Drug Store Chain Database Design

Alberto Lucas, Gabriel De Leon
California State University, Monterey Bay
CST 363: Intro to Database Systems
January 26th, 2021

Introduction

The goal of this project is to develop a relational database that meets the specified criteria outlined in the provided specifications sheet. According to the client, their drug store chain is growing and needs a sophisticated database to catalog various forms of records across their pharmacies. The client provided a total of twelve informational bullet points to follow when designing the database. Upon analysis of these bullet points, the following seven entities were identified: patient, doctor, pharmaceutical company, drug, pharmacy, prescription, and contract. The bullet points also provide hints regarding the attributes of the entities, the relationships between entities, and cardinality constraints.

To efficiently design the database, the following steps will be taken. First, the customer's list of requirements will be analyzed and formalized into a detailed list of database requirements. Next, an entity relationship (ER) diagram will be drafted. This will allow all parties to visualize the entities of the database and the relationships between them. Then, a relational schema will be drafted to show how keys are related between tables. After, the database and sample records will be coded using MySQL. Once the database is coded, the database will be reviewed once more for normalization. Finally, a list of five SQL statements will be presented to demonstrate that the database works accurately. Through this design process, our consultancy group will minimize project time and maximize productivity to bring you the product your business requires.

Database Requirements

The following list outlines how the customer's requirements were interpreted and formalized into database requirements:

Requirement: Patients personal information must be collected and include their SSN, name, age, address, and an identifier for their primary care provider.

- Entity: PATIENT with attributes: PatientID, PrimaryCareID, PatientDOB, PatientSSN, PatientFirstName, PatientLastName, and PatientAddress(PatientZip, PatientStreetNumber, PatientStreetName, PatientCity, PatientState), PrimaryCareID.
- Primary Key: PatientID is the primary key, and SSN is a candidate. However, SSN is not used for security reasons.
- Foreign Key: PrimaryCareID, which points to DocID of the patient's primary care physician.
- Note: Date of birth was used instead of age, as it does not have to be recalculated every year.

Requirement: The Doctors attributes that are needed are their SSN, name, specialty, and their years of experience.

- Entity: DOCTOR with attributes: DocID, DocDateHired, DocSpecialty, DocSSN, DocPhoneNumber, DocYearsExperience, and DocName(DocFirstName, DocLastName).
- Primary Key: DocID is the primary key, DocSSN is a candidate. However, DocSSN is not used for security reasons.
- Note: A patient has exactly one primary physician. A doctor has one to many patients. A 1:M relationship exists. Date hired is obtained so that years of experience (which includes previous experience) can be incremented on a yearly basis.

Requirement: Pharmaceutical companies on file need an identifying name and a phone number.

- Entity: PHARMACOMP with attributes: PCName and PCPhoneNumber

- Primary key: PCName is the primary key, no two Pharmaceutical Companies should hold the exact same name because of copyright laws allowing this to be a unique identifier for each pharmaceutical company.

Requirement: Drugs on file need an identifying trade name, their formula, and an identifying name for the pharmaceutical company who procures it.

- Entity: DRUG with attributes: DrugName and DrugFormula, PCName
- Primary key: DrugName is the primary key, no two drugs should have the same name as copyright would prevent such duplication.
- Foreign key: PCName, aka the pharmaceutical company who procures the drug listed.
- DRUG and PHARMACOMP have a (1:M) relationship but the DRUG needs to have a PCName from PHARMACOMP. It is assumed that if PCName no longer exists, all drugs related to the PCName would no longer need to be tracked.

Requirement: Pharmacies on file need to have a name, address, and phone number logged.

- Entity: PHARMACY with attributes: PharmID, PharmName, PharmPhoneNumber, PharmAddress (PharmZip, PharmStreetNumber, PharmStreetName, PharmCity, PharmState)
- Primary key: PharmID is the primary key, as there can be multiple pharmacies running under the same name.

Requirement: Pharmacies sell drugs and have a price for each drug that they sell. Each pharmacy has its own price for each drug which may be different for the same drug.

- Entity: SELLS with attributes: DrugName, PharmID, Price
- Primary key: DrugName, PharmID
- Foreign key: Drugname, PhamID
- A pharmacy sells many drugs, and a drug is sold at many pharmacies. A M:N relationship exists.
- Relationship attribute: Price

Requirement: Doctors can prescribe drugs to patients and a doctor may prescribe one or more drugs to several patients. A patient may receive prescriptions from a multitude of doctors. Only the latest prescription is stored for a patient-drug combination. When prescriptions are filled the pharmacy that filled the prescription and the date when the prescription was filled should be logged.

