A Database for an

Electronic Medical Record System

Student Clinic Module

Project-2

Group Members

Abhishek Deshpande

Harika Katragadda

Suyash Nande

**Requirement Specification**

Student Clinic is the database project designed to facilitate student health record in a university. This system will be used by the doctor and clinic staff in the student health center of the university. It is developed to improve the clinic management and automates the workflow that happens in a student clinic. It will make it easier for the retrieval of history information of the student.

In this project, we have added few more tables to enhance the database and make it more efficient. We have added appointment table and receptionist table. When the student will visit on clinic, he will connect with the receptionist where they will be asked to provide their name, date of birth, contact number, address and type of problems. Student will make registration first. If the student never registered before, student information is collected and stored in the database. Now, the receptionist creates an appointment for the student and student is then assigned to the doctor, which may yield to diagnosis of the disease based on the problem category.

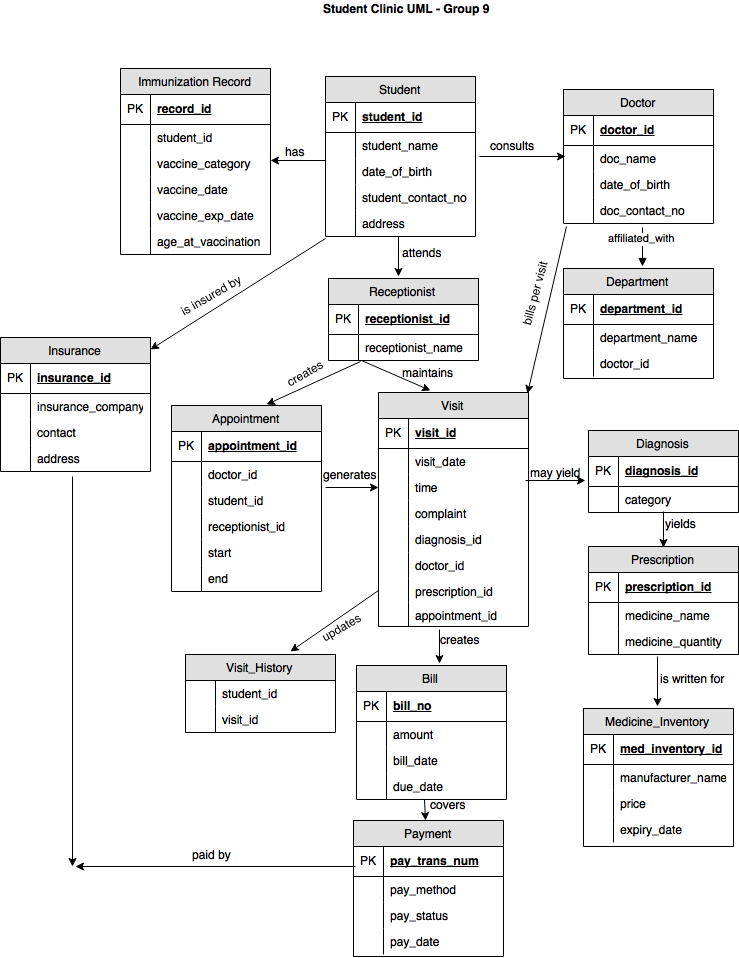
This diagnosis may further yield to treatment and doctor will then prescribe medicines which contains the medicine name, duration and quantity. Once the student gets the treatment, the doctor will send the report to the clinical staff.

The staff will view the report and complete the student record. After that, the staff will prepare the bills for the student. Student will also provide with an option for making payments. They can make their payments either by self or by medical insurance. Student will also have their immunization record maintained by clinical staff. It is easy for the management to maintain record about the student, the time for retrieving the information needed will be less compare to the manual. Then the staff will update the visit information and the student record will be kept in database.

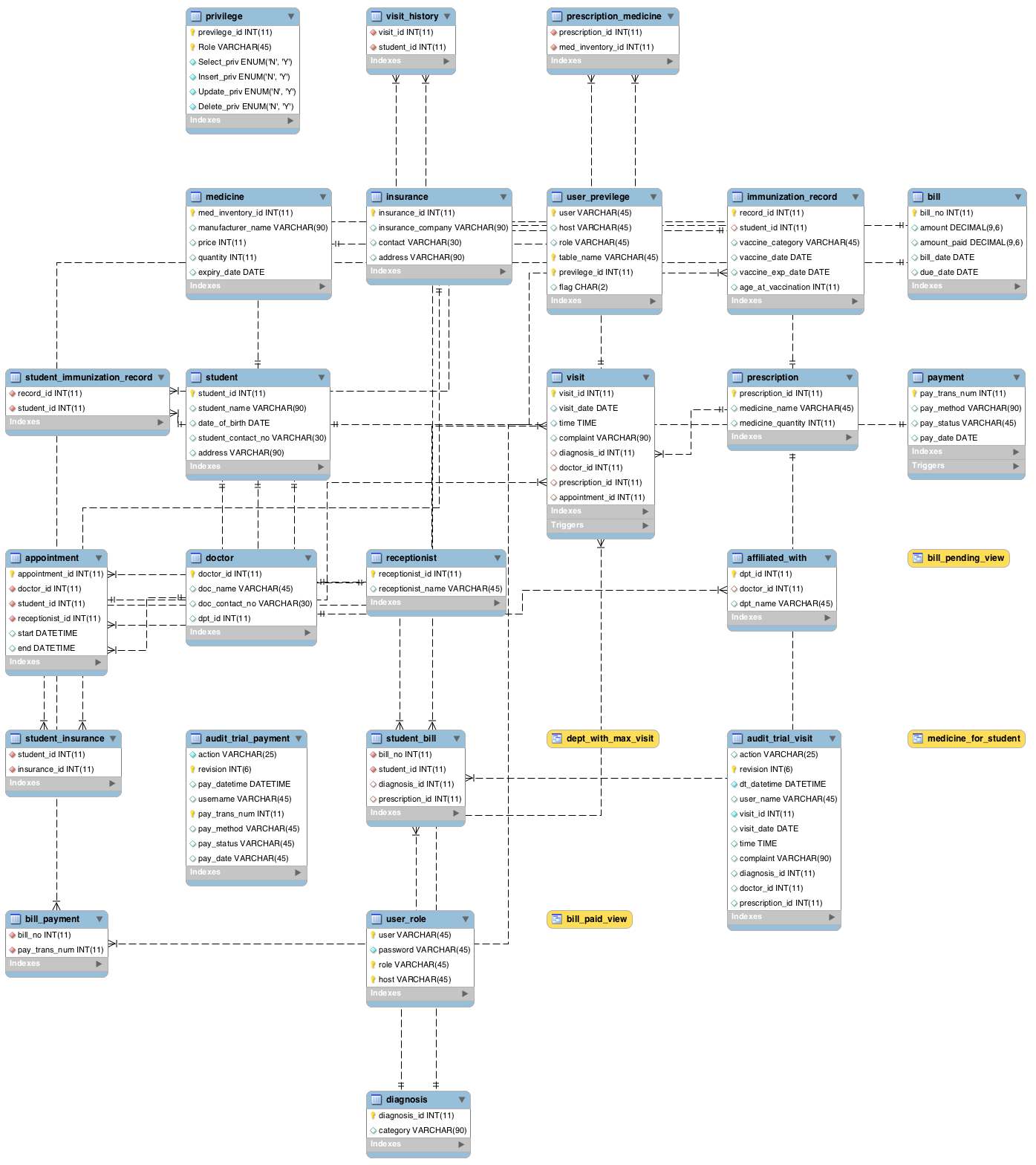
**Features**

* We have implemented stored procedures in the form of API that allows an EMR system developer to work with the database without having to write SQL statements.
* We have also provided security feature wherein only the authorized users will be able to access the database.
* In addition to that, we have also implemented privacy features in the form of role based access control to limit the functions the user can perform.
* To achieve the regulatory requirements, we have added audit trial feature for the transactional tables wherein the logs for every change to the database made are captured.
* To optimize the performance further, we have implemented the usage of views indexes and views.

