TEAM 8

Database Design & Implementation Project

COURSE: CST 363
DUE DATE: January 26, 2021
INSTRUCTOR: Professor David Wisneski

Raymond Shum, Nicholas Stankovich

Change Log:
(2021-02-01) - Our project phase 2 implementation has been appended to the end of the report. It begins on Section 8.0, page 11. Please feel free to use our .PDF bookmarks to easily navigate to it.

1.0 – Phase 1 Overview
The goal of this project is to take a list of written requirements, translate the requirements into an entity-relationship (ER) diagram and map the diagram to a relational database schema. Additionally, we discuss the considerations taken in the design of our schema, the methods behind normalization, the code for implementation and business questions that would be of interest to the client.

Our project takes the perspective of a consulting team contracted by a respective firm, which appears to be either a healthcare organization or health insurance provider based on the written requirements. The written requirements are assumed to be notes taken during our initial client meeting. Following an Agile methodology, we have developed two diagrams in preparation for future meetings and requirement gathering. Continuing with our professional approach, we decided to write this report in the style of project handoff documentation. As such, bookmarks have been enabled for each header of this .PDF and we encourage using them for easy navigation between subsections.

2.0 - ER Model and Relational Database Schema
The first diagram (client ER diagram) follows the ER modeling techniques shown in the text. It includes only the attributes discussed in the requirements and provides a descriptive overview of the relationships between entities. The second diagram (technical ER diagram) shows the normalized relational database schema, which we plan to implement. Both diagrams are included due to a suggestion from the text.

The client ER diagram is more easily consumed by a non-technical client and can be used by our team to communicate our understanding of the requirements. The technical ER diagram is more robust and can be used by technical staff for support after a handoff or as documentation for future projects. The combination of diagrams can be used to explain the process of normalization to a client, to increase the effectiveness of future meetings by training the client to provide robust requirements and to provide justification for project hours during scoping.

Both diagrams are included at the end of the paper, separate from the total page count. We will cover their respective contents by first providing an overview of the client requirements, followed by per-entity discussion of design considerations and implementation.

3.0 - Requirements
The requirements provide a means for the organization to track the clients, staff, suppliers, vendors, products, and relationships (in the form of contracts) managed under its network.

From the organization's client-interfacing side, they want to track their patients, doctors, and their relationships. Each patient is tracked by their name, (unique) SSN, age, and address. Each doctor is tracked by their name, specialty, and number of years of experience. Each patient can have only one primary physician, but each doctor can be the primary physician for (minimum) one or many patients. Each patient can have many prescriptions written by many doctors and each doctor can write many prescriptions for many patients. They request the we record the date, quantity, and drug for each written prescription, storing only the data for the last prescription per drug.

From the organization's vendor side, they want to track their pharmacies. They request that the database record the name, address, and phone number of each pharmacy in their network. They also want to track the drugs sold at each pharmacy and their respective (instance specific) prices. The same drug could be sold at different pharmacies for different prices. When a prescription is filled, the organization also requests that we track the fill date and associated pharmacy.

From the organization's supplier side, they want to track the pharmaceutical companies in their network and the drugs they produce. Each pharmaceutical company is tracked by its name and phone number. Each company also produces drugs that are uniquely identified by its trade name (per company) and its formula should be stored. Each company also sells drugs at its prices.

From the organization's contract side, they also want to track the contracts between pharmaceutical companies and pharmacies. For each contract, the start date, end date, supervisor (appointed by a pharmacy) and its text should be tracked. Each pharmaceutical company can have contracts with many pharmacies and vice versa. Although the contracts are long term, the appointed supervisor can change over its lifetime.

4.0 - Design Overview

This section discusses how the schema satisfies the requirements by splitting the model into four conceptual components discussed in Section 3.0: the client-interfacing, vendor, supplier, and contract sides. It also covers relationships between entities and normalization. Assumptions are described in each subsection related to individual entities and relationships. Please reference the Technical ER diagram in Appendix 2 for all upcoming discussion.

Regarding normalization, the goal for this project was to normalize each entity to third normal form (3NF). Some entities were left denormalized in second normal form (2NF) and justification is provided when this is the case. We decided not to implement BCNF to allow for the presence of multiple candidate keys in a relation.

To expand on the use of designer keys, we approached this project from the perspective of a professional consulting team. Our client appears to be a health care provider, so we decided to design with regulatory compliance (such as with HIPAA) in mind. We decided to use designer keys as part of the database infrastructure, rather than establishing relationships using candidate keys that hold business data.

4.1 – Client-Interfacing Side

The client-interfacing side of the organization involves the patient-doctor relationship up to the point where medication is prescribed, but not where it is filled. It describes the relationship between the PATIENT, DOCTOR, PRESCRIPTION and DRUG entities.

4.1.1 - PATIENT Entity

PATIENT includes a (unique) SSN, FirstName, LastName. It uses an auto-incrementing designer key of PatientID as its PK. The address has been split into the following columns due to its status as a multivalued attribute: Street, City, State, ZipCode. Age will be a calculated column, using the difference between the current day and DateOfBirth. This is to eliminate the need update the value of the field on a yearly basis.

4.1.2 - DOCTOR Entity

DOCTOR includes a (unique) SSN, FirstName and LastName. It uses an auto-incrementing designer key of DoctorID as its PK. We decided to track YearsOfExperience as a calculated column. To do so, we added two columns: ExperienceAtHire and EmploymentStartDate. We decided to add these two columns rather than having a single ProfessionalStartDate attribute because we intend to track specialized experience rather than general experience. The doctor might not have spent the entirety of his professional career practicing his current specialty. YearsOfExperience will be determined through the following formula: CurrentYear – YEAR(EmploymentStartDate) + ExperienceAtHire.

4.1.3 - PRIMARYPHYSICIAN Weak Entity

PRIMARYPHYSICIAN is a bridge table between DOCTOR and PATIENT. Its PK is PatientID, which is a FOREIGN KEY (FK) pointing at PATIENT.PatientID. It holds another attribute, DoctorID, which is a second FK pointing at DOCTOR.DoctorID. The UNIQUE constraint is removed from this column to allow one instance of DOCTOR to be associated with more than one instance of PATIENT. As the PK, PRIMARYPHYSICIAN.PatientID is unique, meaning that only one instance of PATIENT can only appear once in the table and consequently, can only be associated with one instance of DOCTOR.

4.1.4 - PRESCRIPTION ENTITY

PRESCRIPTION is meant to represent the scrip written by the doctor, following an appointment with the patient. It holds the attributes: PrescribedDate and Quantity. Translating the requirements, we added the following FK to express its relationship with the respective entities: PATIENT.PatientID, DOCTOR.DoctorID, DRUG.DrugID. Its PK is PrescriptionID, an auto-incrementing designer key.

When designing PRESCRIPTION, we assumed that each instance of prescription should only contain one drug. If a doctor prescribes multiple drugs to a patient in one day, then a separate prescription would be required for each drug. This is to address the requirement that only the most recent instance of a unique drug prescribed to a patient would be recorded. We could associate multiple drugs to one PrescriptionID by creating a separate entity: PRESCRIPTIONLINES, holding the DrugID and Quantity attributes. However, if one drug in the order is prescribed again, then it would require altering the original order which could potentially violate regulatory guidelines (such as HIPAA)

4.1.5 Relationships

PATIENT is involved in a mandatory one-to-one (1:1) relationship with PRIMARYPHYSICIAN. This addresses the requirement that each patient has a primary physician. On the other end, DOCTOR is involved in a mandatory one-to-many (1:M) relationship with PRIMARYPHYSICIAN. This reflects the requirement that each physician has at least one patient.

PATIENT, DOCTOR and DRUG are each involved in a 1:M relationship with PRESCRIPTION. This is to address the requirement that patients can be prescribed drugs from different doctors and that an individual drug can be prescribed to many different patients. However, each unique instance of PRESCRIPTION can only be associated with a single instance of PATIENT, DOCTOR and DRUG.

4.1.7 Normalization
PATIENT is denormalized and in 2NF. This is due to the presence of the following transitive functional dependency: Street, ZipCode -> City, State. For the relation to be in 3NF, a weak entity could be created to hold these attributes: ADDRESS. The PK of this entity would be a FK pointing at PATIENT.PatientID. We decided not to normalize to 3NF to reduce the complexity of the table and to improve performance because the address fields are not likely to change often. This can be revisited if update anomalies occur due to a high number of clients sharing the same address and moving constantly.

