C2102-2310 Group Project

Part 1: Database Design with ERD

Project Objective

The objective of this team project is for you to apply what you have learned in class to design and develop a database application using PostgreSQL. The project is to be done in teams of four students. The project is organized into three parts
following four tasks:

1) Database Design with ERD
2) Translation of ERD to Relational Database Schema
3) Application development: Stored Functions/Procedures & Triggers

In Part 1, your task is to design a database by creating an Entity-Relationship Diagram (ERD) based on the given requirement analysis. This requirement analysis describes the overall functionality of the target application and details what data needs to be stored in the database, what relationships and constraints need to be managed and how the data is to be used by the application.

Requirement Analysis

Overview

Singapore Merlion University of Science and Engineering (MUSE) wants to build (design, implement, and deploy) a new online quiz platform called Squiz, catering to two distinct sets of users: students and educators. Educators generate questions (along with their answers) and quizzes. Squiz initially only supports multiple-choice questions (MCQ) and multiple-response questions (MRQ). Educators create groups, assign students to groups, and assign quizzes they have created to groups of students they have created. Squiz automatically grades the quizzes.

Squiz manages educator and student information, groups, questions, answers, quizzes, and quiz submissions.

Your task is to design and implement the Squiz online quiz platform database using PostgreSQL. You will create the entity-relationship diagram of the database, translate the diagram into a logical relational schema, write the corresponding SQL data definition language SQL code with all the constraints it can capture, and write the necessary stored functions, procedures, and triggers to implement other requirements and constraints outlined in the application description as effectively as possible.

Detailed Application Description

Both **students** and **educators** log in to Squiz using their student numbers (10-character strings) and university staff number (5-character strings), respectively. The database records their first name and last name. They have a unique email address. Squiz also records the last time a student was active on the platform.

Educators generate questions (along with their answers) and quizzes. Squiz initially only supports multiple-choice questions (MCQ) and multiple-response questions (MRQ).

A **question** has a unique identifier, a mandatory statement, and an optional description to provide comments and details. Each question is either a multiple-choice question (MCQ) or a multiple-response question (MRQ). The type of question (MCQ or MRQ) and its creator, the educator who created the question, are stored. A question is either public or private, where all creators can use public questions to create quizzes, while private questions can only be used by their creator.

The creator of a question assigns zero or more **tags** to it. A tag is a simple keyword (e.g., "SQL", "join", "DBMS") indicating to which topic a question belongs. The set of available tags is predefined, and educators choose from this set. For the sake of simplicity, educators are not allowed to create new tags.

For each question, the creator can define a list of **answers**. Each answer is uniquely associated with a question, i.e., an answer like "Yes" is not shared among multiple questions. An answer cannot exist without a question. Answers also have an order. However, this order of answers is the same across all the quizzes in which the question is used. The creator of a question can change the order of the answers.

A question does not need an answer to be created, but it can only be used in a quiz when it has a valid list of answers. Both MCQs and MRQs must feature at least two answers to be valid. While an MCQ must have exactly one correct answer, an MRQ must have one or several correct answers (possibly all available answers are valid) to be valid. The database of Squiz stores explicitly whether a question is valid or not as an attribute. The value of this attribute needs to be checked and updated if the list of answers for that question changes.

A question may be included in one or more **quizzes**. However, a question may have not been used in any quiz yet. The same question cannot be used twice in the same quiz.

An educator can create quizzes using her own questions, as well as any public questions.

A **quiz** is a list of questions. A quiz may initially be empty. The questions of a quiz are totally ordered. The creator of the quiz defines and modifies the order of the questions in a quiz. Each quiz has a unique identifier and a mandatory but not necessarily unique name. For each quiz, the database of Squiz stores its creator, the time interval during which the quiz is available (i.e., from/until) and whether it is published or not. The creator can also specify if the quiz is public, the maximum number of attempts that students can take the quiz (once by default), and mark if the quiz is mandatory (e.g., for assessment) or optional (e.g., for practice) – a student can choose not to take a mandatory quiz.

A question in a quiz is associated with zero or more points. For example, the same question might yield 3 points in one quiz but only 2 points in another quiz. A question in a quiz can be mandatory or optional.

By default, a quiz is unpublished, is indefinitely available from when it was created, and has no time limit. If a quiz allows multiple attempts, a student can take this quiz up to this maximum allowed number. However, we only store the submitted answers for the last attempt. A quiz is available only if it is published and has at least one question.

For each quiz, the database of Squiz stores the total number of points as an attribute, which is the sum of the points associated for the quiz to the questions in the quiz – that is, when adding or removing a question to or from a quiz, or when changing the points for a question, the total number of points for that quiz needs to be updated.

