Database Design

Jasmine Guevara, Thomas Wooten

CSU - Monterey Bay
January 26, 2021

Introduction

Designing and planning databases is necessary for developing relational databases for large companies or corporations. In this project, the database will be used by a growing drug store chain. The company needs to store patient information and doctor information. They will also need to store information about the pharmacies that they manage, and pharmaceutical companies that they have contracts with. This information includes pharmacies names, addresses, and phone numbers. Pharmacies sell several drugs and may have different pricing for the drugs. Pharmaceutical companies' contracts and information about supervisors appointed to these contracts must also be stored in this database. Information about the drugs that are sold by this large drug store chain will also be stored. Each drug must have a trade name and formula to uniquely identify the drug. Details about prescriptions such as the prescribing doctor, date, and quantity must also be stored. The prescriptions need to be tracked and details about which pharmacy filled the order and the date it was filled needs to be stored as well. All this information needs to be stored in the database, and it needs to be done methodically. This project involves developing an ER diagram, mapping the ER diagram as a database, and writing SQL queries to access table data for a growing drug store chain.

Entity Relationship Summary

Each patient must have a primary physician. Therefore the patient participation is required in the doctor table and the patient can only have one primary doctor relation. Doctors can be primary caregivers for multiple patients but also must be primary for at least one patient. Therefore doctor participation is required in the patient table.

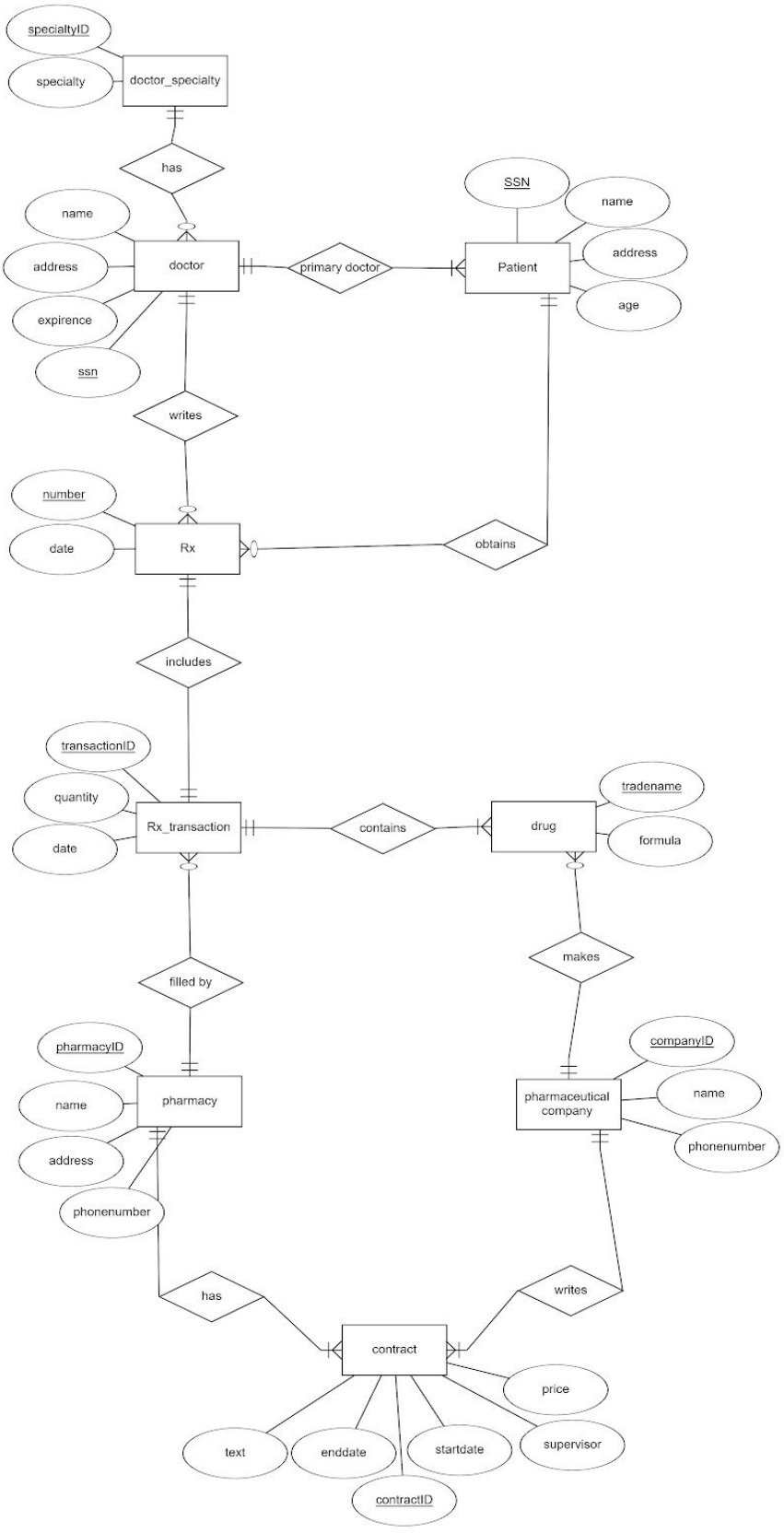
Doctors can prescribe one or more drugs for several patients, and they could also have not written any prescriptions for any one patient, therefore their participation in the prescription table is not a requirement. In addition, a patient can obtain one or more prescriptions from any doctor but is not required to have a prescription to be a patient, therefore participation from both the doctor and patient table in the prescription table is not required. However to have a prescription a doctor must write the prescription for a specific patient therefore participation is required by the doctor and patient in the prescription table. A doctor could write a prescription for one or many patients.

