**ADVANCED FEATURES OF AN EMR DATABASE SYSTEM**

**Prepared By:**

**Shweta Rajaram Patil(800989198)**

**Meetu Uthra(800985777)**

**Tushar Arvind(800823103)**



**Course: Applied Databases**

**University of North Carolina at Charlotte**

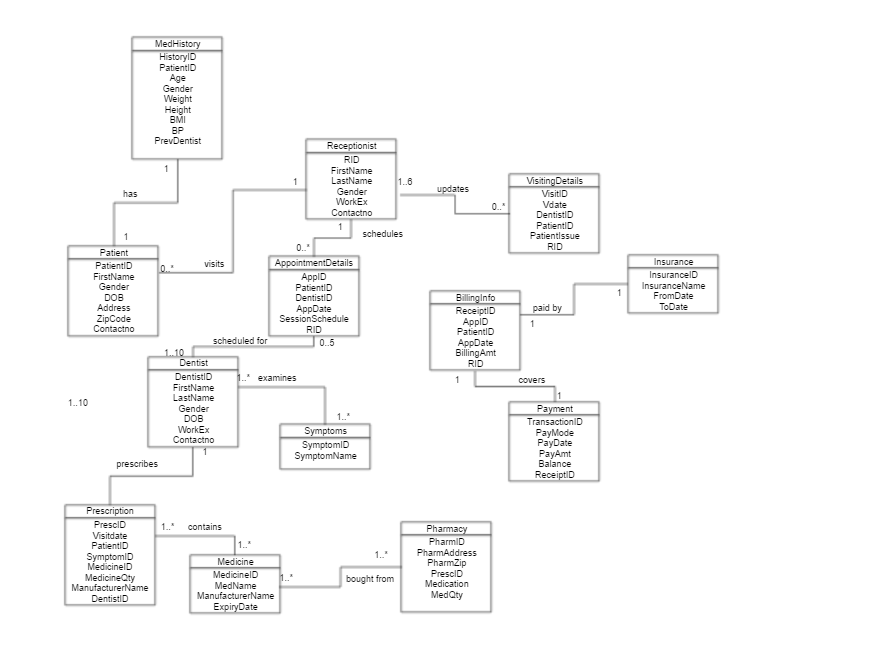
Project Scope:

This Electronic Medical Records system captures information of Patients dealing with Dental disease. The key focus is to be able to retrieve Patient’s Med History, Visiting details, Insurance Info, Appointment details, Billing Info and the Dentist associated with the Patient. The database will support the retrieval of information of all patients that have been admitted in a hospital over the past one month.

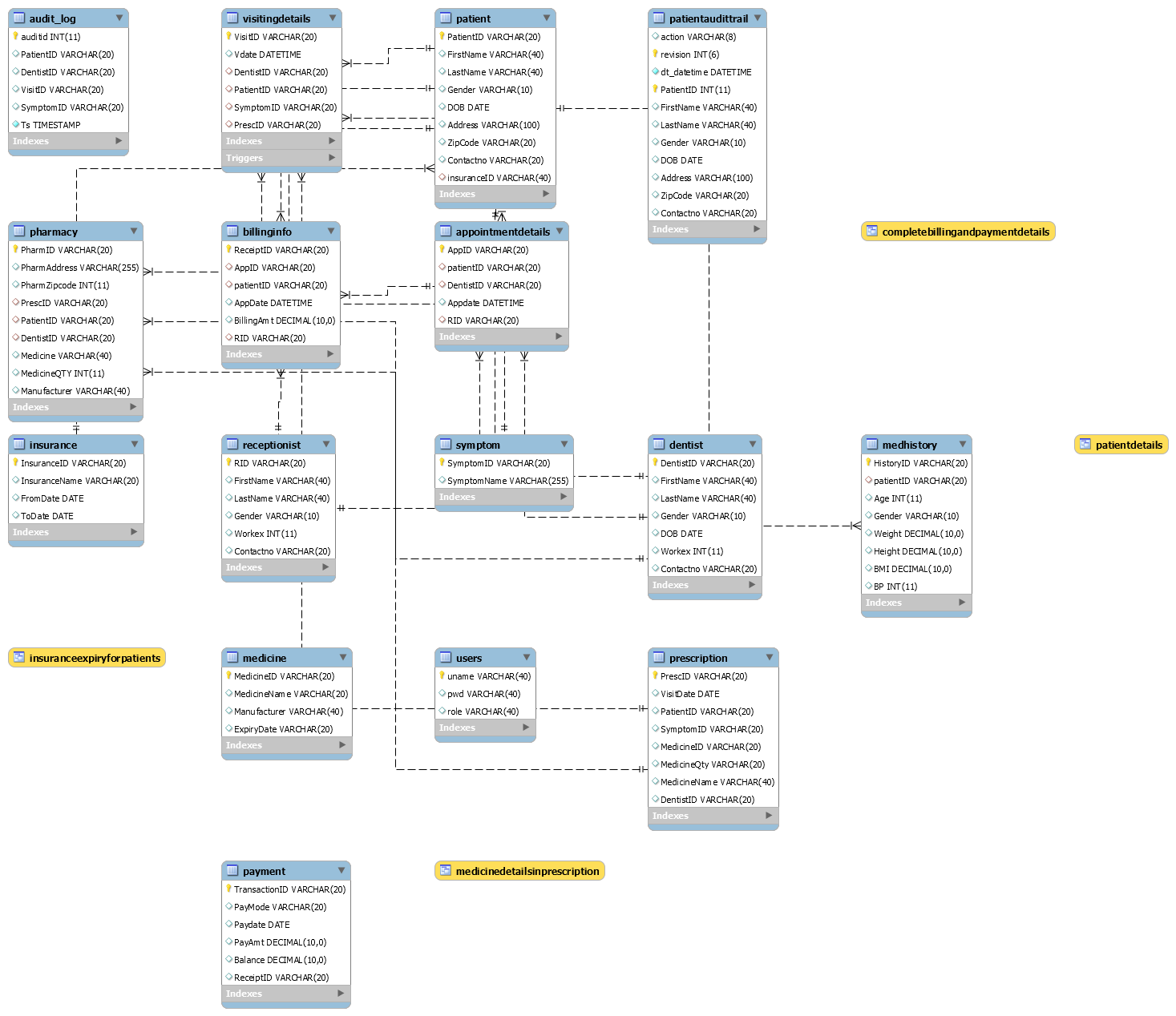
**Project Constraints**

* A patient can take treatment or be associated with only one Dentist at a time.
* Max 10 appointments will be handled by the clinic in a day.
* A patient can book only one appointment in a day.
* For every appointment booked by a patient he will be provided with one Visit ID which will be valid for one day using which he can make multiple visits in a day.

