Patient Prescription Database

Project 2 Report

CST 363: Intro to Database Systems
May 2022

Cyber Database
Janet Pham, Benjamin Textor

Introduction

The purpose of this database design is to deliver medical care providers easy access to the prescription information of patients. The database encapsulates all of the most vital figures regarding patient prescriptions from the doctor who ordered them, to the contracts between pharmaceutical companies and the vendor through which the patient had the prescription filled. The paramount focus of the design was to create logical and powerful tools to aggregate this data while remaining simple to use. To enable this design, precise specifications were followed in its creation:

1. Every patient added to the database is directly linked to several key points of information: the social security number of the patient, the patient's legal name, age and address, as well as the patient's primary physician. The database also supports any physician providing a prescription for any patient.

2. Each doctor's data table specifies their social security number, name, focus of expertise, and years practicing medical care.

3. Pharmaceutical companies are associated with their company name, as well as their telephone number.

4. Each drug is associated with a specific drug ID to differentiate identical prescriptions from different pharmaceutical companies. Each drug in the database also holds it's generic formula name, trade name, and pharmaceutical company that it comes from.

5. The directory of pharmacies include their names, addresses, and phone numbers. Each pharmacy also has a unique id number to accommodate multiple locations of the same franchise of pharmacy.

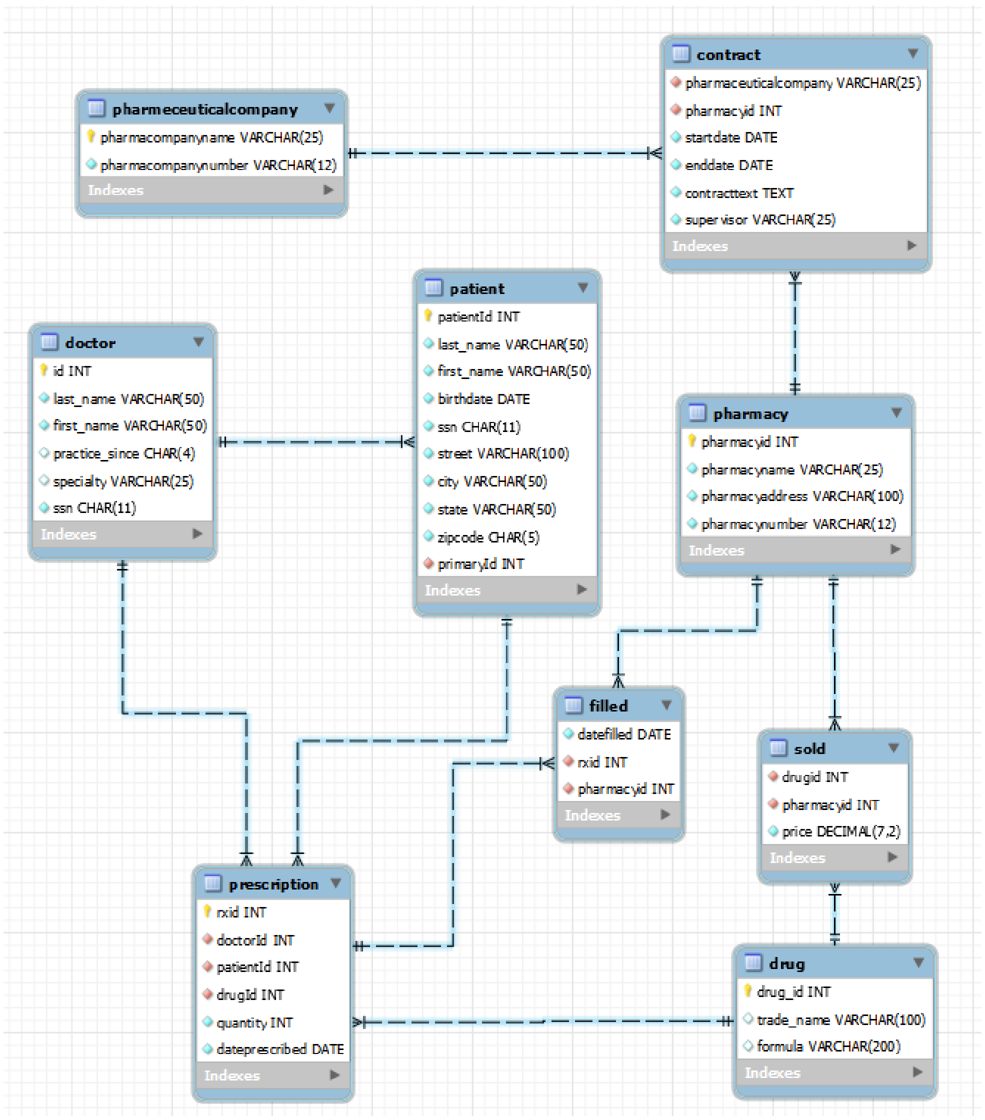
6. To accommodate unique pricing for particular trade name and generic drugs between different pharmacies, a "Sold" database has been created. This table provides the price and names of the prescriptions, along with the pharmacy associated with the product and price.

