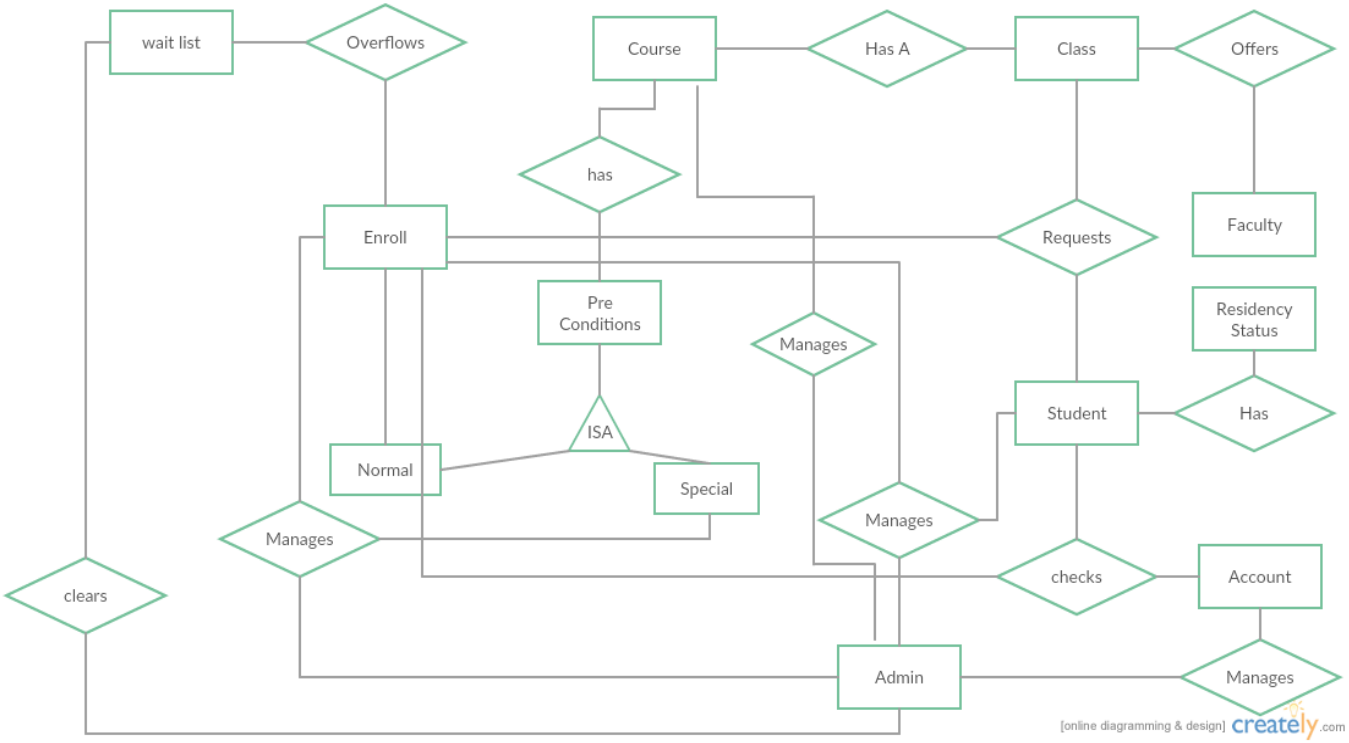
ER MODEL

The ER diagram shows the relationship between the different entity sets. This document describes the different relationships used, participation constraints, ISA relationships used. A description of all the tables and the reason for their use is mentioned in the relational model.



KEY CONSTRAINTS:

Waitlist:
primary key – drop_id, waitlist no
foreign key – sid, class_id

Course:
Primary key - course_id

Class:
Primary key – class_id

Enroll:
Primary key – course_id, sid
Foreign key - course_id, sid

Admin:
Primary key – aid

Account:
Primary key – sid
Foreign key – sid

Preconditions:
Primary key – course_id, prerequisite course_id
Foreign key – course_id