UML Structure:



ER Diagram for the EMR Database: BiteIn Dental Clinic:

**

User Authentication:

Authentication is achieved using username, password and role. Following Stored Procedure is used to implement authentication

*DELIMITER $$*

*CREATE DEFINER=`root`@`localhost` PROCEDURE `UserAuth`(IN `USERNAME` VARCHAR(64), IN `PASS` VARCHAR(20), IN `ROLE` VARCHAR(5))*

*BEGIN*

*SELECT @present := COUNT(uname) FROM users WHERE concat(uname,pwd,role) = concat(@USERNAME,@PASS,@ROLE);*

*IF @present > 0 THEN SET @User\_exists = 1;*

*ELSE SET @User\_exists = 0;*

*END IF;*

*SELECT @User\_exists;*

*END$$*

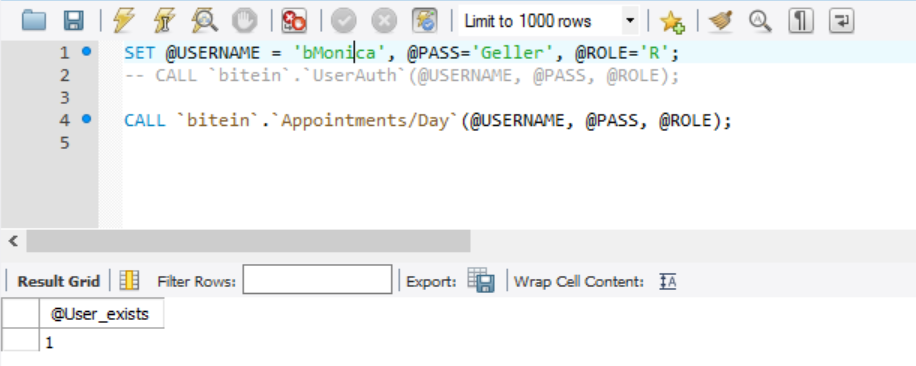
*DELIMITER ;*

CORRECT USERNAME & PASSWORD:

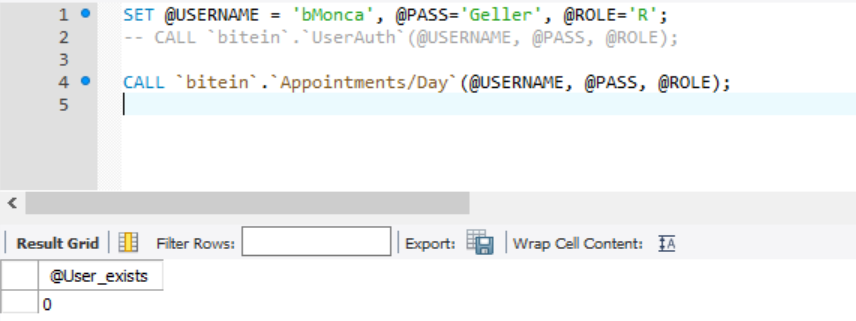
*SET @USERNAME = 'bRoss', @PASS='Geller', @ROLE='D';*

*-- CALL `bitein`.`UserAuth`(@USERNAME, @PASS, @ROLE);*

*CALL `bitein`.`Appointments/Day`(@USERNAME, @PASS, @ROLE);*



INCORRECT USERNAME & PASSWORD:

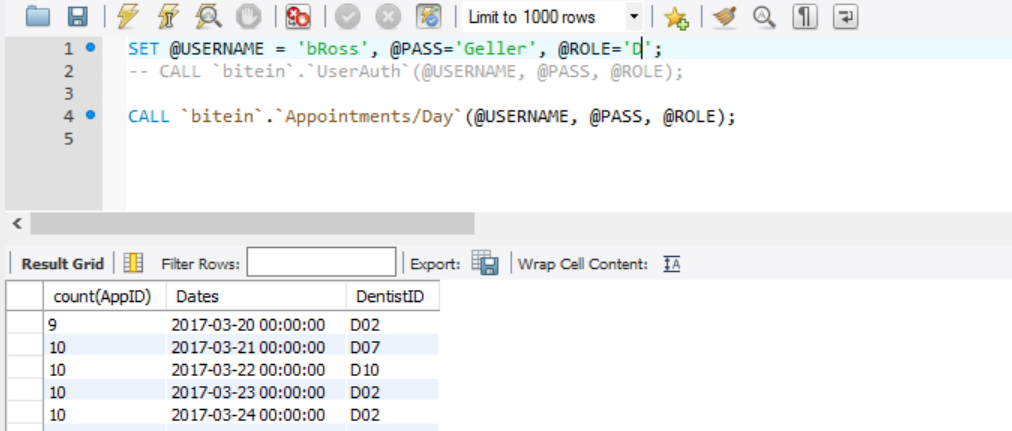


User Authorization

After user is authenticated, for accessing any of the tables, user have an authorized access to that table. This requirement is achieved using Role Based Authorization. We have 5 user roles defined:

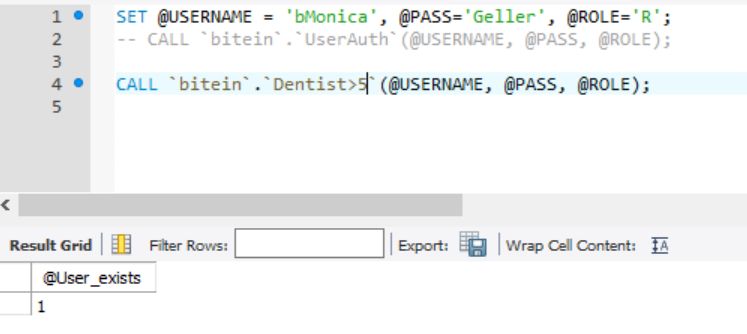
1. Dentist
2. Receptionist
3. Patient
4. Pharmacist
5. Admin

USER EXISTS AND AUTHORIZED:



USER EXITS AND NOT AUTHORIZED:

We authenticate the user that is trying to access a table. If user is not authorized, Error message is thrown: “User not authorized”.



Stored Procedures:

Every stored procedure is a capability that can be assigned as a function of a user role. Below stored procedures have limitations according to the roles specified.

* **Only patient and admin should be able to see dentist work experience**

*DELIMITER $$*

*CREATE DEFINER=`root`@`localhost` PROCEDURE `Dentist>5`(IN `USERNAME` VARCHAR(64), IN `PASS` VARCHAR(20), IN `ROLE` VARCHAR(5))*

*BEGIN*

*CALL `bitein`.`UserAuth`(@USERNAME, @PASS, @ROLE);*

*IF @User\_exists > 0 THEN*

*IF @ROLE='P' OR @ROLE='Adm' THEN*

*SELECT \*FROM BITEIN.DENTIST*

*WHERE WORKEX>5;*

*ELSE*

*SET @message\_text = ('User not authorized');*

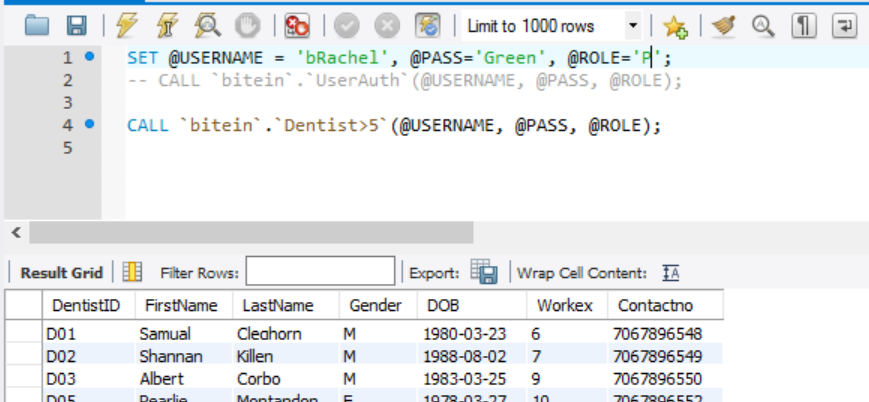
*SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = @message\_text;*

*END IF;*

*END IF;*

*END$$*

*DELIMITER ;*



* **Only Dentist and Admin should be able to see his appointments**

*DELIMITER $$*

*CREATE DEFINER=`root`@`localhost` PROCEDURE `Appointments/Day`(IN `USERNAME` VARCHAR(64), IN `PASS` VARCHAR(20), IN `ROLE` VARCHAR(5))*