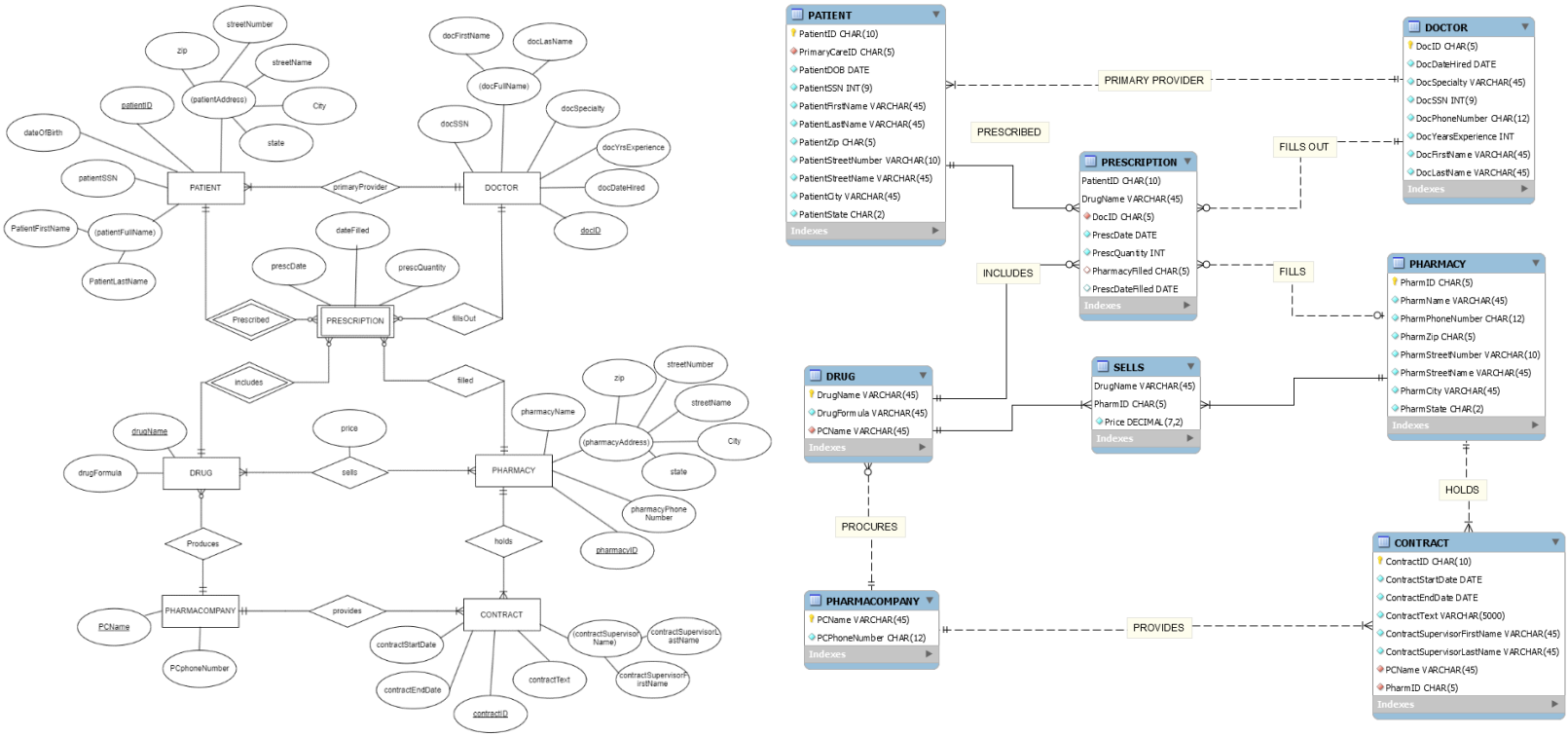
- Entity: PRESCRIPTION with attributes: PatientID, DrugName, PrescDate, PrescQuantity, DocID, PrescDateFilled, PharmacyFilled
- Primary Key: PatientID and DrugName
- Relationships: Doctors create many prescriptions, but prescriptions can only be created by one doctor (1:M). Patients receive many prescriptions, but only one prescription is provided to any given patient (1:M).
- Attributes of note: PatientID and DrugName are both primary keys, as they need to be unique for each record. PrescDateFilled and PharmacyFilled can be null because when a drug is prescribed, it has not yet been filled.
- Foreign Keys: PatientID, DrugName, DocID, PharmacyFilled