**UML Data Model**



**ER Diagram**



**Procedures for database insertion and modification**

**Insert Procedure**

*To insert new doctor data*

DELIMITER $$

CREATE PROCEDURE `NewDoctor\_insert`

(IN DOCTOR\_ID varchar(12),

IN DOCTOR\_NAME varchar(45),

IN DOCTOR\_CONTACT varchar(30))

BEGIN

INSERT INTO doctor(doctor\_id,doc\_name,doc\_contact\_no)

values(DOCTOR\_ID,DOCTOR\_NAME,DOCTOR\_CONTACT);

END$$

DELIMITER ;

CALL `student\_clinic\_db\_final`.`NewDoctor\_insert`(‘108’, ‘Dr.Jake’, ‘9803655355’);

*To insert new student data*

DELIMITER $$

CREATE PROCEDURE `NewStudent\_INSERT`(IN Student\_ID varchar(12),

IN Student\_Name varchar(90),

IN Student\_DOB date,

IN Student\_Contact varchar(30),

IN Student\_Address varchar(90)

)

BEGIN

INSERT INTO student values(Student\_ID,Student\_Name,Student\_DOB,Student\_Contact,Student\_Address);

END$$

DELIMITER ;

SET @Student\_ID='55';

SET @Student\_Name='Harika';

SET @Student\_DOB='1993-01-02';

SET @Student\_Contact='9803650355';

SET @Student\_Address='UTD,NC';

CALL `student\_clinic\_db\_final`.`NewStudent\_INSERT`(@Student\_ID,@Student\_Name,@Student\_DOB,@Student\_Contact,@Student\_Address);

*To insert new insurance company*

DELIMITER $$

CREATE PROCEDURE `InsuranceCompany\_insert`(IN COMPANY\_ID varchar(12),

IN COMPANY\_NAME varchar(45),

IN COMPANY\_CONTACT varchar(30),

IN COMPANY\_ADDRESS varchar(90)

)

BEGIN

INSERT INTO insurance(insurance\_id,insurance\_company,contact,address) values(COMPANY\_ID,COMPANY\_NAME,COMPANY\_CONTACT,COMPANY\_ADDRESS);

END$$

DELIMITER ;

CALL `student\_clinic\_db\_final`.`InsuranceCompany\_insert`('50','ISO','7645423457','ABCD');

*To insert the visit information*

DELIMITER $$

CREATE PROCEDURE `VISIT\_insert`

(IN VISIT\_ID varchar(11),

IN VISIT\_DATE date,

IN VISIT\_TIME time,

IN VISIT\_COMPLAINT varchar(90),

IN DIAGNOSIS\_ID varchar(11),

IN DOC\_ID varchar(12),

IN PRESCR\_ID varchar(11)

)

BEGIN

INSERT INTO visit(visit\_id,visit\_date,time,complaint,diagnosis\_id,doctor\_id,prescription\_id)

values(VISIT\_ID,VISIT\_DATE,VISIT\_TIME,VISIT\_COMPLAINT,DIAGNOSIS\_ID,DOC\_ID,PRESCR\_ID);

END$$

DELIMITER ;

CALL `student\_clinic\_db\_final`.`VISIT\_insert`('6','2017-05-03','01:00:00','Throat infection','14','101','12');

*To enter new diagnosis record*

DELIMITER $$

CREATE PROCEDURE `New\_diagnosis`

(IN DIAGNOSIS\_ID varchar(12),IN DIAGNOSIS\_CATEGORY varchar(45))

BEGIN

INSERT INTO diagnosis(diagnosis\_id,category) values(DIAGNOSIS\_ID,DIAGNOSIS\_CATEGORY);

END$$

DELIMITER ;

CALL `student\_clinic\_db\_final`.`New\_diagnosis`('16','Throatinfection');

*To create new bills:*

DELIMITER $$

CREATE PROCEDURE `Newbill\_insert`(IN BILL\_ID varchar(11),

IN BILL\_AMOUNT decimal(9,6),

IN CREATE\_DATE date,

IN BILL\_DUE date

)

BEGIN

INSERT INTO bill(bill\_no,amount,bill\_date,due\_date)

values(BILL\_ID,BILL\_AMOUNT,CREATE\_DATE,BILL\_DUE);

END$$

DELIMITER ;

CALL `student\_clinic\_db\_final`.`Newbill\_insert`('1',' 555','2017-04-03','2017-05-03');

CALL `student\_clinic\_db\_final`.`Newbill\_insert`('2',' 550','2017-04-03','2017-05-04');

CALL `student\_clinic\_db\_final`.`Newbill\_insert`('3',' 111','2017-03-03','2017-04-05');

CALL `student\_clinic\_db\_final`.`Newbill\_insert`('4',' 220','2017-05-03','2017-06-03');

*To create prescription:*

DELIMITER $$

CREATE PROCEDURE `Prescription\_insert`(IN PRESCRIP\_ID varchar(11),

IN MED\_QTY int(11)

)

BEGIN

INSERT INTO prescription(prescription\_id,medicine\_quantity) values(PRESCRIP\_ID,MED\_QTY);

END$$

DELIMITER ;

CALL `student\_clinic\_db\_final`.`Prescription\_insert`('14','50');

CALL `student\_clinic\_db\_final`.`Prescription\_insert`('15','75');

CALL `student\_clinic\_db\_final`.`Prescription\_insert`('16','95');

*Insert into prescription medicine:*

DELIMITER $$

CREATE PROCEDURE `Prescription\_med\_insert`(IN PRESCRIP\_ID varchar(11),

IN MED\_INVTRY\_ID varchar(11))

BEGIN

INSERT INTO prescription\_medicine(prescription\_id,med\_inventory\_id) values(PRESCRIP\_ID,MED\_INVTRY\_ID);

END$$

DELIMITER ;

CALL `student\_clinic\_db\_final`.`Prescription\_med\_insert`('45','897');

CALL `student\_clinic\_db\_final`.`Prescription\_med\_insert`('55','900');

CALL `student\_clinic\_db\_final`.`Prescription\_med\_insert`('34','789');

*To insert into payment table:*

DELIMITER $$

CREATE PROCEDURE `Payment\_insert`(IN TRANSCTION\_NUM varchar(11),

IN METHOD varchar(90),

IN PAYSTATUS varchar(45),

IN PAYDATE date

)

BEGIN

INSERT INTO Payment(pay\_trans\_num,pay\_method,pay\_status,pay\_date)

values(TRANSCTION\_NUM,METHOD,PAYSTATUS,PAYDATE);

END$$

DELIMITER ;

CALL `student\_clinic\_db\_final`.`Payment\_insert`('107','SELF','PAID','2017-11-09');

CALL `student\_clinic\_db\_final`.`Payment\_insert`('108','INSURANCE','PAID','2017-01-12');

CALL `student\_clinic\_db\_final`.`Payment\_insert`('109','SELF','PENDING','2017-05-17');

CALL `student\_clinic\_db\_final`.`Payment\_insert`('110','INSURANCE','PENDING','2017-08-01');

*Student insurance insert:*

DELIMITER $$

CREATE PROCEDURE `Studentinsurance\_insert`(IN STUD\_ID varchar(7), IN INSURNC\_ID varchar(90))

BEGIN

INSERT INTO student\_insurance(student\_id,insurance\_id)

values(STUD\_ID,INSURNC\_ID);

END$$

DELIMITER ;

CALL `student\_clinic\_db\_final`.`Studentinsurance\_insert`('10','2');

CALL `student\_clinic\_db\_final`.`Studentinsurance\_insert`('55','2');

*To insert billpayment:*

DELIMITER $$

CREATE PROCEDURE `BillPayment\_insert`(IN BILL\_ID varchar(11),

IN TRANS\_NUM varchar(11))

BEGIN

INSERT INTO bill\_payment(bill\_no,pay\_trans\_num)

values(BILL\_ID,TRANS\_NUM);

END$$

DELIMITER ;

CALL `student\_clinic\_db\_final`.`BillPayment\_insert`('1','106');

CALL `student\_clinic\_db\_final`.`BillPayment\_insert`('2','107');

CALL `student\_clinic\_db\_final`.`BillPayment\_insert`('3','108');

CALL `student\_clinic\_db\_final`.`BillPayment\_insert`('4','109');

***Update Procedures:***

***Given student id and bill amount to be updated:***

DELIMITER $$

CREATE PROCEDURE `Update\_BillAmount`(IN STUD\_ID varchar(12),

IN AMOUNT\_UPDATE decimal(9,6)

)

BEGIN

UPDATE bill set amount=AMOUNT\_UPDATE where bill\_no

in(SELECT bill\_no from student\_bill where student\_id=STUD\_ID);

END$$

DELIMITER ;

CALL `student\_clinic\_db\_final`.`Update\_BillAmount`('1','555');

***Given student id and update the payment status:***

DELIMITER $$

CREATE PROCEDURE `Update\_PaymentStatus`(IN STUD\_ID varchar(12),

IN STATUS\_UPDATE varchar(45)

)

