# CSC 4710 Final Report

Team Members: Nathan Larkin, Jack Gordon, Paul Ofremu Jr.

## Introduction

*Nathan Larkin: contributed a few sentences to introduction*
*Jack Gordon: contributed to initial draft*
*Paul Ofremu Jr.: convert outline into final introduction with additional information and details*

The database is designed to be used in primary care, family medicine facilities, and similar establishments in the health industry—specifically, a medical practice with multiple general practitioners and general family medicine services. Paperwork has been a trouble in the healthcare field that has created inefficiencies for providers and those who collect and record data. With so many patients and data to manage, every person that touches a form can potentially introduce errors and inconsistencies that can be easily overlooked. Tracking and logging day-to-day activities such as appointments can be very tedious on paper. Patients who want access to their medical information may have to wait large amounts of time to receive it due to the large amount of paperwork.

This database aims to provide a system for both staff and patients to access data and perform the actions they need. Patients will be able to access only their personal information, such as family history, past appointments and examinations, and health metrics. The medical staff will be able to input all data regarding examinations and appointments, access data relevant to patients, and provide prescriptions and referrals. Other staff, such as administration, can access employee data and additional relevant information. The care facility will be able to store data in coordination with labs. Essential aspects of the design include the functionality to allow patients and medical staff to get the complete medical history of a particular patient.

Additionally, we aim to provide health metrics such as average, minimum, and maximum of commonly recorded appointment attributes. The database scope is limited to only storing appointments that have occurred. Future appointments are scheduled in a different application and are not stored until the appointment takes place.

## ➔ Requirements Analysis (Section Two)

*Nathan Larkin: Wrote all the requirements, wrote some of the functional requirements*
*Jack Gordon: Wrote the constraints, some functional requirements*
*Paul Ofremu Jr.: Contributed ideas for requirements and functional requirements*

- Describe requirements one by one
    - I want to track the employees at my medical practice. Employees can either be a general practitioner, nurse, lab technicians, administrative, residents, and interns. The data I want to track of employees is their name, ssn, birthday, salary, phone, address, their vaccinations, and their job position.
    - Employees who are administrative workers have reduced permissions in what they are allowed to do. They can only create files to save, create appointments as when the patient checks in, and see enough information to aid in scheduling future appointments for patients.
    - All the staff that have permission to prescribe controlled medication, such as the Physician Assistants, general practitioners, and nurses need to be able to access the current patients medical records for the appointment they are in. They also need to be able to prescribe a specific medication, and I want to track those prescriptions. It's important that I can see the date the prescription was prescribed by an employee at my facility.
    - I want to be able to store data for each patient appointment. I do a set of general measurements, which I always take, but I sometimes do more specialized exams on patients. For example, doing a physical exam, administering a vaccination, or doing a blood exam. I want to also be able to track specific information related to each of these specialized exams. I also want a way to see the patient's family medical conditions that they choose to disclose. I only want to store a patient's relatives if they have known medical conditions.
    - I need to keep track of patient contact information, as well as their current vaccinations, insurance provider, prescriptions, and their previous appointments at my facility. I also want to be able to store any additional files or diagnosis that I have given to a patient.
    - I need to be able to give current patients a referral to specialized doctors outside my practice, if necessary. I should be able to save these doctors for future use, if I chose to use them.
    - Sometimes I need to send a specialized test to a specialized lab located off-site. I need to be able to track a report containing the information of the medical conditions it is testing, the contact information of the specialized lab that is processing the report, the appointment the lab report is associated with. When the lab completes the lab report, I want to be able to see the results.

- Include specific constraints related to database
    - Relative and relative conditions (1,N)
    - relative conditions and medical conditions (0,N)
    - Medical condition and tested_for (0,N)
    - Tested_for and Lab report (1,N)
    - Lab report and Report Creators(1,N)
    - Report Creators and specialized lab (0,N)
    - Lab report and Appointment reports (1,1)
    - Appointment reports and appointment (0,N)
    - Medical condition and diagnoses (0,N)
    - Medical Condition and Parent Category (0,1)
    - Medical Condition and Subcategory/Code (0,N)
    - Appointment and Additional Exams (0,N)
    - Additional Exams and Exams (1,1)
    - Appointment and Appointment Medical Conditions(0,N)
    - Appointment Medical Conditions and Medical Conditions(0,N)
    - Diagnoses and Appointment (0,N)
    - Employee and Diagnoses (0,N)
    - Appointment and Participate (1,1)
    - Appointment and Appointment Employees(1,N)
    - Employee and Appointment Employees (0,N)
    - Employee and Prescriber (0,N)
    - Prescription and prescriber (1,1)
    - Prescription and taking (1,1)
    - Patient and taking (0,N)
    - Patient and referral (0,N)
    - Patient and about (0,N)
    - Patient and participate (0,N)
    - Patient and Immunized_patients (0,N)
    - Patient and Patient Contacts (0,1)
    - Emergency Contacts and Patient Contacts (1,N)
    - About and archived files (0,1)
    - Archived files and created_by (1,1)
    - Referrable doctors and referral (0,N)
    - Employee and referral (0,N)
    - Employee and created_by (0,N)
    - Employee and employee immunizations (0,N)
    - Immunized_employees and immunizations (0,N)
    - Immunized_patients and immunizations (0,N)
    - Patient and Insurance Covers (0,N)

- Insurance Covers and Insurance Provider (0,N)
- Patient and family_history (0,N)
- family_history and relative (1,1)
- Prescription and Prescription-pharmacies (1,1)
- Pharmacies and Prescription-pharmacies (0,N)
- Lab report and exam reports (0,1)
- Exam reports and exams (0,1)
- Specialized_labs and Accepted_tests (1,N)
- Tests and Accepted_tests (0,N)
- Every specialized exam must be an exam
- These specialized exams are fully disjoint
- An exam must be specialized (total specialization)

◆ State at least 10 functional requirements

- Get all appointments that a patient has had at this practice.
- Prescribe a medication to a patient
- Provide a patient a referral to a specialized doctor
- Create a new patient, and add their family history
- Get all patients based on their insurance provider
- Get all patients that had appointment with certain doctor in the last 7 days
    - Used for COVID-19 tracing to notify patients
- Get all info about a patient in a single view.
    - (doctor wants to get all info on patient)
- Get all patients who are vaccinated for a specific vaccine.
- Get all patients who were prescribed a specific drug.
    - In case of recall, or price jumps of brand-name
- Get contact info of a patient's emergency contact
- See the date of the most recent appointment of a patient
    - For front-desk appointment scheduling.
- Get health metrics (average, min, max) of patient history within a specified time range.

➔ **ER Model (Section Three)**

*Nathan Larkin: brainstormed attributes for entities, relationships, wrote some of the employee model, worked on ER diagram*
*Jack Gordon: Created entities, attributes, and relationships; worked on ER diagram*

*Paul Ofremu Jr.: Contributed input and ideas on entities, attributes, and relationships; worked on ER diagram*

- Report entities and their attributes, and relationships one by one

Every **employee** has an id, a name, birthday, salary, social security number, phone number and address, a unique DEA number, malpractice insurance number, medical license number, and role. Any employee can create an archived file.

Each **employee** who is a Nurse, General Practitioner, or Physician Assistant can participate in zero to many appointments, prescribe zero to many medications to patients, and create referrals for patients.

Employee.role = {u | u ∈ General Practitioner, Nurses, Physician Assistant, lab technicians, administrative staff, residents, interns}

Every **patient** has an id, a preferred pharmacy (its name and address), billing information (a card number), a phone number, birthday, family history, email, social security number, allergies, emergency contact (name, phone). Every patient can be covered by an insurance provider. A patient can have multiple prescriptions.

**Immunizations** include an id and the type of immunization. A patient can receive an immunization, and also an employee can receive an immunization.

Every **insurance provider** has a policy number, the name of the provider, and status of whether it is covered (in network)

Every **appointment** has an id, a room number, a date, a time, blood pressure, primary physician, weight, height, temperature, miscellaneous notes. Every appointment is affiliated with exactly one patient. Zero to many medical conditions can be recorded at each appointment, depending on what the patient is experiencing.

There are many different types of **exams** and tests that can be done during every appointment. Every exam is affiliated with exactly one appointment. In practice there could be dozens or even hundreds of different types of entities for different exams, since every medical test is very specific and unique in what is being recorded.

For example, a **physical exam** would have an id, and entries for skin, eyes, nose, ears, mouth, body, reflexes, vision, hearing, spine etc.

A **blood exam** would have an id, and information for various parameters such as blood type, blood sugar, antibodies, complete blood count, metabolic panel etc.

A **vaccination** would have an id, the type of vaccination, patient reaction to symptoms, location of injection

etc...

