Database Schema for Patient Scheduling System

A database schema models the architecture of data and its relationships (e.g., among tables). It will allow team members to clearly communicate, understand constraints in our design and how data is aggregated, and improve database usability. This document is a work in progress and will evolve throughout our project's development life cycle.

Below we list tables in a row-and-column layout with each database object's name, datum, and data type (e.g., integer, floating-point, variable characters, etc.) with notations for keys and other indications for relationships to other tables. These attributes are important for development to ensure consistent formatting and naming of database objects and their datum. After the row-and-column representation of each table are a set of SQL statements that will initialize that table and, if appropriate, allow for the insertion of records for development and testing. At the end of this document will be a visual data model.

The relational database for our patient scheduling system is `hospital` and predates it. The SQL statement below would create a new hospital database and access it:

```
CREATE DATABASE hospital;
USE hospital;
```

During the development phase a team member may want a separate instance for their own use. The database team suggests using the nomenclature `hospital_yourname` for your instance where `hospital_jdoe` would be J. Doe's development version of the hospital database.

Person

Person				
id	firstName	lastName	dateOfBirth	Add columns
1	Art	Pepper	2	Chief of Diagnostic Medicine
2	Merideth	Grey	7	Chief of General Surgery

Person				
3	Kris	Yang	1	Chief Medical Officer
4	Douglas	Howser	3	Chief Medical Officer
5	Michelle	Quinn	6	Chief of Pharmacology
6	Mathias	Shephard		Surgeon

Here are the corresponding SQL statements for removing an existing instance and instantiating a new one:

```
CREATE TABLE IF NOT EXISTS Person (
 personID INT AUTO_INCREMENT NOT NULL,
 firstName VARCHAR(50) NOT NULL,
 lastName VARCHAR(50) NOT NULL,
 dateOfBirth VARCHAR(50) NOT NULL DEFAULT ‘*****’,
 emailAddress VARCHAR(50) NOT NULL DEFAULT ‘*****’,
 phoneNumber VARCHAR(15),
 gender VARCHAR(2),
 PRIMARY KEY( personID )
);
```

To add a record you can use the following INSERT statement:

INSERT INTO `Person` (`personID`, `firstName`, `lastName`, `dateOfBirth`, `emailAddress`, `phoneNumber`, `gender`) VALUES (NULL, 'Allan', 'Manangan', '*****', 'user@localhost', NULL, NULL);

Doctor

The `Doctor` table contains records from the hospital's hiring system and also predates the patient scheduling system. Here is the table in row-and-column view with rows of sample data:

Doctors				
id	firstName	lastName	departmentID	position
10	Gerry	House	2	Chief of Diagnostic Medicine
11	Merideth	Grey	7	Chief of General Surgery
12	Kris	Yang	1	Chief Medical Officer
13	Douglas	Howser	3	Chief Medical Officer
14	Michelle	Quinn	6	Chief of Pharmacology
15	Mathias	Shephard		Surgeon

Here are the corresponding SQL statements for removing an existing instance and instantiating a new one:

```
CREATE TABLE IF NOT EXISTS Doctor (
 personID INT UNSIGNED NOT NULL,
 CONSTRAINT fkPerson
 FOREIGN KEY( personID )
 REFERENCES Person( personID )
  ON DELETE CASCADE
  ON UPDATE CASCADE,
 licenseNumber VARCHAR(50) NOT NULL DEFAULT '*****',
 specialty VARCHAR(50)
) ENGINE = INNODB;
```

To add a record you can use the following INSERT statement:

```
INSERT INTO `Doctor` (`personID`, `licenseNumber`, `specialty`) VALUES ('2', 'A 12345',
'Optometry');
```

The `Doctor` and `Person` tables are read within the same query to get a doctor's full information:

```
CREATE OR REPLACE VIEW DoctorView AS SELECT
      Person.*,
      Doctor.licenseNumber,
      Doctor.specialty
FROM
      Person
INNER JOIN Doctor ON Person.personID = Doctor.personID
ORDER BY
      Person.personID ASC
```

The database API will query the `DoctorView` view instead of the `Doctor` table to get full information. However it will abstract away the `DoctorView` so it will seem as if the full information were actually available in the `Doctor` table.