*BEGIN*

*CALL `bitein`.`UserAuth`(@USERNAME, @PASS, @ROLE);*

*IF @User\_exists > 0 THEN*

*IF @ROLE='D' OR @ROLE='Adm' THEN*

*select count(AppID), Appdate as Dates, DentistID from appointmentdetails*

*group by(appdate) order by Appdate ASC;*

*ELSE*

*SET @message\_text = ('User not authorized');*

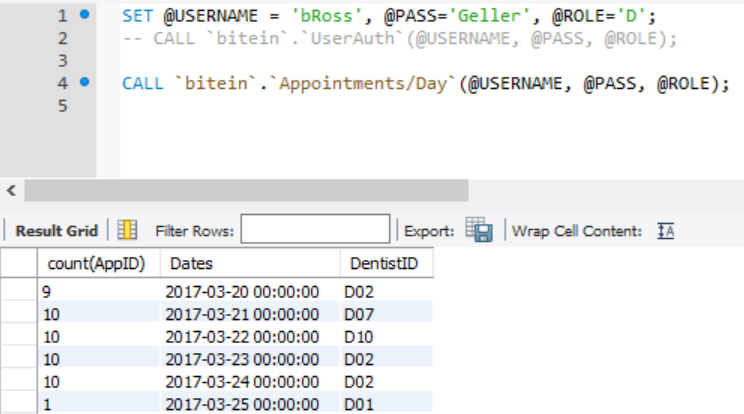
*SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = @message\_text;*

*END IF;*

*END IF;*

*END$$*

*DELIMITER ;*



* **Only Receptionist and Admin should be able to see billing details**

*DELIMITER $$*

*CREATE DEFINER=`root`@`localhost` PROCEDURE `Billing/ApptForEachDentist`(IN `USERNAME` VARCHAR(64), IN `PASS` VARCHAR(20), IN `ROLE` VARCHAR(5))*

*BEGIN*

*CALL `bitein`.`UserAuth`(@USERNAME, @PASS, @ROLE);*

*IF @User\_exists > 0 THEN*

*IF @ROLE='R' OR @ROLE='Adm' THEN*

*select d.DentistID, d.FirstName, d.LastName, BillingAmt from billinginfo as b join appointmentdetails as a*

*on b.AppID=a.AppID join dentist as d on d.DentistID = a.DentistID;*

*ELSE*

*SET @message\_text = ('User not authorized');*

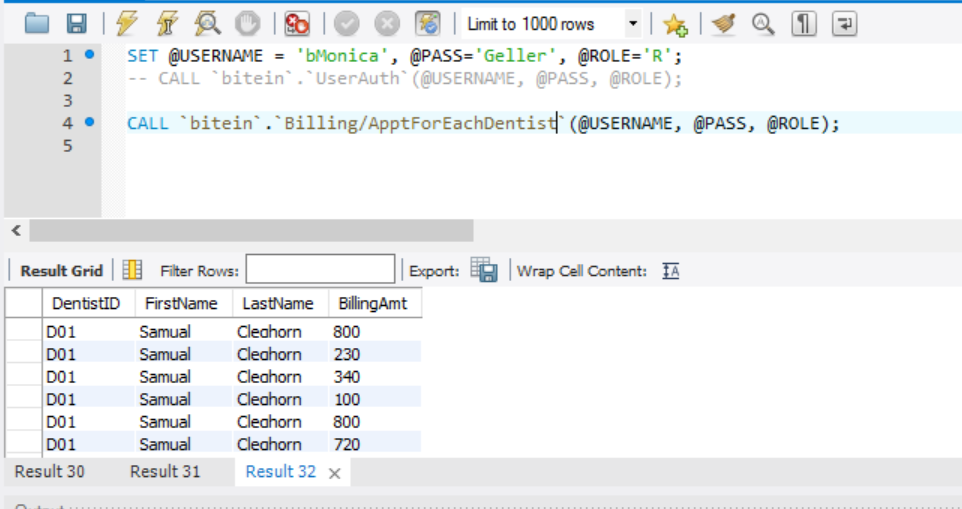
*SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = @message\_text;*

*END IF;*

*END IF;*

*END$$*

*DELIMITER ;*



* **Only Admin should be able to see Dentist earnings for the clinic**

*DELIMITER $$*

*CREATE DEFINER=`root`@`localhost` PROCEDURE `DentistEarnings`(IN `USERNAME` VARCHAR(64), IN `PASS` VARCHAR(20), IN `ROLE` VARCHAR(5))*

*BEGIN*

*CALL `bitein`.`UserAuth`(@USERNAME, @PASS, @ROLE);*

*IF @User\_exists > 0 THEN*

*IF @ROLE='Adm' THEN*

*select d.DentistID, d.FirstName, d.LastName, sum(BillingAmt) as TotEarning from billinginfo as b join appointmentdetails as a*

*on b.AppID=a.AppID join dentist as d on d.DentistID = a.DentistID*

*group by(DentistID);*

*ELSE*

*SET @message\_text = ('User not authorized');*

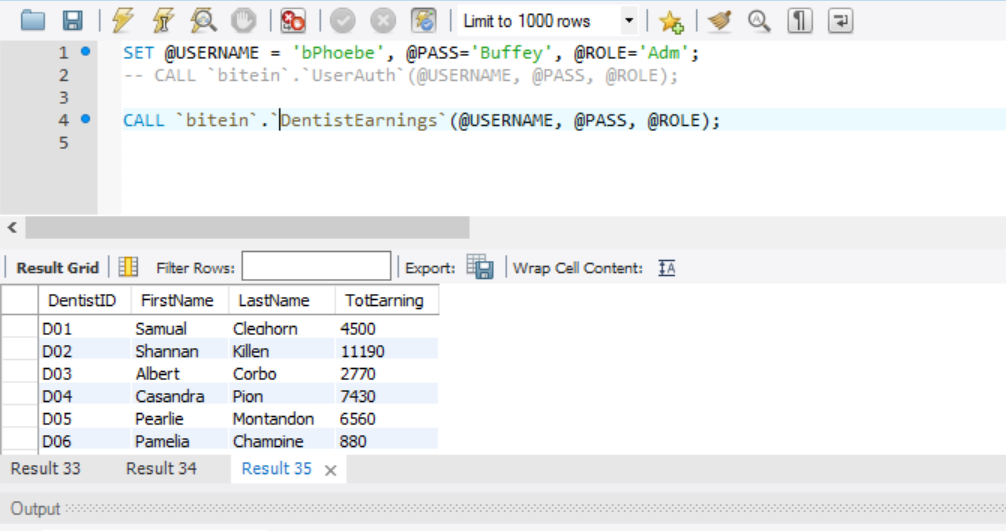
*SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = @message\_text;*

*END IF;*

*END IF;*

*END$$*

*DELIMITER ;*



* **All authorized users can see all dentists**

*DELIMITER $$*

*CREATE DEFINER=`root`@`localhost` PROCEDURE `getAllDentists`(IN `USERNAME` VARCHAR(64), IN `PASS` VARCHAR(20), IN `ROLE` VARCHAR(5))*

*BEGIN*

*CALL `bitein`.`UserAuth`(@USERNAME, @PASS, @ROLE);*

*IF @User\_exists > 0 THEN*

*SELECT \*FROM BITEIN.DENTIST;*

*ELSE*

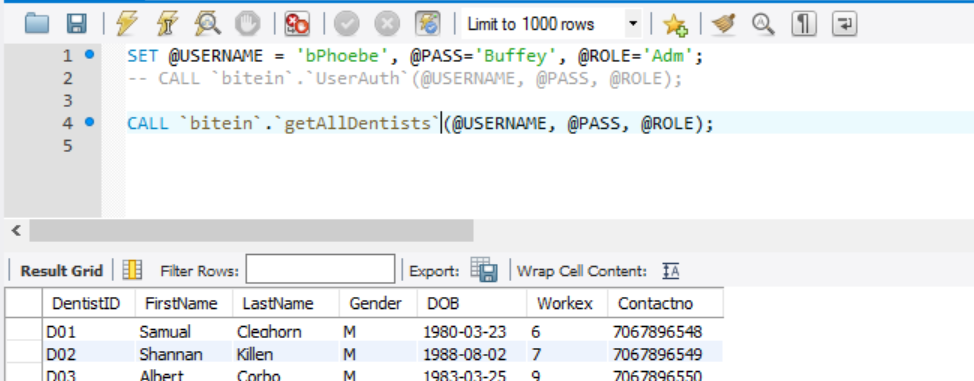
*SET @message\_text = ('User not authorized');*

*SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = @message\_text;*

*END IF;*

*END$$*

*DELIMITER ;*



* **Patient, Receptionist and admin can see all pharmacies**

*DELIMITER $*

*CREATE DEFINER=`root`@`localhost` PROCEDURE `GetAllPharmacies`(IN `USERNAME` VARCHAR(64), IN `PASS` VARCHAR(20), IN `ROLE` VARCHAR(5))*

*BEGIN*

*CALL `bitein`.`UserAuth`(@USERNAME, @PASS, @ROLE);*

*IF @User\_exists > 0 THEN*

*IF @ROLE='R' OR 'P' OR @ROLE='Adm' THEN*

*SELECT \*FROM BITEIN.PHARMACY;*

*ELSE*

*SET @message\_text = ('User not authorized');*

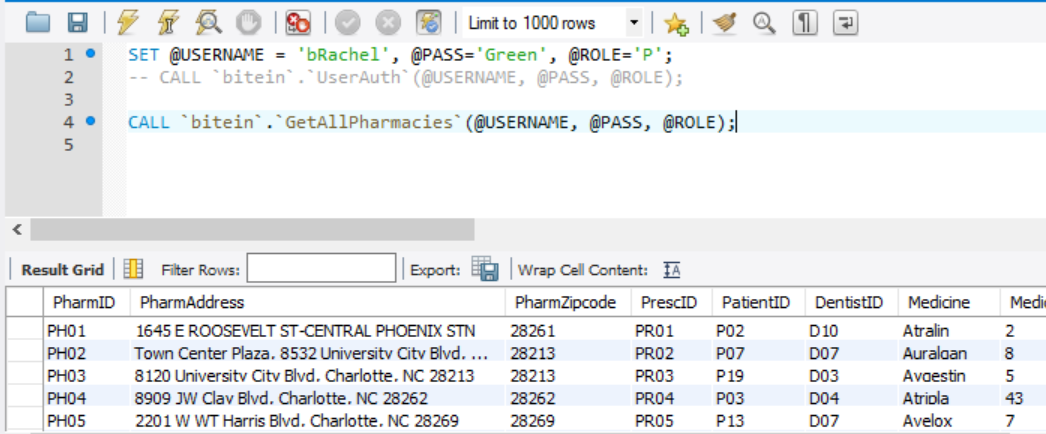
*SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = @message\_text;*

*END IF;*

*END IF;*

*END$$*

*DELIMITER ;*



* **Admin can see Highest earning dentist**

*DELIMITER $$*

*CREATE DEFINER=`root`@`localhost` PROCEDURE `HighestEarningDentist`(IN `USERNAME` VARCHAR(64), IN `PASS` VARCHAR(20), IN `ROLE` VARCHAR(5))*

*BEGIN*

*CALL `bitein`.`UserAuth`(@USERNAME, @PASS, @ROLE);*

*IF @User\_exists > 0 THEN*

*IF @ROLE='Adm' THEN*

*select d.DentistID, d.FirstName, d.LastName, sum(BillingAmt) as TotEarning from billinginfo as b join appointmentdetails as a*

*on b.AppID=a.AppID join dentist as d on d.DentistID = a.DentistID*

*group by(DentistID) order by TotEarning DESC;*

*ELSE*

*SET @message\_text = ('User not authorized');*

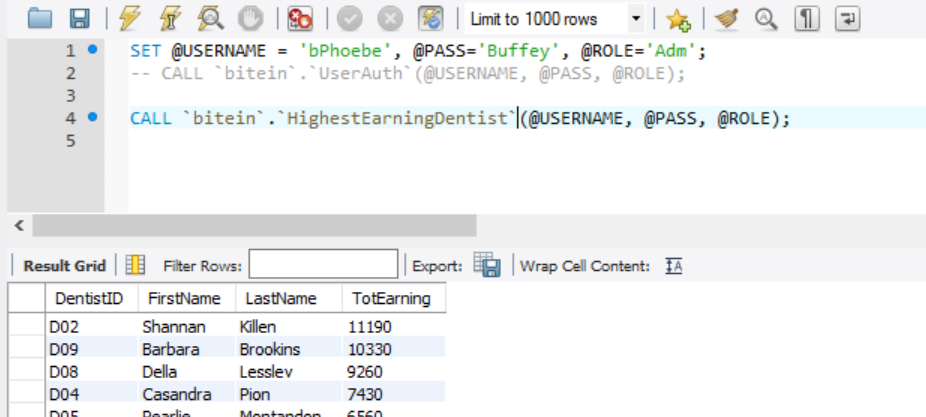
*SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = @message\_text;*

*END IF;*

*END IF;*

*END$$*

*DELIMITER ;*



* **Only Dentist and Admin can see patient issue details**

*DELIMITER $$*

*CREATE DEFINER=`root`@`localhost` PROCEDURE `PatientsWithWornFillings`(IN `USERNAME` VARCHAR(64), IN `PASS` VARCHAR(20), IN `ROLE` VARCHAR(5))*

*BEGIN*

*CALL `bitein`.`UserAuth`(@USERNAME, @PASS, @ROLE);*

*IF @User\_exists > 0 THEN*

*IF @ROLE='D' OR @ROLE='Adm' THEN*

*select distinct FirstName, LastName, SymptomName, DentistID from patient as p join prescription as pr*

*on p.PatientID=pr.PatientID join symptom s on s.SymptomID=pr.SymptomID*

*where SymptomName='worn fillings'*

*group by(dentistID);*

*ELSE*

*SET @message\_text = ('User not authorized');*

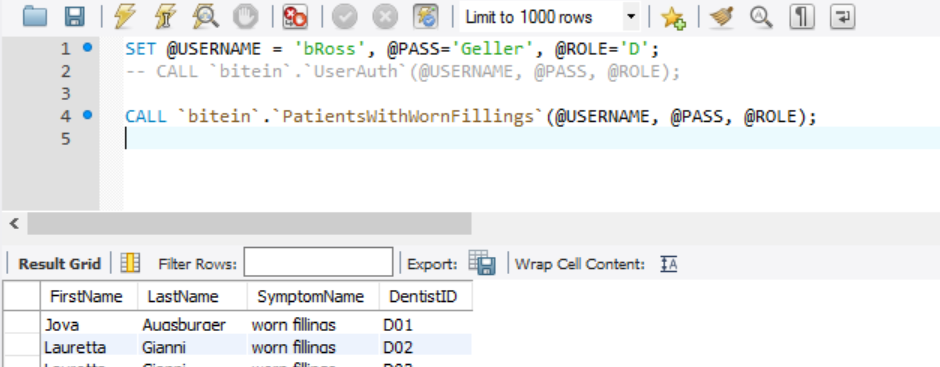
*SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = @message\_text;*

*END IF;*

*END IF;*

*END$$*

*DELIMITER ;*



* **Admin and Receptionist can see patient visits and frequency of their visits**

*DELIMITER $$*

*CREATE DEFINER=`root`@`localhost` PROCEDURE `PatientVisitFrequency`(IN `USERNAME` VARCHAR(64), IN `PASS` VARCHAR(20), IN `ROLE` VARCHAR(5))*

