

Comprehensive Pharmaceutical Database

Project Report

CST 363 - Intro to Database Systems
February 2022

MEDYSYNC Consulting

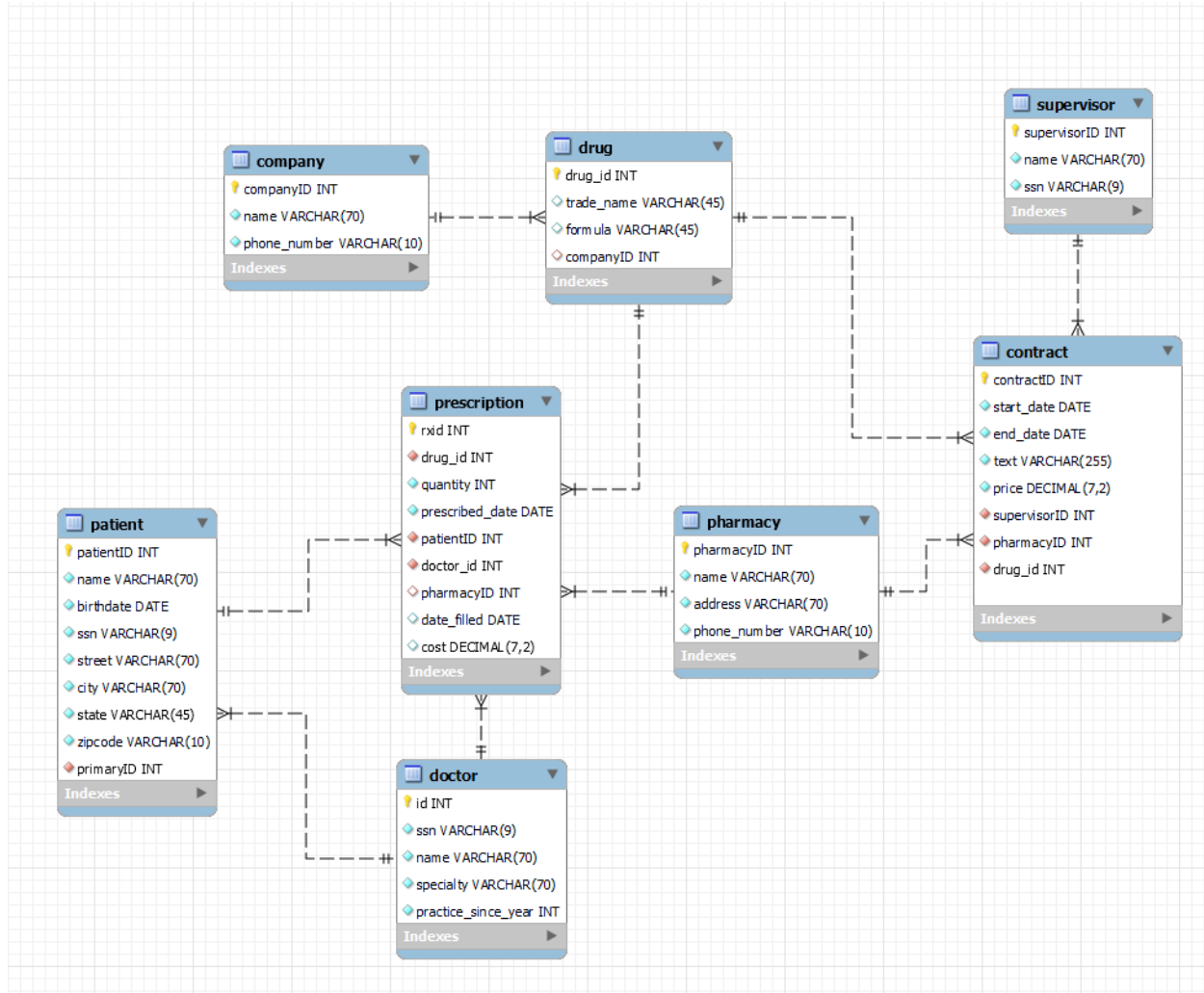
Warren Ngoun, Luis Jimenez-Barrios

Project Overview

Medical patients have many records to keep track of, doctors prescribe many different prescriptions each with numerous different fields like quantity and the brand name of the medication, pharmacies and companies have their own agreements on pricing and availability of their product and much more.