PRIMARYPHYSICIAN is in 3NF. It has a single column PK, meaning that there can be no partial key dependencies. It also has a single attribute other than the PK meaning that there can be no transitive functional dependencies.

DOCTOR is in 3NF. It has a single column PK, meaning that there are no partial key dependencies. Although SSN is also a candidate key, the remaining attributes are not transitively dependent on DoctorID via SSN.

PRESCRIPTION is in 3NF. PrescriptionID is a single column PK, meaning that there are no partial key dependencies. There is a candidate key: PatientID, DoctorID, DrugID, PrescribedDate -> Quantity. However, quantity is not transitively dependent on the PK through this key.

4.2 – Vendor Side
The vendor side represents the pharmacies managed under the organization's network. It is the interface through which drugs move from the supplier to the patient. It describes the relationship between the PHARMACY, FILLS, SELLS and DRUG entities.

4.2.1 PHARMACY Entity
PHARMACY holds the following attributes: PharmacyID, Name, Street, City, State, ZipCode, PhoneNumber. This fulfills the requirements specifying that we track each pharmacy's name, address, and phone number. Address has been split into the following columns due to its status as a multivalued attribute: Street, City, State, ZipCode. PharmacyID is an auto-incrementing designer key used as the PK.

4.2.2 – FILLS Weak Entity
FILLS is a bridge table that expresses the action of filling a prescription at a pharmacy. It contains the following attributes: PrescriptionID, PharmacyID and DateFilled. PrescriptionID is the PK of FILLS and a FK that points to PRESCRIPTION.PrescriptionID. PharmacyID is a FK that points to the PK of PHARMACY. DateFilled is associated with a unique combination of PrescriptionID and PharmacyID, which fulfills the requirement that we track the pharmacy that fills each prescription and date of fulfillment.

4.2.3 – SELLS Weak Entity
SELLS is a bridge table that establish a relationship between unique instances of PHARMACY and DRUG, similarly to FILLS. It contains the following attributes: PharmacyID, DrugID and PharmacyPrice. PharmacyID and DrugID are the composite PK of SELLS. They are also FK pointing to PHARMACY.PharmacyID and DRUG.DrugID. Each PharmacyPrice is associated with a unique PharmacyID and DrugID combination, which fulfills the requirement that each pharmacy can sell drugs at different prices.

4.2.4 – Relationships

FILLS is involved in a 1:1 relationship with both PRESCRIPTION and PHARMACY is involved in a 1:M relationship with FILLS. This illustrates that once filled, each instance PRESCRIPTION can only have been filled by one instance of PHARMACY. Each PrescriptionID can only appear in the FILLS table once. However, a pharmacy may fill multiple prescriptions, so each PharmacyID can appear in the FILLS table more than once. However, because PrescriptionID is unique, each combination of PrescriptionID and PharmacyID will be unique.

PHARMACY and DRUG are both involved in a 1:M relationship with SELLS. This reflects the requirement that many pharmacies can sell multiple drugs at multiple prices. Many pharmacies can sell the same drug at different prices and each pharmacy can sell many different drugs. The same PharmacyID and DrugID can appear in the SELLS table multiple times, but the combination of these keys will always be unique. This allows us to associate PharmacyPrice with a unique pharmacy and drug.

4.2.5 – Normalization

FILLS is in 3NF. It has a single column PK meaning that there are no partial key dependencies. Neither attribute is transitively dependent on PrescriptionID via the other.

SELLS is in 3NF. It has a composite PK, but PharmacyPrice is dependent on the whole of the key. Neither column alone is sufficient to determine PharmacyPrice. There is only a single attribute outside of the key, so there are no transitive functional dependencies.

PHARMACY has been left denormalized in 2NF. Like PATIENT, we decided not to split the Address fields into their own tables. We did so to reduce the complexity of the schema and increase performance due to the static nature of the fields.

4.3 – Supplier Side

The supplier side describes the relationship between the PHARMACEUTICAL, PRODUCTDETAIL and DRUG entities. It describes the sale of drugs starting from the point of production. Though present in all previously described relationships, we consider DRUG to be formally part of this group.

4.3.1 – DRUG Entity

DRUG represents the medicine being supplied by pharmaceutical companies, prescribed by doctors to patients and filled by pharmacies. It holds the following attributes: DrugID, PharmaceuticalID and TradeName. DrugID is an auto-incrementing designer key used as the PK. PharmaceuticalID is a FK pointing to the PK of PHARMACEUTICAL. The TradeName attribute fulfills a specified requirement.

4.3.2 – PRODUCTDETAIL Weak Entity

PRODUCTDETAIL is a table created during the normalization of DRUG. It holds the following attributes: DrugID, Formula and Price. DrugID is the PK of PRODUCTDETAILS and a FK pointing at the PK of DRUG. Formula and Price fulfill previously specified requirements for tracking.

4.3.3 – PHARMACEUTICAL Entity

PHARMACEUTICAL holds the following attributes: PharmaceuticalID, Name, PhoneNumber. PharmaceuticalID is an auto-incrementing designer key used as the PK. The remaining attributes fulfill previously specified requirements for tracking.

4.3.4 – Relationships

PHARMACEUTICAL is involved in a 1:M relationship with DRUG. This is to address the requirement that each pharmacy sells several drugs.

DRUG is in a 1:1 identifying relationship with PRODUCTDETAIL. Each instance of PRODUCTDETAIL is associated with a unique DRUGID, which allows us to satisfy the requirement that each pharmaceutical company has a price for each drug that it sells.

4.3.5 - Normalization

DRUG, PRODUCTDETAILS and PHARMACEUTICAL are in 3NF. They have a single column PK meaning that there are no transitive dependencies. For each entity, the non-key attributes are fully dependent on the PK.

DRUG was originally composed of the combined attributes of the current DRUG and PRODUCTDETAIL entities. We split this into two tables to adhere to 3NF. According to the requirements, each TradeName uniquely identifies a drug among a pharmaceutical company's products. This would have introduced a transitive dependency where TradeName and PharmaceuticalID functionally determine formula and price. We decided to leave TradeName and PharmaceuticalID in DRUG because both attributes could not functionally determine the other. Following that, we created PRODUCTDETAIL to hold Formula and Price.

4.4 – Contract Side

The contract side represents the long-term agreements between companies and pharmacies along with their pharmacy appointed supervisors. It describes the relationship between the CONTRACT, SUPERVISOR, PHARMACEUTICAL and PHARMACY entities.

4.4.1 – CONTRACT Entity

CONTRACT holds the following attributes: ContractID PharmacyID, PharmaceuticalID, SupervisorID, StartDate, EndDate and ContractText. ContractID is an auto-incrementing designer key and PK for CONTRACT. PharmacyID, PharmaceuticalID and SupervisorID are FK pointing to the PK of PHARMACY, PHARMACEUTICAL and SUPERVISOR, respectively. StartDate, EndDate and ContractText are attributes that the organization request that we track in the requirements. PharmacyID and PharmaceuticalID represent that the contract is a relationship between PHARMACY and PHARMACEUTICAL. SupervisorID allows for the Supervisor to change during the duration of the contract and for functionally dependent fields to be added, removed, or modified without affecting the normalization of CONTRACT.

4.4.2 – SUPERVISOR Entity

SUPERVISOR holds the following attributes: SupervisorID, PharmacyID, FirstName and LastName. SupervisorID is an auto-incrementing designer key used as the PK. PharmacyID is an FK pointing at the PK of PHARMACY. It fulfills the requirement that supervisors are appointed by pharmacies. FirstName and LastName were not requested to be tracked in the requirements. However, we can use this example to inform our client that additional attributes (such as means of communication) for supervisors can be tracked if necessitated by business requirements.

4.4.3 – Relationships

PHARMACY, SUPERVISOR and PHARMACEUTICAL are involved in a 1:M relationship with CONTRACT. This means that each unique contract is associated with a single pharmacy, pharmaceutical company, and appointed supervisor. However, each instance of SUPERVISOR, PHARMACY and PHARMACEUTICAL can appear many times in the CONTRACT table. This addresses the requirement that pharmaceutical companies and pharmacies can have several contracts with each other and that a supervisor must be appointed for each contract. The SUPERVISOR relationship also allows for a pharmacy to appoint a single supervisor to many contracts if necessary.

PHARMACY is also involved in a 1:M relationship with SUPERVISOR. This satisfies the requirement that pharmacies appoint supervisors. The requirements state that each pharmacy can have multiple contracts with pharmaceutical companies, so the 1:M nature of the relationship allows a pharmacy to appoint a variable number of supervisors if necessary.

