Design Document

Class: CS425 - Database Design and Applications

Project: Interactive Students/Faculties Networking

Team member: Danna Liu (A20345502)
Shanshan Jiang (A20354599)
Zhizheng Li (A20351352)

Edited by Danna Liu

Introduction

There are 4 parts in this document.

Part I (a revision of HW2)

- Draw ER diagram according to the description of the class project.
- Deliver SQL schemas with relevant keys and functional dependencies of each schema, then create the tables with the specified attribute names and appropriate constraints such as primary keys, foreign keys, and unique attributes.

Part II (a revision of HW3)

- Stored procedures/functions (use PL/SQL) to answer the questions relating to class project.

Part III

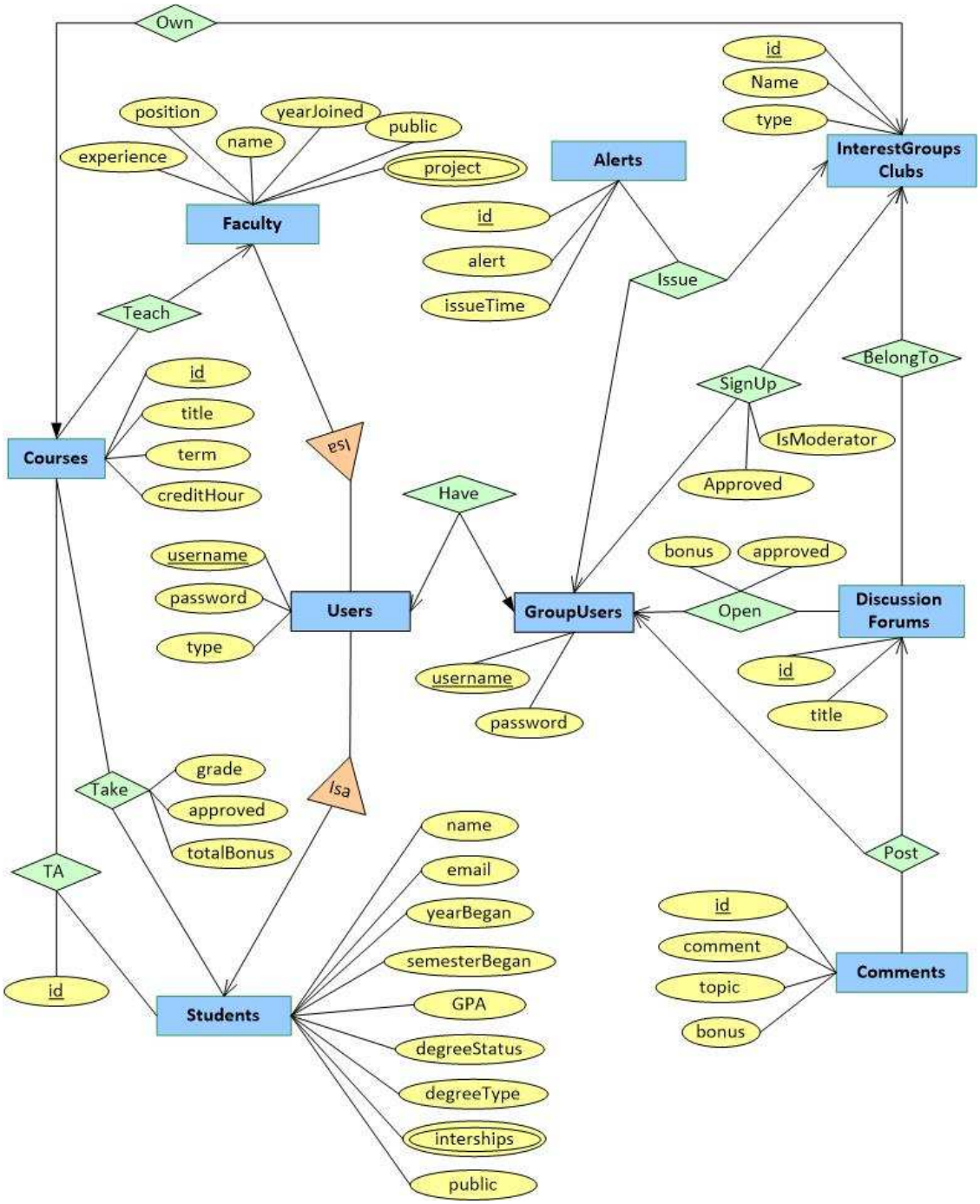
- Specify system requirements and features with user case diagrams and show ownerships among members with a features ownership form.

Part IIII

- The System Architecture for class project.