7. To further provide ease of access, a prescription table provides prescription numbers, the doctor who authored the prescription, the drug associated with the prescription, and the patient who was given the prescription. It also provides the date the prescription was written, as well as the quantity of units the prescription calls for. Prescriptions can be provided under a specific trade name, or as a generic formula. This system also supports patients having multiple prescriptions from any doctor.

8. To track when a patient successfully has their prescription filled at a pharmacy, a "Filled" table has been designed. This table tracks the date the prescription was filled, the prescription number, as well as the pharmacy ID of the specific branch the order was filled in.

9. The Contract table provides information concerning the contract between pharmaceutical companies and the pharmacies who sell their drugs. It lists the names of the pharmacies, the pharmaceutical companies they have contracts with, the start/end date of the contract, and the pharmacy supervisor who approved the contract. Supervisors have the ability to approve any number of contracts.

EER Model



Entities: patient, doctor, pharmaceuticalcompany, drug, pharmacy, sold, filled, contract
Association Entity: prescription
1: Many Relationship: doctor-patient, patient-prescription, doctor-prescription, drug-prescription, pharmaceuticalcompany-contract, drug-sold, pharmacy-sold, pharmacy-contract, pharmacy-filled, pharmaceuticalcompany-drug, prescription-filled

Under the **patient** entity, the attributes include the patient's social security number (*ssn*), name (*first_name, last_name*), address (*street*), bithdate(*birthdate*), and their primary physician (*primaryId*). The SSN is in char(11). The patient name has a varchar of 50 for both first and last name. The patient address data type is varchar(5) to take note of the patients 'zip code'. PrimaryId is an int and references the primary physician. The attributes in this entity are all NOT NULL. The patient's patientId is the primary key and the foreign key, *primaryId* refers to the ID of the doctor in the doctor entity table.

The **doctor** entity contains the doctor's *SSN* in varchar(11), the doctor's name(*first_name, last_name*) are varchar(50), specialty and years of experience(*practice_since*) in data type *char.* All of the attributes are NOT NULL. The primary key in this entity is the doctor's ID number.

Pharmaceutical company(**pharmaceuticalcompany**) is the next entity that contains the name and phone number of the company. The attribute *pharmacompanyname* covers the company's name and is varchar(25), while *pharmacompanynumber* represents the number assigned to that company with data type varchar(12). Both of these are NOT NULL. The primary key to this entity is *pharmacompanyname*.

Drug entity contains the drug identification(*drugid*) , trade name that the drug goes under(*tradename*), amd the generic formula name. *Drugid* is the primary key.

The **pharmacy** entity contains the pharmacy identification(*pharmacyid*) with char(3), the name of the pharmacy(*pharmacyname*) with varchar(25), the address(*pharmacyaddres*s) with varchar(100) and pharmacy phone number(*pharmacynumber*) with varchar(12). The primary key is the pharmacyid.

Entity **sold** represents the price (represented as numerid(7,2) of the drug. And to put a price to each name, the *drugid* and the *pharmacyid* is used to specify them. The *drugid* uses int(11) and is a foreign key to refer back to the drugid in the drug entity. The other foreign key is the *pharmacyid* which refers to the *pharmacyid* in the pharmacy entity. To keep the price positive, the *price* attribute is labeled as unsigned.

