



NUS

National University
of Singapore

CS2102 DATABASE SYSTEMS

WEB-BASED DATABASE APPLICATION PROJECT

TOPIC C: COURSE MANAGEMENT SYSTEM

PROJECT GROUP 37

CHIN JING HONG

CLEMENT CHENG

JOEL CHANG ZHI KAI

LOUIS FREDRICK MISSON

Contents

1 Application Requirements, Functionalities and Constraints	3
1.1 Application Requirements and Functionalities	3
1.2 List of Constraints	4
2 Entity Relationship Model and Relation Schema	5
2.1 Entity Relationship Model (Full)	5
2.2 Entity Relationship Model and Schema (Breakdown)	6
2.3 Relation Schema Analysis	11
3 Non-Trivial Constraints enforced with triggers	12
3.1 Complex Queries	12
3.1.1 Course Average GPA (Semestral and Total)	12
3.1.2 Top Forum Posters	12
3.1.3 Least Active Forum Posters	14
3.2 Triggers	14
3.2.1 Removal of Previous Teaching Assistant Requests and Other Checks	14
3.2.2 New Thread also Begins as First Post	15
3.2.3 Update Once Student Rates a Professor	16
3.2.4 Edit to First Post Reflected in Thread	16
4 Interesting Features	17
4.1 Professor Rating	18
4.2 Course Average GPA	18
4.3 Forum Top Posters and Least Active Posters	18
5 Specification of Framework and Software Tools	20
6 Project Responsibilities Breakdown	20
7 Reflection	20

1 Application Requirements, Functionalities and Constraints

1.1 Application Requirements and Functionalities

Overview

Our web application, LumiRandom, functions as a simple course management system that handles the enrolment of students into various courses, as well as facilitates the access to relevant and necessary materials and tools for the students.

Specifics

Account Management

Every user of LumiRandom is either a student or a professor, and each user has a unique account to login into the system. Registration is not possible, and account ID is fixed. However, users are free to change their passwords and profile picture upon successful login.

Course Registration

Students will be able to request to enter a course of their liking, provided that they have not enrolled in the course before. Professors can then accept or reject a student to enter his/her course. If the request has been accepted, the student can then have access to a webpage with greater details on the course, such as classmates in the same course.

Student Groups

Professors are able to categorise students of the course into groups. Selected groups under the professor are also allowed access to forums created by the professor. In addition, a teaching assistant may be assigned to manage a group of students in the course by the professor.

Forums

A professor may create forms that allow access to certain groups in the course. The professor may read, write and delete entries in the forums, as stated in the task requirements. Students in the selected groups can have access to the forums specific to the course. Students may create threads in the forum, and read and write in the forum, but are not allowed to delete any entries, as stated in the task requirements.

Teaching Assistant Application

Students may request to be a Teaching Assistant (TA) for a course he/she has taken before and the professor can then accept/reject the student's application. Upon approval, the TA will have access to the forums that the group(s) he/she is taking has access to. Similarly, TAs are able to read and write in the forums, but are also able to delete entries in the forums too.

Additional Features

There are many additional features that we have included into the application. Sidebars and navigation bars are created for easy navigation through the pages. The web application has been designed to be simple and easy to use, while at the same time it includes the relevant functionalities for a course management system. The following are some examples of the application functionalities:

- Users can have access to a list of all the professors and students in the institution and also all the available courses, with added search functionalities
- Users with access to the forums will be able to rate individual posts made by others
- Under the complex queries section, we have implemented a function to rank the students such that users can identify the top and least active posters in the forums
- Students who are currently enrolled in a course managed by a professor will be able to rate the professor according to the professor's performance in the current semester.

1.2 List of Constraints

Below are the list of constraints we have set in our application, and are elaborated throughout the report.

Users

Every user has a unique account id

Every user is either a student or a professor, and cannot be both. Every TA is a student.

Courses

A course can be taken by 0 or more students.

A course can be managed by 0 or more professors in the current semester.

Students

Every student can only enrol to a course that is managed by a professor in the current semester

Every student cannot enrol to a course he/she has taken before

Every student can enrol to at most 6 courses per semester.

Every student can only be a TA for at most one course per semester.

Every student can be in 0 or more groups

Every student who is Year 2 and above, and has at least an A- grade for a course he/she has taken before can request to be a TA for that course

Every student can read and write in forums he/she has access to

Every student cannot delete entries in the forum.

Every student can edit the post he/she has made

Every student may rate the professor of a course he/she is currently enrolled only once, and cannot change the rating

Professors

Every professor can categorise students in his/her course into groups

Every professor can create forums, and allow access to the forums for at least 1 group he/she created

Every professor can read, write and delete entries in forums he/she has created

Every professor can delete an entire thread in the forum that he/she has created

Every professor can edit the post he/she has made

Teaching Assistants

Every TA is a student, i.e. a Professor is not a TA

Every TA is a student who is Year 2 and above, and has at least an A- grade in the course

Every TA may manage 0 or more groups assigned by the professor

Every TA has access to the same forum as the group he/she is managing

Every TA can read, write and delete entries to the forum he/she has access to as a TA

Every TA can edit the post he/she has made

Groups

Every group is created by exactly one professor

Every group must have at least 1 student

Every group can be managed by at most one TA

Every group consists of students that are all in the same course as the one managed by the professor

Every group can have access to 0 or more forums

Forums

Every forum is created by exactly one professor

Every forum must allow access to at least one group in the course

Groups that have access to forums must all be created by the same professor who created the forum

Every forum has 0 or more threads

Threads

Every thread 0 or more posts (actually has at least one and is implemented by a trigger explained in the Triggers section)

