UCLouvain

# Moodle Learning Analytics: how to extract log data efficiently ?

***Feedback on the development of social flow plugins***

Isabelle Motte
isabelle.motte@uclouvain.be

November 16, 2025

submitted as an additional qualification
for obtaining the rank of expert computer scientist

## License notice (CC BY-SA)



## Contact

Isabelle Motte
LS/SGSI
Service général du système d'information (SGSI)
SGSI - Pythagore
L6.08.01
Place des Sciences 4
1348 Louvain-la-Neuve
Belgium

Email: isabelle.motte@uclouvain.be

# Table of Contents

# Introduction

In Belgian universities, the career development of IT specialists depends on a project based on individual development. While working in the team that maintains the Moodle online courses platform, I choose to work on a learning analytics project based on Moodle data. Such a project represents a big technical challenge because, in Moodle, user data is stored in a gigantic log table. This report explains how I proceeded to define my learning analytics project and presents my analysis and approach to develop such project in Moodle while considering carefully the performance challenge linked to database structure and queries optimization.

A lot of learning analytics projects are oriented on building various reports for teachers or for strategic leaders. My plan was to build a tool to help students. This project supposed to study some pedagogical references about student engagement to find inspiration. My findings summary is presented in the section 1 and this section also presents my final project: proposing a student “Social flow” view of most frequent actions performed by students to enforce students engagement.

Designing this student view supposes to develop several Moodle plugins and the section 2 therefore presents the Moodle code architecture and Moodle plugin anatomy.

The section 3 presents the analysis of existing Moodle learning analytics plugins and highlights an interesting learning analytics development, based on Thomas Dondorf’s Phd, that will be adapted to develop the social flow student view.

The next sections successively take up the different challenges of the social flow project:

1. The section 4 explains how the analytics data storing was performed;
2. The section 5 presents how the data where retrieved and stored in database tables to ensure optimized database queries;
3. The section 6 shows how some non pertinent data where excluded from social flow while creating new tables for deadlines and implementing some new database queries tunings.

All database queries have been tested and optimized based on a copy of UCLouvain Moodle huge database (146 millions lines) but no live test with users was implemented. An evaluation form is presented in the section 7 to measure student interest for this social flow view and to evaluate the interface usability and performance. But the plugins have not been tested with student because I had a professional reorientation.

This reports ends with a conclusion that gives the main ideas to take into account to make an efficient data treatment while performing a Moodle analytics project.

# 1 The plan: developing a tool to support students' academic success in Moodle

The following subsections propose a summary of the main ideas to retain from articles [1, 2, 3, 4] on student engagement. These references have been suggested by an education researcher, Mikaël De Clercq. This analysis leads to my project definition: proposing a student "Social flow" view on most frequent actions performed by students. Mock-ups of the final student view are proposed in the last subsection.

## 1.1 Clarifying the concept of student engagement

The article [1] proposes a major meta-analysis which lists the different definitions of the concept of student engagement.

Engagement can be broken down into three dimensions which are not independent and which I propose to summarize as follows:

- behavioral engagement: the student commits to the tasks expected of him/her, he/she respects the rules and deadlines; this is physical engagement, participation;
- cognitive engagement: the student mobilizes the intellectual energy necessary for learning; this commitment is linked to the perceived value of the task, to understanding, to the ability to self-regulate and to set goals; it's the engagement of the mind, involvement;
- emotional engagement: the student shows an interest in their studies and feel part of a learning community; engagement of the heart, fulfillment.

## 1.2 Main idea: focus on positive feedback

It sounds like an obvious recommendation, but given that the majority of learning analytics projects seem to focus on indicators of the risk of failure, it's important to point this out. After all, study [3] (admittedly full of limitations) shows a negative correlation between this type of indicator and success... So it's better to be cautious and consider student reporting upstream, with a view to guidance.

Another important point to note is that it is not the quantity of interaction or feedback that has an impact on success, but its quality. And the quality of feedback represents a major challenge for learning analytics approaches, because in learning management systems, it is easy to extract quantitative data, but it is more difficult to extract qualitative data.

On the teaching side, however, it is interesting to identify students at risk of failure, so as to provide targeted feedback, which we hope will be of high quality.

## 1.3 Relevant features already in Moodle: progress and grades

A good way of providing guidance is to indicate how the student is progressing along the course. But this presupposes that the teacher has clearly marked out the course. This approach is implemented natively in Moodle, using the activity completion and progress bar. Teachers can also use the completion report to contact students who have not completed an activity.

Badges can also be used to mark important stages of progress.

The timeline block on the Moodle dashboard shows upcoming deadlines in the various courses, and the course list on the dashboard can show the progress wheel if the administrators allow it.

It is not surprising that grades are a good indicator of success, but this is also a feature that is directly integrated into Moodle, via the gradebook. However, it would be interesting to produce an indicator that would make it easy to distinguish between the certification part of the grade and the formative part. And the student should be able to know what proportion of the final mark his partial mark represents. All of this can be configured in the gradebook, but it has to be said that the Moodle gradebook is really not easy to analyze ...

## 1.4 Other indicators that don't seem very promising: time indicators

Let's also look at the indicators that seem to have a mixed impact on success: temporal indicators such as the time taken to access the course (very difficult to evaluate technically) or the frequency of access to the course.

In fact, it's not the student clicks that are important for their success but what they do between 2 clicks ...

## 1.5 The most interesting project to implement: a notification system on the behavior of the cohort of students on a course

The above references and linked articles enabled me to find a promising indicator that really comes under the heading of student data processing and that goes beyond the traces already present in Moodle. The idea is to set up a "social flow" notification system that alerts the student of the most frequent actions of all the participants, and highlights the actions that the student has not carried out and that the other members have completed.

I find this project interesting from a pedagogical point of view for several reasons:

- This type of indicator was found to be highly relevant for stimulating engagement in a very detailed study [5] (several other indicators were tested) but on a small scale (53 participants).
- These authors take their analysis further in [6, 7] (the second with a larger cohort of learners) where they also indicate that

  - the “social flow” indicator seems to have a greater impact on learner engagement than the tagging proposed by the teacher;
  - the impact is also greater for the students who need it most (those who have difficulty defining goals);
  - it’s not the most connected learners who get the most out of the tool; everyone benefits from it;
  - in their experience, which was based on a MOOC on learning analytics, it was the girls who benefited most, which they explained by their lack of confidence in scientific subjects.
- this type of indicator is aimed at providing positive feedback and at guiding students upstream;
- this type of indicator touches on several components of engagement: behavioral (indicates what needs to be done) and emotional (indicates what the community is doing) and perhaps also cognitive (indirectly, it helps to set objectives);
- this type of indicator is aimed primarily at students and involves processing their personal data in an anonymised way, so it is less intrusive in terms of privacy;
- this type of indicator does not require any prior marking of the course by the teacher.

Such a view could also be used by teachers to provide them with an overall picture of what is happening in the course, always anonymised. This could enable them to adjust their interactions with students, whether targeted or not. In this way, we can hope to contribute to better quality feedback, which is a driving force for success...

## 1.6 Effective reporting is customized reporting on demand

Learning analytics approaches must allow students a certain amount of freedom, otherwise their ability to self-evaluate and be autonomous will be compromised. Depending on their motivation for a course, they should be able to choose whether or not to activate the tracking and notification system. For the “social flow”project, they could also choose the reference period, that can vary depending on their relationship to deadlines. The UCLouvain dissertation [8] already tried to build learning analytics indicators for students, based on Moodle actions. And it is interesting to retain the idea to filter the activities based on consultation and/or participation actions. This distinction will be clarified in the section 3.1 presenting the existing Moodle learning analytics plugins in more details.

## 1.7 Mock-up of the social flow block

The social flow view is supposed to be presented in the student’s dashboard, before to enter a course. It indicates the most frequent actions performed in all the student’s courses, order by frequency and associated to an indicator that tells if the student did or did not perform the action. It indicates to student what peers are doing and tells also which actions they performed or not.

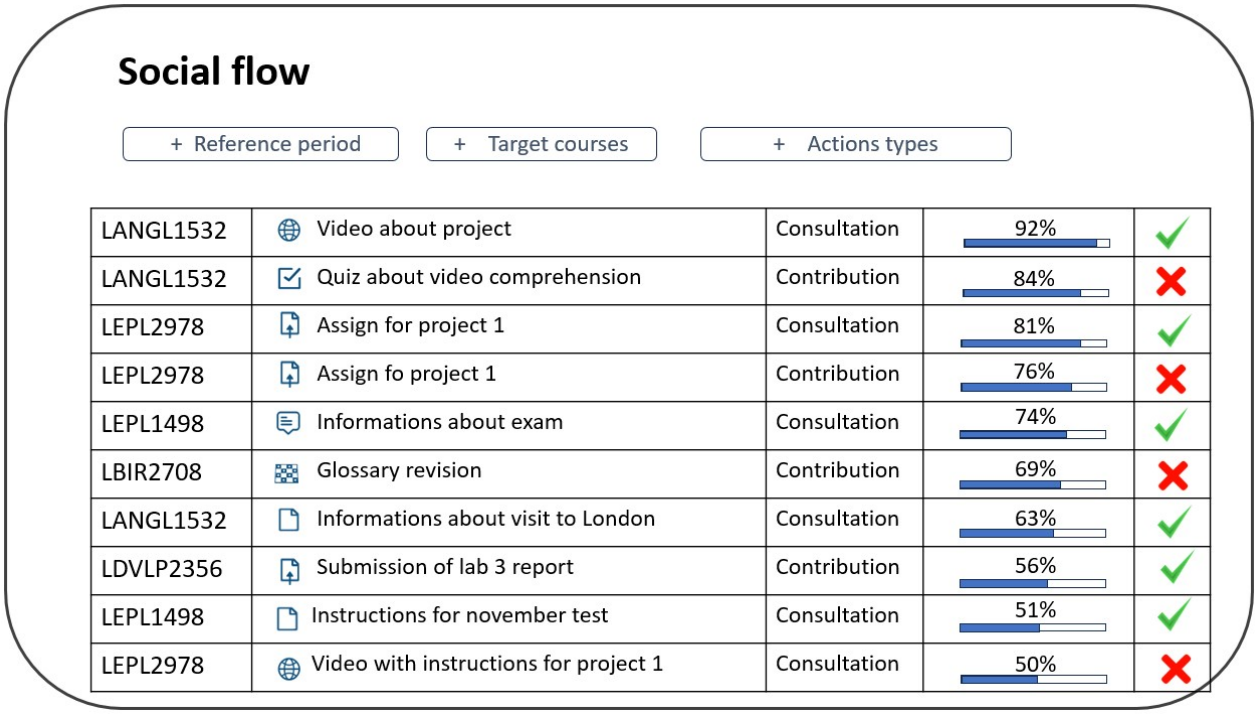



**Figure 1: Global view of social flow block**

The figure 1 gives a global view of the social flow rendering and the five table columns contain the following information :

- the course code where the action was performed;
- the name of the course module concerned, with the module logo to remind the course homepage presentation of activities;
- the action type for this action (“Contribution” or “Consultation”);
- the global frequency of student having performed this action (the number of student having performed the action divided by the number of participants in the course);
- a green/red icon to indicate if current student did or did not perform the action.

The block will be designed to present 3 filters :

- The reference period filter is shown on the figure 2 and enables the student to choose the reference period from 1 day to 2 weeks, to customize the data according to their relationship to deadlines,
- The target courses filter is shown on the figure 3 and makes it possible for student to select courses data to integrate to the social flow view, depending on theirs expectations for each course;
- The actions type filter is shown on the figure 4 make it possible to choose to reference only contribution actions, participation actions or both.