To provide easy access to the overall record, the association entity, **prescription** contains the *rxid* number, *patientId* number, *doctorId* number, the *drugId* number, the *quantity*, and the *datePrescribed*. The *rxid* number, *patientId*, *doctorId*, *drugId*, and *quantity* are all ints.The prescription date uses the data type date in the format of year-month-day. The primary key to this entity is the *rxid* number and the three foreign keys: the patiendId which refers to the ID in the patient entity, doctorId refers to the ID in the doctor entity and the drugid which references the drugid in the drug entity.

The **filled** entity tracks the date the prescription was filled (*datefilled*), the rx number (*rxid*), and the pharmacy the id was filled at (*pharmacyid*). Datefilled is using the data type date and the *rxid* number and *pharmacyId* are using int. The two foreign keys used in this table are the *rxid* which refers to the *rxid* in the prescription entity and the *pharmacyid* which refers to the *pharmacyid* in the pharmacy entity.

Contracts are usually made between the pharmaceutical company and the pharmacy to withhold both sides on the distribution of drugs. The **contract** entity holds the pharmaceutical company name, *pharmacyId*, *startdate* and *enddate* of the contract, the text within the contract, and the supervisor that manages the contract. Data types that were used in this table include int for *pharmacyId*, varchar for the *pharmacompanyname*, and the *supervisor*; and date in both the start and end dates. The data type text in the contract text. Foreign keys that are used in this are *pharmaceuticalcompany* which references *pharmacompanyname* in the pharmaceutical company entity, and *pharmacyid* which refers to the *pharmacyid* in the pharmacy entity table.

Relational Schema

```
-- create the database
DROP DATABASE IF EXISTS cst363;
CREATE DATABASE cst363;
-- select the database

USE cst363;

-- create tables and data
CREATE TABLE doctor (
  id int NOT NULL AUTO_INCREMENT,
  last_name varchar(50) NOT NULL,
  first_name varchar(50) NOT NULL,
  practice_since char(4) DEFAULT NULL,
  specialty varchar(25) DEFAULT NULL,
  ssn char(11) NOT NULL,
  PRIMARY KEY (id)
  );
```

```
CREATE TABLE patient
(
patientId int NOT NULL AUTO_INCREMENT,
last_name varchar(50) NOT NULL,
first_name varchar(50) NOT NULL,
birthdate DATE NOT NULL,
ssn char(11) NOT NULL,
street varchar(100) NOT NULL,
city varchar(50) NOT NULL,
state varchar(50) NOT NULL,
zipcode char(5) NOT NULL,
primaryId int NOT NULL,
PRIMARY KEY (patientId),
FOREIGN KEY (primaryId) REFERENCES doctor(id));

CREATE TABLE pharmeceuticalcompany
(pharmacompanyname VARCHAR(25) NOT NULL,
pharmacompanynumber VARCHAR(12) NOT NULL,
PRIMARY KEY (pharmacompanyname));

CREATE TABLE drug (
  drug_id int(11) NOT NULL,
  trade_name varchar(100) DEFAULT NULL,
  formula varchar(200) DEFAULT NULL,
  PRIMARY KEY (`drug_id`)
);

CREATE TABLE pharmacy
(
pharmacyid int NOT NULL AUTO_INCREMENT,
pharmacyname VARCHAR(25) NOT NULL,
pharmacyaddress VARCHAR (100) NOT NULL,
pharmacynumber VARCHAR(12) NOT NULL,
PRIMARY KEY (pharmacyid));

CREATE TABLE sold
(drugid int(11) NOT NULL,
pharmacyid int NOT NULL,
price NUMERIC(7, 2) UNSIGNED NOT NULL,
FOREIGN KEY (drugid) REFERENCES drug(drug_id),
FOREIGN KEY (pharmacyid) REFERENCES pharmacy(pharmacyid));

CREATE TABLE prescription
(rxid int NOT NULL AUTO_INCREMENT,
doctorId int NOT NULL,
patientId int NOT NULL,
drugId int(11) NOT NULL,
quantity int NOT NULL,
dateprescribed date NOT NULL,
filled char(3) default 'no',
PRIMARY KEY (rxid),
FOREIGN KEY (doctorId) REFERENCES doctor(id),
FOREIGN KEY (patientId) REFERENCES patient(patientId),
FOREIGN KEY (drugId) REFERENCES drug(drug_id));

CREATE TABLE filled
(datefilled DATE NOT NULL,
rxid INT NOT NULL,
pharmacyid INT NOT NULL,
FOREIGN KEY (rxid) REFERENCES Prescription(rxid),
FOREIGN KEY (pharmacyid) REFERENCES Pharmacy(pharmacyid));

CREATE TABLE contract
(pharmaceuticalcompany VARCHAR(25) NOT NULL,
pharmacyid int NOT NULL,
startdate DATE NOT NULL,
enddate DATE NOT NULL,
contracttext TEXT NOT NULL,
supervisor VARCHAR(25) NOT NULL,
FOREIGN KEY (pharmaceuticalcompany) REFERENCES
PharmeceuticalCompany(pharmacompanyname),
FOREIGN KEY (pharmacyid) REFERENCES pharmacy(pharmacyid));
commit;
```
(In the interest of brevity, insert statements for example drugs and pharmacies is not included, but can be found in the schema file)