Every thread is the first post of the thread, i.e. thread topic

Threads can be deleted by professors who created the forum

Posts

Every post may be rated by 0 or more users who have access to the forum

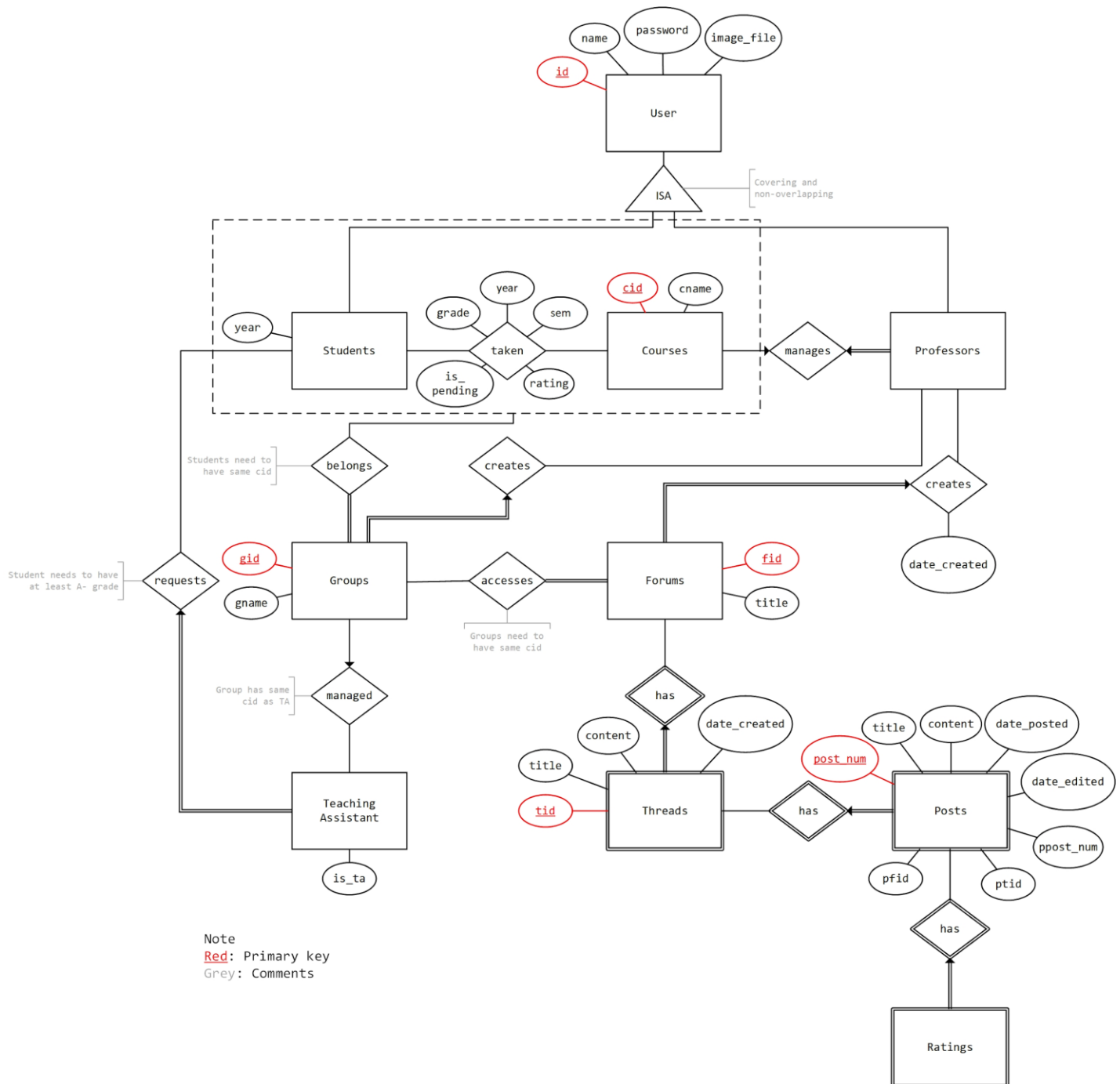
The first post of the thread is the thread topic

Ratings

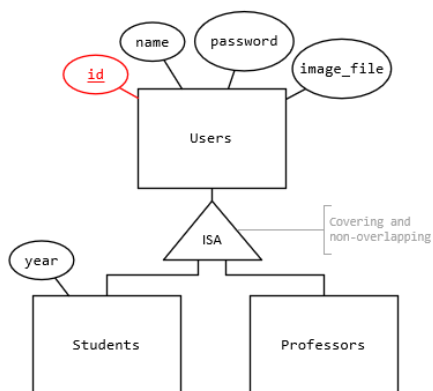
Users can only rate each post at most once, but can update their rating

2 Entity Relationship Model and Relation Schema

2.1 Entity Relationship Model (Full)



2.2 Entity Relationship Model and Schema (Breakdown)



Users

Here, we define every user of the LumiRandom Course Management System to have a unique account. Each user's account has a primary key **id** (which couples as the login account ID), which uniquely identifies the user's name, password and image file (i.e. profile picture).

ISA

Moreover, in our model, every user is either a student or a professor of the institution, but not both (covering and non-overlapping constraint).

Students

Every student is identified by the primary key **sid**, which is a foreign key of Users **id**. Each student also has a year attribute, representing the current year of the student, and is set to be between 1 to 5 inclusive.