4.4.4 – Normalization

SUPERVISOR is in 3NF. There are no partial key dependencies because it holds a single column PK. Each attribute is fully dependent on the PK, meaning there are no transitive dependencies. PharmacyID is insufficient to determine the other attributes.

CONTRACT is in 3NF. There are no partial dependencies because it holds a single column PK. Each column is fully dependent on the PK, so there are no transitive dependencies. Additionally, the combination of FK values is insufficient to determine the rest of the attributes because one pharmacy, one supervisor and one pharmaceutical company can participate in multiple contracts. The combination of FK, StartDate and EndDate do not sufficiently determine ContractText because multiple contracts of equal duration could be signed on the same day.

4.5 – Constraints Not Captured by the ER Diagram

The following user-defined constraints were not captured by the ER diagram. First, we do not have to keep track of products formerly sold be a deleted pharmaceutical company. Second, only the most recently prescribed unique drug is tracked for each patient. For the first constraint, delete cascade options are set so that drugs produced by deleted pharmaceutical companies are no longer tracked across all entities. For the second constraint, we chose not to implement checks within the MySQL code. Ideally, we would like to segment these checks into a logic layer that resides between the database and the application layer. This way, our client’s support team or any engineer that takes ownership of the project after handoff would be able to make related changes at a single location, rather than having to make changes across several layers if business rules need to be modified in the future. Additional constraints such as delete options and the NOT NULL keyword are discussed in Section 5.0.

5.0 – Relational Database Schema

In this section, we will discuss the CREATE TABLE statements used to generate the relational database schema, which we have translated from the ER diagram. We have already discussed attributes, keys, normalization, and relationships in Section 4.0. Instead, this section will focus discussion on delete and update options.

5.1 – CREATE TABLE Statements

Unfortunately, the CREATE TABLE statements would put us well above the page limits of this project if we were to include them in the body of the report. We have included them in the appendix for reference, outside of the page count. They are also included in the attached .SQL file.

5.2 – Delete and Update Options

We designed the delete and update options from the perspective of a professional team consulting with a healthcare organization. Our goal was to reduce manual maintenance while preserving patient medical history when possible. All update options were set to ON UPDATE CASCADE to reduce maintenance, so the discussion primarily focuses on the use of ON DELETE SET NULL options. It is important to note that use of ON DELETE/UPDATE SET DEFAULT on MySQL requires changing the engine to InnoDB. We decided not to use SET DEFAULT options to account for potential compatibility issues. Regarding the use of ON DELETE CASCADE options, we recommend careful consideration and planning before deletions are executed. We would look to work with the client in implementing a backup schedule of daily incremental or differential backups and weekly full backups. A Disaster Recovery Plan (DRP) should also be implemented in conjunction with their backup and restore procedures. Regarding maintenance of entities where records have values set to NULL, this can be part of an automated process, the schedule of which is communicated regularly to stakeholders in advance of execution.

5.2.1 – PRIMARYPHYSICIAN Weak Entity

The PatientID FK has both options set to cascade to reduce maintenance. We cannot implement SET TO NULL due to PatientID's status as the PK. Even if we were to use a designer key, the updated records would not be useful to maintain. Our recommendation is to either maintain records for patients who are currently inactive within the network (implementing additional attributes) or to back up inactive patient medical records (primary physician, prescription history, fill and pharmacy history) before deletion.

DoctorID has the ON DELETE SET NULL option configured. This is to address the requirement that each patient must have a primary physician. Ideally, if a doctor retires or leaves the network, the logic layer would be able to flag the associated PRIMARYPHYSICIAN records that are set to NULL. An operator could identify the patient(s) through the front-end application and inform them to find a new primary physician.

5.2.2 – DRUG Entity

The PharmaceuticalID FK has both update and delete options set to cascade. The delete option was set to comply with the requirement specifying the we no longer track products from deleted pharmaceutical companies.

5.3.3 – PRODUCTDETAILS Weak Entity

The DrugID FK is set to cascade on both update and delete. This is also to comply with the requirement specified in 5.2.2.

5.3.4 – PRESCRIPTION Entity

The DoctorID FK is configured to ON DELETE SET NULL. This is to preserve patient medical history after a prescribing doctor is no longer employed by the organization.

The PatientID FK is set to ON DELETE SET NULL. This is to preserve doctor prescription history for potential analytics purposes.

The DrugID FK is set to ON DELETE SET NULL. If a drug is no longer offered by a pharmaceutical company, it does not invalidate a patient's need for it. Ideally, the logic layer will flag records with NULL DrugID values and an operator can inform either the patient or doctor through the front end.

5.3.5 – SELLS Weak Entity

The PharmacyID FK is set to ON DELETE CASCADE due to its relationship with SELLS and the DrugID FK is also set to ON DELETE CASCADE due to the requirement specified in Section 5.2.2.

5.3.6 – FILLS Weak Entity

Both PharmacyID and PrescriptionID FKs are set to ON DELETE CASCADE due to their role as the composite PK of the table. A case can be made implement a designer key to preserve the record if PrescriptionID were deleted (to preserve transaction information for the pharmacy). However, this would not be useful without the contents of the prescription. A case can also be made to preserve the table if the PharmacyID is deleted (to preserve patient medical history). This is a stronger case because the FillDate attribute is useful in combination with an existing prescription, even without an associated pharmacy. However, knowledge of how this affects internal business rules would be outside the requirements of the project and an issue that we would prioritize in future discussions with our client.

5.3.7 – SUPERVISOR Entity

The PharmacyID FK is set to ON DELETE SET NULL. Even if the pharmacy is no longer in the organization's network, a pharmaceutical company may still have an active contract with them. In this case, they would still have reason to contact the appointed supervisor. However, it can be argued that it is their responsibility to track their own existing agreements outside of our client's network.

5.3.8 – CONTRACT Entity

The PharmacyID, PharmaceuticalID and SupervisorID FKs are set to ON DELETE SET NULL. The SupervisorID option was selected to meet the requirement that the appointed supervisor can change during the duration of the contract. If the supervisor is no longer employed by the pharmacy, the logic layer would ideally flag the record and the pharmacy could be requested to appoint another. We assume that the ContractText attribute will hold information regarding the stakeholders on each contract. In case one or both entities leave the organization's network and request records for active contracts, we would still be able to produce their contract details on request.

6.0 – (5) Sample Business Questions

```
-- Which pharmaceutical company does each doctor prescribe from the most?
(This could give insight into which drug reps are visiting which doctors.)
SELECT DoctorID, FirstName, LastName, Name as TopPharm, MAX(count) as
NumScripts
FROM (SELECT d.DoctorID, d.FirstName, d.LastName, c.PharmaceuticalID,
c.Name, COUNT(*) as count
      FROM doctor d, prescription p, drug g, pharmaceutical c
      WHERE d.doctorid=p.doctorid AND p.drugid=g.drugid AND
g.pharmaceuticalid=c.pharmaceuticalid
      GROUP BY DoctorID, PharmaceuticalID) AS t
GROUP BY DoctorID;
```

```sql
-- What is the total revenue from drug sales for each pharmacy?
SELECT name, SUM(pr.quantity*s.pharmacyprice) as TotalSales
FROM pharmacy ph
INNER JOIN fills f ON f.pharmacyid = ph.pharmacyid
INNER JOIN sells s ON ph.pharmacyid = s.pharmacyid
INNER JOIN prescription pr ON pr.prescriptionid = f.prescriptionid
GROUP BY ph.name;

-- Which prescriptions have been submitted but not yet filled?
SELECT p.PatientID, p.FirstName, p.LastName, n.PrescriptionID, n.DrugID,
d.TradeName, n.PrescribedDate
FROM patient p INNER JOIN prescription n ON p.patientid=n.patientid
INNER JOIN drug d ON n.drugid=d.drugid
LEFT JOIN fills f ON n.prescriptionid=f.prescriptionid
WHERE f.datefilled IS NULL;

-- Which pharmacies and pharmaceutical companies have less than two months
left on their contracts?
SELECT p.PharmacyID, p.name AS Pharmacy, l.name AS Pharmaceutical,
c.ContractID, DATEDIFF(c.enddate, CURDATE()) AS DaysLeft
FROM pharmacy p INNER JOIN contract c ON p.pharmacyid=c.pharmacyid
INNER JOIN pharmaceutical l ON c.pharmaceuticalid=l.pharmaceuticalid
WHERE DATEDIFF(c.enddate, CURDATE()) <= 60;

-- Which pharmeceutical companies do not currently have active contracts
with each pharmacy?
SELECT p.PharmacyID, p.Name AS Pharmacy, l.PharmaceuticalID, l.Name AS
Pharmaceutical
FROM pharmacy p CROSS JOIN pharmaceutical l
LEFT JOIN contract c ON c.pharmacyid=p.pharmacyid AND
c.pharmaceuticalid=l.pharmaceuticalid
```

7.0 – Conclusion

This project represented a difficult but rewarding challenge for our team. From a professional standpoint, we learned how to gather and translate requirements from a client. We learned how to design with an organization's interests and goals in mind. Importantly, we learned how to design around gaps in our understanding, justify our decisions and plan agendas to clarify these issues in subsequent client meetings. From a technical standpoint, we learned how to build a relational schema by first understanding relationships between entities, normalizing relations, and designing with support and future project work in mind. From a team building standpoint, we continue to develop our methods for remote collaboration, understanding our strengths as individuals and subsequently applying our strengths to produce a result that is more than the sum of our parts. In terms of open questions, we mentioned that we wanted to abstract CHECKs into a logic layer that sits between the front-end application and the database. We hope to visit this in the second phase of the project but understand that our intended mode of implementation may change as we understand the requirements.