Requirement: Pharmaceutical companies and pharmacies have contracts with one another. A pharmaceutical company may hold many contracts with many pharmacies and vice versa. These contracts should have a start date, end date, and hold the text of the agreement. Each contract should have a supervisor assigned to it by the pharmacy related to the contract. The supervisor can change throughout the lifetime of the contract.

- Entity: CONTRACT with attributes: ContractID, ContractStartDate, ContractEndDate, ContractText, ContractPCName, ContractPharmacyID, and ContractSupervisorName (ContractSuperVisorFirstName and ContractSuperVisorLastName)
- Primary Key: ContractID
- Relationships: A pharmaceutical company can have many contracts, but a contract must be made by one pharmaceutical company (1:M). A pharmacy can have many contracts, but a contract must be held by one pharmacy (1:M).
- Attribute of note: ContractSupervisorName (ContractSuperVisorFirstName and ContractSuperVisorLastName) Each contract needs a supervisor keeping up with the contract, the supervisor does not need to be the same person for the entire length of the contract.

Entity Relationship (ER) Model

The following ER diagrams follow the database requirements outlined in the previous section.

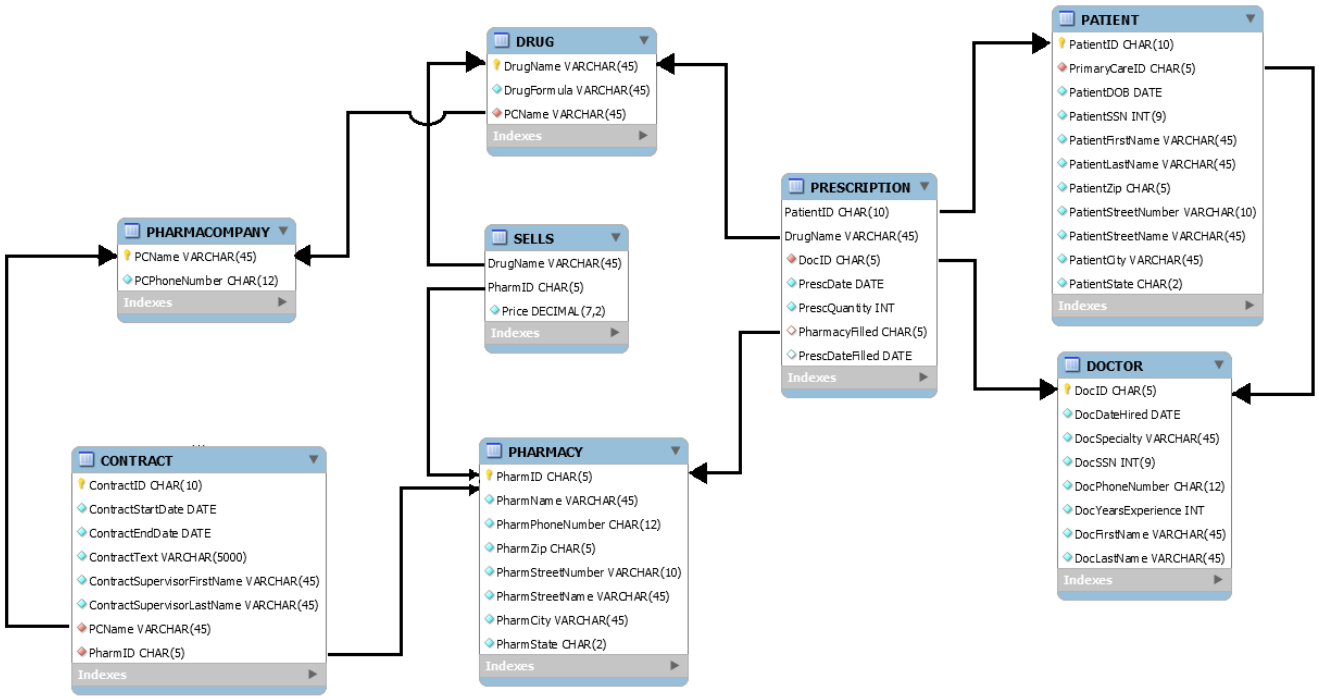


The model on the left is the ER diagram the design group originally developed. The diagram shows the relationships between all entities, and the attributes they contain. One relationship to note is that between PRESCRIPTION, PATIENT, and DRUG. PRESCRIPTION is a weak entity with two foreign keys that are also primary keys. Additionally, the relationship between DRUG and PHARMACY contains a relationship attribute due to the M:N relationship between DRUG and PHARMACY.

The diagram on the right was created using the MySQL ERR Diagram and Model tools. It adheres to the ER diagram on the right and identifies all keys within all tables. The diagram makes it easy to visualize all the attributes of each entity, and the constraints between entities. Note that SELLS becomes a table in this diagram to represent the M:N relationship from the diagram on the left.

Relational Schema

The relational schema below outlines how each table's foreign keys are related. This table is helpful for creating join statements and visualizing the relationship between tables.



Notes on attributes and their constraints:

- All tables have default update options (update restrict) and delete options (delete restrict).
- All date related attributes are presented as DATE data types.
- All phone number attributes are constrained to 12 characters to allow for dashes between sections of the phone number. It is assumed that these are local numbers and do not need a foreign country extension.
