Physician's Office

Stephen Bohner

Database Management

Spring 2016

Pablo Rivas

April 20st 2016

Table of Contents

Contents

Executive Summary

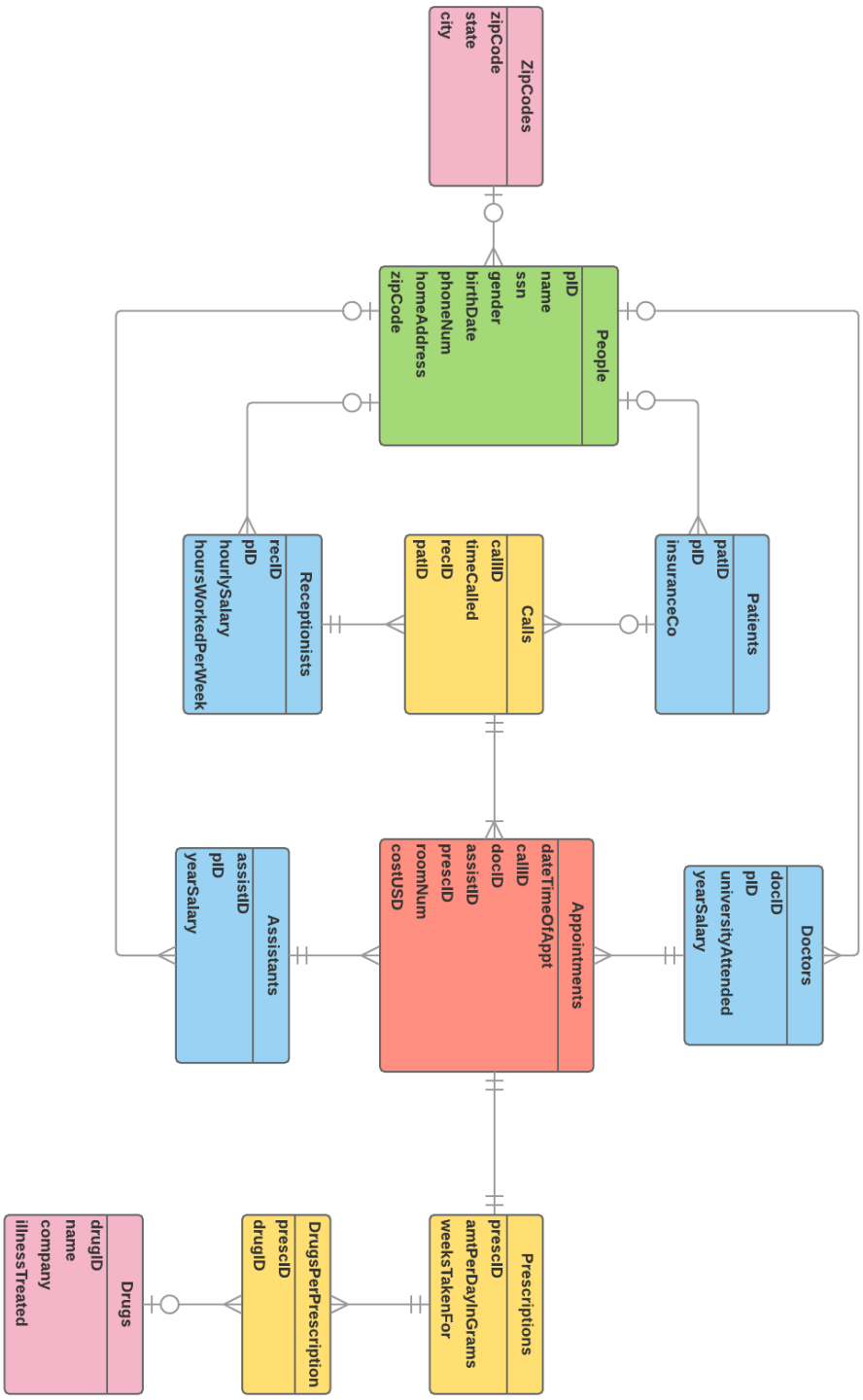
Overview

The healthcare industry is a very important industry in the United States. Efficiency in healthcare organizations is vital for any institution to perform well. This specific relational database model is designed so that a Physician's Office can operate efficiently and effectively.

Objectives

There are several objectives that should be met while creating this relation database. First off, this database will be in Boyce-Codd Normal Form. This means that the database will be following a full relational model, which functional dependencies are dependent on ONLY the key(s) of the appropriate entities. This database will allow a particular Physician's Office to store all of the necessary data they need for their staff and patients. This will allow the Physician's Office to easily access a variety of information they might need at any given time or place while preforming their work day activities.