We here at MedySync wrote this database with the intent of helping with the organization, storage, and retrieval of data that surrounds all of these important contact points within the pharmaceutical and medical industry. The database records sensitive information regarding patients like their primary care doctors, their related prescriptions, and the link between those medications and what companies manufacture them, and where those medications can be found in pharmacies. The database also utilizes a web server to run a portal where patients can view their information and their doctors information as well. New patients and doctors can add themselves to the database and manage their info through it. Another use case, this database can assist companies with their drug management by allowing them to check their presence in pharmacies through their existing contracts, and by allowing them to see patients are currently taking the medications. See below and on the next page for further detail about the database and its connections.

DATABASE TABLE	Description
Company	Contains the name and phone number of the company
Contract	Serves as a link between Company and Pharmacy which details the terms, start and end of terms, and price of drug
Doctors	Contains the full name, specialty and years of experience
Drug	Contains drug name and generic formula
Patient	Contains full name, age, and address
Pharmacy	Contains name, address, and phone number
Prescription	Serves as a link between the patient and the Drug which contains the RX number, quantity, prescription and fill dates
Supervisor	Contains full name and social security number



Above is our ER diagram, depicting the linkage between the aforementioned tables. Primary keys are shown with the yellow key icon, blue diamonds show non-null fields, red diamonds show foreign keys, and any inner clear diamonds show potentially null fields.

Per the requirements, each patient has a unique identifying ID, a social security number (SSN), full name, birthdate, full address represented in smaller fields like street address and state etc. and their single primary care physician (stored as their doctor's ID).

Their patientID is the primary key and is an autoincrementing INT, their SSN is stored as VARCHAR(9) to record the numbers only (ignoring the dashes you commonly see), VARCHAR(70) for full name to accommodate long names, and street, city, state, zipcode as VARCHARs so we can do much more detailed searches like by patients in a certain state or city. We represented their age as a birthdate so that we could dynamically calculate their age depending on the current date.

The patient's doctor's ID is a foreign key that relates to the doctors table for their personal doctor.

Each doctor has a unique ID, a SSN, full name, medical specialty, and their start date for their practice.

For the same reasons as before, ID is the primary key, SSN is represented as VARCHAR(9) and full name with VARCHAR(70). Their years of experience is represented as a start year in INT form, so that you can dynamically calculate their years of experience like birthdate previously. Lastly, their medical specialty is stored as VARCHAR(70) to cover any long fields like 'Physical medicine and rehabilitation.'

A doctor can be multiple patient's personal doctors, so there's no patient foreign key in this table, that link is already covered.

A prescription contains an identifying RXNumber, a quantity of the drug to take, the prescribed date by the doctor, the filled date when the patient picks up their drug, the unique ID of the pharmacy that the patient picked up their medicine from, the cost of the drug, the ID of the drug, and lastly links to the doctor (via ID) and the patient prescribed for (also ID).

The RXNumber is a primary key and is stored as an auto incrementing INT, quantity is also stored as a whole INT. PrescriptionDate and FilledDate are stored as a YYYY-MM-DD date, but the FilledDate can be null if the customer hasn't picked up their medication yet. The FilledPharmacyID stores the unique ID of the pharmacy that they picked up their prescription from and can be null if the patient hasn't done so. Cost is a DECIMAL(7,2) and can be null until picked up. The drug_id is stored as a INT again. Lastly, the patient's/doctor's IDs are stored as an INT.

pharmacyID is a linkage to the Pharmacy table, but should only be used to check for where the patients have fulfilled their orders, not as a general link. Both the patients/doctors IDs and drug_id link to their respective tables.

A drug has a unique ID, a market trade name, a generic formula name, and the ID of the company that manufactures it.

It's drug_id is it's primary key, the trade name, generic name, and company name are all stored as VARCHAR(70) to allow for long names, and all three can be null.

The foreign key is the company ID and links to the manufacturer through the company table as for each drug there is always only a single manufacturing company.

One constraint we noticed was that a drug's price is only noted in a contract (see further down) and one column we could add is a MSRP or market price for a drug.

A company contains a unique ID, their full company name and phone number. ID is an AI INT, name is stored as a VARCHAR(70), & phone number is stored as VARCHAR(10) to account for the numbers only (sans the parentheses/dashes you sometimes see). The linkage is already handled in the drug table by CompanyName.

