Project 1 DB design

Entity-Relationship Model

1/20/2022

Consulting Team: Otters Online 1
Authors: Ian Lowe and Sabrina Ferras

Introduction

Purpose

Design a relational database for a pharmacy chain that will keep track of patients and their prescriptions, the doctors that prescribe them, the pharmaceutical companies the chain works with and the agreements they have together, as well as drug prices across pharmacy locations.

Requirements

(1)Each patient has an identifying SSN, plus a name, age, and address. & (6)Every patient has a primary physician.

A 'patient' table with an automatically generated patient ID as a primary key. Information needed in the 'patient' table are Social Seciurity Number (ssn), name, age, address, and primary physician.

The patient's address will be referenced through an 'address' table which will also have an auto generated id, as well as additional attributes street, city, state and zip. We assume that any given patient will just have one address on file, but that the same address can belong to several different patients.

The patient's primary physician will be referenced through the 'doctor' table.

(2)Physicians also have an identifying SSN. Additionally, each doctor has a name, a specialty, and years of experience.

A 'doctor' table will hold an identifying doctor id as a primary key, along with their Social Security Number, name, specialty, and years of experience.

(3)Each pharmaceutical company is identified by name and has a phone number.

A table for pharmaceutical companies which will include an ID as a primary key along with name and phone number attributes. We assume that each pharmaceutical company has one primary phone number, which will be stored here.

(4)Each drug has a trade name and a generic formula. Each drug is sold by a given pharmaceutical company, and the trade name identifies a drug uniquely.

A 'drug' table with the drug identified by a drug Id, with trade name and generic formula attributes. A pharmaceutical company can have many different drugs, but any given drug can only belong to one company.

(5)Each pharmacy has a name, address, and phone number.

A 'pharmacy' table with name, address, and phone number attributes. We will also have a pharmacyid attribute to act as the primary key, which will be auto-incremented.

(8)Each pharmacy sells several drugs and has a price for each. A drug could be sold at several pharmacies and the price could vary from one pharmacy to another.

Each pharmacy sells one or many drugs and each drug can be sold at zero or many pharmacies with varying prices. This indicates a many to many relationship which will require a table 'sells' with a price attribute.

(9)Doctors prescribe drugs for patients. A prescription has a unique RX number and is for one drug, one patient and is written by one doctor. A patient can have multiple prescriptions from multiple doctors. & (7)Any physician can write a prescription for any patient. & (10)Each prescription has a date and a quantity associated with it. & (11)When a prescription is filled, we want to track the pharmacy that filled it and the date that it was filled.

A 'prescription' table with an RX number attribute as a primary key, along with prescribed_on_date, filled_on_date and quantity attributes. This table will be linked through relationships to the 'doctor', 'patient', 'drug' and 'pharmacy' tables.

Each patient can have multiple prescriptions and each prescription can only be for one patient. A doctor can write multiple prescriptions but each prescription can only be written by one doctor. A pharmacy can fill many prescriptions, but any given prescription can only be filled by one pharmacy.

(12)Pharmaceutical companies have long-term contracts with pharmacies. A pharmaceutical company can contract with several pharmacies, and a pharmacy can contract with several pharmaceutical companies. For each contract, we want to record the start date, an end date, and the text of the contract.

Each pharmaceutical company can make contracts with zero or many pharmacies and each pharmacy can make contracts with zero or many pharmaceutical companies.

