M1 Software Engineering

Advanced Databases for Software Engineering

Mélanie Marques

Louis Prud'homme

EFREI Paris – 2020

Table of contents

Delivery description

The delivery folder should be organized as follows :

```
delivery/
├── Marques_Prud'homme_diagram.png
├── Marques_Prud'homme_full_script.sql
├── Marques_Prud'homme.pdf
└── separated_scripts.zip
```

- The `full_script` contains every line of SQL and PL/SQL made for the project. It is meant to be executed once, and it will do everything from the database creation, to its testing.
- The `diagram` is the E/R diagram of our database. Architectural and design choices motivations can be found in the comments of our SQL scripts.
- The `separated_scripts` archive contains the pieces of which is made the `full_script`, of which here is the required execution order :
    1. dropping script
    2. creation script
    3. function creation script
    4. procedure creation script
    5. procedure creation script
    6. insertion script
    7. testing script

Introduction

The goal of this project is to develop an electronic document management system to archive all the internship and apprenticeship reports for EFREI.

Today, students must email their reports to tutors (businesses and academics). Apprentice students submit their reports on Moodle. Students can submit intermediate documents but only the final report is saved.

In the solution that we propose, the system allows an easy search of documents, and makes them accessible.

This research can be done by keyword, by category, title, etc. It allows the report to be submitted before a specified deadline. This report only becomes readable for students and teachers after validation by the tutors. In addition, only people with access to MyEfrei can access the report after validation.

Naming conventions

Variables

Naming	Meaning	Example
Starts with `l`	Local variable	`ln_id_user`
Starts with `p`	Parameter variable	`pn_id_user`
Second char is `n`	Variable of type number	`ln_id_user`
Second char is `v`	Variable of type varchar	`pv_keyword`
Second char is `c`	Explicit cursor	`lc_reports`
Second char is `d`	Variable of type date	`ld_deadline_report`
Second char is `e`	Declared exception	`le_no_record_found`

