Image source: http://studenthealth.uncc.edu/medical-services

Student Health Care System

Project for the class of Applied Databases ITIS 6120

Submitted by,
Akshay Adagale 800987050
Bhavya Joshi 800985776
Rohan Khairnar 800971126

1. Summary

We decide on designing a database system for Student Clinic in a University offering primary health care services to current students and faculties. We have chosen the University of North Carolina, Charlotte as our model University. For this project, we have planned to modulate a part of scope of already existing health care resources at UNC Charlotte.

2. Scope

As per our system design, our health care project features services including primary Medical Care, HIV Testing, Sports Medicine, Nurse Clinic, Digital Laboratory, Immunizations, Travel Medicine, Allergy Injections, Psychiatric Care and Physical Therapy. The entire unit comprises of departments of health care featuring doctors and services provided, management and maintenance featuring the daily check and resource availability factors. To be able to store and maintain data in normalized form, we have maintained tables for *users*, which is then being referenced by *patient* and *doctor* classes. To enable check on patient's insurance availability, we have created an *insurance* class. This class is correlated with *patient_insurance_relation class* that references the patient's ID from *patient* table to retrieve information regarding the type and validity of insurance. We are maintaining a table for services provided called the *services* table, a basic chart that will contain each service with dedicated doctor's ID. Every patient will be able to book an appointment, and on doing so, will be listed in *appointments* table with patient and appointment details. To maintain record of patient's visits, we have a class *visit_information* that will be updated each time the patient visits clinic with date, time, doctor and reason of visit details. As there are some standard tests to be carried out for particular symptoms, and for each patient the result will be varied, thus, we have maintained three distinct yet co-related tables of *visit_clinic_care_information* comprising of symptoms, prescriptions and suggested diagnosis, *visit_information_tests* to list down particular tests as per symptoms stated in clinic care table and *tests* table that will maintain records for all tests of each patient. Lastly, we have a billing table that adds up service charges and doctor fees thereby reflecting the final bill in *bills* table. The table will have a *status* field indicating the bill being paid or unpaid, and each time the value will be *set* as and when user makes the payment.

The tables created and structure of each with dedicated attributes is shown as follows:

Following tables created:

- usertype
    - `typeID`
    - `TypeName`

- paymenttype
    - `PaymentType_ID`
    - `Payment_Method`

- address
    - `AddressID`
    - `Address`
    - `zipcode`
    - `city`
    - `State`

- `insurance
    - `Insurance_ID`
    - `Start_date`
    - `End_date`
    - `Insurance_Name`

- `user`
    - `user_Id`
    - `user_type`
    - `address_ID`

- `doctor`
    - `ID`
    - `first_name`
    - `last_name`
    - `Speciality`
    - `Email`

- `patient`
    - `ID`

    - `first_name`
    - `last_name`
    - `InsuranceId`
    - `Email`

- `appointment`
    - `ID`
    - `patient_id`
    - `doctor_id`
    - `date`

- `prescription`
    - `Prescription_ID`
    - `Patient Name`
    - `Doctor Name`
    - `Prescription_Date`
    - `Medicine`

- `prescription_doctor_patient`
    - `Prescription_ID`
    - `Doctor_ID`
    - `Patient_id`

- `tests`
    - `ID`
    - `patient_id`
    - `doctor_id`
    - `test_type`
    - `result`

- `bill`
    - `Bill ID`
    - `Patient ID`
    - `Doctor ID`
    - `Test ID`
    - `Insurance Num`
    - `Bill Date`
    - `Total Cost`

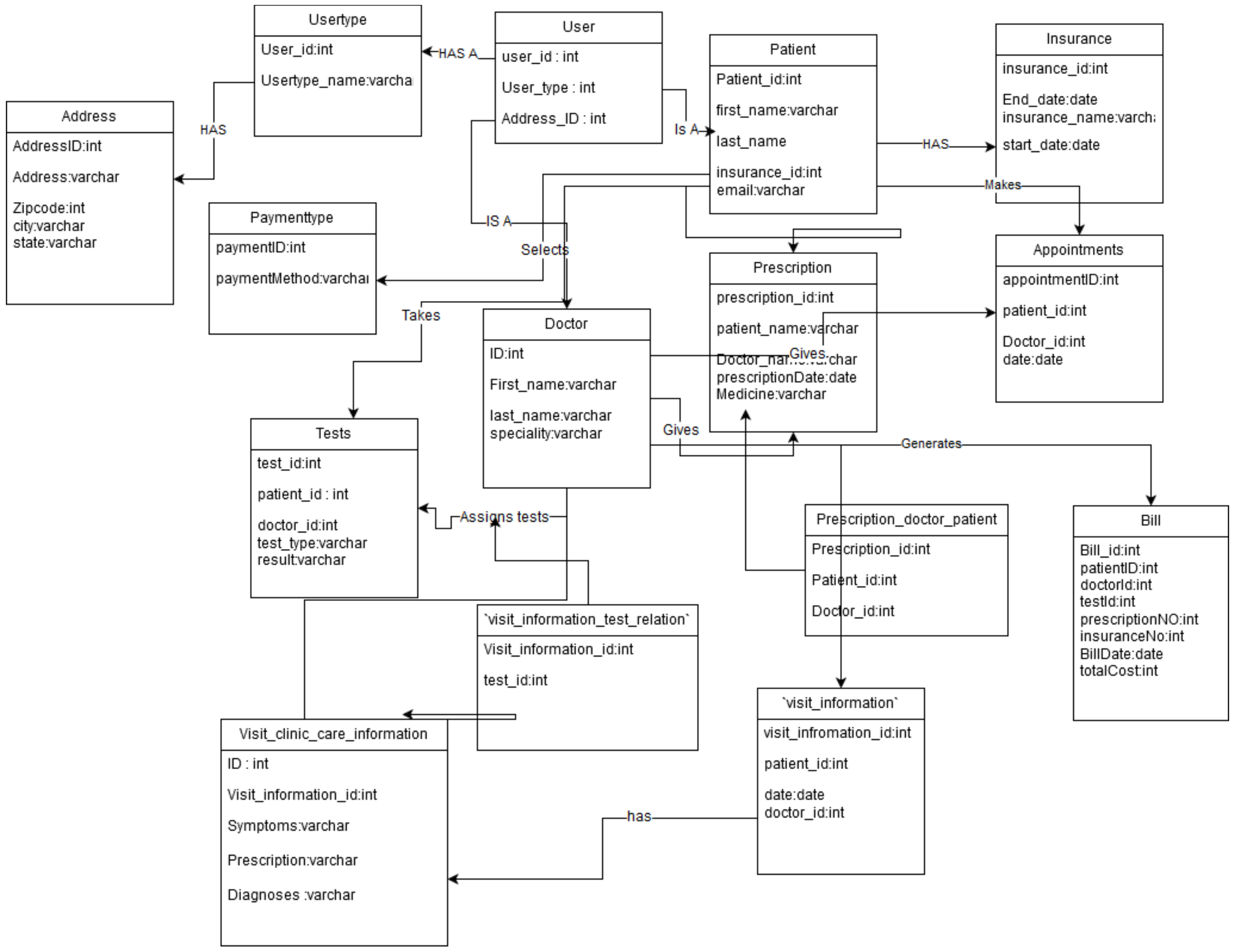
- `` `payment` ``
    - `` `Payment ID` ``
    - `` `Bill ID` ``
    - `` `Payment Type` ``
    - `` `Payment date` ``
    - `` `Total Cost` ``

- `` `visit_information` ``
    - `` `ID` ``
    - `` `Patient_ID` ``
    - `` `Date` ``
    - `` `doctor_ID` ``

- `` `visit_clinic_care_information` ``
    - `` `ID` ``
    - `` `visit_information_id` ``
    - `` `Symptoms` ``
    - `` `Prescription` ``
    - `` `diagnosis` ``