There are **Prescriptions**, including a drug name quantity, dose, refills, instructions, pharmacy address, and date of the prescription. Prescriptions must be created by a single employee, and are taken by a patient.

For **Pharmacies**, there is a pharmacy address and name.

For **referrals**, there is an id, the id of doctor who referred, the id of doctor being referred to, the patient id.

For **referrable doctors**, there is an id, the doctor's name, the specialization, and phone number.

For **specialized laboratories**, include a lab id, tests they accept, contact information (phone and address)

For **lab reports**, it includes an id, info, and result info. The lab report is conducted by a specialized lab, and is affiliated with exactly one appointment. Each lab report tests for a single medical condition.

**Medical conditions** include a name and the ICD-10 code, and a boolean indicating whether it is a code or a category.

A **Diagnosis** relationship includes the medical condition code, appointment id, patient id, id of doctor who made diagnosis, and justification/comments

**Archived Files** stores a file id, file name, file blob. It must be created by an employee and can also be about a patient.

A **relative** is defined by a patient id, the type of relative, and additional notes. Every relative must have multiple medical conditions, and is affiliated with only one patient.

**Emergency contacts** include a name, patient id, and two phone numbers.

**Tests** include a test id and the name of the test

### ➔ Relational Model

**Patient**(patient_id, phone_number, birthday, email, ssn, address)
FD = {patient_id → phone_number, birthday, email, ssn, address}
In 3NF, no non-prime attributes are dependent on other non-prime attributes.

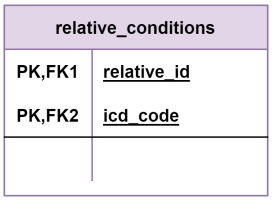
| Patients | |
|---|---|
| PK | patient_id |
| | phone_number |
| | birthday |
| | email |
| | ssn |
| | address |
| | name |
| | gender |

**Relative**(relative_id, relative_type, additional_notes, patient_id)
FD = {relative_id → relative_type, additional_notes, patient_id}
In 3NF, no non-prime attributes are dependent on other non-prime attributes.

| Relatives | |
|---|---|
| PK | relative_id |
| FK | patient_id |
| | relative_type |
| | additional_notes |

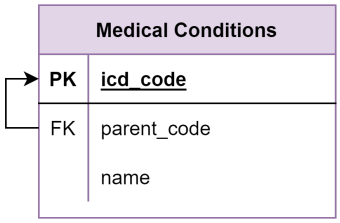
Includes the patient foreign key because it is in a many-to-one relationship with patients.

**relative_conditions**(relative_id, icd_code)
FD = {}
In 3NF, no non-prime attributes are dependent on other non-prime attributes.



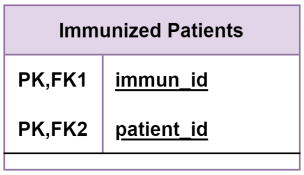

This relation creates a many-to-many relationship between relatives and medical conditions.

**Medical Condition**(icd_code, condition_name)
FD = {icd_code → condition_name; condition_name → icd_code }
In 3NF, no non-prime attributes are dependent on other non-prime attributes.




Medical conditions is in a recursive relationship with itself.

**immunized_patients**(immun_id, patient_id)
FD = {}
In 3NF, no non-prime attributes are dependent on other non-prime attributes.




Creates many-to-many relationship between Immunization and Patient

**Immunization**(immunization_id)
FD = {}
In 3NF, no non-prime attributes are dependent on other non-prime attributes.

| Immunization | |
|---|---|
| PK | immunization_id |
| | immunization_type |

**immunized_employees**(immun_id, emp_id)
FD = {}
In 3NF, no non-prime attributes are dependent on other non-prime attributes.

| Immunized Employees | |
|---|---|
| PK,FK1 | immun_id |
| PK,FK2 | emp_id |
| | |

Creates a many-to-many relationship between immunizations and employees.

**appointment_medical_conditions**(app_id, symptom_code)
FD = {}
In 3NF, no non-prime attributes are dependent on other non-prime attributes.

| Appointment_Medical_Conditions | |
|---|---|
| PK,FK1 | app_id |
| PK,FK2 | icd_code |
| | comment |

Creates a many-to-many relationship between Appointment and Medical Condition

**Archived File**(file_id, file_name, patient_id, emp_id,s3_id)
FD = {file_id → file_name, patient_id, emp_id s3_id;
s3_id→ file_id, file_name, patient_id, emp_id}
In 3NF, no non-prime attributes are dependent on other non-prime attributes.
Creates a ternary relationship between archived file, patient, and employee.

| Archived File | |
|---|---|
| PK | file_id |
| FK | patient_id |
| FK | emp_id |
| | file_name |
| | s3_id |

**Insurance Provider**(provider_id, policy_number, insurance_name, in_network)
FD = {policy_number, insurance_name → in_network;}
In 3NF, no non-prime attributes are dependent on other non-prime attributes.

| Insurance Provider | |
|---|---|
| PK | provider_id |
| | insurance_name |
| | policy_number |
| | in_network |

**insurance_covers**(policy_number, patient_id)
FD = {}
In 3NF, no non-prime attributes are dependent on other non-prime attributes.

| Insurance Covers | |
|---|---|
| PK,FK1 | provider_id |
| PK,FK2 | patient_id |
| | member_id |
| | group_number |
| | policy_holder_name |

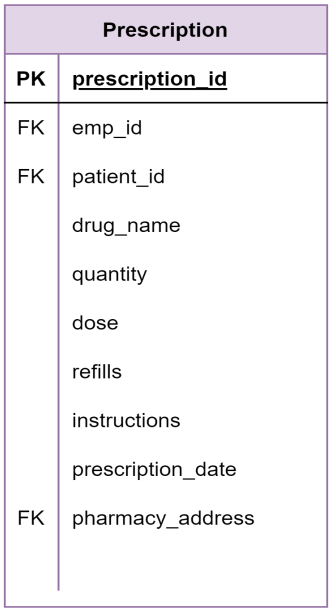
Creates a many-to-many relationship between Insurance Provider and Patient

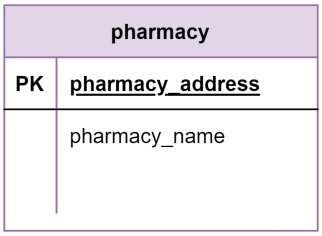
**Prescription**(prescription_id, drug_name, quantity, dose, refills, instructions, prescription_date, pharmacy_name, pharmacy_address)
FD={prescription_id → drug_name, quantity, dose, refills, instructions, prescription_date, pharmacy_name, pharmacy_address;
pharmacy_address → pharmacy_name }

Not 3NF, transitive dependency: prescription_id → pharmacy_name → pharmacy_address

Decomposition:
**Prescription**(prescription_id, drug_name, quantity, dose, refills, instructions, prescription_date, pharmacy_address)

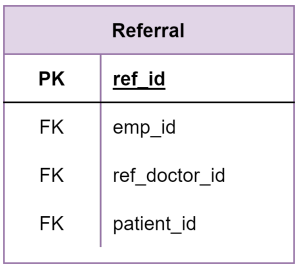
**Pharmacy**(pharamacy_address, pharmacy_name)



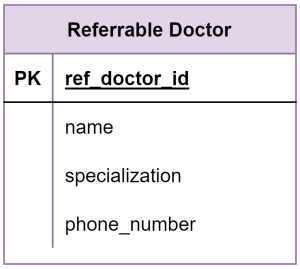




Prescriptions are in a many-to-one relationship with patients, employees, and pharmacies.

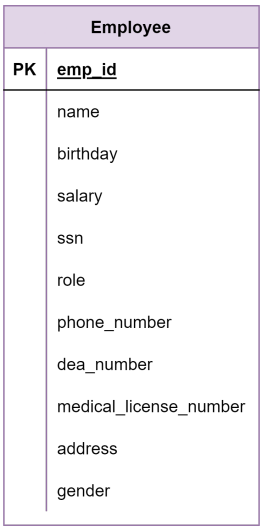
**Referral**(ref_id, emp_id, ref_doctor_id, patient_id)
FD = {ref_id → emp_id, ref_doctor_id, patient_id}
In 3NF, no non-prime attributes are dependent on other non-prime attributes.



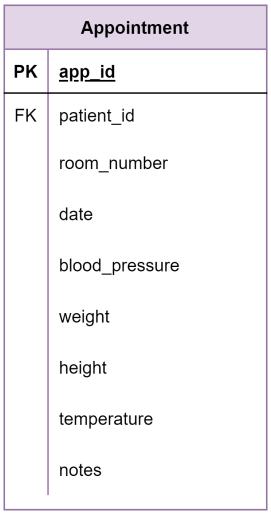

**Referrable Doctor**(ref_doctor_id, name, specialization, phone_number)
FD = {ref_doctor_id → name, specialization, phone_number;
Phone_number → ref_doctor_id, name, specialization}
In 3NF, no non-prime attributes are dependent on other non-prime attributes.



