# Project Report
# Group 1 (DBPirates)

**I Assumptions**

- ➔ Professor is already part of the system and he/she can add courses to their courses list.
- ➔ Professor can add or remove TAs for a course.
- ➔ Based on questions added or removed, the number of questions in the exercise will either increase or decrease.
- ➔ CourseId is represented as number.
- ➔ Professor cannot view the course which is past the end date or before the start date.
- ➔ Notifications will be raised to the users when professor adds a homework.
- ➔ Notifications will be raised to the students the day when the homework is going to end

**II Problem Statement**

This project aims to develop a tool similar to Gradiance. Gradiance is an assessment tool used in many courses to lighten the work load on Professor and Teaching Assistants in evaluating assignments and providing generic notifications to students. It also provides content services with the aim of making learning more effective and efficient in structured contexts. Using this tool, the Students will be able to register for courses, attempt home works, view past submissions and evaluate their performance. This tool provides the flexibility on number of attempts for home works. The professor and Teaching Assistants will be able to view reports on the performance of the students.

**III Entities and Relationships**

There are a lot of entities and relationships that are involved in developing this tool. They are

1 Users: It has data related to Students, Professors and Teaching Assistants. Students have two levels namely Graduate and Under Graduate. Each user has Login ID, Password and generic user details like First Name, Last Name and Email Address and to uniquely identify users a column named user ID is present. Each tuple also has a column named Role ID which helps to identify if a user is student or Professor or Teaching Assistant.

2 Chapter: Each Chapter has an ID, title and must be associated with a Textbook and thus has a column indicating the TextBookISBN.

3 Course: Each course has an ID to uniquely indicate the course. Each course has a name and a level associated with it, as the same course may be present for both Graduates and Under Graduates.

4 CourseOfferings: CourseOfferings has all the generic details related to a given course namely StartDate, EndDate to indicate the duration of the course and MaximumEnrollmentNumber to indicate the maximum capacity of the class. To identify the tuples uniquely there are two columns namely CourseTokenID and CourseID.

5 CourseTeaching:CourseTeaching is a mapping between Users and CourseOfferings. It has UserID, CourseTokeniD, CourseID and RoleID. RoleID is used to indicate whether the user is Professor or Teaching assistant associated with the course.

(rkvardhi, sbobba3, skataka, spulima)

6 CourseTextBookMapping: This is a mapping between a given course and textbook associated with the course. This has the following columns, TextBookISBN and CourseTokenISBN.

7 CourseTopics: Each Course has a set of topics associated with it. This table gives an overview of all the topics present in a given course. It has CourseID, CourseTopicID and CourseName as columns.

8 ExerciseList:This table gives a detailed list of all the exercises/homeworks that are associated with a given Course and it also stores the list of topics on which the given Exercise is based on. It has ExerciseID, CourseTopicID and CourseTokenID.

9 ExerciseQuestion: This table is a mapping between Exercise and QuestionBank. It has both ExerciseID and QuestionID to indicate the list of questions that are related to a given Exercise.

10 Exercises: This table stores the generic information related to a given exercise namely, NumberofRetries, Markspercorrectanswer, Marksperincorrectanswer, Startdatetime, enddatetime, sscoringtype and numberofquestions, difficultylevel and an ID field to uniquely identify exercise.

11 Notifications: Notifications table has notificationid to uniquely identify the notification. Each tuple has notificationtext indicating the notification and CourseTokenID to map course to notification.

12 NotificationUserMapping: This acts as a mapping between notifications table and users table. It has a list of columns namely NotificationId, UserID and readflag. Readflag is used to indicate if a notification is seen or unseen by a given user.

13 QuestionBank: QuestionBank has a list of values that are related to a given question like QuestionText, DifficultyLevel, Hint and Explanation. Mapping between course and Questions is given by coursetokenid. Each question is uniquely identified by QuestionID.

14 QuestionParam: This is a specialized form of QuestionBank. It has QuestionID to uniquely identify a question. It also has ParameterID, ParameterValue and SetID. SetID is used to group the parameters into different sets, as there may be many sets of parameters for a given question.

15 Roles: Roles is a table to indicate the association between roleid and rolename.

16 StudentAttempt: Each StudentAttempt tuple is a relation between Users and Exercises. It has got UserID, ExerciseID, AttemptID, MarksObtained and SubmissionTime. Thus each tuple has all the details related to each attempt of a given user.

17 StudentAttemptQuestions: This is a mapping between Users, Exercises and QuestionBank. It has got UserID, ExerciseID, QuestionId and AttemptID to uniquely identify the tuplese. It also has all the options posted for a given question, the option user has selected and also SetID to indicate the set of parameters the given question is having.

18 StudentEnrollment:This acts a relationship between Users and Course. It indicates the list of courses a student is enrolled in.

(rkvardhi, sbobba3, skataka, spulima)

19 TextBooks: Each course has a textbook and even more than one course can have the same textbook. Eack tuple in a TextBooks table has a unique identification column named TextBookISBN and also details like title and author of the book.

20 Sections: Each chapter has a set of sections. The section table carries this information. It has TextBookISBN and ChapterID to indicate the textbook and chapter to which they belong to. It also has Section ID, Section Name and also Content.

21 Subsections: Each section has a set of sub sections. Thus each tuple in Subsections has SubsectionID, SectionID, ChapterID and TextbookISBN to indicate the link between textbook and Subsection. And it also has subsectionname and Content. Content gives the description of each sub section in a given textbook.

**IV Users**

There are three different users in this system.

**Users as Student:**

A Student can firstly login to the system. Firstly, he has options to update password and update account details and logout of the system.

Coming to the functionalities, a user can register for a particular course using the coursetokenid or look at the list of courses he/she has enrolled. Once he selects a particular course, he can attempt a homework which was posted, view past submissions, View notifications if any and also view the scores. View past submission helps the user understand his performance in a particular homework after he selects it in the Select Homework page, which pops up once view past submission button is clicked. Each time he tries to attempt a given homework, he gets a refreshed set of questions. In the notifications, there are a variety of notifications that get populated, like when a new homework is added, when password was not changed for about 3 months.

**Users as Professor:**

A professor has the generic options to login to the system, update password, update account details and logout of the system.

Coming to the functionalities, a professor can select a course from list of courses she is currently teaching. Once he/she selects a course, she has option to add homework, add or remove questions to a given homework, edit homework, view homework, view notifications, view reports and add or remove TA. In add homework, professor has to give all the details related to it, like Number of questions, marks per correct answers, marks per incorrect answer, start and end date etc. In add or remove questions professor can add or remove questions to a selected homework respectively. Edit homework helps professor to edit the details of a selected homework, like start or end date, number of questions, number of retries, difficulty level of questions etc. In view homework, professor can select a particular homework and check the details pertaining to selected homework and also view all the questions that

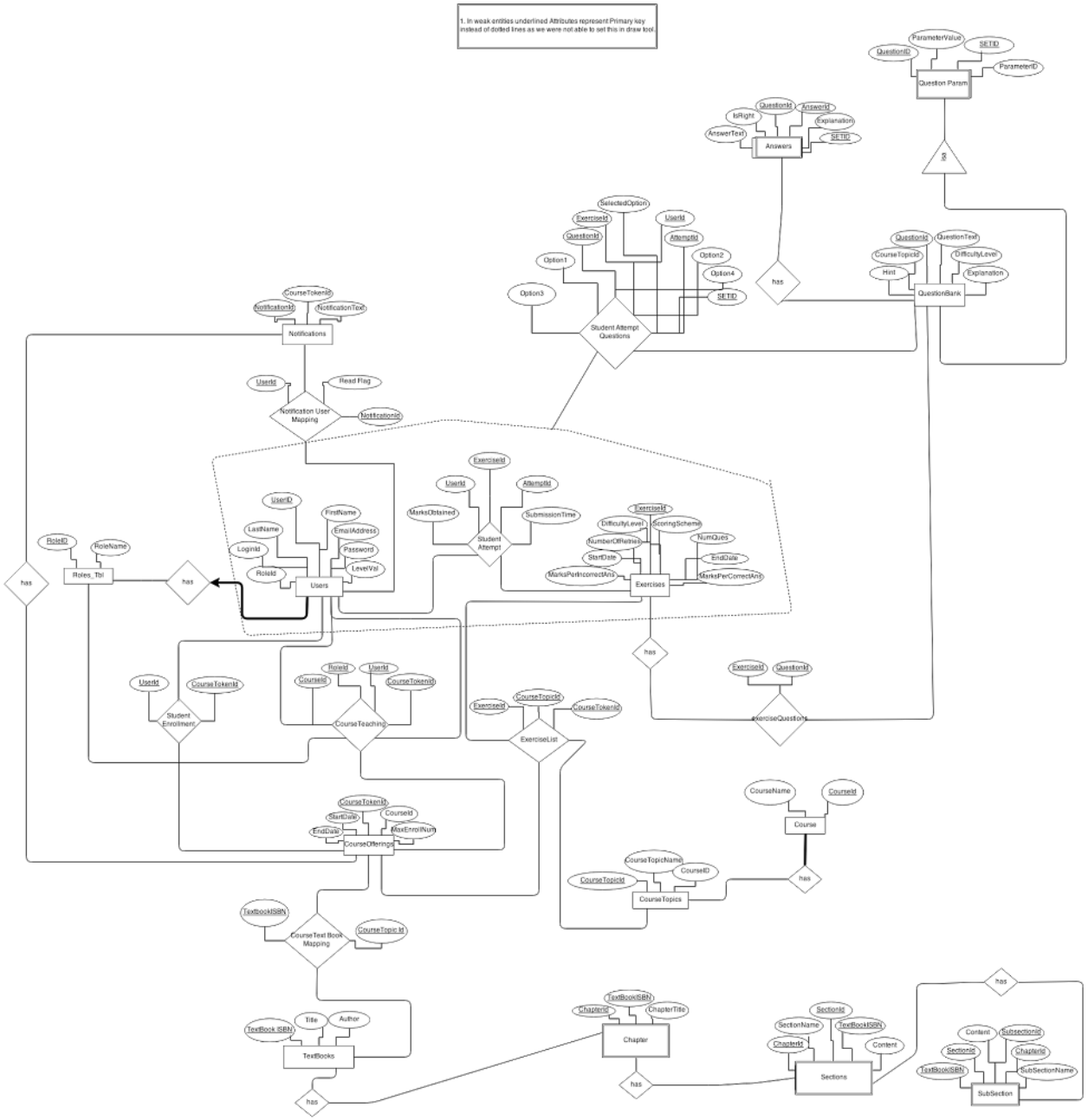
were added to a particular homework. Also professor can select topics to design the homework on those topics only. In notifications, notifications for password change appear. In reports professor can view statistics of the class, like list of students who did not attempt a particular homework, list of students who got maximum marks in a selected homework, average attempts of a given homework etc.