- All SSN attributes are constrained to nine digits as American SSNs only go to nine digits, it is assumed all patients have an American SSN.
- All addresses are comprised of the following attributes: Zip constrained to 5 characters, Street Number constrained to 10 characters, Street Name constrained to 45 characters, City constrained to 45 characters, and State constrained to 2 characters. This allows for almost anyone's address to be input into the database as the longest address the team found was shorter than the longest allowable address. It is assumed that all addresses refer to locations within the United States of America.
- All attributes are required (NOT NULL) for all tables except for PRESCRIPTION (prescription date filled and pharmacy filled). A prescription can exists and not yet be fulfilled so these attributes must be allowed to be left blank.

The following statements describe how the tables were created in SQL, along with minor notes not referenced above:

```
CREATE TABLE doctor
  (
     docid              CHAR(5) NOT NULL,
     docdatehired       DATE NOT NULL,
     docspecialty       VARCHAR(45) NOT NULL,
     docssn             INT(9) NOT NULL,
     docphonenumber     CHAR(12) NOT NULL,
     docyearsexperience INT NOT NULL,
     docfirstname       VARCHAR(45) NOT NULL,
     doclastname        VARCHAR(45) NOT NULL,
     PRIMARY KEY (docid)
  );
-- Note: docid is restricted to five characters as this allows a flexible doctor identification key.
```

```
CREATE TABLE pharmacy
(
     pharmid           CHAR(5) NOT NULL,
     pharmname         VARCHAR(45) NOT NULL,
     pharmphonenumber  CHAR(12) NOT NULL,
     pharmzip          CHAR(5) NOT NULL,
     pharmstreetnumber VARCHAR(10) NOT NULL,
     pharmstreetname   VARCHAR(45) NOT NULL,
     pharmcity         VARCHAR(45) NOT NULL,
     pharmstate        CHAR(2) NOT NULL,
     PRIMARY KEY (pharmid)
  );
-- Note:  pharmid is restricted to five characters as this allows a flexible doctor identification key.
```

```
CREATE TABLE pharmacompany
  (
     pcname        VARCHAR(45) NOT NULL,
     pcphonenumber CHAR(12) NOT NULL,
     PRIMARY KEY (pcname)
  );
```

```
CREATE TABLE patient
  (
     patientid          CHAR(10) NOT NULL,
     patientdob         DATE NOT NULL,
     patientssn         INT(9) NOT NULL,
     patientfirstname   VARCHAR(45) NOT NULL,
     patientlastname    VARCHAR(45) NOT NULL,
     patientzip         CHAR(5) NOT NULL,
     patientstreetnumber VARCHAR(10) NOT NULL,
     patientstreetname  VARCHAR(45) NOT NULL,
     patientcity        VARCHAR(45) NOT NULL,
     patientstate       CHAR(2) NOT NULL,
     primarycareid      CHAR(5) NOT NULL,
     PRIMARY KEY (patientid),
```

```
    FOREIGN KEY (primarycareid) REFERENCES doctor(docid)
  );
```