8.0 – Phase 2 Overview

We have appended our project phase 2 documentation to the end of our phase 1 report. This section covers the changes that we've made to our database and the implementation of the front-end application.

8.1 – Revisions to Database

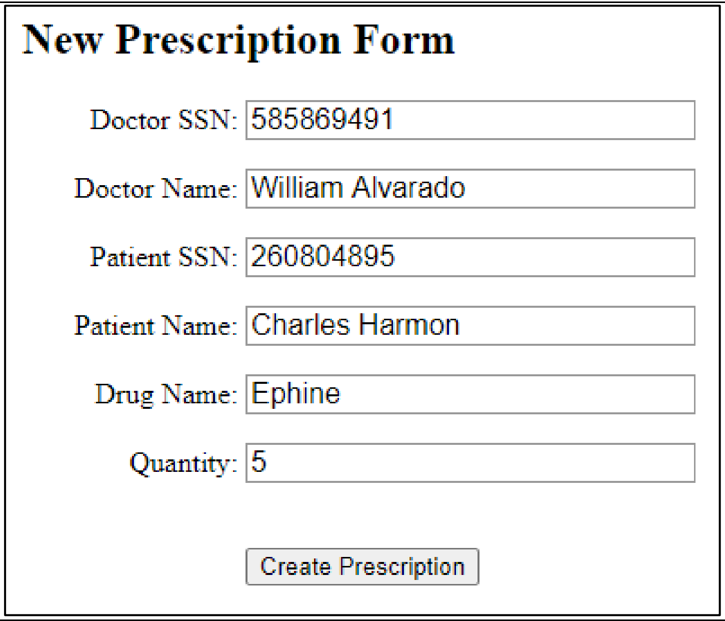
We have made the following revision to our database in Phase 2:

- On PHARMACY, collapsed the following attributes into a single address attribute: Street, City, State, ZipCode. We did so better match the provided form on the fill prescription template, but we could also consider altering the template instead.

8.2 – Front-End Application using Thymeleaf in Spring Boot

The front-end application allows doctors to prescribe medication, patients to request that prescriptions be filled, pharmacists to receive pharmacy-specific drug reports and for FDA officials to generate comprehensive reports on overall drug prescription per doctor. For each menu, error messages are descriptive and describe the exact issue that has occurred. However, this is for demonstration purposes and the specificity of the messages can be adjusted based on security concerns. General messages can be displayed on the browser while the exact error can be output to the console or logs. We have only included a subset of the implemented error messages.

8.2.1 – Write New Prescription Menu



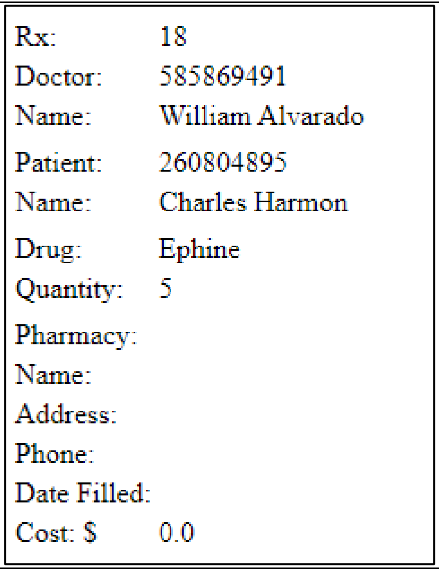
New Prescription Form

Doctor SSN:	585869491
Doctor Name:	William Alvarado
Patient SSN:	260804895
Patient Name:	Charles Harmon
Drug Name:	Ephine
Quantity:	5

Figure 8.2.1 1 - New Prescription - Input form with Correct Input

On this menu, input is validated. The controller checks if the Patient/Doctor SSN's exist and are associated with the provided name. It also checks to see if the provided drug name exists within the database. Error messages are returned in the RXid field of the Show Prescription page if any validation fails. Error messages for failed INSERT operations are also displayed.

Otherwise, this information is added as a record into the FILLS table.



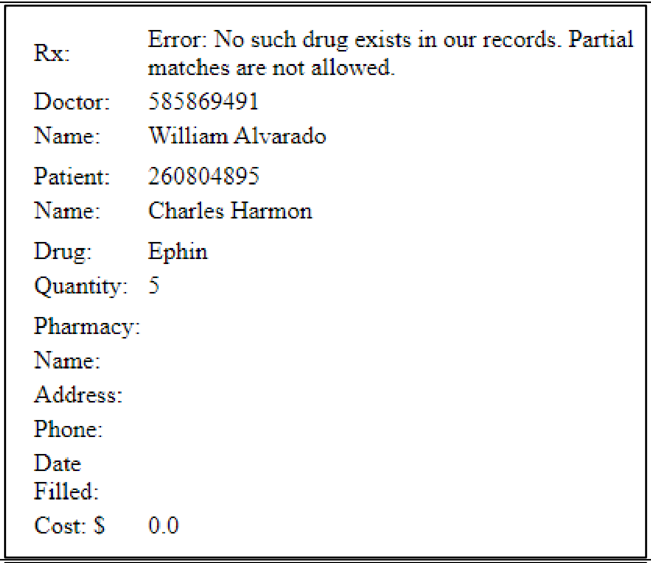
Rx:	18
Doctor:	585869491
Name:	William Alvarado
Patient:	260804895
Name:	Charles Harmon
Drug:	Ephine
Quantity:	5
Pharmacy:	
Name:	
Address:	
Phone:	
Date Filled:	
Cost: $	0.0

Figure 8.2.1 2 - New Prescription - Create Prescription Success

This screen demonstrates a successful creation of a prescription. All validation has been passed and the record is confirmed to have been added to the FILLs table.

The information is displayed back to the user, with the new RX number at the top of the screen.

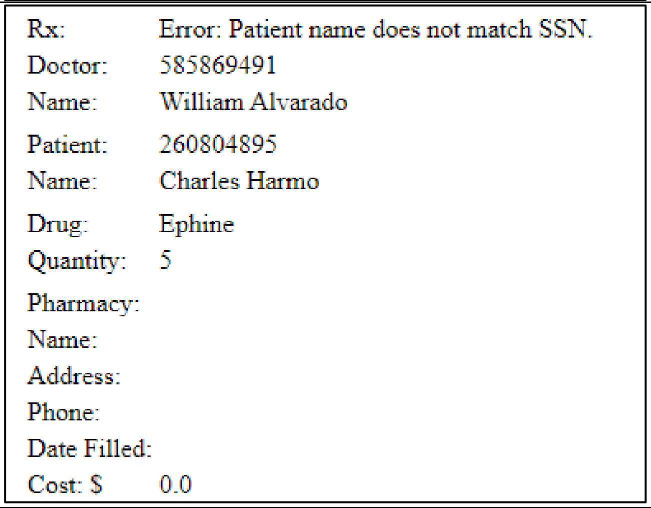
The fields in the lower half of the page remain blank until the prescription is filled.



Rx:	Error: No such drug exists in our records. Partial matches are not allowed.
Doctor:	585869491
Name:	William Alvarado
Patient:	260804895
Name:	Charles Harmon
Drug:	Ephin
Quantity:	5
Pharmacy:	
Name:	
Address:	
Phone:	
Date Filled:	
Cost: $	0.0

Figure 8.2.1 3 - New Prescription - Error - Incorrect Drug Name

This is an error message generated from attempting a partial match or misspelling the drug name on the form page.



