manufacturer_name (ssn varchar(20), bill_date date)
returns TABLE (medicine varchar(200),Manf varchar(100))
AS
$func$
Select (Medicine.medicine,Manufacturer.Manf) from manufacturer join Medicine on Manufacturer.Manf_ID =Medicine.Manf_ID
where  Medicine_ID in
    (Select prescribed_medicines.medicine_id from prescription join prescribed_medicines
    on prescribed_medicines.presc_ID = prescription.presc_ID
    where prescription.ssn =manufacturer_name.ssn and prescription.Presc_ID in
        (Select Prescription.presc_id from Prescription join Bill on  Prescription.presc_id =Bill.presc_id
         where Bill.bill_date =manufacturer_name.bill_date) );
$func$
LANGUAGE sql;

SELECT "manufacturer_name"('442-89-4276','2020-08-06');

```

```
manufacturer_name
record
1  ("Cover Fx BB Gel Mattifying Anti-Blemish N Med-Deep","Topco Associates LLC")
```

10.5 Example 5:

Q: Find the patients, and the pharmacy they bought a particular medicine in a given date range.



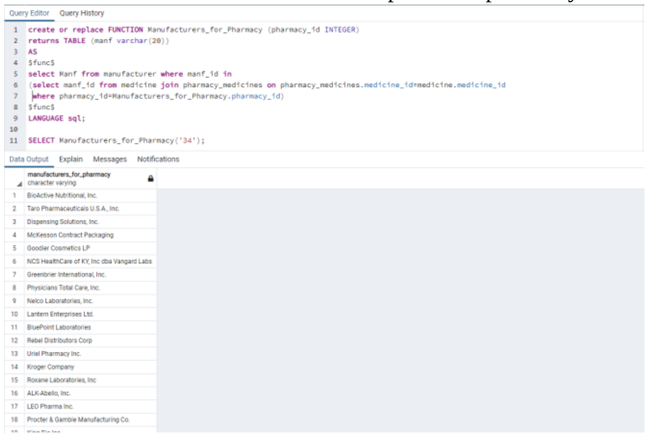
```
create or replace FUNCTION Pharmacy_SSN_of_medicine_bought(Start_Date Date, End_Date Date, Medicine_ID Integer)
returns TABLE (pharmacy varchar(100),pharmacy_id integer, SSN varchar(11))
AS
$func$

select ph.pharmacy,ph.pharmacy_id,pp.ssn from patient pp join prescription prr on pp.SSN=prr.SSN join bill on prr.presc_id=bill.presc_id
                                             join pharmacy ph on ph.pharmacy_id=bill.pharmacy_id
where pp.ssn in
(select pr.SSN from bill bl join prescription pr on pr.presc_id=bl.presc_id join prescribed_medicines prm on pr.presc_id=prm.presc_id
where bl.bill_date>=Pharmacy_SSN_of_medicine_bought.Start_Date and bl.bill_date<=Pharmacy_SSN_of_medicine_bought.End_Date
and prm.medicine_id=Pharmacy_SSN_of_medicine_bought.Medicine_ID)

$func$
LANGUAGE sql;

SELECT Pharmacy_SSN_of_medicine_bought ('2020-03-15','2021-03-12',600);

```

```
pharmacy_ssn_of_medicine_bought
record
1  ("Barrows Inc",16,407-35-3767)
2  ("Reinger LLC",39,476-84-8190)
3  ("Grady and Sons",37,407-35-3767)
4  ("Orn Inc",31,476-84-8190)
```

Below are a few other examples of useful and simple functions:

10.6 Example 6:

List of manufacturers of medicines in a particular pharmacy

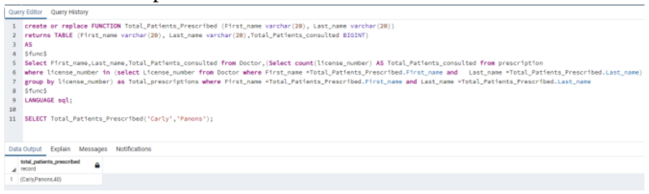


