CSC - 540

PROJECT 1 - GRADIANCE : REPORT

Team 4 : jitesh, nkerkar, sktomer, sgargav

TABLE OF CONTENTS

---

1. Assumptions

- ❏ Question bank is already present in database.
- ❏ Courses (not course token) are already listed in database.
- ❏ Expired courses are not displayed for any user.
- ❏ When a professor adds a TA it is expected that the professor is aware of the courses that the TA has enrolled in and the clashing topics.
- ❏ All alerts will be shown in the notification tab on the home page for users. After viewing a notification, user may delete the notification.

2. Problem Statement

This project is an online assignment management (and course assessment) system. A professor can login and create and edit homework assignments for students. A student can attempt these assignments and can also view past submissions for multiple attempts. A student is not allowed to attempt an assignment after the due date has passed and the assignment gets listed in past submissions. A TA (as well as Professor) can view class roll and all attempts for all subjects for which he/she is a TA(or Professor). A TA can also view any homework as soon as it is posted by the professor. The system also allows students and professors to add/remove course tokens. The Professor may add/remove TAs for his course. A TA, however, cannot add a course as a student for which he is TA (however since a TA is also a student, he may add other courses).

3. Entities and Relationships

★ **ENTITIES**

❏ User (<u>userid</u>, password, email, firstname, lastname, type_id)
  Keeps records of all registered users on the gradiance website with unique userid. User can be admin, student, professor or TA.

❏ Usertypes (<u>type_id</u>, type_description)
  Specifies the types of user in the database, namely - admin, student, professor or TA. Note that a TA who is also a student for another course is still registered as type TA.

❏ Student (<u>userid</u>, study_level)
  Is formed via ISA hierarchy from user. Keeps information about student(userid as foreign key) and their study-level (grad or undergrad)

❏ Professor (<u>userid</u>, <u>token</u>)
  Is formed via ISA hierarchy from user. Keeps information about professor(userid as foreign key) and the courses they have taught (foreign key token from course)

❏ Teaching_Assistant (<u>userid</u>, <u>token</u>)
  Is formed via ISA hierarchy from student(grad). Contains information about the TA and the courses that are assigned TAship.

❏ Notifications (<u>userid</u>, <u>messageid</u>, message)
  Contains all the notifications intended for a user including a unique message id and message text. Can be deleted by user.

❏ Course (<u>token</u>, courseid, semester, coursestartdate, courseenddate, maxenrolled, numenrolled, courselevel)
  Contains all the course tokens with their courseid as foreign key, provides all the information related to the course token as listed in it's attribute list.

❏ CourseList (<u>courseid</u>, coursename)

Lists the courseids and names of the courses available for the professor to choose from and create course tokens. Cannot be modified by professor, is expected to be present in database initially.

❏ Topics (topicid, topicname)
  Lists the topicids and topic names of all the topics stored in database.

❏ Textbook (isbnnum, textbookname, textbookauthor)
  Lists the ISBN numbers, names and authors of all textbooks referenced by courses.

❏ Chapters (isbnnum, chapterid, chaptername)
  Lists the chapter ids and names and links them to their textbooks using ISBN num as foreign key.

❏ Sections (isbnnum, chapterid, sectionid, sectionname)
  Links the sections of a chapter of a textbook to the chapterid and textbook ISBN number both referenced as foreign keys.

❏ Subsections (isbnnum, chapterid, sectionid, subsectionid, subsectionname, subsectiondescr)
  Links the sub-sections of a section of a chapter of a textbook to the sectionid, chapterid and textbook ISBN number, all referenced as foreign keys.

❏ Questions (questionid, questiontext, topicid, diff_level_id, hint, parameterid, explanation)
  Lists down all the information related to a question, including parameterid which identifies parameterized questions.

❏ Difficulty_level (q_diff_level_id, q_diff_level)
  Stores the 6 levels of difficulty referenced by the questions in the question bank.

❏ Bonus (eid, bcount, bpoints)
  Implemented as a cool feature. Contains the exerciseid referenced as foreign key and specifies the bonus points and the number of students that can receive them.

❏ Answers (answerid, questionid, parameterid, answertext, explanation, answertype)
  Contains the answers for the questions in the question bank, references questionid as foreign key. Also contains parameterid which is needed for parameterized questions.

❏ Exercises (topicid, exerciseid, token, userid, estartdate, eenddate, max_retry_count, exer_diff_level, random_seed, points_per_question, negative_points, scoretypeid, number_qt)
  Lists down the exercises created by the professors, uniquely identified by exerciseid using the course token as foreign key. Contains all information related to exercise.

❏ Submission (userid, exerciseid, attemptnum, substarttime, subendtime, answerlist, score, status, qlist)
  Records all the homeworks submitted by the students, attempt number, score and related information.

❏ Scoretype (scoretypeid, scoringtype)
  Lists the various types of homework score calculation types available.

❏ Topq (qid, qcount)
  Implemented as a bonus feature. Lists the topmost difficult questions and their wrong count (number of times they were answered incorrectly by students).

❏ Feedback (fid, feedback_text)

Records the anonymous feedback provided to the admin by users which are displayed to the admin as notifications.

★ **RELATIONSHIPS**

- ❑ **Exercises_Questions (exerciseid, questionid)**
  All questions belonging to all exercises are recorded here.
- ❑ **Textbook_Course (isbnnum, token)**
  Lists textbook(s) for every course.
- ❑ **Topic_section (topicid, isbnnum, chapterid, sectionid)**
  Relates topic ids with sections of a chapter of a textbook.
- ❑ **Enrollment (userid, token)**
  Records students enrolled for courses.
- ❑ **UserHASusertype (type_id, user_id)**
  Specifies whether a user is an admin or student or professor or ta.
- ❑ **TA_assigned_course (userid, token)**
  Keeps track of students who are TAs.
- ❑ **Prof_assigned_course (userid, token)**
  Keeps track of which professors instruct which courses.
- ❑ **User_GETS_notification (userid, messageid)**
  All notifications for all users are recorded.
- ❑ **Course_HAS_courselist (token,courseid)**
  Courses defined for the semester from the master list of courses.
- ❑ **Exercise_HAS_Bonus (eid)**
  Configures optional bonus points for exercises.
- ❑ **Exercise_HAS_Scoretype (exerciseid,scoretypeid)**
  Method for score selection of each exercise.
- ❑ **Questions_HAS_difficultylevel (question_id,q_diff_level_id)**
  Records difficulty level for questions.
- ❑ **Questions_HAS_topq (question_id,qid)**
  Records the five top questions answered most incorrectly.
- ❑ **Questions_related_to_topics (topic_id,question_id**)
  Each question belongs to exactly one topic.
- ❑ **Textbook_HAS_chapters (isbnnum,chapterid)**
  Lists chapters from a textbook.
- ❑ **Chapters_HAS_sections_subsections (chapter_id, section_id,subsection_id)**
  Relates chapters and sections and subsections.
- ❑ **Answers_HAS_Parmeters (answer_id,parameter_id)**
  Specifies which answer belongs to which parameter id.
- ❑ **Topic_Courselist (topic_id,course_id)**
  List of topics that a course has. One course may span across multiple topics.

