ITIS 6120: Applied Databases

AN ELECTRONIC MEDICAL RECORD DATABASE FOR AMBULATORY SURGERY CENTERS

November 24, 2023

Name: David Gary
Student ID: 801325583

1 Introduction

When one imagines the influencing factors of a patient's healthcare, one might think of the patient's lifestyle, their diet, their exercise habits, and so on. The reality, however, is dense with beurocracy, complicated legal barriers, and profit-driven motives. Changing the healthcare system is a daunting task, but there are marginal ways to clean up the processes that are already in place. In this project, we will be focusing on implementing a database that can house the electronic health records (EHRs) of patients in a centralized manner, specifically by piggybacking off of the existing infrastructure of ambulatory surgery centers (ASCs).

1.1 Background and Motivation

Many people assume that all surgeries are performed in hospitals, but this is not the case. Around 43% of all surgeries are performed in freestanding ambulatory surgery centers (ASCs), which are separate from hospitals.[2] These facilities are oftentimes owned by larger healthcare organizations, and they normally belong to a larger group of ASCs, which might have some sort of centralized management. ASCs are a cheaper alternative to hospitals, provide greater flexibility for patients, and are usually covered by Medicare and Medicaid[3].

The interconnectedness that ASCs have with each other and with hospitals provides an excellent opportunity for a centralized electronic health record (EHR) system. Although many would assume that EHR systems should be centered around the patient, many argue that the potential for disorder and confusion is too great[1]. Since doctors within hospitals will often refer patients to ASCs for outpatient procedures, the pipeline of information is already in place.

1.2 Scope

It should be made clear that this project is not meant to be a complete EHR system, nor is it providing any infrastructure to help solve medical mysteries through data analysis. All that this project is meant to do is provide a centralized database for the EHRs of patients that have undergone outpatient procedures at ASCs. This will allow hospitals to see which ASCs provide the procedures they need to refer patients to, and it will allow ASCs to see which hospitals or doctors are referring patients to them. Patients will also be able to see their own EHRs, at least in part.

2 Database Design

This section presents the tables used, provides a brief description of each one's purpose and relationships with other tables, and proves that each table is in BCNF. The tables are meant to reflect real world entities and the relationships between them. Any ambiguous labelling is explained in the table's description. After reviewing the tables, the relationships among them are synthesized into easily digestible diagrams.

2.1 Table 1: `insurance_providers`

```
CREATE TABLE insurance_providers(
    insurance_provider_id INT NOT NULL AUTO_INCREMENT,
    insurance_provider_name VARCHAR(255) NOT NULL,
    PRIMARY KEY (insurance_provider_id)
);
```

This table is to store the list of insurance providers in the EHR system. The provider id is the primary key, and is auto-incremented. This primary key is used as a foreign key in both the `patients` and `operation_records` tables.

To prove that this table is in BCNF, we must show that it is in 3NF, and that every non-trivial functional dependency is a superkey. No column is dependent on any other column, and the primary key is the only determinant of any other column. Therefore, this table is in 3NF. Since the primary key is the only determinant of any other column, and the primary key is auto-incremented, it is a superkey. Therefore, this table is in BCNF.

It should be noted, nearly all tables reviewed after this one have the same properties as this one. As a result, the same BCNF reasoning should be applied. Later, there will be a table with different properties, and the BCNF reasoning will be provided for that table.

2.2 Table 2: `hospitals`

```
CREATE TABLE hospitals(
    hospital_facility_id INT NOT NULL AUTO_INCREMENT,
    hospital_facility_name VARCHAR(255) NOT NULL,
    hospital_street_address VARCHAR(255) NOT NULL,
    hospital_state VARCHAR(2) NOT NULL,
    hospital_city VARCHAR(255) NOT NULL,
    PRIMARY KEY (hospital_facility_id)
);
```

This table is to store the list of hospitals that are affiliated with the ASC system. Typically, employees at a hospital (often a doctor) will refer patients to an ASC for a procedure. The hospital id is the primary key, and is auto-incremented, and is used as a foreign key in the `doctors` and `operation_records` tables.

2.3 Table 3: `ambulatory_surgery_centers`

```
CREATE TABLE ambulatory_surgery_centers(
    asc_facility_id INT NOT NULL AUTO_INCREMENT,
    asc_facility_name VARCHAR(255) NOT NULL,
    asc_street_address VARCHAR(255) NOT NULL,
    asc_state VARCHAR(2) NOT NULL,
    asc_city VARCHAR(255) NOT NULL,
    PRIMARY KEY (asc_facility_id)
);
```

An ambulatory surgery center (ASC) is an outpatient facility where surgical procedures are performed. Their purpose is to provide a faster, cheaper alternative to hospitals for outpatient procedures. Since these are, usually, separate from hospitals, their information must be stored separately. The ASC facility id is the auto-incrementing primary key, and it is used as a foreign key in the `operation_records` table.

2.4 Table 4: `surgical_procedures`

```
CREATE TABLE surgical_procedures(
    surgical_procedure_id INT NOT NULL AUTO_INCREMENT,
    surgical_procedure_name VARCHAR(255) NOT NULL,
    surgical_procedure_description VARCHAR(255) NOT NULL,
    PRIMARY KEY (surgical_procedure_id)
);
```

This table only stores the types of surgical procedures that have been recorded within the ASC system. The name and a short description of the procedure are stored, with a primary key that auto-increments. This primary key is used as a foreign key in the `operation_records` table.

2.5 Table 5: `patients`

```
CREATE TABLE patients(
    patient_id INT NOT NULL AUTO_INCREMENT,
    patient_full_name VARCHAR(255) NOT NULL,
    patient_date_of_birth DATE NOT NULL,
    patient_gender VARCHAR(255) NOT NULL,
    patient_street_address VARCHAR(255) NOT NULL,
    patient_state VARCHAR(2) NOT NULL,
    patient_city VARCHAR(255) NOT NULL,
    patient_email VARCHAR(255) NOT NULL,
    patient_phone_number VARCHAR(255) NOT NULL,
    patient_insurance_provider INT NOT NULL,
    patient_insurance_number VARCHAR(255) NOT NULL,
    PRIMARY KEY (patient_id),
    FOREIGN KEY (patient_insurance_provider)
        REFERENCES insurance_providers(insurance_provider_id)
);
```

This table stores the information of patients that have been recorded within the ASC system. The patient id is the primary key, and it is auto-incremented. This table does use a foreign key to reference the `insurance_providers` table, which is the insurance provider that the patient is using. This foreign key is the only non-trivial functional dependency, and it is a superkey. Therefore, this table is in BCNF.

2.6 Table 6: `doctors`

```
CREATE TABLE doctors(
    doctor_id INT NOT NULL AUTO_INCREMENT,
    doctor_full_name VARCHAR(255) NOT NULL,
    doctor_email VARCHAR(255) NOT NULL,
    doctor_phone_number VARCHAR(255) NOT NULL,
    doctor_hospital_affiliation INT NOT NULL,
    PRIMARY KEY (doctor_id),
    FOREIGN KEY (doctor_hospital_affiliation)
        REFERENCES hospitals(hospital_facility_id)
);
```

This table stores the information of doctors that have been recorded within the ASC system. Similar to the other tables, an auto-incrementing id is used as the primary key. This table does use a foreign key to reference the `hospitals` table, which is the hospital that the doctor is affiliated with. Again, this foreign key is the only non-trivial functional dependency, and it is a superkey. Therefore, this table is in BCNF.