**Employee**(emp_id, name, birthday, salary, ssn, role, phone_number, dea_number,, medical_license_number, address)
FD = {emp_id → name, birthday, salary, ssn, role, phone_number, dea_number, medical_license_number, address}




**Appointment**(app_id, room_number, date, blood_pressure, weight, height, temperature, notes,patient_id)
FD = {app_id → room_number, date, blood_pressure, weight, height, temperature, notes,patient_id}
In 3NF, no non-prime attributes are dependent on other non-prime attributes.




In a many-to-one relationship with patients.

**appointment_employees**(emp_id, app_id)
FD = {}
In 3NF, no non-prime attributes are dependent on other non-prime attributes.

| Appointment Employees | |
|---|---|
| PK,FK1 | emp_id |
| PK,FK2 | app_id |
| | |

Creates a many-to-many relationship between appointments and employees.

**diagnoses**(emp_id, patient_id, app_id, icd_code, comment)
FD = {emp_id, patient_id, app_id, icd_code → comment)
In 3NF, no non-prime attributes are dependent on other non-prime attributes.

| Diagnoses | |
|---|---|
| PK,FK1 | emp_id |
| PK,FK2 | patient_id |
| PK,FK3 | app_id |
| PK,FK4 | icd_code |
| | comment |

Creates a quaternary relationship between patients, employees, appointments, and medical conditions.

**Lab Report**(report_id, info, result_info, icd_code, app_id, file_id)
FD = {report_id → info, result_info, icd_code, app_id, file_id}
In 3NF, no non-prime attributes are dependent on other non-prime attributes.

| Lab Reports | |
|---|---|
| PK | report_id |
| FK | icd_code |
| + | info |
| + | result_info |
| FK | file_id |
| FK | app_id |
| FK | exam_id |

Lab reports are in a many to one relationship with appointments and exams.

**report_creators**(report_id, lab_id)
FD = {}
In 3NF, no non-prime attributes are dependent on other non-prime attributes.

| Report Creators | |
|---|---|
| PK,FK1 | report_id |
| PK,FK2 | lab_id |
| | |

This relation creates a many-to-many relationship between lab reports and specialized labs.

**Specialized Lab**(lab_id, phone_number, address)
FD = {lab_id → phone_number, address}
In 3NF, no non-prime attributes are dependent on other non-prime attributes.

| Specialized Lab | |
|---|---|
| PK | lab_id |
| + | phone_number |
| + | address |

**accepted_tests**(test_id, lab_id)
FD = {}
In 3NF, no non-prime attributes are dependent on other non-prime attributes.

| Accepted Tests | |
|---|---|
| PK,FK1 | test_id |
| PK,FK2 | lab_id |

This relation creates a many-to-many relationship between specialized labs and tests.

**Test**(test_id)
FD = {}

In 3NF, no non-prime attributes are dependent on other non-prime attributes.

| Test | |
|---|---|
| **PK** | **test_id** |
| | test_name |

**Exam**(exam_id, app_id, comment)
FD = {exam_id → app_id, comment}
In 3NF, no non-prime attributes are dependent on other non-prime attributes.

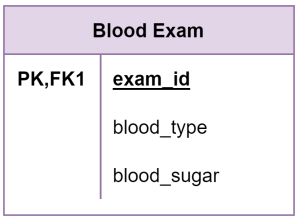
| Exam | |
|---|---|
| **PK** | **exam_id** |
| FK | app_id |
| | comment |

Generalization:
All the specialized exams, such as Blood Exams, Covid Exams, and Administered Vaccines, share common attributes. This formed an ‘ISA’ relationship, and we went to the textbook to understand the best ways to model this relationship. We generalized each shared attribute between all the exams. We discovered that we need to determine two things before moving forward with one of the four approaches listed in the textbook. The first is total vs. partial specialization. Our relationships have total specialization, which means each exam must be specialized. Stated in another sense, no ‘exam’ can exist without being a specialized exam.

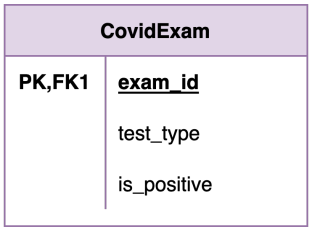
The “disjointness constraint” is the second trait we need to determine a proper model. Every specialized model is completely disjoint, which means if something is a blood exam, it cannot be any other type of specialized exam. With our relations having total specialization and being disjoint, we opted to generalize the shared attributes into a single relation, “Exam,” a superclass with non-shared attributes stored in separate relations. This is similar to option 8A listed in the textbook (p. 299, chapter 9.2.1). I created a diagram for each option (8A-8D) and recorded it in the appendix for convenience. A subclass was created for each specialized exam.

Additionally, the subclass primary key is a foreign key that points to the exam superclass primary key. Option 8B was also a valid option. Options 8C and 8D were not good options because there is little shared overlap between each specialized exam, which would result in many null values.

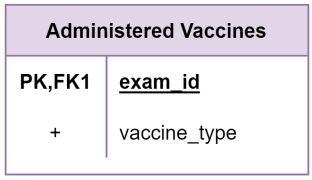
**Blood Exam**(exam_id, blood_type, blood_sugar)
FD = {exam_id → blood_type, blood_sugar}
In 3NF, no non-prime attributes are dependent on other non-prime attributes.




**Covid Exam**(exam_id, test_type, is_positive)
FD={exam_id →test_type, is_positive}
In 3NF, no non-prime attributes are dependent on other non-prime attributes.




**administered_vaccines**(exam_id, vaccine_type)
FD = {exam_id → vaccine_type}
In 3NF, no non-prime attributes are dependent on other non-prime attributes.




### ➔ Data Dictionary

| *Pharmacy* | | | |
|---|---|---|---|
| Attribute Name | Data Type | Field | Constraints |

| | | Length | |
|---|---|---|---|
| **pharmacy_address** | string | 200 | not null primary key |
| pharmacy_name | string | 75 | not null |

| *Patient* | | | |
|---|---|---|---|
| Attribute Name | Data Type | Field Length | Constraints |
| **patient_id** | int | 4 | primary key, not null, must be non-negative integer |
| phone_number | string | 50 | nullable |
| birthday | date | 4 | not null, only store year, month, and day |
| email | string | 255 | nullable |
| ssn | string | 11 | nullable unique |
| address | string | 200 | nullable |
| name | string | 75 | not null |
| gender | string | 50 | not null |

| *Insurance Provider* | | | |
|---|---|---|---|
| Attribute Name | Data Type | Field Length | Constraints |
| **insurance_name** | string | 75 | not null primary key |
| policy_number | string | 20 | not null |
| in_network | boolean | 1 | not null |

| prescriptions | | | |
|---|---|---|---|
| Attribute Name | Data Type | Field Length | Constraints |
| **prescription_id** | int | 4 | primary key, not null, must be non-negative integer |
| emp_id | int | 4 | foreign key, not null, must be non-negative integer |
| patient_id | int | 4 | foreign key, not null, must be non-negative integer |

| | | | |
|---|---|---|---|
| drug_name | string | 100 | not null |
| quantity | int | 4 | >=0 not null |
| dose | string | 50 | not null |
| refills | int | 4 | not null, >=0 |
| instructions | text | | nullable |
| prescription_date | timestamptz | 10 | not null |
| pharmacy_address | string | 200 | not null, foreign key |

| referrals | | | |
|---|---|---|---|
| Attribute Name | Data Type | Field Length | Constraints |
| **ref_id** | int | 4 | primary key, not null, must be non-negative integer |
| emp_id | int | 4 | foreign key, not null, must be non-negative integer |
| ref_doctor_id | int | 4 | foreign key, not null, must be non-negative integer |
| patient_id | int | 4 | foreign key, not null, must be non-negative integer |

| immunizations | | | |
|---|---|---|---|
| Attribute Name | Data Type | Field Length | Constraints |
| **immunization_id** | int | 4 | primary key, not null, must be non-negative integer |
| immunization_type | string | 50 | not null |

| immunized_patients | | | |
|---|---|---|---|
| Attribute Name | Data Type | Field Length | Constraints |
| **immun_id** | int | 4 | primary key, foreign key, not null, must be non-negative integer |
| **patient_id** | int | 4 | primary key, foreign key, not null, must be non-negative integer |

| employees | | | |
|---|---|---|---|
| Attribute Name | Data Type | Field Length | Constraints |
| **emp_id** | id | 4 | primary key, not null, must be non-negative integer |
| name | string | 75 | not null |
| birthday | date | 4 | not null |
| salary | int | 10 | not null, must be non-negative (stored in cents) |
| ssn | string | 11 | not null |
| role | string | 50 | not null |
| phone_number | string | 50 | not null |
| dea_number | string | 9 | nullable |
| medical_license_number | string | 10 | nullable |
| address | string | 200 | not null |
| gender | string | 50 | not null |