*BEGIN*

*CALL `bitein`.`UserAuth`(@USERNAME, @PASS, @ROLE);*

*IF @User\_exists > 0 THEN*

*IF @ROLE='R' OR @ROLE='Adm' THEN*

*select count(AppID) as TotAppointments, p.patientID, FirstName, LastName*

*from appointmentdetails a join patient p on a.patientID=p.PatientID*

*group by(patientID) order by TotAppointments DESC;*

*ELSE*

*SET @message\_text = ('User not authorized');*

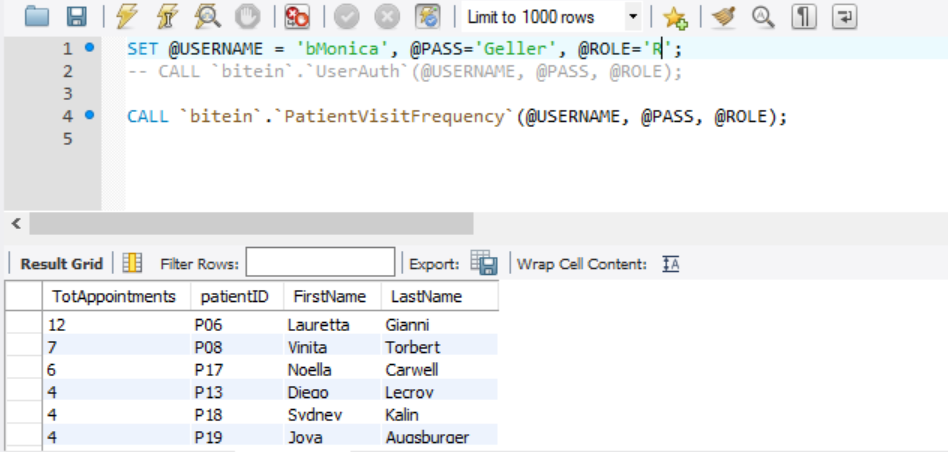
*SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = @message\_text;*

*END IF;*

*END IF;*

*END$$*

*DELIMITER ;*



* **Only Dentist and Admin can see dentist workload**

*DELIMITER $$*

*CREATE DEFINER=`root`@`localhost` PROCEDURE `WorkLoad`(IN `USERNAME` VARCHAR(64), IN `PASS` VARCHAR(20), IN `ROLE` VARCHAR(5))*

*BEGIN*

*CALL `bitein`.`UserAuth`(@USERNAME, @PASS, @ROLE);*

*IF @User\_exists > 0 THEN*

*IF @ROLE='D' OR @ROLE='Adm' THEN*

*select count(AppID) as TotAppointments, DentistID from appointmentdetails*

*group by(DentistID) order by TotAppointments ASC;*

*ELSE*

*SET @message\_text = ('User not authorized');*

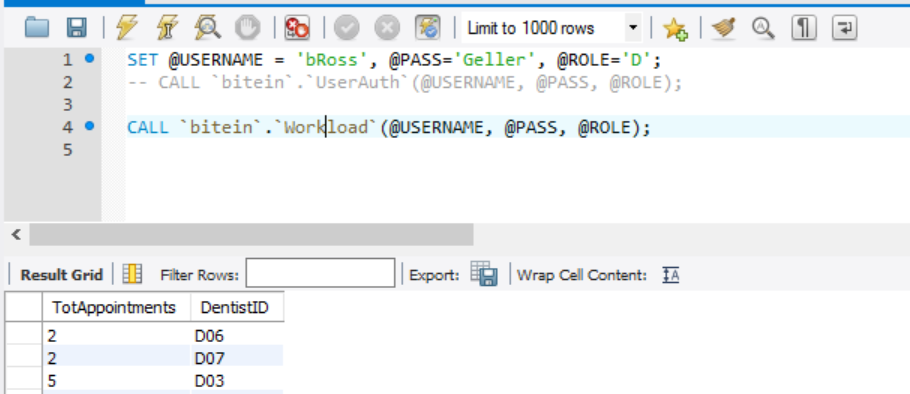
*SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = @message\_text;*

*END IF;*

*END IF;*

*END$$*

*DELIMITER ;*



* **Only pharmacist and admin can see the Medicine details**

*DELIMITER $$*

*CREATE DEFINER=`root`@`localhost` PROCEDURE `MedicinLog`(IN `USERNAME` VARCHAR(64), IN `PASS` VARCHAR(20), IN `ROLE` VARCHAR(5))*

*BEGIN*

*CALL `bitein`.`UserAuth`(@USERNAME, @PASS, @ROLE);*

*IF @User\_exists > 0 THEN*

*IF @ROLE='Ph' THEN*

*SELECT PrescID, PatientID, p.MedicineID, MedicineQty, m.MedicineName, Manufacturer, ExpiryDate*

*FROM*

*Medicine m JOIN prescription p*

*ON m.medicineID=p.medicineID*

*ELSE*

*SET @message\_text = ('User not authorized');*

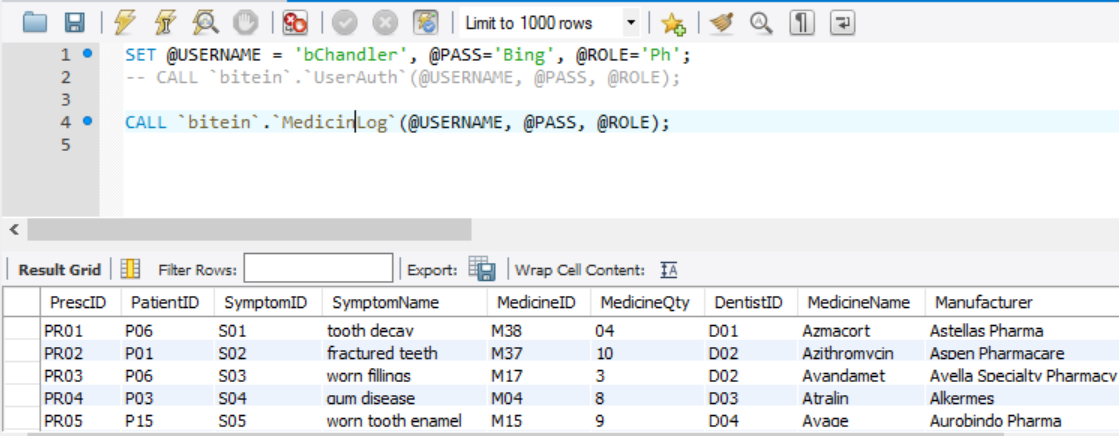
*SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = @message\_text;*

*END IF;*

*END IF;*

*END$$*

*DELIMITER ;*



Database Tables:

* Dentist(DentistID, FirstName, LastName, Gender, DOB, WorkEx, Contactno)
* Insurance( InsuranceID, Name, FromDate, ToDate)
* Patient(PatientID, FirstName, LastName, Gender, DOB, Address, ZipCode,Contactno)
* AppointmentDetails(AppID, PatientID, DentistID, AppDate, SessionSchedule, RID)
* BillingInfo(ReceiptID, AppID, PatientID, AppDate, BillingAmt, RID)
* MedHistory(HistoryID, PatientID, Age, Gender, Weight, Height, BMI, BP)
* Medicine(MedicineID, MedicineName, Manufacturer, ExpiryDate)
* Payment(TransactionID, PayMode, PayDate, PayAmt, Balance, ReceiptID)
* Symptom(SymptomID, SymptomName)
* Precription(PrescID VisitDate, PatientID, SymptomID, MedicineID, MedicineQty, MedicineName, DentistID)
* VisitingDetails(VisitID, Vdate, DentistID, PatientID, SymptomID, PrescriptionID)
* Pharmacy(PharmID, PharmAddress, PharmZipcode, PrescID, PatientID, DentistID, Medicine, MedicineQTY, Manufacturer)
* User(uname, pwd, role)

DB Script:

Attached separately with the report.