2.7 Table 7: `operation_records`

```
CREATE TABLE operation_records(
    operation_record_id INT NOT NULL AUTO_INCREMENT,
    operation_record_date DATE NOT NULL,
    operation_record_facility_id INT NOT NULL,
    operation_record_patient_id INT NOT NULL,
    operation_record_procedure INT NOT NULL,
    operation_record_insurance_provider INT NOT NULL,
    operation_record_affiliated_hospital INT NOT NULL,
    operation_record_doctor_id INT NOT NULL,
    PRIMARY KEY (operation_record_id),
    FOREIGN KEY (operation_record_facility_id)
        REFERENCES ambulatory_surgery_centers(asc_facility_id),
    FOREIGN KEY (operation_record_patient_id)
        REFERENCES patients(patient_id),
    FOREIGN KEY (operation_record_procedure)
        REFERENCES surgical_procedures(surgical_procedure_id),
    FOREIGN KEY (operation_record_insurance_provider)
        REFERENCES insurance_providers(insurance_provider_id),
    FOREIGN KEY (operation_record_affiliated_hospital)
        REFERENCES hospitals(hospital_facility_id),
    FOREIGN KEY (operation_record_doctor_id)
        REFERENCES doctors(doctor_id)
);
```

The last table is the `operation_records` table, which contains the information of each operation that has been recorded within the ASC system. This table uses a primary key that auto-increments, and it uses foreign keys to reference the other tables. The BCNF reasoning from the previous tables does still apply though, since all of the foreign keys are the only

non-trivial functional dependencies, and they are all superkeys. Therefore, this table is in BCNF.

2.8 Diagrams

After reading the previous section, it might be difficult to track the relationships between each table. To help visualize them instead, an entity relationship diagram (ERD) and a UML diagram have been provided in figures 1 and 2, respectively.

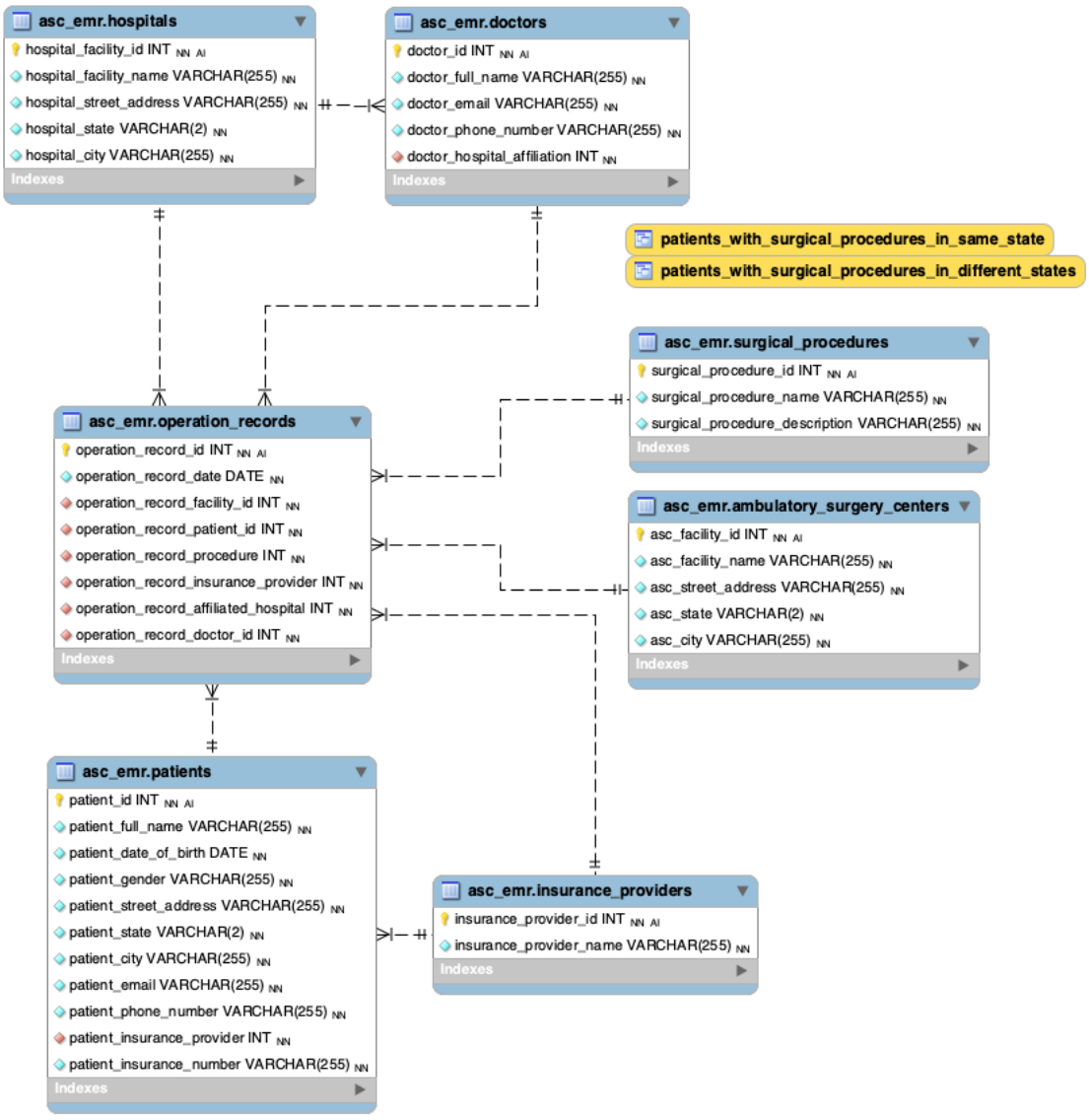

Figure 1: Entity Relationship Diagram

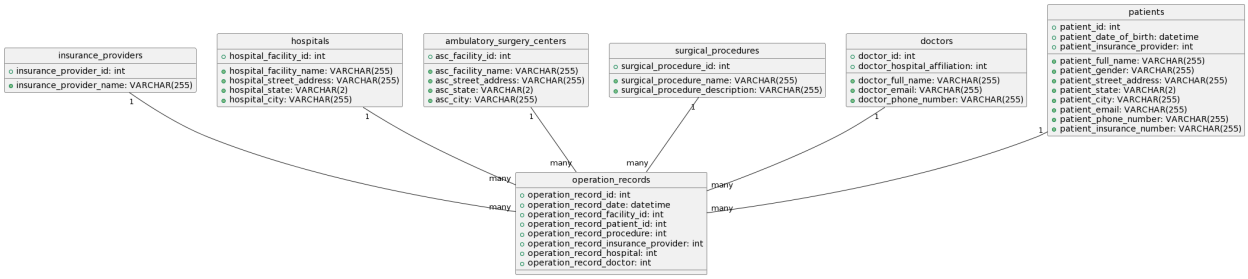

Figure 2: UML Diagram

2.9 Audit Tables

Each table has an audit table associated with it so that changes made to any part of the database can be tracked. The audit tables are named the same as the original table, but with `_audit` appended to the end. For every audit table, there are only four columns: `audit_id`, `source_table_id`, `audit_date`, and `audit_action`. Note that each of those columns have some sort of prefix to indicate their table connection (see schemas below).