- `visit_information_test_relation`
    - `visit_information_id`
    - `test_id`

3. UML model for proposed database design:



- **SQL Scripts for creating tables as follows:**

```
CREATE TABLE `usertype` (
  `typeID` int(11) NOT NULL AUTO_INCREMENT,
  `TypeName` varchar(60) DEFAULT NULL,
  PRIMARY KEY (`typeID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `paymenttype` (
  `PaymentType_ID` int(11) not null auto_increment,
  `Payment_Method` varchar(60) DEFAULT NULL,
  primary key (`PaymentType_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `address` (
  `AddressID` int(20) NOT NULL auto_increment,
  `Address` varchar(40) NOT NULL,
  `zipcode` int(10) NOT NULL,
  `city` varchar(10) NOT NULL,
  `State` varchar(11) DEFAULT NULL,
  PRIMARY KEY (`AddressID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `insurance` (
  `Insurance_ID` int(11) NOT NULL auto_increment,
  `Start_date` date NOT NULL,
  `End_date` date NOT NULL,
  `Insurance_Name` varchar(10) NOT NULL,
  PRIMARY KEY (`Insurance_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `user` (
    `user_Id` INT(11) NOT NULL AUTO_INCREMENT,
    `user_type` INT(11) DEFAULT NULL,
    `address_ID` int(20),
    PRIMARY KEY (`user_Id`),
    KEY `fk_address_ID` (`address_ID`),
    KEY `fk_user_type` (`user_type`),
    CONSTRAINT `fk_address_ID` FOREIGN KEY (`address_ID`)
        REFERENCES `address` (`AddressID`),
    CONSTRAINT `fk_user_type` FOREIGN KEY (`user_type`)
        REFERENCES `usertype` (`typeID`)
)  ENGINE=INNODB DEFAULT CHARSET=UTF8;
```

```
CREATE TABLE `doctor` (
  `ID` int(11) NOT NULL AUTO_INCREMENT,
  `first_name` varchar(60) NOT NULL,
  `last_name` varchar(60) DEFAULT NULL,
  `Speciality` varchar(60) DEFAULT NULL,
  `Email` varchar(40) NOT NULL,
  PRIMARY KEY (`ID`),
  KEY `fk_d_ID`(`ID`),
  CONSTRAINT `fk_d_ID` FOREIGN KEY (`ID`) REFERENCES `user`(`user_Id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;


CREATE TABLE `patient` (
  `ID` int(11) NOT NULL,
  `first_name` varchar(60) NOT NULL,
  `last_name` varchar(60) DEFAULT NULL,
  `InsuranceId` int(20) DEFAULT NULL,
  `Email` varchar(40) NOT NULL,
  PRIMARY KEY (`ID`),
INDEX `fk_ID_idx`(`ID` ASC),
  KEY `fk_p_ID`(`ID`),
  CONSTRAINT `fk_p_ID` FOREIGN KEY (`ID`) REFERENCES user(`user_Id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;


CREATE TABLE `appointment` (
  `ID` int(11) NOT NULL auto_increment,
  `patient_id` int(11) DEFAULT NULL,
  `doctor_id` int(11) DEFAULT NULL,
  `date` datetime DEFAULT NULL,
  PRIMARY KEY (`ID`),
INDEX `fk_ID_idx1`(`patient_id` ASC),
  KEY `fk_p_app` (`patient_id`),
  KEY `fk_d_app` (`doctor_id`),
  CONSTRAINT `fk_d_app` FOREIGN KEY (`doctor_id`) REFERENCES `doctor` (`ID`),
  CONSTRAINT `fk_p_app` FOREIGN KEY (`patient_id`) REFERENCES `patient` (`ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;







CREATE TABLE `prescription` (
  `Prescription_ID` int(11) NOT NULL AUTO_INCREMENT,
  `Patient Name` varchar(20) DEFAULT NULL,
  `Doctor Name` varchar(60) NOT NULL,
  `Prescription_Date` datetime DEFAULT NULL,
  `Medicine` varchar(100) DEFAULT NULL,
  INDEX `index_ID_idx`(`Prescription_ID` ASC),
PRIMARY KEY (`Prescription_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;



CREATE TABLE `prescription_doctor_patient` (
  `Prescription_ID` int(11) DEFAULT NULL,
  `Doctor_ID` int(11) DEFAULT NULL,
  `Patient_id` int(11) DEFAULT NULL,
  KEY `Prescription_ID` (`Prescription_ID`),
  KEY `Doctor_ID` (`Doctor_ID`),
  KEY `Patient_id` (`Patient_id`),
  CONSTRAINT `prescription_doctor_patient_ibfk_1` FOREIGN KEY
(`Prescription_ID`) REFERENCES `prescription` (`Prescription_ID`),
  CONSTRAINT `prescription_doctor_patient_ibfk_2` FOREIGN KEY (`Doctor_ID`)
REFERENCES `doctor` (`ID`),
  CONSTRAINT `prescription_doctor_patient_ibfk_3` FOREIGN KEY (`Patient_id`)
REFERENCES `patient` (`ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;



CREATE TABLE `tests` (
  `ID` int(11) NOT NULL auto_increment,
  `patient_id` int(11) DEFAULT NULL,
  `doctor_id` int(11) DEFAULT NULL,
  `test_type` varchar(50) DEFAULT NULL,
  `result` varchar(50) DEFAULT NULL,
  PRIMARY KEY (`ID`),
  KEY `fk_p_test` (`patient_id`),
  KEY `fk_d_test` (`doctor_id`),
INDEX `tests_ID_idx`(`ID` ASC),
INDEX `tests_patient_ID_idx`(`patient_id` ASC),
  CONSTRAINT `fk_d_test` FOREIGN KEY (`doctor_id`) REFERENCES `doctor` (`ID`),
  CONSTRAINT `fk_p_test` FOREIGN KEY (`patient_id`) REFERENCES `patient`
(`ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;




CREATE TABLE `bill` (
  `Bill ID` int(11) NOT NULL AUTO_INCREMENT,
  `Patient ID` int(11) NOT NULL,
  `Doctor ID` int(11) NOT NULL,
  `Test ID` int(11) NOT NULL,
  `Insurance Num` int(11) NOT NULL,
  `Bill Date` date NOT NULL,
  `Total Cost` int(11) NOT NULL,
  PRIMARY KEY (`Bill ID`),
  KEY `DoctorId` (`Doctor ID`),
  KEY `PatientId` (`Patient ID`),
  KEY `TestId` (`Test ID`),
  KEY `InsuranceNum` (`Insurance Num`),
INDEX `bill_ID_idx`(`BILL ID` ASC),
CONSTRAINT `fk_DoctorID` FOREIGN KEY (`Doctor ID`) REFERENCES `doctor`(`ID`),
  CONSTRAINT `fk_PatientID` FOREIGN KEY (`Patient ID`) REFERENCES
`patient`(`ID`),
  CONSTRAINT `fk_TestID` FOREIGN KEY (`Test ID`) REFERENCES `tests`(`ID`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;


CREATE TABLE `payment` (
  `Payment ID` int NOT NULL AUTO_INCREMENT,
  `Bill ID` int NOT NULL,
  `Payment Type` int DEFAULT NULL,
  `Payment date` date NOT NULL,
  `Total Cost` int NOT NULL,
INDEX `payment_ID_idx`(`Payment ID` ASC),
  PRIMARY KEY (`Payment ID`),
  KEY `fk_billID_pay` (`Bill ID`),
  CONSTRAINT `fk_billID_pay` FOREIGN KEY (`Bill ID`) REFERENCES `bill` (`Bill
ID`)
) ;

CREATE TABLE `visit_information` (
  `ID` int(11) NOT NULL,
  `Patient_ID` int(11) NOT NULL,
  `Date` date DEFAULT NULL,
  `doctor_ID` int(11) NOT NULL,
  PRIMARY KEY (`ID`),
  KEY `fk_pvi_patient` (`Patient_ID`),
  KEY `fk_pvi_doctor` (`doctor_ID`),
  CONSTRAINT `fk_pvi_doctor` FOREIGN KEY (`doctor_ID`) REFERENCES `doctor`
(`ID`),
  CONSTRAINT `fk_pvi_patient` FOREIGN KEY (`Patient_ID`) REFERENCES `patient`
(`ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
CREATE TABLE `visit_clinic_care_information` (
  `ID` int(11) NOT NULL,
  `visit_information_id` int(11) NOT NULL,
  `Symptoms` varchar(50) DEFAULT NULL,
  `Prescription` varchar(100) DEFAULT NULL,
  `diagnosis` varchar(80) DEFAULT NULL,
  KEY `fk_Id` (`visit_information_id`),
  CONSTRAINT `fk_Id` FOREIGN KEY (`visit_information_id`) REFERENCES
`visit_information` (`ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;


CREATE TABLE `visit_information_test_relation` (
  `visit_information_id` int(11) NOT NULL,
  `test_id` int(11) NOT NULL,
  KEY `fk_vi` (`visit_information_id`),
  CONSTRAINT `fk_vi` FOREIGN KEY (`visit_information_id`) REFERENCES
`visit_clinic_care_information` (`visit_information_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

4. AUDITS:

In order to save the logs for each operation (`UPDATE, INSERT, DELETE`) we created log tables corresponding to our database tables that undergo frequent manipulations.

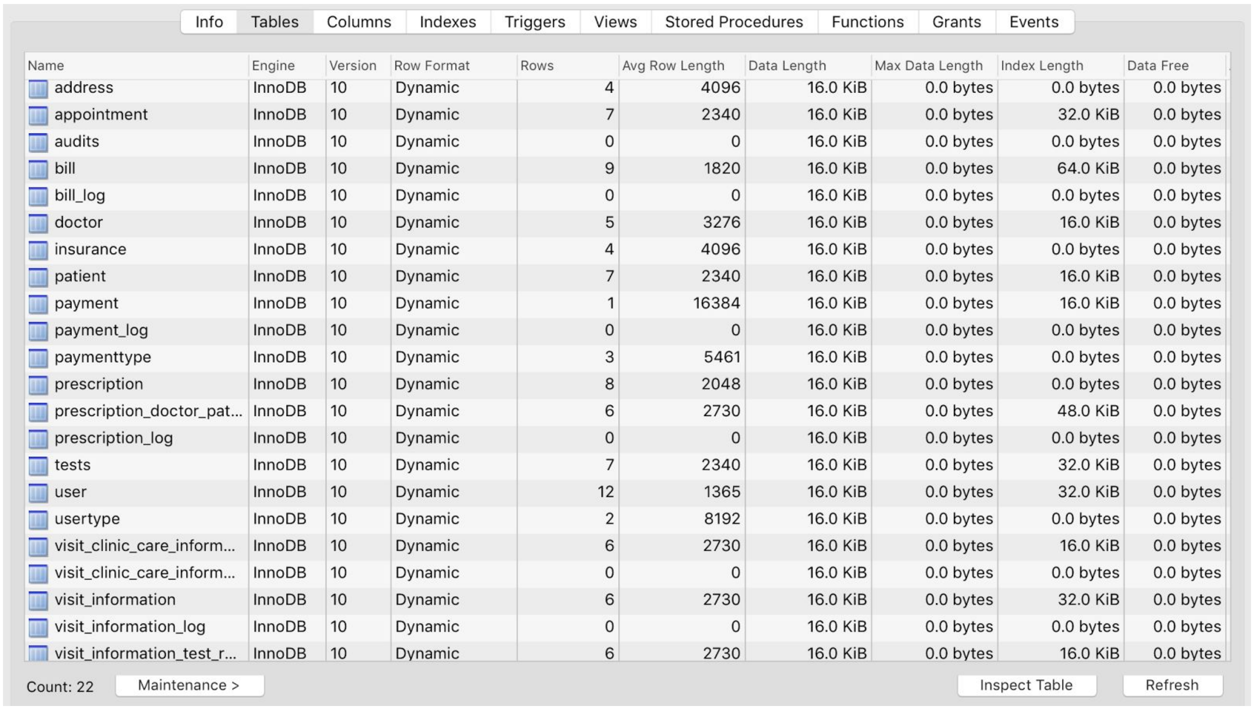
Log Tables: We did not create separate log tables but we have linked different tables to link to a table to have same log.
Ex. Patient and doctor has audits table which maintains the log for the both tables.

We create Log tables with following scripts:

```
CREATE TABLE `audits` (
  `contact_id` int(11) DEFAULT NULL,
  `inserted_date` date DEFAULT NULL,
  `inserted_by` varchar(60) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;


create table visit_clinic_care_information_log
(
     visit_clinic_care_information_ID int,
     visit_information_id int,
     inserted_date date,
     inserted_by varchar(60)
     );

create table prescription_log
(
     Prescription_ID int,
     PatientName varchar(50),
     DoctorName varchar(50),
     Prescription_Date date,
     Medicine varchar(50),
     inserted_date date,
     inserted_by varchar(60)
     );
```

```
create table bill_log
(
     Bill_ID int,
     patientId int,
     doctorId int,
     testid int,
     prescriptionNum int,
     insuranceNum int,
     BillDate date,
     totalcost int,
     inserted_date date,
     inserted_by varchar(60)
     );


   create table payment_log
(
     Payment_ID int,
     Bill_ID int,
     Payment_Type int,
     Payment_date date,
     Total_Cost int,
     inserted_date date,
     inserted_by varchar(60)
     );


create table visit_information_log
(
     visit_information_ID int,
     patientId int,
     doctorId int,
     inserted_date date,
     inserted_by varchar(60)
     );
```

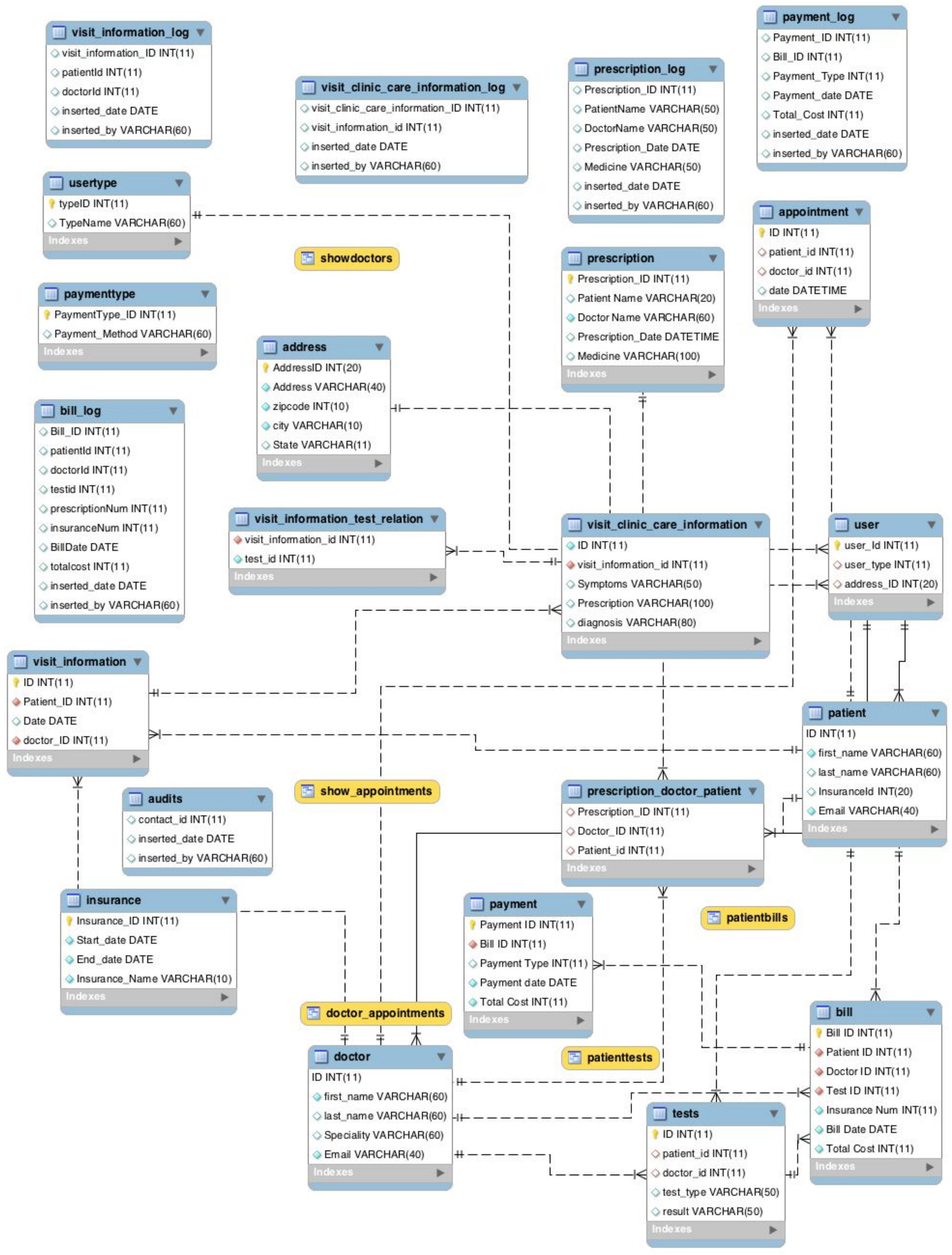
4.1 All tables as seen on selecting schema inspector for *adproject* database:



Name	Engine	Version	Row Format	Rows	Avg Row Length	Data Length	Max Data Length	Index Length	Data Free
address	InnoDB	10	Dynamic	4	4096	16.0 KiB	0.0 bytes	0.0 bytes	0.0 bytes
appointment	InnoDB	10	Dynamic	7	2340	16.0 KiB	0.0 bytes	32.0 KiB	0.0 bytes
audits	InnoDB	10	Dynamic	0	0	16.0 KiB	0.0 bytes	0.0 bytes	0.0 bytes
bill	InnoDB	10	Dynamic	9	1820	16.0 KiB	0.0 bytes	64.0 KiB	0.0 bytes
bill_log	InnoDB	10	Dynamic	0	0	16.0 KiB	0.0 bytes	0.0 bytes	0.0 bytes
doctor	InnoDB	10	Dynamic	5	3276	16.0 KiB	0.0 bytes	16.0 KiB	0.0 bytes
insurance	InnoDB	10	Dynamic	4	4096	16.0 KiB	0.0 bytes	0.0 bytes	0.0 bytes
patient	InnoDB	10	Dynamic	7	2340	16.0 KiB	0.0 bytes	16.0 KiB	0.0 bytes
payment	InnoDB	10	Dynamic	1	16384	16.0 KiB	0.0 bytes	16.0 KiB	0.0 bytes
payment_log	InnoDB	10	Dynamic	0	0	16.0 KiB	0.0 bytes	0.0 bytes	0.0 bytes
paymenttype	InnoDB	10	Dynamic	3	5461	16.0 KiB	0.0 bytes	0.0 bytes	0.0 bytes
prescription	InnoDB	10	Dynamic	8	2048	16.0 KiB	0.0 bytes	0.0 bytes	0.0 bytes
prescription_doctor_pat...	InnoDB	10	Dynamic	6	2730	16.0 KiB	0.0 bytes	48.0 KiB	0.0 bytes
prescription_log	InnoDB	10	Dynamic	0	0	16.0 KiB	0.0 bytes	0.0 bytes	0.0 bytes
tests	InnoDB	10	Dynamic	7	2340	16.0 KiB	0.0 bytes	32.0 KiB	0.0 bytes
user	InnoDB	10	Dynamic	12	1365	16.0 KiB	0.0 bytes	32.0 KiB	0.0 bytes
usertype	InnoDB	10	Dynamic	2	8192	16.0 KiB	0.0 bytes	0.0 bytes	0.0 bytes
visit_clinic_care_inform...	InnoDB	10	Dynamic	6	2730	16.0 KiB	0.0 bytes	16.0 KiB	0.0 bytes
visit_clinic_care_inform...	InnoDB	10	Dynamic	0	0	16.0 KiB	0.0 bytes	0.0 bytes	0.0 bytes
visit_information	InnoDB	10	Dynamic	6	2730	16.0 KiB	0.0 bytes	32.0 KiB	0.0 bytes
