JACK CLARKE (300080674),
AMINE BABA (300145121),
VICTOR BABINEAU (300010115),
CARSON BOYNE (300207779),
JATHUSHAN KARTHIGESAR (300060617)
Group 18

University of Ottawa
Faculty of Engineering

Department of EECS

CSI2132-Databases

Database Report

Winter 2022

Submitted to: Olubisi Runsewe

Date of submission: April 15th, 2022

1. Technologies Used

The technologies used (e.g., DBMS & programming language) used for the implementation of your application.

For this project, we decided to use **PostgreSQL** as a DBMS, **Django** as our web application framework and **Heroku** to host our website on the cloud.

PostgreSQL is an open-source object relational database management system. PostgreSQL extends SQL to allow a plethora of features for the developer to use. The most notable ones are the ability to provide automatically updatable views, meaning that a view resulting from a query will automatically update on data update.

The group chose PostgreSQL because it is familiar to group members.

PostgreSQL also allows the use of triggers which execute code in response to changes to a table or to a view. This DBMS also allows the use of stored procedure. Procedures are stored in the database and provide applications access to common data manipulation, data validation, access control and other methods. Procedures are defined and can be run by users.

Django is a free open-source web application framework written in Python. A framework is a collection of modules that makes development easier and allows developers to create code from existing code not from scratch.

That way, website components such as comment boxes, forms, authentication support, management panels, upload support and more are already built by the framework. Hence, using a framework allows a website to have multiple functionalities without having to code them all from scratch. The developer would only need to configure them properly in order to match their website.

The group chose Django because it seemed like the most efficient way to get the app up. It also uses python, which is extremely easy.

Heroku is a platform as a service (PaaS) that enables developers to build, run, and operate applications entirely on the cloud. The group chose Heroku because it is free.

Heroku has support for modern programming languages such as Node, Ruby, Go, Java, Scala, and Python. It supports continuous integration (CI) of code by allowing applications to be deployed from Git, and Docker with the use of an API.

2. DDL

2.1. Schema

The relational schema of this database has changed from its first iteration. For one, all IDs have been merged into a singular person_id which is the person's Social Security Number (SSN). Every employee, manager, hygienist, dentist, and patient are a Person meaning the merge was seamless and logical to do. This eased the querying of the database.

Additionally, a procedure table with all four available procedures was added. Since the type of procedures offered is not expected to change often, it makes sense to have them as a pre-made, less accessible table.

Person (person_id, b_date, f_name, l_name, city, house, street, postal_code, province, email, gender, phone-number, caregiver_id)
Patient (patient_id, insurance)
Patient_Records (record_id, patient _id, employee_id, employee_notes)
Employee (employee_id, salary)
Dentist (employee_id, branch_id)
Receptionist (employee_id, branch_id)
Manager (employee_id)
Hygienist (employee_id)
User (username, password)
Branch (branch_id, city, manager_id)
Appointment (appointment_id, patient_id, dentist_id, date, start_time, end_time, appointment_type, status, room_assignment)
Appointment_Procedure (appointment_proc_id, appointment_id, procedure_code, procedure_type, date, invoice_id, tooth_involved, amount_of_procedure, patient_charge, insurance_charge, total_charge, insurance_claim_id)
Treatment (treatment_id, patient_id, treatment_type, appointment_id, treatment_details, patient_records)
Fee_Charge (fee_id, invoice_id, fee_code, charge)
Invoice (invoice_id, patient_id, date_of_issue)
Payment (payment_id, invoice_id, patient_charge, insurance_charge, total_charge, payment_type)
Insurance_Claim (appointment_proc, payment_id)
Review (review_id, professionalism, communication, cleanliness, value, patient_id, branch_id)
Procedure (procedure_code, procedure_type)

2.2. Constraints

The constraints have not changed for this second part of the project. Some cascading logic was added to the foreign keys of each table. On deletion or update of a person all patient, employee, dentist, hygienist,

receptionist, and manager entries referencing that person's person_id will be deleted or updated. All these can be seen in the Main.sql file provided on the repository.

3. Accessing the application

The website is fully deployed and is accessible at this link: https://dentist-app7.herokuapp.com/. As is stands, the stack used is intended to be hosted using Heroku and not on a self-hosted server. The 6 queries are split across the patient, dentist and receptionist apps.

If one wants to install the project, he needs to follow these instructions also available on github or in the README.md:

Installation Instructions:

Part A: setup database

1. setup a postgres server somewhere
2. create a database
3. run ./Main.sql on the schema, to create the database (if something goes wrong, you can use ./drop_main_tables.sql)
4. run ./dentistapp/gen_fake_data.sql
5. get the connection string for the database

Part B: setup app

1. *cd ./dentistapp*, from here onwards dentistapp is the root folder
2. create a venv. Run: *python3 -m venv ./.venv*
3. launch the venv *source ./.venv/bin/activate*
4. install dependencies: *pip install -r ./requirements.txt*
5. clone dentistapp/.env.example to dentistapp/.env *cp dentistapp/.env.example* to *dentistapp/.env*
6. change the string in dentistapp/.env to the database url (postgresql:// or postgres://)
7. run *gunicorn dentistapp.wsgi* or *python3 manage.py runserver*
8. Ip should display, and a message "connected to database" should display

.env example

DATABASE_URL=postgresql://postgres:Yipyapyop1@localhost:5432

4. Implementation Details

This section has links, these links should work when we make the project public on April 15th. If they do not, please email the GitHub owner at jclar16@uottawa.ca .

It is also important to note that SSN and ID are used interchangeably, they mean the same thing in the context of this project. For part 1 we had them separate for privacy and for part 2 we combined them for a simpler implementation.

The code is in the *dentistapp* folder. The URL routes can be found in dentistapp/dentistapp/urls. There is a function defined for each route in the *urls* folder. The route functions are in the views. There are 7 files. The index file contains the routes to the organizational pages like index, patient, doctor. The other six files contain the code for the queries.

"checkAppointments.py" is very straightforward, it simply queries the data, converts it to a map and sends the data to the frontend. "checkProcedures.py" is amazingly simple. It has 1 very simple line of SQL, and the rest is trivial Django rendering and returning the html to the frontend.

"createEmployee.py" is the most complicated endpoint. First, it creates a person(link). We used a person table because the instructions said an employee can also be a patient. Next, the endpoint adds the person to the employee table(link). . If the employee is a minor, they will also have had a caregiver_id, which is inserted into the database at this stage(link). After making the employee, there is a dropdown that was filled out to determine what kind of employee this employee will be. We have distinct types of employees because the role of the employee will change the relationship with the branch. After adding the data to the person and employee tables, then the id needs to be added to one of the employee type tables such as doctor, manager, hygienist or receptionist(link).

"createPatient.py" works much the same as "createEmployee.py". First it creates the person, then adds that person the patient table. Unlike an employee, a patient does not have a role so there is nothing else to add.

"createAppointement.py" is a simple insert query to add all the data to the appointment table. "showDentistsInBranch.py" is also remarkably simple. It is just a select query on all the dentists that are in a branch.