Final Project Document

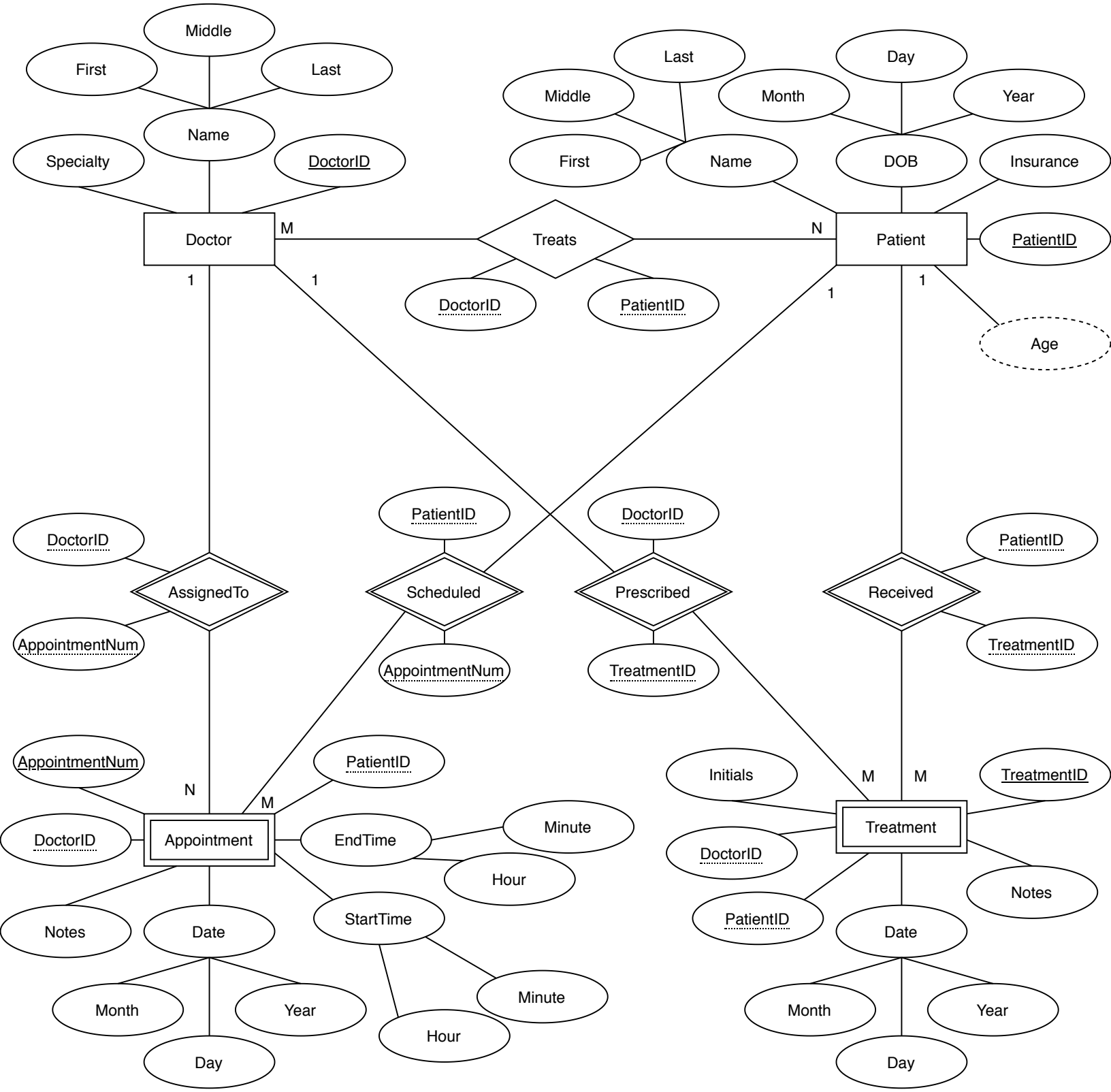
John Cornwell

1 Synopsis

The database that I designed is a medical record system. Its purpose is to store information related to patients and the services provided to them by doctors. Users of the system are able to access the history of patient treatments. This can include patient history from before the patient was treated by the hospital using this system. Users of this system also have the option to set up appointments for patients, cancel future appointments, and modify patient information. Some other data that the medical record system provides is the total number of treatments a doctor has provided, and the history of appointments the doctor has conducted. It is possible for the user to calculate the billing hours for a doctor if they are only interested in the amount of time a doctor is conducting appointments. This record system should allow medical staff to review information on a patient in order to allow specialists to be informed when a patient is transferred to them. This system is not intended to store extended documentation on patient interactions or family history. Instead, it is a record of a hospital's care of a patient. However, users can still see notes that are provided for treatments and appointments. In the event of a lawsuit against the hospital, this record can be provided as proof of care for the patient.