It then follows that the audit tables are in BCNF as well, since the only non-trivial functional dependency is the primary key, which is auto-incremented. The remaining columns are provided from the user's input and are not dependent on any other column. Since all of the audit tables follow the same pattern, the proof extends to each of them.

```
CREATE TABLE insurance_providers_audit(
    insurance_provider_audit_id INT NOT NULL AUTO_INCREMENT,
    insurance_provider_id INT NOT NULL,
    insurance_provider_audit_date DATE NOT NULL,
    insurance_provider_audit_action VARCHAR(255) NOT NULL,
    PRIMARY KEY (insurance_provider_audit_id),
    FOREIGN KEY (insurance_provider_id)
        REFERENCES insurance_providers(insurance_provider_id),
    CONSTRAINT insurance_provider_audit_action
        CHECK (insurance_provider_audit_action
        IN ('INSERT', 'UPDATE', 'DELETE'))
);
```

```
CREATE TABLE hospitals_audit(
    hospital_audit_id INT NOT NULL AUTO_INCREMENT,
    hospital_facility_id INT NOT NULL,
    hospital_audit_date DATE NOT NULL,
    hospital_audit_action VARCHAR(255) NOT NULL,
    PRIMARY KEY (hospital_audit_id),
    FOREIGN KEY (hospital_facility_id)
        REFERENCES hospitals(hospital_facility_id),
    CONSTRAINT hospital_audit_action
        CHECK (hospital_audit_action
        IN ('INSERT', 'UPDATE', 'DELETE'))
);
```

```
CREATE TABLE ambulatory_surgery_centers_audit(
    asc_audit_id INT NOT NULL AUTO_INCREMENT,
    asc_facility_id INT NOT NULL,
    asc_audit_date DATE NOT NULL,
    asc_audit_action VARCHAR(255) NOT NULL,
    PRIMARY KEY (asc_audit_id),
    FOREIGN KEY (asc_facility_id)
        REFERENCES ambulatory_surgery_centers(asc_facility_id),
    CONSTRAINT asc_audit_action
        CHECK (asc_audit_action
        IN ('INSERT', 'UPDATE', 'DELETE'))
);
```

```
CREATE TABLE surgical_procedures_audit(
    surgical_procedure_audit_id INT NOT NULL AUTO_INCREMENT,
    surgical_procedure_id INT NOT NULL,
    surgical_procedure_audit_date DATE NOT NULL,
    surgical_procedure_audit_action VARCHAR(255) NOT NULL,
    PRIMARY KEY (surgical_procedure_audit_id),
    FOREIGN KEY (surgical_procedure_id)
        REFERENCES surgical_procedures(surgical_procedure_id),
    CONSTRAINT surgical_procedure_audit_action
        CHECK (surgical_procedure_audit_action
        IN ('INSERT', 'UPDATE', 'DELETE'))
);
```

```
CREATE TABLE patients_audit(
    patient_audit_id INT NOT NULL AUTO_INCREMENT,
    patient_id INT NOT NULL,
    patient_audit_date DATE NOT NULL,
    patient_audit_action VARCHAR(255) NOT NULL,
    PRIMARY KEY (patient_audit_id),
    FOREIGN KEY (patient_id)
        REFERENCES patients(patient_id),
    CONSTRAINT patient_audit_action
        CHECK (patient_audit_action
        IN ('INSERT', 'UPDATE', 'DELETE'))
);
```

```
CREATE TABLE doctors_audit(
    doctor_audit_id INT NOT NULL AUTO_INCREMENT,
    doctor_id INT NOT NULL,
    doctor_audit_date DATE NOT NULL,
    doctor_audit_action VARCHAR(255) NOT NULL,
    PRIMARY KEY (doctor_audit_id),
    FOREIGN KEY (doctor_id)
        REFERENCES doctors(doctor_id),
    CONSTRAINT doctor_audit_action
```

```
        CHECK (doctor_audit_action
        IN ('INSERT', 'UPDATE', 'DELETE'))
);
```

```
CREATE TABLE operation_records_audit(
    operation_record_audit_id INT NOT NULL AUTO_INCREMENT,
    operation_record_id INT NOT NULL,
    operation_record_audit_date DATE NOT NULL,
    operation_record_audit_action VARCHAR(255) NOT NULL,
    PRIMARY KEY (operation_record_audit_id),
    FOREIGN KEY (operation_record_id)
        REFERENCES operation_records(operation_record_id),
    CONSTRAINT operation_record_audit_action
        CHECK (operation_record_audit_action
        IN ('INSERT', 'UPDATE', 'DELETE'))
);
```

2.10 Updated Diagrams with Audit Tables

While no functional dependencies have changed, the total number of tables has. To reflect these changes and display the structure of each audit table, updated UML and entity relationship diagrams have been provided in figures 3 and 4, respectively.