| diagnoses | | | | |
|---|---|---|---|---|
| Attribute Name | Data Type | Field Length | Constraints | Notes |
| **emp_id** | int | 4 | primary key, not null, must be non-negative integer | |
| **patient_id** | int | 4 | primary key, not null, must be non-negative integer | |
| **app_id** | int | 4 | primary key, not null, must be non-negative integer | |
| **icd_code** | string | 7 | primary key, not null | largest icd code in 70k+ is 7 characters (E083211) |
| comment | text | | nullable | |

| exams | | | |
|---|---|---|---|
| Attribute Name | Data Type | Field Length | Constraints |
| **exam_id** | int | 4 | primary key, not null, must be non-negative integer |
| app_id | int | 4 | foreign key, not null, must be non-negative integer |
| comment | text | | nullable |

| blood_exams | | | |
|---|---|---|---|
| Attribute Name | Data Type | Field Length | Constraints |
| **exam_id** | int | 4 | primary key, foreign key, must be non-negative integer |
| blood_type | string | 3 | not null |
| blood_sugar | string | 12 | not null |

| covid_exams | | | | |
|---|---|---|---|---|
| Attribute Name | Data Type | Field Length | Constraints | Notes |
| **exam_id** | int | 4 | primary key, foreign key, must be non-negative integer | |
| test_type | string | 20 | not null | |
| is_positive | bool | 1 | nullable | doctor probably wants to save the exam while waiting for the covid test results |

| administered_vaccines | | | |
|---|---|---|---|
| Attribute Name | Data Type | Field Length | Constraints |
| **exam_id** | int | 4 | primary key, foreign key, must be non-negative integer |
| vaccine_type | string | 50 | not null |

| relatives | | | |
|---|---|---|---|
| Attribute Name | Data Type | Field Length | Constraints |
| **relative_id** | int | 4 | primary key, must be non-negative integer |
| patient_id | int | 4 | foreign key, not null, must be non-negative integer |
| relative_type | string | 30 | not null |
| additional_notes | text | | nullable |

| relative_conditions | | | |
|---|---|---|---|
| Attribute Name | Data Type | Field Length | Constraints |
| **relative_id** | int | 4 | primary key, foreign key, must be non-negative integer |
| **icd_code** | str | 7 | primary key, foreign key, must be non-negative integer |

| appointment_medical_conditions | | | |
|---|---|---|---|
| Attribute Name | Data Type | Field Length | Constraints |
| **app_id** | int | 4 | primary key, foreign key, must be non-negative integer |
| **icd_code** | str | 7 | primary key, foreign key, must be non-negative integer |
| comment | text | | nullable |

| archived_files | | | |
|---|---|---|---|
| Attribute Name | Data Type | Field Length | Constraints |
| **file_id** | int | 4 | primary key, must be non-negative integer |

| | | | |
|---|---|---|---|
| patient_id | int | 4 | nullable, foreign key, must be non-negative integer |
| emp_id | int | 4 | not nullable, foreign key, must be non-negative integer |
| file_name | string | 255 | not nullable |
| s3_id | string | 255 | not nullable |

| appointments | | | | |
|---|---|---|---|---|
| Attribute Name | Data Type | Field Length | Constraints | Notes |
| **app_id** | int | 4 | primary key, must be non-negative integer | |
| patient_id | int | 4 | not nullable, foreign key, must be non-negative integer | |
| room_number | str | 6 | nullable ? | |
| date | timestamptz | 8 | not nullable, need to track the time, format is: YYYY-MM-DD HH:MM:SS, 24H clock | |
| blood_pressure | str | 7 | not nullable, format "###/###" | |
| weight | float | | Not nullable | Using float because precision is not necessary, improved performance and storage by using float compared to decimal |
| height | decimal | | Not nullable | using decimal for precision |
| temperature | decimal | | Not nullable | using decimal for precision |
| notes | text | | nullable | |

| referrable_doctors | | | |
|---|---|---|---|
| Attribute Name | Data Type | Field Length | Constraints |
| **ref_doctor_id** | int | 4 | primary key, must be non-negative integer |
| name | string | 75 | not nullable |

| | | | |
|---|---|---|---|
| specialization | string | 100 | nullable |
| phone_number | string | 50 | nullable |

| lab_reports | | | |
|---|---|---|---|
| Attribute Name | Data Type | Field Length | Constraints |
| **report_id** | int | 4 | primary key, must be non-negative integer |
| icd_code | string | 7 | foreign key, not nullable |
| result_info | text | | nullable |
| file_id | int | 4 | foregin key, nullable |
| app_id | int | 4 | foregin key, nullable |
| exam_id | int | 4 | Foreign key, nullable |

| report_creators | | | |
|---|---|---|---|
| Attribute Name | Data Type | Field Length | Constraints |
| **report_id** | int | 4 | primary key, foreign key, must be non-negative integer |
| **lab_id** | int | 4 | primary key, foreign key, must be non-negative integer |

| specialized_labs | | | |
|---|---|---|---|
| Attribute Name | Data Type | Field Length | Constraints |
| **lab_id** | int | 4 | primary key, must be non-negative integer |
| phone_number | string | 50 | |
| address | string | 200 | |
| lab_name | string | 200 | nullable |

| insurance_covers | | | |
|---|---|---|---|
| Attribute Name | Data Type | Field Length | Constraints |
| **provider_id** | int | 4 | primary key, foreign key, must be non-negative integer |
| **patient_id** | int | 4 | primary key, foreign key, must be non-negative integer |
| member_id | string | 12 | not null |
| group_number | string | 12 | not null |
| policy_holder_name | string | 75 | not null |

| medical_conditions | | | | |
|---|---|---|---|---|
| Attribute Name | Data Type | Field Length | Constraints | |
| **icd_code** | string | 7 | primary key | |
| name | string | 206 | not null | largest is "Injury of right internal carotid artery, intracranial portion, not elsewhere classified with loss of consciousness greater than 24 hours without return to pre-existing conscious level with patient surviving" |
| parent_code | string | 7 | nullable, foreign key, recursively points to medical_condition.icd_code | |

| immunized_employees | | | |
|---|---|---|---|
| Attribute Name | Data Type | Field Length | Constraints |
| **immun_id** | int | 4 | primary key, foreign key, must be non-negative integer |
| **emp_id** | int | 4 | primary key, foreign key, must be non-negative integer |

| accepted_tests | | | |
|---|---|---|---|
| Attribute Name | Data Type | Field Length | Constraints |
| **test_id** | int | 4 | primary key, foreign key, not null, must be non-negative integer |
| **lab_id** | int | 4 | primary key, foreign key, not null, must be non-negative integer |

| tests | | | |
|---|---|---|---|
| Attribute Name | Data Type | Field Length | Constraints |
| **test_id** | int | 4 | primary key, not null, must be non-negative integer |
| test_name | string | 255 | not null |

| appointment_employees | | | |
|---|---|---|---|
| Attribute Name | Data Type | Field Length | Constraints |
| **emp_id** | int | 4 | foreign key, not null, must be non-negative integer |
| **app_id** | int | 4 | foreign key, not null, must be non-negative integer |

| emergency_contacts | | | |
|---|---|---|---|
| Attribute Name | Data Type | Field Length | Constraints |
| **name** | string | 75 | not null, primary key |
| **patient_id** | int | 4 | foreign key, primary key, not null, must be non-negative integer |
| phone_1 | string | 50 | not null |
| phone_2 | string | 50 | |

➔ **Implementation**

- Get all appointments that a patient has had at this practice.
    - Join **Appointments** table and **Patient** table on *patient_id*, where patient.*patient_id* = id_number
    - With the foreign key ‘patient_id’ in appointments, we are able to search for all appointments a patient has been part of. We filter the appointments to only include appointments that involve the specified patient.
- Prescribe a medication to a patient
    - Insert new row into **Prescription** table with values for columns: *emp_id, patient_id, drug_name, quantity, dose, refills, prescription_date, pharmacy_address*; being not null.
    - An important note with prescribing to patients, is handling the event where a new pharmacy is selected by the patient. We need to create a new pharmacy entity instance, before we can create the prescription going to the new pharmacy. This is caused by our foregin key constraints of ‘pharmacy_address’ within the prescription table.
- Provide a patient a referral to a specialized doctor
    - Insert new row into **Referral** table with values for columns: *emp_id, ref_doctor_id, patient_id*: being not null
- Create a new patient, and add their family history
    - Insert a new **Patient**, returning their *patient_id*, and then insert into **Relatives** the patient’s relatives, returning their *relative_id*, and finally inserting **Relative Conditions** for each of those relatives.
- Get all patients based on their insurance provider
    - Join **Patients** with **Insurance Covers** on *patient_id*, and join **Insurance Covers** with **Insurance Providers** on *provider_id*, where the insurance name is specified
- Get all patients that had appointment with certain doctor in the last 7 days
    - Used for COVID-19 tracing to notify patients
    - Join **Patients** and **Appointments** on *patient_id*, and join **Appointment Employees** and **Appointments** on *app_id*, and join **Appointment Employees** and **Employees** on *emp_id*, and then get the **Patients** for the employee *name* where the appointment *date* is within one week.
- Get all info about a patient in a single view.