Faculty:
Primary Key – fid

Rstatus:
Primary key – level,residency

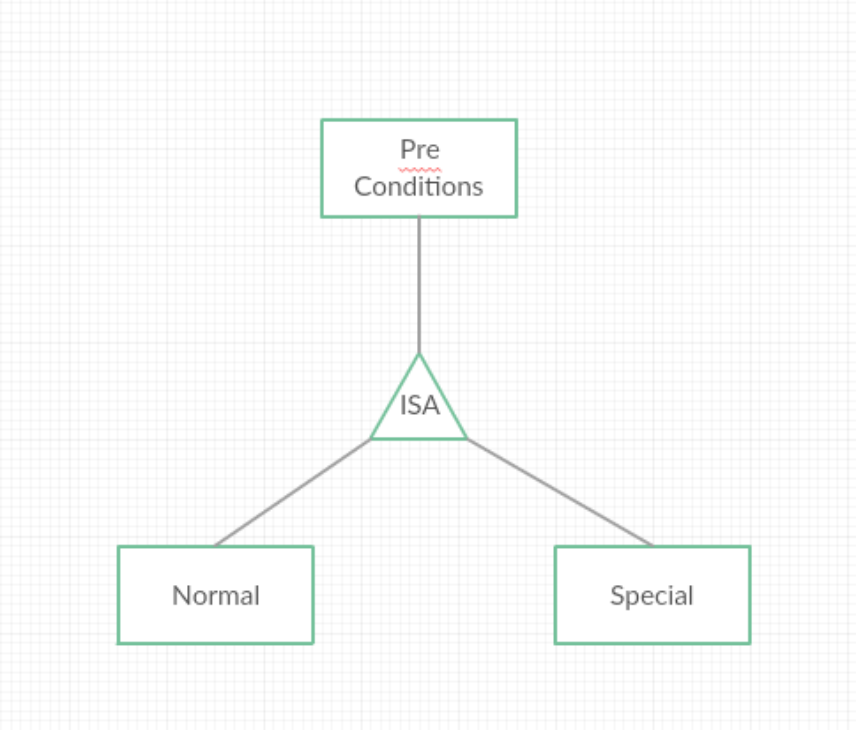
PARTICIPATION CONSTRAINTS:

Atleast one:
Class (entity) to faculty (entity)
Each class is offered by atleast one faculty

Exactly one:
Student (entity) to residency status (entity)
Each Student exactly has one residency status

Class (entity) to course (entity)
Each class has exactly one course.

ISA Relationships:
We used one ISA relationship
Pre conditions – Normal and Special

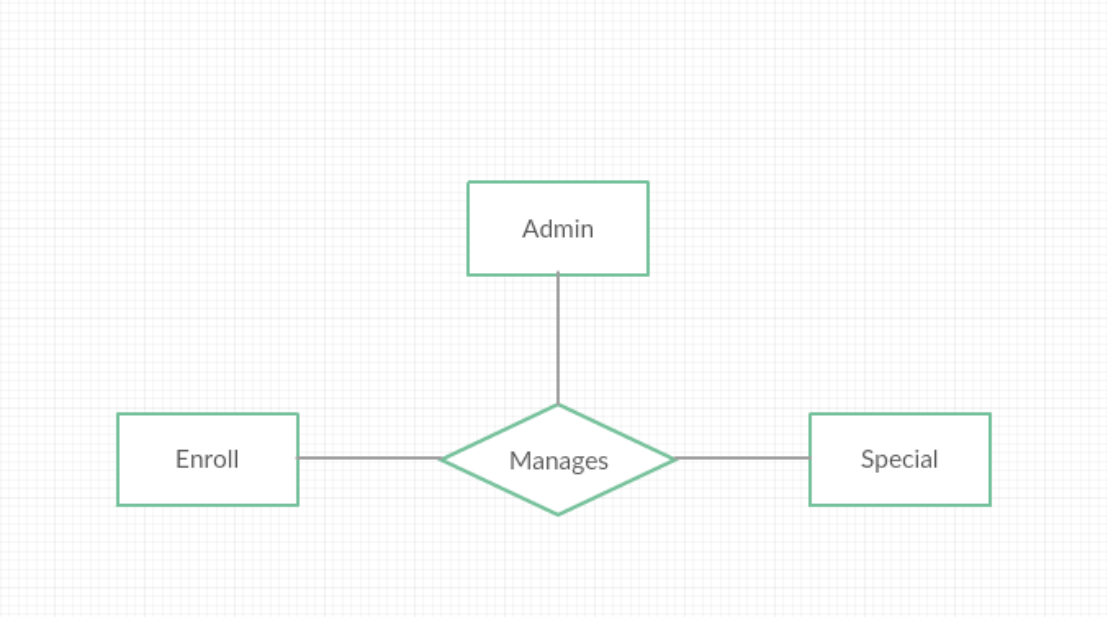


We used ER approach to translate above relationship into table. The pre-conditions table consists common information and normal, special table consists of specific information related to them. This ISA relationship is disjoint and Complete.

TERNARY RELATIONSHIP:

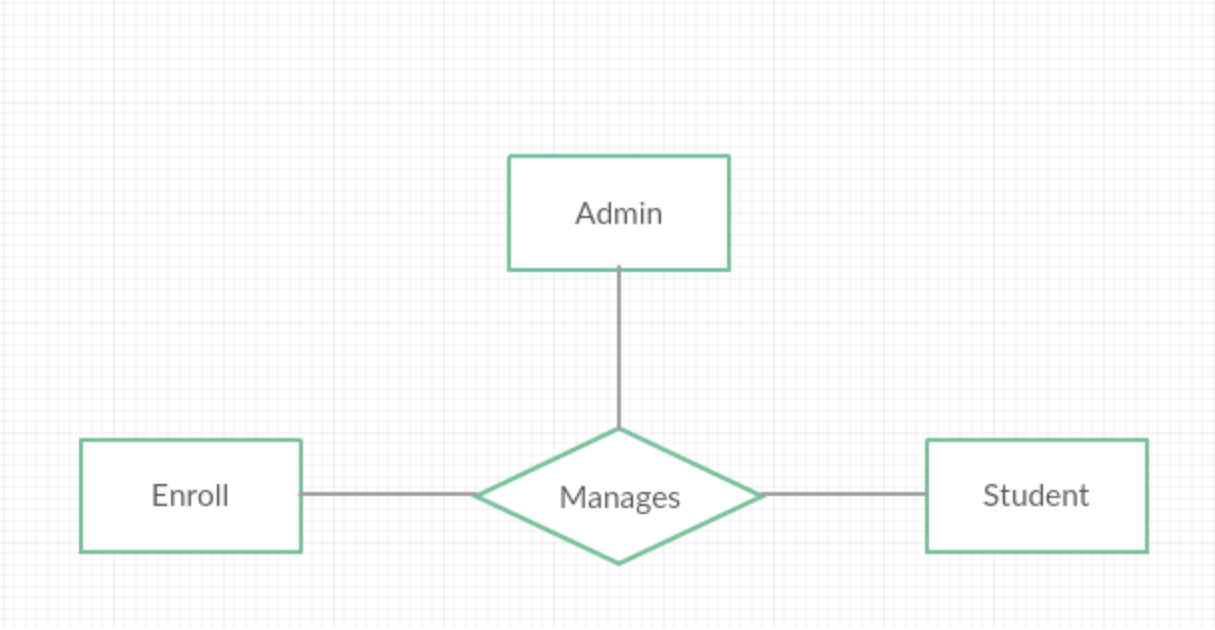
We have the following ternary relationships in our ER diagram.

1.



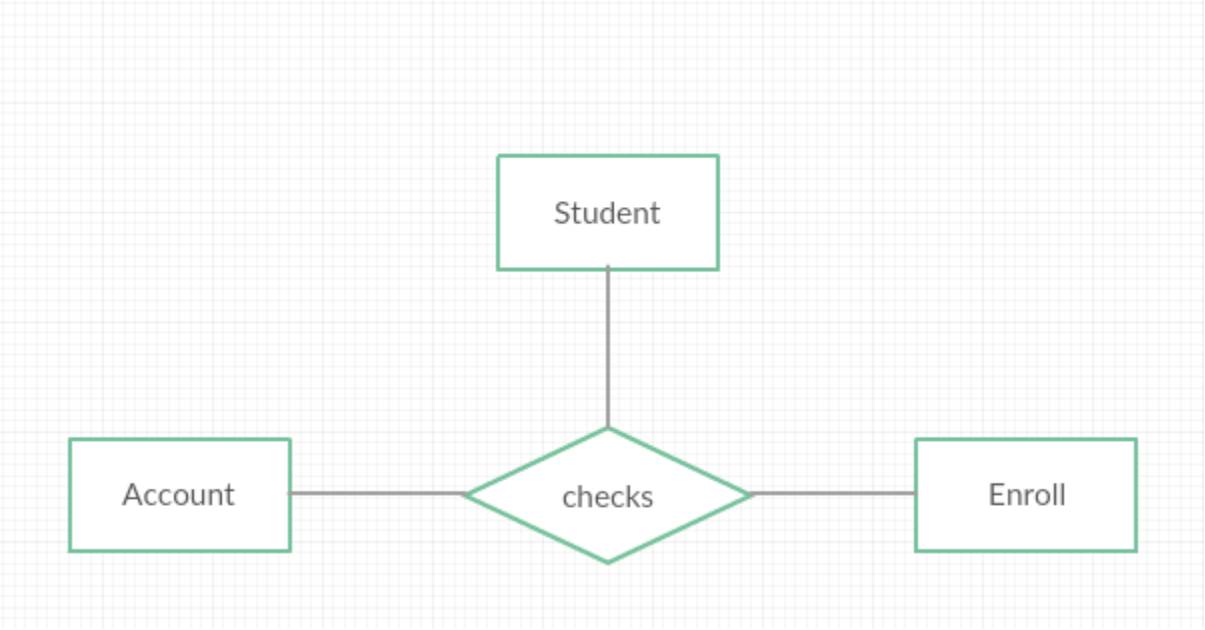
When a student request to enroll into a class with special permissions, the admin has to approve it. Hence the ternary relationship.

2.