A pharmacy has a unique ID, a name for the store, an address, and a phone number.

The ID is the primary key and is an auto incrementing unique INT, the pharmacy's name stored as VARCHAR(70), its address stored as a VARCHAR(255), and phone number as a VARCHAR(10).

There are no foreign keys and the link is handled in the contract table.

A supervisor is in charge of a contract and there can be multiple contracts under their jurisdiction. The table contains their unique ID, SSN, and their full name.

Their ID is a primary key and is an AI INT, their SSN is stored as VARCHAR(9) and full name is stored as VARCHAR(70).

There are no foreign keys and the linkages are handled in the contract table.

A contract contains a unique ID, a start/end date for the contract, a text description, an agreed on price, the drug_id, the contracted pharmacy's ID, and that pharmacy's contract supervisor (stored as their ID).

The ContractID is an auto unique incrementing INT and is the primary key, both the start/end date as a YYYY-MM-DD date, the contract's description can be long so it is stored as VARCHAR(255), the price of the drug in the contract is stored as a decimal and can range from as small as 0.01 to 9999.99.

There are many foreign keys and linkages in this table and their data types have been previously explained. The supervisor's ID is a foreign key and links to the supervisor table, the drug_id is the foreign key that links to the drug table, and the pharmacyID links back to the pharmacy table.

THE RELATIONAL SCHEMA

```
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

-- Schema bigpharma
--
CREATE SCHEMA IF NOT EXISTS `bigpharma` DEFAULT CHARACTER SET utf8 ;
```

```

USE `bigpharma` ;

-----
-- Table `bigpharma`.`doctor`
-----

CREATE TABLE IF NOT EXISTS `bigpharma`.`doctor` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `ssn` VARCHAR(9) NOT NULL,
  `name` VARCHAR(70) NOT NULL,
  `specialty` VARCHAR(70) NOT NULL,
  `practice_since_year` INT NOT NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB;

-----
-- Table `bigpharma`.`patient`
-----

CREATE TABLE IF NOT EXISTS `bigpharma`.`patient` (
  `patientID` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(70) NOT NULL,
  `birthdate` DATE NOT NULL,
  `ssn` VARCHAR(9) NOT NULL,
  `street` VARCHAR(70) NOT NULL,
  `city` VARCHAR(70) NOT NULL,
  `state` VARCHAR(45) NOT NULL,
  `zipcode` VARCHAR(10) NOT NULL,
  `primaryID` INT NOT NULL,
  PRIMARY KEY (`patientID`),
  INDEX `fk_patient_doctor_idx` (`primaryID` ASC) VISIBLE,
  CONSTRAINT `fk_patient_doctor`
    FOREIGN KEY (`primaryID`)
      REFERENCES `bigpharma`.`doctor` (`id`)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION)
ENGINE = InnoDB;

-----
-- Table `bigpharma`.`company`
-----

CREATE TABLE IF NOT EXISTS `bigpharma`.`company` (
  `companyID` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(70) NOT NULL,

```

```

    `phone_number` VARCHAR(10) NOT NULL,
    PRIMARY KEY (`companyID`))
ENGINE = InnoDB;

-----
-- Table `bigpharma`.`drug`
-----

CREATE TABLE IF NOT EXISTS `bigpharma`.`drug` (
  `drug_id` INT(11) NOT NULL,
  `trade_name` VARCHAR(45) NULL,
  `formula` VARCHAR(45) NULL,
  `companyID` INT NULL,
  PRIMARY KEY (`drug_id`),
  INDEX `fk_drug_company1_idx` (`companyID` ASC) VISIBLE,
  CONSTRAINT `fk_drug_company1`
    FOREIGN KEY (`companyID`)
      REFERENCES `bigpharma`.`company` (`companyID`)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION)
ENGINE = InnoDB;

-----
-- Table `bigpharma`.`pharmacy`
-----

CREATE TABLE IF NOT EXISTS `bigpharma`.`pharmacy` (
  `pharmacyID` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(70) NOT NULL,
  `address` VARCHAR(70) NOT NULL,
  `phone_number` VARCHAR(10) NOT NULL,
  PRIMARY KEY (`pharmacyID`))
ENGINE = InnoDB;

-----
-- Table `bigpharma`.`prescription`
-----

CREATE TABLE IF NOT EXISTS `bigpharma`.`prescription` (
  `rxid` INT NOT NULL AUTO_INCREMENT,
  `drug_id` INT(11) NOT NULL,
  `quantity` INT NOT NULL,
  `prescribed_date` DATE NOT NULL,
  `patientID` INT NOT NULL,
  `doctor_id` INT NOT NULL,