```
create or replace FUNCTION Manufacturers_for_Pharmacy (pharmacy_id INTEGER)
returns TABLE (manf varchar(20))
AS
$func$
select Manf from manufacturer where manf_id in
(select manf_id from medicine join pharmacy_medicines on pharmacy_medicines.medicine_id=medicine.medicine_id
 where pharmacy_id=Manufacturers_for_Pharmacy.pharmacy_id)
$func$
LANGUAGE sql;

SELECT Manufacturers_for_Pharmacy('34');
```

```
manufacturers_for_pharmacy
character varying
1   BioActive Nutritional, Inc.
2   Taro Pharmaceuticals U.S.A., Inc.
3   Dispensing Solutions, Inc.
4   McKesson Contract Packaging
5   Goodier Cosmetics LP
6   NCS HealthCare of KY, Inc dba Vangard Labs
7   Greenbrier International, Inc.
8   Physicians Total Care, Inc.
9   Nelco Laboratories, Inc.
10  Lantern Enterprises Ltd.
11  BluePoint Laboratories
12  Rebel Distributors Corp
13  Uriel Pharmacy Inc.
14  Kroger Company
15  Roxane Laboratories, Inc
16  ALK-Abello, Inc.
17  LEO Pharma Inc.
18  Procter & Gamble Manufacturing Co.
```

10.7 Example 7:

The number of patients a doctor has consulted

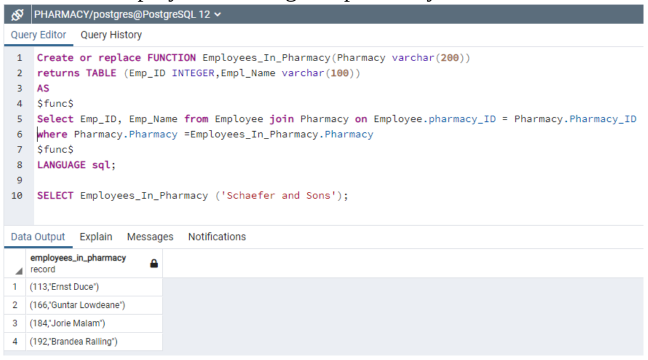


```
create or replace FUNCTION Total_Patients_Prescribed (First_name varchar(20), Last_name varchar(20))
returns TABLE (First_name varchar(20), Last_name varchar(20),Total_Patients_consulted BIGINT)
AS
$func$
Select First_name,Last_name,Total_Patients_consulted from Doctor,(Select count(license_number) AS Total_Patients_consulted from prescription
where license_number in (select License_number from Doctor where First_name =Total_Patients_Prescribed.First_name and   Last_name =Total_Patients_Prescribed.Last_name)
group by license_number) as Total_prescriptions where First_name =Total_Patients_Prescribed.First_name and Last_name =Total_Patients_Prescribed.Last_name
$func$
LANGUAGE sql;

SELECT Total_Patients_Prescribed('Carly','Panons');
```

```
total_patients_prescribed
record
1  (Carly,Panons,40)
```

10.8 Example 8:

The list of employees working in a pharmacy



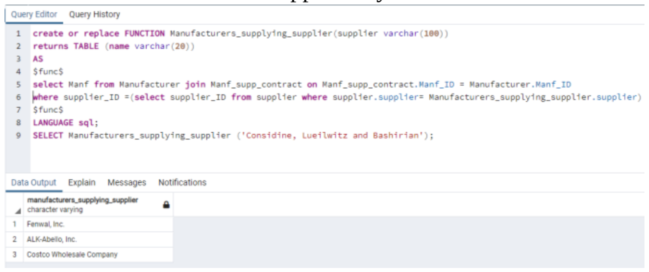
PHARMACY/postgres@PostgreSQL 12

```
Create or replace FUNCTION Employees_In_Pharmacy(Pharmacy varchar(200))
returns TABLE (Emp_ID INTEGER,Empl_Name varchar(100))
AS
$func$
Select Emp_ID, Emp_Name from Employee join Pharmacy on Employee.pharmacy_ID = Pharmacy.Pharmacy_ID
where Pharmacy.Pharmacy =Employees_In_Pharmacy.Pharmacy
$func$
LANGUAGE sql;

