2022

Doctor’s Office Project

STACKOVERFLOW GROUP

MICHAEL BABINEC – DAVID HARBOYAN – JOHANN FERNANDEZ

SITE LINK

http://cpe-70-93-194-149.natsow.res.rr.com./index.php

Table of Contents

TABLE DEFINITION

Our database included a total of six tables; DOCTOR, PATIENT, PRESCRIPTION, APPOINTMENT, MEDICAL_TEST, and AUDIT.

DOCTOR:

```
-- Create DOCTOR
DROP TABLE IF EXISTS DOCTOR;
CREATE TABLE DOCTOR (
    Doctor_ID   varchar(6)  not null,
    Last_Name   varchar(20) not null,
    First_Name  varchar(20) not null,
    Phone_Number    char(10) not null,
    Specialty   varchar(30) null,
    Salary  decimal(10,2),
    primary key (Doctor_ID)
);
```

Our DOCTOR table includes a value for Doctor ID, the Doctor's name, phone number, specialty, and salary. Doctor ID was the primary key, and this table had no foreign keys.

PATIENT:

```
-- CREATE PATIENT
DROP TABLE IF EXISTS PATIENT;
CREATE TABLE PATIENT (
    SSN char(9) not null,
    Last_Name   varchar(20) not null,
    First_Name  varchar(20) not null,
    Primary_Phone_Number    char(10)    not null,
    Street_Name varchar(30) not null,
    Street_Number   varchar(15) not null,
    City    varchar(30) not null,
    Zip_Code    char(5) not null,
    Insurance_ID    varchar(12),
    primary key (SSN)
);
```

PATIENT includes an attribute for SSN, Name, Phone Number, Street Name and Number, City, Zip Code, and the patient's Insurance ID. The primary key is SSN, there are no foreign keys.

PRESCRIPTION:

```sql
-- CREATE PRESCRIPTION
DROP TABLE IF EXISTS PRESCRIPTION;
CREATE TABLE PRESCRIPTION (
    Prescription_ID varchar(10) not null,
    Drug_Name   varchar(25) not null,
    Dosage  varchar(8) not null,
    Number_of_refills   int not null,
    Appointment_num varchar(12) not null,
    Most_Recent_Filling date,
    Prescribed_by   varchar(6)  not null,
    Patient_SSN char(9) not null,
    primary key (Prescription_ID),
    foreign key (Prescribed_by) references DOCTOR (Doctor_ID),
    foreign key (Patient_SSN) references PATIENT (SSN)
);
```

PRESCRIPTION has values for the Prescription ID, Drug Name, Dosage, Refills the Patient has, the Appointment number, the date of the most recent filling of the prescription, the Doctor ID of the issuing physician, and the SSN of the patient to which the prescription is being issued to. The primary key is the Prescription ID. Prescribed_by and Pantient_SSN are foreign keys, as the former identifies the Doctor who issued the prescription by their Doctor ID, and the latter identifies patient through their ssn. The final foreign key is Appointment_num, which will reference the appointment number received from the APPOINTMENT table. This key is appended after the creation of appointment

APPOINTMENT:

```sql
-- CREATE APPOINTMENT
DROP TABLE IF EXISTS APPOINTMENT;
CREATE TABLE APPOINTMENT (
    Appointment_Number  varchar(12) not null,
    Patient_SSN char(9) not null,
    Doctor_ID   varchar(6) not null,
    Appointment_Time time   not null,
    Appointment_Date    date    not null,
    Room_number int not null,
    primary key (Appointment_Number),
    foreign key (Patient_SSN) references PATIENT (SSN),
    foreign key (Doctor_ID) references DOCTOR (Doctor_ID)
);

ALTER TABLE PRESCRIPTION
ADD foreign key (Appointment_num) references APPOINTMENT (Appointment_Number);
```

APPOINTMENT has an attribute for Appointment Number, Patient SSN, Doctor ID, Appointment Time & Date, and Room Number. Appointment Number is the Primary Key, and as they are the identifying attributes for Patients and Doctors, Patient SSN and Doctor ID are foreign keys from the PATIENT and DOCTOR table respectively. Included is the altering of our PRESCRIPTION table to properly reference the appointment number of the APPOINTMENT table.