Entity Relationship Diagram



Stephen Bohner | April 20, 2016

Tables

ZipCodes

Purpose

The purpose of the ZipCodes table is to have zip codes be stored by their valid state and city. This will remove redundancy between states and cities for each person in the People table.

Create Statement

```
CREATE TABLE ZipCodes (
    zipCode int NOT NULL,
    state text NOT NULL,
    city text NOT NULL,
  PRIMARY KEY(zipCode)
);
```

Functional Dependencies

zipCode —> state, city

Sample Data

	zipcode integer	state text	city text
1	11501	NY	Mineola
2	11596	NY	Williston Park
3	15027	PA	Conway
4	12601	NY	Poughkeepsie
5	7712	NJ	Asbury Park
6	1057	MA	Monson
7	10007	NY	New York

People

Purpose

The purpose of the People table is to store all general information that any employees or patients might have. This reduces redundancy between many other tables which is good practice for the relational model.

Create Statement

```
CREATE TABLE People (
    pID int NOT NULL,
    name text NOT NULL,
    ssn int NOT NULL,
    gender text NOT NULL CHECK (gender IN ('MALE', 'FEMALE')),
    birthDate DATE NOT NULL,
    phoneNum text NOT NULL,
    homeAddress text NOT NULL,
    zipCode int NOT NULL REFERENCES ZipCodes(zipCode),
  PRIMARY KEY(pID)
);
```

Functional Dependencies

pID —> name, ssn, gender, birthDate, phoneNum, homeAddress, zipCode

Sample Data

	pid integer	name text	ssn integer	gender text	birthdate date	phonenum text	homeaddress text	zipcode integer
1	0	John Searen	44825487	MALE	1994-02-24	5165484554	5 Durkin Drive	10007
2	1	Billy Currington	487596455	MALE	1992-05-14	84545628788	52 Old Ridge Road	11501
3	2	Hannah Pullip	2140331400	FEMALE	1982-10-31	6314582647	174 Oak Street	1057
4	3	Evan Brooks	478447168	MALE	1972-06-25	5475893120	235 Houston Avenue	10007
5	4	Kirsten Sullivan	487651234	FEMALE	1995-04-02	8451023458	6 Foch Street	10007
6	5	Stephen Bohner	488756325	MALE	1996-12-06	3014896558	2446 Lords Way	12601
7	6	Christian Menk	856417854	MALE	1986-12-22	6945778166	4440 Jefferson Court	11596
8	7	Chris Iacobellis	1875889456	MALE	1996-07-30	5425897555	24 Birch Road	7712
9	8	Reagan Fowler	657123468	FEMALE	1990-02-04	6314517410	4538 James Street	11501
10	9	George Daniels	587106485	MALE	1993-04-02	6873495458	6246 Creek Road	11501
11	10	Linda Owens	487164589	FEMALE	1997-01-03	4781002544	3090 Park Avenue	10007
12	11	David Emory	316022315	MALE	1992-01-16	1249764264	1201 Sunset Drive	15027
13	12	Danielle Codd	122687901	FEMALE	1992-08-08	5481089979	9024 College Street	7712
14	13	Bob Fracker	312049874	MALE	1999-02-15	4421655477	554 Country Club Road	10007
15	14	Katie Gonzalez	265483120	FEMALE	1984-09-18	5478471300	568 Hillcrest Drive	15027
16	15	Ester Tatum	1785493200	FEMALE	1964-06-11	8451548755	757 Cleveland Street	7712
17	16	Devon Elliot	224014787	MALE	1973-08-10	4584783256	4228 Cottage Street	11596
18	17	Alfred Wayne	589789944	MALE	1984-07-07	4579014486	2805 Mechanic Street	15027
19	18	Thorton Earl	102645877	MALE	1983-11-19	2714587767	73 Route 9	11501

Patients

Purpose

The Patients table references the People table and also has a field for the insurance company of each patient.

Create Statement

```
CREATE TABLE Patients (
    patID int NOT NULL,
    pID int NOT NULL REFERENCES People(pID),
    insuranceCo text,
  PRIMARY KEY(patID)
);
```

Functional Dependencies

patID —> pID, insuranceCo

Sample Data

	patid integer	pid integer	insuranceco text
1	0	2	Vista Health
2	1	4	United Care
3	2	6	Aetna
4	3	8	GHI
5	4	9	Vista Health
6	5	13	Embassy Health
7	6	14	Humana

Receptionists

Purpose

The Receptionists table references the People table and also has fields for the amount of hours worked, as well as their hourly salary.