SELECT Employees_In_Pharmacy ('Schaefer and Sons');
```

```
employees_in_pharmacy
record
1  (113,"Ernst Duce")
2  (166,"Guntar Lowdeane")
3  (184,"Jorie Malam")
4  (192,"Brandea Ralling")
```

10.9 Example 9:

The list of manufacturers a supplier buys medicines from.



Query Editor | Query History

```sql
create or replace FUNCTION Manufacturers_supplying_supplier(supplier varchar(100))
returns TABLE (name varchar(20))
AS
$func$
select Manf from Manufacturer join Manf_supp_contract on Manf_supp_contract.Manf_ID = Manufacturer.Manf_ID
where supplier_ID =(select supplier_ID from supplier where supplier.supplier= Manufacturers_supplying_supplier.supplier)
$func$
LANGUAGE sql;
SELECT Manufacturers_supplying_supplier ('Considine, Lueilwitz and Bashirian');
```

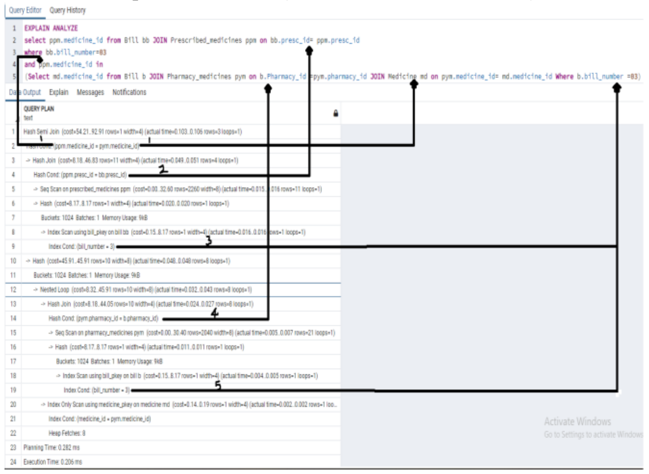
Data Output | Explain | Messages | Notifications

	manufacturers_supplying_supplier character varying
1	Fenwal, Inc.
2	ALK-Abello, Inc.
3	Costco Wholesale Company

11 QUERY OPTIMIZATION WITH "EXPLAIN"

In this Section we analyze a query with EXPLAIN (ANALYZE) to check if there is a way to optimize the query. Let's take the example 1 from Section 10.1.

First, we run the query along with EXPLAIN ANALYZE and check the output as follows: (Please zoom in if unclear)



Query Editor | Query History

```sql
EXPLAIN ANALYZE
select ppm.medicine_id from Bill bb JOIN Prescribed_medicines ppm on bb.presc_id= ppm.presc_id
where bb.bill_number=03
and ppm.medicine_id in
(Select md.medicine_id from Bill b JOIN Pharmacy_medicines pym on b.Pharmacy_id =pym.pharmacy_id JOIN Medicine md on pym.medicine_id= md.medicine_id Where b.bill_number =03)
```

Data Output | Explain | Messages | Notifications