MEDICAL_TEST:

```sql
-- CREATE MEDICAL_TEST
DROP TABLE IF EXISTS MEDICAL_TEST;
CREATE TABLE MEDICAL_TEST (
    Test_ID varchar(6)  not null,
    Doctor_Name varchar(20) not null,
    Doctor_ID   varchar(6)  not null,
    Test_Type   varchar(25) not null,
    Result  varchar(12) not null,
    Patient_SSN char(9) not null,
    Appointment_num varchar(12) not null,
    primary key (Test_ID),
    foreign key (Doctor_ID) references DOCTOR (Doctor_ID),
    foreign key (Patient_SSN) references PATIENT (SSN),
    foreign key (Appointment_num) references APPOINTMENT (Appointment_Number)
);
```

MEDICAL TEST includes values for Test ID, Doctor Name and ID, Test Type, Test Result, Patient SSN, and Appointment Number. Our Primary key is Test ID. Doctor Name and Doctor ID are foreign keys from the DOCTOR table, used to record the identity of the Doctor issuing the test. Patient SSN is a foreign key from the PATIENT table, identifying the patient undergoing tests. Appointment Number is from APPOINTMENT, and documents the Appointment in which the tests were issued.

AUDIT:

```sql
-- CREATE AUDIT
DROP TABLE IF EXISTS AUDIT;
CREATE TABLE AUDIT (
    Doctor_ID   varchar(6)  not null,
    Doctor_Name varchar(20) not null,
    Audit_Action    varchar(15) not null,
    Specialty varchar(25),
    Date_Modified   date    not null,
    foreign key (Doctor_ID) references DOCTOR (Doctor_ID)
);
```

AUDIT has an attribute for Doctor ID, Doctor name, the action preformed in the audit, the doctor's Speciality, and the date of the audit. The only foreign key is Doctor ID, the identifying attribute of a doctor from the DOCTOR table.

LIST OF FEATURES

Deliverable 2:

```sql
-- Deliverable 2
DROP VIEW IF EXISTS Roberts_Patient;
CREATE VIEW Roberts_Patient AS SELECT PATIENT.First_Name, PATIENT.Last_Name, PATIENT.Primary_Phone_Number
                                FROM (SELECT Patient_SSN
                                        FROM APPOINTMENT
                                        WHERE Doctor_ID = 'RO1000') AS SOCIAL
                                INNER JOIN PATIENT
                                ON SOCIAL.Patient_SSN = PATIENT.SSN;
```

This view shows all the names and phone numbers of Dr. Robert Stevens using his Doctor_ID, a unique identifier.

Deliverable 3:

```sql
-- Deliverable 3
DROP VIEW IF EXISTS Medication;
CREATE VIEW Medication AS SELECT DR.First_Name, DR.Last_Name FROM DOCTOR DR, PRESCRIPTION PR
WHERE PR.Prescribed_by = DR.Doctor_ID AND PR.Drug_Name = 'Vicodin';
```

With this SQL statement, we are able to view the full names of doctors who have prescribed Vicodin.

Deliverable 4:

```sql
-- Deliverable 4
DROP VIEW IF EXISTS Specialty;
CREATE VIEW Specialty AS SELECT DR.First_Name, DR.Last_Name, DR.Specialty FROM DOCTOR DR WHERE DR.Specialty IS NOT NULL;
```

This shows the full names of all doctors in the table that have a specialty.

Deliverable 5:

```sql
-- Deliverable 5
ALTER VIEW Specialty AS SELECT DR.First_Name, DR.Last_Name, DR.Specialty FROM DOCTOR DR;
```

This is a modification of the previous view, now including any doctors who do not have specialties.

Deliverable 6:

```
-- Deliverable 6
DROP TRIGGER IF EXISTS AuditSpecialty_UPDATE;
CREATE TRIGGER AuditSpecialty_UPDATE AFTER UPDATE ON DOCTOR for each row
    INSERT INTO AUDIT (Doctor_ID, Doctor_Name, Audit_Action, Specialty, Date_Modified)
        VALUE( NEW.Doctor_ID, NEW.First_Name, 'Update' , NEW.Specialty, curdate());

DROP TRIGGER IF EXISTS AuditSpecialty_INSERT;
CREATE TRIGGER AuditSpecialty_INSERT AFTER INSERT ON DOCTOR for each row
    INSERT INTO AUDIT (Doctor_ID, Doctor_Name, Audit_Action, Specialty, Date_Modified)
        VALUE( NEW.Doctor_ID, NEW.First_Name, 'Insert' , NEW.Specialty, curdate());

```

Every time that a doctor’s specialty is updated or added to, one of these triggers will activate. These triggers both create a new entry in the audit table as a result of the new information provided in the DOCTOR table.