Educators create groups, assign students to groups, and assign quizzes they have created to groups of students they have created.

A **group** has a unique code, a mandatory and unique name, and an optional description. A student belongs to no or multiple groups. A student can only take public quizzes and quizzes assigned to a group to which she belongs. A group can consist of zero to many students.

A quiz can be assigned to zero or more groups; The quiz is only available to students of those groups. This includes that a student can be in multiple groups the quiz is assigned to. It is only important that the student is in at least one of those groups to be able to see and take the quiz.

The database of Squiz also stores students' **submissions.** A submission is the set of answers to the questions in a quiz that a student has taken. Since Squiz only supports MCQs and MRQs, a submitted answer reflects a question's possible answers the student has selected. A student taking a quiz must answer at least all mandatory questions, if any.

Deadline & Deliverable

The deadline for Part 1 is **Sep 17, 23.59**. This is a hard deadline, and late submissions will not be accepted and graded.

Each team is to upload a pdf file named `teamNNN.pdf` where `NNN` is the three digit team number according to your project group number. You should add leading zeroes to your team number (e.g., `team005.pdf`). Submit your pdf file on Canvas assignment named "Project - Part 1: ERD". Only one file is to be submitted per group. If there are multiple submissions for a group, the submission with the lower mark will be chosen.

The submitted pdf file must have a font size of at least 12 point for normal text and consists of the following:

- Project team number & names of all team members (on the first page)
- ERD of your application using the notation introduced in the lecture.
  - If your ERD is too large (i.e., spanning more than a page), you may want to include a single-page simplified ERD (i.e., with non-key attributes omitted) before presenting the detailed ERD.
  - Your ERD cannot be too small. The font in the ER diagram must be readable.
- Justification for any non-trivial design decisions made. This may include the use of hierarchical or ternary relationships, or generally distinct alternatives you have considered.
- List of up to 5 of the application's constraints that are not captured by your ERD. In general, your ERD should:
  - Capture as many of the application's constraints as possible
  - Not capture constraints not specified in the application's constraints

  Any constraints that can be captured but did not (or not specified in specification but erroneously added) may be penalized except for reasonable real-world constraints.

Grading

Part 1 of the project is worth 6 marks in total. The following grading scheme will be used. The details of the grading scheme are hidden.

- (2 Mark) Validity of Entity-Relationship Diagram (ERD): The ERD diagram should not contain invalid constructs.
- (2 Marks) Data Requirements: The ERD should capture all the data needed by the application's requirement.
- (2 Marks) Constraint Requirements: The ERD should capture as many of the application's requirements. The ERD should not add constraints not specified in the application's requirements whenever possible.

SOLUTION: List of up to 5 of the application's constraints that are not captured by your ERD.

- Order of answers for a question (position attributes must be meaningful sequences, e.g.: 1, 2, 3, 4 – assuming 4 answers)
- Order of questions in a quiz (position attributes must be meaningful sequences, e.g.: 1, 2, …, 10 – assuming 10 questions)
- Correct value of the "valid" attribute for a question which depends on the number of answers (at least 2) and the number of correct answers (1 for MCQ, 1..n for MRQ).
- Total number of points is derived from the sum of all points of the quiz questions.
- Generally: many NOT NULL constraints + all default values
- A submission must contain answers for all mandatory questions
- **Depending on modeling:** A user is either a student or a creator but not both.

Assumptions

For this project, we make a series of simplifying assumptions to limit the required complexity of the database design and implementation efforts. In more detail, these assumptions are:

- We assume no hardware or software errors such as browser crashes or power failures on the client and server side. If a student encounters, say, a browser crash while taking a quiz, he or she has to start the quiz again from scratch – otherwise <name> would need to keep and update the most recent intermediate states of a quiz.

- We only consider multiple-choice questions (MCQ) and multiple-response (MRQ) questions. They are very similar and allow for the same type of answer. Also supporting Fill-in-the-Blank questions, matching questions and other types of questions would probably blow up the complexity.

- We assume that the Web interface of <name> only allows students to submit completed quizzes if all mandatory questions have been answered. This means, we do not have to consider partial quiz submissions.

