Database Design and Implementation Project Report

COMP 6126

Andrew Freisthler

7/23/2015

Table of Contents

Introduction 3

Conceptual Model 4

Entity-Relationship Model 4

Text Annotations 5

Implementation Model 6

Relational Schemas 6

Description of Translation 7

Normalization 8

Text Annotations 9

Note on Integrity Checks 9

Database Implementation 11

CREATE TABLE Commands 11

INSERT Commands 11

Trigger Commands 11

Source Code 11

Known Implementation issues/notes 12

Installation and Use Instructions 12

# Introduction

This report accompanies my submission for my COMP 6126 project for Summer Semester, 2015. This project was developed using OSX 10.10.4, Java 6, IntelliJ IDEA 13, and a local MySQL 5.6.22.

# Conceptual Model

## Entity-Relationship Model

A separate image, alf0028\_er\_diagram.pdf is included with this submission so the ER Diagram can be viewed in a larger format if desired.

## 

## Text Annotations

There were many clarifications made on Canvas that lead to the creation of the diagram of above. I am not going to list those here. The annotations that follow are to for integrity checks that go further than what can be understood from the ER diagram itself.

* A room can only be assigned to one open admission at a time
* An inpatient ordered treatment can only treat an admission not discharged
* An outpatient ordered treatment can only treat an outpatient treatment group not ended
* A doctor ordering a treatment on an inpatient basis (to an admitted patient), he must be the primary doctor or be assigned to the patient

# Implementation Model

## Relational Schemas

**Worker** (worker\_id, hire\_date, first\_name, last\_name)

**Volunteer** (to\_worker)

**Non\_Medical\_Employee** (to\_worker)

**Staff** (to\_non\_medical\_employee)

**Administrator** (to\_non\_medical\_employee)

**Medical\_Employee** (to\_worker)

**Nurse** (to\_medical\_employee)

**Technician** (to\_medical\_employee)

**Doctor** (to\_medical\_employee, can\_admit)

**Volunteer\_Service** (volunteer\_service\_id, description)

**Volunteer\_Service\_Assignment** (volunteer\_service\_assignment\_id, to\_volunteer, to\_volunteer\_service, day)

**Non\_Medical\_Area** (non\_medical\_area\_id, description)

**Staff\_Assignment** (to\_staff, to\_non\_medical\_area)

**Patient** (patient\_id, first\_name, last\_name)

**Room** (room\_id, number)

**Diagnosis** (diagnosis\_id, name)

**Admission** (admission\_id, insurance\_policy, emergency\_contact, to\_room, to\_diagnosis, to\_admitting\_administrator, admission\_date, to\_discharging\_administrator, discharge\_date, to\_primary\_doctor, to\_patient)

**Doctor\_Assigned\_Admission** (to\_admission, to\_doctor)

**Treatment** (treatment\_id, name)

**Outpatient\_Ordered\_Treatment\_Group** (outpatient\_ordered\_treatment\_group\_id, to\_diagnosis, start\_date, end\_date, to\_patient)

**Ordered\_Treatment** (ordered\_treatment\_id, to\_treatment, to\_ordering\_doctor, ordered\_date)

**Inpatient\_Ordered\_Treatment**(to\_ordered\_treatment, to\_admission)

**Outpatient\_Ordered\_Treatment**(to\_ordered\_treatment, to\_outpatient\_ordered\_treatment\_group)

**Medical\_Employee\_Administering\_Ordered\_Treatment** (medical\_employee\_administering\_ordered\_treatment\_id, to\_medical\_employee, to\_ordered\_treatment, date)

## Description of Translation

All attributes are lowercase, separated by underscores as required for readability. Above, primary keys are designated by underscores, but types are not shown. They should be intuitively obvious given the attribute naming. The are all bigint, varchar, datetime, or boolean.

The following schema’s are set up in a subclass nature on the diagram and thus all use worker\_id logically as their key: Worker, Volunteer, Non\_Medical\_Employee, Medical\_Employee, Staff, Administrator, Nurse, Technician, Doctor. For the subclasses, the worker\_id serves both a primary key for the subclass table as well as a foreign key to the parent table. This ensures the class relationships are maintained. While that is the logical value, I have a personal preference for FK relationships to be named as to\_tableBeingReferenced so that it is immediately obvious what the link is to, so they are not all actually named that in the schemas.

In order to describe the Volunteer’s assignment to a Volunteer Service, a total of three tables are used. One for the Volunteer themselves, described above as part of the worker class network, one for the various volunteer services, and a third to describe the pairing on a certain day, as it is a many to many relationship. That third table, denoted as Volunteer\_Service\_Assignment, still uses a separate auto incrementing key for reference, but it will require that the combination of the foreign keys to the Volunteer and Volunteer\_Service tables along with the day to be a key, as a Volunteer should only be able to be assigned to a certain service on one day a single time.