Professor has the ability to add or remove TA for a particular course. To add a TA, professor must have an idea of the UserID of the student and to remove a TA, professor can select UserID of a particular TA and remove him/her.

**Users as Teaching Assistant:**

Based on the course he selects, a user is treated either as Student or a TA. For example, If he selects a course for which he is a Student then the view is of a student. And as a TA, he/she will not have all the privileges of a professor; they can only view a selected homework, view notifications and view reports. TA also has all the generic functionalities like update password, update account details, login and log out of the system.

**V ER Diagram**




(rkvardhi, sbobba3, skataka, spulima)

**VI Relational Schemas**

Converting the ER diagram into Relational Schema:

1. CREATE TABLE ANSWERS (QUESTIONID NUMBER, ISRIGHT CHAR(1 BYTE), EXPLANATION VARCHAR2(500 BYTE), ANSWERID NUMBER(2,0), ANSWERTEXT VARCHAR2(500 BYTE) NOT NULL, SETID NUMBER(30,0), ADD CONSTRAINT ISRIGHT_CHK CHECK (ISRIGHT IN ('Y','N')), ADD CONSTRAINT PK_ANSWERS PRIMARY KEY (SETID, ANSWERID, QUESTIONID), ADD CONSTRAINT FK_QUESTIONBANK_ANSWERS FOREIGN KEY (QUESTIONID) REFERENCES QUESTIONBANK(QUESTIONID))

2. CREATE TABLE CHAPTER (TEXTBOOKISBN VARCHAR2 (30 BYTE), CHAPTERID NUMBER(10,0), CHAPTERTITLE VARCHAR2(50 BYTE) NOT NULL, ADD CONSTRAINT PK_CHAPTER PRIMARY KEY (TEXTBOOKISBN, CHAPTERID), ADD CONSTRAINT FK_ISBN FOREIGN KEY (TEXTBOOKISBN) REFERENCES TEXTBOOKS(TEXTBOOKISBN))

3. CREATE TABLE COURSE (COURSEID NUMBER (10,0), COURSENAME VARCHAR2(100 BYTE) NOT NULL, COURSELEVEL VARCHAR2(100 BYTE), ADD CONSTRAINT PK_COURSEID PRIMARY KEY (COURSEID), ADD CONSTRAINT CHECK_LEVEL CHECK (COURSELEVEL IN ('UnderGraduate', 'Graduate')))

4. CREATE TABLE COURSEOFFERINGS (COURSETOKENID VARCHAR2 (15 BYTE), COURSEID NUMBER(10,0), STARTDATE DATE, ENDDATE DATE, MAXIMUMENROLLMENTNUMBER NUMBER(5,0), ADD CONSTRAINT PK_COURSEOFFERING PRIMARY KEY (COURSETOKENID), ADD CONSTRAINT FK_COURSEOFFER_COURSEID FOREIGN KEY (COURSEID) REFERENCES COURSE(COURSEID))

5. CREATE TABLE COURSETEACHING (USERID NUMBER(10,0), COURSEID NUMBER(10,0), COURSETOKENID VARCHAR2(15 BYTE), ROLEID NUMBER(2,0), ADD CONSTRAINT PK_COURSETEACHING PRIMARY KEY (USERID, COURSEID, COURSETOKENID), ADD CONSTRAINT CHECK_ROLEID CHECK (ROLEID IN (1,3)), ADD CONSTRAINT FK_USERS_COURSETEACHING FOREIGN KEY (USERID) REFERENCES USERS (USERID), ADD CONSTRAINT FK_COURSE_COURSETEACHING FOREIGN KEY (COURSEID) REFERENCES COURSE (COURSEID), ADD CONSTRAINT FK_CO_CT FOREIGN KEY (COURSETOKENID) REFERENCES COURSEOFFERINGS(COURSETOKENID))

6. CREATE TABLE COURSETEXTBOOKMAPPING (TEXTBOOKISBN VARCHAR2(30 BYTE) NOT NULL, COURSETOKENID VARCHAR2(30 BYTE) NOT NULL, CONSTRAINT COURSETEXTBOOKMAPPING_PK PRIMARY KEY (TEXTBOOKISBN, COURSETOKENID), ADD CONSTRAINT FK_TEXTBOOKISBN FOREIGN KEY (TEXTBOOKISBN) REFERENCES TEXTBOOKS(TEXTBOOKISBN), ADD CONSTRAINT FK_COURSETOKEN FOREIGN KEY (COURSETOKENID) REFERENCES COURSEOFFERINGS (COURSETOKENID))

7. CREATE TABLE COURSETOPICS (COURSETOPICID NUMBER (10,0), COURSETOPICNAME VARCHAR2(100 BYTE) NOT NULL, COURSEID NUMBER(10,0), ADD CONSTRAINT PK_COURSETOPICID PRIMARY KEY (COURSETOPICID), ADD CONSTRAINT FK_COURSE_COURSEID FOREIGN KEY (COURSEID) REFERENCES COURSE (COURSEID))

8. CREATE TABLE EXERCISELIST (EXERCISEID NUMBER, COURSETOPICID NUMBER NOT NULL, COURSETOKENID VARCHAR2(15 BYTE), ADD CONSTRAINT PK_EL PRIMARY KEY (EXERCISEID,

COURSETOKENID), ADD CONSTRAINT FK_EXEID FOREIGN KEY (EXERCISEID) REFERENCES EXERCISES (EXERCISEID))

9. CREATE TABLE EXERCISEQUESTION (EXERCISEID NUMBER(10,0), QUESTIONID NUMBER(10,0), ADD CONSTRAINT PK_EXERCISEQUESTION PRIMARY KEY (EXERCISEID, QUESTIONID), ADD CONSTRAINT FK_EXERCISES_EXERQUESTION FOREIGN KEY (EXERCISEID) REFERENCES EXERCISES(EXERCISEID), ADD CONSTRAINT FK_QUESTIONBANK_EXERQUESTION FOREIGN KEY(QUESTIONID) REFERENCES QUESTIONBANK(QUESTIONID))

10. CREATE TABLE EXERCISES (EXERCISEID NUMBER(10,0), DIFFICULTYLEVEL NUMBER(2,0) NOT NULL, NUMBEROFRETRIES NUMBER(2,0) NOT NULL, MARKSPERCORRECTANSWER NUMBER(2,0) NOT NULL, MARKSPERINCORRECTANSWER NUMBER(2,0) NOT NULL, STARTDATETIME DATE NOT NULL, ENDDATETIME DATE NOT NULL, SCORINGTYPE VARCHAR2(20 BYTE) NOT NULL, NUMBEROFQUESTIONS NUMBER NOT NULL, ADD CONSTRAINT PK_EXERCISES PRIMARY KEY (EXERCISEID), ADD CONSTRAINT EXERCISES_CHK1 CHECK (DIFFICULTYLEVEL in (1,2,3,4,5,6)))

11. CREATE TABLE NOTIFICATIONS (NOTIFICATIONID NUMBER(3,0), NOTIFICATIONTEXT VARCHAR2(1000 BYTE), COURSETOKENID VARCHAR2(25 BYTE), ADD CONSTRAINT PK_NOTIFICATIONS PRIMARY KEY (NOTIFICATIONID), ADD CONSTRAINT FK_CKID FOREIGN KEY (COURSETOKENID) COURSEOFFERINGS(COURSETOKENID))

12. CREATE TABLE NOTIFICATIONUSERMAPPING (USERID NUMBER(5,0), NOTIFICATIONID NUMBER(3,0), READFLAG NUMBER, ADD CONSTRAINT CHECK_0_1 CHECK (READFLAG IN (0, 1)), ADD CONSTRAINT PK_NUMP PRIMARY KEY (USERID, NOTIFICATIONID), ADD CONSTRAINT FK_USRID FOREIGN KEY (USERID) REFERENCES USERS (USERID), ADD CONSTRAINT FK_NID FOREIGN KEY (NOTIFICATIONID) REFERENCES NOTIFICATIONS (NOTIFICATIONID))