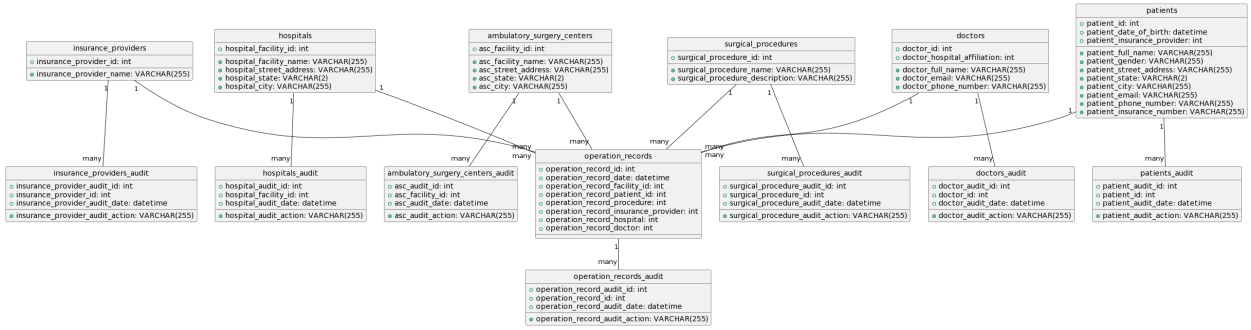

Figure 3: UML Diagram with Audit Tables

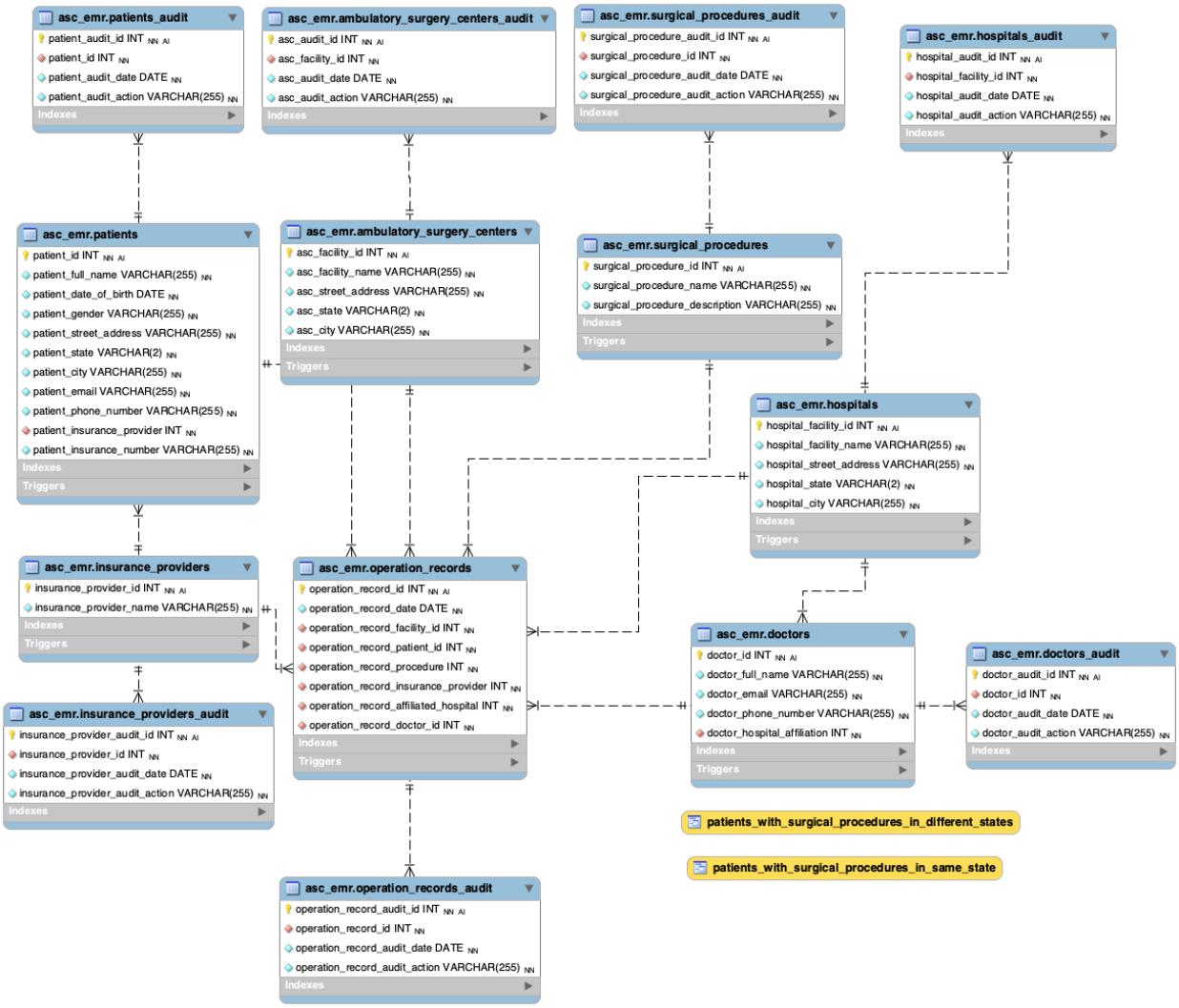

Figure 4: Entity Relationship Diagram with Audit Tables

2.11 Indexes

To help with querying the database, a few indexes were added on the more commonly queried columns. The text interface created for this database relies heavily on backtracking from values like a patient's full name, insurance id number, or a similarly unsorted column. As a result, indexes were added to the `asc_facility_name`, `patient_full_name`, `patient_date_of_birth`, `operation_record_date`, `hospital_name`, and `doctor_full_name` columns. Figures 5, 6, 7, 8, and 9 show the indexes that were added to each table.