BEGIN

UPDATE payment set pay\_status=STATUS\_UPDATE where pay\_trans\_num

in(SELECT pay\_trans\_num from bill\_payment where bill\_no

in(SELECT bill\_no from student\_bill where student\_id=STUD\_ID));

END$$

DELIMITER ;

CALL `student\_clinic\_db\_final`.`Update\_PaymentStatus`('1','Paid');

***Updating insurance company id for a given student\_id:***

DELIMITER $$

CREATE PROCEDURE `Update\_InsuranceCompany`(IN STUD\_ID varchar(12),

IN Company\_UPDATE varchar(7))

BEGIN

UPDATE student\_insurance set insurance\_id=Company\_UPDATE

where student\_id=STUD\_ID;

END$$

DELIMITER ;

CALL `student\_clinic\_db\_final`.`Update\_InsuranceCompany`('1','50');

**Delete Procedure**

***To delete student record***

DELIMITER $$

CREATE PROCEDURE `Del\_Student`

(IN DEL\_STUDENT\_ID varchar(12))

BEGIN

delete from student where student\_id= DEL\_STUDENT\_ID;

END$$

DELIMITER ;

CALL `student\_clinic\_db\_final`.`Del\_Student`(‘3’);

***To delete diagnosis record***

DELIMITER $$

CREATE PROCEDURE `DEL\_diagnosis\_data`

(IN Del\_DIAGNOSIS\_ID varchar(12))

BEGIN

DELETE FROM diagnosis WHERE diagnosis\_id = Del\_DIAGNOSIS\_ID;

END$$

DELIMITER ;

CALL `student\_clinic\_db\_final`.`DEL\_diagnosis\_data`(‘903’);

***To delete doctor data***

SELECT \* FROM student\_clinic\_db\_final.doctor;

DELIMITER $$

CREATE PROCEDURE `Delete\_Doctor`

(IN Del\_Doc varchar(12))

BEGIN

DELETE FROM doctor WHERE doctor\_id = Del\_Doc;

END$$

DELIMITER ;

CALL `student\_clinic\_db\_final`.`Delete\_Doctor`(‘202’);

***To delete insurance company data***

DELIMITER $$

CREATE PROCEDURE `Del\_InsuranceCompany`

(IN DEL\_COMPANY\_ID varchar(12))

BEGIN

DELETE FROM insurance where insurance\_id = DEL\_COMPANY\_ID;

END$$

DELIMITER ;

CALL `student\_clinic\_db\_final`.`Del\_InsuranceCompany`(‘5’);

***To delete medicine record***

DELIMITER $$

CREATE PROCEDURE `Delete\_medicine`(IN Del\_Med varchar(12))

BEGIN

DELETE FROM medicine WHERE med\_inventory\_id = Del\_Med;

END$$

DELIMITER ;

CALL `student\_clinic\_db\_final`.`Delete\_medicine`(‘321’);

APPLICATION PROGRAMMING INTERFACE – STORED PROCEDURES

**Create an application programming interface (API) using stored procedures that allows an**

**EMR system developer to work with the database without having to write SQL statements**

**Ans:** To demonstrate the API we have created stored procedures for following scenario’s.

1. Select all the students having insurance

DELIMITER $$

CREATE PROCEDURE `student\_having\_insurance`()

BEGIN

select s.student\_name, i.insurance\_company

from insurance i

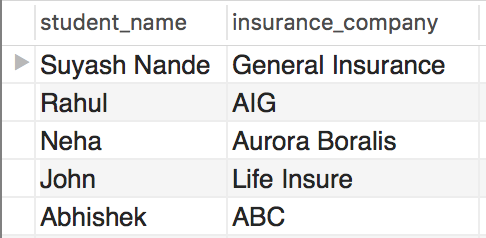
join student\_insurance using(insurance\_id)

join student s using (student\_id);

END$$

DELIMITER;

CALL `student\_clinic\_db\_final`.`student\_having\_insurance`();



2. Get all the visit granted by Dr. Rahul

DELIMITER $$

CREATE PROCEDURE `ALL\_VISIT\_FOR\_DR\_RAHUL`(IN DOC\_NAME\_INPUT

varchar(255))

BEGIN

select visit.visit\_id, visit.visit\_date, doctor.doctor\_id

from visit

join doctor using(doctor\_id)

join diagnosis using(diagnosis\_id)

join prescription using(prescription\_id)

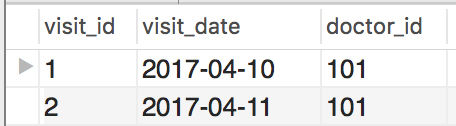
where doctor.doc\_name=DOC\_NAME\_INPUT;

END$$

DELIMITER ;

set @DOC\_NAME\_INPUT='Dr. Rahul';

CALL `student\_clinic\_db\_final`.`ALL\_VISIT\_FOR\_DR\_RAHUL`(@DOC\_NAME\_INPUT);



3. Get the bill details and payment number of Suyash Nande

DELIMITER $$

CREATE PROCEDURE `BILL\_DETAILS\_SUYASH`(IN STU\_NAME varchar(255))

BEGIN

select student.student\_name, student.student\_id, bill.bill\_no, bill.bill\_date,

payment.pay\_trans\_num, payment.pay\_date

from bill

join bill\_payment using (bill\_no)

join payment using (pay\_trans\_num)

join student\_bill using (bill\_no)

join student using (student\_id)

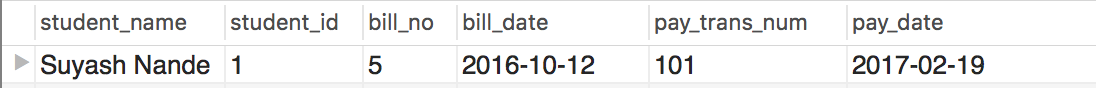
where student.student\_name=STU\_NAME;

END$$

DELIMITER ;

set @STU\_NAME='Suyash Nande';

CALL `student\_clinic\_db\_final`.`BILL\_DETAILS\_SUYASH`(@STU\_NAME);



4. Get the names and contact of all doctors who have visited at least 1 student

DELIMITER $$

CREATE PROCEDURE `DOC\_VISIT\_HISTORY`(IN NO\_OF\_VISITS int)

BEGIN

select doctor.doc\_name, doctor.doc\_contact\_no

from doctor

where doctor\_id in (select doctor\_id

from visit

join visit\_history using (visit\_id)

group by visit.doctor\_id

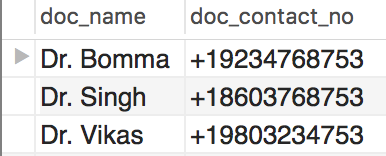
having max(visit\_history.student\_id)>=NO\_OF\_VISITS);

END$$

DELIMITER ;

set @NO\_OF\_VISITS=2;

CALL `student\_clinic\_db\_final`.`DOC\_VISIT\_HISTORY`(@NO\_OF\_VISITS);



5. Find the names of the students and their address who have visited to more than one doctor.

DELIMITER $$

CREATE PROCEDURE `STU\_VISIT\_HISTORY`

(IN NO\_OF\_VISITS int)

BEGIN

select s.student\_name, s.address, count(v.doctor\_id)

from student s, visit v, visit\_history vh, doctor d

where v.visit\_id = vh.visit\_id

and vh.student\_id = s.student\_id

and v.doctor\_id = d.doctor\_id

group by s.student\_name,s.address

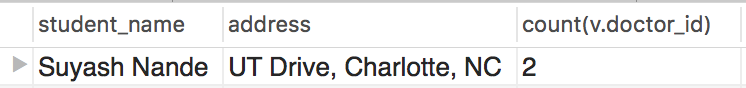
having count(v.doctor\_id) > NO\_OF\_VISITS;

END$$

DELIMITER ;

set @NO\_OF\_VISITS = 1;

CALL `student\_clinic\_db\_final`.`STU\_VISIT\_HISTORY`(@NO\_OF\_VISITS);



6. Find the name of the doctors with their contact numbers whom has been visited by only

student.

