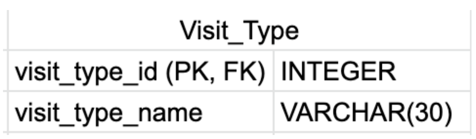
The data model is as follows:
There is a users table that keeps track of all the generic user info, such as username, first_name, last_name, address, etc. The permission_group and gender_id are foreign keys to the permissions_group table and gender table, respectively.

Users	
user_id (PK)	INTEGER
username	VARCHAR(150)
password	VARCHAR(30)
first_name	VARCHAR(55)
last_name	VARCHAR(55)
date_of_birth	DATE
address	VARCHAR(255)
city	VARCHAR(55)
state	CHAR(2)
zip	INTEGER
email	VARCHAR(150)
permission_group_id (FK)	INTEGER
gender_id (FK)	INTEGER

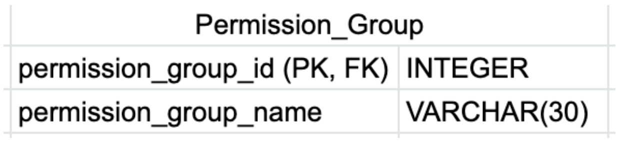
Then there is a Medical_Visits table that tracks what happened at each visit, such as who the doctor was, when the visit occurred, blood pressure, etc.
It uses a medical visit_id as the Primary Key, and it also has foreign keys in patient_id and doctor_id that both map to the Users Table to get the information from there. There is also a visit_type that maps to a Visit_Type Table to determine if it is a Diagnostic or Treatment visit.

Medical_Visits_Info	
medical_visit_id	BIGINTEGER
patient_id (FK to user_id_in Users Table)	INTEGER
date_of_visit	TIMESTAMP
visit_type (FK)	INTEGER
height	INTEGER
weight	FLOAT
blood_pressure	INTEGER
Notes	TEXT
doctor_id (FK to user_id in Users Table)	INTEGER

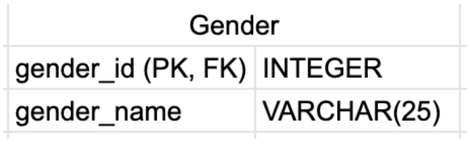
There are also tables for Visit_Type, Permission Group and Gender, which are just enumerations for certain group names. They are listed below:



Visit_Type	
visit_type_id (PK, FK)	INTEGER
visit_type_name	VARCHAR(30)



Permission_Group	
permission_group_id (PK, FK)	INTEGER
permission_group_name	VARCHAR(30)



Gender	
gender_id (PK, FK)	INTEGER
gender_name	VARCHAR(25)

This program is a simple proof of concept, as it is only three classes and does not have a full on database or server class. To run the program, you simply go to the command line and type python3 MedicalInformationService.py and then you list the commands and their parameters via the command. The first parameter after the filename is which user is logged. It is used to initialize the MedicalInformationService. For example, to run simulate logging in user_id=1 and running get_personal_info on user_id=1, python3 MedicalInformationService.py 1 get_personal_info 1 where get_personal_history is the command and 1 is the user_id parameter you wish to select.

For running multiple commands at once, you simply write multiple statements side by side. For example, python3 MedicalInformationService.py 1 get_personal_info 1 get_last_12_months_visits 1

To run an edit command such as edit_personal_history you write the command as usual, but you use the '|' character to separate one command from the next. You use the pipe character to separate commands because there is not a set amount of keywords to update at a time. Here is an example:

```
Python3 MedicalInformationService.py 2 get_personal_info 2 edit_personal_history 2 last_name Johansson first_name Erik |
get_personal_info 2
```

Thus, you separate the commands with a '|' character, or if it is the last statement in the group, you omit the pipe, as shown below

```
Python3 MedicalInformationService.py get_personal_info 2 edit_personal_history 2 last_name Johansson first_name Erik
```

The following are acceptable commands and their parameters:

MedicalInformationService (user, user_table=user_table, visit_table=visit_table, permissions_group_table=permissions_group_table, gender_table=gender_table)
user = the user id we are trying to log in as.
get_weight(user)
user = the user id we are trying to get the weight
Returns a user's most recent weight in pounds as a float
get_height(user)

user = the user id we are trying to get the height
Returns a user’s most recent height in inches as an integer

get_median_blood_pressure_last_12_months(user)
user = the user id we are trying to get the median blood pressure
Returns the user’s median blood pressure based on the last 12 months of visits as float

get_avg_blood_pressure_last_12_months(user)
user = the user id we are trying to get the average blood pressure
Returns the user’s avg blood pressure based on the last 12 months of visits as float

get_min_blood_pressure_last_12_months(user)
user = the user id we are trying to get the min blood pressure
Returns the user’s minimum blood pressure based on the last 12 months of visits as integer

get_max_blood_pressure_last_12_months(user)
user = the user id we are trying to get the max blood pressure
Returns the user’s max blood pressure based on the last 12 months of visits as integer

get_last_12_months_visit(user)
user = the user id we are trying to get the last 12 months visit history
Returns a list of the user’s visits from the last 12 months

get_medical_history(user)
user = the user id we are trying to get the medical history
Returns a list of all of a user’s visits

get_personal_info(user)
user = the user id we are trying to get the personal info
Returns a dictionary of a user’s personal data

get_current_BMI(user)
user = the user id we are trying to get the BMI
Returns a user’s BMI as a float

*edit_medical_history(user, visit, **kwargs)*
user = the user id we are trying to edit the medical history
visit = visit id we are trying to edit
***kwargs = a list of keyword arguments being passed*
Edits a user’s medical visit

*edit_personal_info(user, **kwargs)*
user = the user id we are trying to edit the personal history
***kwargs = a list of keyword arguments being passed*
Edits a user’s personal information

run_tests()
Runs all unit and functional tests