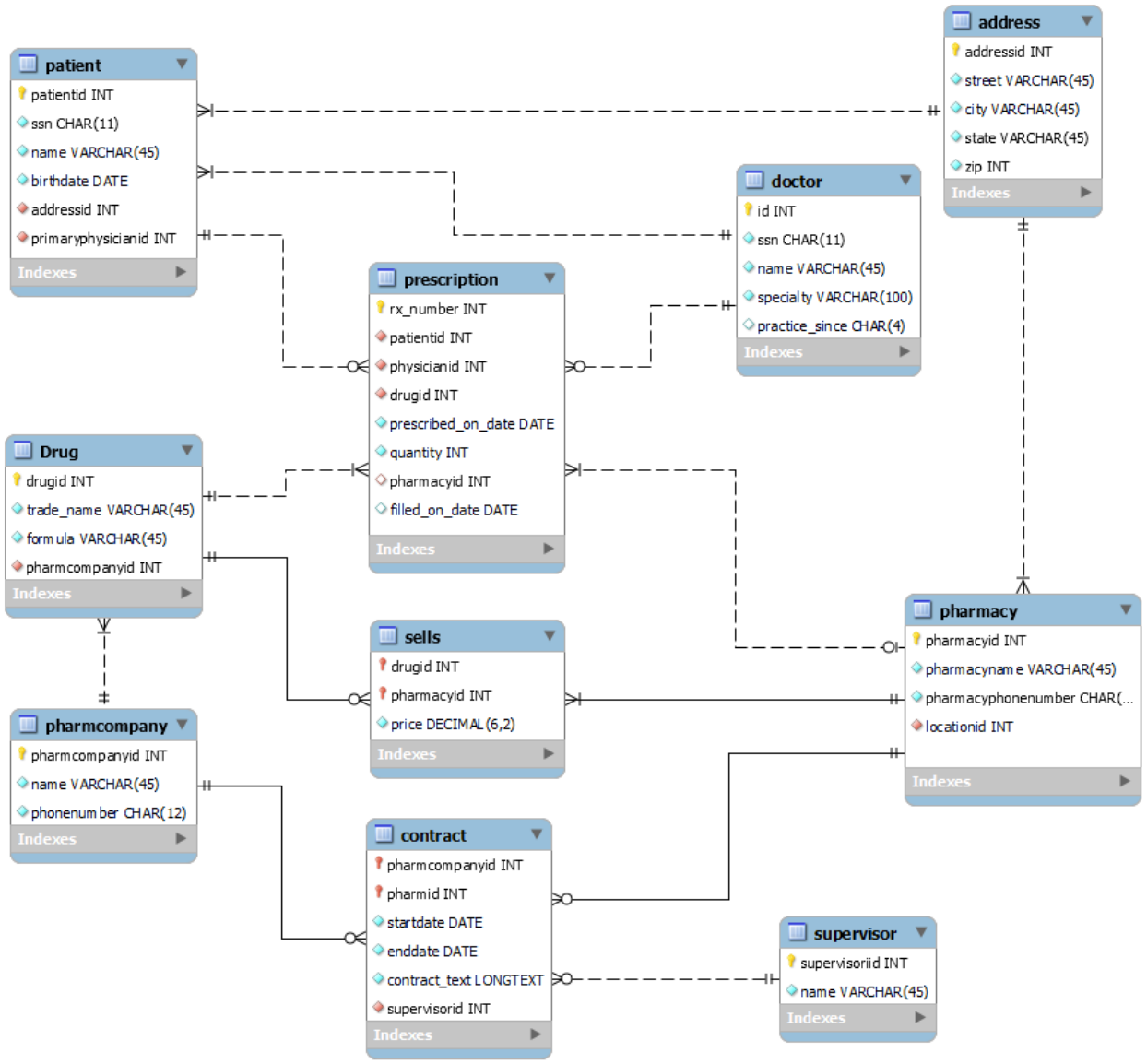
A 'contract' table that identifies the relationship and agreements between the two will be uniquely identified by the respective party's IDs. Additional information such as the contract's start date, end date, contractual text, and contract supervisor will also be included in this table.

(13)Pharmacies appoint a supervisor for each contract. Every contract has a supervisor that can change over time. A supervisor may be a supervisor for multiple contracts.

A 'supervisor' table will list names of supervisors. Each contract must have one supervisor, and each supervisor can oversee zero or many contracts.

Database Design

ER-Model



Normalization Considerations

While designing this database, we strived to reach 3NF for each relation in the database. Due to the top-down nature of our design, there were no circumstances where we chose to forgo designing to 3NF. To ensure tables were in 3NF, we first made sure each table was a relation, and then that there were no partial or transitive dependencies within each one.

Additional Assumptions and Constraints (Entities, attributes, and relationships addressed in requirements section)

- For attributes with varchar datatypes, we decided to limit the length to 45 characters as a standard across the board.
- Social Security Numbers are constrained to the char datatype of length 11.
- Phone numbers are constrained to the char datatype of length 12.
- Contract text is of type LONGTEXT with a maximum size of 4GB.
- We assume that a drugs price shall not exceed $9999.99 (decimal(6,2)).
- SSN and trade_name fields will be designated UNIQUE.
- Application code will handle input validation and formatting for attributes such as age, address, and dates.

