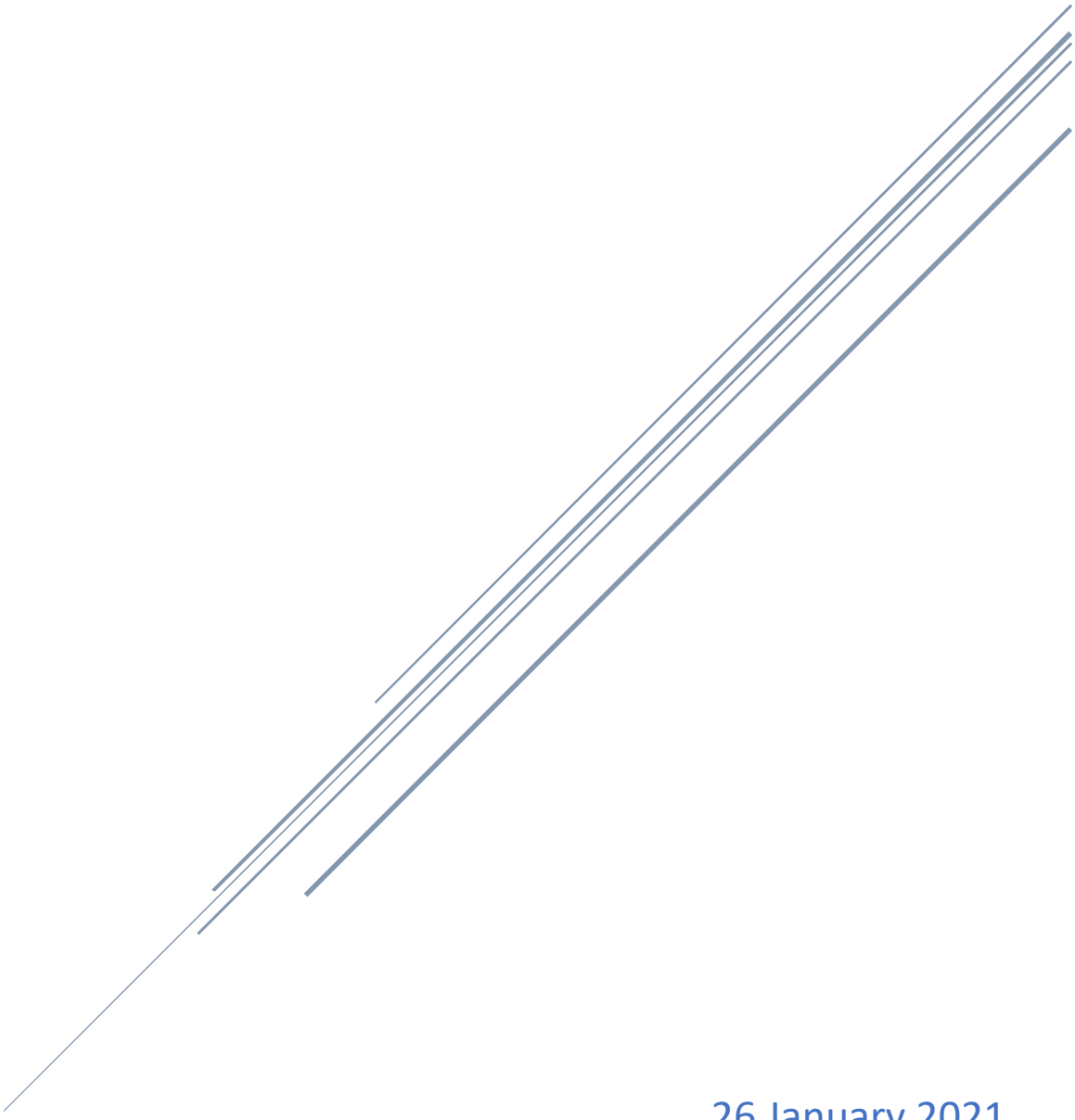
PROJECT 1

Drug Store Chain Data Design

A Collaboration by: Bryan Aguiar and Fernando Arozqueta



26 January 2021

CST 363: Intro to Database Systems

Introduction

Project 1 tasked two students (Bryan and Fernando) to develop a relational database as a consulting company for a drug store chain. Vague instructions were given from what one can infer would have been the interview process. As Fernando and Bryan have understood, the following statements are how we inferred the requirements.