13. CREATE TABLE QUESTIONANSWERMAPPING (EXERCISEID NUMBER(10,0), QUESTIONID NUMBER(10,0), ANSWERID NUMBER(2,0), ADD CONSTRAINT PK_QUESTIONASNWERMAPPING PRIMARY KEY (EXERCISEID, QUESTIONID, ANSWERID, ADD CONSTRAINT FK_EXERCISES_QUESTIONANRMAP FOREIGN KEY (EXERCISEID) REFERENCES EXERCISES (EXERCISEID), ADD CONSTRAINT FK_QUESTIONBANK_QUESANSRMAP FOREIGN KEY (QUESTIONID) REFERENCES QUESTIONBANK (QUESTIONID))

14. CREATE TABLE QUESTIONBANK (QUESTIONID NUMBER(10,0), QUESTIONTEXT VARCHAR2(500 BYTE), COURSETOPICID NUMBER(10,0), DIFFICULTYLEVEL NUMBER(2,0) NOT NULL, HINT VARCHAR2(500 BYTE), EXPLANATION VARCHAR2(500 BYTE), ADD CONSTRAINT CHECK_DIFFICULTYLEVEL CHECK (DifficultyLevel BETWEEN 1 and 6), ADD CONSTRAINT PK_QUESTIONBANK PRIMARY KEY (QUESTIONID), ADD CONSTRAINT FK_COURSETOPICS_QUESTIONBANK FOREIGN KEY (COURSETOPICID) REFERENCES COURSETOPICS (COURSETOPICID))

15. CREATE TABLE QUESTIONPARAM (QUESTIONID NUMBER(30,0) NOT NULL, PARAMETERVALUE VARCHAR2(100 BYTE) NOT NULL, SETID NUMBER(30,0) NOT NULL, PARAMETERID NUMBER(5,0), ADD CONSTRAINT QUESTIONPARAM_PK PRIMARY KEY (QUESTIONID, PARAMETERVALUE, SETID))

(rkvardhi, sbobba3, skataka, spulima)

16. CREATE TABLE ROLES_TBL (ROLEID NUMBER(10,0), ROLENAME VARCHAR2(20 BYTE) NOT NULL, ADD CONSTRAINT PK_ROLEID PRIMARY KEY (ROLEID))

17. CREATE TABLE SECTIONS (SECTIONID NUMBER(10,0), SECTIONNAME VARCHAR2(100 BYTE) NOT NULL, TEXTBOOKISBN VARCHAR2(30 BYTE), CHAPTERID NUMBER(10,0), CONTENTVAL VARCHAR2(4000 BYTE) NOT NULL, ADD CONSTRAINT SECTIONS_PK PRIMARY KEY (SECTIONID, TEXTBOOKISBN, CHAPTERID), ADD CONSTRAINT FK_TXTISBN FOREIGN KEY (TEXTBOOKISBN) REFERENCES TEXTBOOKS (TEXTBOOKISBN))

18. CREATE TABLE STUDENTATTEMPT (EXERCISEID NUMBER(10,0), USERID NUMBER(10,0), ATTEMPTID NUMBER(10,0), MARKSOBTAINED NUMBER(10,0), SUBMISSIONTIME DATE, CONSTRAINT PK_STUDENTATTEMPT PRIMARY KEY (EXERCISEID, USERID, ATTEMPTID), ADD CONSTRAINT FK_EXERCISES_STUDENTATTEMPT FOREIGN KEY (EXERCISEID) REFERENCES EXERCISES (EXERCISEID), ADD CONSTRAINT FK_USERS_STUDENTATTEMPT FOREIGN KEY (USERID) REFERENCES USERS (USERID))

19. CREATE TABLE STUDENTATTEMPTQUESTIONS (EXERCISEID NUMBER(10,0) NOT NULL, USERID NUMBER(10,0) NOT NULL, ATTEMPTID NUMBER(10,0) NOT NULL, QUESTIONID NUMBER(10,0) NOT NULL, OPTION1 NUMBER(2,0), OPTION2 NUMBER(2,0), OPTION3 NUMBER(2,0), OPTION4 NUMBER(2,0), SELECTEDOPTION NUMBER(2,0), SETID NUMBER(5,0), ADD CONSTRAINT PK_STUDATTEMPTQUEST PRIMARY KEY (EXERCISEID, USERID, QUESTIONID, ATTEMPTID), ADD CONSTRAINT STUDENTATTEMPTQUESTIONS_FK1 FOREIGN KEY (QUESTIONID) REFERENCES QUESTIONBANK (QUESTIONID), ADD CONSTRAINT STUDENTATTEMPTQUESTIONS_FK2 FOREIGN KEY (USERID) REFERENCES USERS (USERID), ADD CONSTRAINT STUDENTATTEMPTQUESTIONS_FK3 FOREIGN KEY (EXERCISEID) REFERENCES EXERCISES(EXERCISEID))

20. CREATE TABLE STUDENTENROLLMENT (USERID NUMBER(10,0), COURSETOKENID VARCHAR2(15 BYTE), ADD CONSTRAINT PK_STDENR PRIMARY KEY (COURSETOKENID, USERID), ADD CONSTRAINT FK_STDENR_COURSETOKENID FOREIGN KEY (COURSETOKENID) REFERENCES COURSEOFFERINGS (COURSETOKENID), ADD CONSTRAINT FK_STENR_USERID FOREIGN KEY (USERID) REFERENCES USERS (USERID))

21. CREATE TABLE SUBSECTIONS (SUBSECTIONID NUMBER(10,0), SECTIONID NUMBER(10,0), CHAPTERID NUMBER(10,0), TEXTBOOKISBN VARCHAR2(30 BYTE), SUBSECTIONNAME VARCHAR2(40 BYTE), CONTENT VARCHAR2(4000 BYTE) NOT NULL, ADD CONSTRAINT PK_SUBSECTIONS PRIMARY KEY (SUBSECTIONID, SECTIONID, CHAPTERID, TEXTBOOKISBN, SUBSECTIONNAME), ADD CONSTRAINT FK_ISB FOREIGN KEY (TEXTBOOKISBN) REFERENCES TEXTBOOKS (TEXTBOOKISBN))

22. CREATE TABLE TEXTBOOKS (TEXTBOOKISBN VARCHAR2(30 BYTE), TITLE VARCHAR2(100 BYTE) NOT NULL, AUTHOR VARCHAR2(100 BYTE) NOT NULL, ADD CONSTRAINT PK_TEXTBOOKS PRIMARY KEY (TEXTBOOKISBN))

23. CREATE TABLE USERS (USERID NUMBER(10,0), FIRSTNAME VARCHAR2(40 BYTE), LASTNAME VARCHAR2(20 BYTE), EMAILADDRESS VARCHAR2(50 BYTE), ROLEID NUMBER(10,0), LEVELVAL VARCHAR2(20 BYTE), LOGINID VARCHAR2(10 BYTE), PASSWORDVAL VARCHAR2(20 BYTE), ADD

CONSTRAINT PK_USERS PRIMARY KEY (USERID), ADD CONSTRAINT UNIQUE_LOGINID UNIQUE (LOGINID), ADD CONSTRAINT CHK_LEVEL CHECK (levelval in ('Graduate','UnderGraduate','Professor')))

**VII Functional Dependencies and Normal Forms**

**Course:**

This table is decomposed to BCNF.

CourseId ->CourseName

CourseId is Primary key. Hence it is in BCNF.

**CourseTopic:**

This table is decomposed to BCNF.

{CourseTopicId, CourseId} ->CourseTopicName.

{CourseTopicId, CourseId} is a super key for Course Topic. This is in BCNF.

**CourseOffering:**

CourseTokenId ->{CourseId, Start Date, End Date, CourseId ,Max Enroll Number}

Since Course Token id is primary key and determines all the fields, Course Offering is in BCNF.

**Course Teaching:**

{UserId, CourseTokenId} ->CourseId, RoleId

{UserId, CourseTokenId, CourseId} ->RoleId

This is in 3NF.

**TextBook:**

TextBookISBN ->Title, Author

This is in BCNF as TextbookISBN is the key.

**Chapter:**

ChapterId, TextBookISBN ->ChapterName

This is in BCNF as ChapterId and Textbook ISBN is Primary Key.

**Sections:**

SectionId, ChapterId, TextBookISBN -> Section Name, Content

SectionId, ChapterId, TextBookISBN is primary key and we can derive all columns with it and since this is the only FD possible, this is in BCNF.

**SubSections:**

SubSectionId, SectionId, ChapterId, TextBookISBN -> Section Name, Content

SubSectionId, SectionId, ChapterId, TextBookISBN is primary key and we can derive all columns with it and since this is the only FD possible, this is in BCNF.

**Users:**

UserId ->FirstName, LastName, EmailAddress, Password, RoleId, Level Val, LoginId

LoginId ->FirstName, LastName, EmailAddress, Password, RoleId, Level Val, UserId

This is in BCNF as both UserID and Loginid are keys and determine all other fields.

**Role_tbl:**

RoleId ->RoleName

This is in BCNF as RoleID determines all other fields.

**Exercises:**

ExerciseId ->StartDate, EndDate, Num of Questions, marks per correct answer, marks per incorrect answer, Num of Retries, difficulty level, scoring scheme

This is in BCNF as ExerciseID will determine all the fields.