visit_information_log	InnoDB	10	Dynamic	0	0	16.0 KiB	0.0 bytes	0.0 bytes	0.0 bytes
visit_information_test_r...	InnoDB	10	Dynamic	6	2730	16.0 KiB	0.0 bytes	16.0 KiB	0.0 bytes

Count: 22

5. ER Diagram of proposed system:



6. Importing Data

From a file for *USER* table

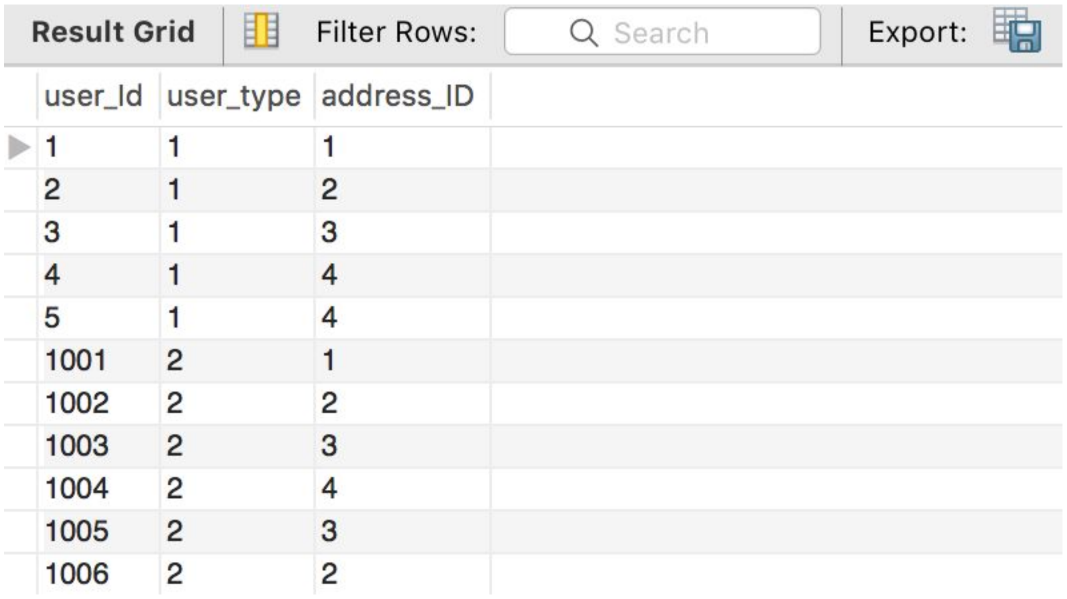
Data file: user.csv

We created a test database in excel, and ran the following script to import csv data into user's table:

```
LOAD DATA LOCAL INFILE '/Users/temp/Downloads/user.csv' INTO TABLE user
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\r\n'
IGNORE 1 LINES
(user_Id, user_type, address_ID);
```

SCENARIOS:

Output of `Select * from user` is reflected as follows:



user_Id	user_type	address_ID
1	1	1
2	1	2
3	1	3
4	1	4
5	1	4
1001	2	1
1002	2	2
1003	2	3
1004	2	4
1005	2	3
1006	2	2

We can retrieve the appointments for any patient from the `appointments` table

```
Select * from appointment where patient_id =1;
Select * from appointment where patient_id=(select Patient_id from patient
where first_name = “akshay” );
```

We can retrieve the insurance details for any user from the `patient_insurance_relationship`

```
Select * from patient
Join patient_insurance_relationship on
Patient.id = patient_insurance_relationship.patient_id
Join insurance
On patient_insurance_relationship.insurance_id = insurance.id;
```

Inserting and updating values based on KEY of one table to another table:

```
insert into usertype values(1,"Doctor");
insert into usertype values(2,"Patient");

insert into address values("1","435 M Barton Creek Dr","28262","Charlotte","NC");
insert into address values("2","9547 M Barton Creek Dr","28262","Charlotte","NC");
insert into address values("3","9547 E University Terrace
Drive","28262","Charlotte","NC");
insert into address values("4","28262 Z Bart Ck Dr","28262","Charlotte","NC");

insert into doctor values(1,"Ranjeet","Khan","Dental Care","Rk@gmail.com");
insert into doctor values(2,"Reet","Kan","Child Specialist","Rek@gmail.com");
insert into doctor values(3,"Rohan","Khan","Psychiatrist","Rohank@gmail.com");
insert into doctor values(4,"Praneet","Bhombha","Mental Care","PB@gmail.com");
insert into doctor values(5,"Suraj","Khan","Social Care","Rk@gmail.com");

insert into patient values(1001,"Akshay","A","101","aks@gmail.com");
insert into patient values(1002,"Bhavya","J","102","BhavyaJ@gmail.com");
insert into patient values(1003,"Rohan","K","103","Rk@gmail.com");
insert into patient values(1004,"Aks","A","104","ak@gmail.com");
insert into patient values(1005,"Akshay","C","101","aksc@gmail.com");
insert into patient values(1006,"ABC","ABC","105","abc@gmail.com");
```

We populate data across all tables using data load or Insert queries. (Insert queries included towards the end of the report, and in the zip folder).

Reflection of updated values in other correlated tables:

Adding speciality field to user table and populating values:

```
ALTER TABLE user
ADD speciality varchar(30);

select * from user;

update user
set speciality = 'Dentist'
where ID = 13;
```

Similarly, added specialty for other doctors.

The values for columns are retrieved directly from *user* table based on ‘speciality as staff’.

As soon as the patient makes visit, the *vsit_information* table will be updated with appropriate values. It can be worked out in following way:

Say, patient with ID 1 visits. Note that here patient_id is referenced as foreign key, and thus, must exist in *patient* ID column. Same is with doctor_id.

Similarly, based on the visits made by unique patients, values from other tables such as visit_clinic_care_information and tests will be retrieved to bills table. Thereby, the payment table shall update billing information of a patient based on patient_id and visit_id, thus stating the status for final bill as ‘paid’ or ‘pending’.

7. How we have Imagined the various scenarios that will happen in the clinic :

Patient Scenarios:

Will Sign Up Using the call addnewPatient() store procedure

Will then be able to take an appointment thorough call insertintoappoinmentifUserExists()

Will be able to see his appointment thorough views

Doctors Scenarios

Will see this appointment through views (view Appointments)

Will then fill the prescription through call procedure insertintoprescriptionifdoctorexists().

Give any tests if necessary

Generate Payment

Bill: The bill will be generated

Payment: If the patient has insurance (this application requires insurance because a student requires insurance before admission in a college).
Payment: Then a payment is generated by stored procedure call generatepayment().
(if insurance then 20% of total cost).

7.1 Stored Procedures:

```
delimiter //
CREATE DEFINER=`root`@`localhost` PROCEDURE `GetAllUsers`()
BEGIN
Select * from user;
END//
delimiter ;
```

```
delimiter //
CREATE DEFINER=`root`@`localhost` PROCEDURE `GetAllPatients`()
BEGIN
select patient.ID,patient.first_name,usertype.TypeName from patient
inner join user on
patient.ID = user.user_Id
inner join usertype on
user.user_type = usertype.typeID;
END//
delimiter ;
```

```
delimiter //
CREATE DEFINER=`root`@`localhost` PROCEDURE `USER_EXISTS`(IN `GIVEN_USERNAME`
VARCHAR(64) CHARSET utf8mb4)
BEGIN
    SET @User_exists = 0;
    SELECT COUNT(1) INTO     @found
    FROM patient
    WHERE `first_name` = GIVEN_USERNAME;
    IF @found > 0 THEN
        SET @User_exists = 1;
    END IF;
    SELECT @User_exists;
END //
delimiter ;
delimiter //

CREATE DEFINER=`root`@`localhost` PROCEDURE `USER_EXISTS_with_details`(IN
`GIVEN_USERNAME` VARCHAR(64) CHARSET utf8mb4)
BEGIN
    SET @User_exists = 0;
    SELECT COUNT(1) INTO @found
    FROM patient
    WHERE first_name = GIVEN_USERNAME;
    IF @found > 0 THEN
        SET @User_exists = 1;
    END IF;
    SELECT @User_exists;
    select * from patient where first_name = GIVEN_USERNAME;
END //

delimiter ;



delimiter //
CREATE DEFINER=`root`@`localhost` PROCEDURE `addnewPatient`(

firstName varchar(40),
lastName varchar(40),
address varchar(100),
insuranceName varchar(60),
email varchar(80))
begin
select @addressID := AddressID from address where Address=address;
select @insuranceID := Insurance_ID from insurance where
Insurance_Name=insuranceName;
select @PatientID := max(ID) from patient;
set @PatientID =  @PatientID + 1;
insert into user(user_Id,user_type,address_ID) values(@PatientID,2,@addressID);
insert into patient values(@PatientID,firstName,lastName,@insuranceID,email);
end//
delimiter ;

-- select * from address;
-- select * from user;
-- select * from patient;
```

```
delimiter //

CREATE DEFINER=`root`@`localhost` PROCEDURE `AddIntoAppointmentIfPatientExists`(
       `GIVEN_USERNAME` VARCHAR(64),
       `DOCTORNAME` varchar(20),
 `GIVENDATE` date
)
BEGIN
SET @User_exists = "User Does not Exists";
    SELECT COUNT(1) INTO @found
    FROM patient
    WHERE `first_name` = GIVEN_USERNAME;
    IF @found > 0 THEN
        SET @User_exists ="User Exists";
        SELECT @PATIENTID := ID FROM patient WHERE `first_name`=`GIVEN_USERNAME`;
         SELECT @DOCTORID := ID FROM DOCTOR WHERE `first_name`=`DOCTORNAME`;

INSERT INTO appointment
(patient_id,doctor_id,date)VALUES(@PATIENTID,@DOCTORID,`GIVENDATE`);
SELECT * FROM appointment;
END IF;
END//
delimiter ;


delimiter //
CREATE DEFINER=`root`@`localhost` PROCEDURE `InsertIntoPrescriptionIFDoctorExists`(
       `PATIENTNAME` VARCHAR(60),
       `DOCTORNAME` VARCHAR(60),
       `PRESCRIPTIONDATE` DATETIME,
       `MEDICINE` VARCHAR(60)
)
BEGIN
SET @User_exists = "User Does not Exists";
    SELECT COUNT(1) INTO @found
    FROM doctor
    WHERE `first_name` = `DOCTORNAME`;
    IF @found > 0 THEN
        SET @User_exists ="User Exists";
        SELECT @PATIENTID := ID FROM patient WHERE `first_name`=`PATIENTNAME`;
        SELECT @DOCTORID := ID FROM DOCTOR WHERE `first_name`=`DOCTORNAME`;
      INSERT INTO prescription(`Patient Name`,`Doctor
Name`,`Prescription_Date`,`Medicine`)
VALUES(`PATIENTNAME`,`DOCTORNAME`,`PRESCRIPTIONDATE`,`MEDICINE`);
      SELECT * FROM prescription;
    END IF;
END//
```

```
delimiter ;
delimiter //
CREATE DEFINER=`root`@`localhost` PROCEDURE `InsertIntoTestsIFDoctorExists`(
       `PATIENTNAME` VARCHAR(60),
       `DOCTORNAME` VARCHAR(60),
       `Test` varchar(60),
    `result` varchar(60)
)
BEGIN
SET @User_exists = "User Does not Exists";
    SELECT COUNT(1) INTO @found
    FROM doctor
    WHERE `first_name` = `DOCTORNAME`;
    IF @found > 0 THEN
        SET @User_exists ="User Exists";
        SELECT @PATIENTID := ID FROM patient WHERE `first_name`=`PATIENTNAME`;
             SELECT @DOCTORID := ID FROM DOCTOR WHERE `first_name`=`DOCTORNAME`;
      INSERT INTO
tests(tests.patient_id,tests.doctor_id,tests.test_type,tests.result)
VALUES(@PATIENTID,@DOCTORID,`Test`,`result`);
SELECT * FROM tests;
    END IF;
END//
delimiter ;



delimiter //
CREATE DEFINER=`root`@`localhost` PROCEDURE `addNewDoctor`(

firstName varchar(40),
lastName varchar(40),
addressID int,
speciality varchar(60),
email varchar(80))
begin
select @doctorID := max(ID) from doctor;
set @doctorID =  @doctorID + 1;
insert into user values(@doctorID,1,addressID);
insert into doctor values(@doctorID,firstName,lastName,speciality,email);
end//
delimiter ;
```