Relational schema derived from the ER model

```
-- 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 pharmacy
-- -----------------------------------------------------

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

-- -----------------------------------------------------
-- Table `pharmacy`.`doctor`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `pharmacy`.`doctor` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `ssn` CHAR(11) NOT NULL,
  `name` VARCHAR(45) NOT NULL,
  `specialty` VARCHAR(100) NOT NULL,
  `practice_since` CHAR(4) NULL,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `snn_UNIQUE` (`ssn` ASC) VISIBLE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `pharmacy`.`address`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `pharmacy`.`address` (
  `addressid` INT NOT NULL AUTO_INCREMENT,
  `street` VARCHAR(45) NOT NULL,
  `city` VARCHAR(45) NOT NULL,
  `state` VARCHAR(45) NOT NULL,
  `zip` INT NOT NULL,
  PRIMARY KEY (`addressid`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `pharmacy`.`patient`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `pharmacy`.`patient` (
  `patientid` INT NOT NULL AUTO_INCREMENT,
  `ssn` CHAR(11) NOT NULL,
  `name` VARCHAR(45) NOT NULL,
  `birthdate` CHAR(11) NOT NULL,
  `addressid` INT NOT NULL,
  `primaryphysicianid` INT NOT NULL,
  PRIMARY KEY (`patientid`),
  INDEX `fk_patient_physician1_idx` (`primaryphysicianid` ASC) VISIBLE,
  INDEX `addressid_idx` (`addressid` ASC) VISIBLE,
  UNIQUE INDEX `snn_UNIQUE` (`ssn` ASC) VISIBLE,
  CONSTRAINT `fk_patient_physician1`
    FOREIGN KEY (`primaryphysicianid`)
    REFERENCES `pharmacy`.`doctor` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `addressid`
    FOREIGN KEY (`addressid`)
    REFERENCES `pharmacy`.`address` (`addressid`)
    ON DELETE NO ACTION
```

```
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `pharmacy`.`pharmcompany`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `pharmacy`.`pharmcompany` (
  `pharmcompanyid` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(45) NOT NULL,
  `phonenumber` CHAR(12) NOT NULL,
  PRIMARY KEY (`pharmcompanyid`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `pharmacy`.`Drug`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `pharmacy`.`Drug` (
  `drugid` INT NOT NULL AUTO_INCREMENT,
  `trade_name` VARCHAR(45) NOT NULL,
  `formula` VARCHAR(45) NOT NULL,
  `pharmcompanyid` INT NOT NULL,
  PRIMARY KEY (`drugid`),
  INDEX `fk_Drug_pharmcompany1_idx` (`pharmcompanyid` ASC) VISIBLE,
  UNIQUE INDEX `tradename_UNIQUE` (`trade_name` ASC) VISIBLE,
  CONSTRAINT `fk_Drug_pharmcompany1`
    FOREIGN KEY (`pharmcompanyid`)
    REFERENCES `pharmacy`.`pharmcompany` (`pharmcompanyid`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `pharmacy`.`pharmacy`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `pharmacy`.`pharmacy` (
  `pharmacyid` INT NOT NULL AUTO_INCREMENT,
  `pharmacyname` VARCHAR(45) NOT NULL,
  `pharmacyphonenumber` CHAR(12) NOT NULL,
  `locationid` INT NOT NULL,
  PRIMARY KEY (`pharmacyid`),
  INDEX `locationid_idx` (`locationid` ASC) VISIBLE,
  CONSTRAINT `locationid`
    FOREIGN KEY (`locationid`)
