COURSE ASSESSMENT SYSTEM PROJECT

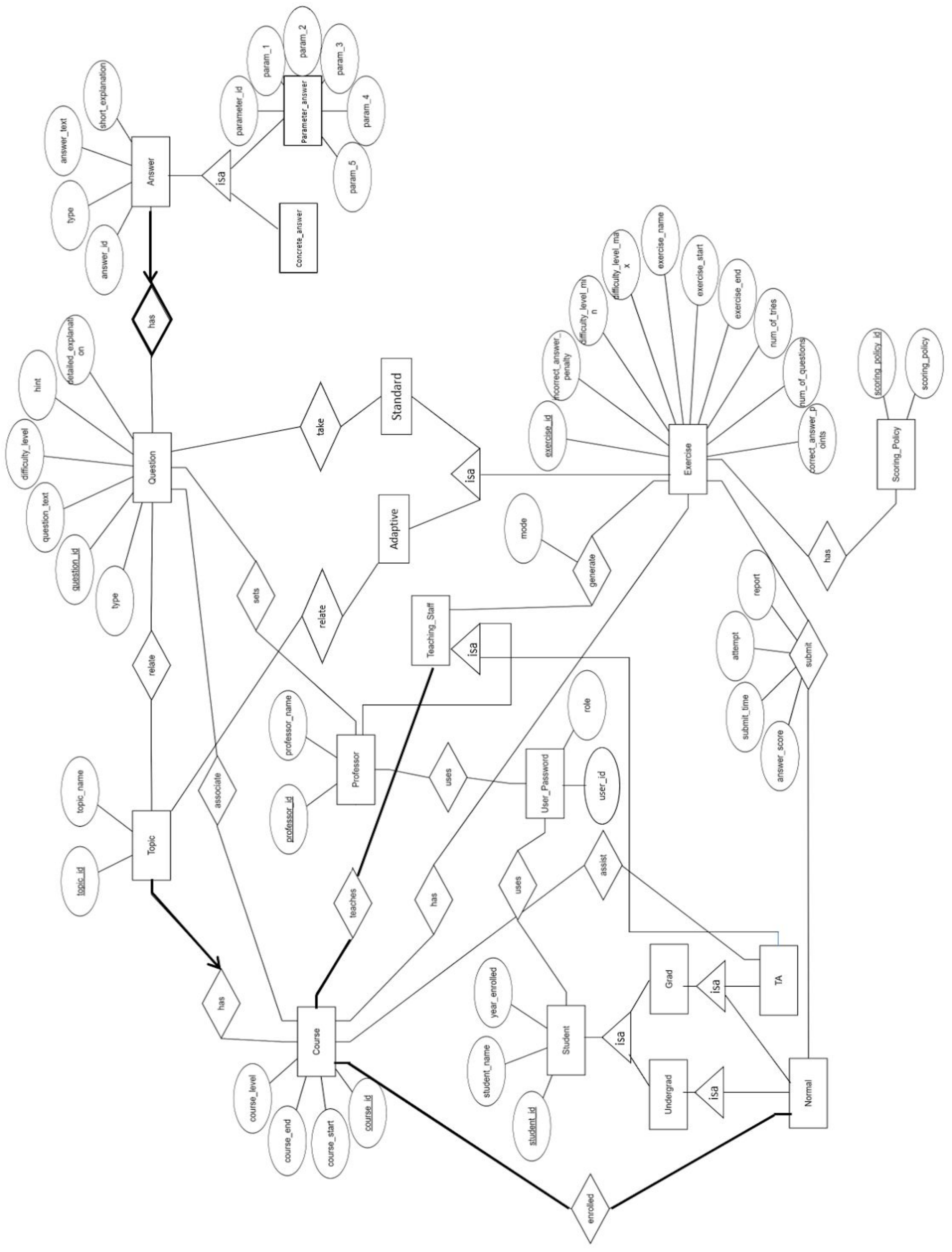
Milestone 2 Report

CSC540

Team Q

Yu-Lun Hong (yhong3)
Ashwin Risbood (arisboo)
Huy Tu (hqtu)
Xi Yang (yxi2)