4. Users

- ❑ Admin (moderator)
  Can view feedback provided for gradiance (cool feature), admin has access to backend.
- ❑ Student
  - ❑ UnderGrad Student
    Can add undergrad courses, can attempt exercises of courses, can view submitted homeworks and scores (can provide feedback)
  - ❑ Graduate Student
    Can add grad courses, can attempt exercises of courses, can view submitted homeworks and scores (can provide feedback)
  - ❑ Grad Student and TA
    Can add grad courses for which topics do not clash which course for which he/she is appointed TA, can attempt exercises of courses enrolled in, can view submitted homeworks and scores (can provide feedback), can view all submissions and scores of course for which he/she is TA
- ❑ Professor
  Can choose course from courselist and create a semester course, can create and edit homeworks, can view all scores and submissions of students, can appoint TAs

5. E-R diagram

Please see ERDiagram.jpg [attached in zip folder]

6. Relational Schemas

```
USE `gradiance`;

CREATE TABLE `answers` (
  `answerid` int(11) NOT NULL DEFAULT '0',
  `questionid` int(11) NOT NULL,
  `parameterid` int(11) NOT NULL,
  `answertext` varchar(100) NOT NULL,
  `explanation` varchar(100) DEFAULT NULL,
  `answertype` varchar(5) NOT NULL,
  PRIMARY KEY (`answerid`),
  KEY `FKanswersquestions` (`questionid`,`parameterid`),
  CONSTRAINT `FKanswersquestions` FOREIGN KEY (`questionid`,
   `parameterid`) REFERENCES `questions` (`questionid`,
   `parameterid`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `bonus` (
  `eid` int(11) NOT NULL,
  `bcount` int(11) NOT NULL DEFAULT '5',
  `bpoints` int(11) NOT NULL DEFAULT '5',
  PRIMARY KEY (`eid`),
  CONSTRAINT `FKbonusexercises ` FOREIGN KEY (`eid`) REFERENCES
   `exercises` (`exerciseid`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `chapter` (
  `isbnnum` varchar(20) NOT NULL DEFAULT '',
  `chapterid` int(11) NOT NULL DEFAULT '0',
  `chaptername` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`isbnnum`,`chapterid`),
  CONSTRAINT `FKchaptertextbook` FOREIGN KEY (`isbnnum`)
   REFERENCES `textbook` (`isbnnum`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `course` (
  `token` varchar(20) NOT NULL DEFAULT '',
  `courseid` varchar(20) DEFAULT NULL,
  `semester` varchar(20) DEFAULT NULL,
  `coursestartdate` date NOT NULL,
  `courseenddate` date NOT NULL,
  `maxenrolled` int(11) NOT NULL,
  `numenrolled` int(11) NOT NULL check (numenrolled <=
   maxenrolled),
  `courselevel` varchar(20) NOT NULL,
  PRIMARY KEY (`token`),
  KEY `FKcoursecourselist` (`courseid`),
  CONSTRAINT `FKcoursecourselist` FOREIGN KEY (`courseid`)
   REFERENCES `courselist` (`courseid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `courselist` (
  `courseid` varchar(20) NOT NULL DEFAULT '',
  `coursename` varchar(30) NOT NULL,
  PRIMARY KEY (`courseid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `difficultylevel` (
  `q_diff_level_id` int(11) NOT NULL DEFAULT '0',
  `q_diff_level` int(11) NOT NULL,
  PRIMARY KEY (`q_diff_level_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `enrollment` (
  `userid` varchar(20) NOT NULL DEFAULT '',
  `token` varchar(20) NOT NULL DEFAULT '',
  PRIMARY KEY (`userid`,`token`),
  KEY `FKenrollmentcourselist` (`token`),
  CONSTRAINT `FKenrollmentcourselist` FOREIGN KEY (`token`)
   REFERENCES `course` (`token`) ON DELETE CASCADE,
  CONSTRAINT `FKenrollmentuser` FOREIGN KEY (`userid`)
   REFERENCES `user` (`userid`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `questions` (
  `questionid` int(11) NOT NULL DEFAULT '0',
  `questiontext` varchar(100) DEFAULT NULL,
  `topicid` int(11) NOT NULL,
  `diff_level_id` int(11) NOT NULL,
  `hint` varchar(100) DEFAULT NULL,
  `explanation` varchar(100) DEFAULT NULL,
  `parameterid` int(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (`questionid`,`parameterid`),
  KEY `FKquestionstopics` (`topicid`),
  KEY `FKquestionsdifflevel` (`diff_level_id`),
  CONSTRAINT `FKquestionsdifflevel` FOREIGN KEY
   (`diff_level_id`) REFERENCES `difficultylevel`
   (`q_diff_level_id`) ON DELETE CASCADE,
  CONSTRAINT `FKquestionstopics` FOREIGN KEY (`topicid`)
   REFERENCES `topics` (`topicid`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `exercises` (
  `topicid` int(11) DEFAULT NULL,
  `exerciseid` int(11) NOT NULL DEFAULT '0',
  `token` varchar(20) NOT NULL,
  `userid` varchar(20) NOT NULL,
  `estartdate` date NOT NULL,
  `eenddate` date NOT NULL,
  `max_retry_count` int(11) NOT NULL check (max_retry_count>0),
  `exer_diff_level` varchar(11) NOT NULL check (exer_diff_level
   >= (select dl.q_diff_level from difficultylevel dl, questions