User Roles:

User Roles are defined in a user table which also contains usernames and passwords of the users.

D=Dentist

P=Patient

Ph=Pharmacist

R=Receptionist

Adm=DB Admin

*CREATE TABLE users  
(uname varchar(40) primary key,*

*pwd varchar(40),*

*role varchar(40));*

*INSERT INTO `bitein`.`users` (`uname`, `pwd`, `role`) VALUES ('bRoss', 'Geller', 'D');*

*INSERT INTO `bitein`.`users` (`uname`, `pwd`, `role`) VALUES ('bMonica', 'Geller', 'R');*

*INSERT INTO `bitein`.`users` (`uname`, `pwd`, `role`) VALUES ('bRachel', 'Green', 'P');*

*INSERT INTO `bitein`.`users` (`uname`, `pwd`, `role`) VALUES ('bChandler', 'Bing', 'Ph');*

*INSERT INTO `bitein`.`users` (`uname`, `pwd`, `role`) VALUES ('bPhoebe', 'Buffey', 'Adm');*

Views:

Views are used for ease of access for different user roles. Below are few views designed for the database.

* Medicine Details in Prescription – For the dentist

*CREATE VIEW `MedicineDetailsInPrescription` AS*

*SELECT PrescID, PatientID, p.SymptomID, SymptomName, p.MedicineID, MedicineQty, DentistID,*

*m.MedicineName, Manufacturer, ExpiryDate*

*FROM*

*Medicine m JOIN prescription p*

*ON m.medicineID=p.medicineID*

*JOIN Symptom s*

*ON s.SymptomID=p.SymptomID;*

* Patient Details visiting the clinic – For the dentist

*CREATE VIEW `PatientDetails` AS*

*SELECT P.PatientID, FirstName, LastName, p.Gender, DOB, Contactno,*

*Age, Weight, Height, BMI, BP*

*FROM patient p JOIN medhistory h*

*ON p.paatientID=h.patientID;;*

*USE `bitein`;*

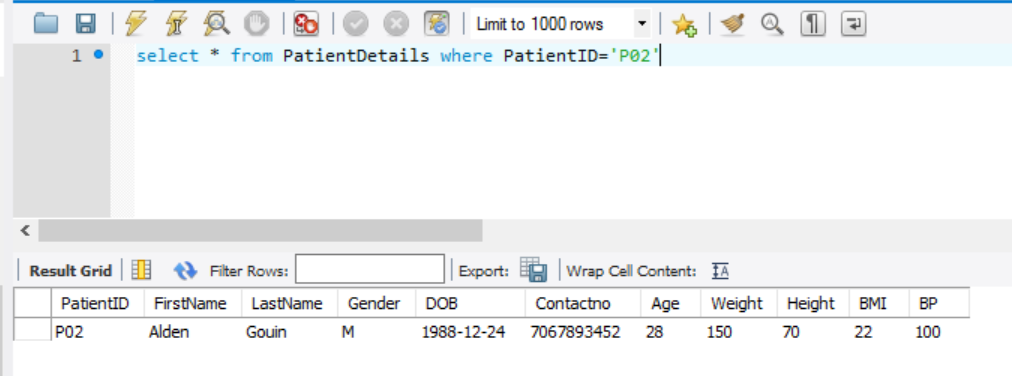
*CREATE OR REPLACE VIEW `PatientDetails` AS*

*SELECT P.PatientID, FirstName, LastName, Gender, DOB, Contactno,*

*Age, Gender, Weight, Height, BMI, BP*

*FROM patient p JOIN medhistory h*

*ON p.patientID=h.patientID;*



* Patient Billing and Payment Details – For the receptionist

*CREATE VIEW `CompleteBillingAndPaymentDetails` AS*

*SELECT b.PatientID, b.ReceiptID, AppID, AppDate, BillingAmt,*

*TransactionID, PayMode, PayDate, PayAmt, Balance*

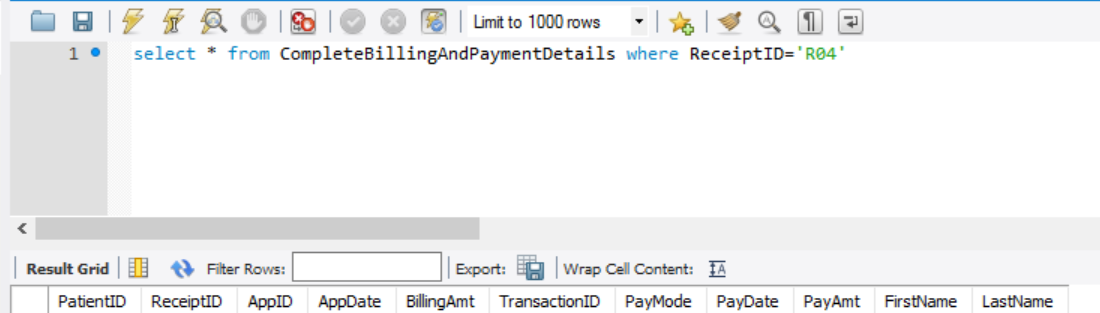
*FirstName, LastName FROM*

*BillingInfo b JOIN Payment p*

*ON b.ReceiptID = p.ReceiptID*

*JOIN Patient PT*

*ON pt.PatientID = b.patientID;*



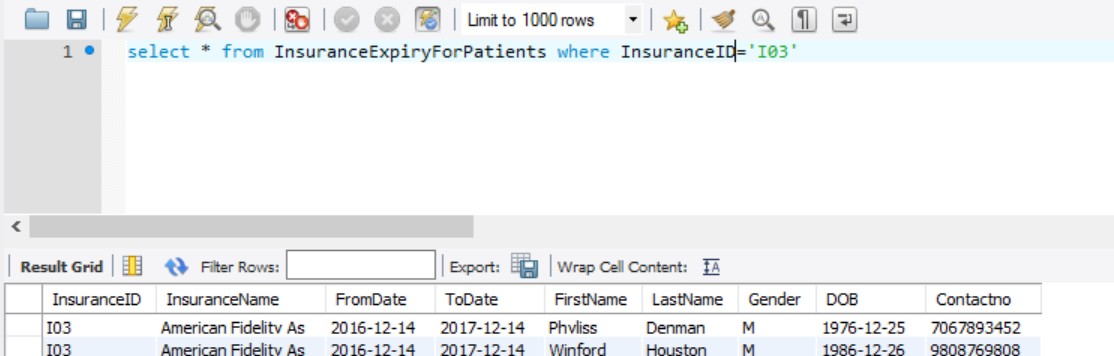
* Insurance Expiry Details for Patients

*CREATE VIEW `InsuranceExpiryForPatients` AS*

*select i.InsuranceID, InsuranceName, FromDate, ToDate, FirstName, LastName, Gender, DOB, Contactno*

*from Insurance i join Patient p*

*on i.InsuranceID = p.InsuranceID;*



Indexes:

All foreign keys and primary keys are indexed to help speed up the retrieval of data from tables.