2 Database

See next page.


Middle
First
Last
Name
Specialty
DoctorID
Doctor
Last
Middle
Month
Day
Year
First
Name
DOB
Insurance
Patient
PatientID
Age
M
N
1
1
1
1
Treats
DoctorID
PatientID
DoctorID
AssignedTo
AppointmentNum
PatientID
Scheduled
AppointmentNum
DoctorID
Prescribed
TreatmentID
PatientID
Received
TreatmentID
AppointmentNum
DoctorID
Appointment
N
M
PatientID
EndTime
Minute
Hour
StartTime
Minute
Hour
Notes
Date
Month
Year
Day
Initials
TreatmentID
Treatment
M
M
DoctorID
PatientID
Notes
Date
Month
Year
Day

This database includes nine tables:

1. Patient
   The patient table is responsible for storing data that identifies and describes a patient. This includes the patient's name, date of birth, and insurance provider. From this information, a user of this application should be able to calculate the patient's age. A unique patient ID is also assigned in order to distinguish this patient from others with the same name. The patientID will be used as a foreign key in the appointment and treatment tables because they are both dependent entities. I decided to store an insurance provider because this is common information that most hospitals require. I decided to leave out family history, because the purpose of this database is to provide data on doctor/patient interactions, not on detailed medical care. Because of this, only a name and DOB are necessary.

2. Doctor
   The doctor table is responsible for storing data that describes a doctor. This includes the doctors name, specialty, and unique ID. This table is intended to be used to set up appointments with patients. This is because an appointment is dependent on both a patient and a doctor to be participants. Similarly, we need doctors in order to prescribe and provide treatments. A doctorID was necessary because doctors could have the same name. Besides storing a name, I also chose to store a specialty. This information is useful to give an idea of what treatments this doctor might be performing. This could be used as part of a query when connecting doctors to treatments.

3. Treats
   This table describes the relationship between doctors and patients. It contains pairs of doctorIDs and PatientIDs. Using this table, we can create a list of doctors, and see which patients are treated by them. The relationship from doctor to patient is many to many because a doctor will see many patients, but a patient might consult with many specialists.

4. Appointment
   This table stores information on an appointment that is scheduled by a patient. The Appointment entity is dependent on the Doctor and Patient entity. For this reason, both the doctorID and patientID foreign keys are stored. Appointment also has its own unique key called AppointmentNum. Other information that is stored includes the Appointment date, start time, and end time. As discussed earlier, the times are useful for bookkeeping and billing purposes. Finally, the Appointment also stored noted that are intended to record the complaint of the patient, any immediate actions taken, and other notes related to the visit. Each appointment should take place in one day (no overnight appointments), so we can query the database for the longest appointment.

5. AssignedTo
   This table describes the relationship between doctors and appointments. This table stores pairs of DoctorIDs and AppointmentNums. This will allow us to create a table of all doctors and the appointments they are assigned to. The relationship from doctor to appointment is one to many because a doctor will attend multiple appointments in a day, but there should only be one medical professional during these visits (excluding nurses).

6. Scheduled
   This table describes the relationship between patients and appointments. It stores pairs of PatientIDs and AppointmentNums. We can use these pairs to see which patients scheduled which appointments. The relationship from patient to appointment is one to many because each appointment made will be for one patient, but a patient might schedule multiple appointments during the course of their lifetime.

7. Treatment
   This table stores information on treatments provided by doctors. This can include prescriptions provided or services given such as x-rays or splints. This table allows us to compile a medical history for all of the patients on file. The Treatment is dependent on there being a doctor to provide care, and a patient to be cared for. For this reason, the DoctorID and PatientID are included as foreign keys. Initials of the doctor are stored to act as a signature in the case that Treatment describes a prescription. The date of the treatment is also provided. This can be useful to see how many treatments were given by a doctor in a date range. It can also be used to order tuples. Finally, Notes on the specifics of the treatment are included to describe if Treatment refers to a prescription or medical care. I have no use for this attribute in a query.

8. Prescribed
   This table describes the relationship between doctors and appointments. It stores pairs of DoctorIds and TreatmentID. From this table, we can see which doctors provided which treatments. The relationship from Doctor to Treatment is one to many because a doctor can provide treatment to many patients, but only one medical professional provides care to a patient at a time.