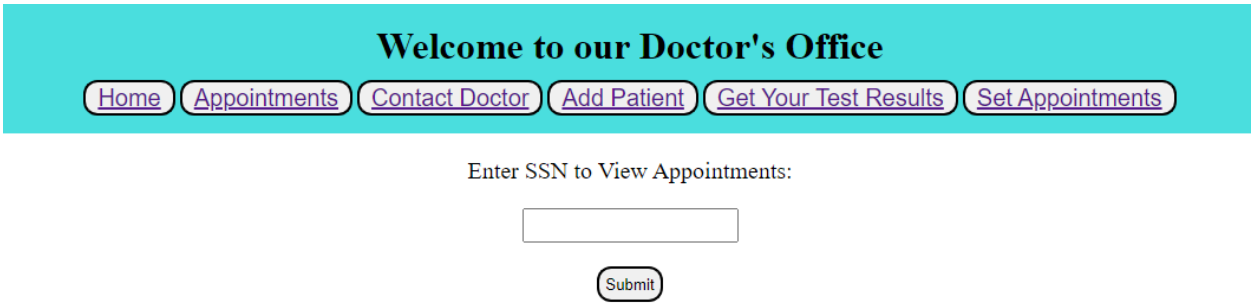
Deliverable 7:

```sql
-- Deliverable 7
DROP TABLE IF EXISTS DOCTOR_BACKUP;
CREATE TABLE DOCTOR_BACKUP LIKE DOCTOR;
INSERT INTO DOCTOR_BACKUP
SELECT* FROM DOCTOR;

DROP TABLE IF EXISTS PATIENT_BACKUP;
CREATE TABLE PATIENT_BACKUP LIKE PATIENT;
INSERT INTO PATIENT_BACKUP
SELECT* FROM PATIENT;

DROP TABLE IF EXISTS PRESCRIPTION_BACKUP;
CREATE TABLE PRESCRIPTION_BACKUP LIKE PRESCRIPTION;
INSERT INTO PRESCRIPTION_BACKUP
SELECT* FROM PRESCRIPTION;

DROP TABLE IF EXISTS APPOINTMENT_BACKUP;
CREATE TABLE APPOINTMENT_BACKUP LIKE APPOINTMENT;
INSERT INTO APPOINTMENT_BACKUP
SELECT* FROM APPOINTMENT;

DROP TABLE IF EXISTS MEDICAL_TEST_BACKUP;
CREATE TABLE MEDICAL_TEST_BACKUP LIKE MEDICAL_TEST;
INSERT INTO MEDICAL_TEST_BACKUP
SELECT* FROM MEDICAL_TEST;

DROP TABLE IF EXISTS AUDIT_BACKUP;
CREATE TABLE AUDIT_BACKUP LIKE AUDIT;
INSERT INTO AUDIT_BACKUP
SELECT* FROM AUDIT;
```

These six tables are just backups of the current tables within the database. Each active table is copied into a backup table, and the data can safely be stored from there.

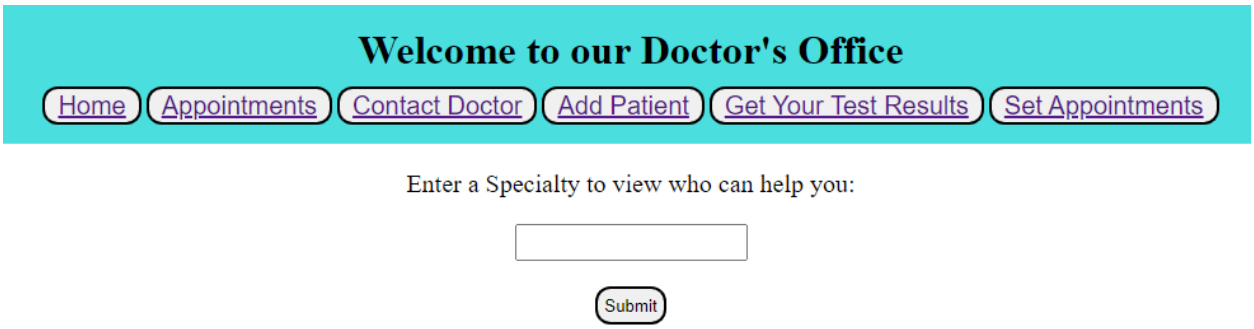
IMPLEMENTED FUNCTIONALITIES & USER GUIDE

Appointments:



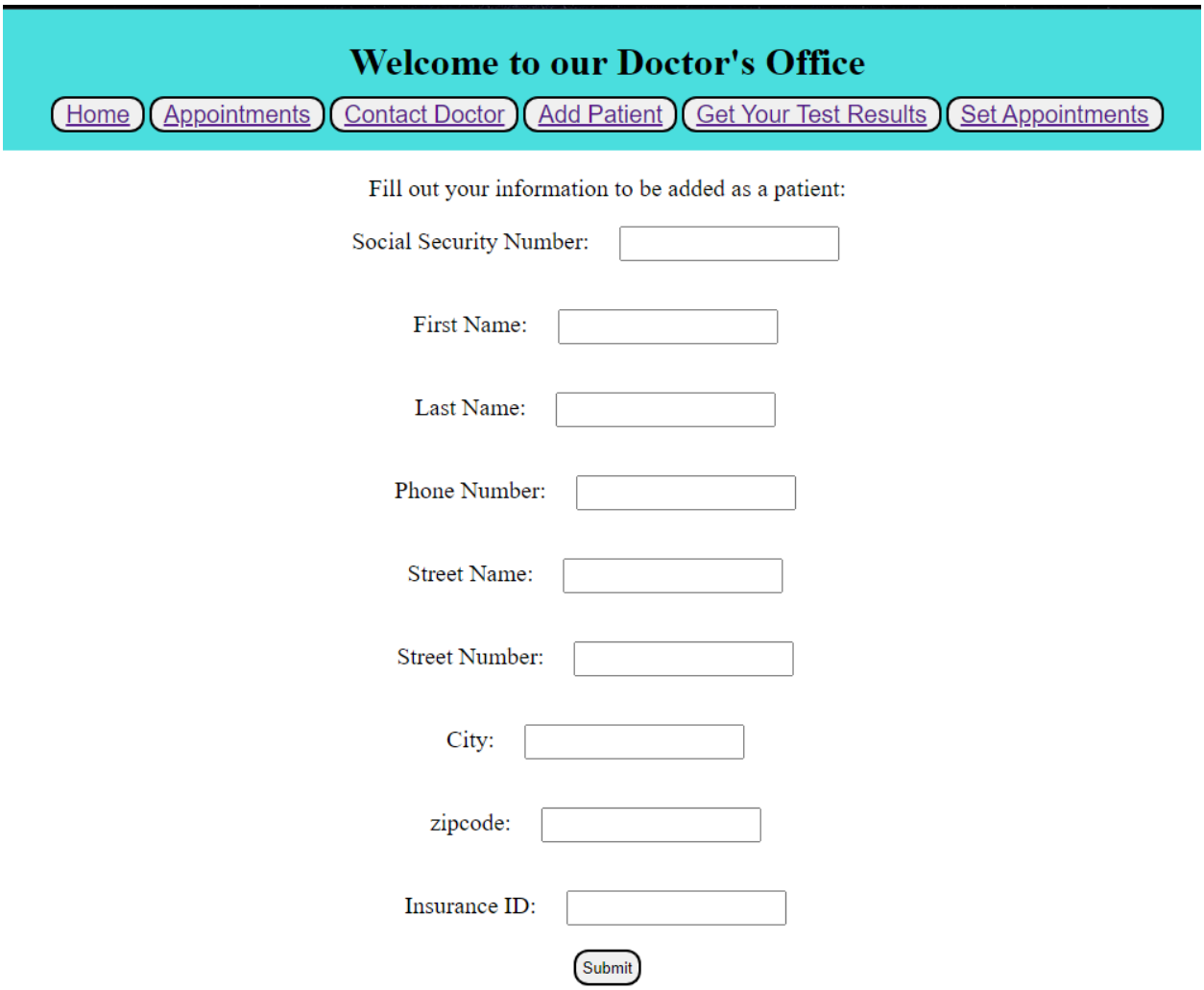
The user is able to input an SSN value and receive a table of the Appointments on record for that Patient SSN.

Contact Doctor:



This field allows users to ‘search’ for a doctor who specializes in the searched field. “Orthopedic” would return the names of any doctors on staff who specialize in Orthopedic Medicine.

Add Patient:



This function is used for the creation of entirely new patient profiles, or PATIENT tuples. By providing each attribute required for a patient; SSN, Name, Phone Number, Address, and Insurance ID, an entirely new patient is created within the database.

Get Your Test Results:

This field allows a patient to view the results of any test they have ever received through the doctor's office. Inputting in a patient's SSN will display each test, the type of test, and the results.

Set Appointments:

This gives the user the ability to create a new appointment to be added to the APPOINTMENT table. By inputting a value for Appointment Number, Patient SSN, Doctor ID, and Appointment Time & Date; one can create an appointment with a specific doctor, at a specific time.

CONCLUSION

We were able to construct a useable database for a Doctor's Office. All the deliverables were satisfied, and our added functionalities give hypothetical users the ability to interact with the database in new ways, such as setting appointments or searching for the ideal doctor for their situation. We also successfully hosted this application, allowing for connection through a link, rather than a localhost server.

With more time, our project could be better streamlined for actual, practical, use. We are missing some error throws when a user inputs invalid or impossible data. In the actual hands of an average user, there would be the possibility of various errors when considering things such as whether or not a Doctor already has an appointment at a certain time when trying to set a new one. Issues of this type could be implemented against, given more time. Personally, I also learned much about PHP and html during this project, and would enjoy the chance to modernize our site, making it appear less like a class project, and more like a professional site.