**QuestionBank:**

QuestionId ->QuestionText, DifficultyLevel, CourseTopicID, Hint, Explanation

This is in BCNF as QuestionId will determine all the fields.

**QuestionsParam:**

QuestionId, SetId, ParameterId ->Parameter Value

This is in BCNF as QuestionId,SetId, ParameterId will determine all the fields.

**Answers:**

QuestionId, AnswerId, SetID ->Explanation, IsRight, AnswerText

This is in BCNF as QuestionId, SetId, AnswerId will determine all the fields.

**StudentAttemptQuestions:**

(rkvardhi, sbobba3, skataka, spulima)

ExerciseId, UserID, QuestionId, AttemptId, SetID -> Option1, Option2, Option3, Option4, SelectedOption

This is in BCNF as QuestionId, SetId, ExerciseId, AttemptId will determine all the fields.

**ExerciseList:**

All the fields (ExerciseId, CourseTopicId, CourseTokenID) in this table are part of Composite Primary Key. This is in BCNF.

**ExerciseQuestions:**

All the fields (ExerciseId, QuestionId) in this table are part of Composite Primary Key. This is in BCNF.

**StudentAttempt:**

UserId, AttemptId, ExerciseId ->MarksObtained, SubmissionTime

This is in BCNF as UserId, ExerciseId, AttemptId determines all the fields.

**Notifications:**

NotificationId ->CourseTokenID, NotificationText

This is in BCNF as NotificationID determines all the fields.

**NotificationUserMapping:**

NotificationId, UserId ->ReadFlag

This is in BCNF as NotificationID determines all the fields.

(rkvardhi, sbobba3, skataka, spulima)

**VIII SQL Queries**

1) Find students who did not take homework 1.

```
SELECT USERID FROM
(SELECTS1.USERID, COUNT (S1.ATTEMPTID) AS
COUNTATTEMPTSFROMSTUDENTATTEMPTS1 WHERES1.EXERCISEID =
(SELECTMIN(EXERCISEID) FROMEXERCISELISTWHERE COURSETOKENID='CSC440FALL14'
GROUPBYCOURSETOKENIDHAVINGCOUNT(EXERCISEID) >0)
GROUP BY USERID) TEMPTABLE
WHERE TEMPTABLE.COUNTATTEMPTS< 2;
```

2) Find students who scored the maximum score on the first attempt for homework 1.

```
SELECT DISTINCT USERID,MARKSOBTAINED FROM
        (SELECT S1.MARKSOBTAINED,S1.USERID FROM
        STUDENTATTEMPT S1,EXERCISELIST EL
        WHERE S1.EXERCISEID = EL.EXERCISEID
        AND EL.COURSETOKENID = 'CSC440FALL14'
        AND S1.ATTEMPTID = 1
        AND S1.EXERCISEID = (SELECT MIN(EL1.EXERCISEID)
        FROM EXERCISELIST EL1 WHERE EL1.COURSETOKENID='CSC440FALL14'
        GROUP BY EL1.COURSETOKENID HAVING COUNT(EL1.EXERCISEID) >0))
TEMPTABLE
        WHERE TEMPTABLE.MARKSOBTAINED =
        (SELECT MAX(S2.MARKSOBTAINED) FROM STUDENTATTEMPT S2,EXERCISELIST
EL2
        WHERE S2.EXERCISEID = EL2.EXERCISEID
        AND EL2.COURSETOKENID = 'CSC440FALL14'
        AND S2.ATTEMPTID = 1
        AND S2.EXERCISEID = (SELECT MIN(EL3.EXERCISEID) FROM EXERCISELIST EL3
WHERE EL3.COURSETOKENID='CSC440FALL14'
        GROUP BY EL3.COURSETOKENID HAVING COUNT(EL3.EXERCISEID) >0));
```

3) Find students who scored the maximum score on the first attempt for each homework.

```
 SELECT DISTINCT S2.USERID,TEMPTABLE.EXERCISEID,TEMPTABLE.MAXMARKS FROM
                (SELECT S.EXERCISEID,MAX(S.MARKSOBTAINED) AS MAXMARKS
                FROM STUDENTATTEMPT S,EXERCISELIST E
                WHERE S.EXERCISEID = E.EXERCISEID
                AND E.COURSETOKENID = 'CSC440FALL14'
                AND S.ATTEMPTID = 1
                GROUP BY S.EXERCISEID) TEMPTABLE,STUDENTATTEMPT S2
                WHERE S2.EXERCISEID = TEMPTABLE.EXERCISEID
                AND S2.MARKSOBTAINED = TEMPTABLE.MAXMARKS;
```

(rkvardhi, sbobba3, skataka, spulima)

4) For each student, show total score for each homework and average score across all home works.

```
CREATE VIEW USERMARKSVIEW AS
SELECT TEMPTABLE2.USERID,TEMPTABLE2.TOTAL_SCORE,TEMPTABLE2.EXERCISEID FROM
        (SELECT TEMPTABLE.USERID,NVL(MAX(TEMPTABLE.MARKSOBTAINED),0) AS
TOTAL_SCORE,TEMPTABLE.EXERCISEID
        FROM (SELECT S.USERID,S.ATTEMPTID,S.EXERCISEID,S.MARKSOBTAINED
        FROM STUDENTATTEMPT S,EXERCISELIST EL,EXERCISES E
        WHERE S.EXERCISEID = EL.EXERCISEID
        AND S.EXERCISEID = E.EXERCISEID
        AND EL.COURSETOKENID = 'CSC440FALL14'
        AND E.SCORINGTYPE = 'MAX') TEMPTABLE
        GROUP BY TEMPTABLE.USERID,TEMPTABLE.EXERCISEID
        UNION
        SELECT TEMPTABLE.USERID,NVL(AVG(TEMPTABLE.MARKSOBTAINED),0) AS
TOTAL_SCORE,TEMPTABLE.EXERCISEID
        FROM (SELECT S.USERID,S.ATTEMPTID,S.EXERCISEID,S.MARKSOBTAINED
        FROM STUDENTATTEMPT S,EXERCISELIST EL,EXERCISES E
        WHERE S.EXERCISEID = EL.EXERCISEID
        AND S.EXERCISEID = E.EXERCISEID
        AND EL.COURSETOKENID = 'CSC440FALL14'
        AND E.SCORINGTYPE = 'AVERAGE') TEMPTABLE
        GROUP BY TEMPTABLE.USERID,TEMPTABLE.EXERCISEID
        UNION
        SELECT S.USERID,NVL(S.MARKSOBTAINED,0) AS TOTAL_SCORE,S.EXERCISEID
        FROM STUDENTATTEMPT S,EXERCISELIST EL,EXERCISES E
        WHERE S.EXERCISEID = EL.EXERCISEID
        AND S.EXERCISEID = E.EXERCISEID
        AND EL.COURSETOKENID = 'CSC440FALL14'
        AND E.SCORINGTYPE = 'AVERAGE'
        AND S.ATTEMPTID = (SELECT MAX(S2.ATTEMPTID) FROM STUDENTATTEMPT
S2
        WHERE S2.EXERCISEID = S.EXERCISEID AND S2.USERID = S.USERID
        GROUP BY S2.USERID,S2.EXERCISEID)) TEMPTABLE2
        ORDER BY TEMPTABLE2.USERID,TEMPTABLE2.EXERCISEID;

 SELECT * FROM USERMARKSVIEW;

SELECT USERID,AVG(TOTAL_SCORE)
FROM USERMARKSVIEW
GROUP BY USERID;
```

(rkvardhi, sbobba3, skataka, spulima)

5) For each homework, show average number of attempts

```
SELECT TEMPTAB.EXERCISEID, AVG(TEMPTAB.COUNTATTEMPTS) AS AVGATTEMPTS FROM
        (SELECT S.EXERCISEID,COUNT(*) AS COUNTATTEMPTS
        FROM EXERCISELIST E,STUDENTATTEMPT S
        WHERE E.COURSETOKENID = 'CSC440FALL14'
        AND S.EXERCISEID = E.EXERCISEID
        AND S.ATTEMPTID > 0
        GROUP BY S.USERID,S.EXERCISEID) TEMPTAB
        GROUP BY EXERCISEID;
```

**Triggers:**

1) When a question is added to the homework i.e ExerciseQuestion table, trigger is fired and the NumberofQuestions count in Exercise table will be incremented.

```
CREATE OR REPLACE TRIGGER MODIFY_QUESTIONSCOUNT_INSERT

AFTER INSERT ON EXERCISEQUESTION

FOR EACH ROW

DECLARE MAXQUESTIONCOUNT INT;

BEGIN

SELECT NUMBEROFQUESTIONS INTO MAXQUESTIONCOUNT FROM EXERCISES

WHERE EXERCISEID = :NEW.EXERCISEID;

UPDATE EXERCISES

SET NUMBEROFQUESTIONS = (MAXQUESTIONCOUNT + 1)

WHERE EXERCISEID = :NEW.EXERCISEID;

END;
```

2) When a question is deleted from the homework i.e ExerciseQuestion table, trigger is fired and the NumberofQuestions count in Exercise table will be decremented.

```
CREATE OR REPLACE TRIGGER MODIFY_QUESTIONSCOUNT_DELETE

AFTER DELETE ON EXERCISEQUESTION

FOR EACH ROW

DECLARE MAXQUESTIONCOUNT INT;

BEGIN

SELECT NUMBEROFQUESTIONS INTO MAXQUESTIONCOUNT FROM EXERCISES

WHERE EXERCISEID = :OLD.EXERCISEID;

UPDATE EXERCISES

SET NUMBEROFQUESTIONS = (MAXQUESTIONCOUNT - 1)

WHERE EXERCISEID = :OLD.EXERCISEID;

END;
```