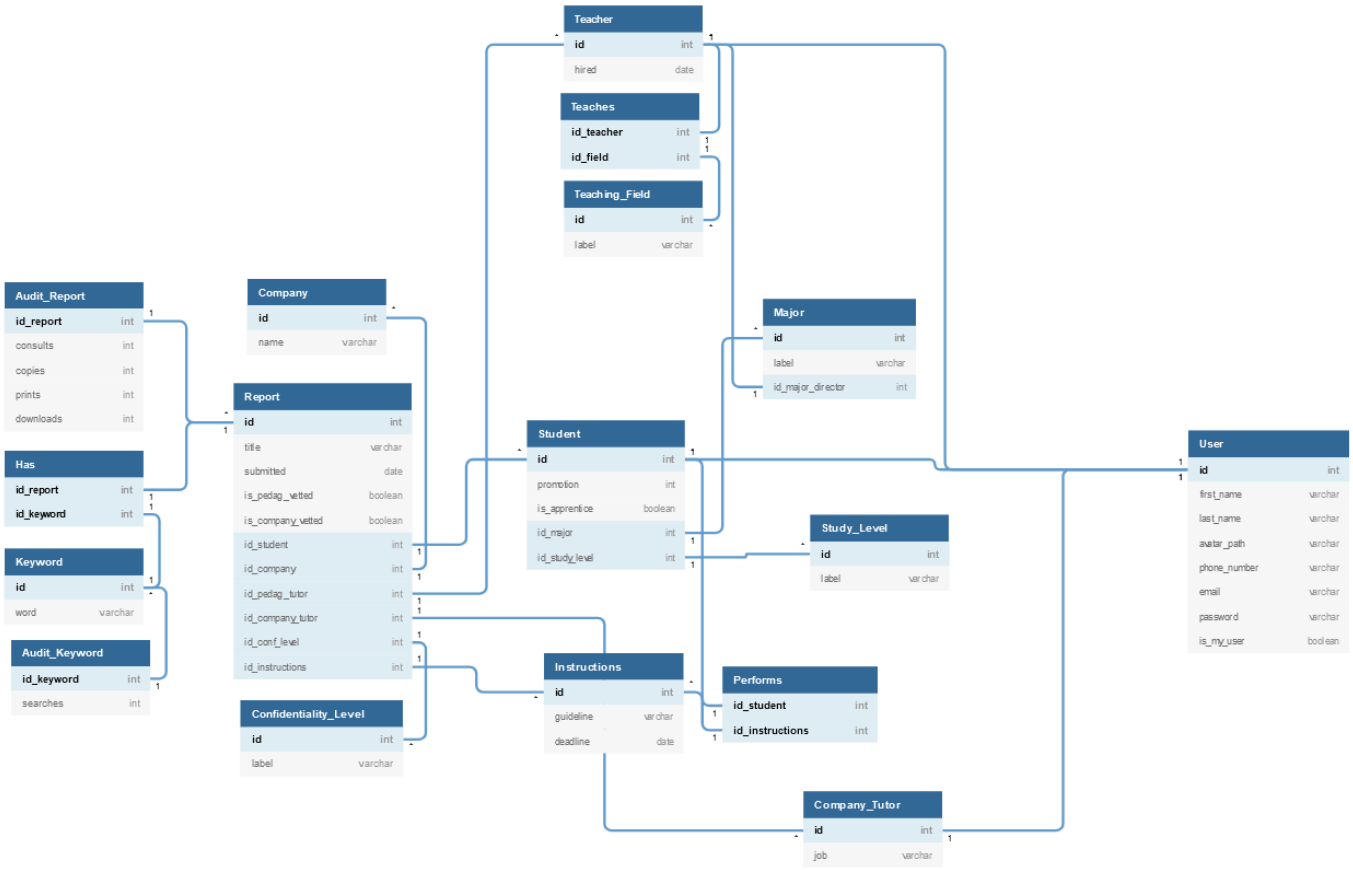
Objects

Naming	Meaning	Example
Starts with `tab`	Table	`tab_student`
Starts with `adt`	Audit table	`adt_keyword`
Starts with `rel`	Relation table	`rel_performs`
Starts with `fun`	Function	`fun_is_allowed`
Starts with `prc`	Procedure	`prc_report_consult`
Starts with `trg`	Trigger	`trg_report_validation`

Error Management

Error codes	Description
-20002	The report is late, the deadline is over.
-20003	The hired date can not be in the future.
-20004	Keyword not found
-20005	Expected at least one keyword for this report.
-20006	Inconsistency between the promotion of the student and his group
-20010	No records were found for either the report id or student id, or both.
-20011	Confidentiality settings disable this action.
-20012	The report has not been validated, action aborted.
-20013	User must be a user of My Efrei.

E/R diagram



You can find the full-sized diagram in the files of the project.

Implementation of important issues

Data consistency

To ensure the consistency of the data, we have undertaken to set up controls at the time of data insertion.

User emails

Using a `CHECK` when creating the User table, we verify that the user's email is of the form `example@example.fr`.

Phone numbers

Using a `CHECK CONSTRAINT` on the table User, we verify that the user phone number respect the pattern of a classic phone number : `+33699999999` or `0699999999`.

Passwords strength

Using a `CHECK CONSTRAINT` on the table User, we check that the password is strong, i.e. if it has at least one capital letter, a lowercase letter, a special character, a number and its length is greater than or equal to 8.

Teacher hired date not in the future

As the `SYSDATE` cannot be used in a `CHECK CONSTRAINT`, we created a trigger : `TRG_TEACHER_HIRED_DATE` to ensure that the teacher hired date is lower than the `SYSDATE`. If this condition isn't respected, it raises an exception `-20003`.

Each report must have minimum one keyword

We added a trigger `TRG_REPORT_VALIDATION` in order to check that every final report has at least one keyword.

Indeed, when a report is declared final, that is to say when it has been vetted by the company tutor and the pedagogic tutor, the trigger counts the number of keywords for the report. If this number is lower than 1 it raises an exception `-20005`.

Consistency between a student's group and their promotion

The trigger `TRG_STUDENT_PROMOTION` checks if the promotion of the student matches its study level. To achieve this, it gets the current year and month. If the month is before september, we take the previous year as reference. Then we calculate the difference between calculated gratuating year and state graduating year.

If the result is inconsistent, it raises an exception `-20006`.

We also take in account that some students dropped, were kicked or graduated out of school ; for such cases, we have created an «OUT» study level.

Search

The solution we suggest allows easy report search by :

Keyword

The function `FUN_REPORTS_BY_KEYWORD` allows to obtain a cursor on all the reports tagged with the provided keyword.

This function works as follows :

1. It gets the id of the provided keyword
2. It opens the cursor and point it on all reports related to the specified keyword
3. It reports where found, it update the keyword audit table.
4. It returns the cursor

In addition, this function is marked as `PRAGMA`, it allows it to be autonomous and thus we can test it in a `SELECT` (since we execute a DML operation on the audit table, and this cannot be done in a query).

Furthermore, if there isn't any keyword with this label, the function raises an exception `-20004`.