9. Received
   This table describes the relationship between patients and treatments. It stores pairs of PatientIDs and TreatmentIDs. From this information, we can see which treatments were provided to which patients. The relationship form Patient to Treatment is one to many because a patient might need multiple medications or medical procedures, but treatment is only given to one patient at a time.

3 Functionality

This application

- Stores patient information including name, date of birth, and an insurance provider.
- Allows a user to to look up patient treatment history.
- Allows a user to look up information related to an individual doctor, patient, appointment, or treatment.
- Allows a user to schedule and delete appointments.
- Updates patient information such as the name and insurance provider.

The table that I have add and delete functionality on is Appointment. The table that I have update functionality on is Patient.

Three advanced queries that I used:

1. Display the total number of appointments (past, present, and future) that each patient has scheduled. This satisfies the aggregate function requirement.
2. Display the doctors with the third to fifth most treatments performed. This satisfies the offset requirement.
3. Display the doctors with above average billing hours. Billing hours are calculated by time spent in appointments. This is calculated from the scheduled appointment duration. This satisfies the subquery requirement.

4 Stakeholders

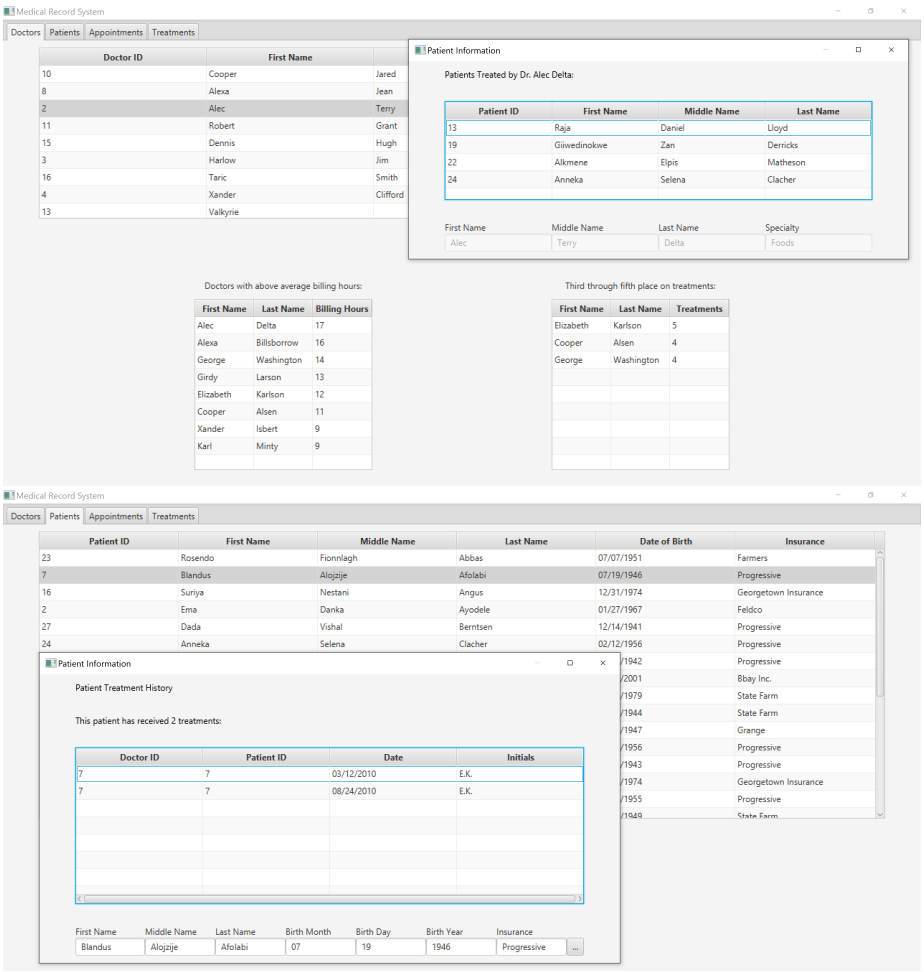
I anticipated three different types of users of this application:

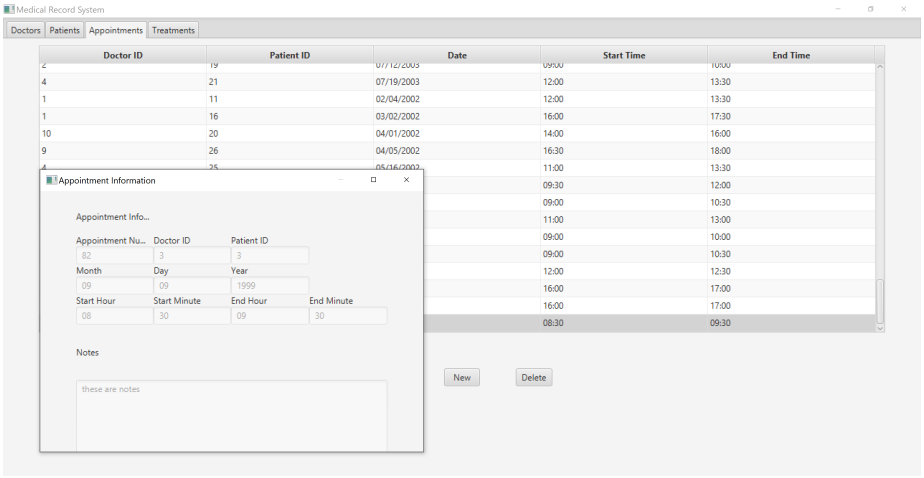
1. Nurses that conduct part of the appointment/treatment. They will primarily modify the notes section of the Appointment tuples.
2. Secretaries will use this system to schedule appointments. If a patient cancels an appointment, they will also be able to delete these tuples.
3. Review boards might use this application for assessing the performance of a doctor. They will be the biggest users of the relational aspect of this database. In the case of a malpractice lawsuit, the review board would need to access treatments provided by a doctor.

5 Technological Requirements

I made a Java desktop application that connects to a database using SQLite. For the user interface, I used JavaFX. I chose to use Java for my application because I have a lot of experience in Java. I had no experience using JavaFX. I used SQLite because it is the only RDBMS I am familiar with. I did not use a code sharing application.

6 Screenshots





7 Advanced Queries

Query 1:

```
SELECT count(*)
FROM Patient JOIN Received
JOIN Treatment
ON Patient.PatientID = Received.PatientID
AND Received.TreatmentID = Treatment.TreatmentId
WHERE Patient.PatientID = ?
```

This query returns the number of treatments that were prescribed to a patient. This satisfies the first advanced query requirement for an aggregate function.

Query 2:

```
SELECT Doctor.DoctorID, Doctor.FirstName, Doctor.MiddleName, Doctor.LastName, Doctor.Specialty, count(*) AS numTreatments
FROM Doctor JOIN Prescribed
JOIN Treatment
ON Doctor.DoctorID = Prescribed.DoctorID
AND Prescribed.TreatmentID = Treatment.TreatmentId
GROUP BY Doctor.DoctorID, Doctor.FirstName, Doctor.LastName, Doctor.Specialty
ORDER BY numTreatments Desc, LastName Asc
LIMIT 3
OFFSET 2
```

This query returns the doctors with the third through fifth most prescribed treatments. If there is a tie, the doctors are ordered alphabetically by last name. This satisfies the second advanced query requirement for offset.

Query 3:

```
SELECT Doctor.DoctorID, Doctor.FirstName, Doctor.MiddleName, Doctor.LastName, Doctor.Specialty, (sum((Appointment.EndHour * 60)
+ Appointment.EndMinute) - sum( (Appointment.StartHour * 60) + Appointment.StartMinute)) / 60 AS BillingHours
FROM Doctor JOIN AssignedTo
JOIN Appointment ON Doctor.DoctorID = AssignedTo.DoctorID
AND AssignedTo.AppointmentNum = Appointment.AppointmentNum
GROUP BY Doctor.DoctorID, Doctor.FirstName, Doctor.MiddleName, Doctor.LastName, Doctor.Specialty
HAVING sum((Appointment.EndHour * 60) + Appointment.EndMinute) - sum((Appointment.StartHour * 60) + Appointment.StartMinute) >
    (SELECT sum(AptMins.mins) / (SELECT count(*) FROM Doctor)
     FROM (SELECT sum((Appointment.EndHour * 60) + Appointment.EndMinute) - sum((Appointment.StartHour * 60) +
     Appointment.StartMinute) AS mins
           FROM Doctor JOIN AssignedTo
           JOIN Appointment
           ON Doctor.DoctorID = AssignedTo.DoctorID
           AND AssignedTo.AppointmentNum = Appointment.AppointmentNum
           GROUP BY Doctor.DoctorID) AS AptMins)
ORDER BY BillingHours Desc
```

This query calculates the average billing hours for doctors and then returns the

doctors that have more than the average billing hours. The average billing hours are calculated by adding up all of the appointment hours, and then dividing by the total number of doctors in the system. This satisfies the third advanced query requirement for a subquery.