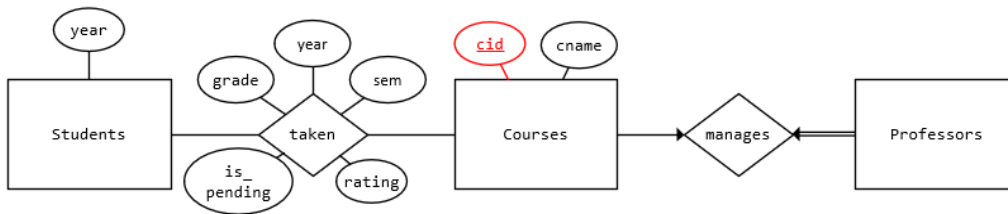
Professors

Similarly, every professor is identified by the primary key **pid**, which is a foreign key of Users **id**. Each professor also manages exactly one course/module, in which the course is identified by its cid. Course managed by each professor is unique and cannot be shared with other professors. The cid of the course is a foreign key of Courses and elaborated later.

```
CREATE TABLE Users (
  id          VARCHAR(10) PRIMARY KEY,
  name        VARCHAR(100) NOT NULL,
  image_file  VARCHAR(100) NOT NULL DEFAULT 'default.jpg',
  password    VARCHAR(100) NOT NULL
);

CREATE TABLE Students (
  sid         VARCHAR(10) PRIMARY KEY REFERENCES Users(id),
  year        INTEGER NOT NULL CHECK (year >= 1 AND year <= 5)
);

CREATE TABLE Professors (
  pid         VARCHAR(10) PRIMARY KEY REFERENCES Users(id),
  cid         VARCHAR(10) NOT NULL UNIQUE REFERENCES Courses
);
```



Courses (Entity)

Every course in the institution has a primary key **cid**, which uniquely identifies its corresponding course name. Also, at most one course is managed by a professor. Each course is taken by 0 or more students.

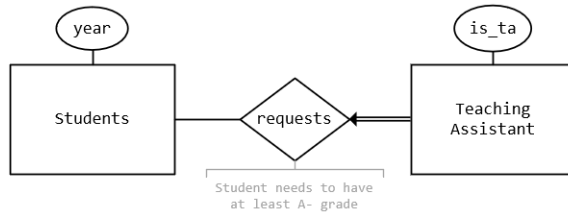
Taken (Relationship)

Each student can take 0 or more courses. Each student-course pair has relevant information of the student's academic tenure such as the year and semester taken, the grade obtained, the rating given by students to the professor/course, and a 'is pending' status. The grade is only reflected for courses taken in the past, and is set to NULL for courses taken in the current year and semester. Moreover, rating can only be given by students taking the course in the current year and semester, and the rating given is out of 10. The 'is pending' status takes a boolean value and is set to true when a student requests to enter a course, and thereafter set to false once the professor accepts the request. Additionally, a limit of 6 courses can be taken by students per semester, but is not reflected in the diagram. Most of the above constraints are implemented on the application side.

```

CREATE TABLE Courses (
    cid          VARCHAR(10) PRIMARY KEY,
    cname        VARCHAR(100) NOT NULL
);

CREATE TABLE TakenCourses (
    sid          VARCHAR(10) REFERENCES Students,
    cid          VARCHAR(10) REFERENCES Courses,
    year         VARCHAR(10) NOT NULL,
    sem          INTEGER NOT NULL CHECK (sem = 1 OR sem = 2),
    grade        VARCHAR(5) DEFAULT NULL,
    rating       INTEGER DEFAULT NULL,
    is_pending   BOOLEAN NOT NULL DEFAULT FALSE,
    PRIMARY KEY (sid, cid)
    CHECK (rating IS NULL OR (rating >= 0 AND rating <= 10))
);
  
```



Teaching Assistants (Entity)

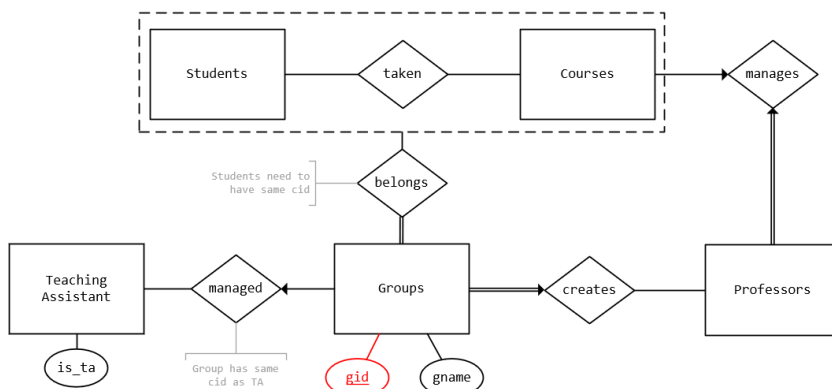
Each student can sign up to be a TA, which is identified by primary key **sid**, **cid**, and are foreign keys of TakenCourses table. The criteria to be a TA we set is that the student needs to be at least Year 2, and has at least an A- grade for the course (enforced in application). TA has a 'is ta' status, and is set to false when a student requests to join, and thereafter set to true once the professor approves the request.

Students may make multiple requests for different courses, but once a student is approved to be a TA for a course, all other requests made by the students will be removed, and students cannot make any other request (i.e. student can only be a TA for one course). This is enforced by a trigger discussed later.

```
CREATE TABLE TeachingAssistants (
  sid          VARCHAR(10),
  cid          VARCHAR(10),
  is_ta        BOOLEAN NOT NULL DEFAULT FALSE,
  PRIMARY KEY (sid, cid),
  FOREIGN KEY (sid, cid) REFERENCES TakenCourses(sid, cid)
);
```

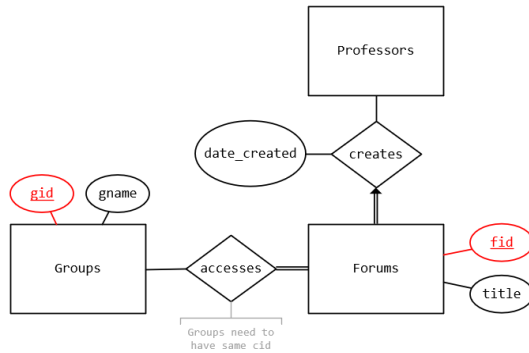
Groups (Entity)