```
DELIMITER \\
CREATE DEFINER=`root`@`localhost` PROCEDURE `GenerateBills`(
       IN `PATIENTNAME` VARCHAR(60),
       IN `DOCTORNAME` VARCHAR(60),
       IN `BillDate` date,
       IN `TOTAL COST` int
       )
       BEGIN
       SELECT @PATIENTID := ID FROM patient WHERE `first_name`= `PATIENTNAME`;
       SELECT @DOCTORID := ID FROM DOCTOR WHERE `first_name`=`DOCTORNAME`;
       SELECT @TESTID := ID FROM tests WHERE patient_id = @PATIENTID;
       SELECT @INSURANCEID := InsuranceId FROM patient WHERE
       patient.first_name=`PATIENTNAME`;

       INSERT INTO BILL(`Patient ID`,`Doctor ID`,`Test ID`,`Insurance Num`,`Bill
       Date`,`Total Cost`)
       VALUES(@PATIENTID,@DOCTORID,@TESTID,@INSURANCEID,`BillDate`,`TOTAL COST`);
       SELECT * FROM bill;
       END \\


delimiter //
CREATE DEFINER=`root`@`localhost` PROCEDURE `generatepayment`(IN `GIVEN_PATIENT_ID`
int)
BEGIN
SELECT @BILLID := `Bill ID` FROM BILL WHERE `Patient ID`=`GIVEN_PATIENT_ID`;
SELECT @BILLDATE := `Bill Date` FROM BILL WHERE `Patient ID`=`GIVEN_PATIENT_ID`;
SELECT @TOTALCOST := `Total Cost` FROM BILL WHERE `Patient ID`=`GIVEN_PATIENT_ID`;
SET @TOTALCOST = (20* @TOTALCOST/100);
INSERT INTO PAYMENT (`Bill ID`,`Payment Type`,`Payment date`,`Total Cost`)
VALUES(@BILLID,2,@BILLDATE,@TOTALCOST);
SELECT * FROM PAYMENT;
END//
delimiter;
```

7.2 Scenario:

1. A Student needs to undergo health check up.
    a. Check: If student's profile already exists
        1. If yes, then proceed the user to book an appointment.
        2. If no, then create a profile.
2. The Student's information is recorded/updated, and student books an appointment.
3. The doctor can check all of his (own) appointments, so can the patient (student) and thus, can update whenever needed.
4. The doctor performs check up on the patient.
    a. If tests required, conducts tests, else proceeds with prescriptions.
5. The doctor prescribes medicines.
6. Student checks out and final bill is generated with list of all params (medicines, tests etc).
7. Based on the bll, final payment is generated. The final cost is calculated considering the insurance type.

For such an scenario, a typical sequence of execution of our stored procedures will be in following manner:

7.3 Sequentially calling above written stored procedures:

```
call GetAllusers();
call GetAllPatients();
call USER_EXISTS('Bhavya');
call USER_EXISTS_with_details('Bhavya');
call addnewPatient('Rick','Arduino','9547 M Barton Creek
Dr','blue101','rickarduino@gmail.com');
call AddIntoAppointmentIfPatientExists('Akshay','Rohan','2017-05-10');
call InsertIntoPrescriptionIFDoctorExists('Bhavya','Rohan',now(),'Vito');
call InsertIntoTestsIFDoctorExists('Bhavya','Rohan','Malaria','Negative');
call GenerateBills('Bhavya','Rohan',now(),1050);
call generatepayment(1001);
```

8. Triggers

- **store_in_log_doctor**

This trigger stores the insert on all the doctors table.

```
delimiter //
CREATE TRIGGER store_in_log_doctor
AFTER INSERT
   ON doctor FOR EACH ROW
BEGIN

   DECLARE vUser varchar(50);

   -- Find username of person performing the INSERT into table
   SELECT USER() INTO vUser;

   -- Insert record into audit table
   INSERT INTO audits
   (
     contact_id,
     inserted_date,
    inserted_by)
   VALUES
   (
   new.ID,
     SYSDATE(),
     vUser );

END; //

DELIMITER ;
```

- **store_in_log**

This trigger stores the insert on all the patient table.

```
DELIMITER //

CREATE TRIGGER store_in_log
AFTER INSERT
   ON patient FOR EACH ROW
BEGIN

   DECLARE vUser varchar(50);
   -- Find username of person performing the INSERT into table
   SELECT USER() INTO vUser;

   -- Insert record into audit table
   INSERT INTO audits
   (
     contact_id,
     inserted_date,
     inserted_by)
   VALUES
   (
   new.ID,
     SYSDATE(),
     vUser );

END; //

DELIMITER ;
```

- **store_in_log_payments**

This trigger stores the insert on all the payments table.

```
DELIMITER //

CREATE TRIGGER store_in_log_payments
AFTER INSERT
   ON payment FOR EACH ROW
BEGIN

   DECLARE vUser varchar(50);

   -- Find username of person performing the INSERT into table
   SELECT USER() INTO vUser;

   -- Insert record into audit table
   INSERT INTO audits
   (
     contact_id,
     inserted_date,
     inserted_by)
   VALUES
   (
   new.`Payment ID`,
     SYSDATE(),
     vUser );
```

```
END //

DELIMITER ;
```

- **store_in_visit_clinic_care_information_log**

This trigger stores the insert on all the visit_clinic_care_information table.

```
DELIMITER //
CREATE TRIGGER store_in_visit_clinic_care_information_log
AFTER INSERT
   ON visit_clinic_care_information FOR EACH ROW
BEGIN

   DECLARE vUser varchar(50);

   -- Find username of person performing the INSERT into table
   SELECT USER() INTO vUser;

   -- Insert record into audit table
   INSERT INTO visit_clinic_care_information_log
   (
     visit_clinic_care_information_ID,
     visit_information_id,
     inserted_date,
     inserted_by)
   VALUES
   (
   new.ID,
   new.visit_information_id,
     SYSDATE(),
     vUser );

END; //

DELIMITER ;
```

- **store_in_visit_information_log**

Log all the inserts on the visit_information table.

```
DELIMITER //
CREATE TRIGGER store_in_visit_information_log
AFTER INSERT
   ON visit_information FOR EACH ROW
BEGIN

   DECLARE vUser varchar(50);

   -- Find username of person performing the INSERT into table
   SELECT USER() INTO vUser;

   -- Insert record into audit table
   INSERT INTO visit_information_log
   (

     visit_information_id,
     patientId,
     doctorId,
     inserted_date,
     inserted_by)
   VALUES
   (
   new.ID,
   new.Patient_ID,
   new.doctor_ID,
     SYSDATE(),
     vUser );

END; //
```

- **store_in_prescription_log**

log for prescription table.

```
DELIMITER ;

DELIMITER //
CREATE TRIGGER store_in_prescription_log
AFTER INSERT
   ON prescription FOR EACH ROW
```

```
BEGIN

   DECLARE vUser varchar(50);

   -- Find username of person performing the INSERT into table
   SELECT USER() INTO vUser;

   -- Insert record into audit table
   INSERT INTO prescription_log
   (

     Prescription_ID,
     PatientName,
     DoctorName,
     Prescription_Date,
     Medicine,
     inserted_date,
     inserted_by)
   VALUES
   (
   new.`Prescription_ID`,
   new.`Patient Name`,
   new.`Doctor Name`,
   new.`Prescription_Date`,
   new.`Medicine`,
     SYSDATE(),
     vUser );

END; //

DELIMITER ;
```

- **store_in_bill_log**

Log for all the insert on bills table.

```
DELIMITER //
CREATE TRIGGER store_in_bill_log
AFTER INSERT
   ON bill FOR EACH ROW
BEGIN

   DECLARE vUser varchar(50);

   -- Find username of person performing the INSERT into table
   SELECT USER() INTO vUser;

   -- Insert record into audit table
   INSERT INTO bill_log
   (
     Bill_ID ,
     patientId,
     doctorId,
     testid,
     insuranceNum,
     BillDate,
     totalcost,
     inserted_date,
     inserted_by
     )
     VALUES
   (
   new.`Bill ID`,
     new.`Patient ID`,
     new.`Doctor ID`,
     new.`Test ID`,
     new.`Insurance Num`,
     new.`Bill Date`,
     totalcost,
     SYSDATE(),
     vUser );

END //

DELIMITER ;
```

- store_in_payment_log

Log for payments table.