The first set of info is regarding the patient. We are given a Social Security number, a name, an age, and an address. We could use the SSN as a primary key for our patient but opted to make a unique patient id instead. The patient is linked to one physician as a primary caregiver. They can meet with more than one care giver, but we opted to not document this and only provide the primary physician.

For doctors, we are given a social security number, a name, a specialty, and the years of experience. As with the patient, we wanted to avoid using the social security number as primary key, this allows us to have a doctor who is both a doctor and patient without overlapping primary keys. We are also informed that every doctor has at least one patient. We decided to ignore this last portion because we had no way how we could approach this issue. Doctors also give out prescriptions but for our understanding the information previously given is enough for one table.

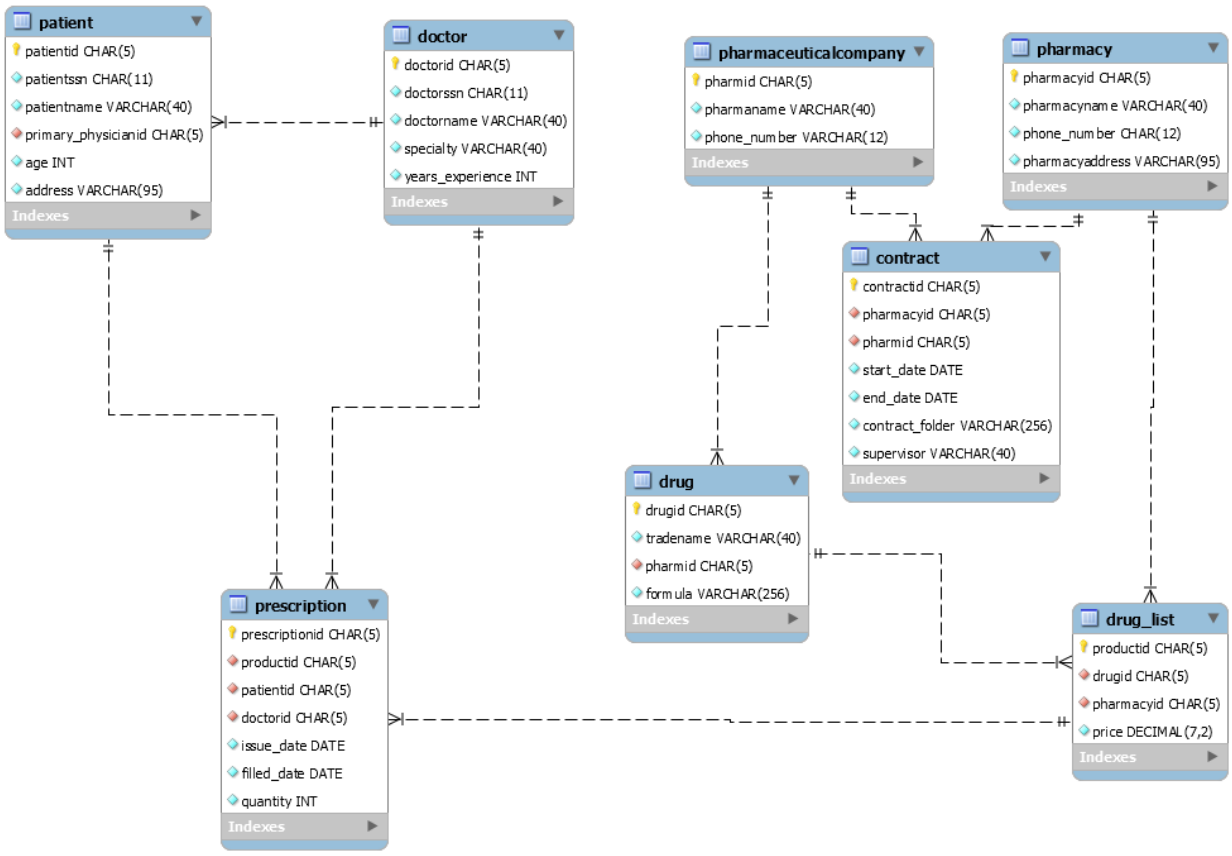
The next tidbit given is on pharmaceutical companies. Pharmaceutical companies have a name and phone number, but no other identifying information is given. We could use the name as a primary key but opted out and decided to generate a unique id for the pharmaceutical company instead. Each pharmaceutical company has a list of products it tracks, and upon deletion of a pharmaceutical company, the list of products will no longer need to be tracked and can be deleted. The list of products of each company, in this case drugs, has a unique trade name to that company and a formula. The unique trade name only applies to that company and based on the wording means other companies can use the trade name. Since we plan on keeping track using another table, we will also require a unique id for each drug as well.