The filters configuration will be stored as user preferences so that they are registered and unchanged when student leaves Moodle and comes back.

Implementing such a view in Moodle supposes to understand the Moodle code structure

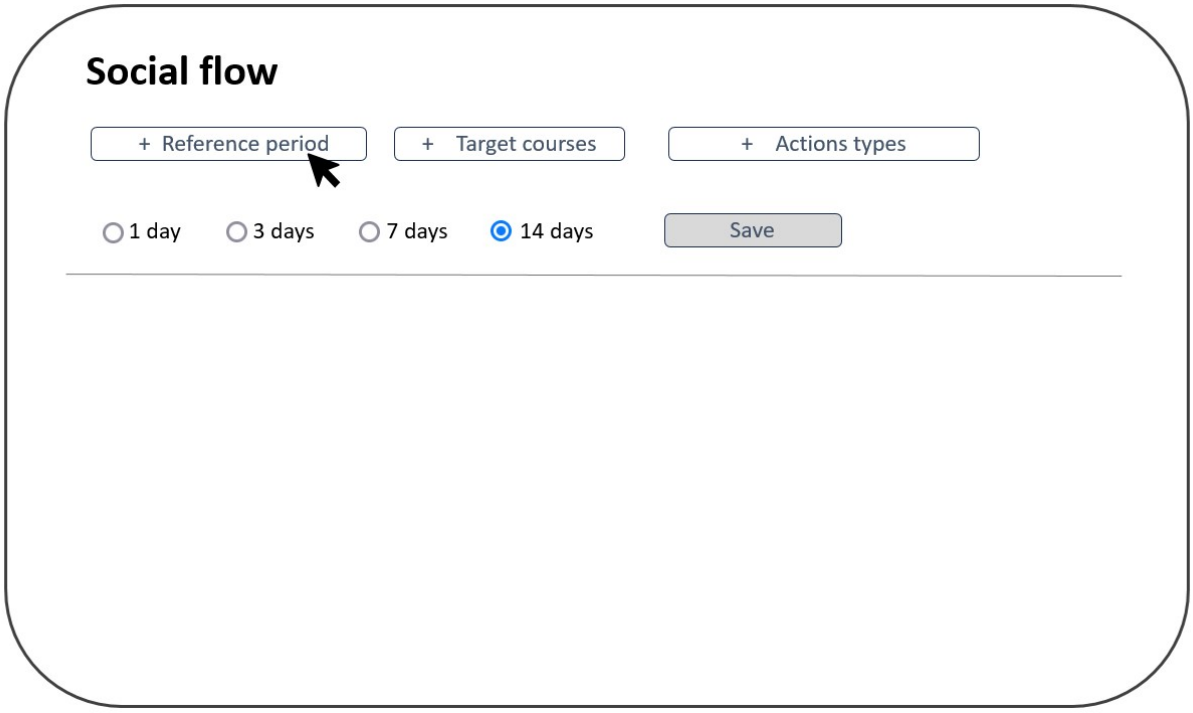



**Figure 2: Reference period filter on social flow block**

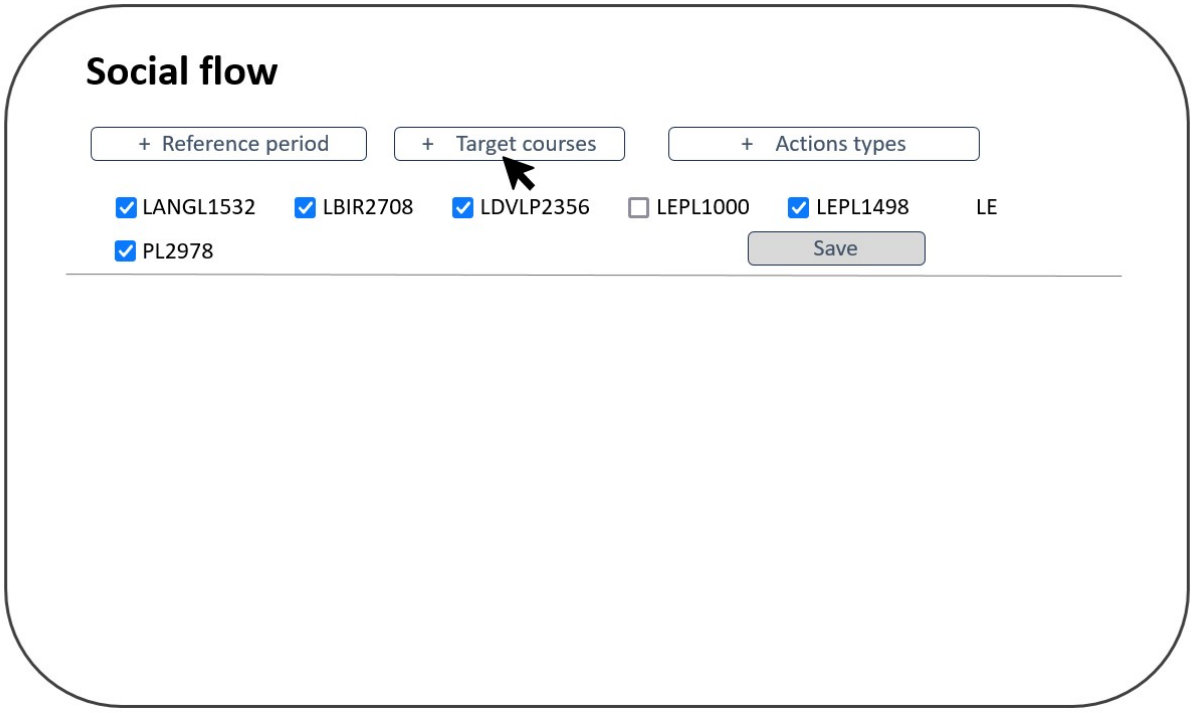



**Figure 3: Target courses filter on social flow block**

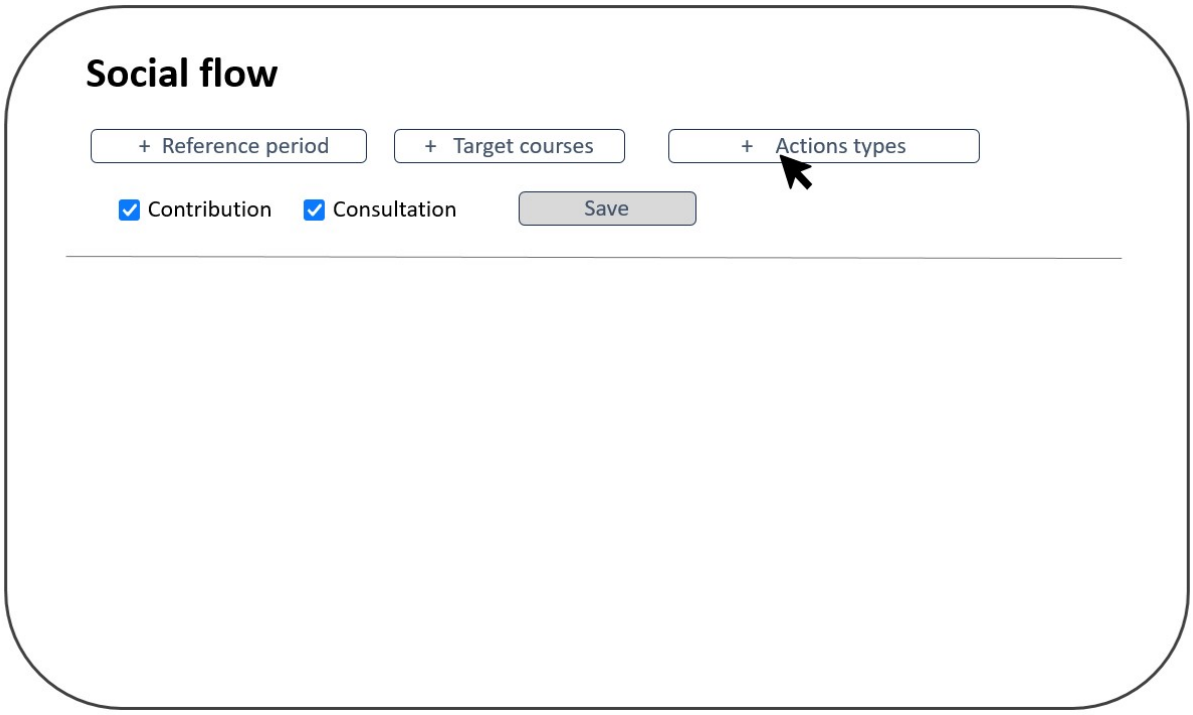



**Figure 4: Actions type filter on social flow block**

and a Moodle plugin anatomy. These elements are presented in the next section, that may be skipped if you already are used to develop Moodle plugins.

# 2 The context : extending the Moodle LMS

Before diving into Moodle learning analytics plugins analysis and development, let us take a tour of the Moodle code. Moodle is an open source Learning Management System (LMS) developed in PHP and supporting a database that may be MySQL, MariaDB, PostgreSQL, MS SQL Server or Oracle. The “Moodle” name is the acronym for “Modular Object-Oriented Dynamic Learning Environment” and the tool’s code structure makes it possible to extend the functionalities with plugins.

The moodle code is shared on Github and you may find the code of the latest Moodle version directly through this link:
https://github.com/moodle/moodle/tree/main/public
Do not hesitate to consult the different Github folders while reading this section to have a good understanding.

## 2.1 The Moodle architecture

Moodle is organized based on a kernel that includes API and basic classes in `/lib` folder and main files and folders are the following :

- `/lib/datalib.php` includes essential functions to access the database;
- `/lib/moodlelib.php` integrates global functions (authentication, role management, text format, ...);
- `/lib/weblib.php` gathers functions for HTML output and handling strings;
- `/lib/classes` includes PHP classes implementing object oriented kernel functionalities;
- `/lib/db` contains kernel database definition;
- `/lib/filestorage` includes API for files management;
- `/lib/form` gathers API for creating forms;

Notice that Moodle's codebase has been undergoing a major restructuring since version 5.1. This restructuring aims to improve security, modernize the architecture, and separate public-facing code from internal logic. The new `/public` directory becomes the only web-accessible area, while the root-level code is kept outside the web server's reach :

- `/lib` contains Moodle's internal engine: core APIs, subsystems, and sensitive logic that must never be accessible via HTTP;
- `/public/lib` contains only web-facing scripts and assets needed to serve pages and handle browser requests.

This separation helps to reduce the attack surface, aligns Moodle with modern framework practices, and makes the platform easier to maintain and extend. In the rest of this report, all folders and files described are located inside the `/public` directory of the new file architecture.

Main functionalities are organized in specific folders and the most important are described below :

- `/course` contains code for creating, displaying, updating and deleting courses;
- `/user` gathers the code for managing users (creating, editing, and deleting accounts) and displaying user profiles;
- `/admin` includes code related to site administration tasks like managing settings, managing plugins, defining roles or performing system upgrades;
- `/grade` gathers the code for Moodle's grading system, it supports grades calculation and recording and also displays the gradebook;
- `/lang` contains language packs and translation files that provide all the text strings used throughout Moodle's interface.