Every group can be identified by its primary key **gid**, and has a group name. Each group is created by one professor, and a professor may create 0 or more groups. The professor may select at least one of his/her students to be in a group. Furthermore, a group can be managed by at most one teaching assistant of that course. The sid, cid pair are foreign keys of the TeachingAssistants representing the TA if any, and pid is a foreign key of Professors representing the creator of the group.



```
CREATE TABLE Groups (
  gid          INTEGER PRIMARY KEY,
  gname        VARCHAR(50) NOT NULL,
  pid          VARCHAR(10) NOT NULL REFERENCES Professors,
  sid          VARCHAR(10),
  cid          VARCHAR(10),
  FOREIGN KEY (sid, cid) REFERENCES TeachingAssistants(sid, cid),
  CHECK ((sid IS NOT NULL AND cid IS NOT NULL) OR (sid IS NULL AND cid IS NULL))
);

CREATE TABLE GroupInfo (
  gid          INTEGER REFERENCES Groups,
  sid          VARCHAR(10) REFERENCES Students,
  PRIMARY KEY (gid, sid)
);
```

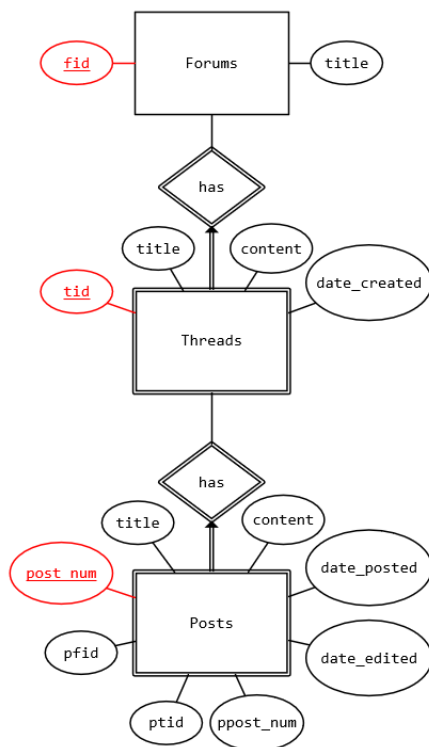
Forums (Entity)

Every forum is identified by its primary key **fid**, and has a forum title. Like groups, each forum is created by one professor, and a professor may create 0 or more forums. The professor may select at least one of the groups he/she created to have access to the forum, students and TA included. Moreover, the date and time when the professor creates a forum is also recorded. There is also a foreign key pid which represents the professor who created the forum.

```

CREATE TABLE Forums (
  fid          INTEGER PRIMARY KEY,
  title        VARCHAR(60) NOT NULL,
  pid          VARCHAR(10) NOT NULL REFERENCES Professors,
  date_created TIMESTAMP NOT NULL
);

CREATE TABLE ForumInfo (
  fid          INTEGER REFERENCES Forums,
  gid          INTEGER REFERENCES Groups,
  PRIMARY KEY (fid, gid)
);
  
```



Threads (Entity)

Users with access to the forum may create a forum thread. Every thread thus has primary key **fid**, **tid**, and fid is a foreign key of Forums. Each thread has a thread title, content and the date it was created. In addition, professor may delete a thread, and this is not reflected in the diagram but instead enforced in the application.

Posts (Entity)

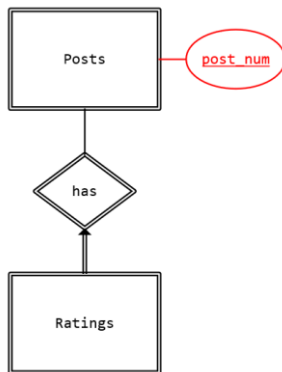
Correspondingly, users may post in a forum thread. Every post has a primary key **fid**, **tid**, **post_num**, and fid, tid is a foreign key of Threads. Each post has a title, content and date/time it was posted. All users with access to the forum can post on the thread by either replying to the thread topic, or to a post. As such, each post has a parent fid, tid, and post_num that references to the post it is replying to. The author of the post may edit the post and the date/time of last modification is recorded. However, only the professor and TAs of the groups can delete posts in the thread. As with Threads, this is not shown in the diagram but is implemented in the application.

```

CREATE TABLE Threads (
  fid          INTEGER REFERENCES Forums ON DELETE CASCADE,
  tid          INTEGER,
  id           VARCHAR(10) NOT NULL REFERENCES Users,
  title        VARCHAR(60) NOT NULL,
  content       TEXT NOT NULL,
  date_created  TIMESTAMP NOT NULL,
  PRIMARY KEY (fid, tid)
);

CREATE TABLE Posts (
  fid          INTEGER,
  tid          INTEGER,
  post_num     INTEGER,
  id           VARCHAR(10) NOT NULL,
  title        VARCHAR(60) NOT NULL,
  content       TEXT NOT NULL,
  date_posted  TIMESTAMP NOT NULL,
  date_edited  TIMESTAMP,
  pfid         INTEGER,
  ptid         INTEGER,
  ppost_num    INTEGER,
  PRIMARY KEY (fid, tid, post_num),
  FOREIGN KEY (fid, tid) REFERENCES Threads(fid, tid) ON DELETE CASCADE,
  FOREIGN KEY (pfid, ptid, ppost_num) REFERENCES Posts(fid, tid, post_num) ON DELETE CASCADE,
  CHECK (pfid = fid),
  CHECK (ptid = tid),
  CHECK (ppost_num IS NULL OR ppost_num <> post_num)
);

```



Ratings (Entity)

Lastly, users may give a rating to a post, and the average rating of the post will be displayed in the application. A rating has the primary key fid, tid, post_num, id, and fid, tid, post_num is a foreign key of Posts. Rating is restricted to between 0 and 5 inclusive, or in other words, users rate a post out of 5.

```

CREATE TABLE Ratings (
  fid          INTEGER,
  tid          INTEGER,
  post_num     INTEGER,
  id           VARCHAR(10) REFERENCES Users,
  rating       INTEGER NOT NULL CHECK (rating >= 0 AND rating <= 5)
  PRIMARY KEY (fid, tid, post_num, id),
  FOREIGN KEY (fid, tid, post_num) REFERENCES Posts(fid, tid, post_num) ON DELETE CASCADE
);

```

2.3 Relation Schema Analysis

The tables we have created are mostly in BCNF/3NF. This is probably because we followed the ER diagram closely and constructed our tables based from the diagram. However, there is one table that is not in either BCNF or 3NF - the Groups table.