Each drug is produced by at most one pharmaceutical company. In order for a drug to exist it must be manufactured by a company therefore the drug table must participate in the pharmaceutical company table, and vice versa. The pharmaceutical company must produce pharmaceutical drugs to be considered a pharmaceutical company therefore participation is required by both drug and pharmaceutical company tables. Each drug has a trade name and a formula. The trade name identifies a drug uniquely from among the products of that pharmaceutical company. If a pharmaceutical company is deleted, we need not keep track of its products any longer. On a delete, we allow cascade delete to remove records from the drug table.

Pharmaceutical companies have long-term contracts with pharmacies. They appoint a supervisor for each contract that can change. A contract must always have a supervisor (non-null), and the supervisor can manage multiple contracts, so update is allowed because it is a non-keyed attribute. Participation is required in the relation between pharmacy, contract, and pharmaceutical companies. Each pharmacy can have multiple contracts with multiple

pharmaceutical companies but each contract has only one pharmacy and one pharmaceutical company.

Entity Relationship Diagram (ER)



Relational Schema Derived from the ER model

```
-- Table .`doctor_specialty``
DROP TABLE IF EXISTS .`doctor_specialty`` ;
CREATE TABLE IF NOT EXISTS .`doctor_specialty` (
  `specialtyID` INT NOT NULL AUTO_INCREMENT,
  `specialty` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`specialtyID`),
  UNIQUE INDEX `specialty_UNIQUE` (`specialty` ASC) VISIBLE)
ENGINE = InnoDB;

-- Table .`doctor`
DROP TABLE IF EXISTS .`doctor` ;
CREATE TABLE IF NOT EXISTS .`doctor` (
  `doctor_ssn` VARCHAR(11) NOT NULL,
  `name` VARCHAR(45) NULL,
  `specialtyID` INT NOT NULL DEFAULT 15,
  `expirence` INT NULL,
  PRIMARY KEY (`doctor_ssn`),
  INDEX `fc_doctor_specialty_idx` (`specialtyID` ASC) VISIBLE,
  CONSTRAINT `fc_doctor_specialty`
    FOREIGN KEY (`specialtyID`)
    REFERENCES .`doctor_specialty` (`specialtyID`)
    ON DELETE RESTRICT
    ON UPDATE RESTRICT)
ENGINE = InnoDB;

-- Table .`patient`
DROP TABLE IF EXISTS .`patient` ;
CREATE TABLE IF NOT EXISTS .`patient` (
  `patient_ssn` VARCHAR(11) NOT NULL,
  `doctor_ssn` VARCHAR(11) NOT NULL COMMENT 'primary care giver ssn',
  `name` VARCHAR(45) NULL,
  `age` INT NULL,
  `address` VARCHAR(45) NULL,
  PRIMARY KEY (`patient_ssn`),
  INDEX `fk_doctor_idx` (`doctor_ssn` ASC) VISIBLE,
  CONSTRAINT `fk_doctor`
    FOREIGN KEY (`doctor_ssn`)
    REFERENCES .`doctor` (`doctor_ssn`)
    ON DELETE RESTRICT
    ON UPDATE RESTRICT)
ENGINE = InnoDB;

-- Table .`pharmaceutical_company`
DROP TABLE IF EXISTS .`pharmaceutical_company` ;
CREATE TABLE IF NOT EXISTS .`pharmaceutical_company` (
  `companyID` INT NOT NULL,
  `name` VARCHAR(45) NULL,
  `phonenumber` VARCHAR(12) NULL,
  PRIMARY KEY (`companyID`))
ENGINE = InnoDB;
```

```
-- Table .`drug`
DROP TABLE IF EXISTS .`drug` ;
CREATE TABLE IF NOT EXISTS .`drug` (
  `tradename` VARCHAR(45) NOT NULL,
  `formula` VARCHAR(45) NULL,
  `companyID` INT NOT NULL,
  PRIMARY KEY (`tradename`),
  INDEX `fk_drug_pharmaceutical company1_idx` (`companyID` ASC) VISIBLE,
  CONSTRAINT `fk_drug_pharmaceutical company1`
    FOREIGN KEY (`companyID`)