   q where q.diff_level_id= dl.diff_level_id)),
  `random_seed` int(11) NOT NULL,
  `points_per_question` int(11) NOT NULL,
  `negative_points` int(11) NOT NULL,
  `scoretypeid` int(11) NOT NULL,
  `number_qt` int(11) NOT NULL DEFAULT '1',
  PRIMARY KEY (`exerciseid`),
  KEY `FKexercisesprofessor` (`userid`,`token`),
  KEY `FKexercisescoretype` (`scoretypeid`),
  KEY `FKexercisestopic` (`topicid`),
  CONSTRAINT `FKexercisescoretype` FOREIGN KEY (`scoretypeid`)
   REFERENCES `scoretype` (`scoretypeid`) ON DELETE CASCADE,
  CONSTRAINT `FKexercisesprofessor` FOREIGN KEY (`userid`,
   `token`) REFERENCES `professor` (`userid`, `token`) ON DELETE
   CASCADE,
  CONSTRAINT `FKexercisestopic` FOREIGN KEY (`topicid`)
   REFERENCES `topics` (`topicid`) ON DELETE SET NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `exercises_questions` (
  `exerciseid` int(11) NOT NULL DEFAULT '0',
  `questionid` int(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (`exerciseid`,`questionid`),
  KEY `FKexerques_questions` (`questionid`),
  CONSTRAINT `FKexerques_exercises` FOREIGN KEY (`exerciseid`)
   REFERENCES `exercises` (`exerciseid`) ON DELETE CASCADE,
  CONSTRAINT `FKexerques_questions` FOREIGN KEY (`questionid`)
   REFERENCES `questions` (`questionid`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `feedback` (
  `fid` int(11) NOT NULL DEFAULT '0',
  `feedback_text` varchar(300) DEFAULT NULL,
  PRIMARY KEY (`fid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `notification` (
  `userid` varchar(20) NOT NULL DEFAULT '',
  `messageid` int(11) NOT NULL DEFAULT '0',
  `message` varchar(300) DEFAULT NULL,
  PRIMARY KEY (`userid`,`messageid`),
  CONSTRAINT `FKnotificationuser` FOREIGN KEY (`userid`)
   REFERENCES `user` (`userid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `professor` (
  `userid` varchar(20) NOT NULL DEFAULT '',
  `token` varchar(20) NOT NULL DEFAULT 'default',
  PRIMARY KEY (`userid`,`token`),
  KEY `FKprofessorcourse` (`token`),
  CONSTRAINT `FKprofessorcourse` FOREIGN KEY (`token`)
   REFERENCES `course` (`token`) ON DELETE CASCADE,
  CONSTRAINT `FKprofessoruser` FOREIGN KEY (`userid`) REFERENCES
   `user` (`userid`) ON DELETE CASCADE
```

```
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `scoretype` (
  `scoretypeid` int(11) NOT NULL DEFAULT '0',
  `scoringtype` varchar(20) NOT NULL,
  PRIMARY KEY (`scoretypeid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `section` (
  `isbnnum` varchar(20) NOT NULL DEFAULT '',
  `chapterid` int(11) NOT NULL DEFAULT '0',
  `sectionid` int(11) NOT NULL DEFAULT '0',
  `sectionname` varchar(100) NOT NULL,
  PRIMARY KEY (`isbnnum`,`chapterid`,`sectionid`),
  CONSTRAINT `FKsectionchapter` FOREIGN KEY (`isbnnum`,
   `chapterid`) REFERENCES `chapter` (`isbnnum`, `chapterid`) ON
   DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `student` (
  `userid` varchar(20) NOT NULL DEFAULT '',
  `studylevel` varchar(20) NOT NULL,
  PRIMARY KEY (`userid`),
  CONSTRAINT `FKstudentuser` FOREIGN KEY (`userid`) REFERENCES
   `user` (`userid`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `submission` (
  `userid` varchar(20) NOT NULL DEFAULT '',
  `exerciseid` int(11) NOT NULL DEFAULT '0',
  `attemptnum` int(11) NOT NULL DEFAULT '0',
  `substarttime` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `subendtime` timestamp NOT NULL DEFAULT '0000-00-00 00:00:00',
  `answerlist` varchar(500) DEFAULT NULL,
  `score` int(11) NOT NULL,
  `status` varchar(10) NOT NULL,
  `qlist` varchar(500) DEFAULT NULL,
  PRIMARY KEY (`userid`,`exerciseid`,`attemptnum`),
  KEY `FKsubmissionexercises` (`exerciseid`),
  CONSTRAINT `FKsubmissionexercises` FOREIGN KEY (`exerciseid`)
   REFERENCES `exercises` (`exerciseid`) ON DELETE CASCADE,
  CONSTRAINT `FKsubmissionuser` FOREIGN KEY (`userid`)
   REFERENCES `user` (`userid`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `subsection` (
  `isbnnum` varchar(20) NOT NULL DEFAULT '',
  `chapterid` int(11) NOT NULL DEFAULT '0',
  `sectionid` int(11) NOT NULL DEFAULT '0',
  `subsectionid` int(11) NOT NULL DEFAULT '0',
  `subsectionname` varchar(30) DEFAULT NULL,
  `subsectiondescr` varchar(100) NOT NULL,
  PRIMARY KEY
   (`isbnnum`,`chapterid`,`sectionid`,`subsectionid`),
  CONSTRAINT `FKsubsectionsection` FOREIGN KEY (`isbnnum`,
   `chapterid`, `sectionid`) REFERENCES `section` (`isbnnum`,
   `chapterid`, `sectionid`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `teaching_assistant` (
  `userid` varchar(20) NOT NULL DEFAULT '',
  `token` varchar(20) NOT NULL DEFAULT '',
  PRIMARY KEY (`userid`,`token`),
  KEY `FKtacourse` (`token`),
  CONSTRAINT `FKtacourse` FOREIGN KEY (`token`) REFERENCES