    - (doctor wants to get all info on patient)
    - Select the 3 most recent appointments by joining **Appointments** and **Patients** on *patient_id*, and ordering by *date* in descending order
    - Select the **Medical conditions** from those 3 most recent appointments by joining the appointments' *app_id* on **Diagnoses** *app_id*, and joining **Diagnoses'** *icd_code* with the **Medical Conditions** *icd_code*
    - Find the 2 most recent prescriptions by joining **Patients** *patient_id* and **Prescriptions** *patient_id*, and querying for the patient's name, and ordering by the *prescription_date* in descending order
- Get all patients who are vaccinated for a specific vaccine.
    - Select *name*, *patient_id*; Join **Appointments** table and **Patient** table on *patient_id*, and Join with **Exam** table on *app_id* and Join with **Administered Vaccines** table on *exam_id*, where *vaccine_type* = "VACCINE"
- Get all patients who were prescribed a specific drug.
    - In case of recall, or price jumps of brand-name
    - Select *name*, *patient_id*; Join **Patient** table and **Prescriptions** table on *patient_id*, where prescription.*drug_name* = "DRUG NAME"
- Get contact info of a patient's emergency contact
    - Select *name, phone_1, phone_2*; Join on **Patient** table and **Emergency Contact** on *patient_id*, where patient.*patient_id* = id_number
- See the date of the most recent appointment of a patient
    - For front-desk appointment scheduling.
    - We used a query to find all **appointments** a **patient** has had, and applied an aggregate function to find the largest date within the patient's appointment history. Since all appointments in the database are historical (no future appointments may be in the database), we know the largest date will be the most recent appointment.
    - This intends to provide little private information to the front-desk, to protect the patient's privacy.
- Get health metrics (average, min, max) of patient history within a specified time range.
    - We used a query to find all **appointments** that have a specific patient_id and then we applied aggregation

functions min, max, and average on all the resulting appointments to get the metrics for the metrics for the patient..

Another important aspect of our implementation was implementing indexes on a subset of frequently used attributes in our queries. We noticed that the “patient_id” foreign key was commonly used in the equality joins. We decided that a hash index is well suited to improve the performance of these queries. The performance benefits of the index will likely be noticeable because the appointment table will likely grow to become one of the largest tables in the database. Similarly, we created a hash index on prescriptions.drug_name and patients.name because we foresee many queries where these attributes will be used in equality joins.

Lastly, we created a B-tree index on appointments.date to improve the read performance of queries involving the newest or oldest appointments or appointments older/younger than a provided timestamp. Based on the PostgreSQL documentation, B-tree indexes will likely improve performance when the data set becomes large and the resulting number of appointments becomes a smaller percentage of overall results. If we were to return the majority of appointments, a simple sequential read and sort would likely be better. Since it is likely most queries will only want to retrieve a small subset of total appointments, we thought the B-tree index on appointments.dates would be beneficial.

Additionally, the medical condition queries were quite the challenge. We wanted to allow the office workers to discover subcategories for a selected category. We imagined the typical flow of applying conditions would be navigating from the broadest categories to the subcategories/codes within the selected category. We discovered that recursive searching in PostgreSQL would be the perfect solution for this. We are particularly proud of the Postgresql recursive solutions we created to provide this functionality.

➔ **Summary**

We successfully implemented a database that realistically can handle the complexity of a general health medical practice. We worked through the steps of creating an ER model, a relational model, and a data dictionary and then implemented it in a Postgresql database instance. We generated complex mock data to populate our database instance. After designing the queries, we further improved performance by utilizing hash and B-tree indexes.

We underestimated the significant amount of domain-specific knowledge required to design the database. We discovered the considerable work needed to maintain

consistency during name revisions and attribute additions, which occurred when making incremental database changes. Similarly, it was time-consuming to revise all past versions of the ER model, relational schema diagram, and the data dictionary. Our experience highlights the benefits of a robust database design at the beginning of a project.

Contributions:
Nathan Larkin: Wrote many requirements and contributed ideas to functional requirements. Brainstormed ideas for entities, relationships and worked on ER diagrams. Played a significant role in designing the superclass/subclass exam relationship. Contributed by creating multiple tables in the relational schema diagram. Wrote roughly one-third of the table creation statements. Created a portion of the SQL queries. I filled in multiple tables in the data dictionary. Researched the design choices behind float vs. decimal vs. integer and provided recommendations for which to select based on the attribute usage. Designed a comprehensive python application to generate data for the entire project. Wrote a large portion of the summary and all the final paragraphs in the implementation section. Wrote the implementation paragraphs that describe the generalization process for specialized exams. Researched and educated members on the role of ICD codes and designed the recursive relationship model and recursive queries.

Jack Gordon: Worked on introduction, requirement analysis, wrote functional requirements. Created entities, relationships and attributes for the ER model using the requirements. Worked on the ER diagram, and on the mapping of the ER model to the relational model diagram. Contributed to the functional requirements of the relational model to ensure 3NF. Worked on converting the relational model into a data dictionary (determining data types). Worked on the table creation statements, and functional requirement queries. Contributed to many parts of the final presentation, and to all parts of the final report.

Paul Ofremu Jr.: Convert outline into final introduction with additional information and details.Contributed to functional requirements of the database provided ideas. Contributed input and ideas on entities, attributes, and relationships, worked on the ER diagram. Worked on mapping of the ER diagram to relational schema diagram. Worked on the normalization of the relations and assuring relations are in 3NF. Worked on the data dictionary and provided input. Worked on implementation of functional requirements, implementing queries.

➔ **Access Link**

https://drive.google.com/file/d/1GEaMB-TxBMUcUrfy9AM8JuKbycCR1Vnt/view?usp=sharing