Moodle supports more than 50 plugin types, and each plugin type corresponds to a specific folder or subfolder. Here are the most useful folders for plugin development :

- `/blocks` gathers the code that present views on contents on dashboard or on the course homepage; blocks may be linked to existing activity plugins like `/blocks/feedback` that presents the list of all `/mod/feedback` activities;
- `/enrol` includes the code that manages enrollment methods (manual enrollment, self-enrollment and external synchronization) and API for enrollment, with sub folders for each enrollment plugin type;

- `/mod` integrates the code of each course resource or activity module, like file, folder, url, assignement, quiz or forum;
- `/local` is the place for custom plugins developed specifically for a given Moodle site, allowing site-specific features or integrations;
- `/question` includes the code of all quiz question types;
- `/report` contains the code of various reports, build for teacher or for administrators, that provide insights and data summaries about Moodle usage, user activities or system performance;
- `/theme` gathers the code of the different Moodle themes to have customization of the Moodle global presentation.

After this global overview of the Moodle code structure, we now may study the general construction of a Moodle plugin.

## 2.2 Moodle plugin anatomy

Every plugin lives in its own folder, named according to its type and shortname (for example, `mod/quiz`, `blocks/html`).This folder contains all the code, language files, and metadata for that plugin and is structured as follows :

```
myplugin/
   version.php
   settings.php
   lib.php
   renderer.php
   styles.css
   amd/
       build/
            myscript.min.js
       src/
            myscript.js
   backup/
   classes/
       event/
       privacy/
       task/
   cli/
   db/
       access.php
       events.php
       install.xml
       install.php
       task.php
       upgrade.php
   lang/
```

```
    en/
        myplugin.php
pix/
templates/
tests/
```

The most frequent files and folders of a Moodle plugin are described below but notice that few of them are mandatory :

- `version.php` (mandatory) defines the plugin's version number, required Moodle version, dependencies, and maturity level;
- `settings.php` defines the plugin-specific admin settings visible in the Moodle admin interface;
- `lib.php` contains the functions that integrate the plugin with Moodle's core APIs (hooks);
- `renderer.php` contains the rendering logic for HTML output;
- `styles.css` or `styles.scss` are plugin-specific CSS styles;
- `amd` folder contains the javascript necessary for the plugin; the native version is stored in `src` subfolder while `build` subfolder contains a minimified version that may integrate Moodle's caching system; several script may be defined and have to be declined in the two folders;
- the `backup` folder contains the code to support backup and restore of plugin data if pertinent;
- the `classes` folder stores the PHP classes that handle the plugin's business logic, data processing and API integration; plugins often contain classes defining triggered events (in `event` folder), classes defining information about personnal data storage (in `privacy` folder) and when the plugin needs specific scheduled tasks, they are defined in the `task` folder;
- the `cli` folder may be created to place admin scripts that may help to use the plugin; they are often used to import data or to clean data;
- the `db` folder contains all the database-related definitions and setup linked to the plugin;
  - if the plugin needs new tables, they are defined in the `install.xml` file,
  - `install.php` script may be used if some data need to be stored in the tables at the install,
  - `upgrade.php` may be defined if a new plugin version requires some table structure modifications,
  - `access.php` may define capabilities linked to the plugin's actions, that will integrate the long list of capabilities in Moodle roles definition,
  - `events.php` defines events this plugin triggers in the logs,
  - and `task.php` defines the scheduled tasks for the plugin (linked to a PHP class defined in `/classes/task/`);
- `lang` folder (mandatory) stores all text strings for the plugin, enabling multi-language support;
- `pix` folder stores necessary images and icons;

- `templates` folder stores Mustache template files for complex HTML rendering;
- `tests` folder implements PHPUnit and Behat test files if wanted.

## 2.3 Running cron tasks on a development environment

A lot of actions in Moodle are carried out by scheduled tasks performed by the server cron service. On a development server, you hopefully will not configure cron service, but you can still run cron tasks.

The main Moodle core tasks (like restoring a course for example) may be ran manually using this command line in a terminal from the Moodle root folder:

```
php admin\cli\cron.php
```

And a specific plugin cron task may be ran while performing this command from the moodle root folder:

```
php admin/cli/scheduled_task.php --execute="\myplugin\task\mytask"
```

Note that it is also possible to run individual scheduled tasks in Moodle administration interface, via 'Run now' links on the scheduled tasks page. For this purpose, the option 'Allow 'Run now' for scheduled tasks' (tool_task/enablerunnow) in Site administration / Security / Site security settings should be enabled AND the option 'Path to PHP CLI' (pathtophp) in Site administration / Server / System paths should be set.

## 2.4 Tip for small javascript implementation in Moodle

Note that to implement javascript functions in Moodle, the official documentation advises installing Nodejs and grunt task runner. But for small javascript functions, you may also paste your js code in an online minifier service, like Minifier.org, and paste the minimified version into a file with the right name and place it into the right folder.

When testing your javascript scripts in Moodle, it is possible to disable javascript cache while adding one line in the `config.php` file:

```
$CFG->cachejs = false;
```

as explained in the official Moodle developer documentation. This enables you to work on the original file in `amd/src` without having to minimify your code to test it in on Moodle.

Now you better understand how a Moodle plugin is built, the next section analysis the existing Moodle learning analytics plugins.

# 3 Preliminary step : analyzing existing learning analytics plugins

This section proposes to study existing analytics projects in Moodle to find an approach to implement the social flow project. This section was inspired by a discussion with Luiggi Sansonetti that presented a session on additional plugins for reporting at the 2022 French MoodleMOOT in Caen.

## 3.1 The FollowUp UCLouvain plugin

The FollowUp plugin was developed as part of William Bonesire's dissertation [8]. It is a course report plugin giving indications on student's global activities in the course. The report is structured in 3 parts (see figure 5) :

- Grades : indicates the global student grade in the course and the number of graded activities;
- Participation : presents the number of consultation actions and contribution actions in the course;
- Progression : displays the number of achieved activities and late activities.

These indicators are associated to the median of the group values to enable student to compare its actions to the median student. And a visual color indicator is built to clearly highlight most important weaknesses.

While supervising this dissertation, I already dived into the Moodle log data information. Moodle stores every user actions in the default table `logstore_standard_log` and these events are described in Moodle events API documentation. In particular, the "Participation" indicators where retrieved while fetching data directly from the gigantic `logstore_standard_log` table. The approach proved to be problematic in terms of performance but some ideas may be retains from the events analysis.

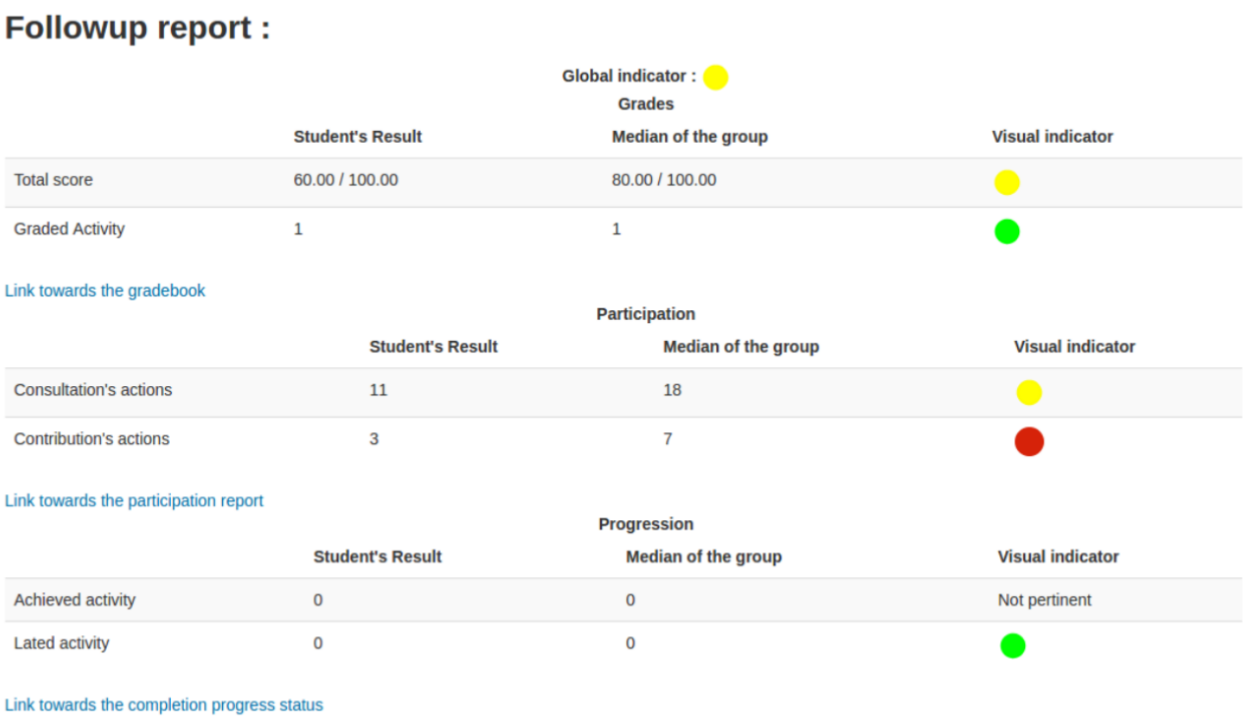
The FollowUp functional analysis pointed out that some events are for example linked to students consultation actions in a course :

- `mod_resource\event\course_module_viewed` is linked to a file consultation;
- `mod_forum\event\forum_viewed` corresponds to a forum consultation;
- `mod_choice\event\course_module_viewed` is associated to a choice activity view;
- `mod_url\event\course_module_viewed` means a link has been consulted.

While other events are linked to student contribution actions in a course, like these :

- `mod_forum\event\post_created` corresponds to a forum post creation;
- `mod_assign\event\submission_created` means a student submitted an assign;
- `mod_quiz\event\attempt_submitted` is associated to a quiz validation action.

These distinction in Moodle events types will be part of our social flow plugins design,



**Followup report :**

Global indicator :

Grades

| | Student's Result | Median of the group | Visual indicator |
|---|---|---|---|
| Total score | 60.00 / 100.00 | 80.00 / 100.00 | |
| Graded Activity | 1 | 1 | |

Link towards the gradebook

Participation

| | Student's Result | Median of the group | Visual indicator |
|---|---|---|---|
| Consultation's actions | 11 | 18 | |
| Contribution's actions | 3 | 7 | |

Link towards the participation report

Progression

| | Student's Result | Median of the group | Visual indicator |
|---|---|---|---|
| Achieved activity | 0 | 0 | Not pertinent |
| Lated activity | 0 | 0 | |

Link towards the completion progress status

**Figure 5: The FollowUp report**

while we will try to present our block view more has a view guiding the student's actions.

## 3.2 The native Inspire analytics plugin is dead

In 2017, the Moodle analytics project has been launched with the Inspire plugin that rapidly became a part of Moodle kernel. Several presentations recording may be found on Youtube about the analytics project like [9] and [10]. But since then nothing new happened.

The Inspire plugin implements next-generation learning analytics using machine learning backends that go beyond simple descriptive analytics to provide predictions of learner success (see figure 6), and ultimately diagnosis and prescriptions (advisements) to admins and teachers. But models and predictions are only visible to teachers and administrators.

Elizabeth Dalton communicated a great deal both technically and pedagogically about native learning analytics and is still listed as the moderator of the Moodle analytics forum. But, according to her curriculum vitae, she left the Moodle project in April 2020 to join Intelliboard, a company that offers a paid external analytics solution. The first version of the Intelliboard Moodle plugin was released in March 2022 and it must integrate Moodle analytics particularly well ... David Monllaó that contributed a lot to the initial Inspire plugin also left the project in January 2020. What's more, the history of Moodle's analytics folder on github doesn't show any activity that would indicate a dynamic development ...