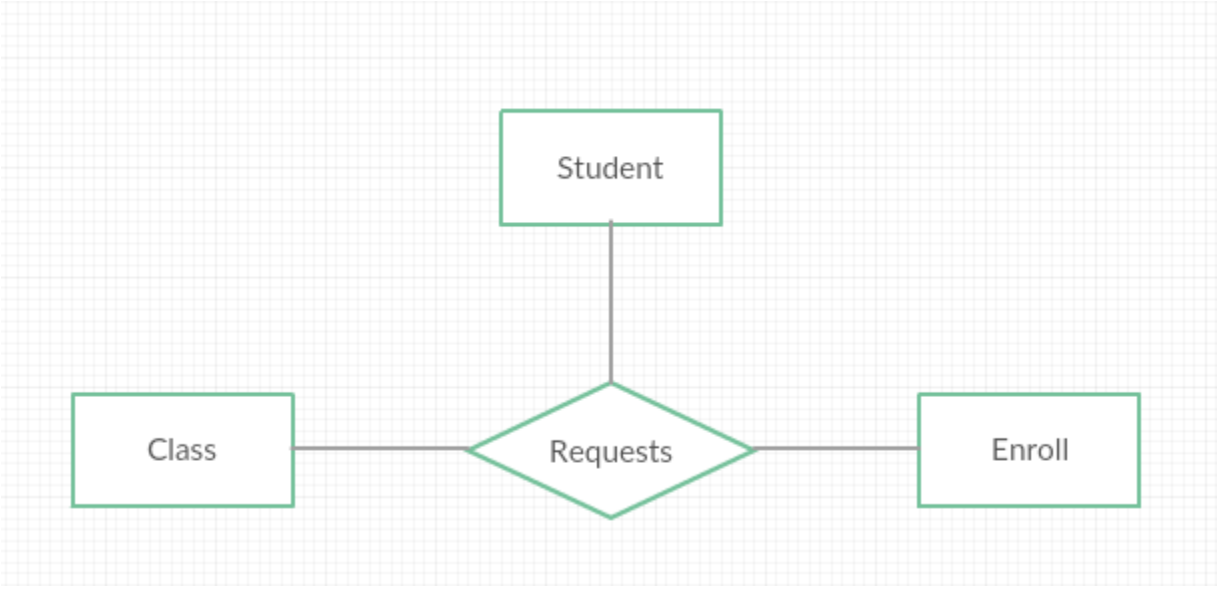
When a student enrolls into a classes, admin has to modify the grades for those classes in enroll table for each student. Also admin can modify student details. Hence the ternary relationship.

3.



When a student enrolls into a course, his account has to be updated with number of credit hours and respective billing has to be calculated. Hence the ternary relationship.

4.



Each time a student wants to enroll, he has to has corresponding course to enroll into. Hence the ternary relationship.

CONSTRAINTS:

1. Checking for admin and student credentials while they login is done on application level. We couldn't achieve it on database level because fetching the database of credentials and validating it against the given credentials was not possible.
2. After enrollment deadline is enforced by administrator, all the pending, waiting enrollment request have to be rejected and the student wont be able to enroll anymore. This was implemented on application level as it was not possible to implement to stop restrict enrollment for this semester and reopen it for next semester.

RELATIONAL MODEL

We tried to maintain 3NF but in some cases 3NF is not satisfied as we would require lots of joins and decided against implementing 3NF.

1. CLASSES: It holds information about the offerings of classes in each semester including class id, maximum number that can be in waitlist (waitlist_max), total students enrolled (total_count), semester, current year, maximum students that can enroll (max_count).

Constraints:
PRIMARY KEY - class_id
waitlist_max<=max_count
max_count>=total_count

SQL >
```
CREATE table classes(
class_id varchar2(20) primary key,
waitlist_max numeric NOT NULL,
total_count numeric default 0,
semester varchar2(20) NOT NULL,
cur_year numeric NOT NULL,
max_count numeric NOT NULL,
constraint wl check(waitlist_max<=max_count),
constraint maxc check(max_count>=total_count));
```

2. COURSES: It holds information about different courses that are offered in the university including course id, min_num_of_credits, max_num_of_credits, course_level, dept, title.

Constraints:
PRIMARY KEY - course_id
course_level is either 1 or 2
min_num_of_credits>0
min_num_of_credits<=max_num_of_credits)

SQL>
```
create table course(
course_id varchar2(20) primary key,
min_num_of_credits numeric,
max_num_of_credits numeric,
course_level numeric NOT NULL,
dept varchar2(20) NOT NULL,
title varchar2(20) NOT NULL,
constraint c_level check (course_level in (1,2)),
constraint credits check(min_num_of_credits>0),
constraint min_max check(min_num_of_credits<=max_num_of_credits));
```

3.SCHEDULE: It holds the information about classes in each semester like class_id , days it is offered (days), from_time, to_time, location.