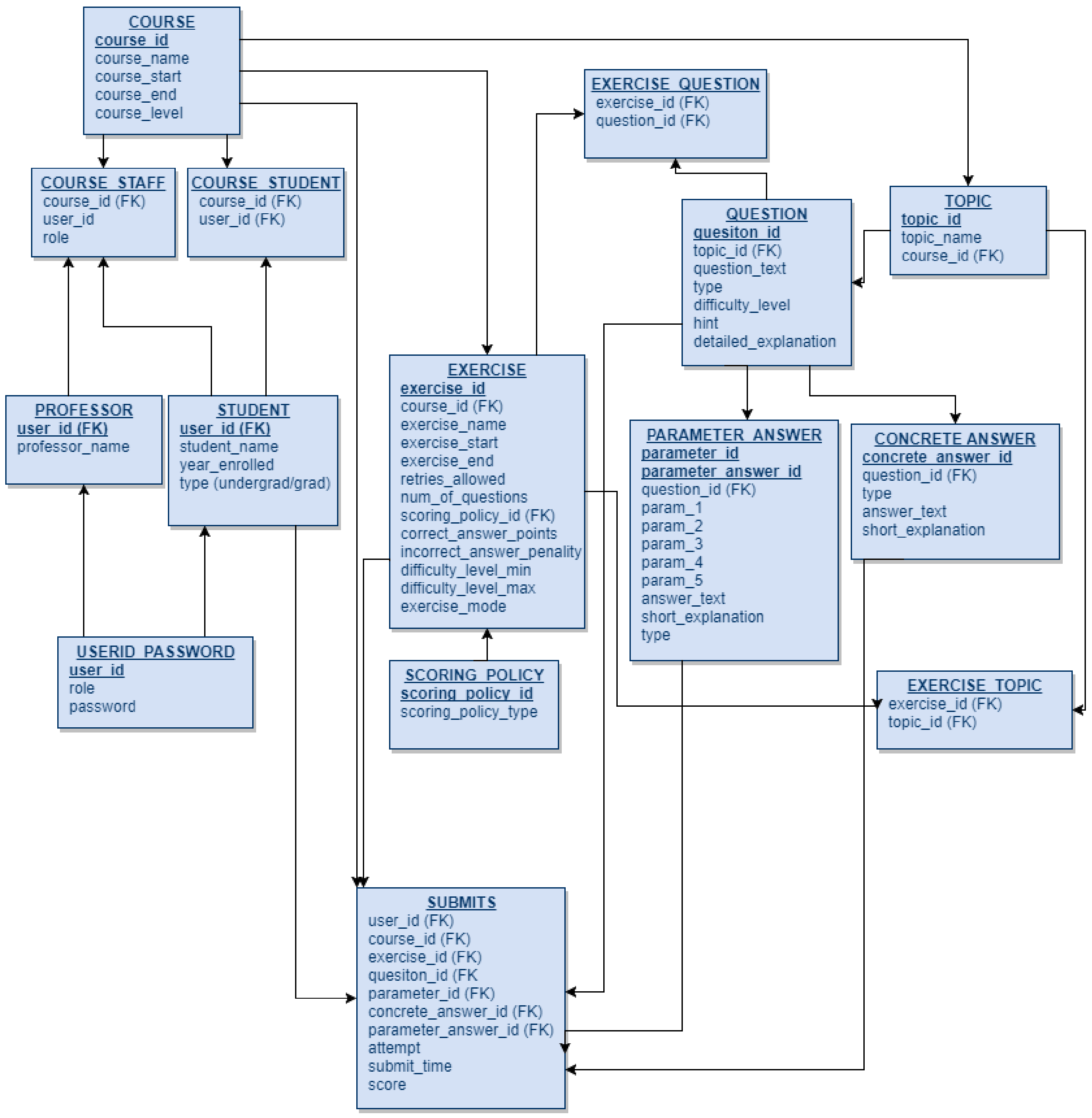
I. ER-Diagram:



Answer
answer_text
short_explanation
type
answer_id
isa
Parameter_answer
parameter_id
param_1
param_2
param_3
param_4
param_5
Concrete_answer
has
Question
detailed_explanation
hint
difficulty_level
question_text
question_id
type
take
Standard
isa
Adaptive
sets
relate
Exercise
exercise_id
incorrect_answer_penalty
difficulty_level_min
difficulty_level_max
exercise_name
exercise_start
exercise_end
num_of_tries
num_of_questions
correct_answer_points
Scoring_Policy
scoring_policy_id
scoring_policy
has
mode
generate
Teaching_Staff
isa
submit
report
attempt
submit_time
answer_score
relate
Topic
topic_name
topic_id
associate
Professor
professor_name
professor_id
uses
User_Password
role
user_id
has
teaches
has
uses
assist
Course
course_level
course_end
course_start
course_id
Student
year_enrolled
student_name
student_id
isa
Grad
Undergrad
isa
TA
isa
Normal
enrolled

II. Relational Model:

<u>Model</u>



A. COURSE
    a. Entity: this tables records the name, the start date, end date of the course, and course level (Grad/Undergrad).
    b. Primary key: course_id
    c. For a new added courses, course_id should not be already exist in the system
B. COURSE_STAFF (Teaches Relationship)
    a. Relationship: user (instructor/TA) is linked as a teaching staff of the course
    b. A course have exactly 1 professor and 0 or more TA. (For exact 1 professor, unique index: professor_dup_at_course)
    c. User need to exist in the system to be assigned as teaching staff (trigger: check_course_staff_existed)
    d. An TA and a professor can teach multiple courses.
    e. For a student to be assigned as TA:
        i. TA cannot be enrolled as a student in the course (trigger: check_ta_before_add)
        ii. they should not already be a TA (unique index: ta_dup_at_course)
        iii. They must be graduate student (trigger: check_ta_before_add)
C. COURSE_STUDENT (Enrolled relationship)
    a. Relationship: which student is enrolled in the course
    b. user_id should not exist here if they are in COURSE_STAFF under same course (trigger: enroll_student_not_ta)
    c. Enroll student to course
        i. Enrolled student should exist in the system before enrolling to the course (user_id is a foreign key)
        ii. Enrolled student has not been enrolled in that course yet, i.e. no duplicate entry of (course_id, user_id)
        iii. Enrolled student meets the course requirement with their level, e.g. Undergraduate student can only been enrolled in the undergraduate course (trigger: check_student_course_lv)
    d. Dropped student: student should has already been enrolled in that course (trigger: check_student_enrolled)
D. PROFESSOR
    a. Entity: a professor user can be potentially the instructor of the course, and they can set up a new course and assign themselves/other instructors for that course
    b. Primary key: user_id
E. STUDENT
    a. Entity: a student user can be enrolled in the class or assigned as TA
    b. Primary key: user_id

   c. If student.type is undergrad, then cannot be a TA (trigger: check_ta_before_add)
   d. If student_id exists in COURSE_STAFF with STUDENT.role = 2 and STUDENT.type is Grad, the student is a TA
   e. A student cannot be in COURSE_STAFF and COURSE_STUDENT at the same time (trigger: enroll_student_not_ta and check_ta_before_add)

F. USERID_PASSWORD
   a. Relationship: the user id and password
   b. Primary key: user_id
   c. Each user_id will have a corresponding password and role of them in the system
   d. Role = {1, 2, 3} (1: "professor", 2: "TA", 3: "student") (constraint: check_user_role)

G. EXERCISE
   a. Entity: shows details about a exercise
   b. Primary key: exercise_id
   c. A exercise is either a standard exercise (exercise_mode = 0), or a adaptive exercise (exercise_mode = 1) (constraint: exercise_mode)
   d. Teaching staff can determine the default setting of each exercise
   e. retries_allowed = {-1, 1, 2, 3} , -1 for unlimited (constraint: check_retries)
   f. Difficulty range = {1, 2, 3, 4, 5}, difficulty_level_min > difficulty_level_max (constraint: check_diff_min, check_diff_max, check_diff_range)
      i. We found both 1-5 and 1-6 in the instructions given, we chose 1-5
   g. Date for exercise_end > exercise_start (constraint: check_exercise_date)
      i. scoring_policy from SCORING_POLICY