   `course` (`token`) ON DELETE CASCADE,
  CONSTRAINT `FKtauserid` FOREIGN KEY (`userid`) REFERENCES
   `user` (`userid`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `textbook` (
  `isbnnum` varchar(20) NOT NULL DEFAULT '',
  `textbookname` varchar(50) NOT NULL,
  `textbookauthor` varchar(50) NOT NULL,
  PRIMARY KEY (`isbnnum`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `textbook_course` (
  `isbnnum` varchar(20) NOT NULL DEFAULT '',
  `token` varchar(20) NOT NULL DEFAULT '',
  PRIMARY KEY (`isbnnum`,`token`),
  KEY `FKtextbookcourse_courselist` (`token`),
  CONSTRAINT `FKtextbookcourse_courselist` FOREIGN KEY (`token`)
   REFERENCES `course` (`token`) ON DELETE CASCADE,
  CONSTRAINT `FKtextbookcourse_textbook` FOREIGN KEY (`isbnnum`)
   REFERENCES `textbook` (`isbnnum`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `topic_courselist` (
  `topicid` int(11) NOT NULL DEFAULT '0',
  `courseid` varchar(10) NOT NULL DEFAULT '',
  PRIMARY KEY (`topicid`,`courseid`),
  KEY `FKtopiccoursecourselist` (`courseid`),
  CONSTRAINT `FKtopiccoursecourselist` FOREIGN KEY (`courseid`)
   REFERENCES `courselist` (`courseid`) ON DELETE CASCADE,
  CONSTRAINT `FKtopiccoursetopics` FOREIGN KEY (`topicid`)
   REFERENCES `topics` (`topicid`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `topic_section` (
  `topicid` int(11) NOT NULL DEFAULT '0',
  `isbnnum` varchar(20) NOT NULL DEFAULT '',
  `chapterid` int(11) NOT NULL DEFAULT '0',
  `sectionid` int(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (`topicid`,`isbnnum`,`chapterid`,`sectionid`),
  KEY `FKtopicsectionsubsection`
   (`isbnnum`,`chapterid`,`sectionid`),
  CONSTRAINT `FKtopicsectionsubsection` FOREIGN KEY (`isbnnum`,
   `chapterid`, `sectionid`) REFERENCES `section` (`isbnnum`,
   `chapterid`, `sectionid`) ON DELETE CASCADE,
  CONSTRAINT `FKtopicsectiontopics` FOREIGN KEY (`topicid`)
   REFERENCES `topics` (`topicid`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `topics` (
  `topicid` int(11) NOT NULL DEFAULT '0',
  `topicname` varchar(50) NOT NULL,
  PRIMARY KEY (`topicid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `topq` (
  `qid` int(11) NOT NULL,
  `qcount` int(11) NOT NULL,
  PRIMARY KEY (`qid`),
  CONSTRAINT `FKtopqquestions` FOREIGN KEY (`qid`) REFERENCES
   `questions` (`questionid`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `user` (
  `userid` varchar(20) NOT NULL DEFAULT '',
  `password` varchar(20) NOT NULL,
  `email` varchar(30) NOT NULL,
  `firstname` varchar(20) NOT NULL,
  `lastname` varchar(20) NOT NULL,
  `type_id` int(11) NOT NULL,
  PRIMARY KEY (`userid`),
  KEY `FKusertype` (`type_id`),
  CONSTRAINT `FKusertype` FOREIGN KEY (`type_id`) REFERENCES
   `usertype` (`type_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `usertype` (
  `type_id` int(11) NOT NULL DEFAULT '0',
  `type_description` varchar(20) DEFAULT NULL,
  PRIMARY KEY (`type_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

DROP TRIGGER IF EXISTS score_trigger
delimiter $$
create trigger score_trigger before insert  on submission
for each row begin
if scoretypeid = 1
then
      set new.score = (select s1.score from submission s1, s2
   where s1.attemptnum = max(s2.attemptnum) and
   s1.userid=s2.userid and s1.exercise=s2.exercise);
end if;

if scoretypeid = 2
then
      set new.score = (select avg(s1.score) from submission s1,
   s2 where s1.userid=s2.userid and s1.exercise=s2.exercise);
end if;

if scoretypeid = 3
then
      set new.score = (select max(s1.score) from submission s1,
   s2 where s1.userid=s2.userid and s1.exercise=s2.exercise);
end if;

if scoretypeid = 4
then
      set new.score = (select s1.score from submission s1, s2
   where s1.attemptnum = 1 and s1.userid=s2.userid and
   s1.exercise=s2.exercise);
end if;
end $$


DROP TRIGGER IF EXISTS ta_check
delimiter $$
create trigger ta_check before insert  on teaching_assistant
for each row begin
if new.userid = (select userid from student where
   studylevel='undergrad')
then
      delete from teaching_assistant where userid=new.userid;
end if;
end $$

DROP TRIGGER IF EXISTS exercise_max_retry
```

```
delimiter $$
create trigger exercise_max_retry before insert  on exercises
for each row begin

      update exercises set max_retry_count = 1 where
   scoretypeid =4;

end $$

DROP TRIGGER IF EXISTS only_one_prof_percourse
delimiter $$
create trigger only_one_prof_percourse before insert  on
   professor
for each row begin
if (select userid from course c, professor p where
   c.token=p.token)= 1
then
      update professor set userid = 'NULL' where
   course.token=professor.token;
end if;
end $$
```