Normalization

To normalize the data of all tables, there was extensive use of unique, automatically-generated ID numbers, as well as foreign keys. The decision to follow this design philosophy allows incredibly thorough data while avoiding repeated information between tables. Over the course of the design the pharmacy, patient, and patient tables went through several revisions to normalize the data as much as possible.

Example Queries

1. Fetching a list of patients who filled their prescriptions through a specific pharmacy is easily achievable with this database:

```
select patient.last_name, pharmacy.pharmacyname
from patient
join prescription on prescription.patientId = patient.patientId
join filled on prescription.rxid = filled.rxid
join pharmacy on filled.pharmacyid = pharmacy.pharmacyid
where filled.pharmacyid = 1;
```

2. The database also allows one to easily fetch a list of prescriptions that have been filled, and called for a specific number of units:

```
select patient.last_name, prescription.quantity
from patient
join prescription on prescription.patientId = patient.patientId
join filled on prescription.rxid = filled.rxid
join pharmacy on filled.pharmacyid = pharmacy.pharmacyid
having prescription.quantity > 20;
```

3. Another example shows the power of the database by enabling users to search for very specific datasets. Such as specific patient information, primary physician name, and the tradename of the drug used to fill a prescription:

```
select distinct patient.last_name, patient.ssn, patient.street, doctor.last_name, drug.trade_name
from patient
join prescription on patient.patientId = prescription.patientId
join filled on prescription.rxid = filled.rxid
join pharmacy on filled.pharmacyid = pharmacy.pharmacyid
join doctor on patient.primaryId = doctor.id join drug on prescription.drugid = drug.drug_id
group by patient.ssn
order by patient.last_name;
```

4. The database also allows users to retrieve pertinent patient, physician, and pharmaceutical information for prescriptions filled after a particular date:

```
select patient.last_name, patient.birthdate, prescription.rxid, doctor.last_name, drug.trade_name,
pharmacy.pharmacyname, filled.datefilled
from patient
join prescription on prescription.patientId = patient.patientId
join filled on prescription.rxid = filled.rxid
join pharmacy on filled.pharmacyid = pharmacy.pharmacyid
join doctor on patient.primaryId = doctor.id join drug on prescription.drugid = drug.drug_id
where filled.datefilled > '2022-05-05' group by patient.ssn
order by filled.datefilled;
```

5. To assist patients to find their prescription at the price that is right for them, the database also allows users to fetch the average price of drugs at all pharmacies:

```
 select pharmacy.pharmacyname, avg(sold.price)
 from pharmacy
 join sold on pharmacy.pharmacyid = sold.pharmacyid
 group by pharmacy.pharmacyid
 order by avg(sold.price);
```