Create Statement

```
CREATE TABLE Receptionists (
    recID int NOT NULL,
    pID int NOT NULL REFERENCES People(pID),
    hourlySalary int NOT NULL,
    hoursWorkedPerWeek int NOT NULL,
  PRIMARY KEY(recID)
);
```

Functional Dependencies

recID —> pID, hourlySalary, hoursWorkedPerWeek

Sample Data

	recid integer	pid integer	hourlysalary numeric(12,2)	hoursworkedperweek integer
1	0	7	10.00	25
2	1	10	12.00	30
3	2	12	11.00	40
4	3	15	11.00	40

Doctors

Purpose

The Doctors table references the People table and also contains information that is unique to a doctor, such as the university attended and their yearly salary.

Create Statement

```
CREATE TABLE Doctors (
    docID int NOT NULL,
    pID int NOT NULL REFERENCES People(pID),
    universityAttended text NOT NULL,
    yearSalary numeric(12,2) NOT NULL,
  PRIMARY KEY(docID)
);
```

Functional Dependencies

docID —> pID, universityAttended, yearSalary

Sample Data

	docid integer	pid integer	universityattended text	yearsalary numeric(12,2)
1	0	1	University of Buffalo	140000.00
2	1	3	University of Massachussetts Amherst	135000.00
3	2	5	University of Maryland	160000.00

Assistants

Purpose

The Assistants table references the People table and has a unique field of their yearly salary. This table is separate from the Doctors table because an assistant is in a different category than a doctor.

Create Statement

```
CREATE TABLE Assistants (
    assistID int NOT NULL,
    pID int NOT NULL REFERENCES People(pID),
    yearSalary numeric(12,2) NOT NULL,
  PRIMARY KEY(assistID)
);
```

Functional Dependencies

assistID —> pID, yearSalary

Sample Data

	assistid integer	pid integer	yearsalary numeric(12,2)
1	0	0	90000.00
2	1	11	95000.00
3	2	16	87500.00
4	3	17	100000.00
5	4	18	985200.00

Calls

Purpose

The purpose of the Calls table is for Patients to schedule appointments. This table references both the Patients table and the Receptionists table. A patient will place a call to a receptionists and schedule an appointment. The Calls table will have the time of the call along with the Patient's ID and Receptionist's ID.

Create Statement

```
CREATE TABLE Calls (
    callID int NOT NULL,
    timeCalled time NOT NULL,
    recID int NOT NULL REFERENCES Receptionists(recID),
    patID int NOT NULL REFERENCES Patients(patID),
  PRIMARY KEY(callID)
);
```

Functional Dependencies

callID —> timeCalled, recID, patID

Sample Data

	callid integer	timecalled time without time zone	recid integer	patid integer
1	0	16:00:00	0	0
2	1	14:47:00	3	2
3	2	10:28:00	3	1
4	3	16:10:00	2	4
5	4	09:17:00	2	3
6	5	11:55:00	1	6
7	6	15:15:00	0	5

Drugs

Purpose

The purpose of this table is to store all the different types of drugs that doctors could issue in their prescriptions. It lists the name, company, and the illness the drug treats.

Create Statement

```
CREATE TABLE Drugs (
    drugID int NOT NULL,
    name text NOT NULL,
    company text NOT NULL,
    illnessTreated text NOT NULL,
  PRIMARY KEY(drugID)
);
```

Functional Dependencies

drugID —> name, company, illnessTreated

Sample Data

	drugid integer	name text	company text	illnesstreated text
1	0	Adderall	Shire	ADHD
2	1	Tramadol	Shire	Pain Killer
3	2	Prozac	Eli Lilly	Depression
4	3	Melatonin	Natural Grocery Company	Insomnia
5	4	Reductil	Abbott	High Cholestrol

DrugsPerPrescription

Purpose

This table separates the many-to-many relationship between drugs and prescriptions, since there can be many drugs in a single prescription.

Create Statement

```
CREATE TABLE DrugsPerPrescription (
    prescID int NOT NULL REFERENCES Prescriptions(prescID),
    drugID int NOT NULL REFERENCES Drugs(drugID),
  PRIMARY KEY(prescID, drugID)
);
```

Functional Dependencies

No Functional Dependencies

Sample Data

	prescid integer	drugid integer
1	0	1
2	1	3
3	2	4
4	3	0
5	4	2

Prescriptions

Purpose

The Prescriptions table is creates so that for each appointment a prescription can be issued. This prescription has an amount for the patient to take, along with how long they need to take their prescription. There can be many drugs in one prescription.