Constraints:
PRIMARY KEY: class_id
FOREIGN KEY : class_id which references to class_id in classes table
from_time<to_time

SQL>
CREATE table schedule(
class_id varchar2(20),
days varchar2(20) NOT NULL,
from_time TIMESTAMP NOT NULL,
to_time TIMESTAMP NOT NULL,
location varchar2(20) NOT NULL,
FOREIGN KEY(class_id) references classes(class_id) on delete cascade,
constraint timee check(from_time<to_time));

4. FACULTY: This table holds information about all the faculty present in the university including faculty id (f_id), first name (f_name), last name (l_name)

CONSTRAINTS:
PRIMARY KEY : f_id
NOT NULL - f_name, l_name

SQL>
create table faculty(
f_id varchar2(20) primary key,
f_name varchar2(20) NOT NULL,
l_name varchar2(20) NOT NULL);

5. CLASS FACULTY: This table holds information about all the faculty that are offering a class in each semester including class_id, faculty id (f_id)

CONSTRAINTS:
NOT NULL - class_id , f_id
FOREIGN KEY – class_id which references to class_id in classes table , f_id which references to f_id in faculty table

SQL>
create table class_faculty(
class_id varchar2(20) NOT NULL,
f_id varchar2(20) NOT NULL,
FOREIGN KEY(class_id) references classes(class_id) on delete cascade,
FOREIGN KEY(f_id) references faculty(f_id));

6. RSTATUS: This table holds information about various restrictions and billing that student incurs based on their residency status including student_level, residency, dollar per credit hour (dollar_hour), minimum credits to be taken (min_credit), maximum credits to be taken (max_credits)

CONSTRAINTS:
PRIMARY KEY – (student_level,residency)

student_level should be either 1 or 2,
residency should be either 1,2 or 3

SQL>
create table rstatus(
student_level numeric NOT NULL,
residency numeric NOT NULL,
dollar_hour numeric NOT NULL,
min_credit numeric NOT NULL,
max_credit numeric NOT NULL,
constraint pk primary key(student_level,residency),
constraint st_level1 check (student_level in (1,2)),
constraint res1 check(residency in(1,2,3)));

7.STUDENT: This table holds information about each student in the university including student id (sid), first name (fname), last name (lname), department (did), password (pwd), email , student_level, residency, gpa

CONSTRAINTS:
student_level be either 1 or 2,
residency be either 1,2 or 3
FOREIGN KEY - (student_level, residency) references (student_level, residency) in rstatus table

SQL>
create table student (
sid varchar2(20) primary key,
fname varchar2(20) NOT NULL,
lname varchar2(20) NOT NULL,
did varchar2(20),
pwd varchar2(20) NOT NULL,
email varchar2(20),
student_level numeric NOT NULL,
residency numeric NOT NULL,
gpa numeric default 0,
constraint st_level check (student_level in (1,2)),
constraint res check(residency in(1,2,3)),
FOREIGN KEY(student_level,residency) references rstatus(student_level,residency) on delete cascade);

8. WAITLIST: This table holds information about student put in waitlist for enrolling into each course including waitlist_no, drop_id , class_id , sid

CONSTRAINTS:
PRIMARY KEY – (class_id,sid)
FOREIGN KEY – class_id references class_id in classes table
FOREIGN_KEY – drop_id references class_id in classes table
FOREIGN_KEY – sid references sid in student table

SQL>
create table waitlist(
waitlist_no varchar2(20) NOT NULL,
drop_id varchar2(20) NOT NULL,
class_id varchar2(20) NOT NULL,
sid varchar2(20) NOT NULL,
PRIMARY KEY(class_id,sid),
FOREIGN KEY(class_id) references classes(class_id) on delete cascade,
FOREIGN KEY(drop_id) references classes(class_id),
FOREIGN KEY(sid) REFERENCES student(sid) on delete cascade);

9. ENROLL: This table holds information related to courses enrolled by each student in each semester including course_id, class_id, sid, num_of_credits, grade.

CONSTRAINTS:
PRIMARY_KEY – (course_id, sid)
FOREIGN_KEY – course_id references course_id in course table
FOREIGN_KEY – class_id references class_id in classes table
FOREIGN_KEY – sid references sid in student table

SQL>
create table enroll(
course_id varchar2(20),
class_id varchar2(20),
sid varchar2(20),
num_of_credits numeric,
grade varchar2(20),
constraint allk PRIMARY KEY(course_id,class_id,sid),
FOREIGN KEY(course_id) references course(course_id) on delete cascade,
FOREIGN KEY(class_id) references classes(class_id) on delete cascade,
FOREIGN KEY(sid) references student(sid) on delete cascade);

10. ADMINISTRATOR: This table holds information about administrators present in the university including administrator id (aid), admin_fname, admin_lname, admin_pwd and admin SSN

CONSTRAINTS:
PRIMARY KEY – aid
NOT NULL – admin_fname, admin_lname, admin_pwd, SSN

SQL>
create table administrator(
aid varchar2(20) primary key,
admin_fname varchar2(20) NOT NULL,
admin_lname varchar2(20) NOT NULL,
admin_pwd varchar2(20) NOT NULL,
SSN varchar2(20) NOT NULL);

11. PRE_CONDITION: This table holds information about pre conditions required to enroll into each course including course_id , prereq_course_id, min_gpa, special permissions (perm)