**Procedures:**

Few of the procedures used:

1) CREATE OR REPLACE PROCEDURE CHECK_COURSES

```
(LOGIN_ID IN VARCHAR2,COURSE_ID IN INT,RETVALUE OUT INT,COURSE_TOKENID OUT
VARCHAR2

) AS

USER_ID NUMBER;

COURSECOUNT NUMBER;

BEGIN

COURSECOUNT := 0;

USER_ID := MAPLOGINIDTOUSERID(LOGIN_ID);

SELECT COUNT(S.COURSETOKENID) INTO COURSECOUNT FROM STUDENTENROLLMENT S,
COURSEOFFERINGS CO, COURSE CS
```

```
        WHERE

        S.COURSETOKENID = CO.COURSETOKENID

        AND CO.COURSEID = COURSE_ID

        AND CO.COURSEID = CS.COURSEID

        AND SYSDATE BETWEEN CO.STARTDATE AND CO.ENDDATE

        AND S.USERID = USER_ID   ;

  IF(COURSECOUNT > 0)

  THEN

  RETVALUE := 1;

  SELECT COURSETOKENID INTO COURSE_TOKENID FROM COURSEOFFERINGS WHERE
COURSEID = COURSE_ID;

  ELSE

  SELECT COUNT(CT.COURSEID) INTO COURSECOUNT FROM COURSETEACHING CT,
COURSEOFFERINGS CO, COURSE CS

        WHERE CT.COURSETOKENID = CO.COURSETOKENID

        AND CT.COURSEID = COURSE_ID

        AND CO.COURSEID = CS.COURSEID

        AND SYSDATE BETWEEN CO.STARTDATE AND CO.ENDDATE

        AND CT.USERID = USER_ID

        AND CT.ROLEID = 3;


     IF(COURSECOUNT > 0)

     THEN

     RETVALUE := 2;

     SELECT COURSETOKENID INTO COURSE_TOKENID FROM COURSEOFFERINGS WHERE
COURSEID = COURSE_ID;

     ELSE
```

```
    RETVALUE := 3;
    END IF;
  END IF;
END;
```

2) 
```
CREATE OR REPLACE PROCEDURE CREATE_USER(FIRST_NAME IN VARCHAR2,
LAST_NAME IN VARCHAR2,
EMAIL_ADDRESS IN VARCHAR2, LEVELVAL IN VARCHAR2, LOGIN_ID IN VARCHAR2,
PASSWORDVAL IN VARCHAR2,
RET_VALUE OUT NUMBER)
AS
MAXUSERID NUMBER(5);
ROLETEMP NUMBER(5);
COUNTE NUMBER(5);
BEGIN

RET_VALUE := 0;
SELECT MAX(USERID)INTO MAXUSERID FROM USERS;

MAXUSERID := MAXUSERID + 1;

IF FIRST_NAME IS NULL
THEN
RET_VALUE := -1;
ELSIF LAST_NAME IS NULL
THEN
RET_VALUE := -2;
```

```
ELSIF EMAIL_ADDRESS IS NULL
THEN
RET_VALUE := -3;
ELSIF LEVELVAL IS NULL
THEN
RET_VALUE := -4;
ELSIF LOGIN_ID IS NULL
THEN
RET_VALUE := -5;
ELSIF PASSWORDVAL IS NULL
THEN
RET_VALUE := -6;
END IF;

IF (LEVELVAL = 'GRADUATE' OR LEVELVAL = 'UNDERGRADUATE')
THEN
SELECT ROLEID INTO ROLETEMP FROM ROLES_TBL WHERE ROLENAME = 'STUDENT';
ELSE
IF (RET_VALUE = 0)
THEN
RET_VALUE := -7;
END IF;
END IF;

SELECT COUNT(*) INTO COUNTE FROM USERS WHERE EMAILADDRESS = EMAIL_ADDRESS;
IF(COUNTE != 0)
```

```
THEN
IF (RET_VALUE = 0)
THEN
RET_VALUE := -8;
END IF;
END IF;

COUNTE := -1;

SELECT COUNT(*) INTO COUNTE FROM USERS WHERE LOGINID = LOGIN_ID;
IF(COUNTE != 0)
THEN
IF (RET_VALUE = 0)
THEN
RET_VALUE := -9;
END IF;
END IF;

IF (RET_VALUE = 0)
THEN
INSERT INTO USERS VALUES(MAXUSERID,FIRST_NAME, LAST_NAME,
EMAIL_ADDRESS,ROLETEMP,LEVELVAL,LOGIN_ID,PASSWORDVAL);
RET_VALUE := 1;
END IF;
END;
```

(rkvardhi, sbobba3, skataka, spulima)

3) CREATE OR REPLACE PROCEDURE EXERCISECREATION(COURSETOPIC_IP IN COURSETOPIC_LIST,

```
STARTDATE IN DATE,

ENDDATE IN DATE,

NUMBEROFATTEMPTS IN NUMBER,

DIFFICULTYRANGE IN NUMBER,

SCORESELECTIONSCHEME IN VARCHAR2,

NUMQUESTIONS IN NUMBER,

CORRECTANSWERPOINTS IN NUMBER ,

COURSETOKENID IN VARCHAR2,

INCORRECTANSWERPOINTS IN NUMBER,

RET_VALUE OUT NUMBER)

AS

  COU_TOK VARCHAR2(40);

  COU_TOPID NUMBER;

  LEN NUMBER;

  K NUMBER;

  COUNT_VAL NUMBER;

  NOT_STRING VARCHAR2(200);

BEGIN

  RET_VALUE := 0;

  SELECT MAX(EXERCISEID) INTO K FROM EXERCISES;


  INSERT INTO EXERCISES VALUES(K+1,DIFFICULTYRANGE,NUMBEROFATTEMPTS,
CORRECTANSWERPOINTS, INCORRECTANSWERPOINTS, STARTDATE, ENDDATE,
SCORESELECTIONSCHEME,NUMQUESTIONS);
```

(rkvardhi, sbobba3, skataka, spulima)

```
  FOR I IN 1 .. COURSETOPIC_IP.COUNT

  LOOP

    SELECT COURSETOKENID INTO COU_TOK FROM COURSEOFFERINGS WHERE COURSEID IN
(SELECT COURSEID FROM COURSETOPICS WHERE COURSETOPICID = COURSETOPIC_IP(I));

    /*SELECT COURSETOPICID INTO COU_TOPID FROM COURSETOPICS WHERE
COURSETOPICNAME = COURSETOPIC_IP(I); */

    INSERT INTO EXERCISELIST VALUES(K+1, COURSETOPIC_IP(I), COU_TOK);

  END LOOP;

  RET_VALUE := K+1;

  NOT_STRING := CONCAT ('ADDED HOMEWORK : ',K+1);

  SELECT COUNT(*) INTO COUNT_VAL FROM NOTIFICATIONS;

  COUNT_VAL := COUNT_VAL+1;

  INSERT INTO NOTIFICATIONS VALUES(COUNT_VAL,NOT_STRING,COU_TOK);

END;
```

**4)** CREATE OR REPLACE PROCEDURE FETCHPASTQUESTIONSSUBMISSIONS(

```
LOGIN_ID IN VARCHAR2,

EXERCISE_ID IN NUMBER,

ATTEMPT_ID IN NUMBER,

PASTSUB OUT SYS_REFCURSOR)

AS

USER_ID NUMBER;

QUESNUM NUMBER;

OP1 NUMBER;

OP2 NUMBER;

OP3 NUMBER;

OP4 NUMBER;

SELOP NUMBER;
```