Part I

1. ER diagram



** GPA will be calculated only when required.*
** The site administrator is one type of users.*

2. SQL schemas

All the primary keys are underlined. Other candidate keys (CK), foreign keys (FK), AND functional dependency (FD) are specified.

Users (username, password, type)
CK: email
FD: username → password name email type

Faculty (username, name, email, position, yearJoined, experience, publicEmail, publicPosition, publicYearJoined, publicExperience)
FK: username references username in Users
FD: username → position yearJoined experience publicEmail publicPosition, publicYearJoined publicExperience

researchProjects (username, projectID, project, publicProject)
FK: username references username in Users
FD: username projectID → project publicProject

Students (username, name, email, yearBegan, SemesterBegan, GPA, degreeStatus, degreeType, publicEmail, publicYearBegan, publicSemesterBegan, publicGPA, publicDegreeStatus, publicDegreeType)
FK: username references username in Users
FD: username → name email yearBegan SemesterBegan GPA degreeStatus degreeType publicEmail publicYearBegan publicSemesterBegan publicGPA publicDegreeStatus publicDegreeType

internships (username, internshipID, internship, publicInternship)
FK: username references username in Users
FD: username internshipID→ internship, publicInternship

InterestGroupsClubs (id, name, type)
FD: id → name type

GroupUsers (groupUser, groupPwd, groupID, username, approved, isModerator)
FK: groupID references id in InterestGroupsClubs
FK: username references username in Users
FD: groupUser → groupPwd, groupID, username, approved, isModerator

Courses (id, title, term, creditHour, instructor, approved, groupID)
FK: instructor references username in Faculty
FK: groupID references id in InterestGroupsClubs
CK: groupID
FD: id → title term creditHour instructor approved groupID
FD: groupID → id title term creditHour instructor approved
Take (student, courseID, grade, approved, totalBonus)

FK: student references username in Students
FK: courseID references id in Courses
FD: student courseID → grade approved totalBonus

TA (id, courseID, courseTA)
FK: courseTA references username in Students
FK: courseID references id in Courses

DisscussionForums (id, title, groupID, creator, bonus, approved)
FK: groupID references id in InterestGroupsClubs
FK: creator references groupUser in GroupUsers
FD: id → title groupID creator bonus approved

Comments (id, topic, message, author, forumID, bouns)
FK: author references groupUser in GroupUsers
FK: forumID references id in DiscussionForums
FD: id → topic message author forumID bouns

Alerts (id, alert, moderator, groupID)
FK: moderator references groupUser in GroupUsers
FK: groupID references id in InterestGroupsClubs
FD: id → alert moderator groupID