```

```

`pharmacyID` INT NULL,
`date_filled` DATE NULL,
`cost` DECIMAL(7,2) NULL,
PRIMARY KEY (`rxid`),
INDEX `fk_prescription_drug1_idx` (`drug_id` ASC) VISIBLE,
INDEX `fk_prescription_patient1_idx` (`patientID` ASC) VISIBLE,
INDEX `fk_prescription_doctor1_idx` (`doctor_id` ASC) VISIBLE,
INDEX `fk_prescription_pharmacy1_idx` (`pharmacyID` ASC) VISIBLE,
CONSTRAINT `fk_prescription_drug1`
    FOREIGN KEY (`drug_id`)
    REFERENCES `bigpharma`.`drug` (`drug_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
CONSTRAINT `fk_prescription_patient1`
    FOREIGN KEY (`patientID`)
    REFERENCES `bigpharma`.`patient` (`patientID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
CONSTRAINT `fk_prescription_doctor1`
    FOREIGN KEY (`doctor_id`)
    REFERENCES `bigpharma`.`doctor` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
CONSTRAINT `fk_prescription_pharmacy1`
    FOREIGN KEY (`pharmacyID`)
    REFERENCES `bigpharma`.`pharmacy` (`pharmacyID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

-----
-- Table `bigpharma`.`supervisor`
-----

CREATE TABLE IF NOT EXISTS `bigpharma`.`supervisor` (
  `supervisorID` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(70) NOT NULL,
  `ssn` VARCHAR(9) NOT NULL,
  PRIMARY KEY (`supervisorID`))
ENGINE = InnoDB;

-----
-- Table `bigpharma`.`contract`
-----

```



```

-----
CREATE TABLE IF NOT EXISTS `bigpharma`.`contract` (
  `contractID` INT NOT NULL AUTO_INCREMENT,
  `start_date` DATE NOT NULL,
  `end_date` DATE NOT NULL,
  `text` VARCHAR(255) NOT NULL,
  `price` DECIMAL(7,2) NOT NULL,
  `supervisorID` INT NOT NULL,
  `pharmacyID` INT NOT NULL,
  `drug_id` INT(11) NOT NULL,
  PRIMARY KEY (`contractID`),
  INDEX `fk_contract_supervisor1_idx` (`supervisorID` ASC) VISIBLE,
  INDEX `fk_contract_pharmacy1_idx` (`pharmacyID` ASC) VISIBLE,
  INDEX `fk_contract_drug1_idx` (`drug_id` ASC) VISIBLE,
  CONSTRAINT `fk_contract_supervisor1`
    FOREIGN KEY (`supervisorID`)
      REFERENCES `bigpharma`.`supervisor` (`supervisorID`)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION,
  CONSTRAINT `fk_contract_pharmacy1`
    FOREIGN KEY (`pharmacyID`)
      REFERENCES `bigpharma`.`pharmacy` (`pharmacyID`)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION,
  CONSTRAINT `fk_contract_drug1`
    FOREIGN KEY (`drug_id`)
      REFERENCES `bigpharma`.`drug` (`drug_id`)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION)
ENGINE = InnoDB;

SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;

-- -----
-- There are missing inserts here, but those would take up too much space in
-- the report and are just in the included SQL schema file at the bottom instead.
-- -----

```

NORMALIZATION

During the process of creating the database we had to consider how data should be managed and stored and how tables should be split up. Looking at the tables, we implemented normalization by separating company and drug. This would be a case of 2NF (Second Normal Form) because the company is linked to the drug that it manufactures while allowing there to be different drugs from many companies. A second example of 2NF implementation would be the relation between the patient and doctor tables because the database allows patients to have one doctor, but doctors can have multiple patients with possibly the same name. In these cases normalization allows for there to be multiple entries of either patients or drugs underneath one name by using a key. After developing the database further along with a java application we decided to normalize the address of a user by making separate columns for the street, city, state, and zip. This was done because it allows for easier validation of the users address and would allow us to do more powerful queries if need be.