Department
The `Department` table also predates the patient scheduling system. It contains names of the hospital's departments. A department does not need to have a named head (see

Department		
id	name	head_doctor_id
1	Adult and Family Medicine	3
2	Emergency	1
3	Intensive Care Unit (ICU)	4
4	Laboratory	
5	Optometry	
6	Pharmacy	5
7	Surgery	2

Here are the corresponding SQL statements for creating an instance if one does not yet exist:

```
CREATE TABLE IF NOT EXISTS Department (
  `departmentID` int unsigned NOT NULL AUTO_INCREMENT,
  `headDoctorID` int unsigned NOT NULL,
  `name` varchar(100) COLLATE utf8mb4_general_ci NOT NULL,
  `phone` varchar(15) COLLATE utf8mb4_general_ci DEFAULT NULL,
  `receptionLocation` varchar(15) COLLATE utf8mb4_general_ci DEFAULT NULL,
  PRIMARY KEY (`departmentID`),
  KEY `fkDepartmentDoctor` (`headDoctorID`),
  CONSTRAINT `fkDepartmentDoctor`
  FOREIGN KEY (`headDoctorID`) REFERENCES `Doctor` (`personID`)
  ON DELETE CASCADE
  ON UPDATE CASCADE
)
```

INSERT INTO Department (departmentID, headDoctorID, name, phone, receptionLocation) VALUES ('0', NULL, 'Adult and Family Medicine', '1234567890', 'Building1');

…

Patient
The `patients` table contains records from the hospital’s membership system. A record is created by either a patient or a hospital employee (e.g., clerk or nurse) assisting a patient. Here is the table in row-and-column view:

Patient				
id	first_name	last_name	patients_id	
1	Jeff	Porcacaro	1000	
2	Phil	Jones	1005	
3	Max	Roach	1010	
4	Tonny	Williams	1013	
5	Charlie	Parker	1018	
6	Art	Tatum	1022	

Here are the corresponding SQL statements for removing an existing instance and instantiating a new one:

```
CREATE TABLE IF NOT EXISTS Patient (
 personID INT UNSIGNED NOT NULL,
 CONSTRAINT fkPersonPatient
 FOREIGN KEY( personID )
 REFERENCES Person( personID )
  ON DELETE CASCADE
  ON UPDATE CASCADE,
 insurerAccountNumber INT NOT NULL,
 insurerName VARCHAR(50) NOT NULL DEFAULT *****’,
 medicalRecordNumber INT NOT NULL,
 PRIMARY KEY( personID )
);
```

INSERT INTO Patient (personID, insurerAccountNumber, insurerName, medicalRecordNumber) VALUES ('1', '1', '*', '12345');

The `Doctor` and `Person` tables are read within the same query to get a doctor’s full information:

```
CREATE OR REPLACE VIEW DoctorView AS SELECT
      Person.*,
      Doctor.licenseNumber,
      Doctor.special
FROM
      Person
INNER JOIN Doctor ON Person.personID = Doctor.personID
ORDER BY
      Person.personID ASC
```

The database API will query the `PatientView` view instead of the `Patient` table to get full information. However it will abstract away the `PatientView` so it will seem as if the full information were actually available in the `Patient` table.

Appointment
Here are the corresponding SQL statements for an instance of the `Appointment` table:

```
CREATE TABLE IF NOT EXISTS Appointment (
 appointmentID INT UNSIGNED NOT NULL,
 doctorID INT UNSIGNED NOT NULL,
 patientID INT UNSIGNED NOT NULL,
 PRIMARY KEY( appointmentID ),
 CONSTRAINT fkDoctor
 FOREIGN KEY( doctorID )
 REFERENCES Doctor( personID )
  ON DELETE CASCADE
  ON UPDATE CASCADE,
 CONSTRAINT fkPatient
 FOREIGN KEY( patientID )
 REFERENCES Patient( personID )
  ON DELETE CASCADE
  ON UPDATE CASCADE,
 date VARCHAR(50) NOT NULL DEFAULT `*****`,
 time VARCHAR(50) NOT NULL DEFAULT `*****`,
 location VARCHAR(50) NOT NULL DEFAULT `*****`,
 reason VARCHAR(50) NOT NULL DEFAULT `*****`,
 status VARCHAR(50) NOT NULL DEFAULT `*****`
);
```

For the Appointment table:

```
INSERT INTO `Appointment` (`appointmentID`, `doctorID`, `patientID`, `date`, `time`, `location`, `reason`, `status`) VALUES ('0', '2',
'1', '*****', '*****', '*****', '*****', '*****');
```

AppointmentView statement

```
CREATE OR REPLACE VIEW AppointmentView AS
SELECT
        Appointment.*,
        DoctorView.*,
        PatientView.*
FROM
        Appointment
INNER JOIN DoctorView ON Appointment.doctorID = DoctorView.personID
INNER JOIN PatientView ON Appointment.patientID = PatientView.personID
ORDER BY Appointment.appointmentID ASC;
```

We also provide a data model below. Please note that there are multiple figures that are segmented for readability with breaks that cut across relationship lines.

(insert figure)

Miscellaneous

We had to alter the collation of the views, tables and database from utf8mb4_0900_ci because of a compatibility issue with importing a database backup into a local development server.

```
ALTER TABLE <table_name> CONVERT TO CHARACTER SET utf8mb4
    COLLATE utf8mb4_unicode_ci;
```

It could have been that their version of MySQL did not support that collation.