(rkvardhi, sbobba3, skataka, spulima)

```
QUESTEXT VARCHAR2(500);
OPT1 VARCHAR2(500);
OPT2 VARCHAR2(500);
OPT3 VARCHAR2(500);
OPT4 VARCHAR2(500);
SELOPT VARCHAR2(500);
HINTVAL VARCHAR2(500);
EXPVAL VARCHAR2(500);
CRCTANS VARCHAR2(500);
SET_ID NUMBER;
TEMPPARAM SYS_REFCURSOR;
TEMPPARAMID NUMBER(5);
TEMPPARAMVALUE VARCHAR2(30);
BEGIN
EXECUTE IMMEDIATE 'TRUNCATE TABLE FETCH_TEMP';
SELECT USERID INTO USER_ID FROM USERS WHERE LOGINID = LOGIN_ID
OPEN PASTSUB FOR
SELECT QUESTIONID,OPTION1,OPTION2,OPTION3,OPTION4,SELECTEDOPTION,SETID
FROM STUDENTATTEMPTQUESTIONS
WHERE EXERCISEID=EXERCISE_ID AND USERID=USER_ID AND ATTEMPTID = ATTEMPT_ID;
LOOP
FETCH PASTSUB INTO QUESNUM,OP1,OP2,OP3,OP4,SELOP,SET_ID;
EXIT WHEN PASTSUB%NOTFOUND;

IF SET_ID = 0
THEN
```

```
SELECT QUESTIONTEXT INTO QUESTEXT FROM QUESTIONBANK WHERE QUESTIONID =
QUESNUM;

ELSE

SELECT QUESTIONTEXT INTO QUESTEXT FROM QUESTIONBANK WHERE QUESTIONID =
QUESNUM;

 OPEN  TEMPPARAM FOR

   SELECT QP.PARAMETERID, QP.PARAMETERVALUE FROM QUESTIONPARAM QP  WHERE
QP.QUESTIONID = QUESNUM AND QP.SETID = SET_ID;

   LOOP

     FETCH TEMPPARAM INTO TEMPPARAMID,TEMPPARAMVALUE;

     EXIT WHEN TEMPPARAM%NOTFOUND;

     QUESTEXT :=
REPLACE(QUESTEXT,CONCAT('PARAM',TO_CHAR(TEMPPARAMID)),TEMPPARAMVALUE);


   END LOOP;

END IF;

SELECT ANSWERTEXT INTO OPT1 FROM ANSWERS WHERE ANSWERID=OP1 AND QUESTIONID =
QUESNUM AND SETID = SET_ID;

SELECT ANSWERTEXT INTO OPT2 FROM ANSWERS WHERE ANSWERID=OP2 AND QUESTIONID =
QUESNUM AND SETID = SET_ID;

SELECT ANSWERTEXT INTO OPT3 FROM ANSWERS WHERE ANSWERID=OP3 AND QUESTIONID =
QUESNUM AND SETID = SET_ID;

SELECT ANSWERTEXT INTO OPT4 FROM ANSWERS WHERE ANSWERID=OP4 AND QUESTIONID =
QUESNUM AND SETID = SET_ID;

SELECT ANSWERTEXT INTO CRCTANS FROM ANSWERS WHERE ISRIGHT= 'Y' AND QUESTIONID =
QUESNUM AND ANSWERID IN (OP1,OP2,OP3,OP4) AND SETID = SET_ID;

SELECT ANSWERTEXT INTO SELOPT FROM ANSWERS WHERE ANSWERID=SELOP AND
QUESTIONID = QUESNUM AND SETID = SET_ID;

SELECT HINT INTO HINTVAL FROM QUESTIONBANK WHERE QUESTIONID = QUESNUM;

SELECT EXPLANATION INTO EXPVAL FROM QUESTIONBANK WHERE QUESTIONID = QUESNUM;
```

(rkvardhi, sbobba3, skataka, spulima)

```
INSERT INTO FETCH_TEMP
VALUES(QUESTEXT,OPT1,OPT2,OPT3,OPT4,SELOPT,HINTVAL,EXPVAL,CRCTANS);

END LOOP;

OPEN PASTSUB FOR

SELECT * FROM FETCH_TEMP;

END;
```

**5)** 
```
CREATE OR REPLACE PROCEDURE FORGOT_PWD(LOGIN_ID IN VARCHAR2,

EMAIL_ID IN VARCHAR2,

RESULTTEXT OUT VARCHAR2,

RESULTVAL OUT NUMBER,

PASSWORD_VAL OUT VARCHAR2) AS

CVAL NUMBER;

LOCALEMAILID VARCHAR2(40);

BEGIN

RESULTVAL :=0;

SELECT COUNT(*) INTO CVAL FROM USERS WHERE LOGINID = LOGIN_ID;

IF CVAL > 0

THEN

SELECT 1 INTO RESULTVAL FROM USERS WHERE LOGINID = LOGIN_ID;

END IF;

IF RESULTVAL = 1

THEN

SELECT EMAILADDRESS INTO LOCALEMAILID FROM USERS WHERE LOGINID = LOGIN_ID;

END IF;


IF LOCALEMAILID = EMAIL_ID

THEN
```

(rkvardhi, sbobba3, skataka, spulima)

```
SELECT 2 INTO RESULTVAL FROM USERS WHERE LOGINID = LOGIN_ID AND EMAILADDRESS =
EMAIL_ID;

SELECT PASSWORDVAL INTO PASSWORD_VAL FROM USERS WHERE LOGINID = LOGIN_ID AND
EMAILADDRESS = EMAIL_ID;

END IF;

IF RESULTVAL = 0

THEN

RESULTTEXT := 'INVALID LOGINID';

ELSE IF RESULTVAL = 1

THEN

RESULTTEXT := 'INVALID EMAIL ID. PLEASE ENTER THE EMAIL ID ';

ELSE

RESULTTEXT := 'A NEW PASSWORD HAS BEEN SENT TO THE EMAIL ID PROVIDED';

END IF;

END IF;

END;
```

6) CREATE OR REPLACE PROCEDURE HOMEWORKREMAINDER AS

```
COUNT_VAL NUMBER;

NOTIFICATIONCOUNT NUMBER;

CURSOR C1 IS SELECT SE.USERID,E.EXERCISEID,EL.COURSETOKENID ,E.ENDDATETIME

FROM EXERCISES E, EXERCISELIST EL,STUDENTENROLLMENT SE

WHERE  E.EXERCISEID = EL.EXERCISEID AND EL.COURSETOKENID = SE.COURSETOKENID

AND E.ENDDATETIME > SYSDATE AND E.STARTDATETIME <SYSDATE;

COU_TOK VARCHAR2(200);

USER_ID NUMBER;

EXERCISE_ID NUMBER;

NOTIFICATION_TEXT VARCHAR(200);
```

(rkvardhi, sbobba3, skataka, spulima)

```
ENDDATE_TIME DATE;
DATEDIFFVAL NUMBER;
BEGIN
  OPEN C1;
  SELECT MAX(NOTIFICATIONID) INTO COUNT_VAL FROM NOTIFICATIONS;
  COUNT_VAL := COUNT_VAL+1;
LOOP
   FETCH C1 INTO USER_ID, EXERCISE_ID,COU_TOK,ENDDATE_TIME;
   EXIT WHEN C1%NOTFOUND;
    SELECT ENDDATE_TIME - TRUNC(SYSDATE) INTO DATEDIFFVAL FROM DUAL;
  IF(DATEDIFFVAL = 1)
  THEN
    INSERT INTO NOTIFICATIONS VALUES(COUNT_VAL,'CURRENT EXERCISE IS DUE BY NEXT DATE.
NO EXTENSIONS IN THE DEADLINE',COU_TOK);
  END IF;
END LOOP;
CLOSE C1;
OPEN C1;
SELECT MAX(NOTIFICATIONID) INTO NOTIFICATIONCOUNT FROM NOTIFICATIONS;
LOOP
  FETCH C1 INTO USER_ID,EXERCISE_ID,COU_TOK,ENDDATE_TIME;
    EXIT WHEN C1%NOTFOUND;


    SELECT ENDDATE_TIME - TRUNC(SYSDATE) INTO DATEDIFFVAL FROM DUAL;


  IF(1=DATEDIFFVAL)
  THEN
```

(rkvardhi, sbobba3, skataka, spulima)

```
   INSERT INTO NOTIFICATIONUSERMAPPING VALUES(USER_ID,NOTIFICATIONCOUNT,1);
  END IF;
END LOOP;


  CLOSE C1;
END;
```

7) CREATE OR REPLACE PROCEDURE RETRIEVEANSWERS(EXERCISEID IN NUMBER,QUESTIONIDS IN QUESTION_LIST ,SETIDS IN SETS_LIST, ANSWERS OUT SYS_REFCURSOR)

```
AS
CURSOR QUESTIONS IS
SELECT DISTINCT EQ.QUESTIONID
FROM EXERCISEQUESTION EQ, QUESTIONBANK QB
WHERE EQ.QUESTIONID = QB.QUESTIONID
AND EQ.EXERCISEID = EXERCISEID
ORDER BY EQ.QUESTIONID;

QUESTION_ID QUESTIONBANK.QUESTIONID%TYPE;

BEGIN
EXECUTE IMMEDIATE 'TRUNCATE TABLE ANSWERS_TEMP';

OPEN QUESTIONS;
LOOP
  FETCH QUESTIONS INTO QUESTION_ID;
  EXIT WHEN QUESTIONS%NOTFOUND;
```

```
INSERT INTO ANSWERS_TEMP(ANSWERID,ANSWERTEXT,EXPLANATION,ISRIGHT,QUESTIONID)
SELECT ANSWERID,ANSWERTEXT,EXPLANATION,ISRIGHT,QUESTIONID FROM (
SELECT ANSWERID,ANSWERTEXT,EXPLANATION,ISRIGHT,QUESTIONID FROM
(SELECT  A.ANSWERID,A.ANSWERTEXT,A.EXPLANATION, A.ISRIGHT, A.QUESTIONID
FROM ANSWERS A
WHERE A.QUESTIONID = QUESTION_ID AND A.ISRIGHT = 'N' AND A.SETID = 0
ORDER BY DBMS_RANDOM.RANDOM)
WHERE ROWNUM < 4
UNION
SELECT ANSWERID,ANSWERTEXT,EXPLANATION,ISRIGHT,QUESTIONID FROM
(SELECT  A.ANSWERID,A.ANSWERTEXT,A.EXPLANATION, A.ISRIGHT, A.QUESTIONID
FROM ANSWERS A
WHERE A.QUESTIONID = QUESTION_ID AND A.ISRIGHT = 'Y' AND A.SETID = 0
ORDER BY DBMS_RANDOM.RANDOM)
WHERE ROWNUM < 2
)
ORDER BY SYS.DBMS_RANDOM.RANDOM;

END LOOP;
CLOSE QUESTIONS;

FOR I IN 1..QUESTIONIDS.COUNT
LOOP
IF(SETIDS(I) > 0)
THEN
INSERT INTO ANSWERS_TEMP(ANSWERID,ANSWERTEXT,EXPLANATION,ISRIGHT,QUESTIONID)
```