In the Groups table, we identified gid as the primary key of the table. However, there exists a non-trivial functional dependency $pid \rightarrow cid$, but pid is not a superkey of the table. The pid attribute is used to identify the professor who created the group, and the cid attribute is actually a foreign key together with sid , referenced from the TeachingAssistants table, to identify the teaching assistant assigned to the group (if any). A possible solution would of course be to split the table further. One approach could be as below, where the information of the TA who managed the group is represented in another table, GroupsTA.

```
CREATE TABLE Groups (  
  gid          INTEGER PRIMARY KEY,  
  gname        VARCHAR(50) NOT NULL,  
  pid          VARCHAR(10) NOT NULL REFERENCES Professors  
);  
  
CREATE TABLE GroupTAs (  
  gid          INTEGER PRIMARY KEY,  
  sid          VARCHAR(10),  
  cid          VARCHAR(10),  
  FOREIGN KEY (sid, cid) REFERENCES TeachingAssistants(sid, cid)  
);
```

Another approach is to represent the information of the professor who created the group in another table. Nonetheless, we contemplated and decided against decomposing the original Groups table further on three fronts:

Firstly, the truncated Groups table and the other decomposed table would share a one-one relationship, and this is usually enforced by combining the two tables together, with a shared primary key. Thus, decomposing the table would defeat this purpose.

Secondly, it would be slightly more difficult to enforce the one-one relationship since we have to ensure that every group has its information present in both tables. Of course, we may use functions and triggers to accomplish this, but again, it is a bit more cumbersome.

Lastly, we have already enforced that the professor can assign TAs of his/her course to the created group. This effectively implies that the functional dependency $pid \rightarrow cid$ would hold since the $cids$ will complement one another.

3 Non-Trivial Constraints enforced with triggers

3.1 Complex Queries

3.1.1 Course Average GPA (Semestral and Total)

```
CREATE OR REPLACE FUNCTION find_gpa(c_id VARCHAR)
RETURNS TABLE(cid VARCHAR, year VARCHAR, sem INTEGER, gpa NUMERIC, avegpa NUMERIC)
AS $content$
    WITH AverageGPA AS (
        SELECT cid, year, sem, ROUND(AVG(grade), 4) GPA
        FROM (
            SELECT sid, cid, year, sem, CASE
                WHEN grade = 'A+' THEN 5.0
                WHEN grade = 'A' THEN 5.0
                WHEN grade = 'A-' THEN 4.5
                WHEN grade = 'B+' THEN 4.0
                WHEN grade = 'B' THEN 3.5
                WHEN grade = 'B-' THEN 3.0
                WHEN grade = 'C+' THEN 2.5
                WHEN grade = 'D+' THEN 1.5
                WHEN grade = 'D' THEN 1.0
                WHEN grade = 'F' THEN 0
            END grade
            FROM TakenCourses
            WHERE grade IS NOT NULL
        ) studentGrades
        GROUP BY cid, year, sem
    ), adjustedGPA AS (
        SELECT cid, year, sem, COALESCE(GPA, 0.0) GPA
        FROM AverageGPA
    )
    SELECT *, (
        SELECT ROUND(AVG(GPA), 4)
        FROM adjustedGPA
        WHERE cid = A.cid
    ) average_GPA
    FROM adjustedGPA A
    WHERE cid = c_id
    ORDER BY cid, year, sem;
$content$
LANGUAGE sql;
```

Description

This complex query computes the average Grade Point Average (GPA) of a course for each semester. This is done by taking the grades students obtained for the course in a particular semester, and converting each grade to its respective Cumulative Average Point (CAP) value. Thereafter, the average is calculated. In addition, the collative average GPA across all the semesters so far is also computed. In the SQL code, the complex query is implemented as a function, and by passing in the cid of the course of interest as a variable, we can extract the information with respect to that course.

Function

The average GPA obtained can be used as an indicator of the general performance of a cohort, and also as comparison with past cohorts to gauge the difficulty of the course based on the students' performance. The professor of the course can then make necessary modifications to the course materials for upcoming semesters. For the students, the information can act as both an incentive or a deterrence which will help them to decide whether they should take the course in that semester.

3.1.2 Top Forum Posters

```
CREATE OR REPLACE FUNCTION rank_posts(forum_id INTEGER, min_posts INTEGER, num INTEGER)
RETURNS TABLE(fid INTEGER, ranking BIGINT, id VARCHAR, forum_rating NUMERIC, total_posts BIGINT)
AS $content$
    WITH FindTotalRatingAndPosts AS (
```

```

SELECT P.fid, P.id, ROUND(AVG(post_rating), 2) forum_rating, TP.total total_posts
FROM Forums F, Threads T, (
    SELECT P.id, P.fid, P.tid, P.post_num, ROUND(AVG(rating), 2) post_rating
    FROM Ratings R RIGHT JOIN Posts P
    ON R.fid = P.fid AND R.tid = P.tid AND R.post_num = P.post_num
    WHERE P.id IN (SELECT sid FROM Students)
    GROUP BY P.id, P.fid, P.tid, P.post_num
) P, (
    SELECT P.id, P.fid, COUNT(*) total
    FROM Forums F, Threads T, Posts P
    WHERE F.fid = T.fid AND T.fid = P.fid AND T.tid = P.tid
    AND P.id IN (SELECT sid FROM Students)
    GROUP BY P.id, P.fid
) TP
WHERE F.fid = T.fid and T.fid = P.fid AND T.tid = P.tid
AND P.fid = TP.fid AND P.id = TP.id AND post_rating IS NOT NULL
GROUP BY P.fid, P.id, total_posts
HAVING TP.total >= min_posts
), RankPosts AS (
    SELECT *, RANK() OVER (
        PARTITION BY fid ORDER BY forum_rating DESC, total_posts DESC
    ) ranking
    FROM (
        SELECT DISTINCT fid, forum_rating, total_posts FROM FindTotalRatingAndPosts
    ) TopDistinct
)
SELECT fid, ranking, id, forum_rating, total_posts
FROM FindTotalRatingAndPosts NATURAL JOIN RankPosts
WHERE ranking <= num AND fid = forum_id
ORDER BY fid, ranking, id;
$content$
LANGUAGE sql;