-- Note: patientid is restricted to 10 characters, allowing for flexible patient identification.

```
CREATE TABLE contract
  (
    contractid                  CHAR(10) NOT NULL,
    contractstartdate           DATE NOT NULL,
    contractenddate             DATE NOT NULL,
    contracttext                VARCHAR(5000) NOT NULL,
    contractsupervisorfirstname VARCHAR(45) NOT NULL,
    contractsupervisorlastname  VARCHAR(45) NOT NULL,
    pcname                      VARCHAR(45) NOT NULL,
    pharmid                     CHAR(5) NOT NULL,
    PRIMARY KEY (contractid),
    FOREIGN KEY (pcname) REFERENCES pharmacompany(pcname),
    FOREIGN KEY (pharmid) REFERENCES pharmacy(pharmid),
    CHECK (contractstartdate <= contractenddate)
  );
```

-- Note: contractid is restricted to 10 characters, allowing for flexible contract identification. contractext is
-- limited to 5000 characters allowing for ample description of the contract's terms while discouraging
-- overly convoluted wording.

```
CREATE TABLE drug
  (
    drugname    VARCHAR(45) NOT NULL,
    drugformula VARCHAR(45) NOT NULL,
    pcname      VARCHAR(45) NOT NULL,
    PRIMARY KEY (drugname),
    FOREIGN KEY (pcname) REFERENCES pharmacompany(pcname)
  );
```

```
CREATE TABLE prescription
  (
    patientid       CHAR(10) NOT NULL,
    drugname        VARCHAR(45) NOT NULL,
    prescdate       DATE NOT NULL,
    prescquantity   INT NOT NULL,
    docid           CHAR(5) NOT NULL,
    prescdatefilled DATE,
    pharmacyfilled  CHAR(5),
    PRIMARY KEY (patientid, drugname),
    FOREIGN KEY (patientid) REFERENCES patient(patientid),
    FOREIGN KEY (drugname) REFERENCES drug(drugname),
    FOREIGN KEY (docid) REFERENCES doctor(docid),
    FOREIGN KEY (pharmacyfilled) REFERENCES pharmacy(pharmid)
  );
```

```
-- Note: quantity is an unrestricted int to allow for flexible prescriptions.


CREATE TABLE sells
  (
     drugname VARCHAR(45) NOT NULL,
     pharmid  CHAR(5) NOT NULL,
     price    NUMERIC(7, 2) NOT NULL,
     PRIMARY KEY (drugname, pharmid),
     FOREIGN KEY (drugname) REFERENCES drug(drugname),
     FOREIGN KEY (pharmid) REFERENCES pharmacy(pharmid)
  );
-- Note: price is restricted to two decimal places as following American currency restrictions.
```

Normalized Relational Schema

One of the main challenges in designing the database was ensuring that all relations were in their normalized form. It was the goal of the database design team to normalize all tables to Boyce-Codd normal form (BCNF). In this normalized form, tables contain no functional dependencies other than full key functional dependencies. As shown in the previous diagrams, almost all tables were able to be normalized to BCNF.

There were several steps taken to ensure that the relational schema was in normalized form. The first step was to use a relational database as a starting point. By using a relational database, every row becomes unique and no column can contain multiple values. This automatically establishes the database as normalized in 1NF form. To convert the tables to 2NF, the design team had to ensure that no table contained partial functional dependencies. All tables were created with a full key functional dependency, which establishes the tables in the database as being in 2NF normalized form.

Most tables were also able to be brought up to 3NF normalized form. That is, most tables do not contain transitive functional dependencies in addition to being in 2NF. Note that the candidate key of SSN in the PATIENT and DOCTOR relations do not create a transitive dependency, as transitive dependencies only occur between non-key columns. It was decided that tables PATIENT and PHARMACY will remain in 2NF form (address attributes create transitive dependencies). This was done to improve database performance and reduce the complexity of the design.

Aside from the tables, the rest of the tables were able to be normalized to BCNF. Every table was revised to ensure that only the primary key/candidate keys fully determine the other columns.