For pharmacies, we are given the name, address, and phone number. Given this information, we have decided that pharmacies will also require a unique id. Pharmacies have several contracts with pharmaceutical companies and vice versa. Each contract has a supervisor appointed by the pharmacy and we are to store a start date, end date, and text of the contract. With regards to storing text, we assume that the actual contract will be stored on the same server as the database. Thus, we thought it be more prudent to just have the full file location of the contract rather that the full text of the contract.

Returning to the doctors, the doctors can provide patient with prescriptions of a certain quantity, but the prescription is not filled with the doctor but instead at a pharmacy. We are given the prescribed date and the filled date. Patients can receive more than one prescription, and can receive the same prescription more than once, but only the most recent prescription will be filled.

ER Model

Figure 1.1



In Figure 1.1(pictured above), the ER diagram for our database. Using the ER tool in MYSQL workbench, we were able to differentiate between the primary and foreign keys of each table using icons next to the column name. Each primary key is denoted using a yellow key and can be seen in instances like patientid in patient or doctorid in doctor. Red diamonds denote that the item in that column is a foreign key. The blue diamonds are used for all other types of data we will be storing.

The patient table has a few quirks that need to be discussed. First, there is a constraint on the age data of the table as the patient cannot be less than 0 years of age. The assumption is also made that a patient cannot be more than 120 years of age. Patient on the ER diagram has a many to one relationship with doctor due to the primary physician foreign key, this is based that multiple patients can have the same primary physician. Patient also has a one-to-many relationship with prescription as a patient can have multiple prescriptions.

The doctor table has a primary key we generated labeled doctorid. As stated earlier, the relationship that we have portrayed for doctor to patient is one to many. Although patients can have multiple doctors, they can only have one primary physician which is the only data we have chosen to store.

Pharmaceutical companies primary key is the pharmid which is another key we have added to the specification. Pharmaceutical companies have a one-to-many relationship with both drug and contract tables. This fits the spec as pharmaceutical companies can create multiple drugs or products and they can form multiple contracts with different pharmaceutical companies.

The primary key for pharmacy is the pharmacyid. Pharmacy like the pharmaceutical company preceding it has to relationship tables with a one-to-many relationship. Pharmacy has a one-to-many

relationship with contract and drug_list. The reasoning behind this is that pharmacies can enter contracts with different pharmaceutical companies. They also have various drugs available at their location.

The contract table has a primary key for contractid which is unique to each individual contract. The contract is what joins both pharmaceutical company and pharmacy together having both their id's as a foreign key. We also do not directly store the text of the contract but rather the location of where it is stored in the server. The contract also has an added constraint on the end and start date of the contract as the start date of the contract cannot be after the end date of the contract.

The drug table contains the drugid as the primary key. The table has a many to one relationship with pharmaceutical company as different drugs can be made by the same company, but the same exact drug cannot be made by different companies. The table also has a one-to-many relationship with drug list as the drug can exist in multiple pharmacies inventory. Drug also automatically deletes a row if the pharmaceutical company that produces the specific drug is deleted from its own table.

The primary key for drug list is the productid. The table has many-to-one relationships with both drug and pharmacy. This is because multiple pharmacies exist with multiple products and different price points. Drug list has a constraint on it's price as the drug cannot cost less than $0.