```
DELIMITER //
CREATE TRIGGER store_in_payment_log
AFTER INSERT
   ON payment FOR EACH ROW
BEGIN

   DECLARE vUser varchar(50);

   -- Find username of person performing the INSERT into table
   SELECT USER() INTO vUser;

   -- Insert record into audit table
   INSERT INTO payment_log
   (
       Payment_ID,
     Bill_ID,
     Payment_Type,
     Payment_date,
     Total_Cost,
     inserted_date,
     inserted_by
     )
     VALUES
   (
     new.`Payment ID`,
     new.`Bill ID`,
     new.`Payment Type`,
     new.`Payment date`,
     new.`Total Cost`,
     SYSDATE(),
     vUser
     );

END //

DELIMITER ;
```

- **patient_after_update**

Update trigger for patient table.

```
delimiter //
CREATE TRIGGER patient_after_update
AFTER UPDATE
   ON patient FOR EACH ROW

BEGIN

   DECLARE vUser varchar(50);

   -- Find username of person performing the INSERT into table
   SELECT USER() INTO vUser;

   -- Insert record into audit table
   INSERT INTO audits
   ( contact_id,
     updated_date,
     updated_by)
   VALUES
   ( NEW.ID,
     SYSDATE(),
     vUser );

END; //

DELIMITER ;
```

- **doctor_after_update**

Update trigger doctors table.It will store it in the log file.

```
delimiter //
CREATE TRIGGER doctor_after_update
AFTER UPDATE
   ON doctor FOR EACH ROW

BEGIN

   DECLARE vUser varchar(50);

   -- Find username of person performing the INSERT into table
   SELECT USER() INTO vUser;

   -- Insert record into audit table
   INSERT INTO audits
   ( contact_id,
     updated_date,
     updated_by)
   VALUES
   ( NEW.ID,
     SYSDATE(),
     vUser );

END; //

DELIMITER ;

delimiter //
CREATE TRIGGER doctor_after_update
AFTER UPDATE
   ON doctor FOR EACH ROW

BEGIN

   DECLARE vUser varchar(50);

   -- Find username of person performing the INSERT into table
   SELECT USER() INTO vUser;
   -- Insert record into audit table
   INSERT INTO audits
   ( contact_id,
     updated_date,
     updated_by)
   VALUES
   ( NEW.ID,
     SYSDATE(),
     vUser );

END; //

DELIMITER ;
```

- **doctor_after_delete**

Delete trigger to store any deletes on doctor table.

```
delimiter //
CREATE TRIGGER doctor_after_delete
AFTER DELETE
   ON doctor FOR EACH ROW

BEGIN

   DECLARE vUser varchar(50);

   -- Find username of person performing the INSERT into table
   SELECT USER() INTO vUser;

   -- Insert record into audit table
   INSERT INTO audits
   ( contact_id,
     updated_date,
     updated_by)
   VALUES
   ( old.ID,
     SYSDATE(),
     vUser );

END; //

DELIMITER ;
```

- **patient_after_delete**

Delete trigger for deletes on the patient table

```
delimiter //
CREATE TRIGGER patient_after_delete
AFTER DELETE
   ON patient FOR EACH ROW

BEGIN
   DECLARE vUser varchar(50);

   -- Find username of person performing the INSERT into table
   SELECT USER() INTO vUser;

   -- Insert record into audit table
   INSERT INTO audits
   ( contact_id,
     updated_date,
     updated_by)
   VALUES
   ( old.ID,
     SYSDATE(),
     vUser );

END; //

DELIMITER ;
```

- **bill_after_delete**

Trigger for delete for bill table.

```
delimiter //
CREATE TRIGGER bill_after_delete
AFTER DELETE
   ON bill FOR EACH ROW

BEGIN

   DECLARE vUser varchar(50);

   -- Find username of person performing the INSERT into table
   SELECT USER() INTO vUser;

   -- Insert record into audit table
   INSERT INTO audits
   ( contact_id,
     updated_date,
     updated_by)
   VALUES
   ( old.ID,
     SYSDATE(),
     vUser );

END; //

DELIMITER ;
```

- appointment_after_delete

Trigger for delete on appointments table.

```
delimiter //
CREATE TRIGGER appointment_after_delete
AFTER DELETE
   ON appointment FOR EACH ROW

BEGIN

   DECLARE vUser varchar(50);

   -- Find username of person performing the INSERT into table
   SELECT USER() INTO vUser;

   -- Insert record into audit table
   INSERT INTO audits
   ( contact_id,
     updated_date,
     updated_by)
   VALUES
   ( old.ID,
     SYSDATE(),
     vUser );

END; //

DELIMITER ;
```

- delete_on_payment_log

Log for payments table.

```
DELIMITER //
CREATE TRIGGER delete_on_payment_log
AFTER delete
   ON payment FOR EACH ROW
BEGIN

   DECLARE vUser varchar(50);

   -- Find username of person performing the INSERT into table
   SELECT USER() INTO vUser;

   -- Insert record into audit table
   INSERT INTO payment_log
   (
        Payment_ID,
     Bill_ID,
     Payment_Type,
     Payment_date,
     Total_Cost,
     inserted_date,
     inserted_by
     )
     VALUES
   (
     old.`Payment ID`,
     old.`Bill ID`,
     old.`Payment Type`,
     old.`Payment date`,
     old.`Total Cost`,
     SYSDATE(),
     vUser
     );

END //

DELIMITER ;
```

9. Views

- showdoctors

This view is for the patient to select a doctor.We display the doctor name and his speciality.The patient can take an appointment with a doctor with different speciality and according to his need.

```
CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL SECURITY DEFINER VIEW
`showdoctors`
AS
select `doctor`.`first_name` AS `first_name`,`doctor`.`last_name` AS
`last_name`,`doctor`.`Speciality` AS `speciality` from `doctor`;

CREATE VIEW `doctor_appointments` AS SELECT patient.first_name AS
Patient_Name,doctor.first_name AS doctor_Name,appointment.date
FROM
patient inner join appointment on
patient.ID = appointment.patient_id
inner join doctor on
doctor.ID = appointment.doctor_id;
```

- showdoctors

This view is to search for a doctor with a specific ID.

```
CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL SECURITY DEFINER VIEW
`showdoctors` AS select `doctor`.`first_name` AS `first_name`,`doctor`.`last_name` AS
`last_name`,`doctor`.`Speciality` AS `speciality` from `doctor`;=
```

- show_appointments

This view is to show all the appointments.We implemented it using a join query.It will display only the appointments of the patients with their doctors.

```
CREATE VIEW `show_appointments` AS SELECT patient.first_name AS
Patient_Name,doctor.first_name AS doctor_Name,appointment.date
FROM
patient inner join appointment on
patient.ID = appointment.patient_id
inner join doctor on
doctor.ID = appointment.doctor_id;
```

- doctor_appointments

This query is similar to the above but here we show this from the doctors point of view.Where the doctor can be able to see all the appointments of the day.

```
CREATE VIEW `doctor_appointments` AS SELECT patient.first_name AS
Patient_Name,doctor.first_name AS doctor_Name,appointment.date
FROM
patient inner join appointment on
patient.ID = appointment.patient_id
inner join doctor on
doctor.ID = appointment.doctor_id;
```

- PatientTests

To display all the patient tests that have been performed.

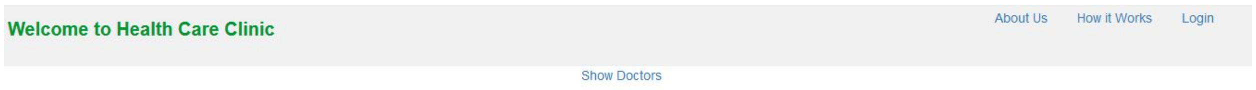
```
create view PatientTests as select patient.first_name as Patient_Name,
doctor.first_name as Doctor_Name, tests.test_type as Test_Type, tests.result as
Test_Result
from patient
inner join tests on patient.ID=tests.patient_id
inner join doctor on
doctor.ID=tests.doctor_id;
```

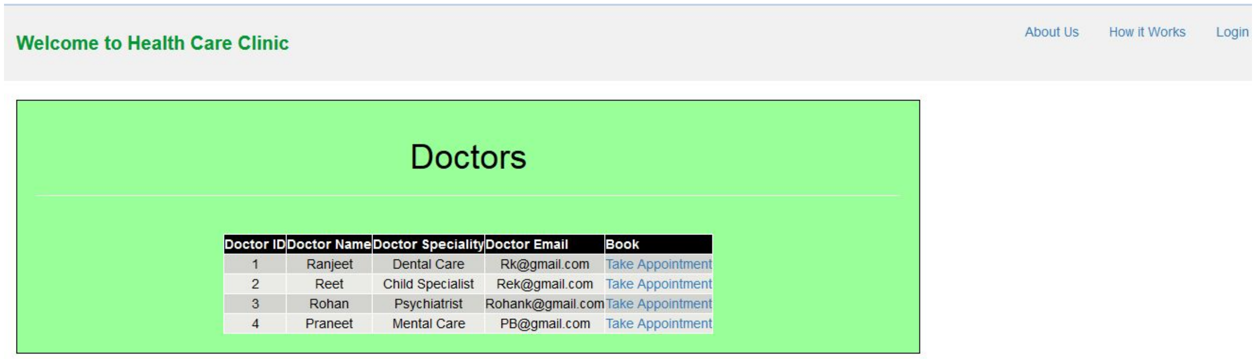
- PatientBills

To see all the bills that are there for all the users.To see the bill generated.