    REFERENCES .`pharmaceutical_company` (`companyID`)
    ON DELETE RESTRICT
    ON UPDATE RESTRICT)
ENGINE = InnoDB;

-- Table .`pharmacy`
DROP TABLE IF EXISTS .`pharmacy` ;
CREATE TABLE IF NOT EXISTS .`pharmacy` (
  `pharmacyID` INT NOT NULL,
  `name` VARCHAR(45) NULL,
  `address` VARCHAR(45) NULL,
  `phonenumber` VARCHAR(12) NULL,
  PRIMARY KEY (`pharmacyID`))
ENGINE = InnoDB;

-- Table .`Rx`
DROP TABLE IF EXISTS .`Rx` ;
CREATE TABLE IF NOT EXISTS .`Rx` (
  `rxnumber` INT NOT NULL,
  `patient_ssn` VARCHAR(11) NOT NULL,
  `doctor_ssn` VARCHAR(11) NOT NULL,
  `date` DATE NOT NULL COMMENT 'date Rx was written',
  PRIMARY KEY (`rxnumber`),
  INDEX `fk_prescription_patient1_idx` (`patient_ssn` ASC) VISIBLE,
  INDEX `fk_prescription_doctor1_idx` (`doctor_ssn` ASC) VISIBLE,
  CONSTRAINT `fk_prescription_patient1`
    FOREIGN KEY (`patient_ssn`)
    REFERENCES .`patient` (`patient_ssn`)
    ON DELETE RESTRICT
    ON UPDATE RESTRICT,
  CONSTRAINT `fk_prescription_doctor1`
    FOREIGN KEY (`doctor_ssn`)
    REFERENCES .`doctor` (`doctor_ssn`)
    ON DELETE RESTRICT
    ON UPDATE RESTRICT)
ENGINE = InnoDB;

-- Table .`Rx_transaction`
DROP TABLE IF EXISTS .`Rx_transaction` ;
CREATE TABLE IF NOT EXISTS .`Rx_transaction` (
  `transactionID` INT NOT NULL,
  `rxnumber` INT NOT NULL,
```

```
  `pharmacyID` INT NOT NULL COMMENT 'Rx has been sent to pharmacy to be
filled',
  `tradename` VARCHAR(45) NOT NULL,
  `quantity` INT NOT NULL,
  `date` DATE NULL COMMENT 'if value null, Rx has not been filled',
  PRIMARY KEY (`transactionID`, `rxnumber`),
  INDEX `fk_includes_prescription1_idx` (`rxnumber` ASC) VISIBLE,
  INDEX `fk_includes_drug1_idx` (`tradename` ASC) VISIBLE,
  INDEX `fk_includes_pharmacy1_idx` (`pharmacyID` ASC) VISIBLE,
  CONSTRAINT `fk_includes_prescription1`
    FOREIGN KEY (`rxnumber`)
    REFERENCES .`Rx` (`rxnumber`)
    ON DELETE RESTRICT
    ON UPDATE RESTRICT,
  CONSTRAINT `fk_includes_drug1`
    FOREIGN KEY (`tradename`)
    REFERENCES .`drug` (`tradename`)
    ON DELETE RESTRICT
    ON UPDATE RESTRICT,
  CONSTRAINT `fk_includes_pharmacy1`
    FOREIGN KEY (`pharmacyID`)
    REFERENCES .`pharmacy` (`pharmacyID`)
    ON DELETE RESTRICT
    ON UPDATE RESTRICT)
ENGINE = InnoDB;

-- Table .`contract`
DROP TABLE IF EXISTS .`contract` ;
CREATE TABLE IF NOT EXISTS .`contract` (
  `contractID` INT NOT NULL,
  `supervisor` VARCHAR(45) NOT NULL,
  `startdate` DATE NULL,
  `enddate` DATE NULL,
  `text` VARCHAR(45) NULL,
  `pharmacyID` INT NOT NULL,
  `companyID` INT NOT NULL,
  `tradename` VARCHAR(45) NOT NULL,
  `price` INT NOT NULL,
  PRIMARY KEY (`contractID`, `supervisor`),
  INDEX `fk_contract_pharmacy1_idx` (`pharmacyID` ASC) VISIBLE,
  INDEX `fk_contract_pharmaceutical company1_idx` (`companyID` ASC) VISIBLE,
  INDEX `fk_contract_drug1_idx` (`tradename` ASC) VISIBLE,
  CONSTRAINT `fk_contract_pharmacy1`
    FOREIGN KEY (`pharmacyID`)
    REFERENCES .`pharmacy` (`pharmacyID`)
    ON DELETE RESTRICT
    ON UPDATE RESTRICT,
  CONSTRAINT `fk_contract_pharmaceutical company1`
    FOREIGN KEY (`companyID`)
    REFERENCES .`pharmaceutical_company` (`companyID`)
    ON DELETE RESTRICT
    ON UPDATE RESTRICT,
  CONSTRAINT `fk_contract_drug1`
    FOREIGN KEY (`tradename`)
```

```
    REFERENCES .`drug` (`tradename`)
    ON DELETE RESTRICT
    ON UPDATE RESTRICT)
ENGINE = InnoDB;
```