Prescription's primary key is the prescriptionid. The table has three many-to-one relationships with the following tables: patient, doctor, and druglist. Prescription is where we store who gets what prescription, when they got it, and who prescribed that medication to them. Patients can have multiple prescriptions of different products from pharmacies given to them by different doctors. The table has two constraints. The first is that the fill date cannot come before the issue date, but they can happen on the same day. The other constraint is that an order for 0 or less quantity is not allowed because that order would not be fi

Relational schema derived from ER model

Table 1.1

```
CREATE TABLE doctor
(doctorid                CHAR(5)          NOT NULL,
doctorssn                CHAR(11)         NOT NULL,
doctorname               VARCHAR(40)      NOT NULL,
Specialty                VARCHAR(40)      NOT NULL,
years_experience         INTEGER          NOT NULL,
PRIMARY KEY (doctorid) );
```

Table 1.2

```
CREATE TABLE patient
( patientid              CHAR(5)          NOT NULL,
patientssn               CHAR(11)         NOT NULL,
patientname              VARCHAR(40)      NOT NULL,
primary_physicianid      CHAR(5)          NOT NULL,
age                      INTEGER          NOT NULL,
address                  VARCHAR(95)      NOT NULL,
CHECK (age > 0),
CHECK (age < 120),
PRIMARY KEY (patientid),
```

```
FOREIGN KEY (primary_physicianid) REFERENCES doctor(doctorid));
```

Table 1.3

```
CREATE TABLE pharmaceuticalcompany
(pharmid                CHAR(5)          NOT NULL,
pharmaname              VARCHAR(40)      NOT NULL,
phone_number            VARCHAR(12)      NOT NULL,
PRIMARY KEY (pharmid) );
```

Table 1.4

```
CREATE TABLE drug
(drugid                 CHAR(5)          NOT NULL,
tradename               VARCHAR(40)      NOT NULL,
Pharmid                 CHAR(5)          NOT NULL,
Formula                 VARCHAR(256)     NOT NULL,
PRIMARY KEY (drugid),
FOREIGN KEY (pharmid) REFERENCES pharmaceuticalcompany (pharmid) ON DELETE
CASCADE);
```

Table 1.5

```
CREATE TABLE pharmacy
(pharmacyid             CHAR(5)          NOT NULL,
pharmacyname            VARCHAR(40)      NOT NULL,
phone_number            CHAR(12)         NOT NULL,
pharmacyaddress         VARCHAR(95)      NOT NULL,
PRIMARY KEY (pharmacyid));
```

Table 1.6

```
CREATE TABLE contract
(contractid             CHAR(5)          NOT NULL,
pharmacyid              CHAR(5)          NOT NULL,
pharmid                 CHAR(5)          NOT NULL,
start_date              DATE             NOT NULL,
end_date                DATE             NOT NULL,
contract_folder         VARCHAR(256)     NOT NULL,
supervisor              VARCHAR(40)      NOT NULL,
CHECK (start_date < end_date),
PRIMARY KEY (contractid),
FOREIGN KEY (pharmacyid) REFERENCES pharmacy (pharmacyid),
FOREIGN KEY (pharmid) REFERENCES pharmaceuticalcompany (pharmid));
```

Table 1.7

```
CREATE TABLE drug_list
(productid                                          CHAR(5)          NOT NULL,
Drugid                                              CHAR(5)          NOT NULL,
pharmacyid                                          CHAR(5)          NOT NULL,
Price                                               NUMERIC(7,2)     NOT NULL,
CHECK (price > 0),
PRIMARY KEY (productid),
FOREIGN KEY (pharmacyid) REFERENCES pharmacy (pharmacyid),
FOREIGN KEY (drugid) REFERENCES drug (drugid));
```

Table 1.8

```
CREATE TABLE prescription
(prescriptionid                                     CHAR(5)          NOT NULL,
productid                                           CHAR(5)          NOT NULL,
patientid                                           CHAR(5)          NOT NULL,
doctorid                                            CHAR(5)          NOT NULL,
issue_date                                          DATE             NOT NULL,
filled_date                                         DATE             NOT NULL,
quantity                                            INTEGER          NOT NULL,
CHECK (filled_date >= issue_date),
CHECK (quantity > 0),
PRIMARY KEY (prescriptionid),
FOREIGN KEY (patientid) REFERENCES patient (patientid),
FOREIGN KEY (doctorid) REFERENCES doctor (doctorid),
FOREIGN KEY (productid) REFERENCES drug_list (productid));
```

Normalized relational schema

Three tables exist in our schema that are not in first normal form. The pharmacy, patient, and doctor tables all contain rows that are not atomic values. Pharmacy address can be further broken down into street, city, zip. The same holds true for patient address. Patient and Doctor both have names that can be further broken down into first and last name. Suffix, Prefix, and Middle Name are optional but can be included as well. We decided to keep our tables denormalized because we figured that deconstructing that data did not have a significant enough effect to the overall idea of our database.