CONSTRAINTS:
PRIMARY KEY – (course_id, prereq_courseid)
FOREIGN KEY – course_id references course_id in course table
FOREIGN KEY – prereq_courseid references course_id in course table
PERM be present in either PREREQ or SPPERM

```
SQL>
create table pre_condition(
course_id varchar2(20),
prereq_courseid varchar2(20),
min_gpa numeric default 0 ,
perm varchar2(20),
constraint pk1 PRIMARY KEY(course_id, prereq_courseid),
FOREIGN KEY(course_id) references course(course_id) on delete cascade,
FOREIGN KEY(prereq_courseid) references course(course_id) on delete set NULL,
constraint p check (perm in ('PREREQ','SPPERM')));
```

12. ACCOUNT: This table holds information about accounts of each student in university including total amount, student id (sid), balance

CONSTRAINTS:
PRIMARY KEY- sid,
FOREIGN KEY- sid references sid in student table

```
SQL>
create table account(
total_amount float,
sid varchar2(20) primary key,
balance float NOT NULL,
FOREIGN KEY(sid) references student(sid) on delete cascade);
```

13. GRADING: This table holds information about mapping of letter grade to grade points including grade and its relevant grade point.

CONSTRAINTS:
PRIMARY KEY : grade
NOT NULL – gp

```
SQL>
create table grading(
grade varchar2(20) primary key,
gp float NOT NULL);
```

14. PENDING: This table holds information about the pending enrollment approvals that administrator has to be approve including class_id, course_id, student id, number of credits

PRIMARY KEY : course_id,class_id,sid
FOREIGN KEY : course_id references course_id from course table
FOREIGN KEY : class_id references class_id from classes table
FOREIGN KEY: sid references sid from students table

SQL>
```
create table pending(
class_id varchar2(20),
course_id varchar2(20),
sid varchar2(20),
num_of_credits numeric,
constraint p primary key(course_id,class_id,sid),
FOREIGN KEY(course_id) references course(course_id),
FOREIGN KEY(class_id) references classes(class_id),
FOREIGN KEY(sid) references student(sid));
```

15. REJECT: This table holds information about the enrollment requests that were rejected either by the administrator or due to other constraints including class_id, course_id, student id, number of credits
PRIMARY KEY : course_id,class_id,sid
FOREIGN KEY : course_id references course_id from course table
FOREIGN KEY : class_id references class_id from classes table
FOREIGN KEY: sid references sid from students table

SQL>
```
create table reject1 (
class_id varchar2(20),
course_id varchar2(20),
sid varchar2(20),
constraint p primary key(course_id,class_id,sid),
FOREIGN KEY(course_id) references course(course_id),
FOREIGN KEY(class_id) references classes(class_id),
FOREIGN KEY(sid) references student(sid));
```

PROCEDURES AND TRIGGERS:

TRIGGER: admindrop

This trigger takes cares of deleting the entries from pending, waitlist, reject tables and updating student account when administrator enforces the enrollment deadline.

```
SQL>
DELIMITER @@
CREATE OR REPLACE trigger NTHANIK.admindrop
after update on administrator
for each row
declare
mincre numeric;
res numeric;
count1 numeric;
lev numeric;
begin
if :new.enforced=1 then
count1:=0;
delete from pending;
delete from reject1;
delete from waitlist;
for s1 in (select sid,sum(num_of_credits) as no_cre from enroll group by sid)
loop
select residency,student_level into res,lev from student where sid=s1.sid;
select min_credit into mincre from rstatus where residency=res and student_level=lev;
if s1.no_cre<mincre then
count1:=count1+1;
end if;
end loop;
for s in (select * from account where balance<>0)
loop
delete from enroll where sid=s.sid;
end loop;
update account set total_amount=0.0;
update account set balance=0.0;
update classes set total_count=0;
for s2 in (select * from student)
loop
insert into comparison values(s2.sid,s2.gpa);
end loop;
if count1<>0 then
RAISE_APPLICATION_ERROR(-20111,'admin drop');
end if;
end if;
end; @@
DELIMITER ;
```

TRIGGER: trig_decrementcount:
This trigger handles the case when a student drops a course, it updates the class count and updates his account with appropriate billing and puts the next person from waitlist into the class in case if there is someone in waitlist.

```
SQL>
DELIMITER @@
CREATE OR REPLACE trigger NTHANIK.trig_decrementcount
before delete on enroll
for each row
declare
c_id varchar2(20);
tot numeric;
max1 numeric;
diff numeric;
s_id varchar2(20);
amount float;
res numeric;
lev numeric;
dollar float;
no_cre numeric;
begin
c_id:=:old.class_id;
s_id:=:old.sid;
no_cre:=:old.num_of_credits;
select residency,student_level into res,lev from student where sid=s_id;
select dollar_hour into dollar from rstatus where residency=res and student_level=lev;
select total_amount into amount from account where sid=s_id;
amount:= amount-no_cre*dollar;
update account set total_amount=amount where sid=s_id;
select total_count into tot from classes where class_id=c_id;
tot:=tot-1;
update classes set total_count= tot where class_id=c_id;
select max_count into max1 from classes where class_id=c_id;
if tot<max1 then
update waitlist set waitlist_no=waitlist_no-1 where class_id=c_id;
end if;
end; @@
DELIMITER ;
```