Create Statement

```
CREATE TABLE Prescriptions (
    prescID int NOT NULL,
    amtPerDayInGrams int NOT NULL,
    weeksTakenFor int NOT NULL,
  PRIMARY KEY(prescID)
);
```

Functional Dependencies

prescID —> amtPerDayInGrams, weeksTakenFor

Sample Data

	prescid integer	amtperdayingrams integer	weekstakenfor integer
1	0	1	5
2	1	2	8
3	2	1	3
4	3	3	16
5	4	2	10

Appointments

Purpose

This table is created so that the Physician's Office can keep track of all of the appointments. An appointment consists of a call ID, a doctor, an assistant, a prescription, a room number, and a cost. This table is the weakest entity in the database.

Create Statement

```
CREATE TABLE Appointments (
    dateTimeOfAppt timestamp NOT NULL,
    callID int NOT NULL REFERENCES Calls(callID),
    docID int NOT NULL REFERENCES Doctors(docID),
    assistID int NOT NULL REFERENCES Assistants(assistID),
    prescID int NOT NULL REFERENCES Prescriptions(prescID),
    roomNum int NOT NULL,
    costUSD numeric(12,2) NOT NULL,
  PRIMARY KEY(dateTimeOfAppt)
);
```

Functional Dependencies

dateTimeOfAppt —> callID, docID, assistID, prescID, roomNum, costUSD

Sample Data

	datetimeofappt timestamp without time zone	callid integer	docid integer	assistid integer	prescid integer	roomnum integer	costusd numeric(12,2)
1	2016-04-20 12:15:00	0	1	4	2	125	20.00
2	2016-03-25 01:37:28	4	0	1	0	140	25.00
3	2016-02-14 08:14:58	2	2	1	4	241	45.50
4	2016-02-27 05:15:00	1	1	2	3	210	200.00
5	2016-03-26 04:15:15	3	1	3	1	110	20.00
6	2016-04-01 11:48:13	6	2	0	1	113	30.00

Views

AppointmentCost

Purpose

This view is created so that you can see how much each patient has payed for each appointment. It returns the Patients ID, name, and the cost of the appointment.

Create Statement

```
CREATE VIEW AppointmentCost
 AS SELECT pat.patID, p.name, a.costUSD
 FROM People p
 INNER JOIN Patients pat ON pat.pID = p.pID
 INNER JOIN Calls c ON c.patID = pat.patID
 INNER JOIN Appointments a ON a.callID = c.callID;
```

Sample Data

	patid integer	name text	costusd numeric(12,2)
1	0	Hannah Pullip	20.00
2	3	Reagan Fowler	25.00
3	1	Kirsten Sullivan	45.50
4	2	Christian Menk	200.00
5	4	George Daniels	20.00
6	5	Bob Fracker	30.00

DoctorAppointments

Purpose

The purpose of this view is to see all the appointments that each doctor has scheduled. This view will contain the Doctor's ID, name, and the date of all of their scheduled appointments.

Create Statement

```
CREATE VIEW DoctorAppointments
 AS SELECT d.docID, p.name, a.dateTimeOfAppt
 FROM People p
 INNER JOIN Doctors d ON d.pID = p.pID
 INNER JOIN Appointments a ON a.docID = d.docID;
```

Sample Data

	docid integer	name text	datetimeofappt timestamp without time zone
1	1	Evan Brooks	2016-04-20 12:15:00
2	0	Billy Currington	2016-03-25 01:37:28
3	2	Stephen Bohner	2016-02-14 08:14:58
4	1	Evan Brooks	2016-02-27 05:15:00
5	1	Evan Brooks	2016-03-26 04:15:15
6	2	Stephen Bohner	2016-04-01 11:48:13

PrescriptionDrug

Purpose

This view will select the prescription ID and the drug that is used for that prescription. This view will be useful when trying to find the drug needed for a patient with a specific prescription.

Create Statement

```
CREATE VIEW PrescriptionDrug
 AS SELECT p.prescID, d.name
 FROM Drugs d
 INNER JOIN DrugsPerPrescription dpp ON dpp.drugID = d.drugID
 INNER JOIN Prescriptions p ON p.prescID = dpp.prescID;
```

Sample Data

	prescid integer	name text
1	0	Tramadol
2	1	Melatonin
3	2	Reductil
4	3	Adderall
5	4	Prozac

Reports

PrescDrugNames

Gives the ID of the prescription and the name of the drug(s) for that prescription.