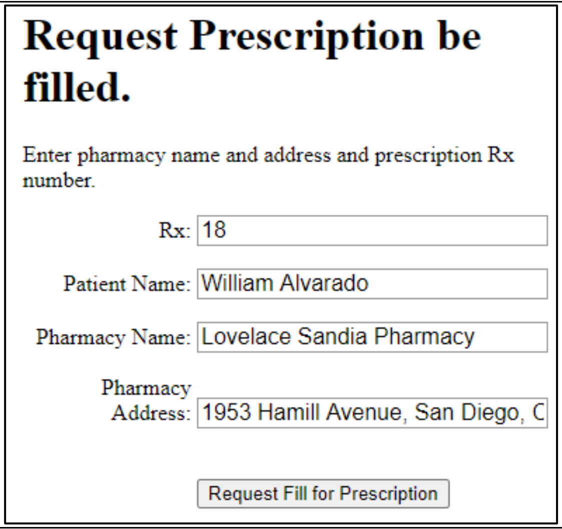
Rx:	Error: Patient name does not match SSN.
Doctor:	585869491
Name:	William Alvarado
Patient:	260804895
Name:	Charles Harmo
Drug:	Ephine
Quantity:	5
Pharmacy:	
Name:	
Address:	
Phone:	
Date Filled:	
Cost: $	0.0

Figure 8.2.1 4 - New Prescription - Error – Incorrect patient name or SSN

This is an error that displays if either the SSN does not exist, or the provided name does not match the provided SSN.

We did not get more specific than this because the exact error may allow a user to eventually match a name with an SSN. It may also be that this error should be revised to be less descriptive to protect patient privacy.

8.2.2 – Request a Prescription Be Filled Menu



Request Prescription be filled.

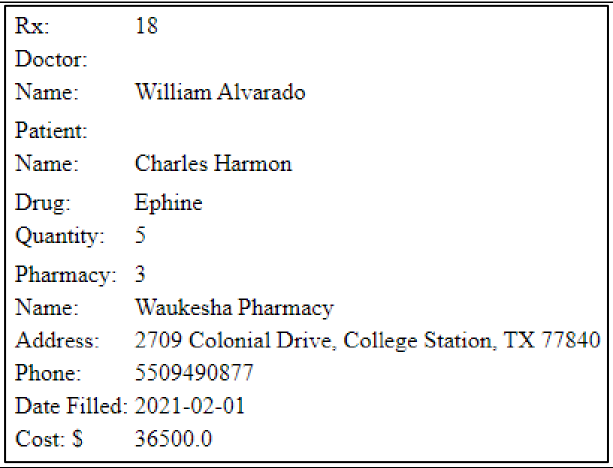
Enter pharmacy name and address and prescription Rx number.

Rx:	18
Patient Name:	William Alvarado
Pharmacy Name:	Lovelace Sandia Pharmacy
Pharmacy Address:	1953 Hamill Avenue, San Diego, C

Request Fill for Prescription

Figure 8.2.2 1 - Fill Prescription - Sample Correct Input

This is a sample of the Fill Prescription menu with correct input entered into the fields. It will validate RX# & patient name, returning an error if either the RX# was not found or is not associated with the provided name. It will also validate pharmacy name and address, returning an error if the name is not associated with the address. It will also check if the selected pharmacy sells the prescribed drug and return an error if it does not. We also decided to display an error if the prescription had been filled rather than redisplaying the success message to the user to avoid confusion.

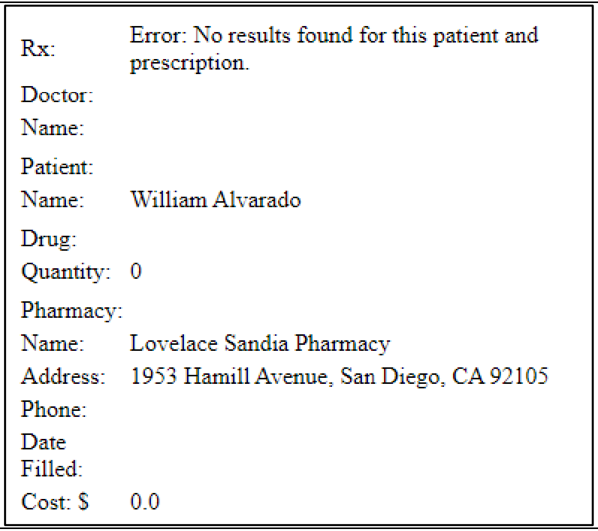


Rx:	18
Doctor:	
Name:	William Alvarado
Patient:	
Name:	Charles Harmon
Drug:	Ephine
Quantity:	5
Pharmacy:	3
Name:	Waukesha Pharmacy
Address:	2709 Colonial Drive, College Station, TX 77840
Phone:	5509490877
Date Filled:	2021-02-01
Cost: $	36500.0

Figure 8.2.2 2 - Fill Prescription - Successful Insertion of Values into Fills Table

This form displays once a prescription has been successfully filled (values inserted into the FILL table).

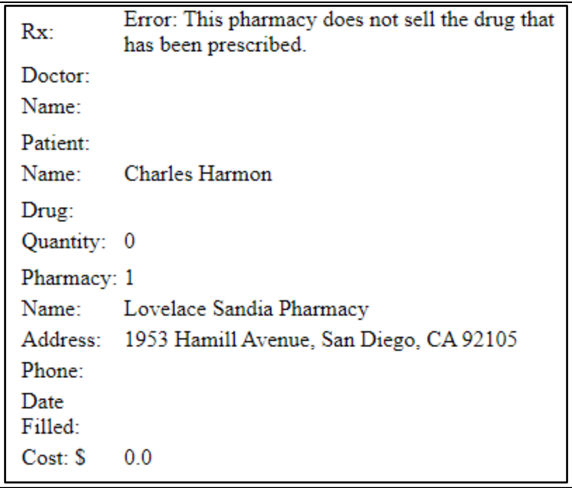
All fields on the provided template are retrieved and displayed to the user, except for SSN, which we omitted for privacy reasons. At the very least, the patient should not see the doctor's SSN.



Rx:	Error: No results found for this patient and prescription.
Doctor:	
Name:	
Patient:	
Name:	William Alvarado
Drug:	
Quantity:	0
Pharmacy:	
Name:	Lovelace Sandia Pharmacy
Address:	1953 Hamill Avenue, San Diego, CA 92105
Phone:	
Date Filled:	
Cost: $	0.0

Figure 8.2.2 3 - Fill Prescription - Error - Incorrect Patient name or Prescription Number

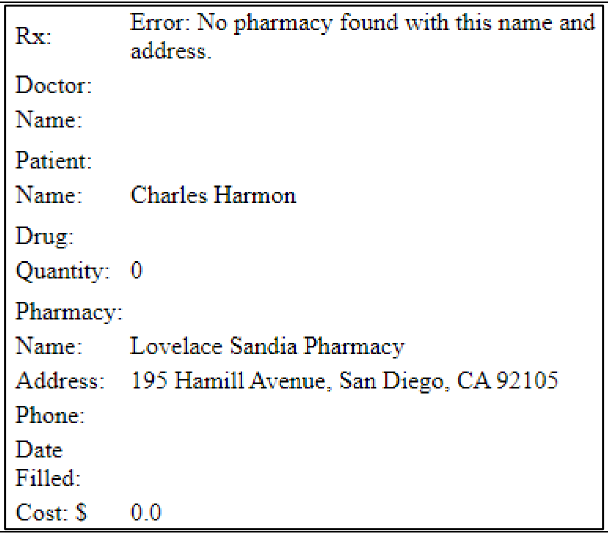
This error message displays if the patient name does not match the RX# or if the RX# is not present in the PRESCRIPTION table.



Rx: Error: This pharmacy does not sell the drug that has been prescribed.
Doctor:
Name:
Patient:
Name: Charles Harmon
Drug:
Quantity: 0
Pharmacy: 1
Name: Lovelace Sandia Pharmacy
Address: 1953 Hamill Avenue, San Diego, CA 92105
Phone:
Date Filled:
Cost: $ 0.0

Figure 8.2.2 4 - Fill Prescription - Error - Cannot fill prescription at selected pharmacy (does not sell drug)

This error message displays if the selected pharmacy does not currently sell the drug that is recorded in the PRESCRIPTION record associated with the provided RX#.