```

Description

This complex query identifies students who have posted at least n number of times in the forum. They are then further ranked according to their average rankings of their posts in the forum, followed by the total number of posts they have posted in the forum if necessary to break ties. Thereafter, students with the top k [ranking, total posts] pair are selected. As such, there may be more than k students selected if they share the same ranking and total posts. A function is used to implement this query since it is quite dynamic, and variables fid of the forum, n and k can be passed into the function to get the ranked students of the forum.

Function

By finding the top posters of the forum, the professor can have a general idea of the students who are not only the most active, but also make substantial forum posts that resulted in higher ratings. A scoreboard of the top 3 posters is also featured in our application, and this may incentivise students to aim for one of the top places.

3.1.3 Least Active Forum Posters

```
CREATE OR REPLACE FUNCTION expose_students(forum_id INTEGER, min_posts INTEGER, num INTEGER)
RETURNS TABLE(fid INTEGER, ranking BIGINT, sid VARCHAR, total_posts BIGINT, last_posted_date
TIMESTAMP)
AS $content$
    WITH SortedStudents AS (
        SELECT BS.fid, BS.sid, CASE
            WHEN total_posts IS NULL THEN 0
            ELSE total_posts
        END total_posts, last_posted_date
        FROM (
            SELECT P.fid, P.id, COUNT(*) total_posts, MAX(date_posted) last_posted_date
            FROM Posts P
            WHERE P.id IN (SELECT sid FROM Students)
            GROUP BY P.fid, P.id
        ) TP RIGHT JOIN (
            SELECT DISTINCT FI.fid, GI.sid
            FROM GroupInfo GI NATURAL JOIN ForumInfo FI
        ) BS
        ON BS.fid = TP.fid AND BS.sid = TP.id
    ), RankStudents AS (
        SELECT *, RANK() OVER (
            PARTITION BY fid ORDER BY total_posts, last_posted_date
        ) ranking
        FROM (
            SELECT DISTINCT fid, total_posts, last_posted_date
            FROM SortedStudents
            WHERE total_posts < min_posts
        ) LastDistinct
    )
    SELECT RS.fid, RS.ranking, SS.sid, RS.total_posts, RS.last_posted_date
    FROM SortedStudents SS INNER JOIN RankStudents RS
    ON RS.fid = SS.fid AND RS.total_posts = SS.total_posts
    AND (RS.last_posted_date = SS.last_posted_date
        OR RS.last_posted_date IS NULL AND SS.last_posted_date IS NULL)
    WHERE ranking <= num AND RS.fid = forum_id
    ORDER BY fid, ranking, sid;
$content$
LANGUAGE sql;
```

Description

This complex query identifies students who have posted at less than n number of times in the forum. They are then further ranked according to their total number of posts made in the forum so far, followed by the date/time of their last post (if any) to break ties. Thereafter, students with the k lowest [total posts, last posted date] pair are selected. As such, there may be more than k students selected if they share the same total posts and last posted date. As like the other complex queries, a function is used to implement this query, and variables fid of the forum, n and k can be passed into the function to get the ranked students of the forum.

Function

Finding the least active posters of the forum allows the professor to easily identify students who rarely use the forums. If forum posts are compulsory for the course, it can also identify potentially lazy or weaker students, and the professor can then pay closer attention to them. Moreover, this query is implemented as a feature in the web application, and may act as a form of impetus for students to participate more actively in the forum to avoid being the last.

3.2 Triggers

3.2.1 Removal of Previous Teaching Assistant Requests and Other Checks

```
CREATE OR REPLACE FUNCTION remove_request()
RETURNS TRIGGER AS
$content$
```

```

BEGIN
  IF (SELECT year FROM Students WHERE sid = NEW.sid) <= 1 OR
  NEW.cid NOT IN (SELECT cid FROM Professors) OR
  NOT EXISTS (
    SELECT *
    FROM TakenCourses
    WHERE sid = NEW.sid AND cid = NEW.cid AND grade LIKE 'A%'
  )
  THEN RETURN NULL;
END IF;
IF EXISTS (
  SELECT *
  FROM TeachingAssistants
  WHERE sid = NEW.sid AND is_ta = True
) THEN RETURN NULL;
END IF;
IF NEW.is_ta = True THEN
  DELETE FROM TeachingAssistants WHERE sid = NEW.sid AND is_ta = False;
END IF;
RETURN NEW;
END;
$content$
LANGUAGE plpgsql;
CREATE TRIGGER delete_ta
BEFORE INSERT OR UPDATE ON TeachingAssistants
FOR EACH ROW
EXECUTE PROCEDURE remove_request();

```

Description

Constraint: A student can only be a TA for one course.

The trigger handles the insertion (and update) of the TeachingAssistants table. Students must be at least Year 2, and have at least an A- grade to request to be a TA for that course. Moreover, that course must be available in the current semester. The trigger returns NULL if those conditions are not met. In addition, a student can only be a TA for one course. Upon approval by the professor to be a TA, the trigger automatically deletes all records of the student's requests for other courses. Any other requests made by the student after being a TA will return as NULL by the trigger. Being a TA for more than one course may potentially affect the student's academic performance if too much responsibility is assigned to the student.