Database Schema

Table **users**

Attributes, constraints and additional details

- A user has a unique id, and a mandatory and unique email address
- A user must specify a first name; the last name is optional
- The table keeps track of the time a user was last active
- A quiz has a time interval when it is published; by default, a quiz is published the moment it was created – using `NOW()` – and gets never unpublished (indicated by `NULL` for attribute `published_until`)

```
CREATE TABLE users (
 id INTEGER PRIMARY KEY,
 email TEXT NOT NULL,
 first_name TEXT NOT NULL,
 last_name TEXT,
 last_active TIMESTAMP NOT NULL
 CONSTRAINT u_email UNIQUE (email)
);
```

Table **groups**

Groups are concepts to organize users into subsets to specify which quizzes are available to which groups. For example, a group can be all users in a course (e.g., CS2102-2310)

Attributes, constraints and additional details

- A group has a unique id, a mandatory title and an optional description
- We also keep track when the group was created and by whom

```
CREATE TABLE groups (
 id INTEGER PRIMARY KEY,
 name TEXT NOT NULL,
 description TEXT,
 creator INTEGER NOT NULL,
 created_at TIMESTAMP NOT NULL DEFAULT NOW(),
 CONSTRAINT fk_creator FOREIGN KEY (creator) REFERENCES users (id),
);
```

There probably needs to be a default entry in this table to represent a group "public", to specify later that a quiz can be open to anyone.

Table **group_assignments**

A user can belong to multiple groups, and a group typically consists of multiple. We therefore need a table to represent the assignments of users or groups.

Attributes, constraints and additional details

- We keep track when a user was added to a group (this helps to identify which quizzes the user has not seen since they were available to a group before him/her joining).

```
CREATE TABLE group_assignments (
 id INTEGER PRIMARY KEY,
 group_id INTEGER,
 user_id INTEGER
 created_at TIMESTAMP NOT NULL DEFAULT NOW(),
 PRIMARY KEY (group_id, user_id).
 CONSTRAINT fk_group FOREIGN KEY (group_id) REFERENCES groups (id),
 CONSTRAINT fk_user FOREIGN KEY (user_id) REFERENCES users (id)
);
```

Table **quizzes**

Attributes, constraints and additional details

- A quiz has a unique id, a mandatory title and an optional description
- A quiz has a creator which must be a registered user
- A quiz has a time interval when it is published; by default, a quiz it's published the moment it was created – using `NOW()` – and gets never unpublished (indicated by `NULL` for attribute `available_until`)

```
CREATE TABLE quiz (
 id INTEGER PRIMARY KEY,
 creator TEXT,
 title TEXT NOT NULL,
 description TEXT,
 published BOOLEAN NOT NULL DEFAULT FALSE,
 available_from TIMESTAMP NOT NULL DEFAULT NOW()
 available_until TIMESTAMP,
 total_points INTEGER NOT NULL DEFAULT 0,
 CONSTRAINT fk_creator FOREIGN KEY (creator) REFERENCES users (id),
 CHECK (available_until IS NULL OR
        available_until >= available_until);
);
```

Table **quiz_assignments**

A quiz can be available to different groups, and a group can have access to different quizzes. So we need a table to store this access/permission information.

Attributes, constraints and additional details

- We keep track when a user was added to a group

```
CREATE TABLE quiz_assignments (
 quiz_id INTEGER,
 group_id INTEGER,
 created_at TIMESTAMP NOT NULL DEFAULT NOW(),
 PRIMARY KEY (quiz_id, group_id).
 CONSTRAINT fk_quiz FOREIGN KEY (quiz_id) REFERENCES quizzes (id),
 CONSTRAINT fk_group FOREIGN KEY (group_id) REFERENCES groups (id)
);
```

If a quiz id does not occur in this table, the corresponding quiz is not available to anyone.

Table **questions**

Attributes, constraints and additional details

- A question has a unique id, a mandatory title and an optional description
- A question has a type: MCQ or MRQ.

```
CREATE TABLE questions (
 id INTEGER PRIMARY KEY,
 question TEXT NOT NULL,
 description TEXT,
 type TEXT NOT NULL CHECK(type IN ('MCQ', 'MRQ'))
 is_public BOOLEAN DEFAULT FALSE,
 is_valid BOOLEAN DEFAULT FALSE,
);
```

Table **answers**

Attributes, constraints and additional details

- An answer existentially depends to the referenced question
- An answer has a position to enforce an order among all answers for a question
-

```
CREATE TABLE answers (
 id INTEGER PRIMARY KEY
 question_id INTEGER,
 answer_text TEXT,
 is_true BOOLEAN NOT NULL,
 position INTEGER NOT NULL,
 CONSTRAINT u_answer_pos UNIQUE (question_id, position)
);
```

Table **quiz_questions**

Since a quiz can contain several questions and a question can be used in different quizzes, we need some table that connections quizzes to questions