To capture the Staff’s assignment to non-medical working areas, three tables were used to capture the many to many relationship, same as with the volunteer service. The Non\_Medical\_Area is given an integer id as a primary key instead of using the text and the Staff\_Assignment table uses the combination of the worker\_id (from the Staff table) and the non\_medical\_area\_id as its primary key as expected.

The Patient table is simply an id and their name. In a real system, additional informational personal information, such as their address, would be captured, but these attributes seem to be sufficient for this project. The patient\_id in this table is not just the primary key for the table, but also the unique identifier as described in the project description. It was clarified during discussion that we could have the system assign this value any time a patient was entered into the system, whether it be an admission or an initial outpatient service, so this implementation makes sense.

Room and Diagnosis are likewise very simple tables. As done elsewhere, and id value is introduced that may not technically be required based on the specification but allows for things in the future such as room renumbering or diagnosis names to be updated without requiring any foreign key rework (my application supports such operations).

Admissions and their related data are captured in two tables. The first, Admission, uniquely captures the physical admission along with 1 to N relationships to Room, Diagnosis, both Administrator relationships, Patient, and Doctor for the primary relationship. With the exception of to\_discharging\_administrator and discharge\_date, which will be NULL until the patient is discharged, all other attributes are required. The Doctor\_Assigned\_Admission table captures the many to many relationships where additional doctors can be assigned to patients by the primary doctor.

Treatments and their associated actions are mapped to six schemas. Treatment is a simple table, holding only an id and a treatment name. Ordered\_Treatment captures the 1 to N relationship with doctor (doctor can order many treatments but a treatment has only one ordering doctor), the date the treatment was ordered, and the treatment to be administered. Inpatient\_Ordered\_Treatment is a subclass of Ordered\_Treatment and is used when the ordered treatment is for an inpatient/admitted patient. Outpatient\_Ordered\_Treatment is a subclass of Ordered\_Treatment and is used when the ordered treatment is for an outpatient patient. In either case, they get the treatment related to a patient and to an entity that can group treatments for either situation.

Medical\_Employee\_Administering\_Ordered\_Treatment captures actual administrations of the ordered treatment . This is considered a many to many and thus held in a separate table under the assumption that some ordered treatments would require multiple administrations. And finally, we have the Outpatient\_Ordered\_Treatment\_Group, which is in place to capture the notion that an outpatient patient can have multiple ordered treatments working toward a single diagnosis. It also captures the start and end of outpatient services for that patient. It should be noted that Treatment classes themselves do not directly reference a patient, but instead reference either an admission or an outpatient\_ordered\_treatment\_group which in turn reference a patient.

## Normalization

There was only one change made during the translation from my ER diagram into Schemas. Originally, the Ordered\_Treatment Schema held all the information that is now split between the three schemas (Ordered\_Treatment, Inpatient\_Ordered\_Treatment, and Outpatient\_Ordered\_Treatment). There were certainly issues both with extraneous attributes as well as normal form.

With that change however, I believe all my tables are now in BCNF. I believe I have the tables simplified down to a point at which if we were to list out the dependencies we’d have nothing but superkeys on the left hand side.

## Text Annotations

Below are a few assumptions/notes I took while implementing the queries for the project:

* For B2 and B3 I had to make a distinction between being admitted to the hospital and receiving inpatient services, which looks like the questions may have thought of differently. Since my system captures temporal data with both admission/discharge (or start/end with an outpatient) along with actual treatments inside of that timeframe, I could have used either. I decided to go with the admin/discharge, which is the larger of the two windows.
* For C.8, a ‘treatment occurrence’ was taken to mean the ordered treatment from a doctor, which could be administered multiple times.
* For D.4, associated diagnosis was taken to mean diagnosis associated with a primary doctor’s admission. For outpatient, there could be multiple doctors.
* For D.6 ‘participated’ was interpreted as having actually administered part of the treatment. Ordering alone would not be sufficient to have participated.