Initiatives now seem to be driven by additional plugins that have nothing to do with the

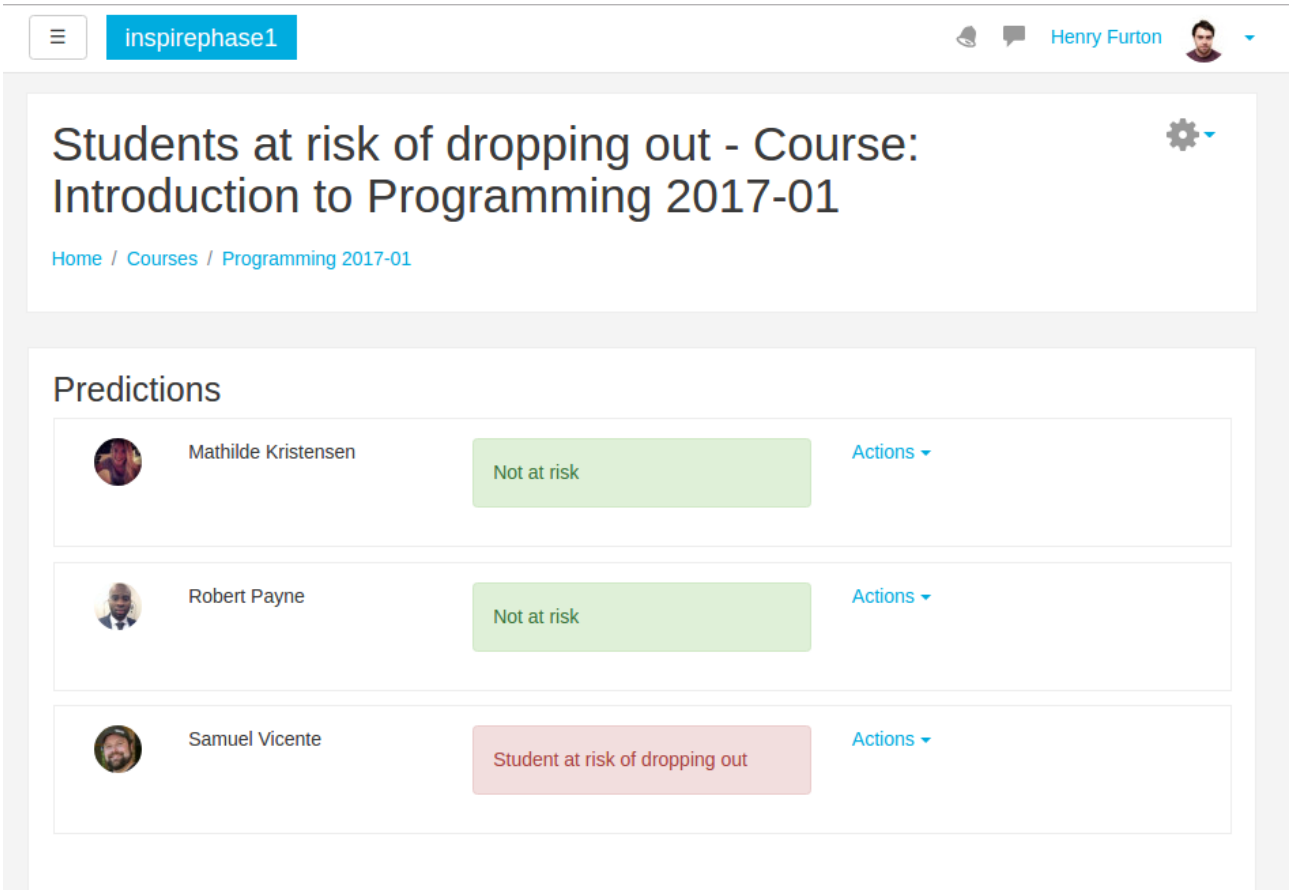



**Figure 6: View on teacher report on student at risk of dropping out integrated to Inspire plugin**

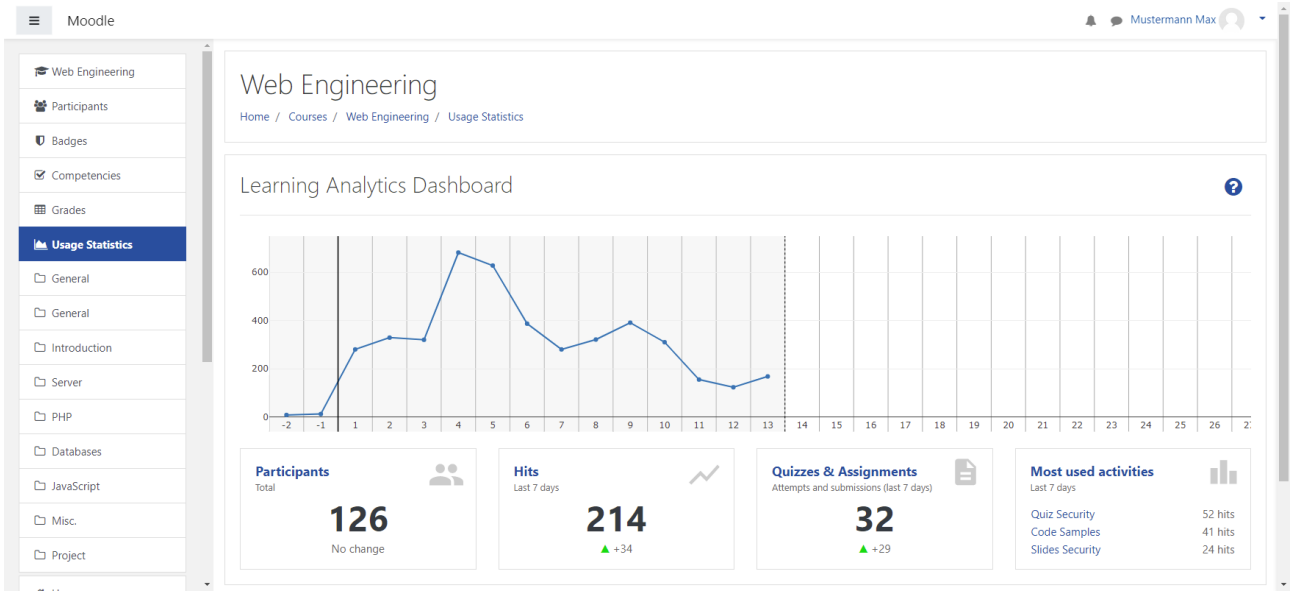



**Figure 7: View on the main report of Thomas Dordorf's learning analytics plugins**

native analysis models, such as the Intelliboard plugin, the Learnerscript plugin, the Learning Analytics plugins or the Engagement analytics plugins.

## 3.3 The powerful Learning Analytics plugins

The Learning Analytics plugins were developed by Thomas Dondorf at RWTH Aachen University in the context of his PhD dissertation [11]. This is a very interesting and detailed learning analytics project that presents an approach to extract data from the big Moodle log table in an efficient way. The plugins add a web analytics report to the course that integrates a view on student activities (similar to social flow view but restricted to a single course), a view on browsers and operating systems and a detailed view on actions in quiz and assignments (see global view at figure7).

### 3.3.1 First Learning Analytics plugin : a dedicated log plugin to store only the pertinent data

While official Moodle developer documentation about log plugin was non-existent, the dissertation [11] explains very well how to build such plugin, and gives recommendations to avoid data redundancy. The native Moodle logstore table `logstore_standard_log` counts about twenty fields where the most of them are redundant (see figure 8).

In comparison, the learning analytics plugin logstore stores data in two tables with as few redundancy as possible (see figure 9):

- `logstore_lanalytics_evtname` stores information about the generic events that may arise;
- `logstore_lanalytics_log` counts only 6 fields that store information about each oc-

| Column | Example | Redundant | Meaning |
|---|---|---|---|
| id | 1234567 | No | Primary Key |
| eventname | \mod_quiz\event\ attempt_viewed | No | Event name |
| component | mod_quiz | Yes (eventname) | Related component |
| action | viewed | Yes (eventname) | Corresponding action in verb form |
| target | attempt | Yes (eventname) | Target of action |
| objecttable | quiz_attempts | Yes (eventname) | Table containing data related to this event |
| objectid | 94698 | No | ID of the corresponding row in the table |
| crud | r | Yes (eventname) | Database action ("CRUD") |
| edulevel | 2 | Yes (eventname) | "Educational value" |
| contextid | 10879 | No | Context ID, often refers to the related course |
| contextlevel | 70 | Yes (contextid) | Context level constant |
| contextinstanceid | 5644 | Yes (contextid) | Instance that the context is referencing |
| userid | 20054 | No | ID of the user who triggered the event |
| courseid | 74 | Yes (contextid) | ID of course in which the event happened |
| relateduserid | NULL | No | ID of second user, in case another user was involved (like sending a message to someone) |
| anonymous | 0 | No | If the action was executed anonymously |
| other | a:1:{s:6:"quizid";s:3: "557";} | No | Additional meta data |
| timecreated | 1589189158 | No | Unix timestamp |
| origin | web | No | How the event was triggered. |

**Figure 8: Analysis of default log tables fields redundancy (extract from Thomas Dondorf's Phd page 93)**

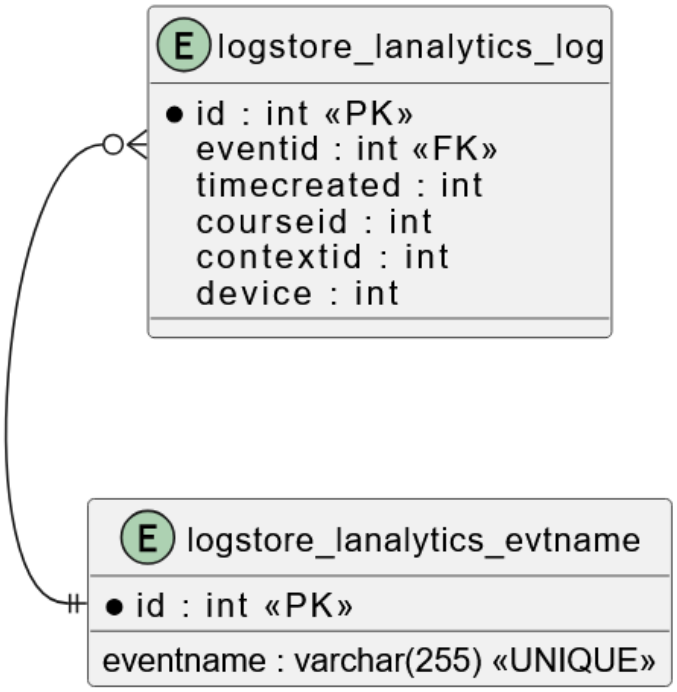



**Figure 9: Learning analytics log table simplified structure**

currence of an event linked to a specific `contextid` and a context in Moodle is directly associated to a precise Moodle activity or resource; as the learning analytics plugin is a web analytics plugin, it stores all actions with device information.

The `courseid` field is the only redundant information that is conserved to reenforce query joins performance based on this field. Based on well chosen indexes, other data about the log may be recovered without redundancy.

Fore good understanding, the table `logstore_lanalytics_evtname` stores information about generic actions, like 'consult a file' or 'submit a quiz attempt' while table `logstore_lanalytics_log` stores information about precise actions with device like 'file 16543 was consulted with device 32' or 'quiz 8776 was attempted with device 5'.