TRIGGER: trig_gpacalc:
This trigger handles the case when a grade for a course in enrollment is updated, it calculates the new gpa for student and updates his student table.

```
SQL>
DELIMITER @@
CREATE OR REPLACE trigger NTHANIK.trig_gpacalc
before update on enroll
for each row
declare
c_id varchar2(20);
co_id varchar2(20);
g_pa float;
g_rade varchar2(20);
cre1 float;
s_id varchar2(20);
begin
s_id:= :new.sid;
g_rade:= :new.grade;
cre1:= :new.num_of_credits;
co_id:= :new.course_id;
gpafunc(s_id,g_rade,cre1,co_id,c_id,g_pa);
update student set gpa=g_pa where sid=s_id;
end; @@
DELIMITER ;
```

TRIGGER: trig_incrementcount:

This trigger handles the case when a student tries to enroll into a class, it checks if student needs special permission, if the student has completed required prerequisite courses to enroll this particular course, if the class count is less than the maximum count possible, if the student has minimum gpa to enroll into this course, if the student exceeds the maximum allowable credits after he registers this course and then enrolls the student into this class by modifying enroll table and also updating student's account table.

```
SQL>
DELIMITER @@
CREATE OR REPLACE trigger NTHANIK.trig_incrementcount
   before insert on enroll
   for each row
   declare
   c_id varchar2(20);
   tot numeric;
   count2 numeric;
   wait numeric;
   max1 numeric;
   s_id varchar2(20);
   s_p numeric;
   no_cre numeric;
   res numeric;
   lev numeric;
   dollar float;
   amount float;
   maxcre numeric;
```

```
    co_id varchar2(20);
    cre1 numeric;
    cur_gpa float;
    mingpa float;
    pre1 varchar2(20);
    count1 numeric;
    co_level numeric;
    spl varchar2(20);
    begin
    s_p:=800;
    c_id:= :new.class_id;
    s_id:= :new.sid;
    co_id:= :new.course_id;
    no_cre:= :new.num_of_credits;
    cre1:=0;
    select total_count into tot from classes where class_id=c_id;
    tot:= tot+1;
    select max_count into max1 from classes where class_id=c_id;
select perm into spl from pre_condition where course_id=co_id;
select course_level into co_level from course where course_id=co_id;
select residency,student_level into res,lev from student where sid=s_id;
select prereq_courseid into pre1 from pre_condition where course_id=co_id;
select dollar_hour,max_credit into dollar,maxcre from rstatus where residency=res and
student_level=lev;
creditcount(s_id,cre1);
proc22(s_id,pre1,count1);
select gpa into cur_gpa from student where sid=s_id;
proctemp(co_id,mingpa);
tempproc(s_id,c_id,co_id,count2);
if ((co_level<>lev or spl='SPPERM') and count2=0)  then
pendingproc(s_id,c_id,co_id,no_cre);
RAISE_APPLICATION_ERROR(-20111,'waiting for special permission');
elsif tot>max1 then
log_error_p(c_id,s_id,co_id);
RAISE_APPLICATION_ERROR(-20111,'total count greater than max count');
elsif count1=0 AND pre1<> 'NONE' then
rejectproc(s_id,c_id,co_id);
RAISE_APPLICATION_ERROR(-20001,'prerequisite not satisfied');
elsif cur_gpa<mingpa then
rejectproc(s_id,c_id,co_id);
RAISE_APPLICATION_ERROR(-20111,'minimum gpa not satisfied');
elsif cre1+no_cre >maxcre then
rejectproc(s_id,c_id,co_id);
RAISE_APPLICATION_ERROR(-20111,'credits greater than max credits');
else
update classes set total_count= tot where class_id=c_id;
select total_amount into amount from account where sid=s_id;
amount:= amount+no_cre*dollar;
update account set total_amount=amount where sid=s_id;
```

```
END IF;
END; @@
DELIMITER ;
```

PROCEDURE: Creditcount
This procedure counts the number of credits enrolled by each student

```
SQL>
DELIMITER @@
CREATE OR REPLACE PROCEDURE NTHANIK.creditcount(s_id IN varchar2,cre1 OUT numeric)
is
sum1 numeric;
PRAGMA AUTONOMOUS_TRANSACTION;
 BEGIN
commit;
sum1:=0;
for s1 in (select * from enroll where sid=s_id)
loop
if s1.grade is NULL then
sum1:=sum1+s1.num_of_credits;
end if;
end loop;
cre1:=sum1;
commit;
END; @@
DELIMITER ;
```