## Note on Integrity Checks

The vast majority of the integrity checks described in the problem description are enforced through the relational schema design. For example, the following are enforced via the subclass layout, foreign keys to those subclasses, etc. (this is not an exhaustive list):

* Only volunteers can provide volunteer services
* Only doctors can be assigned as primary doctors
* Only nurses, technicians, and doctors can administer treatments
* Only doctors can order treatments

However, there were a few integrity checks that were not easy to include as part of the relational schemas themselves. These were the ones that were noted along with the ER diagram and typically were situations when we had foreign key relationships but the foreign keys were only valid if some other attribute on the schema being linked to had a certain attribute value at the time the foreign key was created. To handle these situations, I turned to triggers. The following integrity checks are implemented via triggers:

* A patient cannot be admitted if they are currently admitted at the time
* A patient cannot be admitted with a room already in use at the time
* The doctor admitting must have admitting privileges at the time he admits a patient
* A treatment ordered on an inpatient basis must be for a patient currently admitted
* A treatment ordered on an outpatient basis must be a patient currently receiving outpatient services

There was a single integrity check that I was not able to find a way to enforce easily with either my schema implementation or easily with triggers. While I’m sure it is possible with triggers it looked to be above my current skill level. The integrity condition not currently being checked

* Doctor ordering a treatment on inpatient basis must be assigned to or be primary for admitted patient

# Database Implementation

## CREATE TABLE Commands

The create table commands are included in a separate file with this submission, createTables.sql. This can also be viewed on github here:

<https://github.com/afreisthler/comp6126/blob/master/out/artifacts/comp6126_jar/createTables.sql>

## INSERT Commands

The create insert commands are included in a separate file with this submission, initData.sql. This can also be viewed on github here:

<https://github.com/afreisthler/comp6126/blob/master/out/artifacts/comp6126_jar/initData.sql>

## Trigger Commands

The commands to install triggers are included in a separate file with this submission, installTriggers.sql. This can also be viewed on github here:

<https://github.com/afreisthler/comp6126/blob/master/out/artifacts/comp6126_jar/installTriggers.sql>

## Source Code

The jar file submitted has the source bundled with it. If the jar file is unzipped, that java files will all be there and be browsable. Howerver, the entirety of my project is on github, so that is probably the easier way to look at it. Here are the important links:

Project Root:

<https://github.com/afreisthler/comp6126>

Room.java – has A query set

<https://github.com/afreisthler/comp6126/blob/master/src/Room.java>

Patient.java – has B query set

<https://github.com/afreisthler/comp6126/blob/master/src/Patient.java>

DiagAndTreatInfo.java – has C query set

<https://github.com/afreisthler/comp6126/blob/master/src/DiagAndTreatInfo.java>

Employee.java – has D query set

<https://github.com/afreisthler/comp6126/blob/master/src/Employee.java>

## Known Implementation issues/notes

* For simplicity, all queries were done without using prepared statements. I know this is not the best for security but was a short cut taken to allow me to easily abstract out some of my query calling in some situations.
* I did not use transactions at all and know now that I should have in a couple situations. As far as I’m aware, with the way my subclasses are setup, it doesn’t actually cause any real harm, but can lead to some rows being left around at higher levels in the class structure if the creation of a lower class were to fail due to an integrity check during creation because the creation of the parent was not rolled back.
* Deleting was not configured to cascade. So, any attempt to delete a row that is referenced by another object will fail due to FK constraints.
* Java is not my best language, so who knows on style!

## Installation and Use Instructions

The readme file bundled with this submission, README.txt, describes the basics of the installation. It can also be viewed in github, here:

<https://github.com/afreisthler/comp6126/blob/master/out/artifacts/comp6126_jar/README.txt>

Once the database is correctly instantiated and configured via the instructions in the readme, the interface itself is hopefully fairly self-explanatory.

The application itself is a text based, menu driven interface. The main menu presents six options. These are:

1. Query Menu – This selection will allow a user to drill down and execute all queries described in the “required query support” section of the project description. The ordering and numbering in the sub menus should match the problem statement exactly
2. Admin Action Menu – This houses the majority of the data entry options. It presents quite a few options, including List/Add/Edit/Delete/Admit Patients, Admit/Discharge Patients, List/Add/Edit/Delete Doctors, List/Add/Edit/Delete Volunteers, List/Add/Edit/Delete Staff, List/Add/Edit/Delete Admins, List/Add/Edit/Delete Nurses, List/Add/Edit/Delete Techs, and List/Add/Edit/Delete Rooms.
3. Doctor Action Menu – This presents the actions available to doctors, including List/Add/Edit/Delete Treatment Options, List/Add/Edit/Delete Diagnosis Options, Order a treatment, Change diagnosis for a patient, Administer a treatment, Assign another doctor to an admitted patient, and conclude treatment for an outpatient patient.
4. Technician Action Menu – This presents the actions available to technicians, which is currently only administering a treatment.
5. Nurse Action Menu – This presents the actions available nurses, which is currently only administering a treatment.
6. Volunteer Action Menu – This presents the actions available to volunteers. This includes: List/Add/Edit/Delete Volunteer Options and List/Add/Edit/Delete Volunteer Assignments.