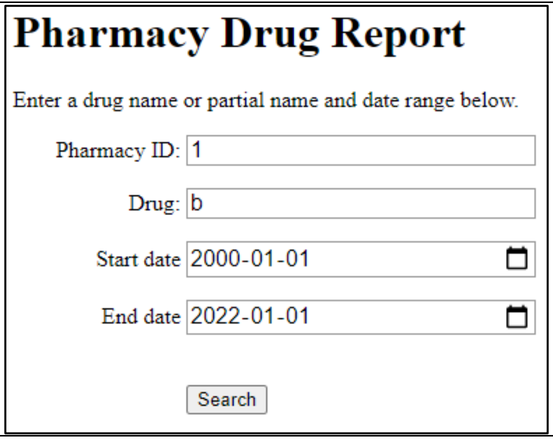
Rx: Error: No pharmacy found with this name and address.
Doctor:
Name:
Patient:
Name: Charles Harmon
Drug:
Quantity: 0
Pharmacy:
Name: Lovelace Sandia Pharmacy
Address: 195 Hamill Avenue, San Diego, CA 92105
Phone:
Date Filled:
Cost: $ 0.0

Figure 8.2.2 5 - Fill Prescription - Error - Incorrect Pharmacy Address and Name Combination

This error message displays if either the pharmacy name or address was entered incorrectly.

We considered auto-filling the fields on behalf of the user if they were able to provide either the pharmacy name or address, or allow for partial matching. However, we decided not to do this in case we have businesses with shared names or addresses in the future. We could consider implementing this by including a pharmacy id field in future iterations of this project.

8.2.3 – Report on Drug Usage Menu



Pharmacy Drug Report

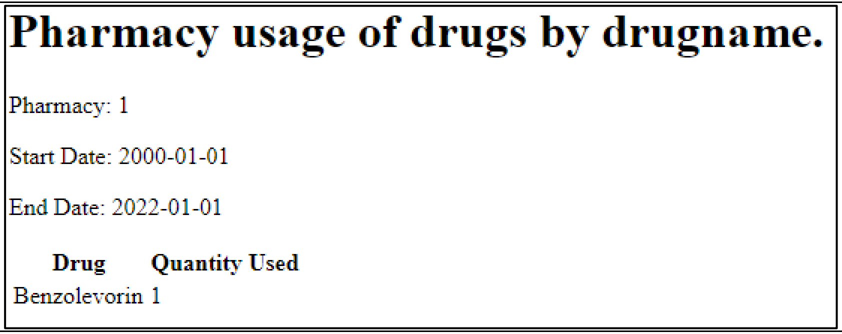
Enter a drug name or partial name and date range below.

Pharmacy ID: 1
Drug: b
Start date 2000-01-01
End date 2022-01-01

Figure 8.2.3 1 - Pharmacy Drug Report - Sample Correct Input

This is a sample pharmacy drug report menu with the user attempting a match on a partial drug name.

It will check if the pharmacy id exists and whether the drug name exists (allows partial matches). It will then retrieve any records of matching drugs having been filled within the provided time range.



Pharmacy usage of drugs by drugname.

Pharmacy: 1

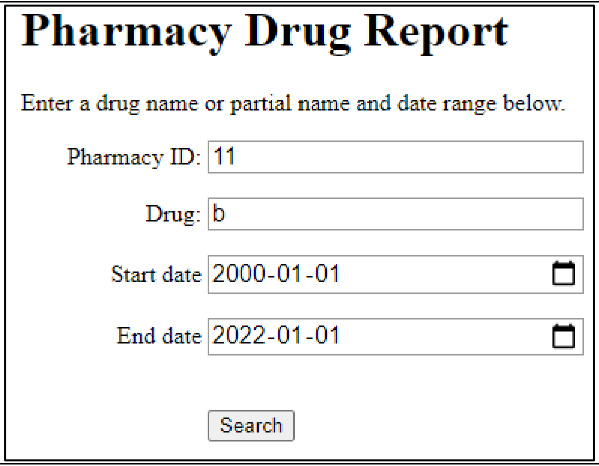
Start Date: 2000-01-01

End Date: 2022-01-01

Drug	Quantity Used
Benzolevorin	1

Figure 8.2.3 2 - Pharmacy Drug Report - Sample Correct Output

This output displays if a matching drug has been filled within the provided date range.



Pharmacy Drug Report

Enter a drug name or partial name and date range below.

Pharmacy ID: 11

Drug: b

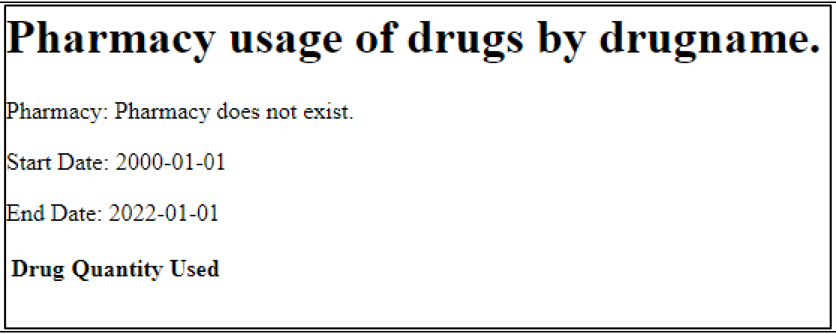
Start date 2000-01-01

End date 2022-01-01

Search

Figure 8.2.3 3 - Pharmacy Drug Report - Sample Incorrect Pharmacy ID Input

This is an input screen using a pharmacy id that does not exist.



Pharmacy usage of drugs by drugname.

Pharmacy: Pharmacy does not exist.

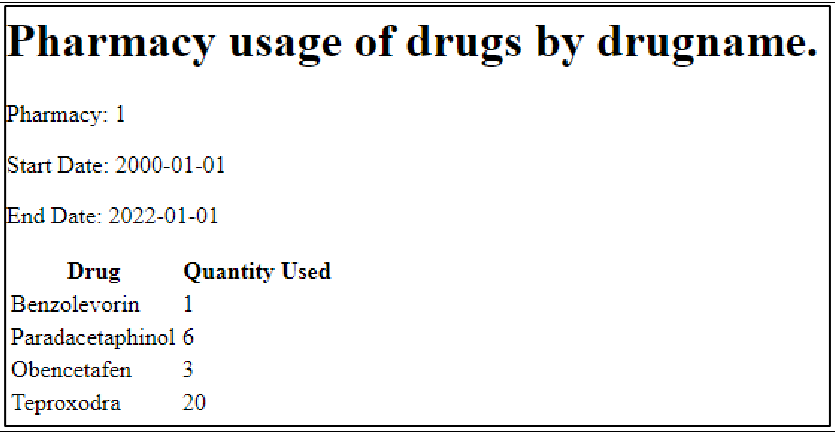
Start Date: 2000-01-01

End Date: 2022-01-01

Drug Quantity Used

Figure 8.2.3 4 - Pharmacy Drug Report - Error - Incorrect Pharmacy ID

This error displays if the pharmacy id fails validation.



Pharmacy usage of drugs by drugname.

Pharmacy: 1

Start Date: 2000-01-01

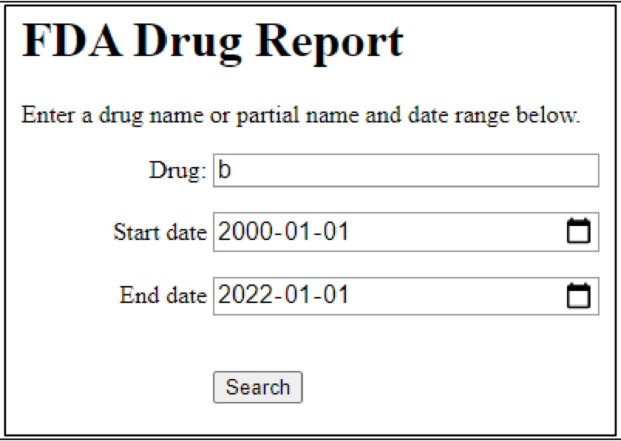
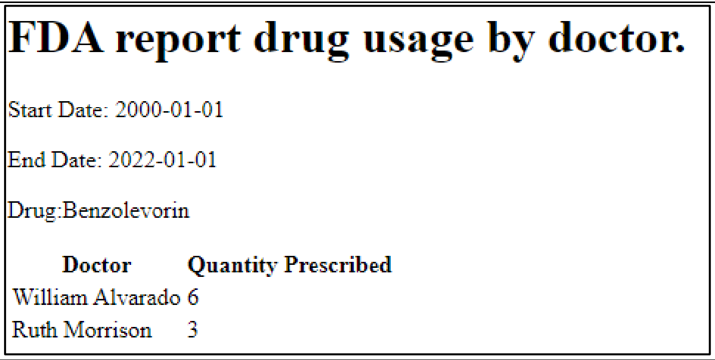
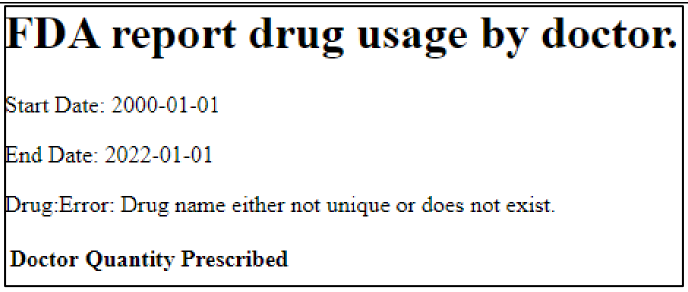
End Date: 2022-01-01