7. Functional Dependencies and Normal Forms

- user (userid, firstname, lastname, email, password, type_id)
  - FD : userid → firstname, lastname, email, password, type_id
  - Primary Key : userid
  - Normal Form : BCNF
- usertype (type_id, type_description)
  - FD: type_id → type_description
  - Primary Key: type_id
  - Normal Form: BCNF
- student (userid, studylevel)
  - FD: userid → studylevel
  - Primary Key: userid
  - Normal Form: BCNF
- teaching_assistant (userid, token)
  - FD: -
  - Primary Key: userid,token
  - Normal Form: BCNF
- student (userid, studylevel)
  - FD: userid → studylevel

    - Primary Key: userid, studyevel
    - Normal Form: BCNF
- professor (userid, token)
    - FD: -
    - Primary Key: userid,token
    - Normal Form: BCNF
- courselist (courseid, coursename)
    - FD: courseid ⟶ coursename
    - Primary Key: courseid
    - Normal Form: BCNF
- course (token, courseid, semester, coursestartdate, courseenddate, maxenrolled, numenrolled, courselevel)
    - FD: token ⟶ courseid, semester, coursestartdate, courseenddate, maxenrolled, numenrolled, courselevel
    - Primary Key: token
    - Normal Form: BCNF
- textbook (isbnnum, textbookname, textbookauthor)
    - FD: isbnnum ⟶ textbookname, textbookauthor
    - Primary Key: isbnnum
    - Normal Form: BCNF
- textbook_course (isbnnum, token)
    - FD: -
    - Primary Key: isbnnum,token
    - Normal Form: BCNF
- chapter (isbnnum, chapterid, chaptername)
    - FD: isbnnum ⟶ chapterid, chaptername
    - Primary Key: isbnnum
    - Normal Form: BCNF
- section (isbnnum, chapterid, sectionid, sectionname)
    - FD: isbnnum, chapterid ⟶ sectionid, sectionname
    - Primary Key: isbnnum, chapterid
    - Normal Form: BCNF
- subsection (isbnnum, chapterid, sectionid, subsectionid, subsectionname, subsectiondescr)
    - FD: isbnnum, chapterid, sectionid ⟶ subsectionid, subsectionname, subsectiondescr
    - Primary Key: isbnnum, chapterid, sectionid
    - Normal Form: BCNF
- topic (topicid, topicname)
    - FD: topicid ⟶ topicname
    - Primary Key: topicid
    - Normal Form: BCNF
- topic_courselist (topicid, courseid)

- FD: -
- Primary Key: topicid, courseid
- Normal Form: BCNF

- topic_section (topicid, isbnnum, chapterid, sectionid)
  - FD: -
  - Primary Key: topicid, isbnnum, chapterid, sectionid
  - Normal Form: BCNF
- enrollment (userid, token)
  - FD: -
  - Primary Key: userid, token
  - Normal Form: BCNF
- questions (questionid, questiontext, topicid, diff_level_id, hint, explanation, parameterid)
  - FD: questionid, parameterid → questiontext, topicid, diff_level_id, hint, explanation
  - Primary Key: questionid, parameterid
  - Normal Form: BCNF
- answers (answerid, questionid, parameterid, answertext, explanation, answertype)
  - FD: answerid → questionid, parameterid, answertext, explanation, answertype
  - Primary Key: answerid
  - Normal Form: BCNF
- difficultylevel (q_diff_level_id, q_diff_level)
  - FD: q_diff_level_id → q_diff_level
  - Primary Key: q_diff_level_id
  - Normal Form: BCNF
- exercises (topicid, exerciseid, token, userid, estartdate, eenddate, max_retry_count, exer_diff_level, random_seed, points_per_question, nexgative_points, scoretypeid, number_qt)
  - FD: exerciseid → topicid, token, userid, estartdate, eenddate, max_retry_count, exer_diff_level, random_seed, points_per_question, nexgative_points, scoretypeid, number_qt
  - Primary Key: exerciseid
  - Normal Form: BCNF
- exercise_questions (exerciseid, questionid)
  - FD: -
  - Primary Key: exerciseid, questionid
  - Normal Form: BCNF
- scoretype (scoretypeid, scoringtype)
  - FD: scoretypeid → scoringtype
  - Primary Key: scoretypeid
  - Normal Form: BCNF
- bonus (eid, bcount, bpoints)
  - FD: eid → bcount, bpoints

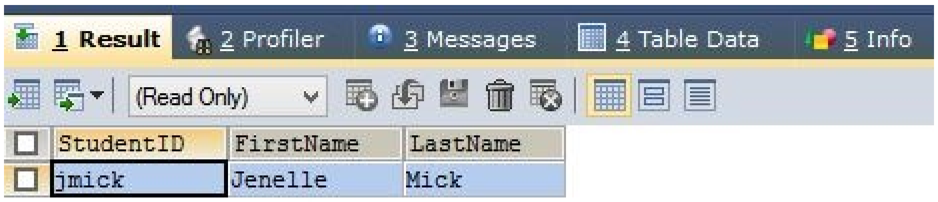
    - Primary Key: eid
    - Normal Form: BCNF
- feedback (fid, feedback_text)
    - FD: fid → feedback_text
    - Primary Key: fid
    - Normal Form: BCNF
- notification (userid, messageid, message)
    - FD: userid, messageid → message
    - Primary Key: userid, messageid
    - Normal Form: BCNF
- topq (qid, qount)
    - FD: qid → qcount
    - Primary Key: qid
    - Normal Form: BCNF

8. SQL queries

❑ **Find students who did not take homework 1 ?**

- ❑ *Assumption* : HomeworkID is 54002 and CourseToken is CSC540FALL14

```
SELECT userid AS StudentID, firstname AS FirstName, lastname AS LastName
FROM user
WHERE userid IN (
SELECT userid
FROM enrollment E
WHERE E.token = "CSC540FALL14" AND E.userid NOT IN (
SELECT userid
FROM submission
WHERE exerciseid = 54002
GROUP BY userid ) ) ;
```

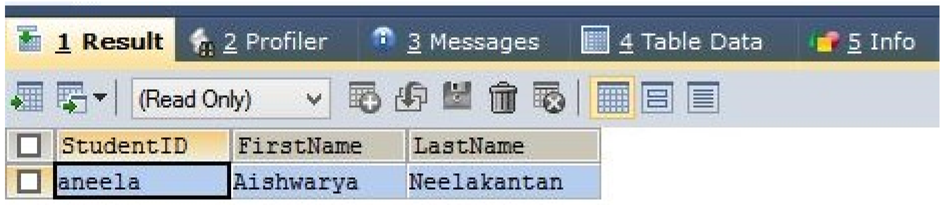


```
StudentID  FirstName  LastName
jmick      Jenelle    Mick
```

❑ **Find students who scored the maximum score on the first attempt for homework 1 ?**