Query

```
SELECT p.prescID, d.name
FROM Drugs d
INNER JOIN DrugsPerPrescription dpp ON dpp.drugID = d.drugID
INNER JOIN Prescriptions p ON p.prescID = dpp.prescID;
```

Output

	prescid integer	name text
1	0	Tramadol
2	1	Melatonin
3	2	Reductil
4	3	Adderall
5	4	Prozac

AvgDoctorSalary

Returns the average yearly salary doctors make.

Query

```
SELECT AVG(yearSalary) AS AvgDoctorSalary
FROM Doctors;
```

Output

	avgdoctorsalary numeric
1	145000.000000000000

AvgAssistantsSalary

Returns the average yearly salary assistants make.

Query

```
SELECT AVG(yearSalary) AS AvgAssistantsSalary
FROM Assistants;
```

Output

	avgassistantssalary numeric
1	271540.000000000000

AvgCostForAppointment

Returns the average cost of an appointment.

Query

```
SELECT AVG(costUSD) AS AvgCostForAppointment
FROM Appointments
```

Output

	avgcostforappointment numeric
1	56.7500000000000000

Stored Procedures

If this function is called it will check if there is any negative integer entries.

```
CREATE FUNCTION negNumber()
 RETURNS trigger AS $negNumber$
  BEGIN
        IF NEW.costUSD < 0 THEN
         RAISE EXCEPTION 'cannot have a negative price';
        END IF;
  END;
  $negNumber$ LANGUAGE plpgsql;
```

Trigger

This trigger will call the negNumber function when something is inserted or updated on the Appointments table.

```
CREATE TRIGGER checkNum BEFORE UPDATE OR INSERT
  ON Appointments
  FOR EACH ROW EXECUTE PROCEDURE negNumber();
```

This is the error it will throw.

```
ERROR:  cannot have a negative price
********** Error **********
```

Security

Database Administrator

```
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO dbAdmin;
```

Patients

```
GRANT INSERT ON ZipCodes TO patientUser;
GRANT SELECT, INSERT ON People TO patientUser;
GRANT INSERT ON Patients TO patientUser;
```

Receptionists

```
GRANT INSERT ON ZipCodes TO receptionistUser;
GRANT SELECT, INSERT ON People TO receptionistUser;
GRANT SELECT, INSERT, UPDATE ON Receptionists TO receptionistUser;
GRANT SELECT, INSERT, UPDATE, DELETE ON Calls TO receptionistUser;
GRANT SELECT, INSERT, UPDATE, DELETE ON Appointments TO receptionistUser;
```

Medical Staff

```
GRANT INSERT ON ZipCodes TO medicalStaff;
GRANT SELECT, INSERT ON People TO medicalStaff;
GRANT SELECT, INSERT, UPDATE ON Doctors TO medicalStaff;
Grant SELECT, INSERT, UPDATE ON Assistants TO medicalStaff;
GRANT SELECT, INSERT, UPDATE ON Prescriptions TO medicalStaff;
GRANT SELECT, INSERT, UPDATE ON Appointments TO medicalStaff;
```

Implementation Notes

The following should be followed so that there will be no undesirable occurrences:

- Any patient/staff that will be stored in the database MUST be inserted in the People table first before any other tables. Person must also provide a valid Zip Code.
- If you wish to be specified to a worker or patient table you can insert data into those fields as well. A patient must schedule an appointment through a call ID which contains Patient and Receptionist ID numbers.
- To create an appointment you must have the call ID, date of appointment, doctor ID, assistant ID, and the room number. Once the appointment takes place, complete the entry by entering in the prescription ID and the cost of the visit.

Known Problems

- You cannot create an appointment until all other fields are met. For example, only until after the appointment is completed can you enter the prescription ID and the cost.
- Reports and views can be modified to check a specific point in time (ex: day, month, year).
- If a patient does not call then the receptionists must still enter a call ID in order to set up an appointment.
- Leading zeros are not recognized in the zip code.

Future Enhancements

- A wide variety of drugs should be available in order for accurate prescriptions to be created.
- More reports for many different scenarios in the future.
- Allow appointments to be created without having a prescription or cost.
- Create a more secure privilege system (more specific grants).
- More stored procedures and triggers.