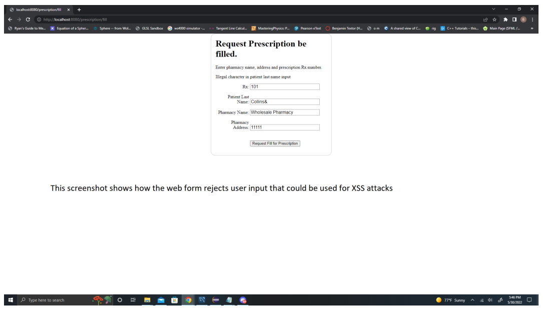
6. Building upon the last example, the database also makes it easy to find pharmaceutical companies which offer drugs at higher than the average price:

```
 select drug.trade_name, drug.formula, pharmacy.pharmacyname, sold.price
 from drug
 join sold on drug.drug_id = sold.drugid
 join pharmacy on sold.pharmacyId = pharmacy.pharmacyId
 where sold.price > (select avg(sold.price) from sold) and drug.formula = 'alendronate'
 order by drug.drug_id;
```

7. Finally, the layout of the database makes it simple to created combined datasets that shows a list of prescriptions that meet one or more criteria, such as a list of prescriptions that call for a quantity of greater than 45 units, and were prescribed after a specific date:

```
 select * from prescription where quantity > 45
 union
 select * from prescription where dateprescribed > '2022-05-05'
 group by rxid order by rxid;
```

Screenshots



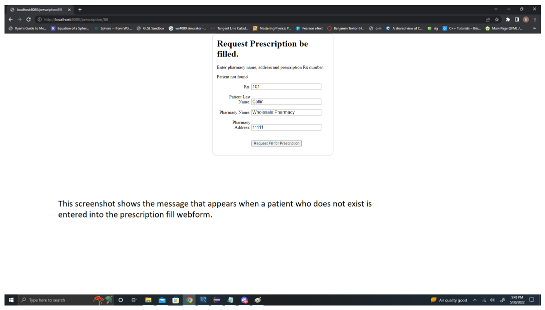
Request Prescription be filled.

Enter pharmacy name, address and prescription Rx number.

Illegal character in patient last name input

Rx:	101
Patient Last Name:	Collins&
Pharmacy Name:	Wholesale Pharmacy
Pharmacy Address:	11111

This screenshot shows how the web form rejects user input that could be used for XSS attacks



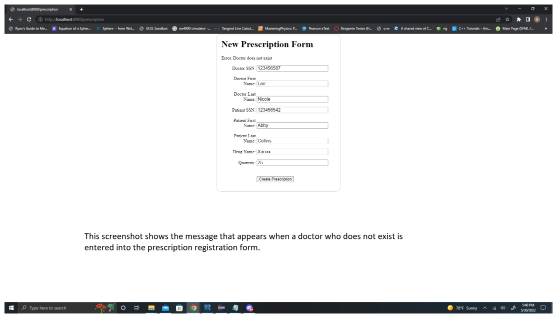
Request Prescription be filled.

Enter pharmacy name, address and prescription Rx number.

Patient not found

Rx:	101
Patient Last Name:	Collin
Pharmacy Name:	Wholesale Pharmacy
Pharmacy Address:	11111

This screenshot shows the message that appears when a patient who does not exist is entered into the prescription fill webform.

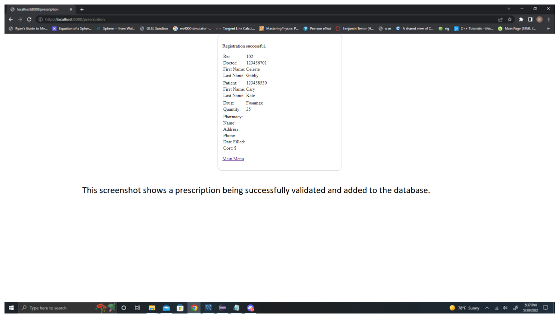


New Prescription Form

Error. Doctor does not exist

Doctor SSN:	123456587
Doctor First Name:	Larr
Doctor Last Name:	Nicole
Patient SSN:	123456542
Patient First Name:	Abby
Patient Last Name:	Collins
Drug Name:	Xanax
Quantity:	25