DELIMITER $$

CREATE PROCEDURE `ONE\_STUDENT\_DOC`(IN NO\_OF\_VISIT int)

BEGIN

select d.doc\_name, count(v.doctor\_id)

from doctor d, visit v

where d.doctor\_id = v.doctor\_id

group by doc\_name,doc\_contact\_no

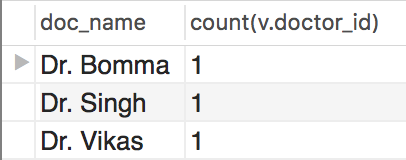
having count(v.doctor\_id)=NO\_OF\_VISIT;

END$$

DELIMITER ;

SET @NO\_OF\_VISIT=1;

CALL `student\_clinic\_db\_final`.`ONE\_STUDENT\_DOC`(@NO\_OF\_VISIT);



7. Find names, id’s, record id and vaccine category for students who have been given

vaccination of FLU and order by student name

DELIMITER $$

CREATE PROCEDURE `STU\_WITH\_VACC`(IN VACC\_NAME varchar(255))

BEGIN

select student.student\_name, student.student\_id, i.record\_id ,i.vaccine\_category

from student

join immunization\_record i using(student\_id)

where i.vaccine\_category = VACC\_NAME

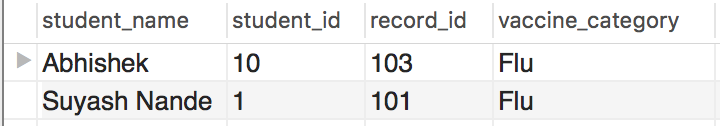
order by student\_name;

END$$

DELIMITER ;

SET @VACC\_NAME='FLU';

CALL `student\_clinic\_db\_final`.`STU\_WITH\_VACC`(@VACC\_NAME);



8. Find number of visits for any student

DELIMITER $$

CREATE PROCEDURE `VISITS\_OF\_ANY\_STUDENT`(IN STU\_NAME

varchar(255))

BEGIN

select student.student\_name, COUNT(visit\_history.visit\_id) AS NumofVisits

from student

join visit\_history using(student\_id)

where student\_name = STU\_NAME;

END$$

DELIMITER ;

SET @STU\_NAME='Suyash Nande';

CALL `student\_clinic\_db\_final`.`VISITS\_OF\_ANY\_STUDENT`(@STU\_NAME);



9. Find the number of Visits on a particular day

DELIMITER $$

CREATE PROCEDURE `No\_Of\_Visit\_On\_Any\_Date`(IN VISIT\_DATE date, out

NUM\_OF\_VISITS int)

BEGIN

select COUNT(visit\_id) AS NumofVisits into NUM\_OF\_VISITS

from visit where visit\_date = VISIT\_DATE;

END$$

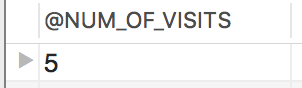
DELIMITER ;

SET @VISIT\_DATE = '2017-03-15';

CALL `student\_clinic\_db\_final`.`No\_Of\_Visit\_On\_Any\_Date`(@VISIT\_DATE,

@NUM\_OF\_VISITS);

SELECT @NUM\_OF\_VISITS;



10. Find the student name and payment status whose payment status is pending. Order by

name.

DELIMITER $$

CREATE PROCEDURE `Payment\_Status\_Of\_Student`(IN STATUS varchar(255))

BEGIN

select student.student\_name , student.student\_id, payment.pay\_status

from student

join student\_bill using (student\_id)

join bill\_payment using (bill\_no)

join payment using ( pay\_trans\_num)

where pay\_status = STATUS

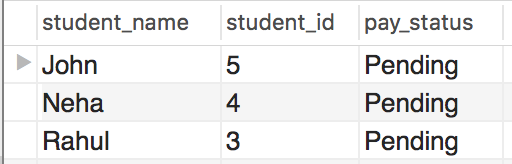
order by student.student\_name;

END$$

DELIMITER ;

SET @STATUS='Pending';

CALL `student\_clinic\_db\_final`.`Payment\_Status\_Of\_Student`(@STATUS);



11. Find out the department name from any complaint from the student

DROP PROCEDURE IF EXISTS dptname\_forcomplaint;

DELIMITER $$

CREATE PROCEDURE `dptname\_forcomplaint`(IN complaint varchar(55))

begin

select d.doc\_name,d.doctor\_id, a.dpt\_name,a.dpt\_id

from doctor d, affiliated\_with a,visit v

where d.doctor\_id = a.doctor\_id

and v.doctor\_id = a.doctor\_id

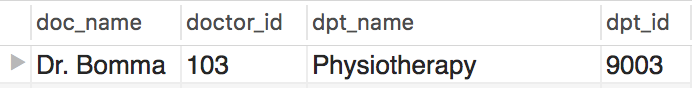
and v.complaint= complaint;

END$$

DELIMITER ;

SET @Complaint='Hypertension';

CALL `student\_clinic\_db\_final\_final`.`dptname\_forcomplaint`(@Complaint);



12. Find out details of the doctor for any department

Drop procedure if exists `docname\_fordept`;

Delimiter $$

CREATE procedure `docname\_fordept`(IN department\_name varchar(55))

begin

select d.doc\_name, d.doctor\_id, a.dpt\_name

from doctor d, affiliated\_with a

where d.doctor\_id = a.doctor\_id

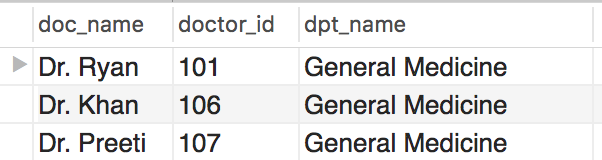
and a.dpt\_name like department\_name;

END; $$

DELIMITER ;

set @department\_name = 'General Medicine';

CALL `student\_clinic\_db\_final\_final`.`docname\_fordept`(@department\_name);



**Add tables and other features to enable audit trail so that every query or change of every record in the database is monitored and the entire history of the data in the database is captured. Basically, every time a record is accessed (queried, inserted, or changed), the user and time of access is recorded. Every time any field of a record is updated or deleted, the previous value of the record is saved.**

**Ans:** We have created audit trial for following tables:

* visit table - Update, Delete and Insert operation.
* payment table - Update, Delete and Insert operation.

**Audit Trial for Visit table**

* **Trigger for before update operation**

DROP TRIGGER IF EXISTS student\_clinic\_db\_final.audit\_visit\_bu;

DELIMITER $$

create trigger audit\_visit\_bu BEFORE UPDATE on Visit

for each row

begin

insert into audit\_trial\_visit SELECT 'BEFORE UPDATE', NULL, NOW(),

current\_user(), V.\* from visit as V

where V.visit\_id = old.visit\_id;

end;$$

DELIMITER ;

* **Trigger for after update operation**

DROP TRIGGER IF EXISTS student\_clinic\_db\_final.audit\_visit\_au;

DELIMITER $$

create trigger audit\_visit\_au AFTER UPDATE on Visit

for each row

begin

insert into audit\_trial\_visit SELECT 'AFTER UPDATE', NULL, NOW(), current\_user(),

V.\* from visit as V

where V.visit\_id = new.visit\_id;

end;$$

DELIMITER ;

* **Trigger for insert operation**

DROP TRIGGER IF EXISTS student\_clinic\_db\_final.visit\_after\_insert

DELIMITER $$

create trigger visit\_after\_insert AFTER INSERT on Visit

for each row

Begin

insert into audit\_trial\_visit SELECT 'INSERT', NULL, NOW(), current\_user(), V.\* from

visit as V

where V.visit\_id = new.visit\_id;

End; $$

DELIMITER ;

* **Trigger for delete operation**

DROP TRIGGER IF EXISTS student\_clinic\_db\_final.visit\_before\_delete

DELIMITER $$

create trigger visit\_before\_delete BEFORE DELETE on Visit

for each row

Begin

insert into audit\_trial\_visit SELECT 'DELETE', NULL, NOW(), current\_user(), V.\* from

visit as V

where V.visit\_id = old.visit\_id;

End; $$

DELIMITER ;



**Audit Trial for Payment table**

* **Trigger for before update operation**

DROP TRIGGER IF EXISTS student\_clinic\_db\_final.audit\_payment\_bu;

DELIMITER $$

create trigger audit\_payment\_bu BEFORE UPDATE on payment

for each row

begin

insert into audit\_trial\_payment SELECT 'BEFORE UPDATE', NULL, NOW(), current\_user(), P.\* from payment as P

where P.pay\_trans\_num = old.pay\_trans\_num;

end;$$

DELIMITER ;

* **Trigger for after update operation**

DROP TRIGGER IF EXISTS student\_clinic\_db\_final.audit\_payment\_au;

DELIMITER $$

create trigger audit\_payment\_au AFTER UPDATE on payment

for each row

begin

insert into audit\_trial\_payment SELECT 'AFTER UPDATE', NULL, NOW(), current\_user(), P.\* from payment as P

where P.pay\_trans\_num = new.pay\_trans\_num;

end;$$

DELIMITER ;

