**Doctor**

**(PK -> DoctorID)**

First Name

Last Name

Specialty/Practise

Start Date

**Patient**

**(PK -> PatientID)**

First Name

Last Name

Password

**Patient Records**

**(FK -> PatientID)**

name

address

phone number

Health Card (assume all patients from Ontario)

social insurance number

(number of visits) -> can be determined from another table

default doctor

Current Status 🡪 indication of current health (e.g. healthy, diagnosis in progress, prescription currently

being taken, receiving treatment, scheduled for surgery, transferred to another hospital...)

**Doctor/Patient mapping**

**(FK 🡪 PatientID, DoctorID)**

**Patient Visitations**

**(PK -> VisitationID, DoctorID, PatientID)**

**(FK -> ProcedureID)**

Date/Time

Length

Procedures \*\*\*(can be multiple)

Diagnosis

Prescriptions (\*\*multiple?)

Scheduling of treatment

Comments

**Appointments**

**(PK 🡪 PatientID, DoctorID)**

**(FK 🡪 VisitationID, StaffWhoBookedID, BookingState)**

Date/Time

Staff member who scheduled appointment\*\*\*

**?Procedures**

**(PK 🡪 ProcedureID)**

List of possible procedures that can be taken

**BookingStates**

**(PK 🡪 BookingStateID)**

TYPES: BOOKED, CANCELLED, COMPLETE

**Security Access Levels**

**(PK 🡪 SecurityLevel)**

TYPES: ADMIN, DOCTOR, STAFF, PATIENT, LEGAL, FINANCIAL

**Patient Access**

**1. What are the Entity-Relationship (E-R) diagrams for the application? What was the initial**

**design of the database schemas (based on the E-R diagrams)? What constraints (i.e. security,**

**flexibility, ease of use, etc.) were considered? What were the superkeys, functional**

**dependencies identified? What decomposition techniques were employed (BCNF, 3NF, etc.). It**

**would be difficult to arrive at some good schema design without being aware of these points.**

The majority of security constraints are going to be left for the application level to handle. Adding this functionality will add a lot of unnecessary complexity to the database.

Many constraints were added to make sure that the referential integrity of the database was maintained. Examples of this include the “NOT NULL” requirement on the majority of primary/foreign key fields to make sure the mappings between the tables stay intact. Some of the key fields do not require this non-null constraint, such as VisitationID contained in the Appointment table since you can have an appointment scheduled without having the visitation results entered yet.

Other constraints include the unique requirements of certain fields in the database. For example, the PatientID and DoctorID must be kept unique so you can easily identify which patient and doctor is who without risk of mixing up data between other existing doctors and patients. Finally, there are data type constraints that make sure the field has appropriate values, for example, VARCHAR’s are used for data that requires either letters, numbers, and/or symbols and the type has an appropriate limitation to the length of characters allowed as well.

**2. What is the final database design - what are the schema definitions and tables in your design.**

**For this part to minimize your work you can provide the schema definitions using the SQL**

**create table clause. Here, you identify all attributes, the domain for each attribute, the primary**

**key, any foreign keys and any constraints on the values for any of the attributes. Justify any**

**foreign keys.**

CREATE TABLE Doctors (

DoctorID INT NOT NULL AUTO\_INCREMENT PRIMARY KEY,

FirstName VARCHAR(100),

LastName VARCHAR(100),  
Specialty VARCHAR(100)

);

CREATE TABLE Patients (

PatientID INT NOT NULL AUTO\_INCREMENT,  
FirstName VARCHAR(100),

LastName VARCHAR(100),

PRIMARY KEY (PatientID)  
 );

CREATE TABLE PatientRecords (  
 RecordID INT NOT NULL AUTO\_INCREMENT,

PatientID INT NOT NULL,

FirstName VARCHAR(100),

LastName VARCHAR(100),

Address VARCHAR(200),

PhoneNumber VARCHAR(20),

HealthCardNumber VARCHAR(25),

SocialInsuranceNumber VARCHAR(50),

DefaultDoctor\*\*,

CurrentStatus\*\*,

PRIMARY KEY (RecordID),

FOREIGN KEY (PatientID) REFERENCES Patients(PatientID)  
 );

// Foreign key is used to match each patient record to the related patient

\*\*\*CREATE TABLE PatientDoctorMapping (

ID INT NOT NULL AUTO\_INCREMENT,

PatientID INT NOT NULL,

DoctorID INT NOT NULL,

PRIMARY KEY (ID),

FOREIGN KEY (DoctorID) REFERENCES Doctors(DoctorID),

FOREIGN KEY (PatientID) REFERENCES Patients(PatientID)  
);

// Each foreign key is to identify the doctor/patient that is being paired together. This allows for the doctor to be assigned to a patient.

CREATE TABLE PatientVisitations (

VisitationID INT NOT NULL AUTO\_INCREMENT,

AppointmentID INT NOT NULL,

VisitDate DATETIME,

LengthOfVisitInMinutes INT,

Diagnosis VARCHAR(MAX),

Procedures VARCHAR(MAX),

Prescriptions VARCHAR(MAX),

Comments VARCHAR(MAX),

PRIMARY KEY (VisitationID),

FOREIGN KEY (AppointmentID) REFERENCES Appointments(AppointmentID)  
);

// Each patient visitation will be referred to by an appointment. Therefore, there must be a foreign key from this table to the Appointments table to match each appointment to the corresponding visitation.

CREATE TABLE Appointments (

AppointmentID INT NOT NULL,

VistationID INT NOT NULL,

StaffID INT NOT NULL,

Date DATETIME NOT NULL,

PRIMARY KEY (AppointmentID),

FOREIGN KEY (AppointmentID) REFERENCES Visitations(VisitationID)

FOREIGN KEY (StaffID) REFERENCES Staff(StaffID)

);

//As mentioned on the visitation table, you must reference the corresponding visitation record with each appointment record.