SQL Queries

Using this database, many interesting queries can be made. The following five queries can be used to gather and analyze information about the drug chain:

QUERY 1: *List the name of the patient, pharmacy id, and prescription filled date for all patients who have filled their prescriptions at any CVS or Walgreens*

```
SELECT p.patientfirstname,
       p.patientlastname,
       ph.pharmid,
       rx.prescdatefilled
FROM   patient p
       INNER JOIN prescription rx
```

```
               ON p.patientid = rx.patientid
       INNER JOIN pharmacy ph
               ON rx.pharmacyfilled = ph.pharmid
WHERE  ph.pharmname IN ( 'CVS', 'Walgreens' );
```

The query above is of interest because it allows a drug chain to see how many prescriptions were filled at the specified pharmacies. This data can be refined to show how many prescriptions were filled at any given store on any range of days.

QUERY 2: *List the doctors, the doctor ids, their years of experience, and the number of prescriptions they have given out in the year 2020*

```
SELECT rx.docid,
       d.docfirstname,
       d.doclastname,
       d.docyearsexperience,
       Count(*) AS rxGiven
FROM   prescription rx
       INNER JOIN doctor d
               ON rx.docid = d.docid
WHERE  rx.prescdate LIKE '2020%'
GROUP  BY rx.docid;
```

The query above can be used by analysts to inspect the quantity of drugs prescribed by doctors. Analysts can check if a doctor is giving out too many drugs in any given year. They can also see if there is a correlation between a doctor's years of experience and the number of drugs they prescribe.

QUERY 3: *Display the average price of all the drugs sold by a specific pharmaceutical company given that the average price is greater than the average of all the drugs.*

```
WITH avgprice(pharmacompname, avgdrugprice)
     AS (SELECT phc.pcname,
                Avg(s.price)
         FROM   sells s
                INNER JOIN drug d
                        ON s.drugname = d.drugname
                INNER JOIN pharmacompany phc
                        ON phc.pcname = d.pcname
         GROUP  BY phc.pcname)
SELECT pharmacompname,
       avgdrugprice
FROM   avgprice
WHERE  avgdrugprice > (SELECT Avg(price) FROM sells);
```

This query allows an analyst to see which pharmaceutical companies are most expensive to purchase drugs from. This query can be further customized to display average drug prices across companies. This data can then be used to make sales decisions.

QUERY 4: *Display the total count of contracts that a pharmacy has with a specific pharmaceutical company.*

```
SELECT ph.pharmname,
       phc.pcname,
       Count(*) AS contractcount
FROM   contract c
```

```
       INNER JOIN pharmacompany phc
               ON phc.pcname = c.pcname
       INNER JOIN pharmacy ph
               ON ph.pharmid = c.pharmid
GROUP  BY ph.pharmname,
          phc.pcname;
```

This query shows all the pharmacies and the number of contracts that they have with any given pharmaceutical company. It can be used to quickly inspect which pharmaceutical companies provide the most contracts to pharmacies. This data can then be used for negotiation efforts.

QUERY 5: *Display a list of patient names who were prescribed drugs by Blackmart and who's prescription was filled by Walgreens. Include the drug name and formula name they were prescribed.*

```
SELECT p.patientfirstname,
       p.patientlastname,
       rx.drugname,
       d.drugformula
FROM   patient p
       INNER JOIN prescription rx
               ON p.patientid = rx.patientid
       INNER JOIN drug d
               ON rx.drugname = d.drugname
       INNER JOIN pharmacy ph
               ON rx.pharmacyfilled = ph.pharmid
WHERE  d.pcname = 'Blackmart'
       AND ph.pharmname = 'Walgreens';
```

This query can be used to see how many customers filled their prescriptions with drugs sold from a specific pharmaceutical company. It can then be further customized to provide data from specific stores or regions. This data can then be used to make future sales negotiations.

Conclusion

The database created for this drug chain was designed according to all the specified objectives. Various iterations of design were completed to create a database that collects high quality data. To design the database, the customer's specifications were first broken down into formal specifications. Afterwards, a highly detailed ER diagram was drafted using the formalized details. Next, a relational schema was created to show how keys are related between tables. Using these diagrams, the SQL database was coded. At this stage, many relations were normalized to create a more efficient database design. After designing the database, many sample records were added to test several query statements. The database's design allows for great flexibility in the queries that are made to it. Five sample queries are provided to demonstrate the power and flexibility inherent in the database. Data analysts can create detailed reports with the data obtained from the database. Through these reports, a company can make more informed decisions about its future.