H. SCORING_POLICY
   a. Entity: show the scoring policy can be used for grading
   b. Primary key: scoring_policy_id
   c. Possible entry: {latest attempt, maximum score, average score}
   d. The calculation for each policy cannot be determined in DBMS, covered in java

I. SUBMITS
   a. Relationship: record the student's attempt for each homework per question
   b. A enrolled student in COURSE_STUDENT will have different submission in SUBMITS
   c. There is no duplicate entry at (course_id, user_id, exercise_id, question_id, attempt)
   d. (exercise_id, question_id) need to exist in the exercise if it is a standard exercise (Trigger: chk_std_submit)
   e. Attempt should never exceed the retries_allowed at exercise table (Trigger: submit_attempt_exceed), unless it is listed as unlimited

J. EXERCISE_QUESTION (take relationship)

   a. Relationship: shows which questions are related to a standard exercise
   b. Constraint:
      i. No duplication entry of (exercise_id, question_id) allowed
      ii. Only standard exercise with (exercise_mode = 0) is allowed to be added. (Trigger: chk_ex_standard)

K. QUESTION
   a. Entity: Each question is related to a topic in Topic identified by topic_id.
   b. Primary key: question_id
      Each question has the question text, difficulty level(1-5), a hint to the solution and the explanation of the solution. (constraint: check_ques_diff)
   c. There are two type of question identified by the type attribute. (constraint: check_ques_type)
      i. Type 0: Concrete question.
      ii. Type 1: Parameterized question.
   d. Each parameter value combination should have correct and incorrect answers separately
   e. A question can allow a maximum of 5 parameters

L. PARAMETER_ANSWER (has relationship)
   a. Entity: Each different parameter_id correspond to a different set of the parameters available for the question_id and question's type 1 (parameterized question).
   b. Primary key: parameter_id, parameter_answer_id
   c. The maximum number of parameters for parameterized question will be 5, parameters not used can be leaved as null
   d. Each different parameter_answer_id correspond to a different answer_text, parameter_id, and question_id
   e. Type: {0, 1} (0: Incorrect, 1: Correct) (constraint: check_param_ans_type)
   f. No duplication entry of (parameter_id, parameter_answer_id, question_id) allowed
   g. For each question id and parameter_id in the concrete answer table, there are 1 correct answer type (1) and 3 incorrect answer type (0) (trigger: param_question_answer_count)

M. CONCRETE_ANSWER (has relationship)
   a. Entity: Each different concrete_answer_id correspond to the question_id and question's type 0 (concrete question).
   b. Primary key: concrete_answer_id
   c. No duplication entry of (concrete_answer_id, question_id) allowed
   d. Type: {0, 1} (0: Incorrect, 1: Correct) (constraint: check_conc_ans_type)
   e. For each question_id in the concrete answer table, there are 1 correct answer type (1) and 3 incorrect answer type (0) (trigger: concrete_question_answer_count)

N. TOPIC
    a. Entity: Records the topic_name and the related course_id in course.
    b. Primary key : topic_id
    c. Each topic can belong to zero or more courses.

O. EXERCISE_TOPIC (relats relationship)
    a. Relationship: This table records the topic_id and the exercise_id.
    b. topic_id refers from topic table,whereas exercise_id refers from exercise table
    c. Each exercise can have multiple topics also each topic can appear in multiple exercises.
    d. No duplication entry of (exercise_id, question_id) allowed.