3.2.2 New Thread also Begins as First Post

```

CREATE OR REPLACE FUNCTION add_post()
RETURNS TRIGGER AS
$content$
BEGIN
  INSERT INTO Posts VALUES (NEW.fid, NEW.tid, 1, NEW.id, NEW.title, NEW.content,
    NEW.date_created, NULL, NULL, NULL, NULL);
  RETURN NULL;
END;
$content$

```

```

LANGUAGE plpgsql;
CREATE TRIGGER new_thread
AFTER INSERT ON Threads
FOR EACH ROW
EXECUTE PROCEDURE add_post();

```

Description

Constraint: The first post of a thread is always based from the thread itself, and has the same title and content as the thread. The trigger automatically inserts a new post when a new thread is inserted into Threads. The post_num of posts is set to a default value of 1, indicating that it is the first post of the thread, while the rest of the common values follow exactly as that of the inserted Thread. The first post is meant to initiate a thread discussion as the thread topic since users can only make posts (aside from creating a new thread) by replying to other posts in the forum.

3.2.3 Update Once Student Rates a Professor

```

CREATE OR REPLACE FUNCTION update_rated()
RETURNS TRIGGER AS
$$
BEGIN
    IF NEW.rating IS NULL THEN
        RETURN NULL;
    END IF;
    IF NEW.year <> '2019/2020' AND NEW.sem <> 1 THEN
        RETURN NULL;
    END IF;
    IF OLD.rating IS NOT NULL THEN
        RETURN NULL;
    ELSE
        RETURN NEW;
    END IF;
END;
$$
LANGUAGE plpgsql;
CREATE TRIGGER update_rated
BEFORE UPDATE on takencourses
FOR EACH ROW EXECUTE PROCEDURE update_rated();

```

Description

Constraint: Student can rate a professor of the course he/she is enrolled in the current semester only once. This trigger handles the update to the rating of the professor/course. A student may rate a professor only once. If the student has already rated before, the trigger returns NULL instead. This prevents students from rating the professors too many times so as to give users of the site an accurate representation of whether the students deemed the materials taught by the professor are effective. The rating may also help to motivate or urge the professor to make improvements.

3.2.4 Edit to First Post Reflected in Thread

```

CREATE OR REPLACE FUNCTION edit_thread()
RETURNS TRIGGER AS
$content$
BEGIN
    IF NEW.post_num = 1 THEN
        UPDATE Threads
        SET title = NEW.title, content = NEW.content
        WHERE fid = NEW.fid AND tid = NEW.tid;
    END IF;
    RETURN NULL;
END;
$content$
LANGUAGE plpgsql;
CREATE TRIGGER new_thread

```



```
AFTER UPDATE ON Posts
FOR EACH ROW
EXECUTE PROCEDURE edit_thread();
```

Description

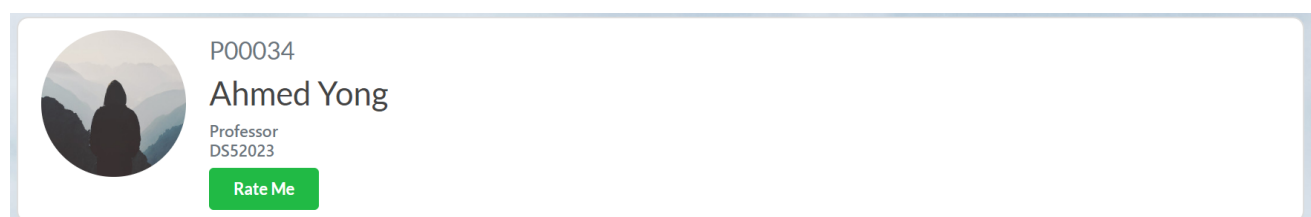
Constraint: The thread always has the same title and content as the first post, which acts as the thread topic.

The trigger automatically updates the thread when there is an update made to the first post of the thread, but not so for the other posts of the thread. This normally occurs when the thread creator edits the thread topic. The thread title and content should always match the information of the first post, and should not be different.

4 Interesting Features

4.1 Professor Rating

The LumiRandom application provides a feature to rate each professor, according to how well he/she performs in the semester. The rating will then be displayed on the professor's page, as a guide for future students to decide on whether they should take a module based on the professor's ratings.



Firstly, the student will view the profile of the professor of one of his/her current courses. If he/she has not rated the professor yet, an option to rate the professor will be available. A pop up will appear upon clicking the button. The student can input a rating between 0 and 10. On the professor's end, they are able to view their average rating.

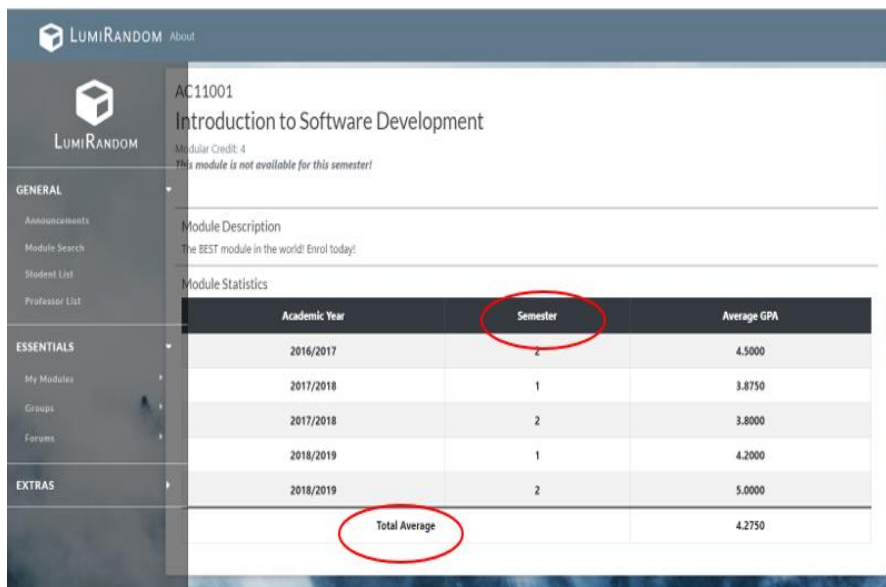
The handling of the rating is assisted by Trigger 3

4.2 Course Average GPA

This feature provides students and professors with an understanding of the difficulty of the module by using the average GPA of the courses as an indicator of difficulty level. This allows professors for better moderation of future module tests and allows students to make a more informed choices in planning their modules for the semester.