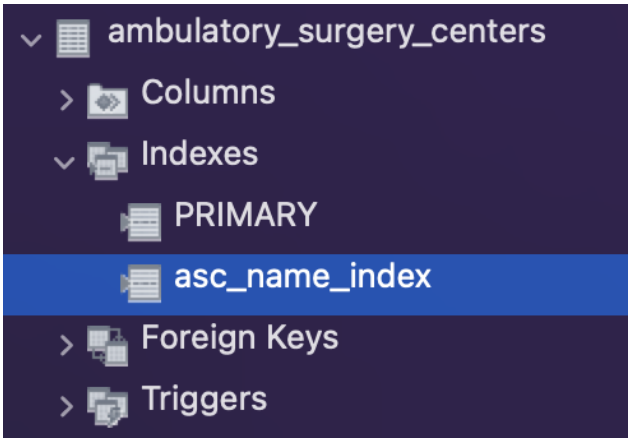

Figure 5: ASC Index

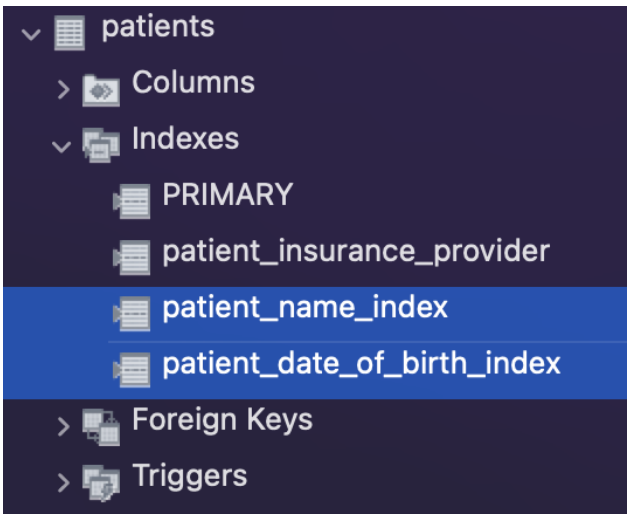

Figure 6: Patients Index

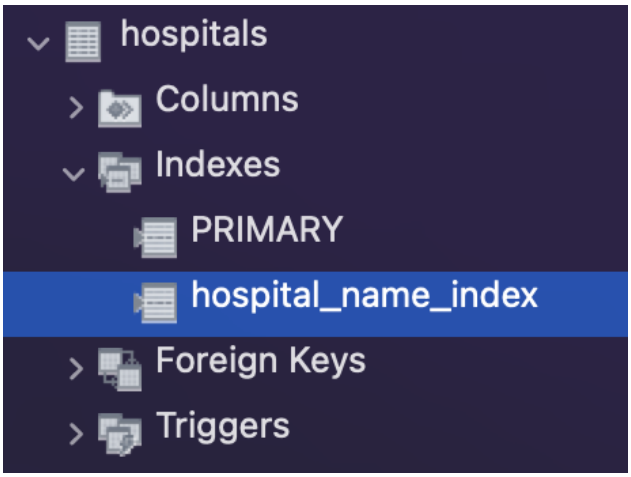

Figure 7: Hospitals Index

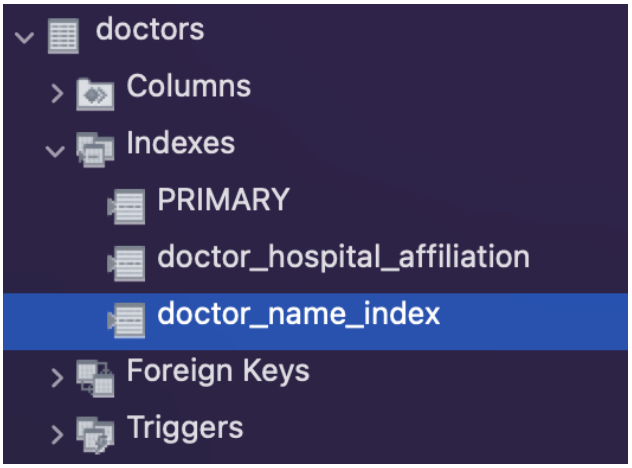

Figure 8: Doctors Index

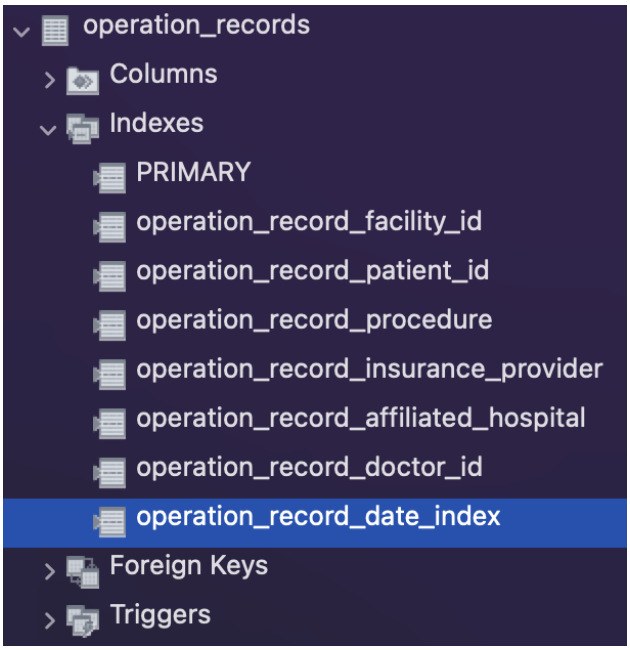

Figure 9: Operation Records Index

2.12 Views

The existing views were kept, as there were no changes to the tables that would require them to be updated. One new view was added to the database, which is the `default_facility_view`. This is the only view used in the actual text interface, and it is, actually, not a functional component for most of the users. Instead, the default user has access to this view, and it is used to display the hospitals and ASCs that are in the system. More of this view in action is shown in the API Overview & Usage section. Figure 10 shows the views in the database from the MySQL Workbench interface, and the list below provides a brief description of each view.

- `patients_with_surgical_procedures_in_different_states` - Shows all patients who have had two or more surgical procedures in different states
- `patients_with_surgical_procedures_in_same_state` - Shows all patients who have had a specific surgical procedure and are in the same state as another patient with the same surgical procedure
- `default_facility_view` - Shows all hospitals and ASCs grouped by city and state (for default user)

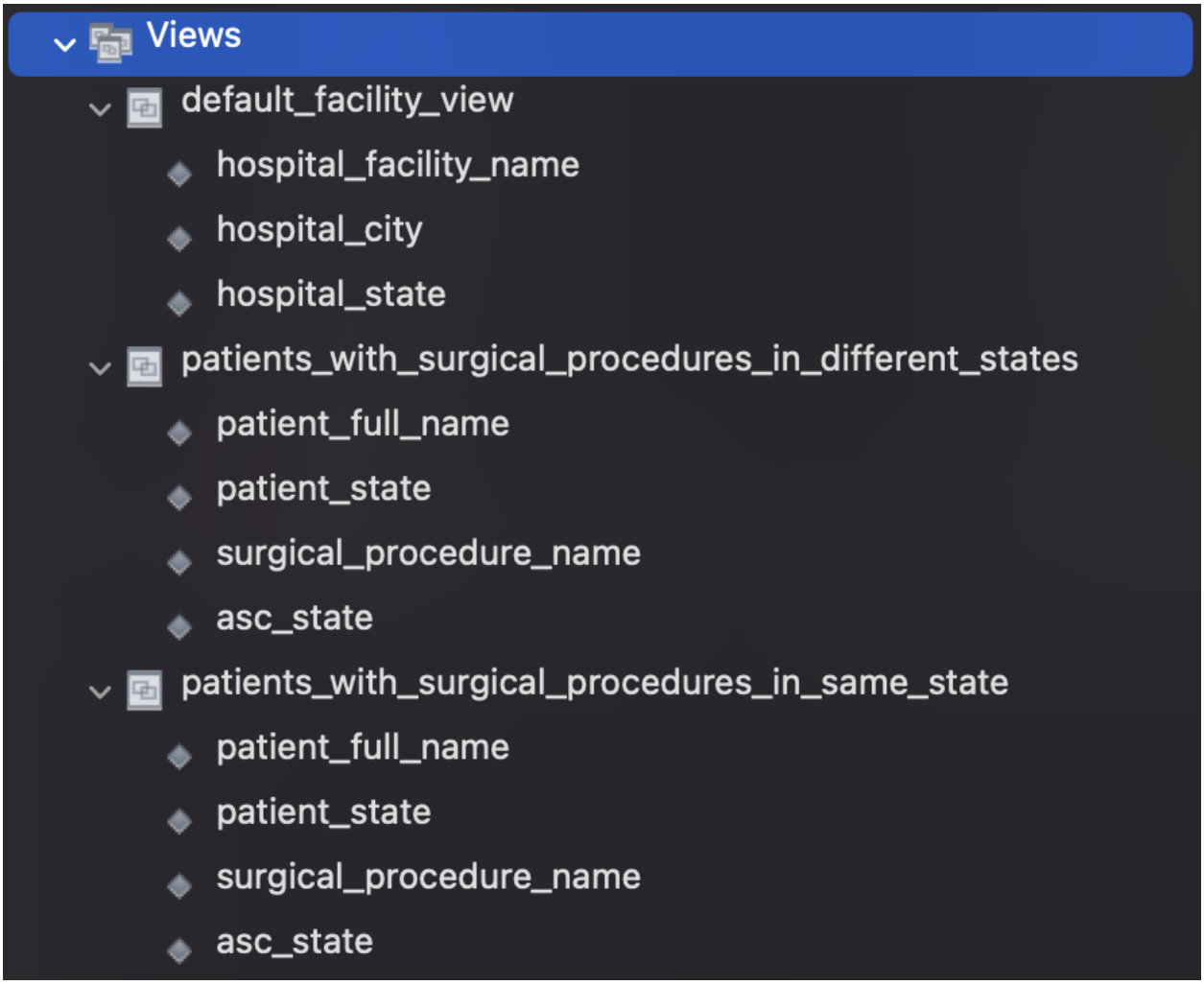

Figure 10: Views

2.13 Triggers

A total of 21 triggers were implemented to monitor the database for changes. Each time either an insert, update, or delete is made to one of the core tables (i.e. not an audit table), a trigger is fired to record the change in the respective audit table. The triggers are named

the same as the audit tables, but with a suffix to indicate the type of change that was made. Figure 11 shows the triggers in the database from the MySQL Workbench interface. There is no need to provide further explanation for each trigger, as they all follow the same pattern and their names directly indicate what they do.