* **Trigger for insert operation**

DROP TRIGGER IF EXISTS student\_clinic\_db\_final.audit\_payment\_after\_insert;

DELIMITER $$

create trigger audit\_payment\_after\_insert AFTER INSERT on payment

for each row

Begin

insert into audit\_trial\_payment SELECT 'INSERT', NULL, NOW(), current\_user(), P.\* from payment as P

where P.pay\_trans\_num = new.pay\_trans\_num;

End; $$

DELIMITER ;

* **Trigger for delete operation**

DROP TRIGGER IF EXISTS student\_clinic\_db\_final.audit\_payment\_after\_delete;

DELIMITER $$

create trigger audit\_payment\_after\_delete BEFORE DELETE on payment

for each row

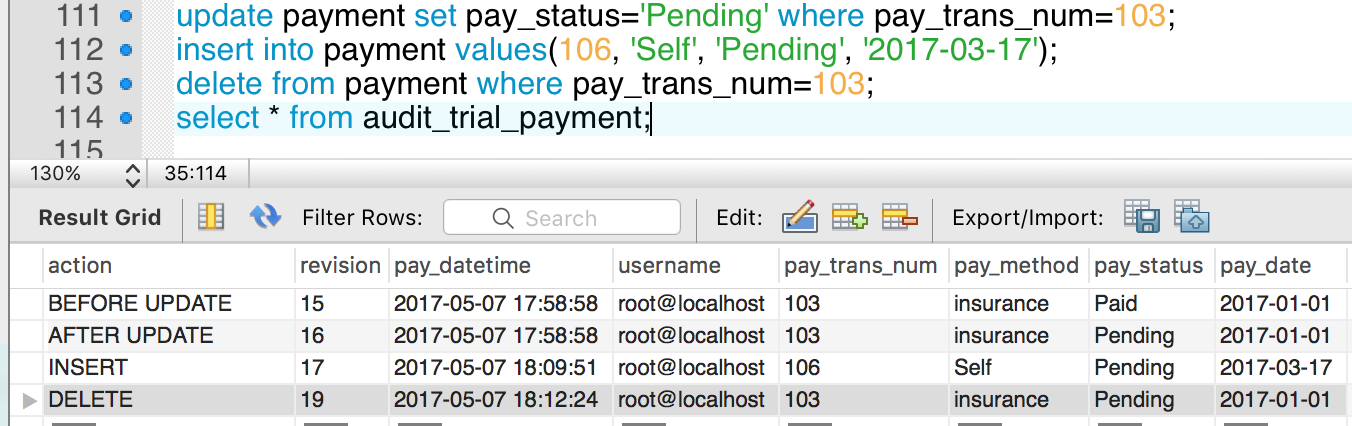
Begin

insert into audit\_trial\_payment SELECT 'DELETE', NULL, NOW(), current\_user(), P.\* from payment as P

where P.pay\_trans\_num = old.pay\_trans\_num;

End; $$

DELIMITER ;



**Add user authentication so that only authorized users can access the database.**

**Ans:** The primary aim of the access control mechanism in MySQL is to authenticate a user based on his username, host and password and to associate the user with privileges on different databases and tables such as Select, Update, Insert, etc.

Following are the salient features:

* Users are assigned to certain roles which are based on their job assignment and access privileges.
* These relationships are many-to-many i.e. the user may have several roles and each role can be

assigned to several users.

* Similarly, an action may be assigned to several roles while a role may have several actions.
* It provides administrator with the capability to place constraints on role authorization, role activation and operation execution.

**user\_role table**

CREATE TABLE `user\_role` (

`user` varchar(45) NOT NULL,

`password` varchar(45) NOT NULL,

`role` varchar(45) NOT NULL,

`host` varchar(45) NOT NULL,

PRIMARY KEY (`user`,`role`,`host`)

) ;

**Procedure for User Authentication**

DROP PROCEDURE IF EXISTS user\_auth;

DELIMITER |

CREATE PROCEDURE user\_auth

(IN name CHAR(32), IN pass CHAR(64), OUT role CHAR(36))

BEGIN

SELECT user\_role.user, user\_role.password, user\_role.role INTO @name, @pass, @role FROM user\_role WHERE user\_role.user = name;

IF (SELECT COUNT(user\_role.user) FROM user\_role WHERE user\_role.user = name AND user\_role.password = pass)!=1 THEN

SET @message\_text = CONCAT ('Login Incorrect for user \'',@name, '\'');

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

ELSE

SIGNAL SQLSTATE '01000' SET MESSAGE\_TEXT = 'successfully authenticated';

SELECT @role as role INTO role;

END IF;

END;|

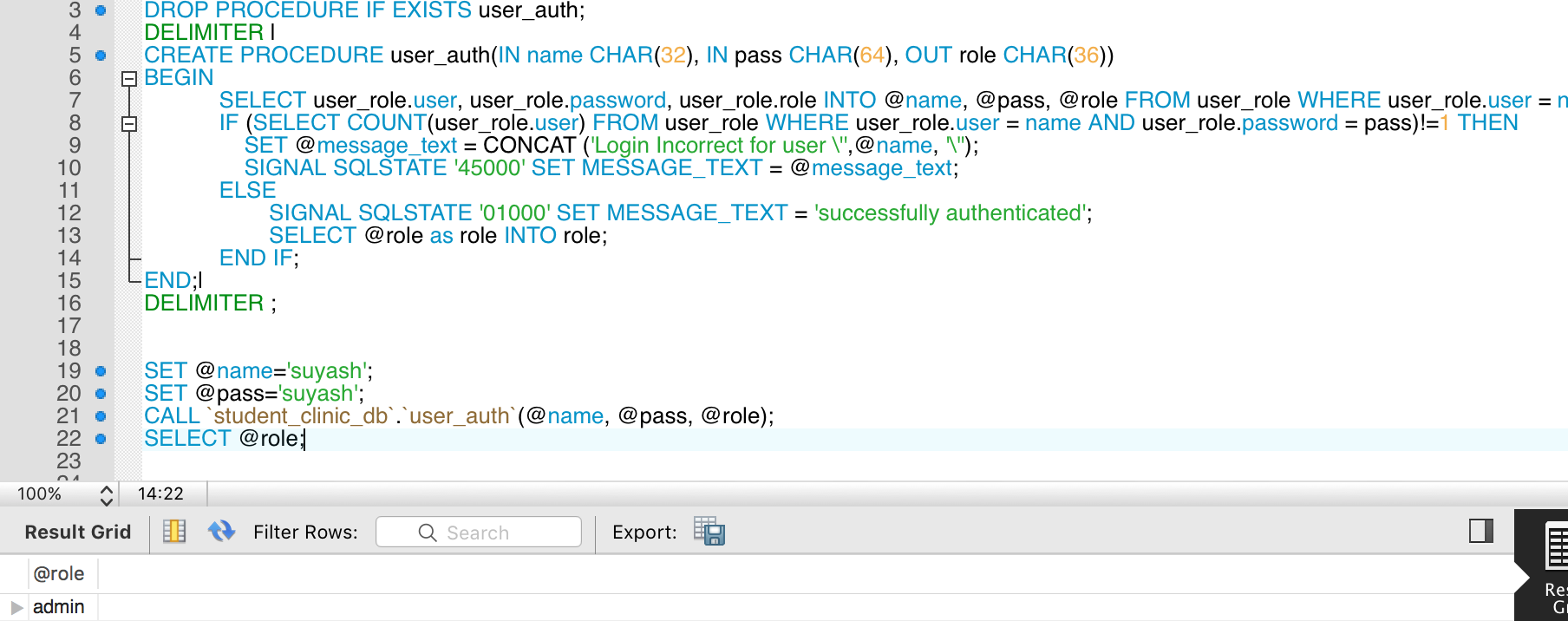
DELIMITER ;

SET @name='suyash';

SET @pass='suyash';

CALL `student\_clinic\_db\_final`.`user\_auth`(@name, @pass, @role);

SELECT @role;



**User Authentication Implementation**

For every procedure call we ensured only authorized user can execute the procedure. Henceforth, we have created role based privilege table.