III. Functional Dependencies:

A. These are the functional dependencies for our ER model:-

```
course_id → course_start, course_end, course_name, course_level

exercise_id → exercise_name, exercise_start, exercise_end, scoring_policy_id,
retries_allowed, num_of_questions, correct_answer_points,Incorrect_answer_penalty,
difficulty_level_min, difficulty_level_max, Exercise_mode

scoring_policy_id -> scoring_policy_type

user_id -> password, role

question_id → difficulty_level, hint, explanation, text, type, topic_id

student_id* → student_name, year_enrolled, type, staff_id

professor_id** → professor_name, staff_id

user_id, exercise_id, question_id,course_id, attempt → score

concrete_answer_id->question_id, type, answer_text, short_explanation

Parameter_id, parameter_answer_id -> question_id, param_1, param_2, param_3,
param_4, param_5, answer_text, short_explanation, type

topic_id-> topic_name, course_id

Note:
*  Student_id refers to USER_ID in student table i.e all USER_ids with  value 2/3
** professor_id refers to USER_ID in student table i.e all USER_ids with value 1
```

B. Next, all the functional dependencies are checked if they are in 3NF which means that they do not violate these following rules if X -> A is a functional dependency:
- A belongs to X.
- Or, X is a super key.
- Or, A is a prime attribute.

For example: In the functional dependency -
exercise_id → exercise_name, exercise_start, exercise_end, scoring_policy_id, retries_allowed, num_of_questions, correct_answer_points,Incorrect_answer_penalty, difficulty_level_min, difficulty_level_max, Exercise_mode

exercise_id is a superkey, so this functional dependency is in 3NF.

By checking all of the functional dependencies in our relational model, our relational model is determined to be in 3NF.

IV. Non-DBMS Constraints and Assumptions:

A. Constraints can not be implemented using DBMS:
Due to the nature of DBMS, most of the view and action are role-authorized. Since DBMS does not know who take the action at the front-end, we need to restrict these from the front end.

1. Exercise visibility:
   a) Normal Students:
      (1) only students who are enrolled in the course by teaching staff can view homeworks
      (2) not available to those students outside of the timeframe of the start and end dates (only available when current_time > exercise_start AND current_time < exercise_end)
   b) Teaching Staffs: available all the time
2. Overwrite Exercises Access:
   a) Instructor/Professor: have access to view and edit the question bank
   b) Normal Students: None
   c) TA: no access to question bank or to overwrite exercises created by instructor
3. A professor and a TA (teaching staff) can also create a exercise in different mode
   a) Find pre-exist exercises by exercise_id with same topic_id, duplicate with new exercise_id, then reset other attributes (e.g. exercise_start, exercise_end, retries_allowed, etc), and change exercise_name.

    b) Edit exercies: they can add and remove questions from existed exericse_id
4. Exercise generation mode: Random or Adaptive
    a) Adaptive mode - dynamically selected by the system based on performance
        (1) If answer is wrong, next question difficulty_level decrease by 1, till 1.
        (2) If answer is correct, next question difficulty_level increase by 1, till 6.
5. Course Access
    a) Instructors:
        (1) For courses taken by instructor: basic infos and modify permissions.
        (2) For courses not taken by instructors: only basic infos.
    b) TA: can only view basic infos of courses they working with.
    c) Normal students: can only view basic infos of courses they have been enrolled.
6. Add Course Access
    a) Only available for instructor
    b) The course_start and course_end should have the format of "yyyy/mm/dd"
7. Enroll Student Access:
    a) Only available for instructor/TA who work with that course
    b) Student can only been enrolled into a course with the same level with their role, e.g. Undergrad cannot been enrolled in Grad course
8. Drop student Access only available for instructor/TA who work with that course
9. Add TA Access
    a) Available for user when adding a new course
    b) Available for instructor who work with that course
10. Past Homework Access
    a) Student can view their grades or/and their attempts on the past homework when they finish an attempt of their homework
    b) Student can view the explanation/hints in their report:
        (1) If question is incorrectly answered and current_time < exercise_end, show question hints in student report
        (2) If current_time > exercise_end, show detailed_explanation in student report, even exercise has not been attempted
    c) Student can re-attempt if:
        (1) num_or_tries < retries_allowed
        (2) current_time > exercise_start AND current_time < exercise_end
11. Instructor can add questions to the bank by:
    a) The question text should include “<?>” to indicate the placeholder for parameter which means that the question being inserted would be a parameterized one.
    b) The maximum number of “<?>” should be only 5.

c) If none of the “<?>” is included in the question text than the question is a concrete question type.