    REFERENCES `pharmacy`.`address` (`addressid`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `pharmacy`.`sells`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `pharmacy`.`sells` (
  `drugid` INT NOT NULL,
  `pharmacyid` INT NOT NULL,
  `price` DECIMAL(6,2) NOT NULL,
  INDEX `fk_Drug_has_pharmacy_pharmacy1_idx` (`pharmacyid` ASC) VISIBLE,
  INDEX `fk_Drug_has_pharmacy_Drug1_idx` (`drugid` ASC) VISIBLE,
  PRIMARY KEY (`pharmacyid`, `drugid`),
  CONSTRAINT `fk_Drug_has_pharmacy_Drug1`
    FOREIGN KEY (`drugid`)
    REFERENCES `pharmacy`.`Drug` (`drugid`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Drug_has_pharmacy_pharmacy1`
    FOREIGN KEY (`pharmacyid`)
    REFERENCES `pharmacy`.`pharmacy` (`pharmacyid`)
    ON DELETE NO ACTION
```

```
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `pharmacy`.`prescription`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `pharmacy`.`prescription` (
  `rx_number` INT NOT NULL,
  `patientid` INT NOT NULL,
  `physicianid` INT NOT NULL,
  `drugid` INT NOT NULL,
  `prescribed_on_date` DATE NOT NULL,
  `quantity` INT NOT NULL,
  `pharmacyid` INT NULL,
  `filled_on_date` DATE NULL,
  PRIMARY KEY (`rx_number`),
  INDEX `physicianid_idx` (`physicianid` ASC) VISIBLE,
  INDEX `drugid_idx` (`drugid` ASC) VISIBLE,
  INDEX `patientid_idx` (`patientid` ASC) VISIBLE,
  INDEX `pharmacyid_idx` (`pharmacyid` ASC) VISIBLE,
  CONSTRAINT `patientid`
    FOREIGN KEY (`patientid`)
    REFERENCES `pharmacy`.`patient` (`patientid`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `physicianid`
    FOREIGN KEY (`physicianid`)
    REFERENCES `pharmacy`.`doctor` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `drugid`
    FOREIGN KEY (`drugid`)
    REFERENCES `pharmacy`.`Drug` (`drugid`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `pharmacyid`
    FOREIGN KEY (`pharmacyid`)
    REFERENCES `pharmacy`.`pharmacy` (`pharmacyid`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `pharmacy`.`supervisor`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `pharmacy`.`supervisor` (
  `supervisoriid` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`supervisoriid`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `pharmacy`.`contract`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `pharmacy`.`contract` (
  `pharmcompanyid` INT NOT NULL,
  `pharmid` INT NOT NULL,
  `startdate` DATE NOT NULL,
  `enddate` DATE NOT NULL,
  `contract_text` LONGTEXT NOT NULL,
  `supervisorid` INT NOT NULL,
  PRIMARY KEY (`pharmcompanyid`, `pharmid`),
  INDEX `fk_pharmcompany_has_pharmacy_pharmacy1_idx` (`pharmid` ASC) VISIBLE,
  INDEX `fk_pharmcompany_has_pharmacy_pharmcompany1_idx` (`pharmcompanyid` ASC) VISIBLE,
  INDEX `supervisorid_idx` (`supervisorid` ASC) VISIBLE,
  CONSTRAINT `pharmcompanyid`
    FOREIGN KEY (`pharmcompanyid`)
    REFERENCES `pharmacy`.`pharmcompany` (`pharmcompanyid`)
```

```
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `pharmid`
    FOREIGN KEY (`pharmid`)
    REFERENCES `pharmacy`.`pharmacy` (`pharmacyid`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `supervisorid`
    FOREIGN KEY (`supervisorid`)
    REFERENCES `pharmacy`.`supervisor` (`supervisoriid`)
    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;
```