Name	Event	Table	Timing
ambulatory_surgery_centers_audit_insert_trigger	INSERT	ambulatory_s...	AFTER
ambulatory_surgery_centers_audit_update_trigger	UPDATE	ambulatory_s...	AFTER
ambulatory_surgery_centers_audit_delete_trigger	DELETE	ambulatory_s...	AFTER
doctors_audit_insert_trigger	INSERT	doctors	AFTER
doctors_audit_update_trigger	UPDATE	doctors	AFTER
doctors_audit_delete_trigger	DELETE	doctors	AFTER
hospitals_audit_insert_trigger	INSERT	hospitals	AFTER
hospitals_audit_update_trigger	UPDATE	hospitals	AFTER
hospitals_audit_delete_trigger	DELETE	hospitals	AFTER
insurance_providers_audit_insert_trigger	INSERT	insurance_pr...	AFTER
insurance_providers_audit_update_trigger	UPDATE	insurance_pr...	AFTER
insurance_providers_audit_delete_trigger	DELETE	insurance_pr...	AFTER
operation_records_audit_insert_trigger	INSERT	operation_rec...	AFTER
operation_records_audit_update_trigger	UPDATE	operation_rec...	AFTER
operation_records_audit_delete_trigger	DELETE	operation_rec...	AFTER
patients_audit_insert_trigger	INSERT	patients	AFTER
patients_audit_update_trigger	UPDATE	patients	AFTER
patients_audit_delete_trigger	DELETE	patients	AFTER
surgical_procedures_audit_insert_trigger	INSERT	surgical_proc...	AFTER
surgical_procedures_audit_update_trigger	UPDATE	surgical_proc...	AFTER
surgical_procedures_audit_delete_trigger	DELETE	surgical_proc...	AFTER

Figure 11: Triggers

3 Roles & Users

As a way to show the relevant user types for this system, the following roles and users were created: `nurse`, `doctor`, `asc_admin`, & `sys_admin`. In this section, we will go over the idea behind each role and the privileges that they have.

3.1 Nurse

The `nurse` role can be seen as one of the lower level roles in the system. A nurse is usually who the patient will coordinate with the most, but they are unlikely to make system level changes for any information that is held outside of the `patients` table. The `nurse` role has the following privileges:

- `SELECT` on all tables
- `INSERT, UPDATE, & DELETE` on `patients`
- `EXECUTE` on procedures `get_patient_info, & get_patient_operation_records`
- `ALL PRIVILEGES` on procedure `get_list_of_asc_for_procedure`

Having these permissions enables users within this role to retreive medical records for a patient, update their current personal information, and find the ASCs that are able to perform a given procedure that the patient may need.

3.2 Doctor

The `doctor` role is the next level up from the `nurse` role, but they do not inherit the permissions of the `nurse` role. Doctors are the providers who perform the procedures on the patients, and their role is more closely related to operation records and the specific surgical procedures offered. As such, the `doctor` role has the following privileges:

- `SELECT & INSERT` on `operation_records`
- `SELECT, INSERT, & UPDATE` on `surgical_procedures`
- `EXECUTE` on procedures `log_new_operation_record, update_surgical_procedure_name, update_surgical_procedure_description, & add_new_surgical_procedure`

These permissions allow users within this role to create new operation records for patients, update the surgical procedures that are offered, and add new surgical procedures to the system.

3.3 ASC Admin

An ASC admin is the person who is responsible for various administration tasks over an ASC or group of ASCs. This role is the first of the two roles that have the ability to modify the hospitals, ASCs, and doctors that are in the system. The `asc_admin` role has the following privileges:

- `ALL PRIVILEGES` on `hospitals, ambulatory_surgery_centers, & doctors`
- `EXECUTE` on procedures `view_asc_info_by_name, update_asc_address, update_asc_name`

More permissions could be added to this role to fully flesh out the idea of an ASC admin, but for the purposes of this project, these permissions give the user the ability to view information about an ASC, update the address of an ASC, and update the name of an ASC.

3.4 System Admin

The `sys_admin` role is the highest level role in the system.

- `ALL PRIVILEGES` on all tables, views, & procedures

This role has the ability to do anything in the system, and as such, it is the only role that can create new users and assign them to roles.

3.5 Users

The following users were created to be test users for each of the roles:

- `nurse_jackie` - the `nurse` user with the password `nurse_jackie`
- `default_user` - a user with no role and the password `default_user`
- `doctor_house` - the `doctor` user with the password `doctor_house`
- `asc_admin_stu` - the `asc_admin` user with the password `asc_admin_stu`
- `sys_admin_john` - the `sys_admin` user with the password `sys_admin_john`

4 API Overview & Usage

A number of procedures and views are available to be called via the API. These procedures act as the primary way to interact with the database without having to write any SQL queries. In this section, we will review the procedures that are available, and go over how the text based interface works to call these procedures.

4.1 Stored Procedures

A core component of this project is the use of stored procedures. This section will go over the stored procedures that were created for this project. The following stored procedures were created:

- `add_new_surgical_procedure` - Adds a new surgical procedure to the `surgical_procedures` table
- `get_asc_audit_delete_records` - Returns a list of all ASC audit delete records
- `get_asc_audit_insert_records` - Returns a list of all ASC audit insert records
- `get_asc_audit_update_records` - Returns a list of all ASC audit update records
- `get_default_facility_view` - Returns a View of all hospitals and ASCs (for default user)

- `get_hospitals_audit_delete_records` - Returns a list of all hospital audit delete records
- `get_hospitals_audit_insert_records` - Returns a list of all hospital audit insert records
- `get_hospitals_audit_update_records` - Returns a list of all hospital audit update records
- `get_operation_records_audit_delete_records` - Returns a list of all operation record audit delete records
- `get_operation_records_audit_insert_records` - Returns a list of all operation record audit insert records
- `get_operation_records_audit_update_records` - Returns a list of all operation record audit update records
- `get_patients_audit_delete_records` - Returns a list of all patient audit delete records
- `get_patients_audit_insert_records` - Returns a list of all patient audit insert records
- `get_patients_audit_update_records` - Returns a list of all patient audit update records
- `get_list_of_asc_for_procedure` - Returns a list of ASCs that can perform a given procedure
- `get_patient_info` - Returns a patient's information
- `get_patient_operation_records` - Returns a patient's operation records
- `log_new_operation_record` - Adds a new operation record to the `operation_records` table
- `update_asc_address` - Updates the address of an ASC
- `update_asc_name` - Updates the name of an ASC
- `update_surgical_procedure_description`
- `update_surgical_procedure_name` - Updates the name of a surgical procedure
- `view_asc_info_by_name` - Returns information about an ASC

4.2 Using the API as a Default User

If the text interface is executed without any command line arguments, the user will be prompted to enter a username, password, host, and database name. The information will then be checked against existing users in the config file, and if no match is found, the user will be logged in as a default user. Shown in figure 12 is an example of the default user login:

```
davidgary@Davids-MacBook-Pro-2 interface % ./main
Username: david
Password: gary
Host: localhost:3306
Database: asc_emr
Username not found in config file. Using default_user.
Connected to database as default_user on localhost:3306/asc_emr
Please select an option:
1. View facility information
```

Figure 12: Default user login

The default user can only view the hospitals and ASCs that are in the system, and cannot make any changes to the system. Figure 13 shows an example of the default user viewing the hospitals and ASCs in the system:

```
Result Set 694:
hospital_state                WV
hospital_facility_name        Colgate-Palmolive
hospital_city                 Huntington

Result Set 695:
hospital_state                WV
hospital_facility_name        Alva-Amco Pharmacal Companies, Inc.
hospital_city                 Huntington

Result Set 696:
hospital_city                 Huntington
hospital_state                WV
hospital_facility_name        Family Dollar Services,Inc.

Result Set 697:
hospital_facility_name        Aphena Pharma Solutions - Tennessee, LLC
hospital_city                 Morgantown
hospital_state                WV

Would you like to continue? (y/n)
```

