Advanced Web development project

Evaluation

Advanced Web development

The Web development is evaluated through a project done by teams of three members. Each member is evaluated individually. The features must be shared out between the members, to have the same weight.

The requirements are described below, with a DB model and a trigger specification.

Ajax (and then REST Web services) will be used when useful. User friendlliness (not decoration) is taken in account. The deliverable is a zip file including:

1. The stored procedure used to reset the database in order to test the application.
2. The PL/SQL code for the triggers.
3. The whole application source code (PHP, JavaScript, CSS).
4. A readme explaining:
    1. How to test the application: login/password of each testing account + home page URL.
    2. The design choices and their reason.
5. The excell file of the acceptance test results.

Project features

We express them as user stories with their corresponding acceptance tests.

As a *trainer*, I want an application to evaluate semi-automatically *students* skills about SQL (more precisely the SELECT statement, which is the difficult part of the language). A classic quiz will not do. So the idea is the following: the application will record queries written by students, execute them on the target database, and compare their results to those of the corresponding correct query, previously recorded by the trainer.

That will work if the data in the database cover enough cases to make a wrong query produce wrong results. This condition is not easy to ensure. Therefore the application will display and compare, for each student and each question, the student and trainer query and result. The trainer will then validate or invalidate each answer, the application recording that and displaying finally the student note.

This semi-automatic way will save time to the trainer, as long as the students number is reasonable. However, if we want it to scale, we must make sure the data will discriminate safely correct and wrong queries. This will require more work on data, much refinement and test. This is out of our scope.

The application could be used as well as a training tool: students may perform tests outside the scope of an evaluation, to test themselves their SQL skills. The application will then display them their query and a right one already recorded, with their results. If both give the same result, they can check their query is similar, or discover an alternative writing. If their result differs, they can study their mistake.

Main user stories

We express here the main user stories to develop first.

1. As a *student*, I can record online an SQL evaluation, so that the trainer will correct it semi-automatically.
    1. I can not access the evaluation before the date and time it has been scheduled for my *class*.
       (class is for example « Msc fall 2016 »)
    2. Its page displays the database diagram which the questions are based on, and an explanation of this diagram.

3. My evaluation begins when I click on a start button.
4. The evaluation appears then in my evaluation list, with the status « ongoing ».
5. The remaining time till the evaluation end is displayed in real time during the evaluation, since it has started.
6. For each question, I can input the corresponding SQL query.
7. Each answer is recorded when I validate it.
8. I can terminate my evaluation. I must confirm when doing that.
9. The evaluation terminates automatically when the time is elapsed.
10. I can no more validate any answer when my evaluation is terminated.
11. The evaluation status changes to « done » in the evaluation list when it is terminated.

2. As a *trainer*, I can list my evaluations recently passed by the students, so that I can see or validate their notes.
    1. Each evaluation displays its related class, date, begining and ending time, and if have corrected all its student copies by myself.
    2. I can mark an evaluation as completed if, and only if, I have corrected all its copies.
    3. For each evaluation, I can display the subject (list of questions) and the notes of all students whom I have corrected the copy.
    4. Each evaluation displays the average note, if at least a copy has been corrected, the number of copies and the number of corrected ones.
    5. I can display the copy of each student.
3. As a *trainer*, I can correct online a student evaluation,so that I am helped by the application.
    1. The complete set of questions is displayed.
    2. Each question displays the question text, at left the student query and below its result, at right the trainer query and below its result.
    3. The expected and actual results are displayed as HTML tables, with the column names as the first row of headers (th).
    4. If the class evaluation is not completed, I can validate or invalidate any question in the student copy (we call that « complete the question »), or I can delay its correction.
    5. When I validate or invalidate a question, or delay its correction, the change is immediately recorded (if I reload the page, its state remains the same).
    6. The application shows if an answer is considered correct by the application and by myself.
    7. A question is considered as currently correct if I have validated it, or if I have not yet completed it and if the application considers it as correct.
    8. The application displays the total student note, according to the correctness rule above.
    9. The student copy is marked as completed when and only when I have completed all of its questions.
    10. A click on a question displays the page correcting this question, all students included (see next feature).
    11. (Suggestion for the appearance)
        1. A question considered correct by the application has a faded green background.
        2. If considered incorrect, its background is faded red.
        3. It is full green when validated by the trainer.
        4. It is full red when invalidated by the trainer.
4. As a *trainer*, I can correct online a question, all students included, in order to be equitable for all students and concentrated on the question.
    1. The question data is displayed at left at a fixed position.
    2. It includes the question text, its expected answer, expected result, number of correct copies on this question, number of answered copies, number of copies.
    3. At right, the complete set of answers for this question is displayed.
    4. For each copy, the student name, the answer, the result, the proposed and final status are displayed.
    5. If the class evaluation is not completed, I can validate or invalidate the answer of any student, or I can delay its correction.
    6. When I validate or invalidate a question, or delay its correction, the change is immediately recorded (if I reload the page, its state remains the same).
    7. A click on the student name displays the whole student copy.
    8. The same appearance rules as in the student evaluation apply.
5. As a *student*, I can ask to join a class, so that I can access its evaluations.
    1. I must be connected (so I must have signed up if I am not yet a registered user).
    2. I must confirm my request.
    3. The class must be still open.
    4. The class appears in my asked classes list.