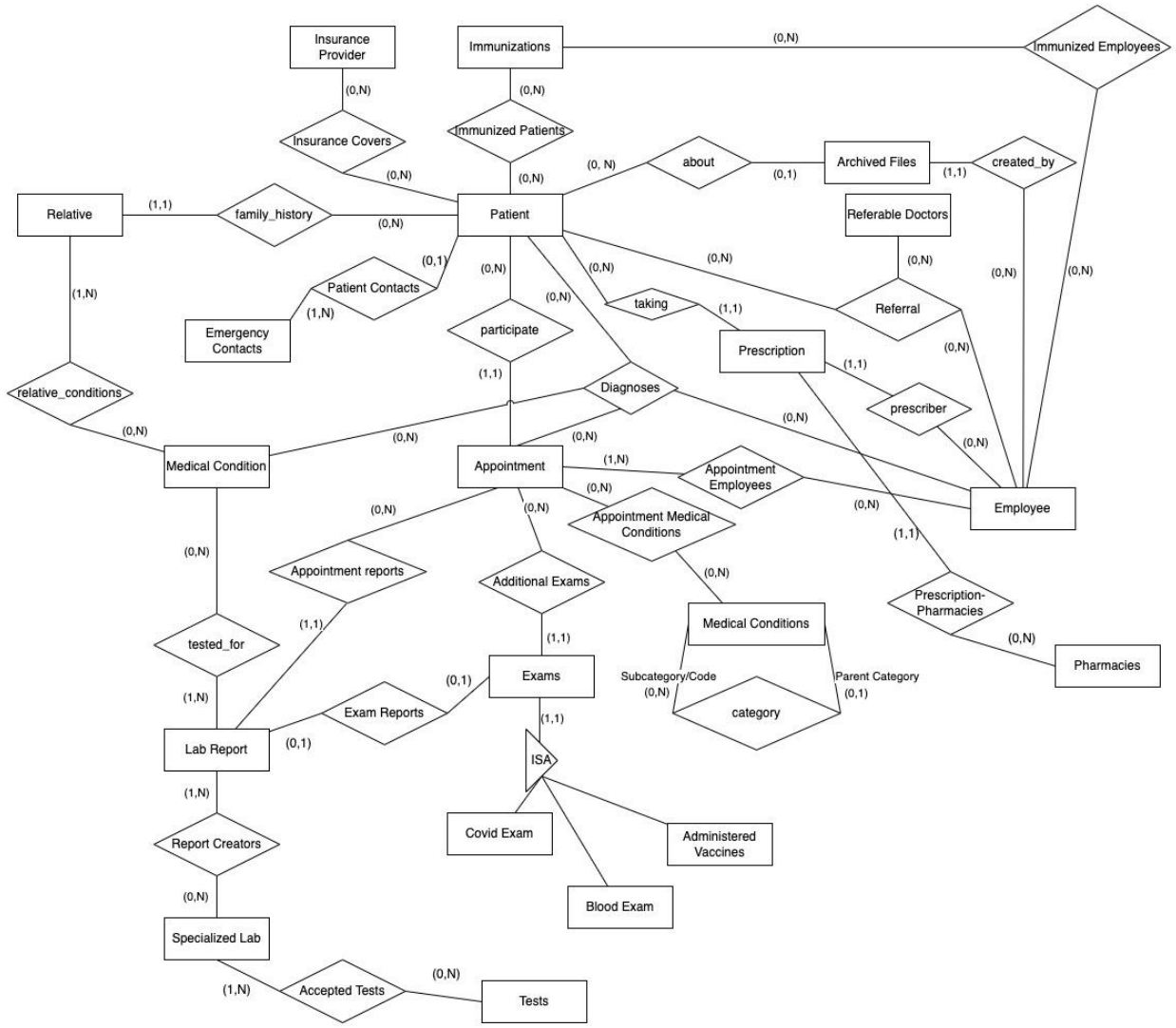
https://youtu.be/3JLZdg9P3-A

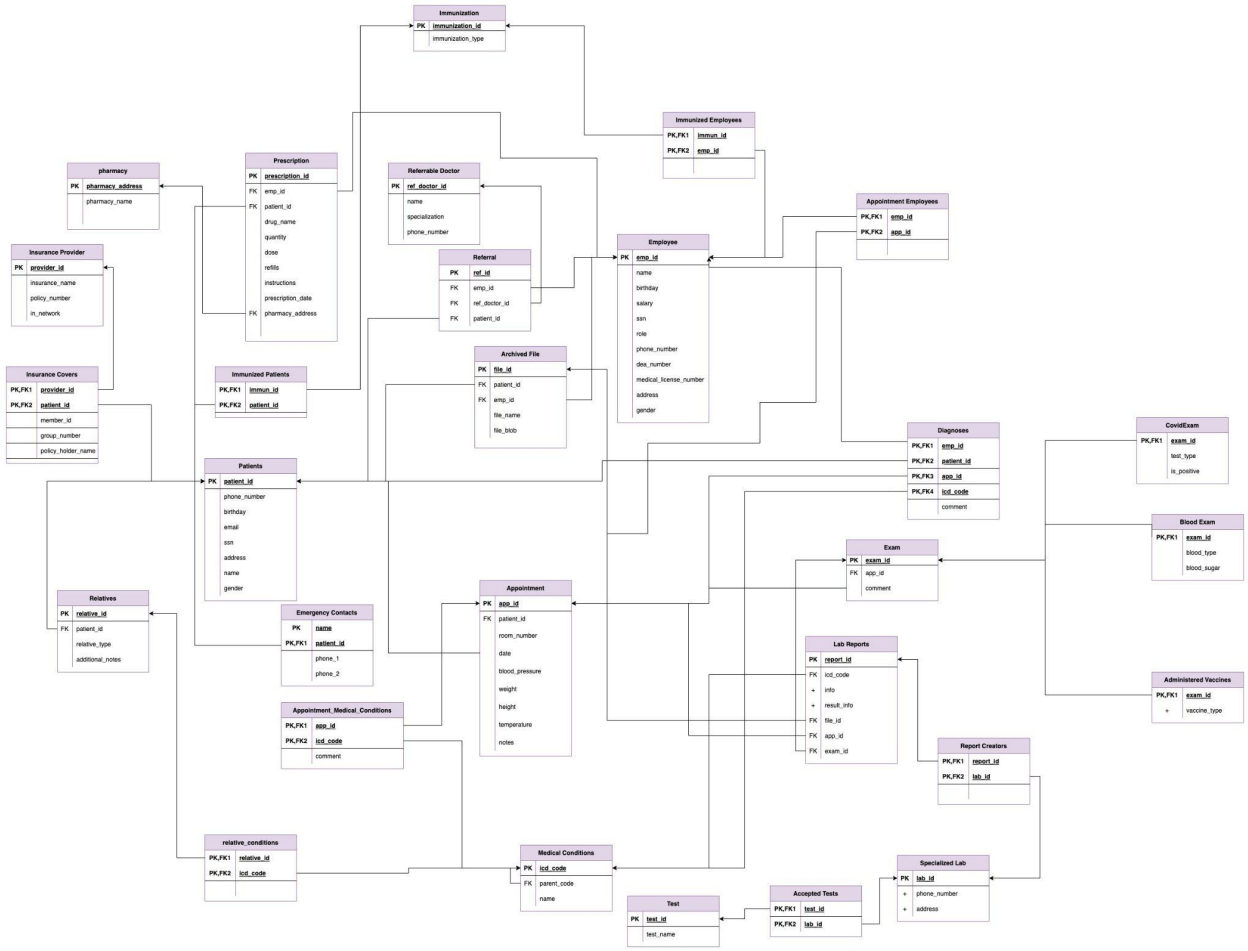
➔ **Appendix**

ISA superclass/subclass relationship designs
https://dbdiagram.io/d/6251f1e52514c97903000cc2

Data generation link
https://github.com/nalarkin/databases-mock-medical-data

Github With SQL for Creating/Deleting/Queries
https://github.com/BinaryRealm/healthcare-db


Insurance Provider
Immunizations
Immunized Employees
Insurance Covers
Immunized Patients
about
Archived Files
created_by
Relative
family_history
Patient
Referable Doctors
Patient Contacts
Emergency Contacts
participate
taking
Referral
Prescription
relative_conditions
Diagnoses
prescriber
Medical Condition
Appointment
Appointment Employees
Employee
Appointment Medical Conditions
Appointment reports
Additional Exams
Medical Conditions
Prescription-Pharmacies
Pharmacies
tested_for
Exams
Subcategory/Code
Parent Category
category
Exam Reports
Lab Report
ISA
Covid Exam
Administered Vaccines
Report Creators
Blood Exam
Specialized Lab
Accepted Tests
Tests




Hi res image of Relational Schema Diagram: https://i.imgur.com/cNwP3wi.png
Hi res image of ER Diagram: https://i.imgur.com/rxjhuLm.png