Create all the tables with the specified attribute names AND appropriate constraints

```
CREATE TABLE Users(
username VARCHAR(25) NOT NULL,
password VARCHAR(10) NOT NULL ,
type VARCHAR(15) NOT NULL,
PRIMARY KEY (username)
);

CREATE TABLE Faculty(
username VARCHAR(25) NOT NULL,
name VARCHAR(25) NOT NULL,
email VARCHAR(30) NOT NULL,
position VARCHAR(25) NOT NULL ,
yearJoined NUMBER(10) NOT NULL,
experience VARCHAR(255) NOT NULL,
publicEmail VARCHAR(10) NOT NULL,
publicPosition VARCHAR(10) NOT NULL,
publicYearJoined VARCHAR(10) NOT NULL,
publicExperience VARCHAR(10) NOT NULL,
```

```
PRIMARY KEY (username),
FOREIGN KEY (username) REFERENCES Users (username)
);

CREATE TABLE researchProjects(
username VARCHAR(25) NOT NULL,
projectID VARCHAR(100) NOT NULL ,
project VARCHAR(255) NOT NULL,
publicProject VARCHAR(10) NOT NULL,
PRIMARY KEY (username, projectID),
FOREIGN KEY (username) REFERENCES Users (username)
);

CREATE TABLE Students(
username VARCHAR(25) NOT NULL,
name VARCHAR(25) NOT NULL,
email VARCHAR(30) NOT NULL,
yearBegan NUMBER(10) NOT NULL,
semesterBegan VARCHAR(10)NOT NULL,
degreeStatus VARCHAR(25) NOT NULL,
degreeType VARCHAR(25) NOT NULL,
publicEmail VARCHAR(10) NOT NULL,
publicYearBegan VARCHAR(10) NOT NULL,
publicSemesterBegan VARCHAR(10) NOT NULL,
publicGPA VARCHAR(10) NOT NULL,
publicDegreeStatus VARCHAR(10) NOT NULL,
publicDegreeType VARCHAR(10) NOT NULL,
PRIMARY KEY (username),
FOREIGN KEY (username) REFERENCES Users (username)
);

CREATE TABLE internships(
username VARCHAR(25) NOT NULL,
internshipID VARCHAR(100) NOT NULL ,
internship VARCHAR(255) NOT NULL,
publicInternship VARCHAR(10) NOT NULL,
PRIMARY KEY (username, internshipID),
FOREIGN KEY (username) REFERENCES Users (username)
);

-- three type: courseGroup, otherGroup, Club
CREATE TABLE InterestGroupsClubs(
id VARCHAR(50) NOT NULL,
```

```
name VARCHAR(30) NOT NULL,
type VARCHAR(30) NOT NULL,
PRIMARY KEY (id)
);

CREATE TABLE GroupUsers(
groupUser VARCHAR(25) NOT NULL,
groupPwd NUMBER(10) NOT NULL,
groupID VARCHAR(50) NOT NULL,
username VARCHAR(25) NOT NULL,
approved VARCHAR(25) NOT NULL,
isModerator VARCHAR(10) NOT NULL,
PRIMARY KEY (groupUser),
FOREIGN KEY (groupID) REFERENCES InterestGroupsClubs(id),
FOREIGN KEY (username) REFERENCES Users (username)
);

CREATE TABLE Courses(
id VARCHAR(50)NOT NULL,
title VARCHAR(50) NOT NULL,
term VARCHAR(10)NOT NULL,
creditHour NUMBER(2) NOT NULL,
instructor VARCHAR(25) NOT NULL,
groupID VARCHAR(50),
PRIMARY KEY (id),
FOREIGN KEY (instructor) REFERENCES Faculty (username),
FOREIGN KEY (groupID) REFERENCES InterestGroupsClubs (id)
);

CREATE TABLE Take(
student VARCHAR(25) NOT NULL,
courseID VARCHAR(50)NOT NULL,
grade VARCHAR(5) NOT NULL,
approved VARCHAR(5) NOT NULL,
totalBonus NUMBER(10) DEFAULT 0,
PRIMARY KEY (student,courseID),
FOREIGN KEY (student) REFERENCES Students (username),
FOREIGN KEY (courseID) REFERENCES Courses (id)
);

CREATE TABLE TAs(
id VARCHAR(50)NOT NULL,
courseID VARCHAR(10) NOT NULL,
```

```
courseTA VARCHAR(25) NOT NULL,
Primary key (courseID, courseTA),
FOREIGN KEY (courseID) REFERENCES Courses (id),
FOREIGN KEY (courseTA) REFERENCES Students (username)
);

CREATE TABLE DiscussionForums(
id VARCHAR(50) NOT NULL,
title VARCHAR(30) NOT NULL,
groupID VARCHAR(50) NOT NULL,
creator VARCHAR(25) NOT NULL,
approved VARCHAR(25) NOT NULL,
bonus NUMBER(5) DEFAULT 0,
PRIMARY KEY (id),
FOREIGN KEY (groupID) REFERENCES InterestGroupsClubs(id),
FOREIGN KEY (creator) REFERENCES GroupUsers (groupUser)
);

-- points criteria:  courseGroup 5points, otherGroup 4points, Club 3points
CREATE TABLE Comments(
id INT NOT NULL,
topic VARCHAR(100) NOT NULL,
message VARCHAR(255) NOT NULL,
author VARCHAR(25) NOT NULL,
forumID VARCHAR(50) NOT NULL,
bonus NUMBER(5) DEFAULT 0,
PRIMARY KEY (id),
FOREIGN KEY (author) REFERENCES GroupUsers (groupUser),
FOREIGN KEY (forumID) REFERENCES DiscussionForums (id)
);

CREATE TABLE Alerts(
id VARCHAR(50) NOT NULL,
alert VARCHAR(255) NOT NULL,
moderator VARCHAR(25) NOT NULL,
groupID VARCHAR(50) NOT NULL,
PRIMARY KEY (id),
FOREIGN KEY (moderator) REFERENCES GroupUsers (groupUser),
FOREIGN KEY (groupID) REFERENCES InterestGroupsClubs (id)
);
```

Part II

The follwoing are all procedures, functions, and triggers:

Question 1: Write a stored procedure/function that generates a username and password for any user. The username should be a combination of name and the year they joined /began pursuing a degree.

```
CREATE OR REPLACE FUNCTION FacultyUsername(
  name IN Faculty.name%TYPE,
  yearJoined IN Faculty.yearJoined%TYPE
)
RETURN string IS
BEGIN
 RETURN name||TO_CHAR(yearJoined,'FM9999');
END;

CREATE OR REPLACE FUNCTION StudentUsername(
  name IN Students.name%TYPE,
  yearBegan IN Students.yearBegan%TYPE
)
RETURN string IS
BEGIN
 RETURN name||TO_CHAR(yearBegan,'FM9999');
END;

CREATE OR REPLACE FUNCTION CreatePwd
RETURN string IS
BEGIN
 RETURN dbms_random.string('U', 5);
END;
```