**privilege table – all the privileges are stored with privilege\_id and role as as Primary Key**

CREATE TABLE `privilege` (

`previlege\_id` int(11) NOT NULL,

`Role` varchar(45) NOT NULL,

`Select\_priv` enum('N','Y') NOT NULL,

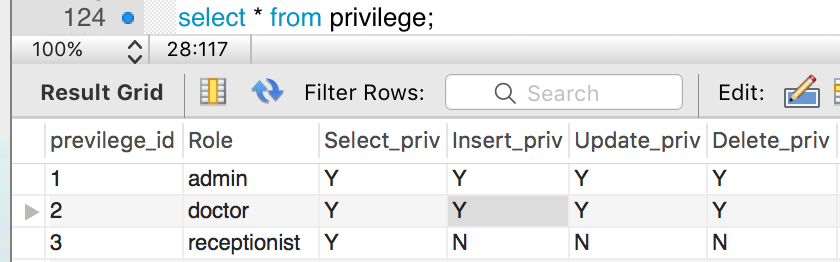
`Insert\_priv` enum('N','Y') NOT NULL,

`Update\_priv` enum('N','Y') NOT NULL,

`Delete\_priv` enum('N','Y') NOT NULL,

PRIMARY KEY (`previlege\_id`,`Role`)

) ;



**Role based Scenario**

When a student consults doctor, then doctor can create, delete and update his prescription.

Here only doctor should have privilege to create, delete and update the prescription of the student. Other role such as receptionist can only view the prescription.

**Prescription created by the doctor**

DROP PROCEDURE IF EXISTS create\_prescription;

DELIMITER $$

CREATE PROCEDURE `create\_prescription`

(IN username varchar(255), IN pass varchar(255),IN presc\_id int, IN med\_name varchar(255), IN med\_quantity int)

BEGIN

CALL user\_auth(username, pass, @role);

IF(@role like 'doctor')

THEN

IF ((select Insert\_priv from privilege where Role like 'doctor') like 'Y')

THEN

insert into prescription values(presc\_id, med\_name, med\_quantity);

ELSE

SET @message\_text = CONCAT(username, ' donot have insert privilege');

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

END IF;

ELSE

SET @message\_text = CONCAT(username, ' with role ', @role, ' is not authorized to create the prescription');

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

END IF;

END$$

DELIMITER ;

SET @username = 'abhishek';

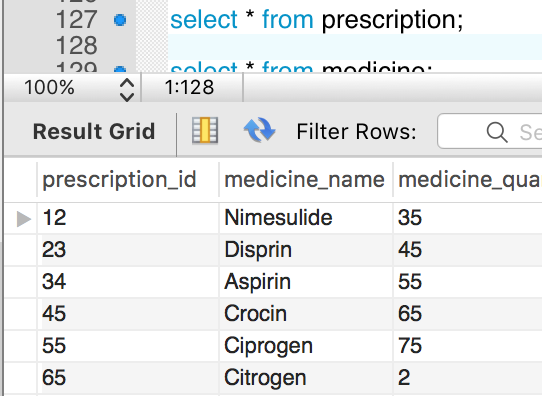
SET @pass = 'abhishek';

SET @presc\_id = 65;

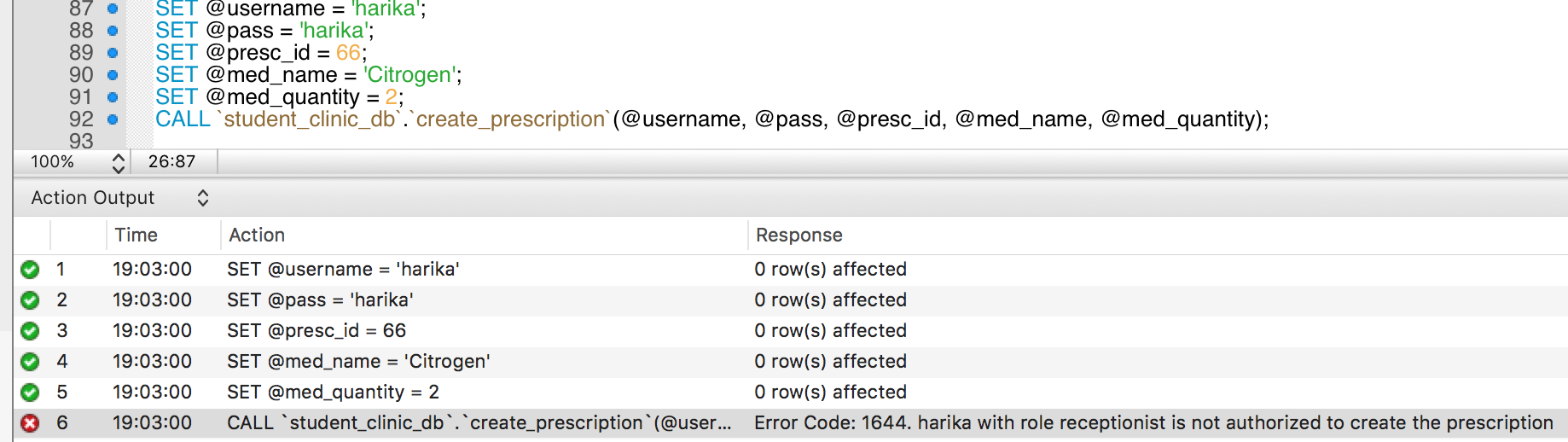
SET @med\_name = 'Citrogen';

SET @med\_quantity = 2;

CALL `student\_clinic\_db\_final`.`create\_prescription`(@username, @pass, @presc\_id, @med\_name, @med\_quantity);



**Now receptionist cannot create a prescription**



**Prescription deleted by the doctor**

DROP PROCEDURE IF EXISTS delete\_prescription;

DELIMITER $$

CREATE PROCEDURE `delete\_prescription`

(IN username varchar(255), IN pass varchar(255),IN presc\_id int)

BEGIN

CALL user\_auth(username, pass, @role);

IF(@role like 'doctor')

THEN

IF ((select Delete\_priv from privilege where Role like 'doctor') like 'Y')

THEN

delete from prescription where prescription\_id=presc\_id;

ELSE

SET @message\_text = CONCAT(username, ' donot have delete privilege');

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

END IF;

ELSE

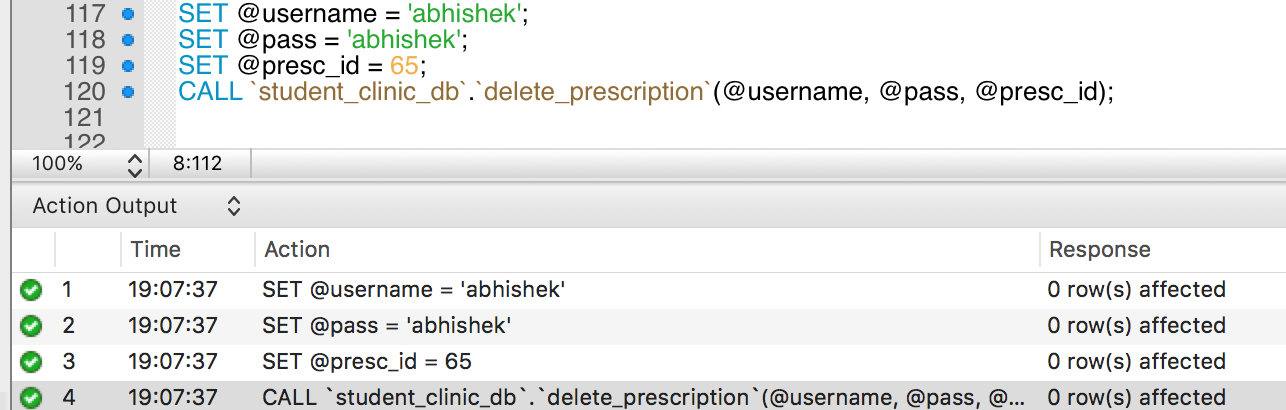
SET @message\_text = CONCAT(username, ' with role ', @role, ' is not authorized to delete the prescription');

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

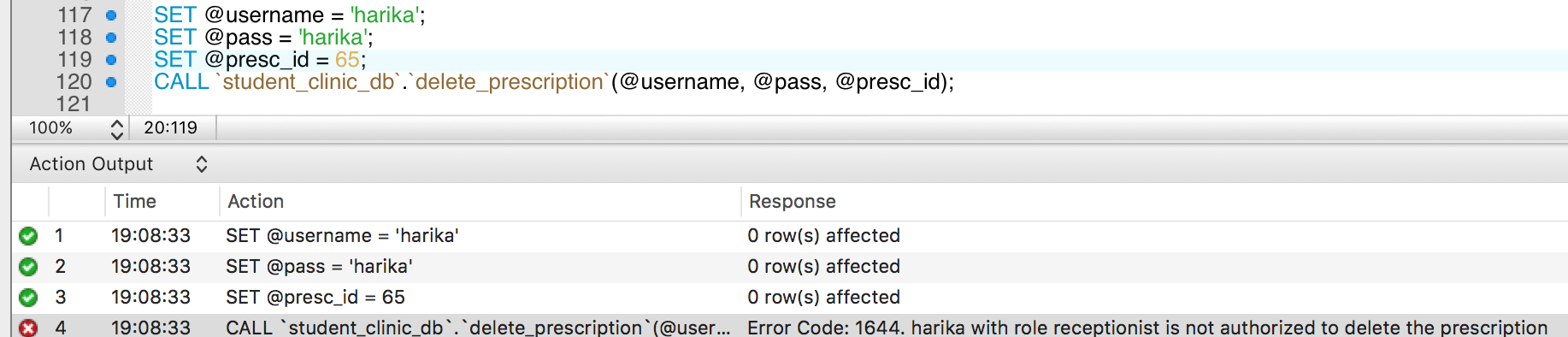
END IF;

END$$

DELIMITER ;



**Now receptionist cannot delete a prescription**



* **Add indexes and views so that frequently used queries and changes to the database are most efficient.**

**Ans: Views** are stored queries that when invoked produce a result set. A **view** acts as a virtual table.

We have created views for following scenarios.