SQL Queries

Table 2.1

Display patientname, patientid, doctorname, and doctorid where patient age is greater than 30.
SELECT patientname, patientid, doctorname, doctorid FROM patient, doctor WHERE patient.primary_physicianid = doctor.doctorid and patient.age > 30;

Table 2.1 displays a patient name, their id, and a doctor name and id when the patient is older that 30 years of age. This query is answering for a check for a list of patients within a certain age group and the doctor for these patients. This query can be important because if someone were to recommend a doctor to a prospective patient within some age group, then recommending a doctor who typically works with this age group might be more beneficial

Table 2.2

Display drugid, tradename, and total sales of presciptions
SELECT drug.drugid, drug.tradename, sum(Price*Quantity) AS TotalSales FROM drug INNER JOIN drug_list ON drug.drugid = drug_list.drugid INNER JOIN prescription ON drug_list.productid = prescription.productid GROUP BY drug.drugid, tradename;

Table 2.2 displays the drugid, the tradename of the drug, and the total sales of the drug. The query is used to list the available drugs and check the with the highest amount of profitability. The query is important because pharmaceutical can use this to gage how well the drug is selling in terms of total sales.

Table 2.3

Display the tradename from prescriptions that have sold more than 3 (total quantity).
SELECT r.tradename FROM prescription p, drug_list d, drug r WHERE d.productid = p.productid AND r.drugid = d.drugid GROUP BY r.tradename HAVING SUM(quantity) > 3;

Table 2.3 takes a different approach to table 2.2. It only displays the name of the drug that is selling over whatever specified quantity is queried. The query is important like the last query in that pharmacies and pharmaceutical companies can track how well a drug is doing and whether more needs to be produced or bought.

Table 2.4

Display pharmacyname, tradename, price, price at a 20% discount from contracts with 'Pfizer'
SELECT p.pharmacyname, d.tradename, price, round(price * .80, 2) as discounted_price FROM pharmacy p, drug d, drug_list l, contract c WHERE p.pharmacyid = c.pharmacyid AND p.pharmacyid = l.pharmacyid AND l.drugid = d.drugid AND c.pharmid in (SELECT pharmid FROM pharmaceuticalcompany WHERE pharmaname = 'Pfizer') AND c.pharmid = d.pharmid;

Table 2.4 displays the pharmacy name, the trade name of the drug, the current price, and the price discounted for the pharmacies contracted with that pharmaceutical company. The query allows the user to display and view how a change in the contract to allow for a discount on its products would affect all pharmacies under contract with that company. This is important because if a pharmaceutical were to offer discounts on one of its products towards its contracted pharmacies, then they could procure a list of those pharmacies with the drugs they offer that are affected and how much the discount would change the price.

Table 2.5

Display doctor name that has less than 3 patients
SELECT d.doctorname FROM doctor d, patient p WHERE d.doctorid = p.primary_physicianid GROUP BY d.doctorid Having COUNT(d.doctorid) < 3;

Table 2.5 displays just a list of doctors that have less than a certain number of patients. The query can be used and changed to see if a doctor has no patients or too many patients. The importance of the query is that you can recommend a doctor with a freer schedule because they have fewer patients or a doctor more sought after because they have more patients.

Conclusion

The amount of work used to create databases is daunting and made us realize why teams are needed to undertake this venture. We assume an actual database is more comprehensive but with the work we have accomplished a cleared picture was painted. We also learned important questions to identify when questions a company during the interview portion of data and requirements for the database. Creating ER models in MYSQL workbench is not overly complicated but does not give us the exact diagram explained but something that can still help differentiate primary keys from other sets of data. The use of the ER diagram helps immensely when plotting our questions for SELECT statements. We were able to traverse through the diagram to reach the joins necessary to display the data we want to display. Overall the experience was enjoyable and educational.