Attributes, constraints and additional details

- Each question of a quiz has a position in the quiz to imply an order.
- Each question in a quiz is associated with a number of points.
- A question in a quiz can be mandatory or not (default: mandatory; if optional would be the default, it would be too easy to forget to turn on)

```
CREATE TABLE quiz_questions (
 id INTEGER PRIMARY KEY,
 quiz INTEGER NOT NULL,
 question INTEGER NOT NULL,
 points INTEGER NOT NULL CHECK(points >= 0),
 position INTEGER NOT NULL,
 mandatory BOOLEAN NOT NULL DEFAULT TRUE,
 CONSTRAINT u_question UNIQUE (quiz, question),
 CONSTRAINT fk_quiz FOREIGN KEY (quiz) REFERENCES quizzes (id),
 CONSTRAINT fk_question FOREIGN KEY (question)
                        REFERENCES questions (id),

);
```

Table **quiz_submissions**

This table contains all the answers for all the questions in a quiz.

Attributes, constraints and additional details

-

```
CREATE TABLE quiz_submissions (
 id INTEGER PRIMARY KEY.
 user INTEGER,
 quiz INTEGER
 submitted_at TIMESTAMP NOT NULL DEFAULT NOW(),
 CONSTRAINT u_submission UNIQUE (user, quiz, submitted_at),
 CONSTRAINT fk_user FOREIGN KEY (user) REFERENCES users (id),
 CONSTRAINT fk_quiz FOREIGN KEY (quiz) REFERENCES quizzes (id),
);
```

Table **quiz_submitted_abswers**

Attributes, constraints and additional details

-

```
CREATE TABLE quiz_submitted_abswers (
 submission INTEGER NOT NULL,
 question INTEGER NOT NULL,
 answer INTEGER NOT NULL,
 CONSTRAINT fk_submission FOREIGN KEY (submission)
                          REFERENCES quiz_submissions (id),
 CONSTRAINT fk_question FOREIGN KEY (question)
                        REFERENCES questions (id),
 CONSTRAINT fk_answer FOREIGN KEY (answer)
                      REFERENCES answers (id),
);
```

Table **tags**

Tags are useful to search for quizzes and questions as well as for the generation of random quizzes based on a topic. A topic is defined by a set of tags

Attributes, constraints and additional details

- Technically, we just need a unique string

```
CREATE TABLE tags (
 tag TEXT PRIMARY KEY
);
```

Table question_tags

Assign tags to individual questions and quizzes.

```
CREATE TABLE question_tags (
 question INTEGER
 tag TEXT,
 PRIMARY KEY (question, tag),
 CONSTRAINT fk_qestion FOREIGN KEY (question)
                      REFERENCES questions (id),
 CONSTRAINT fk_tag FOREIGN KEY (tag) REFERENCES tags (tag)
);
```

PSM

Required Functions & Procedures

Moving Questions or Answers

The creator of a quiz might want to move a question or an answer. We assume that this is done by dragging and dropping the question or answer using a Web interface. After dropping, the Web interface provides the new location (e.g., 1, 2, 3, 4 for a MCQ with 4 answers).

- move_question(quiz_id, question_id, new_location)
- move_answer(quiz_id, question_id, answer_id, new_location)

Invalid locations need to be handle appropriately (if there are only 4 answers, 5 can never be a valid value for new_location)

Changing access to quizzes

When changing the access of quizzes to groups (insert/updates/deletes on quiz_access), only the creator of a quiz can change the access.

Evaluate Quiz

For a given quiz and given user, compute the achieved number of points. We can keep it simple by only distinguishing correct (full marks) and incorrect (zero marks). In principle, MRQs – and other types of questions in the future (e.g., Fill-in-the-Blank) – could yield partial marks. It might be good to have a auxiliary function for each question type to check if a submitted answer is correct, e.g.:

- `check_mcq_answer()`
- `check_mrq_answer()`
- …

Generate Training Quiz

For a chosen topic and a specified set of questions, we want to create a random quiz. The selection of questions from a topic is random but uniform: questions new to the user and questions the user got wrong are more likely to be selected.

If a topic is defined by a set of tags or keywords one boolean parameter could specify if a question must feature AT LEAST ONE or ALL tags.

We probably also want/need to ensure that not all questions are being used to generate training quizzes. For example, a lecturer creating a quiz for marking might not want this question popping up anywhere else.

Required Triggers

Update Total Points of a Quiz

When adding or deleting a question to or from a quiz, or when changing the number of points of a single question, we automatically update the `total_points` attribute for this quiz.

Update "valid" Attribute for Questions

Any time the set of (correct) answers for a question changes, it needs to be checked if the question has become valid or invalid