```
QUERY PLAN
text
1   Hash Semi Join  (cost=54.21..92.91 rows=1 width=4) (actual time=0.103..0.106 rows=3 loops=1)
2     Hash Cond: (ppm.medicine_id = pym.medicine_id)
3     ->  Hash Join  (cost=8.18..46.83 rows=11 width=4) (actual time=0.049..0.051 rows=4 loops=1)
4           Hash Cond: (ppm.presc_id = bb.presc_id)
5           ->  Seq Scan on prescribed_medicines ppm  (cost=0.00..32.60 rows=2260 width=8) (actual time=0.015..0.016 rows=11 loops=1)
6           ->  Hash  (cost=8.17..8.17 rows=1 width=4) (actual time=0.020..0.020 rows=1 loops=1)
7                 Buckets: 1024  Batches: 1  Memory Usage: 9kB
8                 ->  Index Scan using bill_pkey on bill bb  (cost=0.15..8.17 rows=1 width=4) (actual time=0.016..0.016 rows=1 loops=1)
9                       Index Cond: (bill_number = 3)
10    ->  Hash  (cost=45.91..45.91 rows=10 width=8) (actual time=0.048..0.048 rows=8 loops=1)
11          Buckets: 1024  Batches: 1  Memory Usage: 9kB
12          ->  Nested Loop  (cost=8.32..45.91 rows=10 width=8) (actual time=0.032..0.043 rows=8 loops=1)
13                ->  Hash Join  (cost=8.18..44.05 rows=10 width=4) (actual time=0.024..0.027 rows=8 loops=1)
14                      Hash Cond: (pym.pharmacy_id = b.pharmacy_id)
15                      ->  Seq Scan on pharmacy_medicines pym  (cost=0.00..30.40 rows=2040 width=8) (actual time=0.005..0.007 rows=21 loops=1)
16                      ->  Hash  (cost=8.17..8.17 rows=1 width=4) (actual time=0.011..0.011 rows=1 loops=1)
17                            Buckets: 1024  Batches: 1  Memory Usage: 9kB
18                            ->  Index Scan using bill_pkey on bill b  (cost=0.15..8.17 rows=1 width=4) (actual time=0.004..0.005 rows=1 loops=1)
19                                  Index Cond: (bill_number = 3)
20                ->  Index Only Scan using medicine_pkey on medicine md  (cost=0.14..0.19 rows=1 width=4) (actual time=0.002..0.002 rows=1 loo...
21                      Index Cond: (medicine_id = pym.medicine_id)
22                      Heap Fetches: 8
23  Planning Time: 0.282 ms
24  Execution Time: 0.206 ms
```

From the output, it can be inferred that the cost for the query is the cumulative cost of each operation that is performed in the query as displayed. The Execution time is also displayed which can be taken into account for the query optimization step which needs to be adapted to the current query in order to reduce the cost and other resources used. As described in the screenshot, on the condition marked as 1, there isn't any explicit join specified in the query as of the output. But still this is considered as a join because of the query structure that intends to join the medicine ids from prescribed medicines outside the subquery and medicines from inside the subquery. Join conditions marked as 2 and 4 are present explicitly in the query which is of least significance. If we look at conditions 3 and 5, they are the same join condition which is ran 2 times in a single run of the complete query i.e., equating the bill number to a constant value. An optimization step in the query involves reducing the two conditions 3 and 5 into a single condition where the inner most branch of the tree structure output of the explain analyze query will have only one condition corresponding to condition 3(or)5. Similarly, a query is considered optimized when the tree structure of the explain analyze output does not grow outwards and only inwards. That is, after row 9 in the screenshot above, it can be seen that till the particular line the tree has been growing inwards while on the 10th row, the tree has again gone to the left that is it started to grow outwards. Optimizing the query in such a way that all the conditions above are satisfied can enable us to create an optimized query for the above-mentioned example.

11 Conclusion

In this paper, we proposed a system to tackle the problem of difficulties faced while maintaining a pharmaceutical database system manually by providing a legit DBMS system implemented using postgreSQL/MySQL to maintain the data revolving around a chain of pharmacies in a well-ordered manner that enhances the user-experience of DDL/DML/DCL in terms of SQL when using our Pharmaceutical Database Management System.

Our primary aim of this project is to test our system with vast variety of queries for robustness which has proved to be successful as we have given the above-mentioned (Section 10) few of the numerous queries we tried and to solve for any inconsistencies within our system to provide a perfect and solid pharmaceutical database system that enables complete reliability and ease-of-use environment for the end-users.

12 References

[1] H. Garcia-Molina, J. D. Ullman, J. Widom, Database Systems: The Complete Book, Second Edition, Prentice Hall, 2009.
[2] Avi Silberschatz, Henry F. Korth, S. Sudarshan, Database System Concepts, Seventh Edition, McGraw-Hill, 2019.
[3] https://www.mockaroo.com/
[4] https://www.postgresql.org/docs/9.1/sql-createtrigger.html
[5] https://www.postgresql.org/docs/9.1/sql-createfunction.html
[6] https://docs.staruml.io/working-with-additional-diagrams/entity-relationship-diagram
[7] https://www.postgresql.org/docs/9.1/sql-explain.html
[8] https://thoughtbot.com/blog/reading-an-explain-analyze-query-plan