As seen in the screenshot provided, the course average GPA will be displayed together with the module search page, so that any users looking at the modules will be able to access the average GPA with ease.

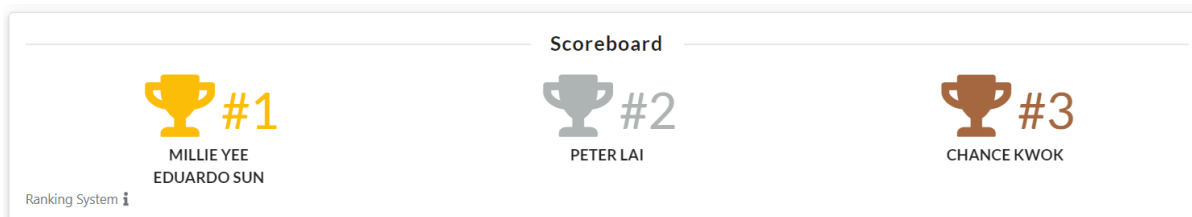
The implementation of find the course's average GPA is assisted by complex query 1



Academic Year	Semester	Average GPA
2016/2017	2	4.5000
2017/2018	1	3.8750
2017/2018	2	3.8000
2018/2019	1	4.2000
2018/2019	2	5.0000
Total Average		4.2750

4.3 Forum Top Posters and Least Active Posters

The first of these features identifies the top posters of the forum. A scoreboard is displayed that shows the top 3 posters, given a criteria where students must make at least 5 posts. Students are ranked based on the average ratings of their posts, followed by their total number of posts to break ties. A sample is shown below:



Rank	Poster Name
#1	MILLIE YEE EDUARDO SUN
#2	PETER LAI
#3	CHANCE KWOK

Aside from that, users are also able to rank students with a different set of criteria, such as minimum number of posts, and the number of ranks to be shown. Similarly, users can also query for the least active posters, which are ranked based on their total number of posts, followed by the date and time of their last post. A snapshot is shown below:

TOP FORUM POSTS

Rank 🏆	Name	Rating	Number of Posts
1	Millie Yee	4.00	2
1	Eduardo Sun	4.00	2
2	Peter Lai	3.75	4
3	Chance Kwok	3.71	5

Minimum number of posts

Number of positions

Rank!

ENDANGERED STUDENTS

Rank	Name	Number of Posts	Date Last Posted
1	Solomon Ong	0	-
1	Omar Chau	0	-
1	Warren Ho	0	-
1	Danielle Bishop	0	-

Minimum number of posts

Number of positions

Expose!

5 Specification of Framework and Software Tools

Web Application Components	Framework/Software Tools
Frontend	Cascading Style Sheets (CSS) Bootstrap Semantic UI jQuery
Backend	Python Flask SQLAlchemy WTForms psycopg2
Database	PostgreSQL pgAdmin 4 SQLite DB Browser (SQLite)

6 Project Responsibilities Breakdown

Name	Responsibilities
CHIN JING HONG	Complex queries, triggers, report
CLEMENT CHENG	Front and backend implementation, complex queries, triggers, report
JOEL CHANG ZHI KAI	Front and backend implementation, complex queries, triggers, report
LOUIS FREDRICK MISSON	Complex queries, triggers, report

7 Reflection

The project boasts of its heavy emphasis on web-based application design to facilitate the implementation of database queries and other functionalities required by the project task. This proved to be of overwhelming difficulty to our group as no one had any prior experience with web design beforehand. In addition, the resources provided in the project guidelines is of little help at all. A large amount of time is invested by just self-learning the various procedures to set up and begin designing the web application. Most of the sources we got help from were the favourite StackOverflow and YouTube videos. Moreover, the syntax for HTML was quite a tough experience to get a grasp of, especially when it comes to the styling of the web page. Various components such as a scrollable table was not readily available online, and we had to customise the CSS accordingly, which took some time. Other areas of consideration include finding out how forms work, how to query the database using SQLAlchemy, and using jQuery to execute specific functions in our web application also consume time to fully understand and implement.

Another challenging aspect of this project is the design of the Entity Relationship diagram. There were multiple revisions made to the ER diagram along the way and the changes have to be reflected in other aspects of the project. For instance, any changes in the relation schema has to be updated in the queries and functions of the database design otherwise errors will be thrown. Moreover, new sets of data need to be prepared to be inserted into the affected tables. The data we constructed to insert into the tables were done through our own scripts, which took effort and time. The number of records varies from around a hundred to even a few thousands per table, and was quite troublesome to redo with every modification to the ER diagram.

Overall, there were various lessons we have learnt, such as prioritising the tasks that are more important, as well as proper delegation of roles within the group. There were many things that were new to us, but we just have to take it a step at a time in order to produce substantial results. Learning to develop a database web application from scratch was nonetheless a very satisfying experience as we see how it slowly improved and took shape as time passed. The project essentially had us to be somewhat proficient in web designing, but whether it will be useful in the future is another matter. We would say that the project had truly been a unique and insightful, though quite painful, experience for us all.