queries.sql:

```
/* HOW TO FIND ALL PARENT (SUB)CATEGORIES A CONDITION 'R94118' IS A
PART OF */
WITH RECURSIVE search_graph(icd_code, parent_code, name, is_code,
depth) AS (
   SELECT mc.icd_code, mc.parent_code, mc.name, mc.is_code, 1
   FROM medical_conditions mc
   where mc.icd_code ='R94118'
 UNION ALL
   SELECT mc.icd_code, mc.parent_code, mc.name, mc.is_code, sg.depth
+ 1
```

```
   FROM medical_conditions mc, search_graph sg
   WHERE mc.icd_code = sg.parent_code
)
SELECT * FROM search_graph order by depth;


/* HOW TO FIND ALL CHILDREN SUB(CATEGORIES)/CODES THE CURRENT
(SUB)CATEGORY 'R93' HAS */
WITH RECURSIVE search_graph(icd_code, parent_code, name, is_code,
depth) AS (
   SELECT mc.icd_code, mc.parent_code, mc.name, mc.is_code, 1
   FROM medical_conditions mc
   where mc.icd_code = 'R93'
 UNION ALL
   SELECT mc.icd_code, mc.parent_code, mc.name, mc.is_code, sg.depth
+ 1
   FROM medical_conditions mc, search_graph sg
   WHERE sg.icd_code = mc.parent_code
)
SELECT * FROM search_graph order by depth;

/*Get all appointments that a patient has had at this practice.*/
SELECT a.*
FROM appointments a, patients p
WHERE  a.patient_id = p.patient_id AND p.patient_id = 1;

/*Prescribe a medication to a patient*/
INSERT INTO prescriptions (prescription_id, emp_id, patient_id,
drug_name, quantity, dose, refills, instructions, prescription_date,
pharmacy_address)
VALUES (DEFAULT, 1, 2, 'Tylenol', 100, '500 mg', 1, 'Take when
headache', '2022-04-25 11:32:30+00', '16401 Erin Inlet\nPhillipstad,
AL 58261');

/*Provide a patient a referral to a specialized doctor*/
```

```
INSERT INTO referrals (emp_id, ref_doctor_id, patient_id)
VALUES (19, 7, 25);

/* Get recently frequent patients (More than 5 appointments in the
past week)*/
SELECT p.patient_id, p.name
FROM patients p JOIN appointments a USING(patient_id)
WHERE a."date" > current_timestamp - interval '1 week'
GROUP BY p.patient_id
HAVING count(*) > 5;

/*Create a new patient, and add their family history*/

WITH pid AS (
   INSERT INTO patients (address, birthday, email, gender, name,
patient_id, phone_number, ssn)
   VALUES ('1600 Pennsylvania Avenue NW, Washington, DC 20500',
'1900-11-14', 'test@test.com', 'Male', 'Bill Bob', DEFAULT,
'(604)876-1234', '123-78-8888')
   RETURNING patient_id
   ),
   rid AS(
       INSERT INTO relatives (additional_notes, patient_id,
relative_id, relative_type)
       VALUES ('History of cancer', (select patient_id from pid),
DEFAULT, 'father'),
       ('History of stroke', (select patient_id from pid), DEFAULT,
'mother')
       RETURNING relative_id
   )
INSERT INTO relative_conditions (icd_code, relative_id)
VALUES ('R6884', (SELECT relative_id FROM rid LIMIT 1)),
      ('R40211', (SELECT relative_id FROM rid LIMIT 1)),
      ('R87614', (SELECT relative_id FROM rid LIMIT 1 OFFSET 1)),
      ('R297', (SELECT relative_id FROM rid LIMIT 1 OFFSET 1));
```

```
/*Get all patients based on their insurance provider*/

SELECT p.patient_id, p.name
FROM patients p, insurance_covers c, insurance_providers i
WHERE p.patient_id = c.patient_id
   AND c.provider_id = i.provider_id
   AND i.insurance_name = 'Purple Shield';

/*Get all patients that had appointment with certain doctor in the
last 7 days
Used for COVID-19 tracing to notify patients*/

SELECT DISTINCT(p.patient_id), p.name, a.date
FROM patients p, appointments a, appointment_employees ae, employees
e
WHERE p.patient_id = a.patient_id
   AND ae.app_id = a.app_id
   AND ae.emp_id = e.emp_id
   AND e.emp_id = 116
   AND a."date" > current_timestamp - interval '1 week';
/*Get all info about a patient in a single view.
(doctor wants to get all info on patient)*/

/*Find last 3 appointments */
SELECT a.* as last_three
FROM appointments a, patients p
WHERE a.patient_id = p.patient_id
 AND p.patient_id = 26
ORDER BY a."date" DESC
LIMIT 3;

/*Find medical conditions reported in the last 3 appointments */
select mc.*
```

```
from (SELECT a.*
   FROM appointments a, patients p
   WHERE a.patient_id = p.patient_id
   AND p.patient_id = 26
   ORDER BY a."date" DESC
   LIMIT 3) as last_three,
diagnoses d, medical_conditions mc
where last_three.app_id = d.app_id and d.icd_code = mc.icd_code;

/*Find 2 most recent prescriptions*/
SELECT p2.*
FROM patients p, prescriptions p2
WHERE p.patient_id = 26
 AND p.patient_id = p2.patient_id
ORDER BY p2.prescription_date  DESC
LIMIT 2;



/*Get all patients who are were administered a specific vaccinated at
our practice.*/
SELECT p.patient_id, p.name
FROM patients p, appointments a, exams e, administered_vaccines v
WHERE a.patient_id = p.patient_id
   AND e.app_id = a.app_id
   AND v.exam_id = e.exam_id
   AND v.vaccine_type = 'Poliovirus';

/* IMPROVED VACCINATION because it accounts for vaccinations that
occur inside AND outside practice  */
SELECT p.*
FROM immunized_patients ip
   JOIN patients p USING(patient_id)
   JOIN immunizations i ON ip.immun_id = i.immunization_id
WHERE i.immunization_type = 'Poliovirus';
```

```
/*Get all patients who were prescribed a specific drug.
In case of recall, or price jumps of brand-name */
SELECT DISTINCT p.patient_id, p.name
FROM patients p, prescriptions d
WHERE d.patient_id = p.patient_id
   AND d.drug_name = 'Albuterol';

/*Get contact info of a patient's emergency contact*/
SELECT c.name, c.phone_1, c.phone_2
FROM patients p, emergency_contacts c
WHERE p.patient_id = c.patient_id
   AND p.patient_id = 211;

/*See the date of the most recent appointment of a patient
For front-desk appointment scheduling.*/
SELECT MAX(a.date)
FROM patients p, appointments a
WHERE p.patient_id = 47
   AND p.patient_id = a.patient_id;

/*Get health metrics (average, min, max) of patient history within a
specified time range.*/
SELECT p.*, AVG(a.weight) AS avg_weight, MIN(a.weight) AS min_weight,
   MAX(a.weight) AS max_weight, AVG(a.temperature) AS
avg_temperature,
   MIN(a.temperature) AS min_temperature, MAX(a.temperature) AS
max_temperature,
   MAX(a.height) AS height
FROM appointments a NATURAL JOIN patients p
WHERE p.patient_id = 2
 AND a."date" > current_timestamp - INTERVAL '1 year'
GROUP BY p.patient_id;

/* Find the most prescribed medications */
SELECT drug_name, count(*) AS count
```

```
FROM prescriptions p
GROUP BY drug_name
ORDER BY count DESC;


/* Find all the patients with the top 5 most appointments */
SELECT count(*) AS count, p.*
FROM appointments a, patients p
WHERE a.patient_id = p.patient_id
GROUP BY p.patient_id
ORDER BY count DESC
LIMIT 5;

/* Sort all patients in descending order of the number of
appointments they had in the past week */
SELECT count(*) AS count, p.*
FROM appointments a, patients p
WHERE a.patient_id = p.patient_id
 AND a."date" > current_timestamp - INTERVAL '1 week'
GROUP BY p.patient_id
ORDER BY count DESC;
```