QUERIES

“Locate pharmacies that have prescriptions filled.”

This query is important because it allows a patient to find a pharmacy where they can pick up their medications.

```
SELECT date_filled, drug.drug_id, name FROM pharmacy JOIN  
Prescription ON prescription.pharmacyID = pharmacy.pharmacyID  
JOIN drug ON prescription.drug_id = drug.drug_id ORDER BY  
drug.drug_id;
```

“Display the total potential revenue to be gained from the medication when the drugs are sold at the highest potential amount in a pharmacy”

This query is important because it can show a company their potential revenue if all contracts had the maximum potential value such that they can pursue those contracts and negotiate higher prices.

```
SELECT d.drug_id, SUM(cost * quantity)
FROM prescription outerc
JOIN drug d ON outerc.drug_id = d.drug_id
WHERE date_filled IS NULL
GROUP BY drug_id;
```

“Find all the pharmacies that don't carry your specific drug and provide a phone number so you can contact them and make a contract with them.”

This query lets companies find missing contracts with pharmacies and provides a phone number for the company so that they can contact the pharmacy if they want to create a contract with them. Overall, boosting revenue by finding missing points of income.

```
SELECT p.name AS PharmacyName, p.Phone_Number, d.drug_id AS
MissingDrug

FROM pharmacy p NATURAL JOIN drug d NATURAL LEFT JOIN prescription
WHERE date_filled IS NULL ORDER BY MissingDrug;
```

“Find the patients who are taking similar medications and group them by medication name.”

This query allows doctors and pharmacists to find which patients require what medications. Pharmacists can view how much medication they should have on hand.

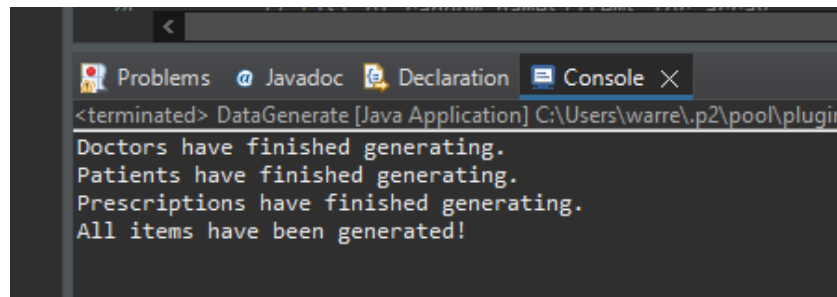
```
SELECT name, drug_id
FROM patient JOIN prescription ON patient.patientID = prescription.patientID
JOIN drug ON prescription.drug_id = drug.drug_id
WHERE drug_id NOT IN (SELECT drug_id
FROM prescription HAVING COUNT(drug_id) > 1)
GROUP BY drug_id
ORDER BY drug_id;
```

Display a count of how many patients that have been prescribed a drug have not picked up their orders yet grouped by drug id.

This query is important because it shows companys the orders that have been unfulfilled which are confirmed sources of revenue that have been unclaimed at the current query time.

```
SELECT drug.drug_id, COUNT(patientID) AS unfilledPickups
FROM prescription
JOIN drug ON drug.drug_id = prescription.drug_id
WHERE date_filled IS NULL
GROUP BY patientID;
```

JDBC FILES



```
<terminated> DataGenerate [Java Application] C:\Users\warre\.p2\pool\plugin
Doctors have finished generating.
Patients have finished generating.
Prescriptions have finished generating.
All items have been generated!
```

Our data generate performs 1000 random patients, 10 random doctors, and 5000 random prescriptions where some are unfulfilled still. Above is our confirmation that the application has run successfully. Below is an example of what the database looks like after prescriptions have generated.