Figure 13: Default user view of hospitals and ASCs

4.3 Using the API as a Nurse

If the text interface is executed with an argument of `nurse`, the user will immediately be logged in as a nurse. And in figure 14, we see the given options for a nurse.

```
davidgary@Davids-MacBook-Pro-2 interface % m nurse
go mod tidy
go build main
./main nurse
Connected to database as nurse_jackie on localhost:3306/asc_emr
Please select an option:
1. View patient information
2. View patient operation records
3. View ASCs where a specific operation was performed
```

Figure 14: Nurse options

The nurse can view a patient's information, view a patient's operation records, and find ASCs that can perform a given procedure. Shown in figure 15, we see an example of a nurse viewing a patient's information:

```
Connected to database as nurse_jackie on localhost:3306/asc_emr
Please select an option:
1. View patient information
2. View patient operation records
3. View ASCs where a specific operation was performed
1
search_name: Devlin Pechan
search_dob: 1985-04-15
search_insurance_number: 16t6uGLqtPi6HsbDYw1x1HVhXHUycsWajz
Rows Affected: 0
Rows Returned: 11

Result Set 1:
patient_id                  493
patient_city                Topeka
patient_street_address      91 Weeping Birch Place
patient_state               KS
patient_email               dpechando@spotify.com
patient_phone_number        (785) 7176395
patient_insurance_provider  7
patient_full_name           Devlin Pechan
patient_date_of_birth       1985-04-15
patient_gender              Male
patient_insurance_number    16t6uGLqtPi6HsbDYw1x1HVhXHUycsWajz

Would you like to continue? (y/n)
```

Figure 15: Nurse viewing a patient's information

And in figure 16, we see an example of a nurse viewing a patient's operation records:

```
1. View patient information
2. View patient operation records
3. View ASCs where a specific operation was performed
2
search_name: Devlin Pechan
search_dob: 1985-04-15
search_insurance_number: 16t6uGLqtPi6HsbDYw1x1HVhXHUycsWajz
Rows Affected: 0
Rows Returned: 3

Result Set 1:
Patient Name                    Devlin Pechan
Surgical Procedure              Catheterization
Operation Date                  2020-11-20

Result Set 2:
Patient Name                    Devlin Pechan
Surgical Procedure              Appendectomy
Operation Date                  2023-04-15

Result Set 3:
Patient Name                    Devlin Pechan
Surgical Procedure              Appendectomy
Operation Date                  2023-04-15

Would you like to continue? (y/n)
```

Figure 16: Nurse viewing a patient's operation records

Finally, figure 17 shows an example of a nurse finding ASCs that can perform a given procedure:

```
Please select an option:
1. View patient information
2. View patient operation records
3. View ASCs where a specific operation was performed
3
search_procedure_name: Injections
Rows Affected: 0
Rows Returned: 4

Result Set 1:
ASC Name                        DUKAL Corporation
Address                         0772 Fallview Court
City                            New York City
State                           NY

Result Set 2:
ASC Name                        NARS Cosmetics
Address                         9 Burrows Pass
City                            Sacramento
State                           CA

Result Set 3:
ASC Name                        Devakam Apothecary Hall, Co, Ltd.
Address                         7 3rd Parkway
City                            Portland
State                           OR
```

Figure 17: Nurse finding ASCs that can perform a given procedure

4.4 Using the API as a Doctor

If the text interface is executed with an argument of `doctor`, the user will immediately be logged in as a doctor. In figure 18, we see the given options for a doctor.

```
davidgary@Davids-MacBook-Pro-2 interface % make doctor
go mod tidy
go build main
./main doctor
Connected to database as doctor_house on localhost:3306/asc_emr
Please select an option:
1. Log an operation record
2. Update surgical procedure name
3. Update surgical procedure description
4. Add a new surgical procedure
```

Figure 18: Doctor login and options

The doctor can add a new operation record, update a surgical procedure's name, and update a surgical procedure's description. Figure 19 shows an example of a doctor adding a new operation record:

```
Connected to database as doctor_house on localhost:3306/asc_emr
Please select an option:
1. Log an operation record
2. Update surgical procedure name
3. Update surgical procedure description
4. Add a new surgical procedure
1
search_doctor_id: 5
search_patient_insurance_number: 16t6uGLqtPi6HsbDYw1x1HVhXHUycsWajz
search_procedure_name: Injections
search_asc_facility_id: 25
search_operation_date: 2000-01-01
Rows Affected: 1
Rows Returned: 0

Would you like to continue? (y/n)
```

Figure 19: Doctor adding a new operation record

And in figures 20 and 21, we see examples of a doctor updating a surgical procedure's name and description, respectively:

```
Would you like to continue? (y/n) y
Please select an option:
1. Log an operation record
2. Update surgical procedure name
3. Update surgical procedure description
4. Add a new surgical procedure
2
search_procedure_name: Injections
updated_procedure_name: Drug Injections
Rows Affected: 0
Rows Returned: 1

Result Set 1:
Output Message                Updated surgical procedure with name Injections to have name Drug Injections

Would you like to continue? (y/n)
```

Figure 20: Doctor updating a surgical procedure's name

```
Would you like to continue? (y/n) y
Please select an option:
1. Log an operation record
2. Update surgical procedure name
3. Update surgical procedure description
4. Add a new surgical procedure
3
search_procedure_name: Drug Injections
updated_procedure_description: injections of a drug, performed by a professional
Rows Affected: 0
Rows Returned: 1

Result Set 1:
Output Message                Updated surgical procedure with name Drug Injections to
have description injections of a drug, performed by a professional
```

Figure 21: Doctor updating a surgical procedure's description

The final implementation option for Doctor users allows them to add a new surgical procedure to the system. Figure 22 shows an example of a doctor adding a new surgical procedure to the system:

```
Would you like to continue? (y/n) y
Please select an option:
1. Log an operation record
2. Update surgical procedure name
3. Update surgical procedure description
4. Add a new surgical procedure
4
new_procedure_name: Lobotomy
new_procedure_description: removal of a lobe in the brain
Rows Affected: 1
Rows Returned: 0
```

Figure 22: Doctor adding a new surgical procedure

4.5 Using the API as an ASC Admin

If the text interface is executed with an argument of `asc_admin`, the user will immediately be logged in as an ASC admin. These users can search for an ASC's full information by name and update the name and address of an ASC. Figure 23 shows the search functionality for an ASC's full information:

```
davidgary@Davids-MacBook-Pro-2 interface % make asc_admin
go mod tidy
go build main
./main asc_admin
Connected to database as asc_admin_stu on localhost:3306/asc_emr
Please select an option:
1. View ASC information from a specific name
2. Update an ASC's name
3. Update an ASC's address
1
search_asc_name: Cardinal Health
Rows Affected: 0
Rows Returned: 5

Result Set 1:
asc_state                       PA
asc_city                        Philadelphia
asc_facility_id                 26
asc_facility_name               Cardinal Health
asc_street_address              6 American Ash Crossing

Result Set 2:
asc_facility_name               Cardinal Health
asc_street_address              75341 Warrior Road
asc_state                       TX
asc_city                        Corpus Christi
```