Question 2: Write a stored procedure/function that lets a user register to the site, with basic profile information. Use the PSM from question1 to generate a username and a password.

```
create or replace PROCEDURE FacultyReg(
  name IN Faculty.name%TYPE,
  email IN Faculty.email%TYPE,
  position IN Faculty.position%TYPE,
  year IN Faculty.yearJoined%TYPE,
  experience IN Faculty.experience%TYPE,
  publicProfile IN Faculty.publicProfile%TYPE
)AS
 BEGIN
   INSERT INTO Users
```

```
   VALUES(FacultyUsername(name,year),CreatePwd,'Faculty');
   INSERT INTO Faculty
   VALUES(name||TO_CHAR(year,'FM9999'), name, email, position, year, experience,
publicProfile);
 END;

create or replace PROCEDURE StudentReg(
  name IN Students.name%TYPE,
  email IN Students.email%TYPE,
  yearBegan IN Students.yearBegan%TYPE,
  semesterBegan IN Students.semesterBegan%TYPE,
  degreeStatus IN Students.degreeStatus%TYPE,
  degreeType IN Students.degreeType%TYPE,
  publicProfile IN Students.publicProfile%TYPE
)AS
   user Students.username%TYPE;
 BEGIN
   user := StudentUsername(name, yearBegan);
   INSERT INTO Users
   VALUES(user,CreatePwd,'Student');
   INSERT INTO Students
   VALUES(user, name, email, yearBegan, semesterBegan,DEFAULT, DEFAULT, DEFAULT,
   degreeStatus, degreeType, publicProfile);
 END;
```

Question 3: Write a stored procedure/function that creates interest groups for all the courses that don't have one.

```
create or replace PROCEDURE CreateCoursesGroup
AS
  course_id Courses.id%TYPE;
  group_id Courses.groupID%TYPE;
  title Courses.title%TYPE;
  term Courses.term%TYPE;
  CURSOR c IS
    SELECT id, title, term, groupID FROM courses;
BEGIN
  OPEN c;
  LOOP
     FETCH c INTO course_id, title, term, group_id;
     EXIT WHEN c%NOTFOUND;
     IF group_id is null THEN
       group_id := dbms_random.string('U', 3);
       INSERT INTO InterestGroupsClubs
```

```
      VALUES(group_id, title||'_group' , 'courseGroup');

      UPDATE Courses
      SET groupID = group_id
      WHERE id = course_id;

    END IF;
  END LOOP;
  CLOSE c;
END;
```

Question 4: When the user inputs a student's name, he should view all the details that are marked public.

```
create or replace PROCEDURE viewStu(
  input IN Students.name%TYPE
)
AS
  CURSOR c IS
  SELECT * FROM Students
  WHERE name = input;
  some_student c%ROWTYPE;
BEGIN
  OPEN c;
  LOOP
    FETCH c INTO some_student;
    EXIT WHEN c%NOTFOUND;
    IF lower(some_student.publicProfile) = 'yes' THEN
      dbms_output.put_line('Here is the basic information about ' ||some_student.name);
      dbms_output.put_line('username:     ' ||some_student.username);
      dbms_output.put_line('email:        ' ||some_student.email);
      dbms_output.put_line('yearBegan:    ' ||some_student.yearBegan);
      dbms_output.put_line('semesterBegan: ' ||some_student.semesterBegan);
      dbms_output.put_line('GPA:          ' ||some_student.GPA);
      dbms_output.put_line('email:        ' ||some_student.email);
      dbms_output.put_line('degreeStatus:  ' ||some_student.degreeStatus);
      dbms_output.put_line('degreeType:   ' ||some_student.degreeType);
    ELSE
      dbms_output.put_line('Profile hidden.');
    END IF;
  END LOOP;
  CLOSE c;
END;
```

Question 5: When the user inputs a forum name, display the three most popular topics i.e the topics with most comments (in descending order).

```
create or replace PROCEDURE popTopic(
  input IN DiscussionForums.title%TYPE
)
AS
  CURSOR c IS
  SELECT topic FROM Comments
  WHERE forumID = (SELECT id FROM DiscussionForums
              WHERE title = input)
  GROUP BY topic
  ORDER BY count(*) desc;
  pop_topic c%ROWTYPE;

BEGIN
OPEN c;
  FOR i IN 1..3 LOOP
     FETCH c INTO pop_topic;
     EXIT WHEN c%NOTFOUND;
     dbms_output.put_line(pop_topic.topic);
  END LOOP;
CLOSE c;
END;
```