The log plugins are placed in `/admin/tool/log/store` folder and the learning analytics log plugin structure is the following :

```
moodle-logstore_lanalytics/
   classes/
      log/
         store.php
      privacy/
         provider.php
      task/
         cleanup_task.php
```

```
    devices.php
cli/
    apply.php
    import.php
    test-devices.php
db/
    install.xml
    tasks.php
    upgrade.php
lang/
    en/
        logstore_lanalytics.php
settings.php
version.php
README.md
LICENSE
```

and the most important actions are driven by these files:

- `classes/log/store.php` is the main action that performs events storage;
- `tasks/cleanup_task.php` performs the cron task to cleanup old data that are not more needed;
- all files in `cli` folder are cli scripts that may be useful to import data from the default moodle logstore table;
- all files in `db` folder are common plugin files with tables to create in `install.xml` and listed cron tasks in `task.php`;
- the `lang` folder has the common structure for managing language strings integrated in the plugin;
- the `version.php` and `settings.php` files are common plugin files that respectively describe the plugin version and administrator settings.

The `settings.php` file learns us that this plugin integrates admin settings to define

- Moodle roles to be tracked or excluded of the actions log;
- the list of course ids to be tracked or excluded from the actions log;
- the log lifetime (after this delay, actions information are suppressed from the log).

The `db/install.xml` file creates the 2 tables necessary for this plugin.

The `classes/log/store.php` file contains functions that help to treat the data and the function `insert_event_entries` stores the pertinent data in the log tables based on the admin settings defined. The code below presents an extract of the `insert_event_entries` function that proceeds the events information storage. If event has already been stored in the event table, its id is retrieved (at line 8) otherwise generic information about the event is stored in the table event (at line 12). Then the information to record in log table are prepared to be stored at line 24.

```
protected function insert_event_entries($events) {
        // … (code omitted)
        foreach ($events as $event) {
                // … (code omitted)
                  $records = [];
            $dbevent = $DB->get_record('logstore_lanalytics_evtname', ['
               eventname' => $event['eventname']], 'id');
            if ($dbevent) {
                $eventid = $dbevent->id;
            } else {
                $evtrecord = new stdClass();
                $evtrecord->eventname = $event['eventname'];
                $eventid = $DB->insert_record('
                   logstore_lanalytics_evtname', $evtrecord, true);
            }
            $record = new stdClass();
            $record->eventid = $eventid;
            $record->timecreated = $event['timecreated'];
            $record->courseid = $event['courseid'];
            $record->contextid = $event['contextid'];
            $record->device = $event['device'];
            $records[] = $record;
        }

        if (count($records) !== 0) {
            $DB->insert_records('logstore_lanalytics_log', $records);
                  // … (code omitted)
          }
{
```

The `classes/task/cleanup_task.php` file main function retrieves the config loglifetime administrator setting (via `get_config` function at line 4) and deletes all records that are older than the loglifetime (via `delete_records_select` function at line 19) :

```
public function execute() {
      global $DB;

      $loglifetime = (int) get_config('logstore_lanalytics', 'loglifetime
         ');

      if (empty($loglifetime) || $loglifetime < 0) {
            return;
      }

      $loglifetime = time() - ($loglifetime * 3600 * 24); // Value in
         days.
      $lifetimep = array($loglifetime);
```

```
12        $start = time();
13
14        while ($min = $DB->get_field_select("logstore_lanalytics_log", "MIN
              (timecreated)", "timecreated < ?", $lifetimep)) {
15            // Break this down into chunks to avoid transaction for too
                  long and generally thrashing database.
16            // Experiments suggest deleting one day takes up to a few
                  seconds; probably a reasonable chunk size usually.
17            // If the cleanup has just been enabled, it might take e.g a
                  month to clean the years of logs.
18            $params = array(min($min + 3600 * 24, $loglifetime));
19            $DB->delete_records_select("logstore_lanalytics_log", "
                  timecreated < ?", $params);
20            if (time() > $start + 300) {
21                // Do not churn on log deletion for too long each run.
22                break;
23            }
24        }
25
26        mtrace(" Deleted old log records from lanalytics log store.");
27    }
```

This first plugin stores the data about actions and devices. Notice that the log plugin install supposes the associated log store to be activated before to begin recording data. This is done while accessing the admin menu Site Administration > Plugins > Logging > Manage log stores. Click on the eye icon in front of the row ”Learning Analytics Log” to enable the log store.

Let now analyze how this data is retrieved and presented in associated report.

### 3.3.2 Second plugin : a report plugin presenting the course analytics with student and teacher view

The second plugin has a more complicated structure because it is a local plugin but main report files are stored in the `reports` folders. We will not analyze this plugin code in detail because it is a report plugin and the social flow view needs to be shown on the user dashboard and will therefore be a block plugin.

But it is interesting to notice all strategies that enabled the author to improve performances:

- the log table stores only integer values; this approach is called “database normalization” and is proven on improve database queries performance;
- the indexes on the table are chosen to ensure the queries performance;
- the queries are optimized;
- a data aggregation approach was also adopted as far as processes only require anonymous data.

Before to work on these plugins adaptation to implement the social flow view, it is inter-

esting to notice how the legal issues have been taken into account in Thomas Dondrof's Phd.

### 3.3.3 Remark on legal issues

Thomas Dondrof's dissertation also underlines the importance of legal issues in a learning analytics project. He stresses the importance of carrying out anonymous or pseudonymous processing as far as possible. He also cites [12] that propose a height-point checklist named DELICATE that can be applied by researchers, policy makers and institutional managers to facilitate a trusted implementation of Learning Analytics (see figure 10).

These recommendations also align with the results of [13] who examined students' perceptions of privacy principles related to learning analytics. They indicate the importance of involving students in learning analytics projects, keeping data only for as long as is necessary, working with complete and correct data, and informing students of the following points:

- how the data is collected;
- who has access to the data;
- the types of analysis carried out;
- the purpose of these analysis;
- how long the data is kept;
- the type of identification and aggregation carried out.

For the social flow project, no new data will be stored in Moodle. But a new treatment on these data will be proceeded and a student view will be implemented. We will therefore have to clearly inform students about this new treatment of their data.

## 3.4 Some plugins with user preferences

As we want our social flow plugin to have user preferences, it is also interesting to study plugins that propose user preferences like block_myoverview, block_timeline. These two plugins belong to Moodle core and may be found in the `blocks` folder of the Moodle code.

And the user preferences are well documented in Moodle official developer documentation.

In the existing blocks, I note that user preferences are defined via the `set_user_preference()` function call and preferences are retrieved via the `get_user_preferences()` function. The default values for user preferences are defined at the beginning of `lib.php`. This file also contains the function used to define user preferences.

The `settings.php` file is used to define all the options that can be set by the administrator, not to be confused with user preferences.

Note that user preferences are stored in Moodle default table `user_preferences`.

To exercise, I wrote a simple block plugin with user preferences:
https://github.com/zabellemotte/block_customprefs