6. As a *trainer*, I can validate for a class the requests to be a member of it, so that they are controlled.
    1. The list of asked and validated members displays the first and last name, the email, the joining datetime, and if the member is validated.
    2. The validated, invalidated and pending requests have an associated style (eg. a green, red and transparent background).
    3. I can validate or invalidate each request, or postpone its validation.
    4. The validation is immediately recorded.
    5. I can close the class (no member may afterwards join it).

Remarks

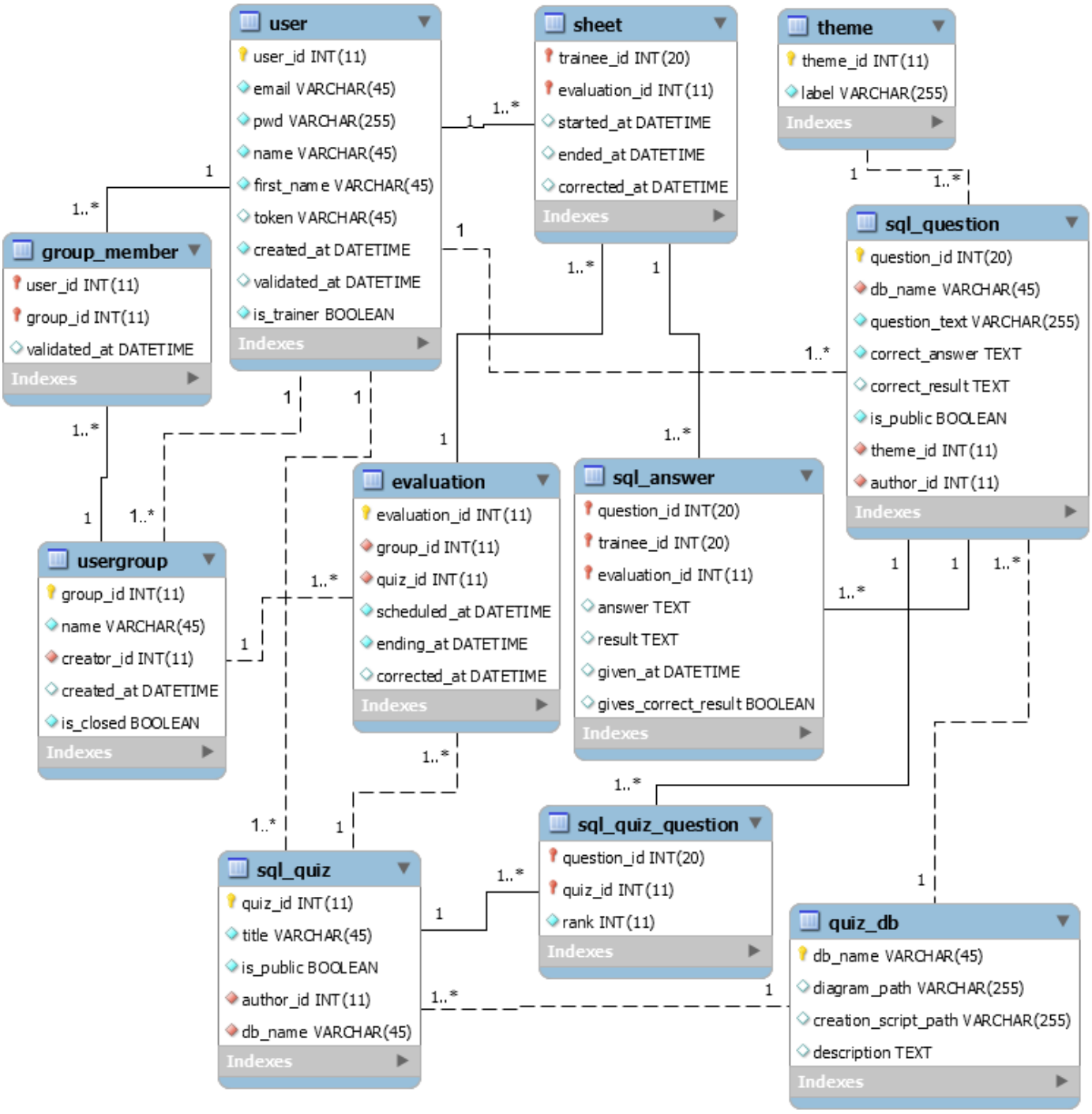
Of course, **sign up and log in are required features**. You will only adapt the version found in the PHP examples or on the Web.