Question 6: Write a trigger to update the GPA of a student when a faculty modifies the grade in a course.

```
create or replace TRIGGER GPAupdate
AFTER INSERT OR UPDATE ON Take
FOR EACH ROW
  DECLARE
     currentGPA Students.GPA%TYPE;
     currentCredit Courses.creditHour%TYPE;
     newGPA students.GPA%TYPE := 0;
     credit_hour Courses.creditHour%TYPE;
     CreditTaken Courses.creditHour%TYPE;
     CourseTaken Students.totalCourseTaken%TYPE;

  BEGIN
     SELECT GPA, totalCreditTaken, totalCourseTaken
     INTO currentGPA, CreditTaken, CourseTaken
```

```
FROM students
WHERE username = :NEW.student;

SELECT creditHour INTO credit_hour
FROM courses
WHERE id = :NEW.courseID;

IF CourseTaken = 1 AND upper(:OLD.grade) <> 'I' THEN
    currentGPA := 0;
    CreditTaken:= CreditTaken;
    CourseTaken:= CourseTaken;
ELSIF CourseTaken = 1 AND upper(:OLD.grade) = 'I' THEN
   currentGPA := currentGPA;
   currentCredit := CreditTaken;
   CreditTaken := CreditTaken + credit_hour;
   CourseTaken := CourseTaken + 1;
ELSE
  IF upper(:OLD.grade) = 'A' THEN
    currentGPA := (currentGPA * CreditTaken - credit_hour * 4)/(CreditTaken - credit_hour);
  ELSIF upper(:OLD.grade) = 'B' THEN
    currentGPA := (currentGPA * CreditTaken - credit_hour * 3)/(CreditTaken - credit_hour);
  ELSIF upper(:OLD.grade) = 'C' THEN
    currentGPA := (currentGPA * CreditTaken - credit_hour * 2)/(CreditTaken - credit_hour);
  ELSIF upper(:OLD.grade) = 'D' THEN
    currentGPA := (currentGPA * CreditTaken - credit_hour * 1)/(CreditTaken - credit_hour);
  ELSE
    currentGPA := currentGPA;
    currentCredit := CreditTaken;
    CreditTaken := CreditTaken + credit_hour;
    CourseTaken:= CourseTaken + 1;
  END IF;
  currentCredit := CreditTaken - credit_hour;
  CourseTaken:= CourseTaken;
END IF;

IF upper(:NEW.grade) = 'A' THEN
   newGPA := (currentGPA * currentCredit + credit_hour * 4)/(currentCredit + credit_hour);
ELSIF upper(:NEW.grade) = 'B' THEN
   newGPA := (currentGPA* currentCredit + credit_hour * 3)/(currentCredit + credit_hour);
ELSIF upper(:NEW.grade) = 'C' THEN
   newGPA := (currentGPA* currentCredit + credit_hour * 2)/(currentCredit + credit_hour);
ELSIF upper(:NEW.grade) = 'D' THEN
   newGPA := (currentGPA* currentCredit+ credit_hour * 1)/(currentCredit + credit_hour);
```

```
    ELSE
      newGPA := currentGPA;
      CreditTaken := CreditTaken - credit_hour;
      CourseTaken:= CourseTaken - 1;
    END IF;

    UPDATE students
    SET  GPA = newGPA, totalCreditTaken = CreditTaken, totalCourseTaken = CourseTaken
    WHERE username = :NEW.student;
  END;
```

Question 7: When a student posts a comment in a forum, his total points should be updated based on the points criteria. Use a trigger.

```
create or replace TRIGGER addBonus
AFTER INSERT ON Comments
FOR EACH ROW
DECLARE
  group_type InterestGroupsClubs.type%TYPE;
  group_id InterestGroupsClubs.id%TYPE;
BEGIN
  SELECT groupID INTO group_id
  FROM DiscussionForums
  WHERE id = :New.forumID;

  SELECT type INTO group_type
  FROM InterestGroupsClubs
  WHERE id = group_id;

  IF LOWER(group_type) = 'coursegroup' THEN
    UPDATE Take
    SET TotalBouns = TotalBouns + 5
    WHERE student = (SELECT username FROM GroupUsers
                WHERE groupUser = :New.author)
        AND courseID = (SELECT id FROM Courses
                    WHERE groupID = group_id);
  END IF;
END;
```