PROCEDURE: gpafunc
This procedure is used for calculating gpa of a student from his grades.

```
SQL>
DELIMITER @@
CREATE OR REPLACE PROCEDURE NTHANIK.gpafunc(s_id IN varchar2,g_rade IN
varchar2,cre1 IN float,co_id IN varchar2,c_id IN varchar2,g_pa OUT float)
is
val float;
i number:=0;
add float;
f float;
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
g_pa:=0.0;
add:= 0.0;
for s in (select * from enroll where sid=s_id)
loop
if s.grade is NOT NULL then
select gp into val from grading where grade=s.grade;
f:=s.num_of_credits*1.0;
```

```
g_pa:=g_pa+val*f;
add:= add+ f;
end if;
end loop;
f:=cre1*1.0;
select gp into val from grading where grade=g_rade;
g_pa:=g_pa+val*f;
add:=add+f;
g_pa:=g_pa/add;
END; @@
DELIMITER ;
```

PROCEDURE: log_error_p

This procedure is when a student registers into a class waitlist and the waitlist is at maximum count, the student enrollment request has to be added to reject table.

```
SQL>
DELIMITER @@
CREATE OR REPLACE PROCEDURE NTHANIK.log_error_p(c_id IN varchar2,s_id IN
varchar2,co_id IN varchar2)
is wait1 numeric;
waitmax numeric;
PRAGMA AUTONOMOUS_TRANSACTION;
 BEGIN
commit;
wait1:=0;
select waitlist_max into waitmax from classes where class_id=c_id;
select count(*) into wait1 from waitlist where class_id=c_id;
wait1:=wait1+1;
if wait1<=waitmax then
insert into waitlist(waitlist_no,class_id,sid) values(wait1,c_id,s_id);
else
insert into reject1 values(c_id,co_id,s_id);
end if;
commit;
END; @@
DELIMITER ;
```

PROCEDURE: pendingproc

This procedure is used when a student's enrollment has to be approved by administrator and it has be added to pending table in order for the administrator to look it up.

```
SQL>
DELIMITER @@
CREATE OR REPLACE PROCEDURE NTHANIK.pendingproc(s_id IN varchar2,c_id IN
varchar2,co_id IN varchar2,no_cre IN numeric)
is
cre1 numeric;
PRAGMA AUTONOMOUS_TRANSACTION;
```

```
 BEGIN
commit;
cre1:=no_cre;
insert into pending values(c_id,co_id,s_id,cre1);
commit;
END;
@@DELIMITER ;
```

PROCEDURE: proc22

This procedure counts the number of prerequisite courses that student has satisfied for registering into a course.

```
SQL>
DELIMITER @@
CREATE OR REPLACE PROCEDURE NTHANIK.proc22(s_id IN varchar2,pre1 IN varchar2,count1
OUT numeric)
is
PRAGMA AUTONOMOUS_TRANSACTION;
 BEGIN
commit;
count1:=0;
for s in (select * from enroll where sid=s_id)
loop
IF(S.COURSE_ID = PRE1) THEN
count1:=count1+1;
END IF;
end loop;
commit;
END; @@
DELIMITER ;
```

PROCEDURE: proctemp

This procedure is used to acquire the minimum gpa required to enroll into a course which is mentioned in the preconditions.

```
SQL>
DELIMITER @@
CREATE OR REPLACE PROCEDURE NTHANIK.proctemp(co_id IN varchar2,mingpa OUT float)
is
PRAGMA AUTONOMOUS_TRANSACTION;
 BEGIN
commit;
mingpa:=0.0;
for s in (select * from pre_condition)
loop
if(s.course_id=co_id) then
mingpa:=s.min_gpa;
end if;
end loop;
```

```
mingpa:=mingpa*1.0;
commit;
END; @@
DELIMITER ;
```

PROCEDURE: rejectproc
This procedure is used to insert rejected enrollment requests into reject table.

```
SQL>
DELIMITER @@
CREATE OR REPLACE PROCEDURE NTHANIK.rejectproc(s_id IN varchar2,c_id IN
varchar2,co_id IN varchar2)
is
PRAGMA AUTONOMOUS_TRANSACTION;
 BEGIN
commit;
insert into reject1 values(c_id,co_id,s_id);
commit;
END; @@
DELIMITER ;
```

PROCEDURE: tempproc
This procedure is used to count the number of pending course.

```
SQL>
DELIMITER @@
CREATE OR REPLACE PROCEDURE NTHANIK.tempproc(s_id IN varchar2,c_id IN
varchar2,co_id IN varchar2,count2 OUT numeric)
is
PRAGMA AUTONOMOUS_TRANSACTION;
 BEGIN
commit;
count2:=0;
for s in (select * from pending)
loop
if s.sid=s_id AND s.course_id=co_id and s.class_id= c_id
then
count2:=count2+1;
end if;
end loop;
end; @@
DELIMITER ;
```