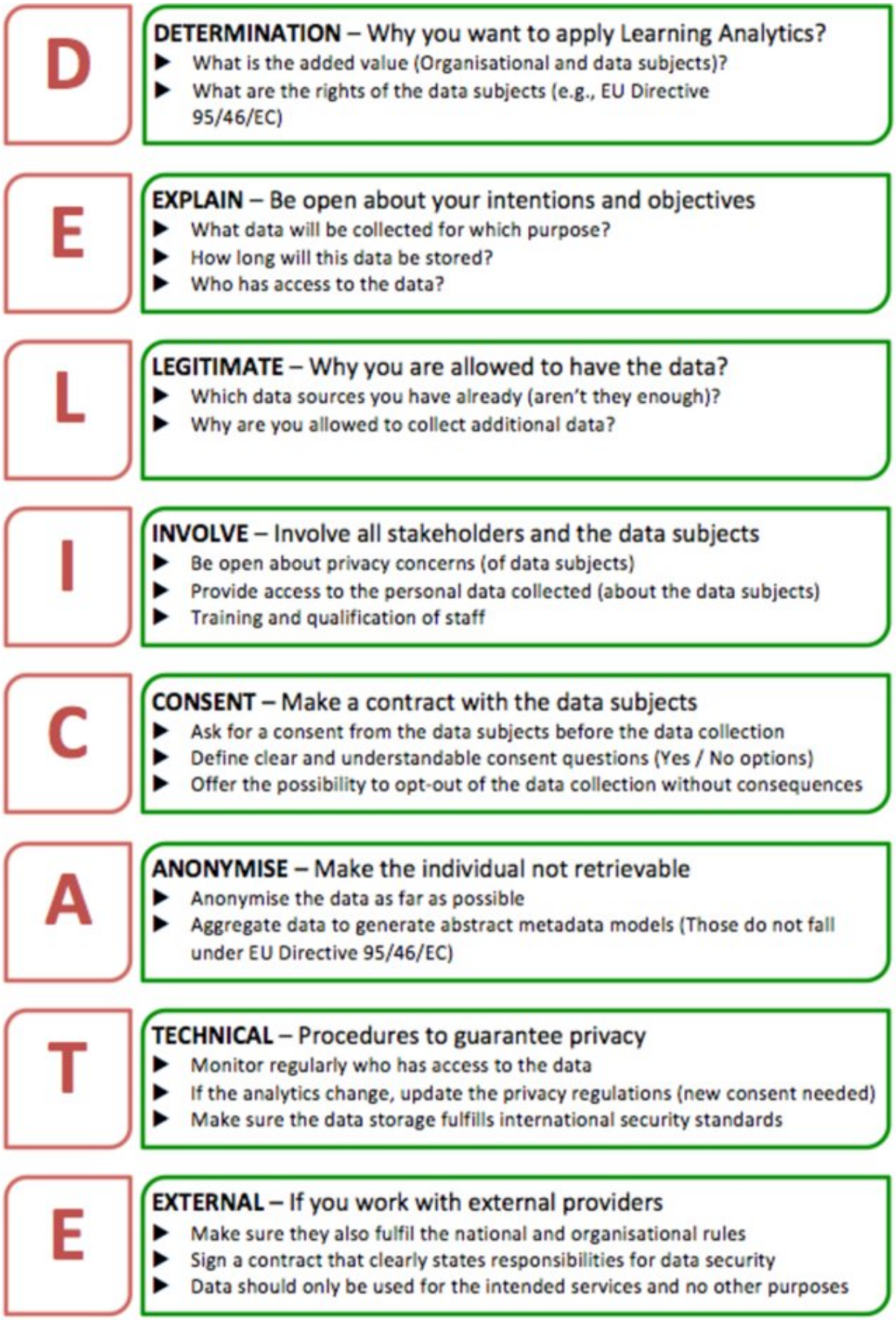



**Figure 10: The DELICATE Checklist. © Drachsler Greller, 2016.**

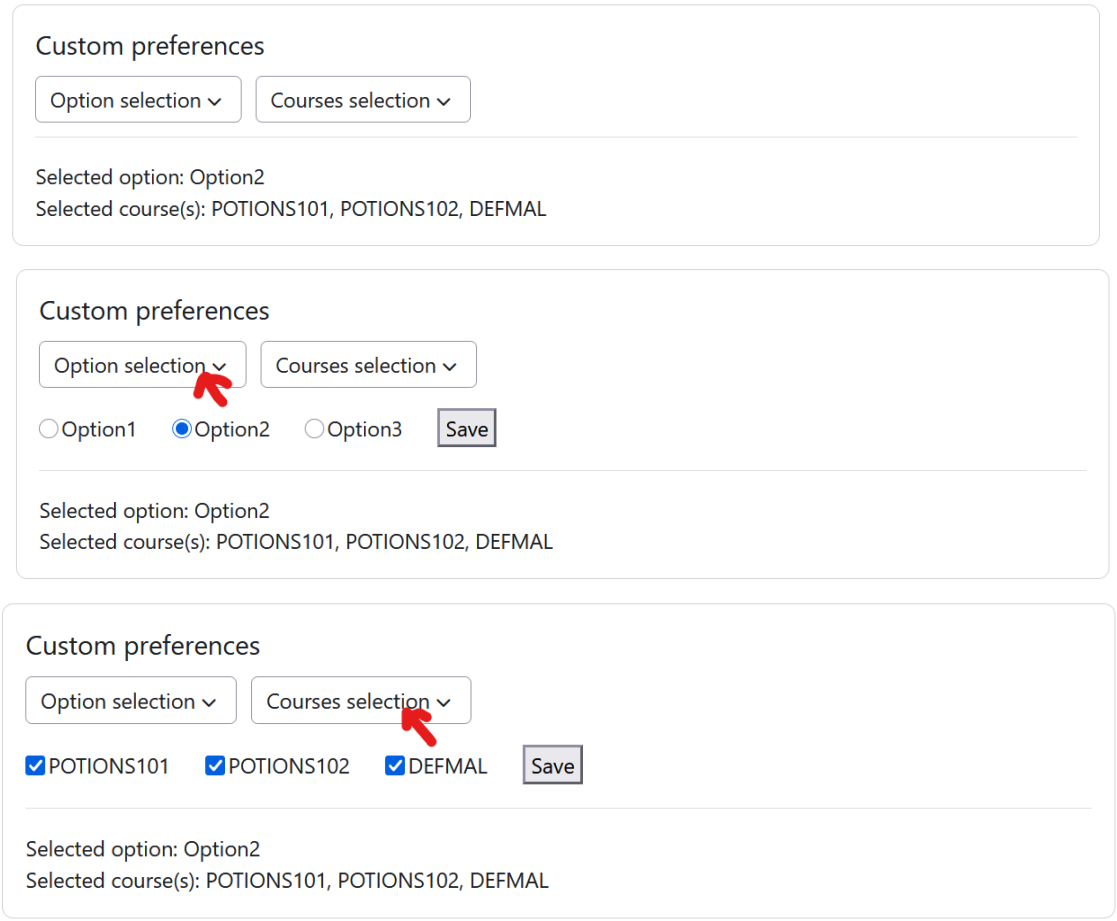



**Figure 11: Custom preferences block presentation**

The final rendering of the block is presented at figure 11.

This block has 2 preferences :

- a single option choice among 3 options,
- an option to make a selection in the user course list.

The block content displays a simple string describing the options values.

This plugin is a block plugin, therefore placed in `blocks` folder, and it has the following file structure :

```
moodle-block_customprefs/
   version.php
   lib.php
   block_customprefs.php
   access.php
   provider.php
   styles.css
   amd/
       build/
            optionselection.min.js
       src/
            optionselection.js
   classes/
       privacy/
   db/
       access.php
   lang/
        en/
        customprefs.php
```

The `access.php` file is needed in for a block plugin to define permissions necessary to manage the block. The `amd` folder contains the javascript function to show or hide the options menu.

The default preferences are defined in the `lib.php` file and the `block_customprefs.php` presents the block content with the options buttons displayed based on user preferences if defined.

Here is for example the code that defines the `$currentchoice` variable with the user first option value based on the choice made in the option menu, or based on the defined user preference :

```
if( isset( $_POST['customprefs_optionchoice'] ) ){
    $currentchoice = $_POST['customprefs_optionchoice'];
        set_user_preference('customprefs_optionchoice', $currentchoice);
    }
    else {
        $currentchoice=get_user_preferences('customprefs_optionchoice');
    }
```

These files code is commented with more details in Appendix A.

Notice a block installation does not automatically activate the block; it has to be placed on the default dashboard for all users from admin page Administration > Site administration

> Appearance > Default Dashboard page. You then have to activate the edit mode, add the block at you desired place and then click on the button 'Reset Dashboard for all users' to apply these settings to the Dashboard for everyone on the site. As an admin, it is normal that the block presents no data. Connect as a student in an active course to see this block in action.

We now dispose of all puzzle pieces necessary to begin building our social flow view plugins.

# 4 First challenge : storing the pertinent data

The social flow plugins are presented in two subsections : database structure and code implementation.

## 4.1 Social flow log database structure

Our social flow log plugin is built on a similar log table as the learning analytics log plugin but an event table is pre-filled with all events we want to record. No new event will be stored in the events table by the plugin but a Moodle administrator could add new events to record, linked for example to an additional plugin.

For good understanding, a graphical representation of useful tables is presented at figure 12 where the social flow plugin tables are presented with green titles while the blank titled tables are common Moodle tables that will be needed to build social flow information.

To make these tables content clear, let say that the table `logstore_socialflow_evts` stores information about generic actions, like 'consult a file' or 'submit a quiz attempt' while table `logstore_socialflowlog_log` stores information about precise user actions like 'user 73423 consulted file 16543' or 'user 73919 attempted quiz 8776'. Remind that a context in Moodle is directly associated to a precise Moodle activity or resource. The `courseid` field in log table is a redundant information that will help queries to be more efficient, as far as the social flow block implements a filter on course selection.

The events table is pre-filled with the pertinent events for our project through the `db/install.php` file. Here is an extract of the function content that may directly be linked to events as presented in the 3.1 section. The indexes on this tables will be discussed later.

```
1 function xmldb_logstore_socialflow_install() {
2     global $DB;
3     $records = array(
4         array('id'=>'1','eventname'=>'\\mod_assign\\event\\
              course_module_viewed','actiontype'=>'consult'),
5         array('id'=>'2','eventname'=>'\\mod_assign\\event\\
```

```
            assessable_submitted','actiontype'=>'contrib'),
        array('id'=>'3','eventname'=>'\\mod_workshop\\event\\
            course_module_viewed','actiontype'=>'consult'),
        array('id'=>'4','eventname'=>'\\mod_workshop\\event\\
            submission_created','actiontype'=>'contrib'),
        array('id'=>'5','eventname'=>'\\mod_workshop\\event\\
            submission_assessed','actiontype'=>'contrib'),
      // … (code omitted)
    );
    $DB->insert_records('logstore_socialflow_evts',$records);
}
```

## 4.2 Social flow log code structure

The social flow log file structure is similar to the learning analytics equivalent. The log plugins are placed in `/admin/tool/log/store` folder and the learning analytics log plugin structure is the following :

```
moodle-logstore_socialflow/
   classes/
       log/
           store.php
       privacy/
           provider.php
       task/
            cleanup_task.php
   cli/
       import.php
   db/
       install.xml
       install.php
       tasks.php
   lang/
       en/
           logstore_socialflow.php
   settings.php
   version.php
   README.md
   LICENSE
```

The `classes/log/store.php` file acts to record data in the log table only if it concerns an event that exists in events table.

The main code difference compared to the learning analytics plugin is in the code extract below, which shows the function that stores the events :

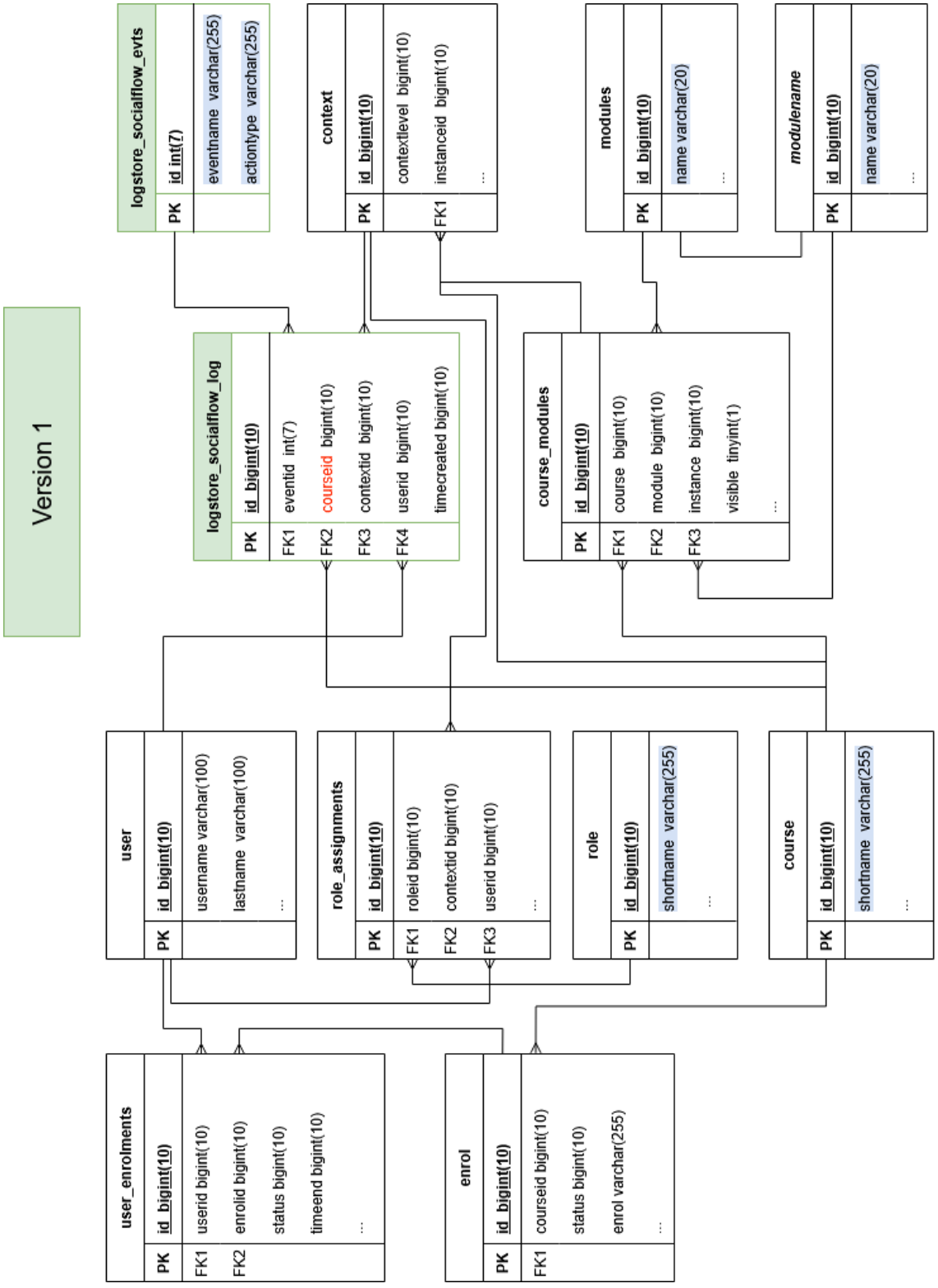



**Figure 12: Social flow plugin database structure version 1**

```
1  protected function insert_event_entries($events) {
2          // … (code omitted)
3          $records = [];
4          foreach ($events as $event) {
5              // … (code omitted)
6              $eventid = 0;
7              $dbevent = $DB->get_record('logstore_socialflow_evts', ['
                   eventname' => $event['eventname']], 'id');
8              if ($dbevent) {
9                  $eventid = $dbevent->id;
10             } else {
11                 continue;
12             }
13             $record = new stdClass();
14             $record->eventid = $eventid;
15             $record->timecreated = $event['timecreated'];
16             $record->courseid = $event['courseid'];
17             $record->contextid = $event['contextid'];
18             $record->userid = $event['userid'];
19             $records[] = $record;
20         }
21
22         if (count($records) !== 0) {
23             $DB->insert_records('logstore_socialflow_log', $records);
24                     // … (code omitted)
25         }
26 }
```

The `$dbevent` query (at line 7) result tells if the actual event is one of those recorded in our table of predefined events. If it is not the case, no data is stored (and the script continues at line 11).