***Scenario I***

When we need to find out the students with pending bills.

***Solution:*** To find out the students who didn’t paid the full amount i.e. whose bills are pending; we need to join **five tables** Student, Bill, Payment, bill\_payment and student\_bill. Henceforth we created a view wherein we joined all these five tables.

create or replace view bill\_pending\_view as

select student.student\_id, student.student\_name, student\_bill.bill\_no, bill.amount, bill.amount\_paid, payment.pay\_trans\_num, bill.due\_date

from bill

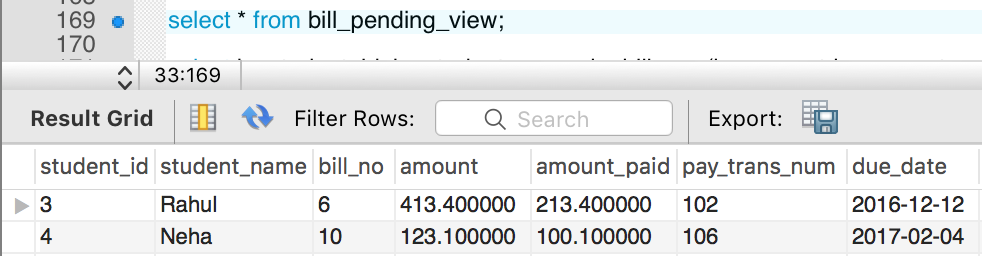
join bill\_payment using (bill\_no)

join payment using (pay\_trans\_num)

join student\_bill using (bill\_no)

join student using (student\_id)

where (bill.amount\_paid<bill.amount);



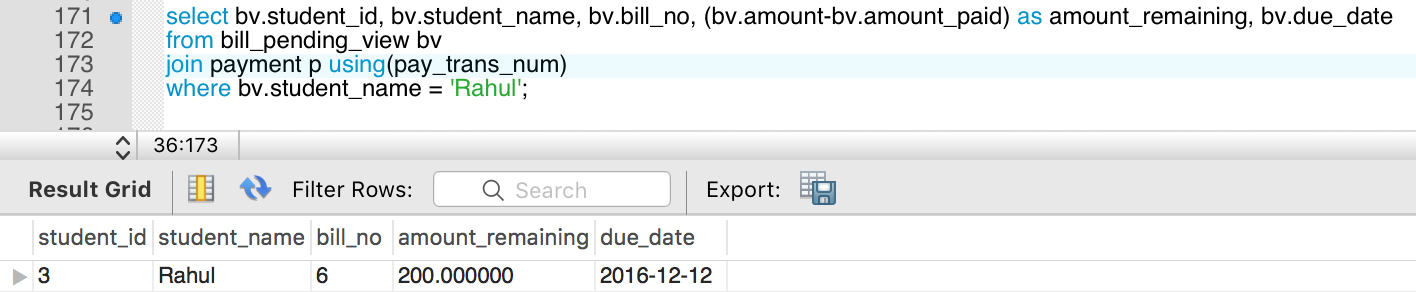
Now we’ll use the above created view to calculate the remaining amount to be paid by the student with his due date.

select bv.student\_id, bv.student\_name, bv.bill\_no, (bv.amount-bv.amount\_paid) as amount\_remaining, bv.due\_date

from bill\_pending\_view bv

join payment p using(pay\_trans\_num)

where bv.student\_name = 'Rahul';



***Scenario II***

When we need to find out the students with paid bills.

***Solution:*** To find out the students who already paid the full amount i.e. whose bills are paid; we need to join **five tables** Student, Bill, Payment, bill\_payment and student\_bill. Henceforth we created a view wherein we joined all these five tables.

create or replace view **bill\_paid\_view** as

select student.student\_id, student.student\_name, student\_bill.bill\_no, bill.amount, bill.amount\_paid, payment.pay\_trans\_num

from bill

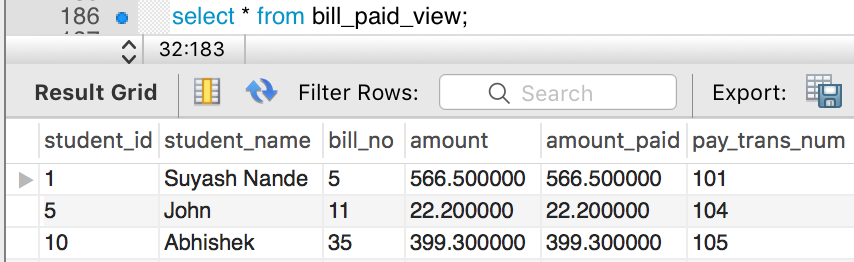
join bill\_payment using (bill\_no)

join payment using (pay\_trans\_num)

join student\_bill using (bill\_no)

join student using (student\_id)

where (bill.amount\_paid = bill.amount);



***Scenario III***

When we need to find out total number of medicines prescribed to a student ‘Suyash Nande’

***Solution:*** To find total number of medicines prescribed to a student we need to join **five tables** Student, Visit, visit\_history, Doctor and Prescription. Henceforth we created a view which consists of results by joining all these tables for any student.

create or replace view Medicine\_For\_Student as

select s.student\_id,s.student\_name,v.visit\_id,d.doc\_name, p.medicine\_name AS Prescribed\_Medicine

from student s

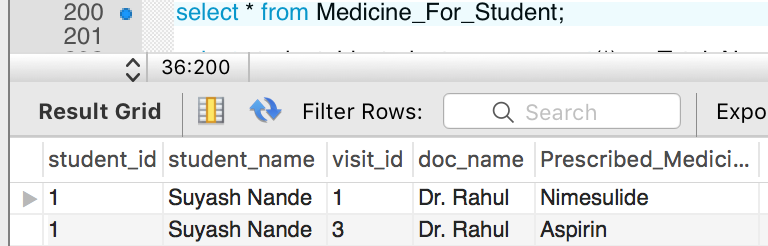
join visit\_history using(student\_id)

join visit v using (visit\_id)

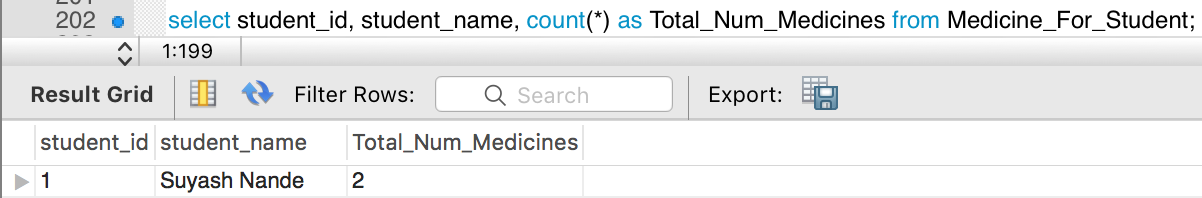
join doctor d using(doctor\_id)

join prescription p using(prescription\_id)

where s.student\_name = 'Suyash Nande';



Now we’ll use above created view to calculate the total number of medicines prescribed for that student.



***Scenario IV***

When we need to find out maximum visit for any department.