Drug	Quantity Used
Benzolevorin	1
Paradacetaphinol	6
Obencetafen	3
Teproxodra	20

Figure 8.2.3 5 - Pharmacy Drug Report - Sample Full Drug Report (Empty Drug Field)

We allowed the pharmacist to receive a full report by submitting an empty field for drug name. We can revert this by validating for empty strings or if the drug name is not unique (when matching partial names).

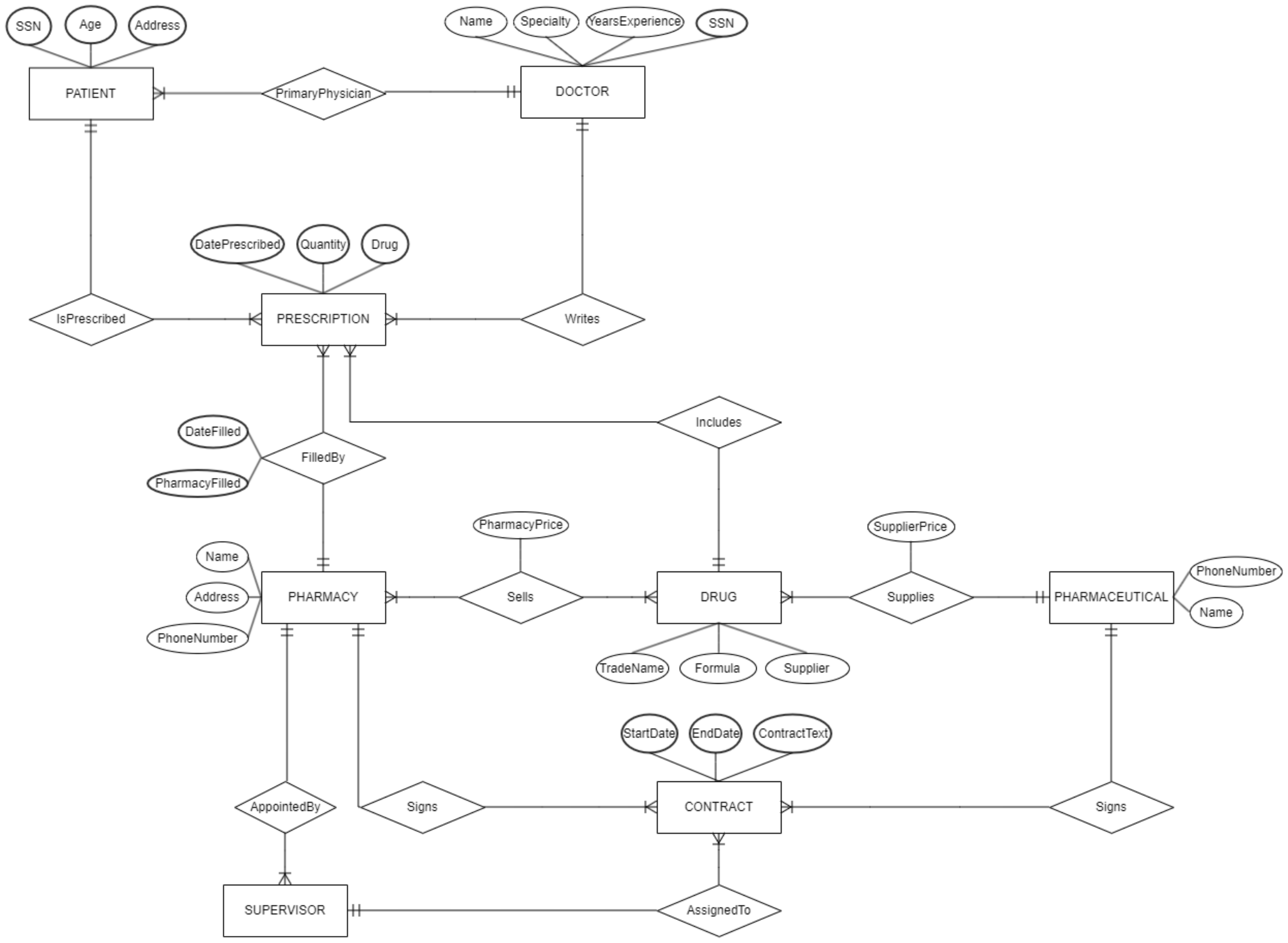
8.2.4 – FDA Reporting Menu

 FDA Drug Report Enter a drug name or partial name and date range below. Drug: b Start date 2000-01-01 End date 2022-01-01 Search	*Figure 8.2.4 1 - FDA Report - Sample Input with Partial Matching* This is an input screen for the FDA drug report where the agent is attempting to match a partial drug name. It performs similar validation to the Pharmacy Report. However, it will display an error message if the partial drug name is not sufficient to generate a unique match (due to the structure of the report).
 FDA report drug usage by doctor. Start Date: 2000-01-01 End Date: 2022-01-01 Drug:Benzolevorin Doctor Quantity Prescribed William Alvarado 6 Ruth Morrison 3	*Figure 8.2.4 2 - FDA Drug Report – Sample Correct Output* This is a sample report generated from the previous input screen. The drug name is filled in with the retrieved unique match. The doctor's name is displayed alongside the quantity of the drug they've prescribed during the provided date range.
 FDA report drug usage by doctor. Start Date: 2000-01-01 End Date: 2022-01-01 Drug:Error: Drug name either not unique or does not exist. Doctor Quantity Prescribed	*Figure 8.2.4 3- FDA Drug Report – Error - Unique Drug Name Not Identified From Partial Match* This error displays if the partial drug name was not sufficient to generate a unique match. We can consider changing this by turning the "Drug" field into a column in the lower table.