Figure 23: ASC admin searching for an ASC's full information

And in figures 24 and 25, we see examples of an ASC admin updating an ASC's name and address, respectively:

```
Please select an option:
1. View ASC information from a specific name
2. Update an ASC's name
3. Update an ASC's address
2
search_asc_id: 477
updated_asc_name: CARDINAL HEALTH
Rows Affected: 1
Rows Returned: 0

Would you like to continue? (y/n)
```

Figure 24: ASC admin updating an ASC's name

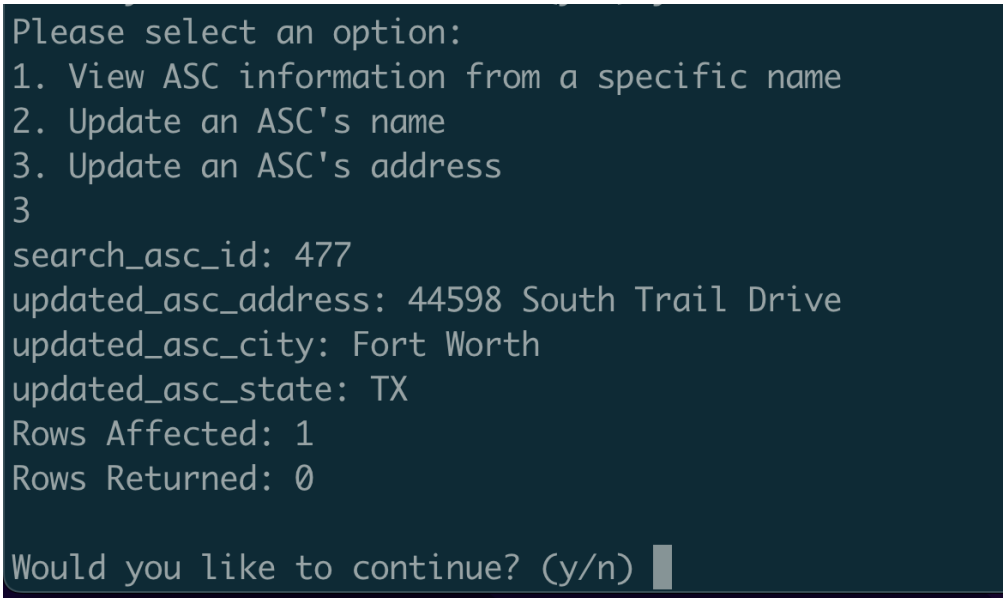

Figure 25: ASC admin updating an ASC's address

4.6 Using the API as a System Admin

If the text interface is executed with an argument of `sys_admin`, the user will immediately be logged in as a system admin. These users are intended to be able to view the audit logs for any table in the system, but at the time of this writing, the text interface only allows for viewing the audit logs for the `ambulatory_surgery_centers` table. Figure 26 shows an example of a system admin viewing the audit insert records for the `ambulatory_surgery_centers` table:

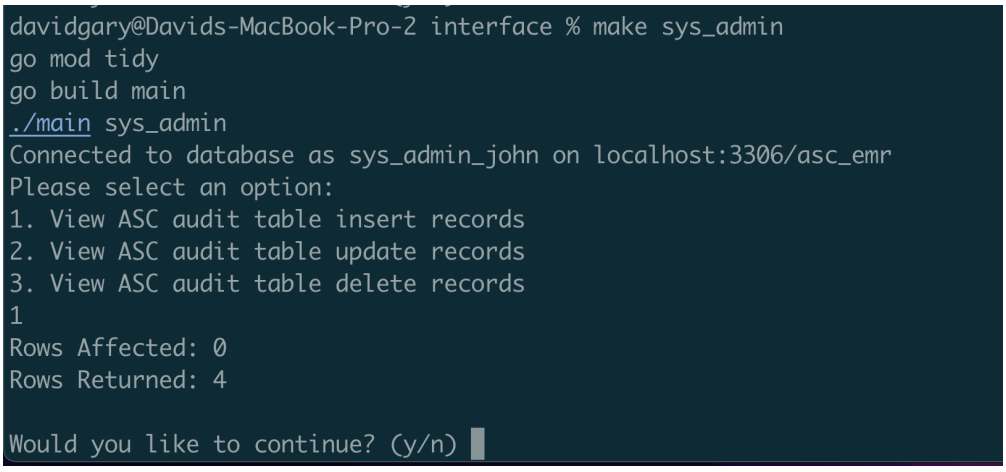

Figure 26: System admin viewing the audit insert records

And in figures 27 and 28, we see examples of a system admin viewing the audit update and delete records for the `ambulatory_surgery_centers` table, respectively:

```
Please select an option:
1. View ASC audit table insert records
2. View ASC audit table update records
3. View ASC audit table delete records
2
Rows Affected: 0
Rows Returned: 4

Result Set 1:
asc_audit_id                    1
asc_facility_id                 22
asc_audit_date                  2023-11-20
asc_audit_action                UPDATE

Result Set 2:
asc_audit_date                  2023-11-20
asc_audit_action                UPDATE
asc_audit_id                    2
asc_facility_id                 22

Result Set 3:
asc_audit_action                UPDATE
asc_audit_id                    3
asc_facility_id                 22
asc_audit_date                  2023-11-20
```

Figure 27: System admin viewing the audit update record

```
Please select an option:
1. View ASC audit table insert records
2. View ASC audit table update records
3. View ASC audit table delete records
3
Rows Affected: 0
Rows Returned: 4

Would you like to continue? (y/n)
```

Figure 28: System admin viewing the audit delete records

5 Tools Used

This section of the report will detail all tools used in the creation of this database.

5.1 MySQL Workbench

This tool was utilized as the primary execution environment for all SQL queries. Additionally, the entity relationship diagram was generated using the 'Reverse Engineer' feature that this tool provides.

5.2 Sqlite3 and Python3.10

As a backup environment for faster testing, execution, and debugging, sqlite3 was used in combination with Python3.10. Simple scripts were written in Python to help execute some SQL queries, rather than having to utilize the (somewhat buggy) command line version of sqlite3. This provided excellent insight and debugging capabilities, and it forced a better understanding of the database's structure.

5.3 Mockaroo

Mockaroo is a tool used to generate mock data for testing purposes, and is available at https://mockaroo.com/. This tool was used to generate the mock data for the database.

5.4 PlantUML

To generate the UML diagram, a simple PlantUML file was written. PlantUML is a tool that generates UML diagrams from code, and is available at https://plantuml.com/.

5.5 Go

The Go programming language was used to create the API. This includes various packages that are common to the Go ecosystem.

References

[1] Ebadollahi, S., Coden, A. R., Tanenblatt, M. A., Chang, S.-F., Syeda-Mahmood, T., and Amir, A. Concept-based electronic health records: Opportunities and challenges. In *Proceedings of the 14th ACM International Conference on Multimedia* (New York, NY, USA, 2006), MM '06, Association for Computing Machinery, p. 997–1006.

[2] for Disease Control, C., and Prevention. Ambulatory surgery factsheet, 2017.

[3] Wiggins, C., Peterson, T., and Moss, C. Ambulatory surgery centers use of health information technology. *Health Policy and Technology 4*, 2 (2015), 100–106.