(rkvardhi, sbobba3, skataka, spulima)

```
SELECT ANSWERID,ANSWERTEXT,EXPLANATION,ISRIGHT,QUESTIONID FROM (
SELECT ANSWERID,ANSWERTEXT,EXPLANATION,ISRIGHT,QUESTIONID FROM
(SELECT  A.ANSWERID,A.ANSWERTEXT,A.EXPLANATION, A.ISRIGHT, A.QUESTIONID
FROM ANSWERS A
WHERE A.QUESTIONID = QUESTIONIDS(I) AND A.ISRIGHT = 'N' AND A.SETID = SETIDS(I)
ORDER BY DBMS_RANDOM.RANDOM)
WHERE ROWNUM < 4
UNION
SELECT ANSWERID,ANSWERTEXT,EXPLANATION,ISRIGHT,QUESTIONID FROM
(SELECT  A.ANSWERID,A.ANSWERTEXT,A.EXPLANATION, A.ISRIGHT, A.QUESTIONID
FROM ANSWERS A
WHERE A.QUESTIONID = QUESTIONIDS(I) AND A.ISRIGHT = 'Y' AND A.SETID = SETIDS(I)
ORDER BY DBMS_RANDOM.RANDOM)
WHERE ROWNUM < 2
)
ORDER BY SYS.DBMS_RANDOM.RANDOM;


END IF;
END LOOP;
OPEN ANSWERS FOR
SELECT * FROM ANSWERS_TEMP ORDER BY QUESTIONID;
END;
```

8) CREATE OR REPLACE PROCEDURE RETRIEVEEXERCISEQUESTIONS(EXERCISE_ID IN NUMBER, QUESTIONS OUT SYS_REFCURSOR, COUNTREC OUT NUMBER) AS

```
NUMBEROFQUESTIONS INT;
TEMPQUESTIONS SYS_REFCURSOR;
TEMPQUESTIONIDS SYS_REFCURSOR;
```

(rkvardhi, sbobba3, skataka, spulima)

```
TEMPPARAM SYS_REFCURSOR;
TEMPSETID INT;
TEMPQID INT;
TEMPPARAMID INT;
TEMPPARAMVALUE VARCHAR2(50);
TEMPQUESTIONTEXT VARCHAR2(500);
BEGIN

EXECUTE IMMEDIATE 'TRUNCATE TABLE TEMP_QUESTIONSINFO';

INSERT INTO TEMP_QUESTIONSINFO
SELECT DISTINCT EQ.QUESTIONID,QB.QUESTIONTEXT,QB.HINT,QB.EXPLANATION ,0 AS SETID
FROM EXERCISEQUESTION EQ, QUESTIONBANK QB
WHERE EQ.QUESTIONID = QB.QUESTIONID
AND EQ.EXERCISEID = EXERCISE_ID
AND NOT EXISTS
(
SELECT 1 FROM QUESTIONPARAM QP WHERE QP.QUESTIONID = QB.QUESTIONID
);

OPEN TEMPQUESTIONIDS FOR
SELECT DISTINCT QP.QUESTIONID FROM QUESTIONPARAM QP, EXERCISEQUESTION EQ
WHERE EQ.QUESTIONID = QP.QUESTIONID AND EQ.EXERCISEID = EXERCISE_ID;

LOOP
  FETCH TEMPQUESTIONIDS INTO TEMPQID;
```

(rkvardhi, sbobba3, skataka, spulima)

```
  EXIT WHEN TEMPQUESTIONIDS%NOTFOUND;


  OPEN TEMPQUESTIONS FOR

  SELECT SETID FROM

  (SELECT DISTINCT QP.SETID FROM QUESTIONPARAM QP, EXERCISEQUESTION EQ

  WHERE EQ.QUESTIONID = TEMPQID AND EQ.EXERCISEID = EXERCISE_ID

  ORDER BY DBMS_RANDOM.RANDOM) WHERE ROWNUM <=1;


  SELECT QUESTIONTEXT INTO TEMPQUESTIONTEXT FROM QUESTIONBANK WHERE
QUESTIONID = TEMPQID;


  LOOP

    FETCH TEMPQUESTIONS INTO TEMPSETID;

    EXIT WHEN TEMPQUESTIONS%NOTFOUND;


    OPEN  TEMPPARAM FOR

    SELECT QP.PARAMETERID, QP.PARAMETERVALUE FROM QUESTIONPARAM QP  WHERE
QP.QUESTIONID = TEMPQID AND QP.SETID = TEMPSETID;


    LOOP

      FETCH TEMPPARAM INTO TEMPPARAMID,TEMPPARAMVALUE;

      EXIT WHEN TEMPPARAM%NOTFOUND;


      TEMPQUESTIONTEXT :=
REPLACE(TEMPQUESTIONTEXT,CONCAT('PARAM',TO_CHAR(TEMPPARAMID)),TEMPPARAMVAL
UE);

    END LOOP;

    INSERT INTO TEMP_QUESTIONSINFO

(rkvardhi, sbobba3, skataka, spulima)
```

```
    SELECT TEMPQID,TEMPQUESTIONTEXT,QB.HINT,QB.EXPLANATION,TEMPSETID FROM
QUESTIONBANK QB

    WHERE QB.QUESTIONID = TEMPQID;


  END LOOP;

END LOOP;


SELECT COUNT(*) INTO COUNTREC FROM TEMP_QUESTIONSINFO;


OPEN QUESTIONS FOR

SELECT * FROM TEMP_QUESTIONSINFO

ORDER BY QUESTIONID;

END;
```

**Jobs:**

**1)We have created below DBMS_scheduler job which runs every 3 months and executes the procedure password_change which notifies users to change their passwords.**

```
BEGIN

Dbms_Scheduler.create_job(

job_name   => 'PASSWORD_CHANGE',

job_type   => 'STORED_PROCEDURE',

job_action   => 'ADDNOTIFICATION',

start_date   => SYSDATE ,

repeat_interval => 'freq=QUARTERLY;interval=1'                ,

enabled   => TRUE,

comments   => 'Demo for job schedule.');

END;
```

(rkvardhi, sbobba3, skataka, spulima)

**2)DBMS_scheduler job which runs daily and executes the procedure homeworknotifiication which checks homeworks due that day and notifies users enrolled for that course**

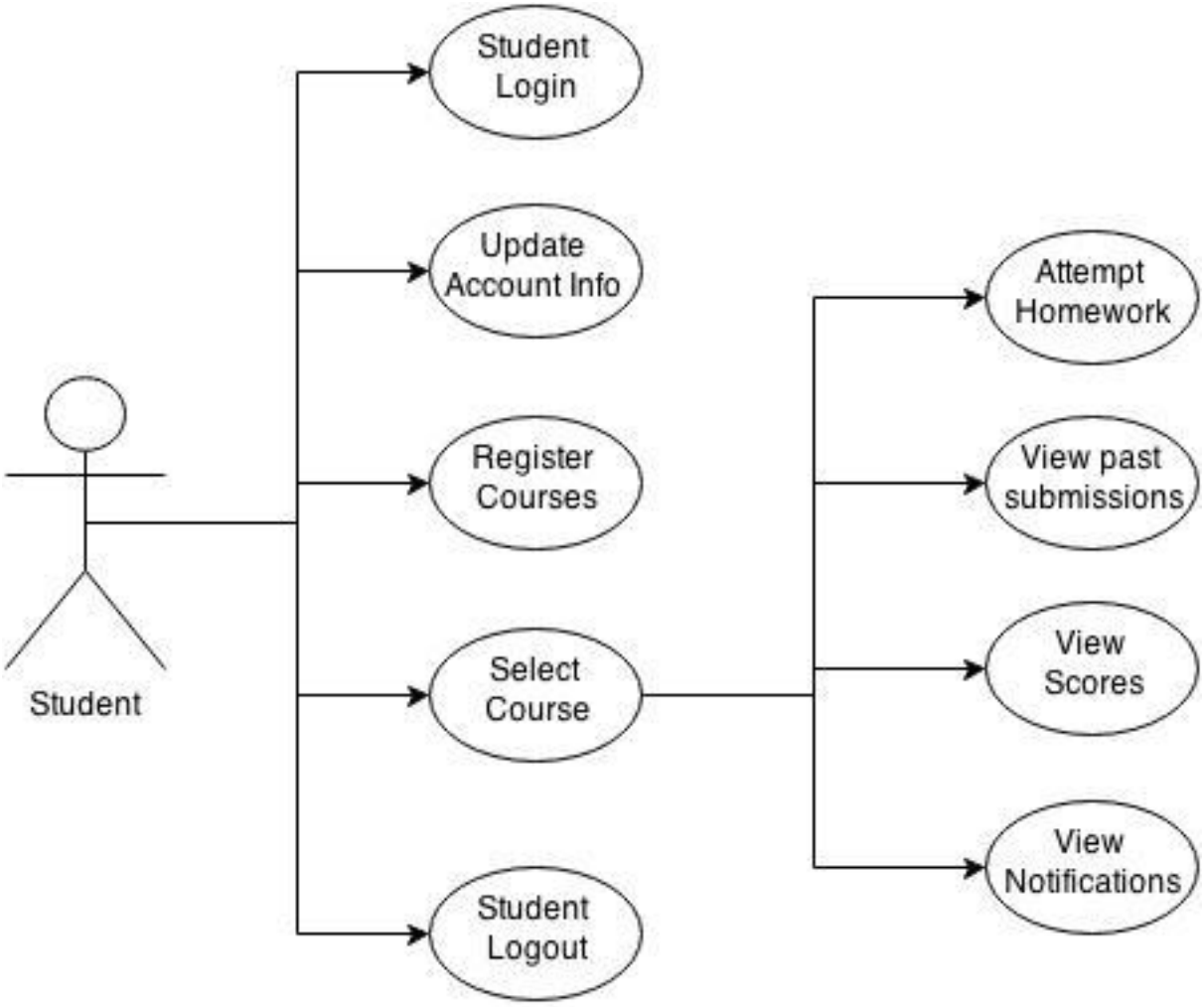
```
BEGIN
Dbms_Scheduler.create_job(
job_name   => 'HOMEWORKNOTIFICATION',
job_type   => 'STORED_PROCEDURE',
job_action   => 'HOMEWORKREMAINDER',
start_date   => SYSDATE ,
repeat_interval => 'freq=DAILY;interval=1'                ,
enabled   => TRUE,
comments   => 'Demo for job schedule.');
END;
```