Tables creation are in create.sql:

```
BEGIN;

CREATE TABLE IF NOT EXISTS pharmacies (
   pharmacy_address VARCHAR(200) NOT NULL,
   pharmacy_name VARCHAR(75) NOT NULL,
   PRIMARY KEY (pharmacy_address)
);

CREATE TABLE IF NOT EXISTS patients (
   patient_id SERIAL,
   phone_number VARCHAR(50),
```

```
    birthday DATE NOT NULL,
    email VARCHAR(255) NOT NULL,
    ssn VARCHAR(11),
    address VARCHAR(200),
    name VARCHAR(75) NOT NULL,
    gender VARCHAR(50) NOT NULL,
    PRIMARY KEY (patient_id)
);

CREATE TABLE IF NOT EXISTS insurance_providers (
    provider_id SERIAL,
    insurance_name VARCHAR(75) NOT NULL,
    policy_number VARCHAR(20) NOT NULL,
    in_network BOOLEAN NOT NULL,
    PRIMARY KEY (provider_id)
);

CREATE TABLE IF NOT EXISTS employees (
    emp_id SERIAL,
    name VARCHAR(75) NOT NULL,
    birthday DATE NOT NULL,
    salary int NOT NULL CHECK (salary >= 0),
    ssn VARCHAR(11) NOT NULL,
    role VARCHAR(50) NOT NULL,
    email VARCHAR(255),
    phone_number VARCHAR(50) NOT NULL,
    dea_number VARCHAR(9),
    medical_license_number VARCHAR(10),
    address VARCHAR(200) NOT NULL,
    gender VARCHAR(50) NOT NULL,
    PRIMARY KEY (emp_id)
);

CREATE TABLE IF NOT EXISTS prescriptions (
    prescription_id SERIAL,
```

```
    emp_id INT NOT NULL CHECK (emp_id >= 0),
    patient_id INT NOT NULL CHECK (patient_id >= 0),
    drug_name VARCHAR(100) NOT NULL,
    quantity INT NOT NULL CHECK (quantity >= 0),
    dose VARCHAR(50) NOT NULL,
    refills INT NOT NULL CHECK (refills >= 0),
    instructions TEXT,
    prescription_date TIMESTAMPTZ NOT NULL,
    pharmacy_address VARCHAR(150) NOT NULL,
    PRIMARY KEY (prescription_id),
    FOREIGN KEY (emp_id) REFERENCES employees(emp_id),
    FOREIGN KEY (patient_id) REFERENCES patients(patient_id),
    FOREIGN KEY (pharmacy_address) REFERENCES
pharmacies(pharmacy_address)
);

CREATE TABLE IF NOT EXISTS relatives (
    relative_id SERIAL,
    patient_id INT NOT NULL CHECK (patient_id >= 0),
    relative_type VARCHAR(30) NOT NULL,
    additional_notes TEXT,
    PRIMARY KEY (relative_id),
    FOREIGN KEY (patient_id) REFERENCES patients(patient_id) ON DELETE
CASCADE
);

CREATE TABLE IF NOT EXISTS referrable_doctors (
    ref_doctor_id SERIAL,
    name VARCHAR(75) NOT NULL,
    specialization VARCHAR(100),
    phone_number VARCHAR(50),
    PRIMARY KEY (ref_doctor_id)
);

CREATE TABLE IF NOT EXISTS referrals (
```

```
    ref_id SERIAL,
    emp_id INT NOT NULL CHECK (emp_id >= 0),
    ref_doctor_id INT NOT NULL CHECK (ref_doctor_id >= 0),
    patient_id INT NOT NULL CHECK (patient_id >= 0),
    PRIMARY KEY (ref_id),
    FOREIGN KEY (emp_id) REFERENCES employees(emp_id),
    FOREIGN KEY (ref_doctor_id) REFERENCES
referrable_doctors(ref_doctor_id),
    FOREIGN KEY (patient_id) REFERENCES patients(patient_id)
);


CREATE TABLE IF NOT EXISTS immunizations (
    immunization_id SERIAL,
    immunization_type VARCHAR(50) NOT NULL,
    PRIMARY KEY (immunization_id)
);

CREATE TABLE IF NOT EXISTS immunized_patients (
    immun_id INT NOT NULL CHECK (immun_id >= 0),
    patient_id INT NOT NULL CHECK (patient_id >= 0),
    PRIMARY KEY (immun_id, patient_id),
    FOREIGN KEY (immun_id) REFERENCES immunizations(immunization_id),
    FOREIGN KEY (patient_id) REFERENCES patients(patient_id)
);

CREATE TABLE IF NOT EXISTS medical_conditions (
    icd_code VARCHAR(7) NOT NULL,
    name VARCHAR(255) NOT NULL,
    parent_code VARCHAR(7),
    is_code BOOLEAN DEFAULT False,
    PRIMARY KEY (icd_code),
    FOREIGN KEY (parent_code) REFERENCES medical_conditions(icd_code)
);
```

```
CREATE TABLE IF NOT EXISTS appointments (
   app_id SERIAL,
   patient_id int NOT NULL,
   room_number VARCHAR(6),
   date TIMESTAMP NOT NULL,
   blood_pressure VARCHAR(7) NOT NULL,
   weight real NOT NULL,
   height NUMERIC NOT NULL,
   temperature NUMERIC NOT NULL,
   notes TEXT,
   PRIMARY KEY (app_id),
   FOREIGN KEY (patient_id) REFERENCES patients(patient_id)
);

CREATE TABLE IF NOT EXISTS tests (
   test_id SERIAL,
   test_name VARCHAR(255) NOT NULL,
   PRIMARY KEY (test_id)
);

CREATE TABLE IF NOT EXISTS archived_files (
   file_id SERIAL,
   patient_id INT,
   emp_id INT,
   file_name VARCHAR(255) NOT NULL,
   s3_id VARCHAR(255) NOT NULL,
   PRIMARY KEY (file_id),
   FOREIGN KEY (patient_id) REFERENCES patients(patient_id),
   FOREIGN KEY (emp_id) REFERENCES employees(emp_id)
);


CREATE TABLE IF NOT EXISTS exams (
   exam_id SERIAL,
```

```
    app_id INT NOT NULL,
    comment TEXT,
    PRIMARY KEY (exam_id),
    FOREIGN KEY (app_id) REFERENCES appointments(app_id)
);

CREATE TABLE IF NOT EXISTS lab_reports (
    report_id SERIAL,
    icd_code VARCHAR(7) NOT NULL,
    file_id INT,
    app_id INT,
    exam_id INT,
    result_info TEXT,
    PRIMARY KEY (report_id),
    FOREIGN KEY (app_id) REFERENCES appointments(app_id),
    FOREIGN KEY (file_id) REFERENCES archived_files(file_id),
    FOREIGN KEY (icd_code) REFERENCES medical_conditions(icd_code),
    FOREIGN KEY (exam_id) REFERENCES exams(exam_id)
);

CREATE TABLE IF NOT EXISTS blood_exams (
    exam_id INT,
    blood_type VARCHAR(3) NOT NULL,
    blood_sugar VARCHAR(12) NOT NULL,
    PRIMARY KEY (exam_id),
    FOREIGN KEY (exam_id) REFERENCES exams(exam_id)
);

CREATE TABLE IF NOT EXISTS covid_exams (
    exam_id INT,
    test_type VARCHAR(20) NOT NULL,
    is_positive BOOLEAN,
    PRIMARY KEY (exam_id),
    FOREIGN KEY (exam_id) REFERENCES exams(exam_id)
);
```

```
CREATE TABLE IF NOT EXISTS administered_vaccines (
   exam_id INT,
   vaccine_type VARCHAR(50) NOT NULL,
   PRIMARY KEY (exam_id),
   FOREIGN KEY (exam_id) REFERENCES exams(exam_id)
);


CREATE TABLE IF NOT EXISTS specialized_labs (
   lab_id SERIAL,
   phone_number VARCHAR(50),
   address VARCHAR(200),
   lab_name VARCHAR(200),
   PRIMARY KEY (lab_id)
);

CREATE TABLE IF NOT EXISTS relative_conditions (
   relative_id INT,
   icd_code VARCHAR(7) NOT NULL,
   PRIMARY KEY (relative_id, icd_code),
   FOREIGN KEY (relative_id) REFERENCES relatives(relative_id) ON
DELETE CASCADE,
   FOREIGN KEY (icd_code) REFERENCES medical_conditions(icd_code)
);

CREATE TABLE IF NOT EXISTS diagnoses (
   emp_id INT,
   patient_id INT,
   app_id INT,
   icd_code VARCHAR(7),
   comment TEXT,
   PRIMARY KEY (emp_id, patient_id, app_id, icd_code),
   FOREIGN KEY (emp_id) REFERENCES employees(emp_id),
   FOREIGN KEY (patient_id) REFERENCES patients(patient_id),
```

```
    FOREIGN KEY (app_id) REFERENCES appointments(app_id),
    FOREIGN KEY (icd_code) REFERENCES medical_conditions(icd_code)
);

CREATE TABLE IF NOT EXISTS appointment_medical_conditions (
    app_id INT,
    icd_code VARCHAR(7),
    comment TEXT,
    PRIMARY KEY (app_id, icd_code),
    FOREIGN KEY (app_id) REFERENCES appointments(app_id),
    FOREIGN KEY (icd_code) REFERENCES medical_conditions(icd_code)
);

CREATE TABLE IF NOT EXISTS report_creators (
    report_id   INT,
    lab_id  INT,
    PRIMARY KEY (report_id, lab_id),
    FOREIGN KEY (report_id) REFERENCES lab_reports(report_id),
    FOREIGN KEY (lab_id) REFERENCES specialized_labs(lab_id)
);

CREATE TABLE IF NOT EXISTS insurance_covers (
    provider_id INT,
    patient_id  INT,
    member_id   VARCHAR(12) NOT NULL,
    group_number    VARCHAR(12) NOT NULL,
    policy_holder_name  VARCHAR(75) NOT NULL,
    PRIMARY KEY (provider_id, patient_id),
    FOREIGN KEY (provider_id) REFERENCES
insurance_providers(provider_id),
    FOREIGN KEY (patient_id) REFERENCES patients(patient_id)
);

CREATE TABLE IF NOT EXISTS immunized_employees (
    immun_id    INT,
```

```
   emp_id  INT,
   PRIMARY KEY (immun_id, emp_id),
   FOREIGN KEY (emp_id) REFERENCES employees(emp_id),
   FOREIGN KEY (immun_id) REFERENCES immunizations(immunization_id)
);

CREATE TABLE IF NOT EXISTS accepted_tests (
   test_id INT,
   lab_id  INT,
   PRIMARY KEY (test_id, lab_id),
   FOREIGN KEY (test_id) REFERENCES tests(test_id),
   FOREIGN KEY (lab_id) REFERENCES specialized_labs(lab_id)
);

CREATE TABLE IF NOT EXISTS appointment_employees (
   emp_id  INT,
   app_id  INT,
   PRIMARY KEY (emp_id, app_id),
   FOREIGN KEY (emp_id) REFERENCES employees(emp_id),
   FOREIGN KEY (app_id) REFERENCES appointments(app_id)
);

CREATE TABLE IF NOT EXISTS emergency_contacts (
 name VARCHAR(75) NOT NULL,
 patient_id INT NOT NULL CHECK (patient_id >= 0),
 phone_1 VARCHAR(50) NOT NULL,
 phone_2 VARCHAR(50),
 PRIMARY KEY (name, patient_id),
 FOREIGN KEY (patient_id) REFERENCES patients(patient_id)
);

CREATE INDEX idx_patient_name ON patients USING HASH (name);
CREATE INDEX idx_appointment_date ON appointments (date DESC);
CREATE INDEX appointment_patient_id ON appointments USING HASH
(patient_id);
```

```
CREATE INDEX prescription_drug_name ON prescriptions USING HASH
(drug_name);
COMMIT;
```