- ❑ *Assumption* : HomeworkID is 54002

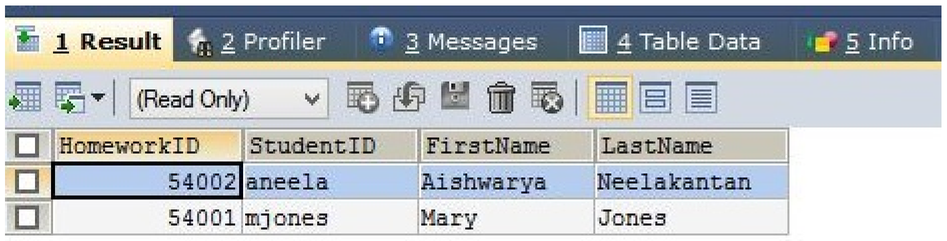
```
SELECT userid AS StudentID, firstname AS FirstName, lastname AS LastName
FROM user
WHERE userid IN (
SELECT userid
FROM submission S
WHERE S.exerciseid = 54002 AND S.attemptnum = 1 AND S.score IN (
SELECT MAX(score) AS score
FROM submission S2
WHERE S2.exerciseid = 54002 AND S2.attemptnum = 1 ) ) ;
```



StudentID	FirstName	LastName
aneela	Aishwarya	Neelakantan

- **Find students who scored the maximum score on the first attempt for each homework ?**
  - *Assumption* : CourseToken is CSC540FALL14

```
SELECT E.exerciseid AS HomeworkID, S1.userid AS StudentID, U.firstname AS FirstName,
U.lastname AS LastName
FROM submission S1, exercises E, user U
WHERE U.userid = S1.userid AND E.token = "CSC540FALL14" AND E.exerciseid =
S1.exerciseid AND S1.attemptnum = 1 AND S1.score >= ALL (
SELECT S.score
FROM submission S
WHERE S.attemptnum = 1 AND S.exerciseid = S1.exerciseid ) ;
```

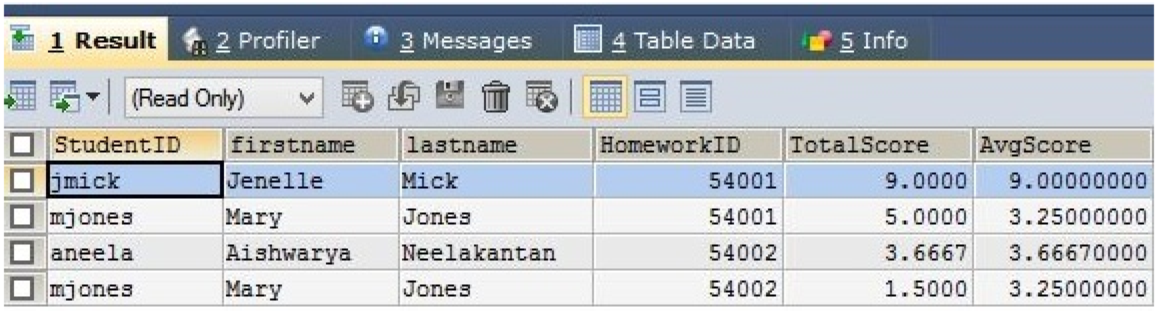


HomeworkID	StudentID	FirstName	LastName
54002	aneela	Aishwarya	Neelakantan
54001	mjones	Mary	Jones

- ❑ **For each student, show total score for each homework and average score across all homeworks ?**
    - ❑ _Assumption_ : CourseToken is CSC540FLL14, TotalScore is score for a homework calculated on basis of scoretypeid and AvgScore is average of final scores across all homeworks for a student.

```
SELECT ft1.StudentID, U.firstname, U.lastname, ft1.HomeworkID, ft1.TotalScore, ft2.AvgScore
FROM (
SELECT FINAL_SCORE AS TotalScore, userid AS StudentID, exerciseid AS HomeworkID
FROM (
SELECT S.score AS FINAL_SCORE, S.userid, S.exerciseid
FROM submission S, exercises E
WHERE S.exerciseid = E.exerciseid AND E.scoretypeid = 1 AND E.token = "CSC540FALL14" AND
S.attemptnum >= ALL (
SELECT attemptnum
FROM submission S1
WHERE S1.exerciseid = S.exerciseid AND S1.userid = S.userid)
GROUP BY S.userid, S.exerciseid
UNION
SELECT AVG(S.score) AS FINAL_SCORE, S.userid, S.exerciseid
FROM submission S, exercises E
WHERE S.exerciseid = E.exerciseid AND E.scoretypeid = 2 AND E.token = "CSC540FALL14"
GROUP BY S.userid, S.exerciseid
UNION
SELECT MAX(S.score) AS FINAL_SCORE, S.userid, S.exerciseid
FROM submission S, exercises E
WHERE S.exerciseid = E.exerciseid AND E.scoretypeid = 3 AND E.token = "CSC540FALL14"
GROUP BY S.userid, S.exerciseid) AS temp
) AS ft1,
(
SELECT AVG(temp.FINAL_SCORE) AS AvgScore, userid AS StudentID
FROM (SELECT S.score AS FINAL_SCORE, S.userid, S.exerciseid
FROM submission S, exercises E
WHERE S.exerciseid = E.exerciseid AND E.scoretypeid = 1 AND E.token = "CSC540FALL14" AND
S.attemptnum >= ALL (
SELECT attemptnum
FROM submission S1
WHERE S1.exerciseid = S.exerciseid AND S1.userid = S.userid)
GROUP BY S.userid, S.exerciseid
UNION
SELECT AVG(S.score) AS FINAL_SCORE, S.userid, S.exerciseid
FROM submission S, exercises E
WHERE S.exerciseid = E.exerciseid AND E.scoretypeid = 2 AND E.token = "CSC540FALL14"
GROUP BY S.userid, S.exerciseid
UNION
SELECT MAX(S.score) AS FINAL_SCORE, S.userid, S.exerciseid
```