```
create view PatientBills as select patient.first_name as Patient_Name,
doctor.first_name as Doctor_Name, bill.`Total Cost` as Total_cost from patient
inner join bill on patient.ID=bill.`Patient ID`
inner join doctor on
doctor.ID= bill.`Doctor ID`
```

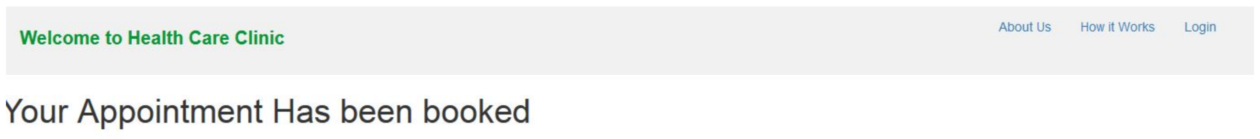
10. Web Application Demo





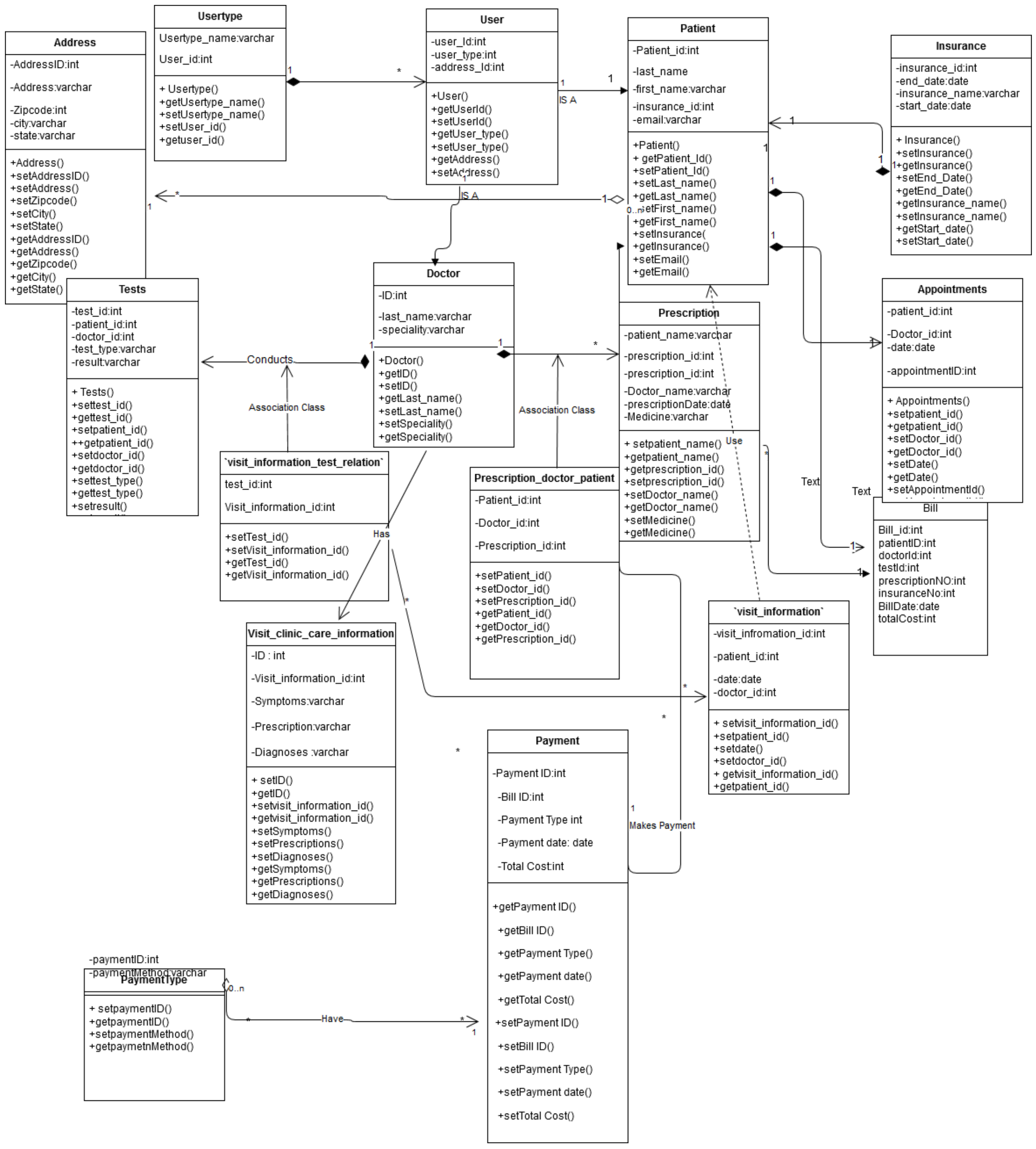
Doctors

Doctor ID	Doctor Name	Doctor Speciality	Doctor Email	Book
1	Ranjeet	Dental Care	Rk@gmail.com	Take Appointment
2	Reet	Child Specialist	Rek@gmail.com	Take Appointment
3	Rohan	Psychiatrist	Rohank@gmail.com	Take Appointment
4	Praneet	Mental Care	PB@gmail.com	Take Appointment



Your Appointment Has been booked

11. Application Class Diagram



12. Conclusion

We implemented a Health Care System for the University students, and as per our goal, we were able to cover maximum functionalities as per our model inspiration of Health care unit at UNC Charlotte.
We learnt and implemented stored procedures, views, triggers to handle all functions with an on-demand click actions, just as desired for an web application.
Our implementation covers safe logs for data manipulations actions of insert, delete and update, with log for the most frequently accessed and manipulated tables.
Our future scope included extrapolating the scope of project and cover up the implementation in NoSQL.

Individual Responsibilities

Akshay M Adagale:

In this project I implemented the store procedures ,views and triggers for the database. I implemented the store procedures keeping in mind the scenario of patient and doctor.Hence according to that I created the getAppointmentIfPatientExists store procedure and like wise the showDoctor view from where the patient can take an appointment with the doctor and book an appointment.
Further I was able to implement the triggers like insert,update and delete on the tables which will be used frequently.Through those triggers we store the various updates in the logs table and audit tables.The ones which we did not have a log are because they will be maintained in the database and are not accessed frequently.
Thus keeping this in mind we also had various views which are necessary for patient appointment and doctor
to check his appointment and generate the bills through stored procedures and created a web application to demonstrate the working of the procedures..Also assisted bhavya and rohan in database normalization and database creation and applying indexes.

Bhavya Joshi:

I implemented the class and ER diagram for the application.Figuring out the necessary classes that are necessary for the application during web development and the necessary scenarios that are necessary and are actually present in a clinic.
Thus through this I was able to help my teammates to design the database classes and tables that are necessary for the implementation of the project.Helped Akshay and Rohan in implementing the database and report generation.

Rohan Khairnar

Worked on database structure and creation, normalization. For the later part, worked on stored procedures and views creation. Assisted Akshay and Bhavya with incorporating rest of the functionalities throughout the project as well as presentation and report contents.