Audit Trail using Triggers:

Audit Trail is a way of tracking nay changes into the database. Frequently edited items like

* **VISITING DETAILS:**

*Creating a audit table for VisitingDetails Table:*

*create table Visitaudit\_log*

*(auditid int NOT NULL Auto\_Increment primary key ,*

*PatientID varchar(20), DentistID varchar(20),*

*VisitID varchar(20) , SymptomID varchar(20),*

*modifiedby varchar(40),*

*Ts DATETIME NOT NULL DEFAULT CURRENT\_TIMESTAMP);*

*This trigger will insert values into audit table after delete operation*

*Delimiter $*

*create trigger Visit\_Delete*

*After Delete on visitingdetails*

*for each row*

*begin*

*insert into Visitaudit\_log*

*Values (auditid,Old.PatientID,Old.DentistID,Old.VisitID,Old.SymptomID, current\_user(), current\_timestamp());*

*end;*

*$*

*This trigger will insert values into audit table after update operation*

*Delimiter $*

*create trigger Visit\_Update*

*After update on visitingdetails*

*for each row*

*begin*

*insert into Visitaudit\_log*

*Values (auditid,Old.PatientID,Old.DentistID,Old.VisitID,Old.SymptomID, current\_user(), current\_timestamp());*

*end;*

*$*

*This trigger will insert values into audit table before insert operation*

*Delimiter $*

*create trigger Visit\_Insert*

*After insert on visitingdetails*

*for each row*

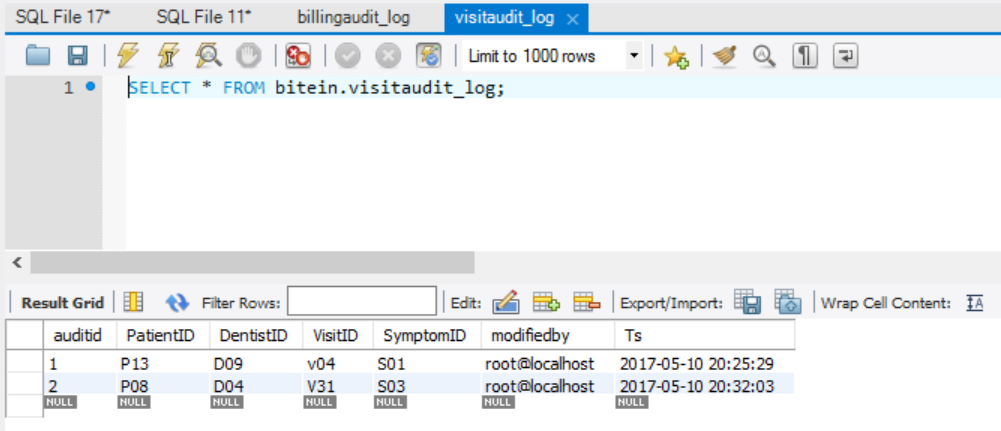
*begin*

*insert into Visitaudit\_log*

*Values (auditid,new.PatientID,new.DentistID,new.VisitID,new.SymptomID, current\_user(), current\_timestamp());*

*end;*

*$*



**BILLINGINFO:**

*Creating a audit table*

*(auditid int NOT NULL Auto\_Increment primary key ,*

*ReceiptID varchar(20),*

*AppID varchar(20) ,*

*patientID varchar(20) ,*

*AppDate datetime ,*

*BillingAmt decimal(10,0) ,*

*RID varchar(20),*

*modifiedby varchar(40),*

*Ts DATETIME NOT NULL DEFAULT CURRENT\_TIMESTAMP);*

*This trigger will insert values into audit table after delete operation*

*Delimiter $*

*create trigger Billing\_Delete*

*After Delete on billinginfo*

*for each row*

*begin*

*insert into billingaudit\_log*

*Values (auditid,Old.ReceiptID,Old.AppID,Old.PatientID,Old.AppDate,Old.BillingAmt,Old.RID, current\_user(), current\_timestamp());*

*end;*

*$*

*This trigger will insert values into audit table after update operation*

*Delimiter $*

*create trigger Billing\_Update*

*After update on billinginfo*

*for each row*

*begin*

*insert into billingaudit\_log*

*Values (auditid,Old.ReceiptID,Old.AppID,Old.PatientID,Old.AppDate,Old.BillingAmt,Old.RID, current\_user(), current\_timestamp());*

*end;*

*$*

*This trigger will insert values into audit table before insert operation*

*Delimiter $*

*create trigger Billing\_Insert*

*After insert on billinginfo*

*for each row*

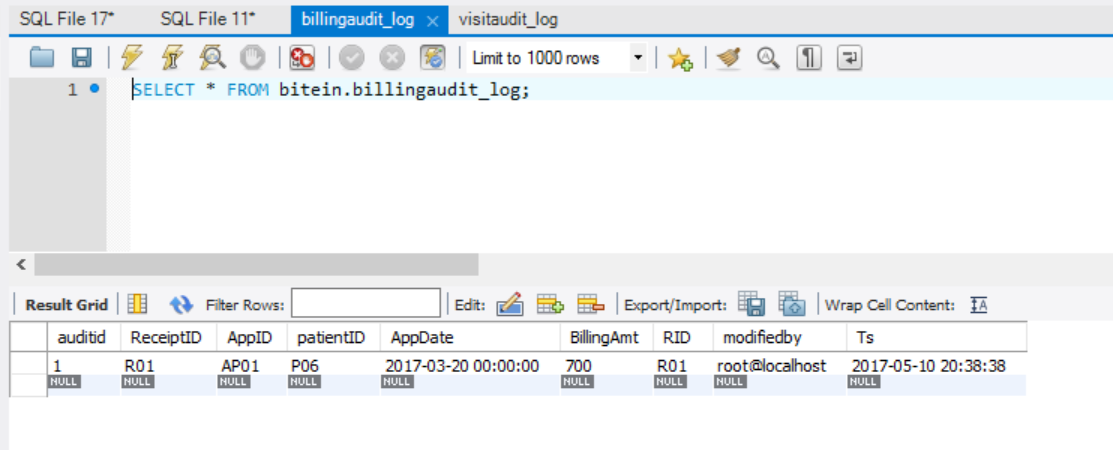
*begin*

*insert into billingaudit\_log*

*Values (auditid,new.ReceiptID,new.AppID,new.PatientID,new.AppDate,new.BillingAmt,new.RID, current\_user(), current\_timestamp());*

*end;*

*$*



**PRESCRIPTION:**

*Creating audit table*

*create table Presc\_log*

*(auditid int NOT NULL Auto\_Increment primary key ,*

*PrescID varchar(20),*

*VisitDate date ,*

*PatientID varchar(20) ,*

*SymptomID varchar(20) ,*

*MedicineID varchar(20) ,MedicineQty varchar(20) ,*

*MedicineName varchar(40) ,*

*DentistID varchar(20),*

*modifiedby varchar(40),*

*Ts DATETIME NOT NULL DEFAULT CURRENT\_TIMESTAMP);*

*This trigger will insert values into audit table after delete operation*

*Delimiter $*

*create trigger Presc\_Delete*

*After Delete on prescription*

*for each row*

*begin*

*insert into presc\_log*

*Values (auditid,Old.PrescID,Old.VisitDate,Old.PatientID,Old.SymptomID,Old.MedicineID,Old.MedicineQty, Old.MedicineName, Old.DentistID, current\_user(), current\_timestamp());*

*end;*

*$*

*This trigger will insert values into audit table after update operation*

*Delimiter $*

*create trigger Presc\_Update*

*After update on prescription*

*for each row*

*begin*

*insert into presc\_log*

*Values (auditid,Old.PrescID,Old.VisitDate,Old.PatientID,Old.SymptomID,Old.MedicineID,Old.MedicineQty, Old.MedicineName, Old.DentistID, current\_user(), current\_timestamp());*