Question 8: Write a stored procedure/function to add a student as the TA of a course. Write a trigger that will prevent the assignment if the student is enrolled in the course.

```
create or replace PROCEDURE addTA(
```

```
  course_id IN TAs.courseID%TYPE,
  course_TA IN TAs.TA%TYPE
)
AS
BEGIN
  INSERT INTO TAs
  VALUES(dbms_random.string('L', 5),course_id,course_TA);
END;

create or replace TRIGGER assignTA
BEFORE INSERT OR UPDATE ON TAs
FOR EACH ROW
DECLARE
  CURSOR c IS
    SELECT student FROM Take
    WHERE Take.courseID = :NEW.courseID;
  stu_enrolled c%ROWTYPE;
BEGIN
 OPEN c;
   LOOP
     FETCH c INTO stu_enrolled;
     EXIT WHEN c%NOTFOUND;
     IF :NEW.TA = stu_enrolled.student THEN
       raise_application_error (-20001, 'Sorry, the student ENROLLED in this course is not allowed
to be TA.');
     END IF;
   END LOOP;
 CLOSE c;
END;
```

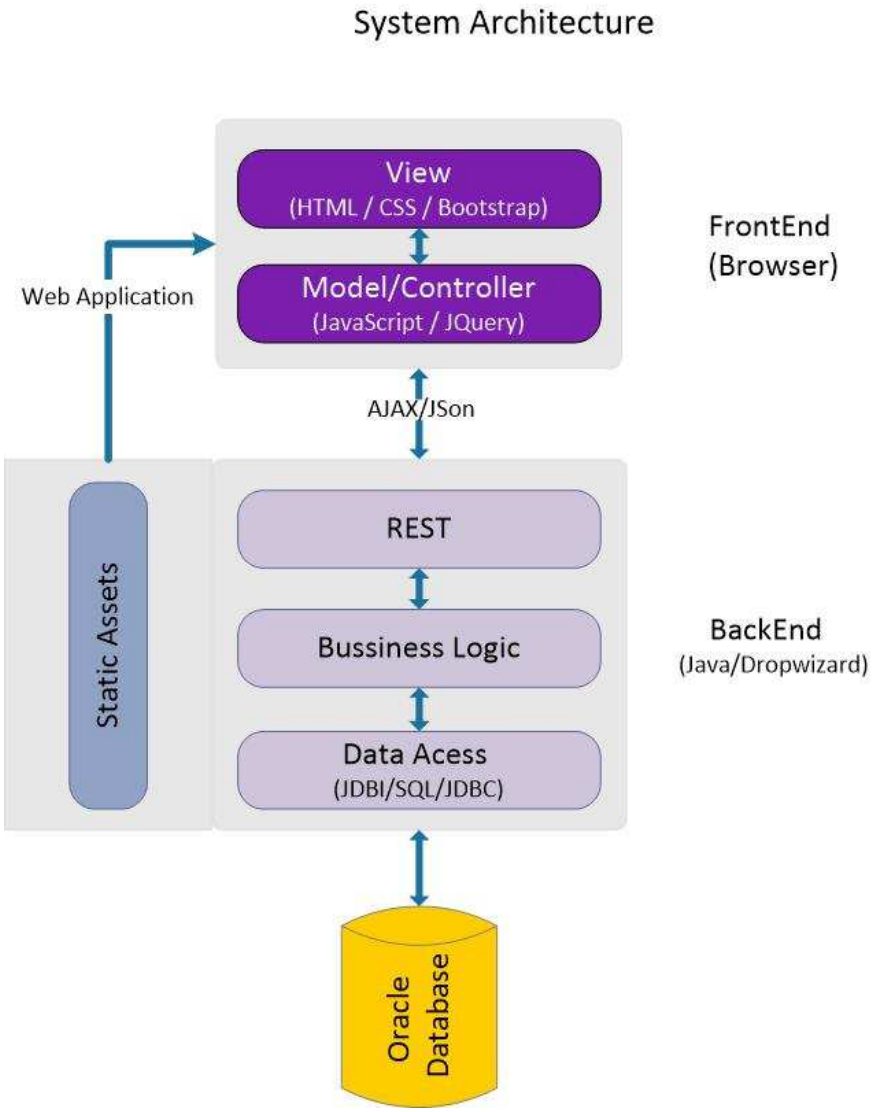
Question9: Write a procedure that allows the users of a group to post comments on the discussion forum. The procedure should verify that the user is a member of the group/club. If a non member tries to post, an alert should be sent to the moderators of the group.

```
create or replace PROCEDURE POSTCOMMENT(
  T IN Comments.topic%TYPE,
  M IN Comments.message%TYPE,
  A IN Comments.author%TYPE,
  F IN Comments.forumID%TYPE
)
AS
  group_id  GroupUsers.groupID%TYPE;
  moderator GroupUsers.groupUser%TYPE;
```

```
  CURSOR c IS
    SELECT groupUser
    FROM GroupUsers
    WHERE groupID =(SELECT groupID
                FROM DiscussionForums
                WHERE id = F);
  game c%ROWTYPE;

BEGIN
  INSERT INTO Comments
  VALUES (dbms_random.string('U', 5), T, M, A, F, sysdate);

  SELECT groupID INTO group_id
  FROM DiscussionForums
  WHERE id = F;

  OPEN c;
  LOOP
     FETCH c INTO game;
     EXIT WHEN c%NOTFOUND;
     IF game.groupUser = A THEN
       dbms_output.put_line('Thanks for your post.');
       GOTO stop_PROCEDURE;
     ELSE
       CONTINUE;
     END IF;
   END LOOP;
  CLOSE c;

  SELECT groupUser INTO moderator
  FROM GroupUsers
  WHERE groupID = group_id AND LOWER(isModerator) = 'yes'
  AND ROWNUM = 1;

  INSERT INTO Alerts
  VALUES (dbms_random.string('U', 5), 'A non member tries to post comment on group' ||
group_id, moderator, group_id);
<<stop_PROCEDURE>>
   dbms_output.put_line('');
END;
```