//Also, we have a foreign key to the Staff table in order to track which staff member created or most recently updated the appointment.

CREATE TABLE Staff (

StaffID INT NOT NULL,

PrivilegeID INT NOT NULL,

FirstName VARCHAR(100),

LastName VARCHAR(100),

JobTitle VARCHAR(100),

PRIMARY KEY (StaffID),

FOREIGN KEY (PrivilegeID) REFERENCES AccessPrivilegeLevels(PrivilegeID)

);

CREATE TABLE AccessPrivilege (

PrivilegeID INT NOT NULL,

Name VARCHAR(100),

PRIMARY KEY (PrivilegeID)

);

**3. A brief overview of the types of SQL queries that you are prepared to handle. Obviously you do**

**not want to provide the user with complete flexibility to specify any type of query.**

Our application will likely have a separate user interface for each of the departments using the application. Therefore, there will likely be a legal, financial, doctor, staff, and patient interface since each user will have different information available to them.

Our database will have predefined queries for different data sets that we be required. The application will handle which user class can access these queries. Below is a list of queries that will need to be created (not an exhaustive list). The queries have been split apart based on department, however, some of the queries will overlap between departments any will not be repeated in each section.

Legal:

* Select all of the visitation information for a particular patient.

Doctor

* Select all the patients that a doctor has assigned to him/her which will include optional filtering capabilities such as name, patient #, etc.
* Selecting all of a single patient’s records which would include the history for the patient.
* Selecting a patient’s records from a series of optional filters including diagnosis, patient name, date, prescriptions given, etc.
* Insert a patient visitation into the system which would map to a specific appointment. This includes information such as length of visit, diagnosis, prescriptions, “freeform comments”, etc.

Staff

* Insert new appointments or update existing appointments for patients.
* Query for a given doctor’s schedule of appointments to see his availability.
* Delete existing appointments for patients.
* Insert new patients into the system and insert new information for a patient (will not overwrite previous patient record data), which can include assigning a doctor to this patient.

Financial Department

* Select the number visits by each patient assigned to a doctor in a given timeframe.
* Select the total number of procedures performed grouped by doctor and type of procedure.

Patients

* Select history of appointments and the results of each visitation.
* Update personal information (insert new record – to maintain history).

**4. Describe how you are planning to implement the Audit Trail (History of changes).**

The primary audit trail that will be required will be to show the history of a patient’s records. The patient may change address, phone number, or other information pertaining to that person. In order to track this information, we will never delete entries from the table PatientRecords which tracks the patient record data. Instead, we will just add new entries into the table and leave the old entries. When we want to pull current patient information, we will just use the most recent patient record information – the one most recently entered into the table.

Using the design, we are able to maintain all the patient’s records throughout their time at the hospital. In our application, if we want to display this information, we can pull all the records for a particular patient and compare each of the results with the next consecutive record to see what changes were made in each iteration of the records. Note that we will have to be sure to order the patient records based on the order they were entered into the table which can easily be done by sorting by the RecordID which is an auto-incrementing number in the table – the higher the number, the more recent it is.

By comparing each result, we can establish a full history of the patient and see every change that was made over time. Note that since we have not changed the actual PatientID, we are sure that we are still referencing the same patient.

**5. Describe how you will handle access rights.**

Handling access rights is somewhat tricky for this database system. There are very specific access rights that are allowed for each user, for example, the staff of the legal department, a doctor, and a patient will all have very different access rights. Furthermore, since patient information is very confidential, we must be careful to ensure privacy is maintained.

In our project, we are going to have the majority of access rights handled on the application side instead of the database. In other words, the application itself will have security built into it to track which user is currently logged in and what access they should have. The application will only display fields that the particular user should have access to for simplicity and clarity.

The database, however, will need to track the login information for each user in the system which is a crucial part of tracking which user is logged in and what they’ll have access to. Furthermore, there will be a table in the database that has predefined user levels of access based on the user’s department which include, for example, doctor, legal, patient, and more. This will give a baseline of what the user will be able to access.

While we will have a baseline for what access a user should have given their user access level, but we also need to take into account other requirements such as doctor’s only being able to access their own patients (unless provided access by another doctor), staff members not being able to translate a patient ID to their actual names, and patients not being able to see a doctor’s “freeform” comments about a patient. This will all have to be integrated into the application to ensure proper access privileges.

There will also be an administrator access level which will have full unrestricted access to the entire system. This will allow for a super-user that can make whatever changes are required including creating/deleting user accounts for the application and determining access privileges of other users.

This is the basic outline of how access rights will be handled. However, further details will be mapped out when we reach the phase where the application is being designed and implemented. The database level design should be sufficient for our needs, but will make changes to the schema design if there is anything that has been overlooked.

**6. What development environment will you be using to implement the project? We supply only**

**Java and MySQL. If you plan to use some other environment (e.g. Microsoft J++, ...) or some**

**other SQL server, then how are you planning to demo your final project? Short paragraph.**

We have not made a final decision as to what technology we will be using for our application. In all likelihood, we will stick with the supplied technologies Java and MySQL. This can easily be demoed using on any system if we install our database on that local system.

There is currently discussion about using C# Winforms with MySQL in order to implement the application. This will be fully supported on any system with the basic .NET runtime libraries installed, which should not be problematic since it is a standard for all systems. We would still be using the same database system which make it compatible with any system that has the necessary MySQL software installed. We will bring our own computer if these technologies are not available on the standard school computer system.

In either case, we will need to have a database with dummy data entered in prior to the demo time to allow for demonstration of the application without tediously entering a full set of data.