Category (internship or apprentices) thanks to a select query :

```
SELECT id
  FROM report
  WHERE id_student IN (
    SELECT DISTINCT id
      FROM student
      WHERE is_apprentice = 1);
```

Other searches are possible (such as: by student name, title *etc*...) thanks to simple `SELECT` queries.

Report

All students have to submit intermediate documents but only the final report will be saved

When a report is declared as final, i.e when it has been vetted by the company tutor and the pedagogic tutor, the trigger `TRG_REPORT_VALIDATION` will call the procedure `PRC_DELETE_INTERMEDIARY_REPORTS` in order to delete intermediary reports.

Submit the report before a deadline

After inserting or updating of the field submitted on `TAB_REPORT`, the trigger `TRG_REPORT_DEADLINE` checks if the report submission date is greater than the deadline. If so, an exception `-20002` is raised.

Report statistics

Most wanted keywords

The function `FUN_MOST_WANTED_KEYWORDS` returns a cursor pointing on the first *n* most wanted keywords, *n* being the parameter given to the function.

Most wanted reports

The function `FUN_MOST_WANTED_REPORTS` returns a cursor pointing on the first *n* most wanted reports, *n* being the parameter given to the function.

Number of consults, copies, prints, downloads for each report

The table `ADT_REPORT`, thanks to simple `SELECT` queries, allows to get the number of consultation, copies, prints and download for each report.

Confidentiality

Implementation of report confidentiality

Thanks to the function `FUN_IS_ALLOWED`, we can manage the confidentiality of the reports. Indeed, this function plays a central role in the user's interaction with reports. It takes the `ID`s of a user and a report, as well as an operation's confidentiality level as an input.

Then, it performs a serie of checks :

1. Checks if both the report and the user exist
2. Checks if the operation is permitted for this report (printing, for instance, is forbidden for level-2 confidentiality reports)
3. Checks if the user is also a My Efrei user or if he was involved in the making of the report (for company tutors, mainly)
4. Checks if the report has been validated or if he was involved in the making of the report (non-validated reports cannot accept incoming operations)

If any of those checks fails, the function raises an exception between `-20010` and `-20013`. Otherwise, it simply returns `1`.

This function is not directly used by the user, but rather a common denominator for the procedures detailed thereafter.

Download, copy or print a reportd

When a user wants to download, copy or print a report, check that the action requested are allowed by the level of confidentiality.

The procedures `PRC_REPORT_*` represent the ability of the user to interact with reports. Their are four of them, `CONSULT`, `COPY`, `DOWNLOAD` and `PRINT`.

Their name are pretty self-explanatory in what each procedure represents.

Besides, they are very few and slight differences between them ; they basically work in the exact same way.

- They call to `FUN_IS_ALLOWED` to know if the given user can perform the operation on the given report

- If `FUN_IS_ALLOWED` greenlights the request, the corresponding field in audit table `ADT_REPORT` is incremented by one on the record of the given report

In fact, `FUN_IS_ALLOWED` does all the heavy lifting for these procedures ; there is only two differences between all of them :

1. They all update different fields in the audit table `ADT_REPORT` (`prints` for `PRC_REPORT_PRINT`, *etc*)
2. They may have different confidentiality levels ; as per the requirements, we consider `COPY`, `DOWNLOAD` and `PRINT` as level-1 confidentiality operations (which can only be executed on a level-1 confidentiality report) and `CONSULT` to be a level-2 (execution up to level-2 report)

Problems encountered

Subject understanding

We had some hard times understanding the instructions, its needs and requirements.

Virtual machine : not an ideal environment

Working on a virtual machine considerably increases our working time. Indeed, we had to face many crashes, black screens, and bugs, sometimes resulting in data loss.

Moreover, even when it works the best it can, it is slow. So slow, in fact, that is it a better idea to code on an external IDE and then only test in the virtual machine. But even that is slow, given the tremendous input lag caused by the virtual nature of the machine.

At some point, we were so fed up with those lags we developed a bash scripts to automaticaly send and retrieve our SQL scripts.

Oracle errors management

Oracle error handling being quite imprecise, it was quite dificult to debug when our script had an error.

For instance, it wasn't uncommon to see PL/SQL blocks that would compile but crash at execution.

Need to commit

Since our SQL script is executed in one time, there are no checkpoints, no commits, until the end of the execution.

This provoked a handful of bugs, notably with PL/SQL blocks, that would compile and run without bugs, but also without doing anything. This was a very strange and weird bug, which we resolved by adding some commit instructions along the script.

Conclusion

This project allow us to face and response to new type of challenges. Indeed, it enabled us to participate to a concrete database project which goes from the base architecture to the scripts writing without forgetting the testing phase.