***Solution:*** To find maximum number of visit for any department we need to join two tables affiliated\_with and visit and the result set should contain information about all the department with their visit count.

create or replace view dept\_with\_max\_visit as

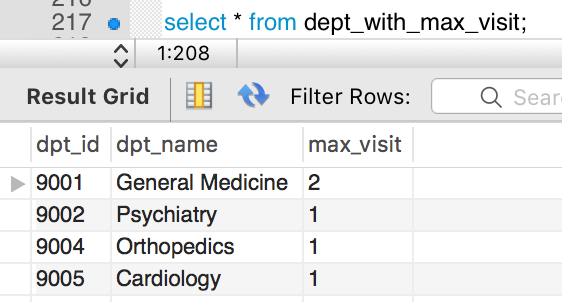
select a.dpt\_id, a.dpt\_name, count(a.dpt\_name) as max\_visit

from affiliated\_with a

join visit v using(doctor\_id)

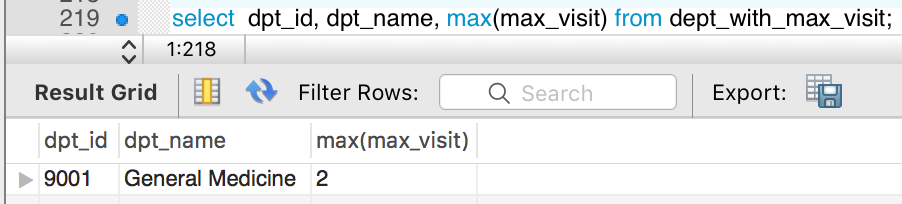
group by a.dpt\_id, a.dpt\_name

HAVING max(v.doctor\_id);



Now among these we need to select the department with the maximum visit count.

select dpt\_id, dpt\_name, max(max\_visit) from dept\_with\_max\_visit;



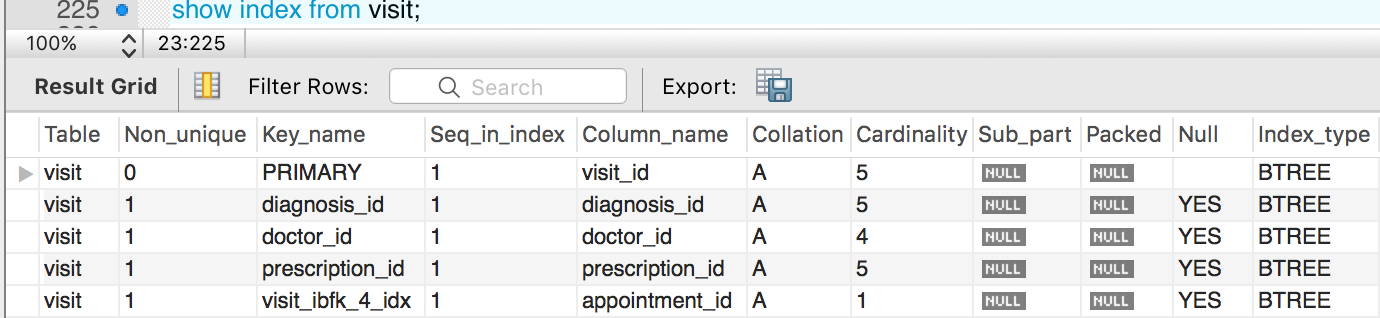
**Indexes**

Index is a data structure that improves the speed of search operation in a table. It can be created using one or more columns. Indexes are also type of tables, which keep primary key or index field and a pointer to each record into the actual table.

Users cannot see the indexes, they are just to speed up queries and will be used by database search engine to locate records very fast.

We have created indexes for every tables based on primary key and foreign key.

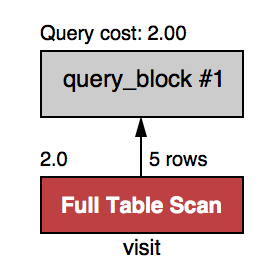
***Scenario I: Index for visit table***

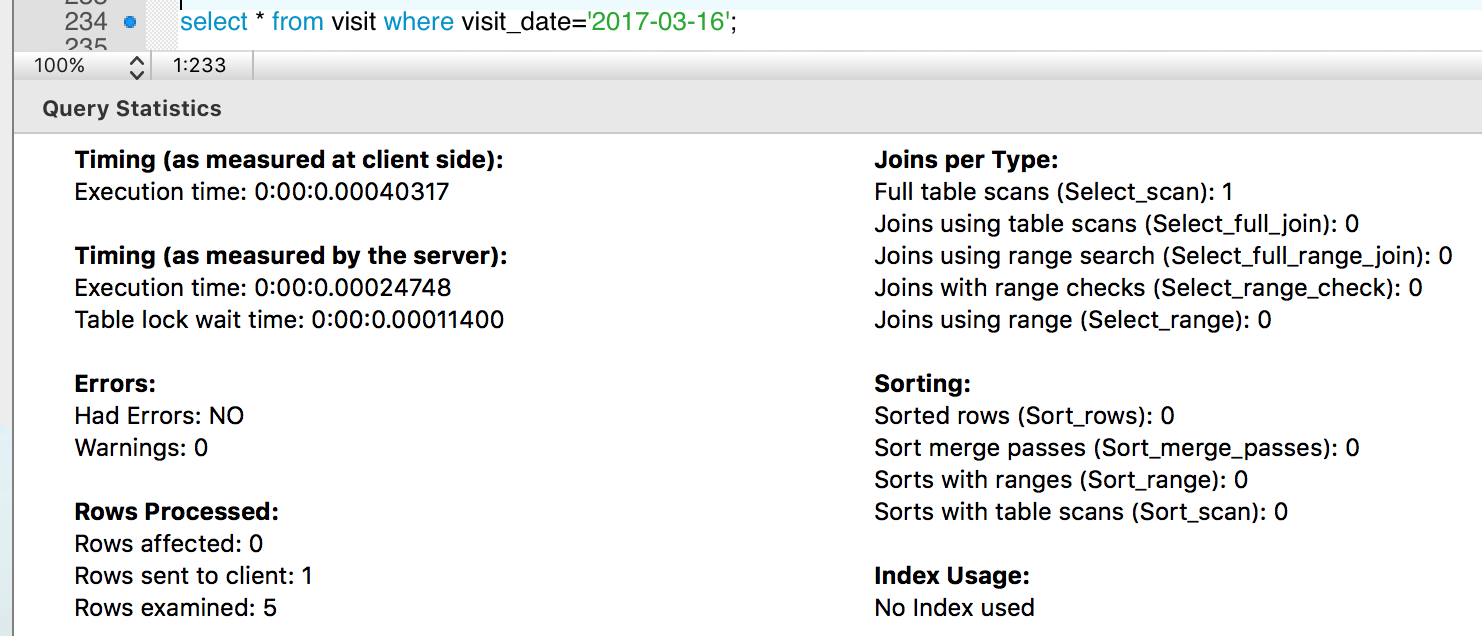


In the above visit table, **visit\_id is the primary key** and **diagnosis\_id, doctor\_id, prescription\_id and appointment\_id are the foreign keys.** Hence we have created indexes for all the keys.

**Find the visit details of a student based on non-index field visit\_date.**

**Ans:** select \* from visit where visit\_date='2017-03-16';

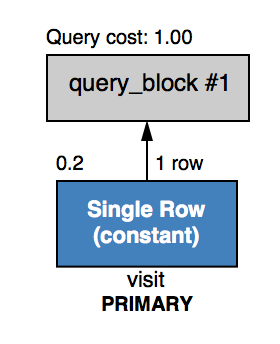




In the above query, full table is scanned as no index were used. Therefore, the **query cost is 2.0**. And hence execution time is **0.00040317 seconds**.

**Now on using the index field *visit\_id* we will perform the same operation.**

**Ans:** select \* from visit where visit\_id=5;





In the above query index was used, **query cost is 1.0** and time taken to execute the query was **0.00031805 seconds.**

Hence, by using the index the same result set takes **0.00008512 seconds** less time to execute so it is faster to search via an index field.

**Experience**

* Creating audit trial was not an easy task, as it involves triggers that run before and after each query execution for that table, so we need to carefully examine the tuple data before inserting and updating it.
* Implementing user authentication and role based access was a challenging task. We have implemented the same into the stored procedures so that every time we call these procedures it will first check the authorization of the user and if the user is authorized then it will again check for that user privilege to execute the SQL statements written in the stored procedures.
* Merging the relations was made easy since foreign keys existed that referenced the various tables that were merged.
* The SQL statement became complex and more error-prone as the number of relations to be merged increased.

**Individual Contribution**

|  |  |
| --- | --- |
| ***Harika Katragadda*** | * Insert API * Delete API * Update API * Indexes * ER Diagram |
| ***Abhishek Deshpande*** | * Views * UML Diagram * Audit Trial * Triggers |
| ***Suyash Nande*** | * APIs for scenario’s * Triggers * User Authentication * Role Based Access Control |