Part III

The following graph is the **System Architecture** for class project. It would help you to understand how the whole process works and it also indicates the languages and the frameworks we are going to use.



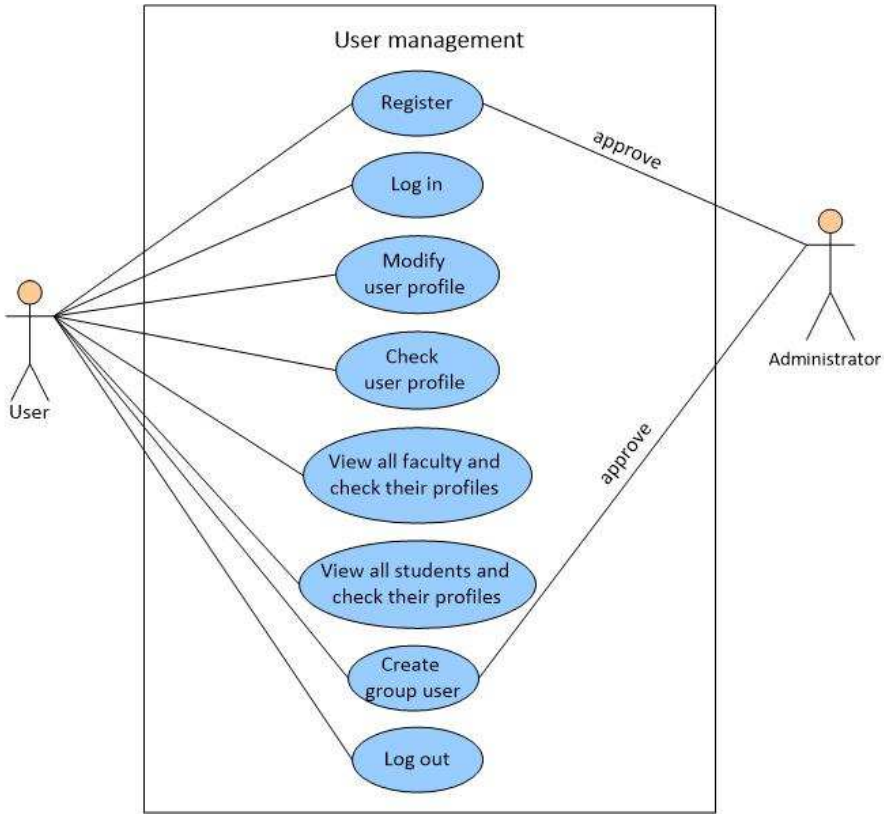
The main language we user is **JAVA**, **HTML, CSS, JavaScript, JQuery or Bootstrap**, the framework we use is **Dropwizard**.