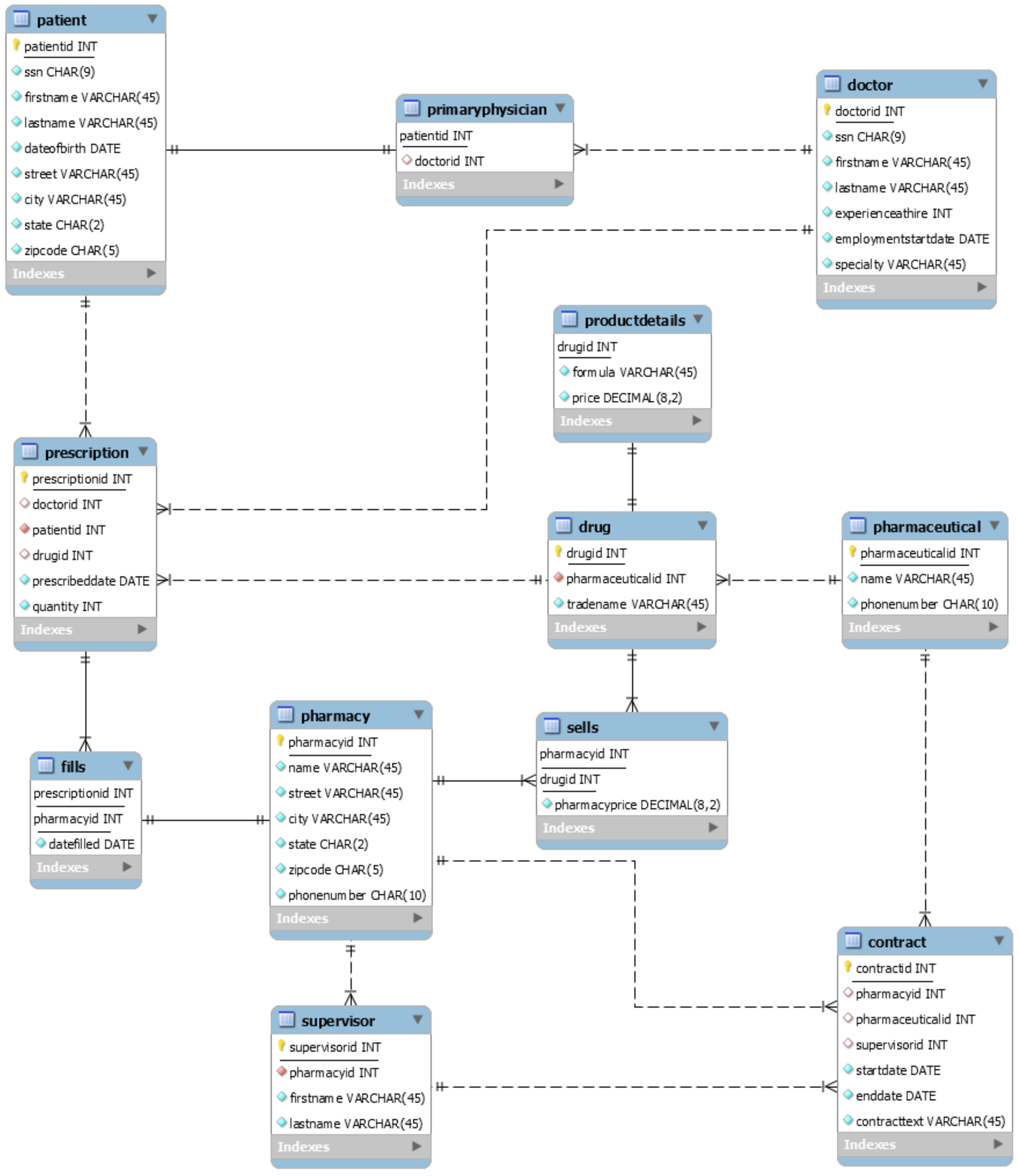
8.3 - Conclusion

This project presented another challenging but rewarding effort for the members of our team. We learned about the components of a web application: displaying the front-end interface within our browser using HTML, passing the user input into our web server using Spring Boot and Thymeleaf, performing validation and CRUD operations through Java, and connecting our web server to the MySQL database designed in the first phase. We were able to develop a comprehensive overview of the flow of information through these components. Importantly, we were able to do so while developing our product from the perspective of and on behalf of our client. By doing so, we hope that our efforts have helped us grow the skills necessary for success in the professional world.

Appendix 1: Client ER Diagram



Appendix 2 – Technical ER Diagram



Appendix 3: CREATE TABLE Statements

```
CREATE TABLE patient (
    patientid INT NOT NULL AUTO_INCREMENT,
    ssn CHAR(9) NOT NULL UNIQUE,
    firstname VARCHAR(45) NOT NULL,
    lastname VARCHAR(45) NOT NULL,
    dateofbirth DATE NOT NULL,
    street VARCHAR(45) NOT NULL,
    city VARCHAR(45) NOT NULL,
    state CHAR(2) NOT NULL,
    zipcode CHAR(5) NOT NULL,
    PRIMARY KEY (patientid)
);

CREATE TABLE doctor (
    doctorid INT NOT NULL AUTO_INCREMENT,
    ssn CHAR(9) NOT NULL UNIQUE,
    firstname VARCHAR(45) NOT NULL,
    lastname VARCHAR(45) NOT NULL,
    experienceathire INT NOT NULL,
    employmentstartdate DATE NOT NULL,
    specialty VARCHAR(45) NOT NULL,
    PRIMARY KEY (doctorid)
);

CREATE TABLE primaryphysician (
    doctorid INT,
    patientid INT NOT NULL UNIQUE,
    PRIMARY KEY (patientid),
    FOREIGN KEY (doctorid) REFERENCES doctor(doctorid)
        ON DELETE SET NULL
        ON UPDATE CASCADE,
    FOREIGN KEY (patientid) REFERENCES patient(patientid)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

CREATE TABLE pharmaceutical (
    pharmaceuticalid INT NOT NULL AUTO_INCREMENT,
    name VARCHAR(45) NOT NULL,
    phonenumber CHAR(10) NOT NULL,
    PRIMARY KEY (pharmaceuticalid)
);

CREATE TABLE drug (
    drugid INT NOT NULL AUTO_INCREMENT,
    pharmaceuticalid INT NOT NULL,
    tradename VARCHAR(45) NOT NULL,
    PRIMARY KEY (drugid),
    FOREIGN KEY (pharmaceuticalid)
        REFERENCES pharmaceutical(pharmaceuticalid)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

CREATE TABLE productdetails (
    drugid INT NOT NULL,
    formula VARCHAR(45) NOT NULL,
    price DECIMAL(8,2) NOT NULL,
    PRIMARY KEY (drugid),
    FOREIGN KEY (drugid) REFERENCES drug(drugid)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

CREATE TABLE prescription (
    prescriptionid INT NOT NULL AUTO_INCREMENT,
    doctorid INT,
    patientid INT,
    drugid INT,
    prescribeddate DATE NOT NULL,
    quantity INT NOT NULL,
    PRIMARY KEY (prescriptionid),
    FOREIGN KEY (doctorid) REFERENCES doctor(doctorid)
        ON UPDATE CASCADE
        ON DELETE SET NULL,
    FOREIGN KEY (patientid) REFERENCES patient(patientid)
        ON UPDATE CASCADE
        ON DELETE SET NULL,
    FOREIGN KEY (drugid) REFERENCES drug(drugid)
        ON UPDATE CASCADE
        ON DELETE SET NULL
);

CREATE TABLE pharmacy (
    pharmacyid INT NOT NULL AUTO_INCREMENT,
    name VARCHAR(45) NOT NULL,
    street VARCHAR(45) NOT NULL,
    city VARCHAR(45) NOT NULL,
    state CHAR(2) NOT NULL,
    zipcode CHAR(5) NOT NULL,
    phonenumber CHAR(10) NOT NULL,
    PRIMARY KEY (pharmacyid)
);

CREATE TABLE sells (
    pharmacyid INT NOT NULL,
    drugid INT NOT NULL,
    pharmacyprice DECIMAL(8,2) NOT NULL,
    PRIMARY KEY (pharmacyid, drugid),
    FOREIGN KEY (pharmacyid) REFERENCES pharmacy(pharmacyid)
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    FOREIGN KEY (drugid) REFERENCES drug(drugid)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
```

```
CREATE TABLE fills (
    prescriptionid INT NOT NULL,
    pharmacyid INT NOT NULL,
    datefilled DATE NOT NULL,
    PRIMARY KEY (prescriptionid, pharmacyid),
    FOREIGN KEY (prescriptionid) REFERENCES prescription(prescriptionid)
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    FOREIGN KEY (pharmacyid) REFERENCES pharmacy(pharmacyid)
        ON UPDATE CASCADE
        ON DELETE CASCADE
);

CREATE TABLE supervisor (
    supervisorid INT NOT NULL AUTO_INCREMENT,
    pharmacyid INT,
    firstname VARCHAR(45) NOT NULL,
    lastname VARCHAR(45) NOT NULL,
    PRIMARY KEY (supervisorid),
    FOREIGN KEY (pharmacyid) REFERENCES pharmacy(pharmacyid)
        ON DELETE SET NULL
        ON UPDATE CASCADE
);

CREATE TABLE contract (
    contractid INT NOT NULL AUTO_INCREMENT,
    pharmacyid INT,
    pharmaceuticalid INT,
    supervisorid INT,
    startdate DATE NOT NULL,
    enddate DATE NOT NULL,
    contracttext VARCHAR(45) NOT NULL,
    PRIMARY KEY (contractid),
    FOREIGN KEY (pharmacyid) REFERENCES pharmacy(pharmacyid)
    ON DELETE SET NULL
    ON UPDATE CASCADE,
    FOREIGN KEY (pharmaceuticalid)
        REFERENCES pharmaceutical(pharmaceuticalid)
        ON DELETE SET NULL
        ON UPDATE CASCADE,
    FOREIGN KEY (supervisorid) REFERENCES supervisor(supervisorid)
        ON DELETE SET NULL
        ON UPDATE CASCADE
);
```