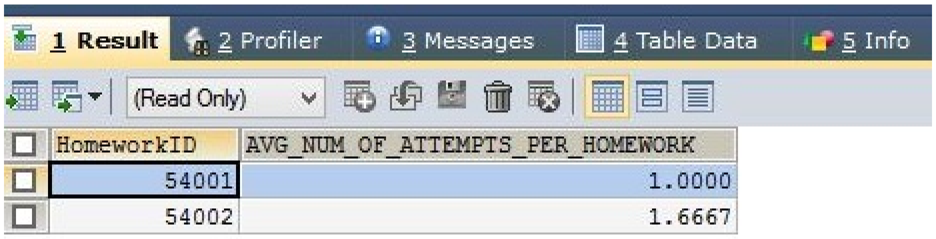
FROM submission S, exercises E
WHERE S.exerciseid = E.exerciseid AND E.scoretypeid = 3 AND E.token = "CSC540FALL14"
GROUP BY S.userid, S.exerciseid) AS temp
GROUP BY userid
) AS ft2, user U
WHERE ft1.StudentID = ft2.StudentID AND ft1.StudentID = U.userid;



StudentID	firstname	lastname	HomeworkID	TotalScore	AvgScore
jmick	Jenelle	Mick	54001	9.0000	9.00000000
mjones	Mary	Jones	54001	5.0000	3.25000000
aneela	Aishwarya	Neelakantan	54002	3.6667	3.66670000
mjones	Mary	Jones	54002	1.5000	3.25000000

- ❑ **For each homework, show average number of attempts ?**
  - ❑ *Assumption* : CourseToken is CSC540FALL14

SELECT temp1.exerciseid AS HomeworkID, temp2.acount / temp1.scount AS AVG_NUM_OF_ATTEMPTS_PER_HOMEWORK
FROM (
SELECT COUNT(*) AS scount, X.exerciseid AS exerciseid
FROM enrollment E, exercises X
WHERE E.token = X.token AND E.token="CSC540FALL14"
GROUP BY X.exerciseid) AS temp1,
(SELECT COUNT(*) AS acount, exerciseid
FROM submission
GROUP BY exerciseid) AS temp2
WHERE temp1.exerciseid = temp2.exerciseid;



HomeworkID	AVG_NUM_OF_ATTEMPTS_PER_HOMEWORK
54001	1.0000
54002	1.6667

9. Use case scenarios

9.1. Use Case 1: User creates account in system

Primary Actors: Student, Professor, TA

Preconditions:

- Network connection is active
- User does not already have an account in the system

Basic Flow of Events:

- User creates a profile by providing information namely First Name, Last Name, username, password, email address, type of user and optionally education level (if the user is a student).
- The user profile is created and the user is taken to his/her homepage.

Alternative Flows:

- The user leaves any of the required fields blank
    - A message is prompted to the user to fill the required missing fields
    - Account is not created.
- The user provides a username that is already taken
    - A message is displayed informing the user that the username is unavailable
    - He is prompted to enter an alternate username

9.2. Use Case 2: Student enrolls for a course

Primary Actors: Student

Preconditions:

- Network connection is active
- Student is already enrolled in the system

Basic Flow of Events:

- Student signs in to the system
- Student enters the course token provided by the professor
- System checks if the course selected is valid and adds the course to list of courses enrolled by the student

Alternative Flows:

- Student provides an invalid course token
    - A message is displayed informing that the token is invalid
    - Student is taken back to home page
- Student provides course token of an expired course
    - A message is displayed informing that the course has expired
    - Student is taken back to home page
- Student provides course token which is already full
    - A message is displayed informing the course is full
    - Student is taken back to home page
- Student provides course token to which is already enrolled
    - A message is displayed informing the course is already enrolled.
    - Student is taken back to home page
- Student tries to enroll for a course of different study level (Graduate/Undergraduate)
    - A message is displayed informing the student that the course if of a different level
    - Student is taken back to home page

- Student tries to enroll to a course which has one or more topics that match a course he in which he is already TA.
    - A message is displayed to the student stating the professor needs to approve.
    - A notification is sent to the professors of the clashing courses informing them that the student tried to enroll in a course with similar topic.
    - Student is taken back to home page.

9.3. Use Case 3: Student cancels enrollment from a course

Primary Actors: Student
Preconditions:

- Network connection is active
- Student is already enrolled in the system

Basic Flow of Events:

- Student signs in to the system
- Student enters the course token for which he/she wishes to cancel enrollment
- System checks if the course selected is valid and removes the course to list of courses enrolled by the student

Alternative Flows:

- Student provides an invalid course token
    - A message is displayed informing that the token is invalid
    - Student is taken back to home page
- Student provides course token of a course in which he is not enrolled
    - A message is displayed informing that the course has not been taken by the student
    - Student is taken back to home page
- Student provides course token which has expired
    - A message is displayed informing the course has expired
    - Student is taken back to home page

9.4. Use Case 4: Student attempts exercise

Primary Actors: Student
Preconditions:

- Network connection is active
- Student is already enrolled in the system
- Student is already enrolled in the course for which he/she is about to attempt the homework

Basic Flow of Events:

- Student signs in to the system
- Student enters the course token for which he/she wants to attempt the exercise
- Student selects exercise he/she wants to attempt
- Student attempts exercise and submits the attempt
- Student is provided marks based on his submission.

Alternate Flows:

- Student has expired the maximum number of allowed attempts
    - A message is displayed maximum number of attempts reached.

    - Student is taken back to course homepage

9.5. Use Case 5: Student views past submissions

Primary Actors: Student

Preconditions:

- Network connection is active
- Student is already enrolled in the system
- Student is already enrolled in the course for which he/she is about to view past submissions
- Student has already submitted assignments in the course

Basic Flow of Events:

- Student signs in to the system
- Student enters the course token for which he/she wants to view past submissions
- Student selects the submission he/she wants to view

Alternate Flows:

- No alternate flows

9.6. Use Case 7: Professor enrolls for teaching a course

Primary Actors: Professor

Preconditions:

- Network connection is active
- Professor is already enrolled in the system
- The course token the professor is trying to enroll for is already present in the database

Basic Flow of Events:

- Professor signs into the system.
- Professor enters the course token for which he/she wants to enroll
- Professor is enrolled in the course.