	rxid	drug_id	quantity	prescribed_date	patientID	doctor_id	pharmacyID	date_filled	cost
▶	1	3	10	2003-03-08	2608	40	6	2022-11-18	96.09
	2	54	9	2000-11-09	2163	39	9	2022-08-01	26.02
	3	31	10	2013-03-17	2200	36	5	2021-05-19	38.46
	4	78	10	2008-11-21	2440	35	4	2021-05-08	4.29
	5	55	1	2002-05-16	2051	40	1	2020-01-03	59.40
	6	68	28	2010-08-14	2817	37	7	2020-01-07	3.25
	7	40	27	2003-01-12	2202	36	NULL	NULL	NULL
	8	31	17	2003-08-23	2379	35	8	2021-07-16	45.43
	9	99	7	2019-08-07	2676	38	6	2021-06-15	85.56
	10	86	18	2007-06-18	2478	40	4	2022-05-10	92.11
	11	82	18	2005-06-25	2457	36	2	2022-05-04	76.66
	12	52	20	2012-04-25	2464	34	8	2022-08-13	55.27
	13	25	7	2016-11-07	2695	34	1	2020-12-26	11.59
	14	82	21	2008-12-12	2314	35	NULL	NULL	NULL
	15	31	26	2003-10-07	2063	36	4	2022-09-27	28.42
	16	61	13	2009-03-07	2679	39	2	2021-09-27	14.81
	17	31	3	2007-06-17	2807	35	4	2022-12-23	57.03
	18	26	4	2015-01-25	2417	33	8	2020-01-26	76.05
	19	84	19	2004-03-16	2835	39	9	2020-06-25	80.20

Next we chose the ManagerReport file. For our input, we separated it to three lines to make it easier for validating. Below is a screenshot of the ID validation working. Further below that is an example of the output for a successful query from the database.

```
Please enter the pharmacyID number:
-1
Invalid ID. Try Again.
1.0
Invalid ID: java.lang.NumberFormatException: For input string: "1.0". Try Again.
1.123
Invalid ID: java.lang.NumberFormatException: For input string: "1.123". Try Again.
-12.31
Invalid ID: java.lang.NumberFormatException: For input string: "-12.31". Try Again.
Apples
Invalid ID: java.lang.NumberFormatException: For input string: "Apples". Try Again.
4
Now the Start Date to look for:
```

```
Please enter the pharmacyID number:
4
Now the start date to look for in YYYY-MM-DD format:
2020-01-01
and end date to search between in YYYY-MM-DD format:
2022-12-31
|--- Accuneb 44
--- Actos 44
--- Adderall XR 82
--- Advair 58
--- Aldactone 56
--- Allegra 115
--- Amaryl 77
--- Ambien 163
--- Amoxil 111
--- Aristocort 58
--- Ativan 79
--- Augmentin 69
--- Bactrim DS 74
--- Calan SR 30
```

For our web applications we chose the patient lineup (newPatient, getPatient, and updatePatient), so all of ControllerPatient. Through the web portal, we can register a new user. Below is a screenshot of that in action and then our resulting patient_show. Lastly is a screenshot of the database with our new values for our patient.

Patient ID:	4012
Name:	Ryan Jauzemus
Birthdate:	1999-10-10
Street:	41 Met Street
City:	Capitola
State:	California
Zipcode:	95060
Primary Physician:	Roma Carr

[Edit](#) | [Main Menu](#)

[illegible]

Below are screenshots of us attempting to enter invalid characters or even empty values while updating our patient, we have similar checks for all fields in new patient as well.

Update Patient Profile

Error: Invalid Street

ID:

Name:

BirthDate:

Street:

City:

State:

Zipcode:

Primary Physician
Name:

Update Patient Profile

Error: Blank Street

ID:

Name:

BirthDate:

Street:

CONCLUSIONS

In summary, we have both learned a great deal from this project. We both learned about how useful ER diagramming can be as well as how to set up tables with them, we learned about the different linkages between tables while using an ER tool and how this process also allows us to visualize if we need to normalize any tables or not, we learned about how Java can interact with a database and on top of that how you can implement a web server on top and create a functioning website that can handle persistent user data. The diagramming required us to take into account which table should handle what data, which further helped us understand the importance of the tables and their placement. We learned how to normalize data and how to separate fields into other tables to optimize our database. We also reinforced all of our previous knowledge of SQL and its queries by having to come up with real world examples that customers might be interested in. Apart from having to come up with examples, we also had to implement the queries to validate that they worked. Overall, we believe it was a very in depth learning opportunity with lots of hands on experience and really helped solidify the essentials.