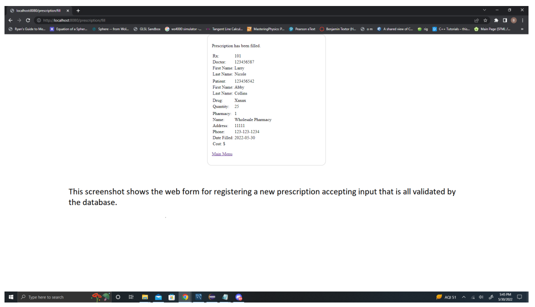
This screenshot shows the message that appears when a doctor who does not exist is entered into the prescription registration form.



Registration successful.

Rx:	102
Doctor:	123456701
First Name:	Celeste
Last Name:	Gabby
Patient:	123458530
First Name:	Cary
Last Name:	Kate
Drug:	Fosamax
Quantity:	25
Pharmacy:	
Name:	
Address:	
Phone:	
Date Filled:	
Cost: $	

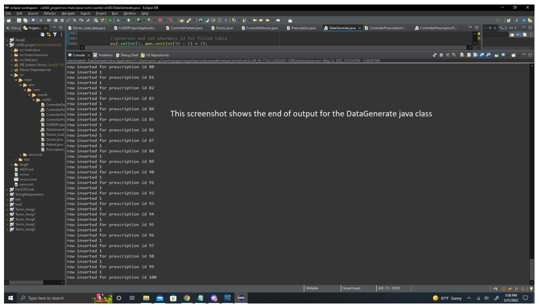
This screenshot shows a prescription being successfully validated and added to the database.



Prescription has been filled.

Rx:	101
Doctor:	123456587
First Name:	Larry
Last Name:	Nicole
Patient:	123456542
First Name:	Abby
Last Name:	Collins
Drug:	Xanax
Quantity:	25
Pharmacy:	1
Name:	Wholesale Pharmacy
Address:	11111
Phone:	123-123-1234
Date Filled:	2022-05-30
Cost: $	

This screenshot shows the web form for registering a new prescription accepting input that is all validated by the database.



```
            //generate and set pharmacy id for filled table
            ps2.setInt(3, gen.nextInt(10 - 1) + 1);
```

```
row inserted for prescription id 80
row inserted 1
row inserted for prescription id 81
row inserted 1
row inserted for prescription id 82
row inserted 1
row inserted for prescription id 83
row inserted 1
row inserted for prescription id 84
row inserted 1
row inserted for prescription id 85
row inserted 1
row inserted for prescription id 86
row inserted 1
row inserted for prescription id 87
row inserted 1
row inserted for prescription id 88
row inserted 1
row inserted for prescription id 89
row inserted 1
row inserted for prescription id 90
row inserted 1
row inserted for prescription id 91
row inserted 1
row inserted for prescription id 92
row inserted 1
row inserted for prescription id 93
row inserted 1
row inserted for prescription id 94
row inserted 1
row inserted for prescription id 95
row inserted 1
row inserted for prescription id 96
row inserted 1
row inserted for prescription id 97
row inserted 1
row inserted for prescription id 98
row inserted 1
row inserted for prescription id 99
row inserted 1
row inserted for prescription id 100
```

This screenshot shows the end of output for the DataGenerate java class

Conclusion

The database described above consists of a powerful, but simple to use, set of tables to hold all manner of pharmaceutical data. It strictly follows the provided requirements and efficiently allows for users to write robust queries using easy to follow logic. This project has been challenging in many ways – as creating a complex database should be – and allowed us to exercise our understanding of SQL. Through revisions we were able to successfully normalize our data to remove repetitive data points, and create a tool that works quite nicely. For the second phase of the project, the generation of example patients, doctors, prescriptions, and prices has been achieved, as well as functional webforms that interface with the database for prescription creation, and prescription filling. It was fun to figure out how to add some additional functionality to get the queries from the Project 1 submission to work, such as creating a generator for prices in the Sold table, adding to the "filled" table, setting the "filled" variable in the prescription table whenever a prescription is filled in the data generator and the prescription filling application, and associating pharmacies to filled prescriptions.