(rkvardhi, sbobba3, skataka, spulima)

**IX Use Case Scenarios:**

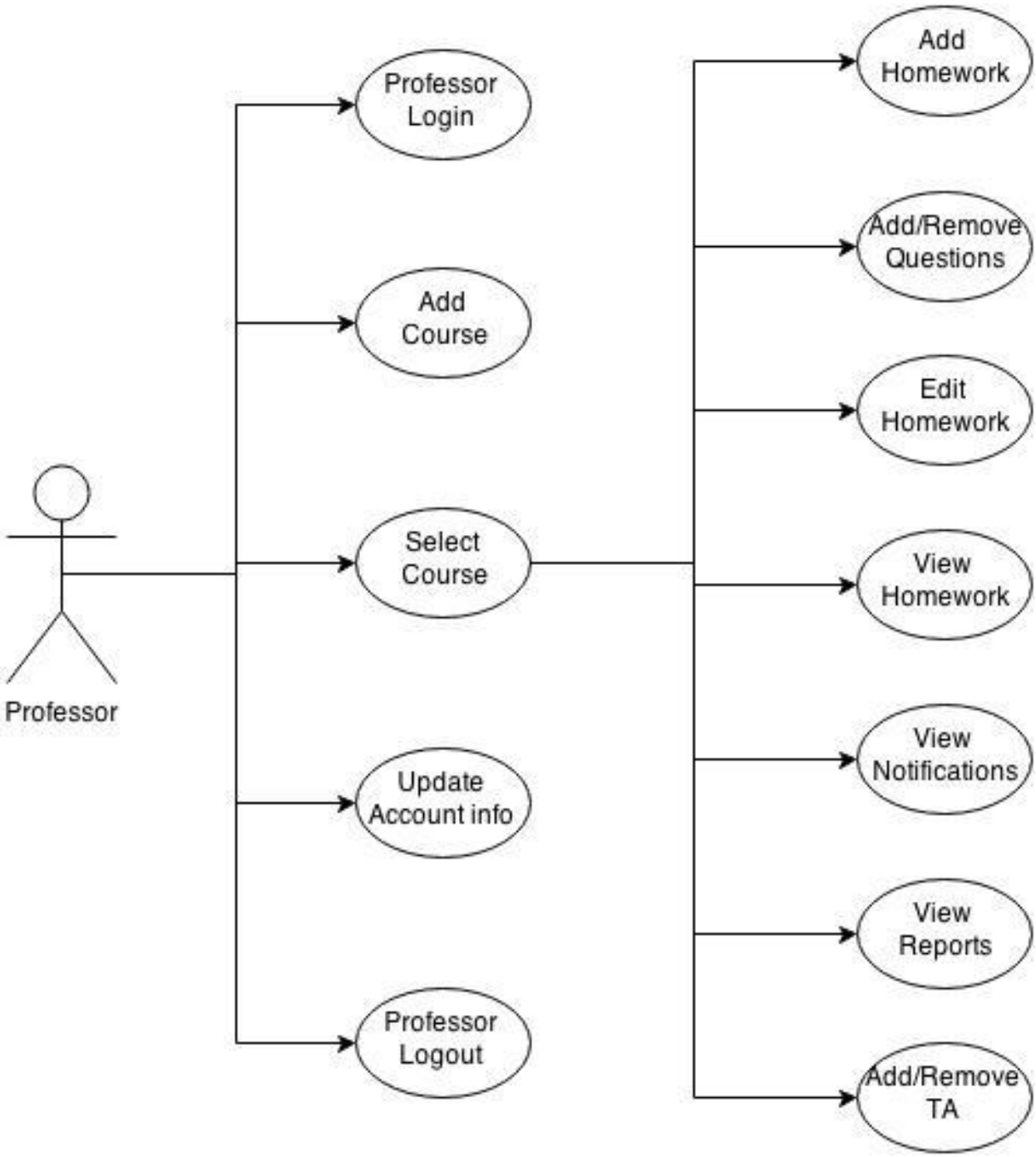
There are three types of users – Student, Professor and TA. The use cases of each user is depicted below in Use Case diagram.

Student:



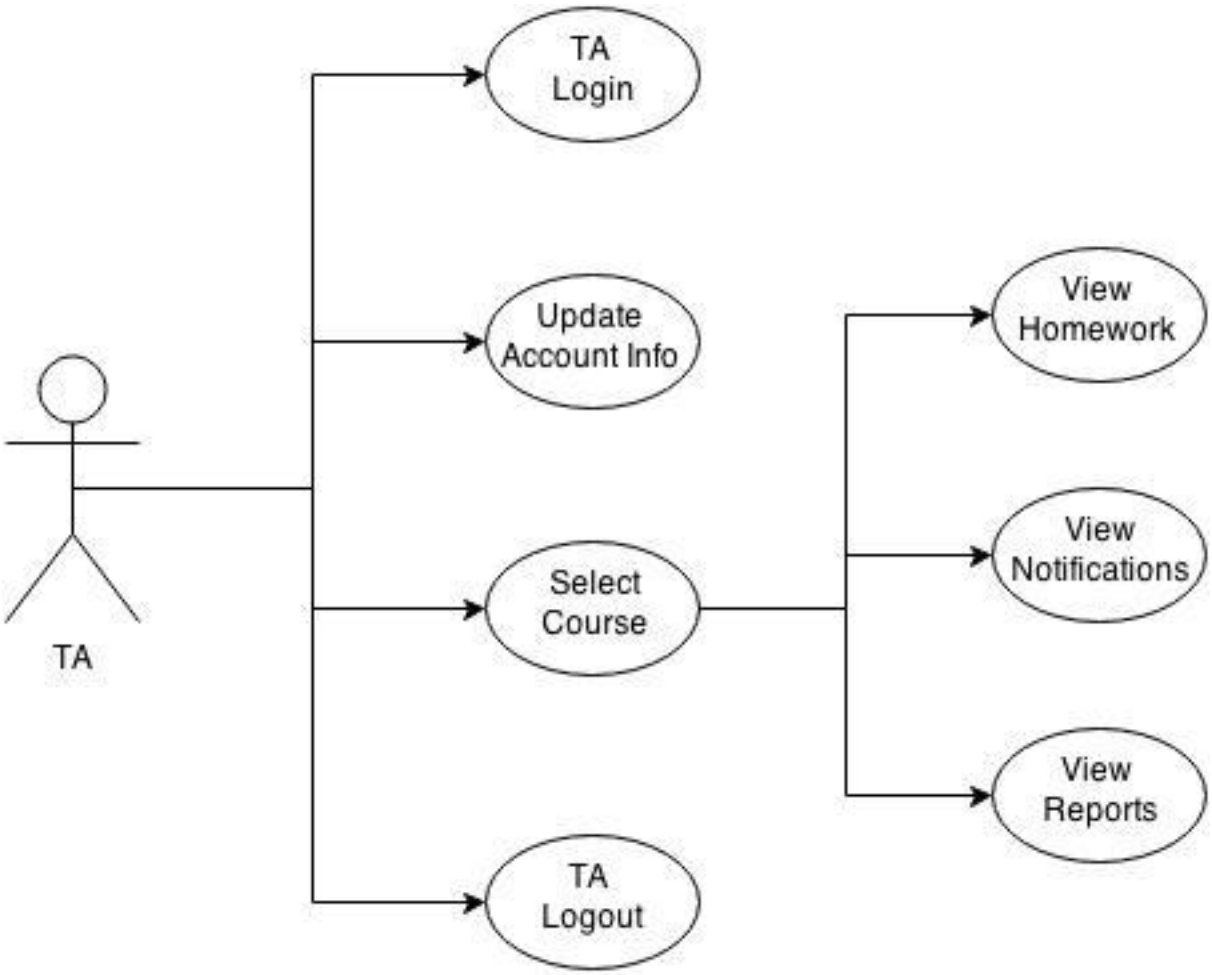

(rkvardhi, sbobba3, skataka, spulima)

Professor:




(rkvardhi, sbobba3, skataka, spulima)

TA:




(rkvardhi, sbobba3, skataka, spulima)