Normalized Relational Schema

The reasoning behind normalizing the design is to convert non-relational tables into relational tables to prevent duplicate data and make sure that data that is related to each other is stored in their respective tables. The most important normal forms to check for are first normal form (1NF), second normal form (2NF), and third normal form (3NF). In order to check for a particular normal form, one must progress from the first normal form to the third normal form to reach their form that they need to achieve.

As for our design, steps were performed to modify the design to make sure the tables were in a normalized form. The first step was to check for the first normal form. We checked to make sure each table has a primary key and to check if each column is unique. This was done for all our tables. The next step was to check for the second normal form. This was done by making sure that only data that is related to the table's primary key is stored within that table plus all the requirements for the first normal, which was handled. Finally, the third step was to look for in table dependencies between columns in all the tables plus all the requirements for the second form. This was done by separating doctor's specialties into its own table. In the doctor table, the doctor's specialty was needed, but it was dependent on the doctor's SSN. These were the steps used to achieve normalization.

SQL Queries

There are several questions that the drug store chain's upper management may find interesting regarding their database. The SQL queries will be described below the question posed. First of all, it would be important to find the pharmacies whose sales exceed $1000. This would be important to know to see which pharmacies may need more support or more attractive marketing.

```
SELECT p.pharmacyID, p.name, SUM(c.price * ri.quantity) AS
'TotalSales'
FROM pharmacy p
JOIN Rx_transaction ri ON p.pharmacyID = ri.pharmacyID
JOIN contract c ON p.pharmacyID = c.pharmacyID
GROUP BY p.pharmacyID
HAVING TotalSales >= 1000;
```

Secondly, upper management may find it important to know which doctors specialize in emergency medicine. This would be useful in case there is an emergency situation, and the company needs to find doctors who specialize in this.

```
SELECT doctor.name, doctor_specialty.specialty
FROM doctor
JOIN doctor_specialty ON doctor.specialtyID =
doctor_specialty.specialtyID
WHERE doctor_specialty.specialty = 'Emergency medicine';
```

Another question upper management may find interesting is: which pharmacies' contracts will be expiring before December 31st, 2021? This would be important to know since they must update their contracts before the end date and not be in violation with any laws or regulations.

```
SELECT p.pharmacyID, c.contractID, p.name, c.startdate, c.enddate
FROM pharmacy p
JOIN contract c on p.pharmacyID = c.pharmacyID
HAVING c.enddate < '20211231';
```

Upper management may also find it important to know which trade name or drug is the most expensive. This is important for upper management to know which company produces the most expensive drug. They can use this information in the future to somehow bring down the prices for their loyal customers.

```
SELECT pharmaceutical_company.name as company, drug.tradename,
contract.price
FROM contract
JOIN drug ON contract.tradename = drug.tradename
JOIN pharmaceutical_company ON drug.companyID =
pharmaceutical_company.companyID
HAVING price = (SELECT MAX(contract.price) FROM contract);
```

Finally, upper management may also find it interesting to know about how many of their customers are considered senior citizens and who is their doctor. This would be useful for offering or applying discounts or assisting the elderly patients. Also, it would be useful to know who is treating the elderly and what kinds of supplies they may need to make the experience more comfortable.

```
SELECT p.name as 'patient', p.age as 'patient age', d.name as
'primary doctor'
FROM patient p
JOIN doctor d ON p.doctor_ssn = d.doctor_ssn
JOIN doctor_specialty ds ON d.specialtyID = ds.specialtyID
WHERE p.age > 65;
```

Conclusion

In conclusion, it is useful to use an ER diagram to design databases. In this project, we used an ER diagram, mapped the ER diagram to a relational database, and developed SQL queries for upper management at the drug store chain. An ER diagram is used to map out the entities in the database and the types of relationships each entity has with each other. The ER diagram is also used to develop the relational schema and create tables in SQL. Normalization is used to improve the design of database tables. This is done by converting non-relational tables into relational tables. This is helpful for reducing duplicates and errors. Finally, the database may be used to find information, or update information if necessary, with the use of SQL queries. For this project, it was necessary to develop questions that upper management would have been interested in, and it was useful for testing if the data returned was accurate.