For this project, the `classes/task/cleanup_task.php` does not delete all tasks that are older than the defined logtime because we need to keep the information about all active events. The chosen option is to delete all data about events that have not been performed by at least one user during log lifetime. Selecting old data is therefore based on a database query integrating a `NOT IN` clause at line 18.

```
public function execute() {
        global $DB;
        mtrace("Log cleanup begins ...");
        $loglifetime = (int) get_config('logstore_socialflow', '
           loglifetime');
        if (empty($loglifetime) || $loglifetime < 0) {
            $loglifetime=14;
        }

        $loglifetime = time() - ($loglifetime * 3600 * 24); // Value in
           days.
        $start = time();

        // Events are deleted when no event arised on the contextid
           during the longlifetime.
        // As long as a there are events linked to a contextid during the
            loglifetime, data is preserved
        // to be able to indicate user if he had an action linked to this
            contextid
        // (even if this action is outside the loglifetime).
        $old_data = $DB->get_records_sql(
           "SELECT id FROM {logstore_socialflow_log} WHERE timecreated <=
              $loglifetime
            AND contextid NOT IN (SELECT DISTINCT contextid
                                  FROM {logstore_socialflow_log}
                                  WHERE timecreated > $loglifetime)"  );
    if ($old_data) {
        $ids = array();
        foreach ($old_data as $row) {
            $ids[] = $row->id;
        }
        // data suppression is chunked in several delete actions of 500
           records to avoid database trashing
        while (count($ids)>0){
            $lids=count($ids);
            $cids=array_slice($ids, 0, 50);
            $ids= array_slice($ids, 50, $lids - 50);
            $clauseIN = implode(', ', $cids);
            $select = "id IN ($clauseIN)";
            $DB->delete_records_select("logstore_socialflow_log",$select)
               ;
         }
        mtrace("Old log records from socialflow log store deleted.");
    }
    else{
            mtrace("No old data to delete in socialflow logs.");
    }
}
```

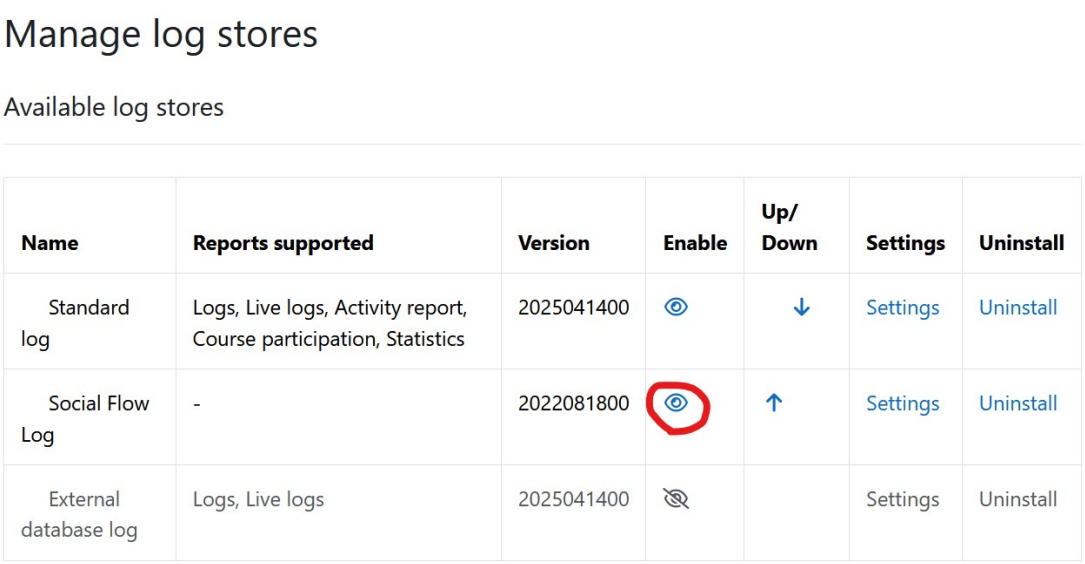



**Figure 13: Activation of the social flow log store in Moodle administration**

## 4.3 Remark on log plugin installation

This first plugin stores the data about the selected actions. Like the learning analytics log plugin, our log plugin install supposes the associated log store to be activated before to begin recording data. This is done while accessing the admin menu Site Administration > Plugins > Logging > Manage log stores. Click on the eye icon in front of the row ”Learning Analytics Log” to enable the log store as shown in figure 13.

# 5 Second challenge : building the database and queries

Now we have the pertinent user events in the log table, we have to build queries to extract the data to show the social flow information.

The first piece of information we need to build the social flow view is the number of participants in each course. Associated to the hits information (the number of times each event has been performed), it will enable us to compute the frequency associated to each event. And, for the selected course list, we have to sort all events based on frequency and display the first lines to the student.

To help understanding the process, we present a graphic at figure 14 of the updated database structure with intermediate tables proposed in this section. The tables with green title are the social flow dedicated tables while the white titled tables are common Moodle tables.

This section cuts the social flow view building in several progressive steps.

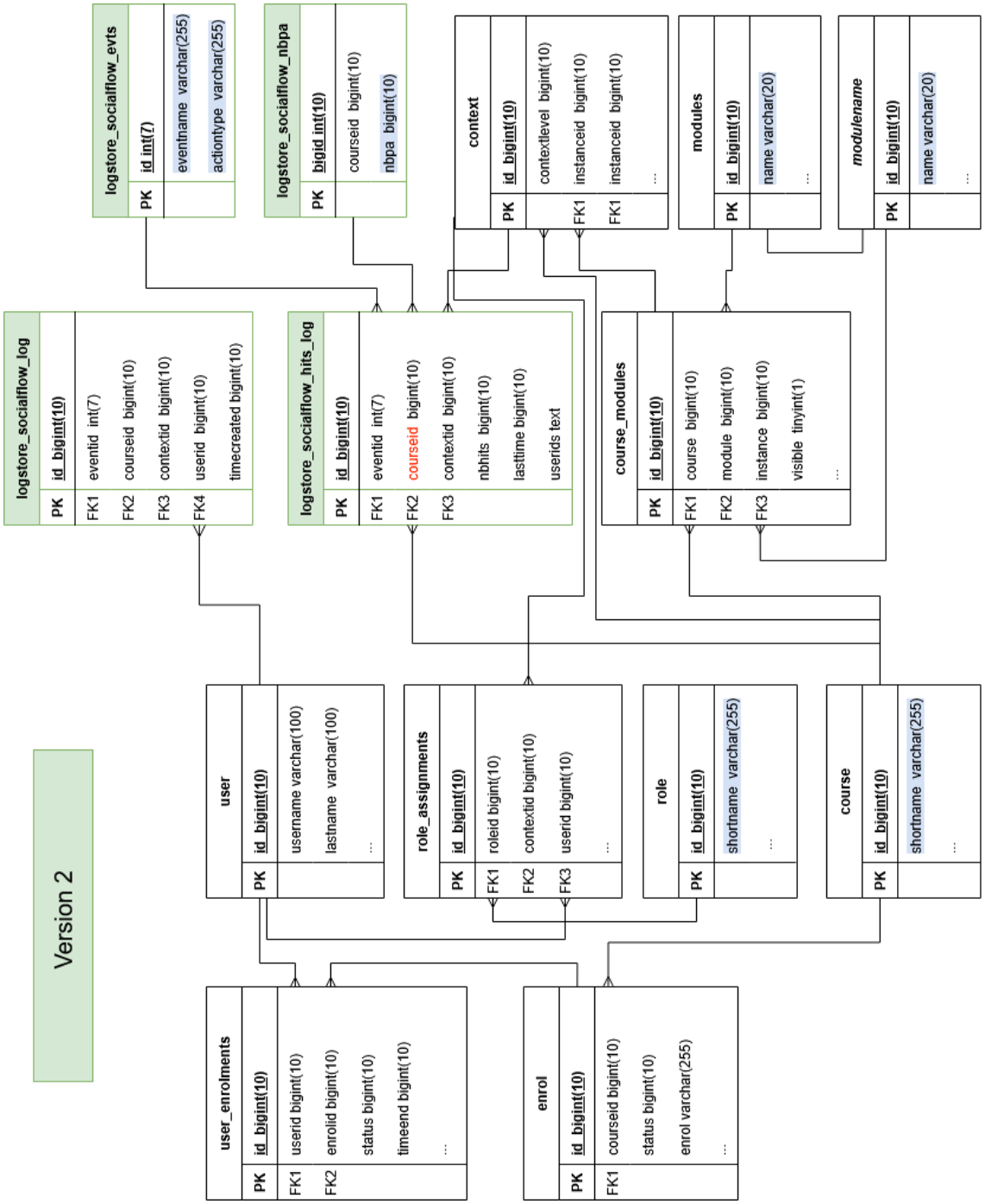



**Figure 14: Social flow plugin database structure version 2**

## 5.1 The number of participants in a course placed in an intermediate table

To be able to compute an action frequency, we need to have the number of participants in the course.

And the query to compute the number of participants in a course is not easy because it requires excluding

- the enrollments associated to an inactive enrollment method;
- the enrollments associated to suspended or deleted users;
- the enrollments with passed expiration date;
- the disabled enrollments;

and it therefore looks like follows:

```
SELECT COUNT(DISTINCT(u.id)) AS nbstu
FROM {user} u
INNER JOIN {user_enrollments} ue ON ue.userid = u.id
INNER JOIN {enrol} e ON e.id = ue.enrolid
INNER JOIN {role_assignments} ra ON ra.userid = u.id
INNER JOIN {context} ct ON ct.id = ra.contextid
     AND ct.contextlevel = 50
INNER JOIN {course} c ON c.id = ct.instanceid
     AND e.courseid = c.id
INNER JOIN {role} r ON r.id = ra.roleid
     AND r.shortname = 'student'
WHERE e.status = 0 AND u.suspended = 0 AND u.deleted = 0 AND (ue.timeend
   = 0 OR ue.timeend > $now)
AND ue.status = 0 AND c.id = $courseid;
```

as explained in a Stackoverflow discussion.

I noticed it took 119 ms on our production Moodle to execute such a query on a course with 250 participants. But this query does not need to be performed every time a student displays the social flow view. This data may be computed once a night and stored in an intermediate table.

We therefore add a new table in our log plugin to store this information. The `db/instal.xml` file integrates a new table named `logstore_socialflow_nbpa` that is presented at figure figure 14.

As the number of participants will be retrieved based on the `courseid`, an index is placed on that field.

And our log plugin counts a new task in the `classes/task` folder, defined in `nbpa_task.php` that extracts the list of courses where at least on action took place and stores this list with the computed number of participants.

To update information in the intermediate `nbpa` table, I have noticed it was much faster to completely regenerate all the table data than to adjust the table content. A new temporary table is therefore created to store the new data and the table content is finally replaced and temporary table dropped.

The main action depends on the `execute` function that is commented in Appendix B.

## 5.2 The hits actions placed in an intermediate table

When a student loads the social flow view, the number of hits for each actions performed in its selected courses have to be computed. Computing the hits is also very resource-intensive and, from a functional view, it may become stressful for students if social flow data changes too often.

I decided that a reasonable frequency to update data in the social flow block was once a day. And I therefore decided to create a new table to store hits information and a new cron task to store the hits information.