12. Scoring policy:
    a) The calculation of each scoring policy (latest attempt, maximum score, average score) is implemented on application based since it requires the developer’s knowledge to interpret the proper equation to use.

B. Assumptions:
1. A question bank can be reused or share with another class (for example: class that are graduate and undergraduate level sharing questions/assignments from CSC522 and CSC422 classes)
2. A professor have the access and overwrite the question bank
3. A TA can exist without the course (an instructor can hire TA(s) specifically to the course outside the pool of available TAs)
4. For adaptive exercise, the question bank need to have sufficient questions under the topic for a adaptive exercise for each difficulty level. Due to the nature of changing difficulty, if the total number of question is not sufficient, the test itself will not be able to achieve the desired result.

V. Sample Queries on our database

A. Retrieval SQL queries - used to find specific information
   a. Find students who did not take exercise 1.
   b. Find students who scored the maximum score on the first attempt for exercise 1.
   c. Find students who scored the maximum score on the first attempt for any exercise.
   d. Retrieve all attempts for exercise 1 for Student 5.

```
a. SELECT USER_ID
   FROM student s
   WHERE not exists
    ( SELECT *
   FROM submits s1
   WHERE s.user_id =s1.user_id
   AND s1.user_id=1);

b. CREATE OR REPLACE View student_scores as
   SELECT sum(answer_score) as total, user_id
   FROM submits where attempt = 1 and exercise_id=1
   GROUP by user_id;
   SELECT user_id
```

```
    From student_scores
    Where total = (select max(total) from student_scores);
```

c.
```
CREATE OR REPLACE View student_scores as
SELECT sum(answer_score) AStotal, user_id
FROM submits WHERE attempt = 2
GROUP BY user_id;

SELECT user_id
FROM student_scores
WHERE total = (select max(total) FROM student_scores);
```

d.
```
SELECT sum(answer_score) AS total, attempt
FROM submits
WHERE user_id='jmick'
GROUP by attempt;
```

B. Reporting Queries . – used to find more general information

a. For each student, show total score for each exercise and average score across all exercises.
b. For each exercise and question, show the maximum and minimum score.
c. For all exercises to date, show the average number of attempts .

a.
```
SELECT sum(answer_score) AS tot, avg(answer_score) AS avg, exercise_id
FROM submits
WHERE user_id='jmick'
GROUP BY exercise_id;
```

b.
```
SELECT MAX(answer_score) , MIN(answer_score), exercise_id
FROM submits
GROUP BY exercise_id;
SELECT MAX(answer_score) , MIN(answer_score), question_id
FROM submits
GROUP BY question_id;
```

c.
```
SELECT avg(attempt)
FROM submits;
```

C. The following queries will be run on Demo day so make sure your application can produce the appropriate results.

a. Find names of all students of CSC540 that attempted Hw1 but did not attend Hw2.
b. Give list of students whose score increased on second attempt
c. List all courses and number of students enrolled
d. Show a report of all homework and attempts for all students enrolled in CSC540

a.
```
SELECT DISTINCT  user_id
FROM  submits s
 WHERE exists ( select * from submits s1
                Where s.user_id=s1.user_id and exercise_id=1)
 AND NOT exists (select * from submits s2
                Where s.user_id =s2.user_id and exercise_id=2);
```

b.
```
CREATE or REPLACE view look as
SELECT sum(answer_score) as total, attempt, exercise_id,user_id
FROM submits
GROUP BY attempt,exercise_id,user_id;
SELECT look.user_id
FROM look as v1 WHERE attempt = 1 AND v.total < (SELECT  u.total FROM v1
AS u WHERE attempt=2 AND v.user_id=u.user_id);
```

c.
```
SELECT count(USER_ID),course_id
FROM COURSE_STUDENT
GROUP BY course_id
```

d.
```
SELECT sum(answer_score) as total, attempt, exercise_id, user_id
FROM submits
WHERE course_id='CSC540'
GROUP BY attempt,exercise_id,user_id;
```