Sample SQL Queries

1. **Get the name of all physicians who are the primary physician of 2 or more patients.**

   ```
   select d.name, d.id
   from doctor d join patient p on d.id=p.primaryphysicianid
   group by id
   having count(p.primaryphysicianid) >= 100;
   ```

2. **Get pharmacy patients' id and name who have been prescribed the medications 'xylithol' or 'manthranicin'.**

   ```
   select p.patientid, name, trade_name
   from patient p join prescription pre on p.patientid = pre.patientid
   join drug d on d.drugid = pre.drugid
   where trade_name = "Fosamax" or trade_name = "Xanax"
   group by p.patientid;
   ```

3. **Get the drug trade name and average price of the drug for each drug sold across all pharmacies.**

   ```
   select trade_name, round(avg(price), 2)
   from drug d join sells s on d.drugid = s.drugid
   group by trade_name;
   ```

4. **Get patient id and patient names of all patients whose primary physician is either 'Greg Conroy', or 'Manual Beier'.**

   ```
   select patientid, p.name, d.name
   from patient p join doctor d on p.primaryphysicianid = d.id
   where d.name = " Greg Conroy "
   union
   select patientid, p.name , d.name
   ```

```
from patient p join doctor d on p.primaryphysicianid = d.id
where d.name = " Manual Beier ";
```

5. **Get the name of the pharmaceutical company(or more) that have the highest price for a drug on the market.**

```
select name
from pharmcompany pc join drug d on pc.pharmcompanyid = d.pharmcompanyid
join sells s on s.drugid = d.drugid
where price = (select max(price) from sells);
```

Java Application Examples

Manager Report

Example bad input

```
<terminated> ManagerReport [Java Application] C:\Program Files\Ja
Enter pharmacy id: 3
Enter start date (yyyy-mm-dd): 2022-02-02
Enter end date (yyyy-mm-dd): 2021-02-02
Start date is after end date. Exiting.
```

Example good input

```
<terminated> ManagerReport [Java Application] C:\Program Files
Enter pharmacy id: 3
Enter start date (yyyy-mm-dd): 2021-05-01
Enter end date (yyyy-mm-dd): 2021-05-10

----------------------------------------
            MANAGER REPORT
----------------------------------------
Pharmacy Id: 3
Filled between: 2021-05-01 and 2021-05-10
----------------------------------------
Drug                          Quantity
----                          --------
Allegra                        31
Flonase                        6
Pravachol                      23
Nexium                         35
Aristocort                     22
Vibramycin                     78
Singulair                      18
Lantus                         90
Advair                         96
Ativan                         97
Tylenol with Codeine           97
Adderall XR                    42
Catapres                       62
Celexa                         31
Lexapro                        85
Zestoretic                     44
Yaz                            25
Topamax                        52
Flexeril                       8
Lasix                          89
----------------------------------------
```

Data Generator

```
<terminated> DataGenerate [Java Application] C:\Program Files\Java\jdk-11.0.12\bin\
Generating 10 doctors..
Generating 1000 patients (this may take a few minutes)..
Generating 5000 prescriptions..
Generating 100 pharmacy drug prices..
Complete
```

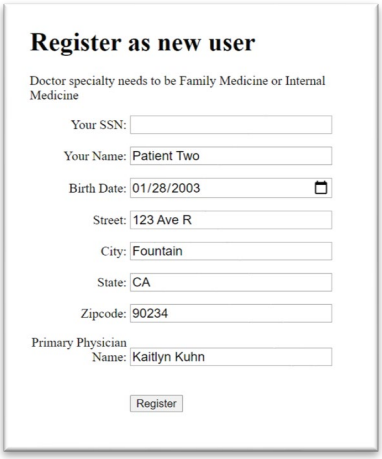
Web Application Examples

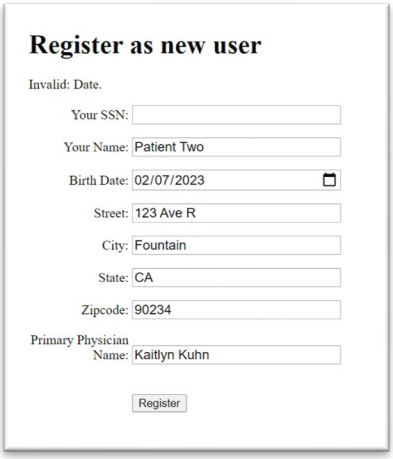
New Patient Method

All input fields are checked special characters, blank entries, and format.

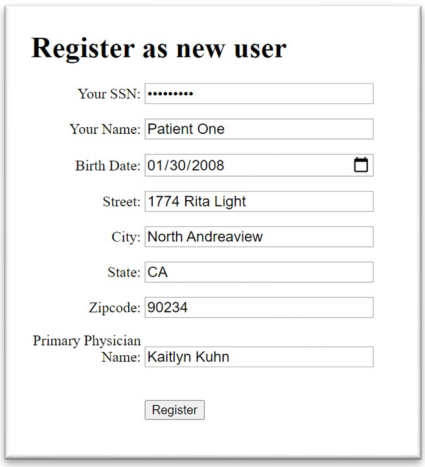
Other error messages for:

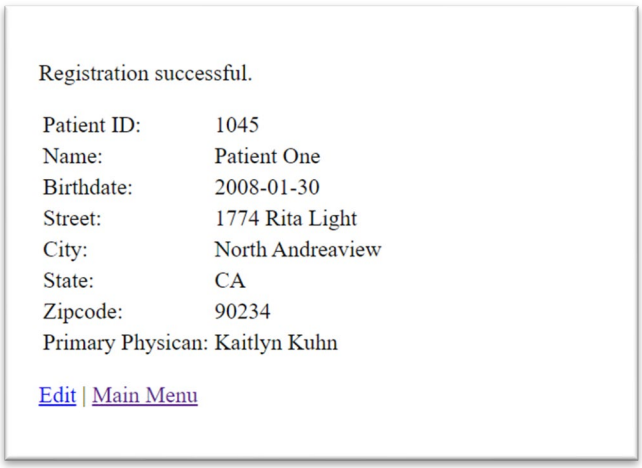
1. If the SSN is already in the system (above right)
2. Doctor is not in the system or specialty not right for age of the patient (below left)
3. Date is out of 1900-2022 range (below right)





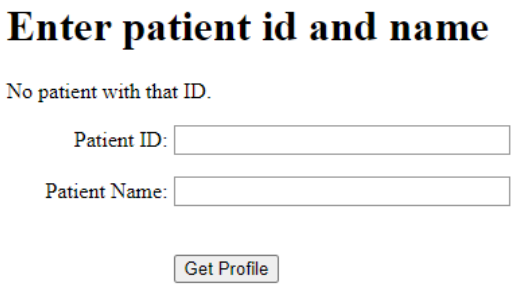
When all fields are input correctly



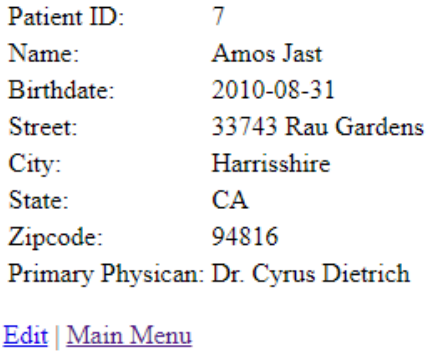


Get Patient Method

Check is done to see if patient ID exists in the database, and if the name supplied matches that ID

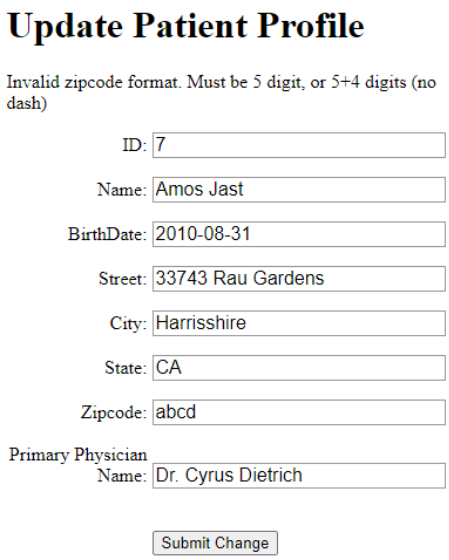


Successful retrieval of patient

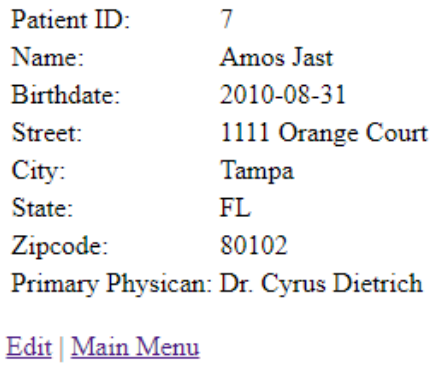


Update Patient Method

Example invalid zip code input



Example valid patient update



Conclusion

In this project we have interpreted a list of business requirements into a relational database schema utilizing an ER model diagram. To accomplish this, we independently designed our models and then came together to refine a singular approach to best solve the requirements. We gained valuable experience in collaboration and communication, as well as practice in thinking of relevant business questions and the SQL statements that might help to answer them. Creating these SQL queries without a test data set challenged us to think through our statements systematically and as a team in order to determine their correctness. In part 2 of this assignment, we gained valuable experience using our designed database in both plain java applications as well as a web application. On top of this, we collaborated through the use of shared version control software and performed vigorous testing on each other's work to ensure quality design and implementation.