The creation of evaluations and quizes is out of scope, even though they will be useful, as they are less prior than the features detailed above. Of course, we need at least a quiz and and evaluation, based on this quiz, to test the application. It is strongly suggested to set it up through a stored procedure reseting the database content. You can make quizes from the bank SQL exercices, whose answers and database you already have.

Please be aware that you will have to deal with two databases: `sql_skills`, defined below, and the target database for executing the requests provided as answers.

Database

The database to use is modeled by the following relational model.



Explanations

1. Students and trainers are users. Both have the same details (fields), but different relationships. We choose to store them in the same `user` table, in order to simplify the log in process.
2. We must then add `BEFORE INSERT` (and `UPDATE`) triggers on `sql_quiz`, `sql_question` and `usergroup` to ensure that their `author_id` or `creator_id` refers to a trainer, not a simple user.
3. A student may join a group (feature 5), if this is not closed. We call `usergroup` such a class. Feature 5 express that a student may join *many* user groups, so the `n-m` relationship `group_member`.
4. Nothing in the requirements precise if a question belongs to 1 quiz or many. This should be covered by additional requirements about the quiz management and authoring. We choose here a flexible solution: a `n-m` relationship.
5. An `evaluation` is given to a `usergroup` on a `sql_quiz`. It begins at `scheduled_at` and end at `ending_at`. We store in `completed_at` the date/time at which the trainer has completed its correction.
6. A student (trainee) may run n evaluations, which are run by m students, and a student may run only once the same evaluation. So we have a `n-m` relationship `sheet`.

7. In the same way, `sql_answer` is the n-m relationship between `sheet` and `sql_question`.
8. We have of course the constraint: `evaluation.scheduled_at` ≤ `sheet.started_at` ≤ `sheet.ended_at` ≤ `evaluation.ending_at`.
9. A `sql_answer` stores the SQL query written by the student in `answer`, and the result of this query executed in the target database in `result`. When the trainer sets if it is a correct result or not, the `gives_correct_result` is set.
10. Both `sheet` and `evaluation` store in `corrected_at` the date/time at which the trainer has marked it as completely corrected.
    1. A sheet cannot be marked as corrected till all its answers have a valued `gives_correct_result`.
    2. An evaluation cannot be marked as corrected till all its sheets are corrected.
11. `sql_question` and `sql_quiz` have a `is_public` field indicating if it may be used for training. In this case, it should no more be used in evaluations.

Triggers

We must code the following triggers:

1. Before `insert` or `update` on `user`: trim and capitalize `name` and `first_name`, trim `email`.
2. Before insert on `group_member`: raise a `3001` error code if the group is closed.
3. Before `insert` or `update` on `sql_quiz`, `sql_question` and `usergroup`: raise a `3002` error if `author_id` does not refer to a trainer, and `3003` for the `creator_id` (we use different codes to customize the associated error message).
4. After `insert` on `evaluation`, we insert the `sheet` of all the related class students. This way, if a student does not run the evaluation, a sheet is nethertheless created and the note computed to 0.
5. In the same way, after insert on `sheet`, we insert all the related `sql_answer` (with a `NULL` `answer` and `result`). This way, the application only updates the answers, it has not to insert first and then to update: the process (and the PHP code) is simplified.
6. Before insert on `evaluation`, raise a `3004` error if `scheduled_at` ≥ `ending_at`.
   Normally, this shoud nor require a trigger and be only a `CHECK` declarative integrity constraint, but unfortunately, mySQL does not check them.
7. Before update on sheet: `3005` if we are not in time (start or end not in the scheduled dates) and `3006` if `started_at > ended_at` (when they are set).
8. Before insert on `sql_answer`, raise a `3007` if it is not the time (before `sheet.scheduled_at` or after `sheet.ending_at`).