Part IIII

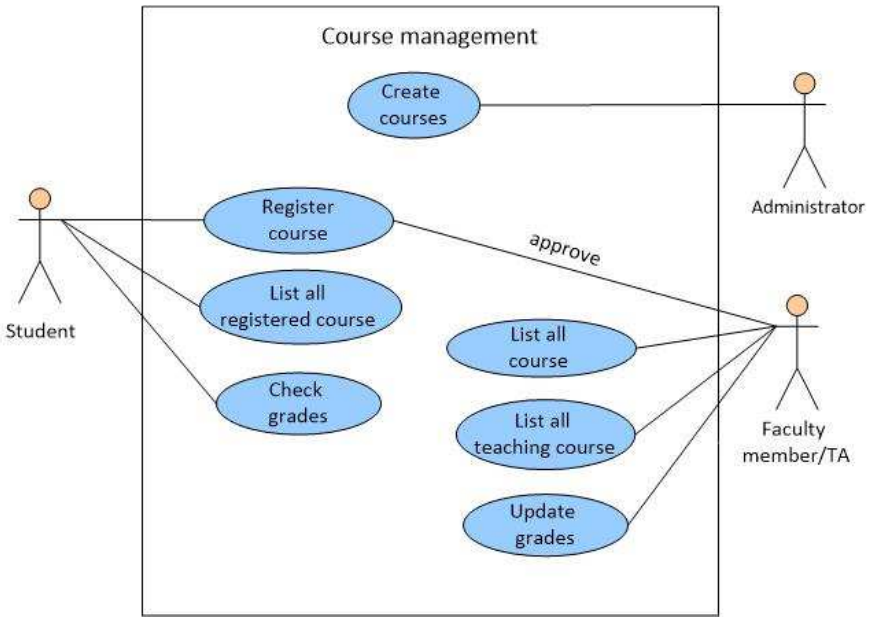
The specified feature ownership among team members

To identify and organize system requirements, the class project *Interactive Students/Faculties Networking system* has been divided into three large parts – **user management, course management, and discussion group management**, as following:

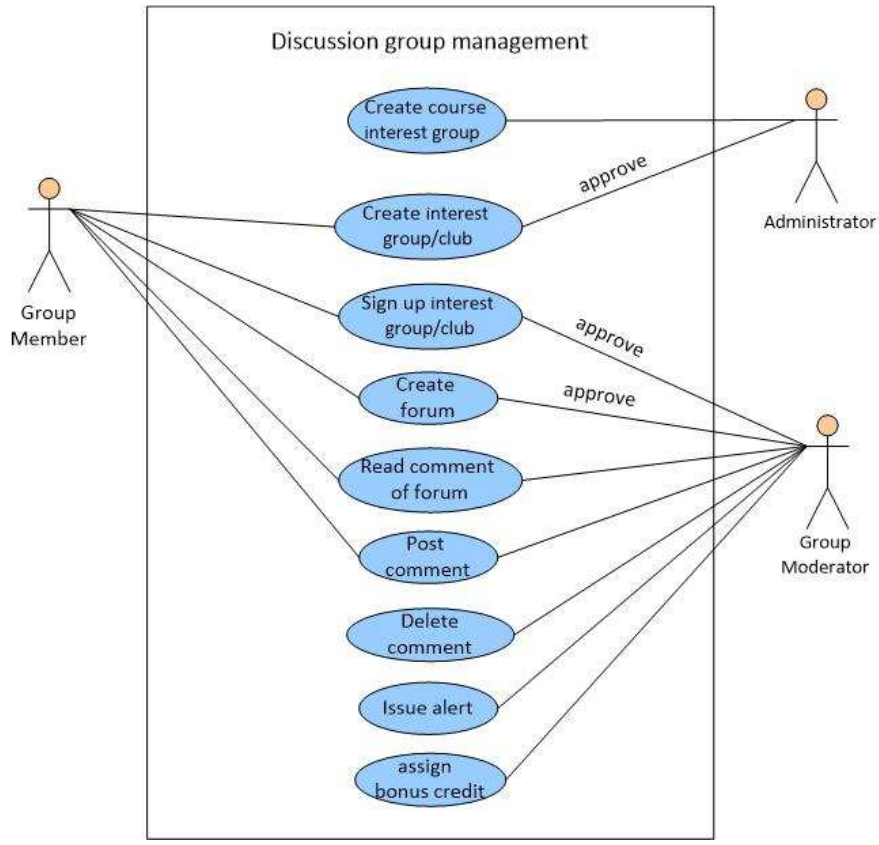
1. **User management**



2. Course management



3. Discussion group management



Feature ownership among group member is allocated according to user case diagram above

Feature ownership form

	User management	**Course management**	**Discussion group management**
Danna Liu	√		
Shanshan Jiang		√	
Zhizheng Li			√

The detail of role playing in the project:

Insert, update and delete:

	User/Faculty/Student/ researchProjects/internships/ GroupUsers	**Courses/Take/TAs**	**InterestGroupsClubs/ /Comments/ DiscussionForums/Alerts**
Danna Liu	√		
Shanshan Jiang		√	
Zhizheng Li			√

	Authorization and privileges	**General processes**	**Queries**
Danna Liu	1	1	2,5
Shanshan Jiang	2,3	3	3,4
Zhizheng Li		2,4,5	1,6