Alternate Flows:

- Professor provides an invalid course token
    - A message is displayed informing that the token is invalid
    - Option is provided for the professor to add the token
- Professor provides course token of an expired course
    - A message is displayed informing that the course has expired
    - Professor is taken back to home page
- Professor provides course token to which is already enrolled
    - A message is displayed informing the course is already enrolled.
    - Professor is taken back to home page

9.7. Use Case 6: Professor adds/removes TA

Primary Actors: Professor

Preconditions:

- Network connection is active

- Professor is already enrolled in the system
- Professor is enrolled to the course to which he/she would like to add TA
- The TA added by the professor is trying to enroll for is already present in the user database

Basic Flow of Events:

- Professor signs into the system
- Professor selects the course to which he wishes to add the TA
- Professor provides the username of the student who needs to be made the TA
- Student is added as TA for the course

Alternate Flows:

- Professor tries to add a student who is not a graduate
    - A message is displayed saying that the student the professor trying to add is not a graduate student.
    - The student is not added as a TA.
- Professor tries to add a student who is already added as a TA to the same course
    - A message is displayed stating the professor has already added the student as a TA for the same course.
    - The student is not added as a TA
- Professor tries to add a person who is not present in the user database
    - A message is displayed saying the username entered is invalid.
    - The professor is taken back to the course homepage.

9.8. Use Case 8: Professor adds exercise

Primary Actors: Professor

Preconditions:

- Network connection is active
- Professor is already enrolled in the system
- Professor is enrolled to the course to which he/she would like to add Homework

Basic Flow of Events:

- Professor signs into the system
- Professor selects the course to which he wishes to add the Homework
- Professor searches for questions based on a difficulty range and a topic.
- Professor provides the details of the homeworks such as questions, start date, end date, number of attempts and score selection method.
- Exercise is added to the course.

Alternate Flows:

- Professor sets end date of the exercise to a date after which the course has expired.
    - A message is displayed stating that the exercise duration exceeds the course duration.
    - The exercise is not added to the course.
- Professor misses to add data describing the exercise

    - A message is displayed stating the professor needs to enter required data for the exercise to be successfully posted.
    - The exercise is not posted to the course.

9.9. Use Case 9: Professor removes exercise

Primary Actors: Professor

Preconditions:

- Network connection is active
- Professor is already enrolled in the system
- Professor is enrolled to the course to which he/she would like to remove Homework
- The homework which the professor is trying to remove is already present in the database.

Basic Flow of Events:

- Professor signs into the system
- Professor selects the course from which he wishes to remove the Homework
- Exercise is removed from the course.

Alternate Flows:

- Professor selects exercise which has already started
    - A message is displayed stating that the exercise being deleted is already in progress.
    - The exercise is not removed from to the course.

9.10. Use Case 9: Professor modifies exercise

Primary Actors: Professor

Preconditions:

- Network connection is active
- Professor is already enrolled in the system
- Professor is enrolled to the course which contains the homework he/she is is trying to modify
- The homework which the professor is trying to modify is already present in the database.

Basic Flow of Events:

- Professor signs into the system
- Professor selects the course containing the homework which he wishes to modify
- Professor modifies the exercise by changing configurations appropriately
- Exercise configuration is modified.

Alternate Flows:

- Professor selects exercise which has already started
    - A message is displayed stating that the exercise being modified is already in progress.
    - The exercise is not modified

- Professor provides invalid configurations
    - A message is displayed which says that the exercise configuration is invalid
    - The exercise is not modified.

9.11. Use Case 10: Views reports of students

Primary Actors: Professor, TA

Preconditions:

- Network connection is active
- User is already enrolled in the system.
- The course token for which report needs to be viewed should be valid
- Professor/TA is enrolled to the course for which the report needs to be viewed

Basic Flow of Events:

- Professor/TA signs into the system
- Professor/TA selects the course for which they need to view the result
- Professor/TA selects the student whose details needs to be viewed

Alternate Flows:

- No alternate flow

9.12. Use Case 12: View homework

Primary Actors: Professor, TA

Preconditions:

- Network connection is active
- User is already enrolled in the system
- The course for which homework needs to be viewed should be valid and the user should be enrolled to the course.

Basic Flow of Events:

- Professor/TA signs into the system
- Professor/TA selects the course for which they need to view the result
- Professor/TA selects the homework which they need to view.

Alternate Flows:

- No alternate flows

9.13. Use Case 13: View notification

Primary Actors: Student, Professor, TA, Admin

Preconditions:

- Network connection is active
- User is already enrolled in the system

Basic Flow of Events:

- User signs into the system.
- User is shown the pending notifications.
- User can mark the notifications as seen.

Alternate Flows:

- No alternate flows.

9.14. Use Case 14: Generate notification

Primary Actors: Gradiance System, Student, Professor, TA, Admin

Preconditions:

- Network connection is active
- User to whom notification needs to be sent has an account on the system.
- At least one of the action that generates a notification needs to be performed.

Basic Flow of events:

- A student who is logged in tries to enroll to a course which has a matching topic with one of the courses for which he is a TA.
- A notification is sent to the professor of the 2 courses that the student who is a TA is trying to enroll to a course with matching topic.
- 24 hours prior to the end of homework a notification is sent to the students of the course.

Alternate Flows:

- No alternate flows.

10. Bonus Features

We have implemented 3 cool bonus features in our database. They are explained below.

★ **Top 5 difficult questions for each course token**

The difficult questions are determined by the number of times students answer them incorrectly. So when homeworks are submitted, the 'qcount' field in table 'topq' is incremented for the wrongly answered questions referenced by 'qid' (or entry created for a new incorrectly answered question). This way the professor of a course token can view the top 5 most difficult questions belonging to the topics covered in that course. This feature helps the professor understand the problems students face while tackling problems. Also it tells professors about the topics which need to be covered in more depth.

★ **Bonus points for a limited number of students for each homework**

The professor can optionally specify if they want to give bonus points for a particular homework. If yes, they need to specify the number of students N that can receive bonus points and the bonus points B. This information gets stored in the bonus table and is used while calculating homework scores (full score + B) on submission. Bonus is given to first N distinct students which finish the homework and score full marks. This feature promotes competition among the students and provides opportunity to students to make up for marks lost elsewhere.

★ **Feedback to system administrator**

This is a simple feature which stores the feedback provided by the Gradiance users to the database. This feedback is then displayed to the admin as notifications. This provides an essential medium to users to provide constructive feedback to website creators to make enhancements or improvements. All feedbacks are anonymous and even if notifications are deleted by admin, the feedbacks remain stored in a separate feedback table.