*end;*

*$*

*This trigger will insert values into audit table before insert operation*

*Delimiter $*

*create trigger Presc\_Insert*

*After insert on prescription*

*for each row*

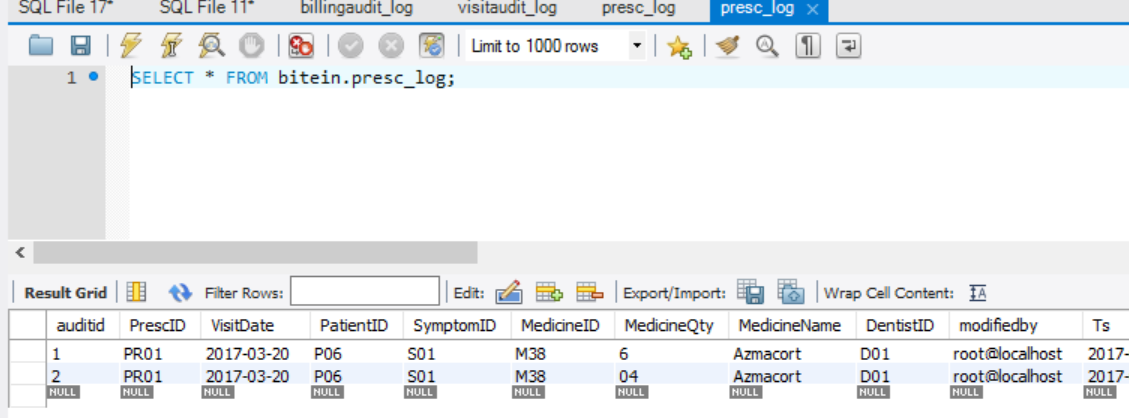
*begin*

*insert into presc\_log*

*Values (auditid,new.PrescID,new.VisitDate,new.PatientID,new.SymptomID,new.MedicineID,new.MedicineQty,new.MedicineName, new.DentistID, current\_user(), current\_timestamp());*

*end;*

*$*



**APPOINTMENTDETAILS:**

*Creating Audit table:*

*create table Appt\_log*

*(auditid int NOT NULL Auto\_Increment primary key ,*

*AppID varchar(20),*

*patientID varchar(20) ,*

*DentistID varchar(20) ,*

*Appdate datetime ,*

*RID varchar(20),*

*Modifiedby varchar(40),*

*Ts DATETIME NOT NULL DEFAULT CURRENT\_TIMESTAMP);*

*This trigger will insert values into audit table after delete operation*

*Delimiter $*

*create trigger Appt\_Delete*

*After Delete on appointmentdetails*

*for each row*

*begin*

*insert into Appt\_log*

*Values (auditid,Old.AppID,Old.PatientID,Old.DentistID, Old.AppDate,Old.RID, current\_user(), current\_timestamp());*

*end;*

*$*

*This trigger will insert values into audit table after update operation*

*Delimiter $*

*create trigger Appt\_Update*

*After update on appointmentdetails*

*for each row*

*begin*

*insert into appt\_log*

*Values (auditid,Old.AppID,Old.PatientID,Old.DentistID, Old.AppDate,Old.RID, current\_user(), current\_timestamp());*

*end;*

*$*

*This trigger will insert values into audit table before insert operation*

*Delimiter $*

*create trigger Appt\_Insert*

*After insert on appointmentdetails*

*for each row*

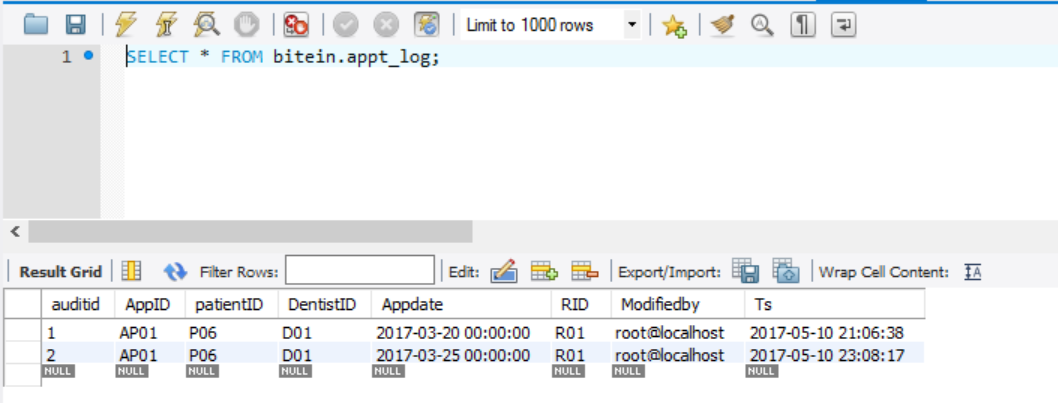
*begin*

*insert into appt\_log*

*Values (auditid,new.AppID,new.PatientID,new.DentistID, new.AppDate,new.RID, current\_user(), current\_timestamp());*

*end;*

*$*



**PAYMENT:**

*Creating audit table:*

*create table Pay\_log*

*(auditid int NOT NULL Auto\_Increment primary key ,*

*TransactionID varchar(20),*

*PayMode varchar(20) ,*

*Paydate date ,*

*PayAmt decimal(10,0),*

*Balance decimal(10,0),*

*ReceiptID varchar(20),*

*modifiedby varchar(40),*

*Ts DATETIME NOT NULL DEFAULT CURRENT\_TIMESTAMP);*

*This trigger will insert values into audit table after delete operation*

*Delimiter $*

*create trigger Pay\_Delete*

*After Delete on payment*

*for each row*

*begin*

*insert into pay\_log*

*Values (auditid,Old.TransactionID,Old.PayMode,Old.Paydate, Old.PayAmt,Old.Balance, Old.ReceiptID, current\_user(), current\_timestamp());*

*end;*

*$*

*This trigger will insert values into audit table after update operation*

*Delimiter $*

*create trigger Pay\_Update*

*After update on payment*

*for each row*

*begin*

*insert into pay\_log*

*Values (auditid,Old.TransactionID,Old.PayMode,Old.Paydate, Old.PayAmt,Old.Balance, Old.ReceiptID, current\_user(), current\_timestamp());*

*end;*

*$*

*This trigger will insert values into audit table before insert operation*

*Delimiter $*

*create trigger Pay\_Insert*

*After insert on payment*

*for each row*

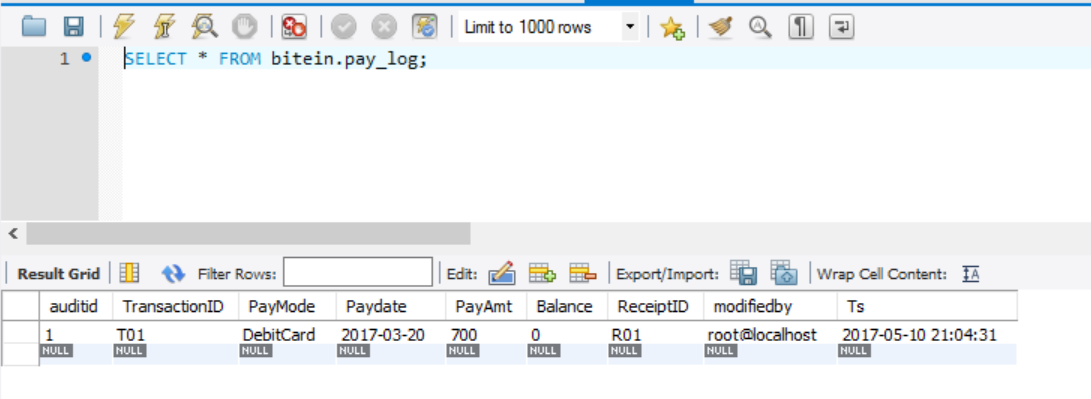
*begin*

*insert into pay\_log*

*Values (auditid,new.TransactionID,new.PayMode,new.Paydate, new.PayAmt,new.Balance, new.ReceiptID, current\_user(), current\_timestamp());*

*end;*

*$*



Conclusion:

Additional features were developed in the database system - stored procedures, triggers, indexes, views and user authentication and role based authentication - were implemented for the database.

This facilitated more robust and efficient application development and to support security, privacy, audit trail and other requirements of the database created for BiteIn Clinic.