The `logstore_socialflow_hits` table structure is defined in the log plugin `db/install.xml` file and is presented at figure 14. Each line of this table records an information like 'file 45532 was consulted 165 times' or 'quiz 5432 was attempted 76 times'.

The `userids` field stores a concatenate text with all the user ids that performed the associated action. Storing this data in this table is not heavy for the query because these events have to be retrieved to be counted to have the hit number. And this will enables us to easily know if a user already carried out an action, without having to search in the main social log table.

The associated cron task is defined in `classes/task/hits_task.php` as an `execute()` function. It also uses a temporary table to completely regenerate the table content. This function code is commented in Appendix C.

Based on the hits table and on the number of participants table, it is now possible to write a query to retrieve the social flow data.

## 5.3 Building an optimized social flow query with common table expression

As advised in the database bible [14], the query with join optimization is based on a preliminary selection of the target data before implementing the join. This approach is called 'common table expression'. And as far as the joins are read form left to right, it is better to begin implementing the more restrictive join to ensure query efficiency.

The student filter options are presented in the final rendering at figure 15, 16, 17 and 18 and stored respectively in these variables :

- `$loglifetime` is the date timestamp determining the time period to be taken into

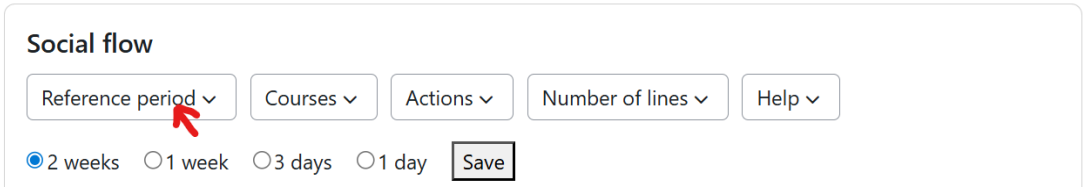



**Figure 15: Final presentation of reference period filter in the social flow view**

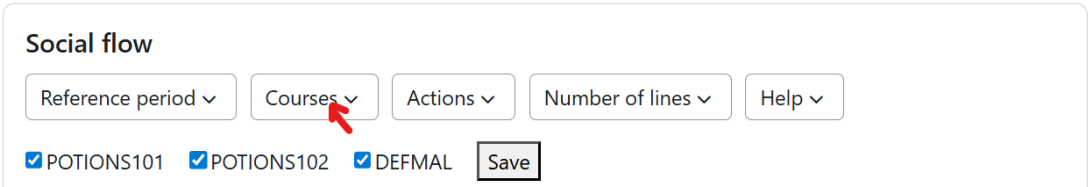



**Figure 16: Final presentation of course filter in the social flow view**

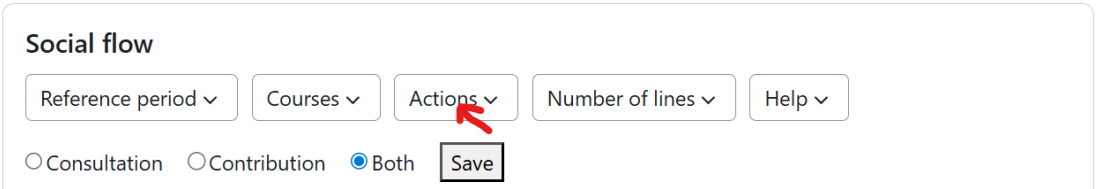



**Figure 17: Final presentation of action type filter in the social flow view**

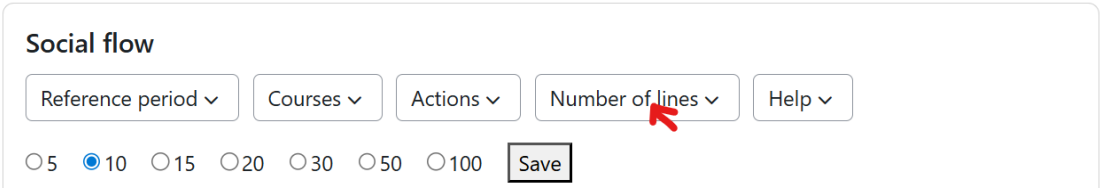



**Figure 18: Final presentation of number of lines filter in the social flow view**

account in the social flow (defined in the log plugin admin settings);

- `$currentcourses` is an array with the student selected course ids and a concatenate version of the course ids separated with commas (to be used in social flow query) is generated in `$currentcoursesstring` ;
- `$currenttype` indicates the action types selected : 'consult', 'contrib', or 'both';
- `$currentitemnum` is the value of a new option introduced to help student to define the number of lines to show in social flow block.

Finally, `$coursenbpa` is an associative array allowing to retrieve the number of participants of a course, from the course id.

With these conventions, the optimal version of the social flow query is formulated as follows :

```
1  $sql ="WITH event_hits AS (
2                          SELECT h.id, h.courseid, h.contextid, h.eventid,
                               h.nbhits
3                          FROM {logstore_socialflow_hits} h
4                          WHERE h.courseid IN (" . $currentcoursesstring .
                               ") AND h.lasttime > " . $loglifetime . "
5                   )
6                   SELECT ei.id AS hitid, ei.contextid, ei.eventid, ei.
                       courseid, evts.actiontype, c.instanceid, cm.
                       instance, m.name, ei.userids,
7                   CASE";
8  foreach ($currentcourses as $id) {
9         $nbpa = $coursenbpa[$id];
10        $sql .= "WHEN ei.courseid = " . $id . " THEN ei.nbhits / " . $nbpa
              . " ";
11 }
12 $sql .= "END AS freq
13             FROM event_hits ei
14             INNER JOIN {logstore_socialflow_evts} evts ON ei.eventid =
                   evts.id
15             INNER JOIN {context} c ON ei.contextid = c.id
16             INNER JOIN {course_modules} cm ON c.instanceid = cm.id
17             INNER JOIN {modules} m ON cm.module = m.id ";
18
19 if ($currenttype != 'both') {
20       $sql .= "AND evts.actiontype = '" . $currenttype . "' ";
21 }
22
23 $sql .= "ORDER BY freq DESC LIMIT " . $currentitemnum . ";"
```

This optimized query proceeds as follows:

- it first extracts the part of hits table concerning the chosen courses and reference period (lines 1 to 5);

- then it computes the frequency associated to each event while dividing the hits value by the course number of participants (lines 8 to 12);
- it then makes a join on the events table to be able to fetch the action type (line 14)
- at lines 15 to 17, the joins are applied to be able to access the module name;
- the result is ordered decreasingly with limit based on the user preference stored in `$currentitemnum`.

## 5.4 Fetching all information about each action in social flow block

Our main social flow query enables us to retrieve a list of `contextids` associated to the most frequent actions with defined filter options. From the `contextids`, we now need to fetch the other information to be shown in the social flow block.

All the needed data is stored in the fields in blue in the database graphic presented at figure 14.

We do not describe these queries in detail because there are easy `SELECT` queries or easy call to dedicated Moodle functions, enabling to fetch data in a table based on a primary key search :

- the course title comes form the `course` table, fetching `shortname` while searching on the course `id`;
- the module icon is build from `pix_icon` Moodle function called with the module `name` in `modules` table;
- the module instance name (the name of the activity defined by the teacher) is found in `name` field in `modulename` table and requires one single query while searching the `instanceid`.
- the event action type (’consult’ or ’contrib’) is directly fetched trough the above query;
- the ‘Done, Not done’ indicator for the user relies on a search on the hits table containing user ids associated to the action.

An extract of the `block_socialflow.php` main function is commented ins appendix D for good understanding.

## 5.5 Choosing efficient tables indexes

The database queries performance is also linked to correct indexes configuration. Indexes have to be defined to make table queries with `WHERE`, `JOIN`, `ORDER BY` and `GROUP BY` clauses more efficient. When several fields are mentioned in one of the above clause, different indexes combination may be tested, that may lead to different results depending on the data structure. The best way to find optimal combination is to use the query performance analysis, for example using the `EXPLAIN ANALYZE` command in PostgreSQL or in MySQL.

Our `evts` table is only queried on the basis of the `id` field and therefore only needs this index.

Our `nbpa` table is queried only on the basis of `courseid` field and therefore only needs this index.

The main `log` table is queried for several purposes:

- building the `nbpa` table supposes to extract the `courseid` list that may be helped while adding an index on `courseid` field;
- building the `hits` tables was more efficient while adding an index on `timecreated` field and no other index nor index combination succeeded to make the query more efficient;
- retrieving the user 'Done, Not done' indicator was proven to be more efficient while adding a `(contextid,userid)` index.

Finally, the `hits` table, queried in our big social flow query, was proven to be more efficient based on indexes on `courseid` (used in first `WHERE` clause) and combined index on `(contextid,eventid)` (used in the two first joins).

These indexes optimization may depend on your data structure. Efficient indexes may be different on a Moodle with few courses, a lot of modules and a lot of users than those working on a Moodle with a lot of courses, few modules and few users.

Other queries on the default Moodle tables where proven to be efficient with default defined indexes.

## 5.6 Performance analysis on a copy of our production database

All the queries and their preliminary more naive versions have been tested on a copy of our production Moodle (6000 courses, 4800 teachers, 42000 students) copied at the end of an academic year (in may). The default Moodle log table reached more than 146 million lines while the associated social flow log table was filled with almost 1 million lines. The preliminary versions of the queries, without intermediate tables and without the common table expression did not retrieve results after several hours.

With all the above precautions (intermediate tables, common table expression, intelligent joins and indexes), the queries became very fast :

- extract the number of participants in a course takes 51ms;
- big social flow queries take 60ms;
- get the 'Done, Not Done' information for one student action takes 35ms.

The approach consisting in doing a complete rebuilt of `nbpa` and `hits` tables at each cron run was also proven to be much more efficient than trying to add tyhe new actions and suppress the old actions in these tables.

## 5.7 Improvement to show 'Done, Not Done' student status in real time

The use of intermediate tables enable us to ensure good performance, but while testing the social flow, I noticed the 'Done, Not Done' flag needed to be rendered in real time, so that a student that just made an action can see it is well done in social flow block.

The strategy in the social flow block function is easy : if the student `id` is not associated to the action hit, we look in the log table queue, since last cron run, if the action has not been performed by student. The code below is part of the `block_socialflow.php` file :

```
1  if (in_array($cuserid, $userids)){
2        $isdone=1;
3  } else {
4      // verify if user operated action recently while searching in recent
          logs (since last hits cron run)
5      $sql52="SELECT lastruntime FROM {task_scheduled} WHERE component LIKE
           'logstore_socialflow' AND classname LIKE '%hits%' ";
6      $result52 = $DB->get_record_sql($sql52);
7      if ($result52) {
8            $lastruntimet=$result52->lastruntime;
9            $sql53="WITH recent_log AS (SELECT * FROM {
                logstore_socialflow_log}  WHERE timecreated>$lastruntimet)
10                        SELECT COUNT(id) AS nbdone
11                        FROM recent_log
12                        WHERE courseid=$courseid
13                        AND contextid=$contextid
14                        AND eventid=$eventid
15                        AND userid=$cuserid";
16           $result53 = $DB->get_record_sql($sql53);
17           if ($result53){
18                 $nbrecent=$result53->nbdone;
19                 if ($nbrecent>0){
20                       $isdone=1;
21                 } else {
22                       $isdone=0;
23                 }
24           } else {
25                 $isdone=0;
26           }
27     }
28 }
```

The implementation proceeds as follows:

- the last runtime of hits cron task is retrieved at line 5-9 ;
- then a `SELECT` query is ran to find if student made the action since last runtime.

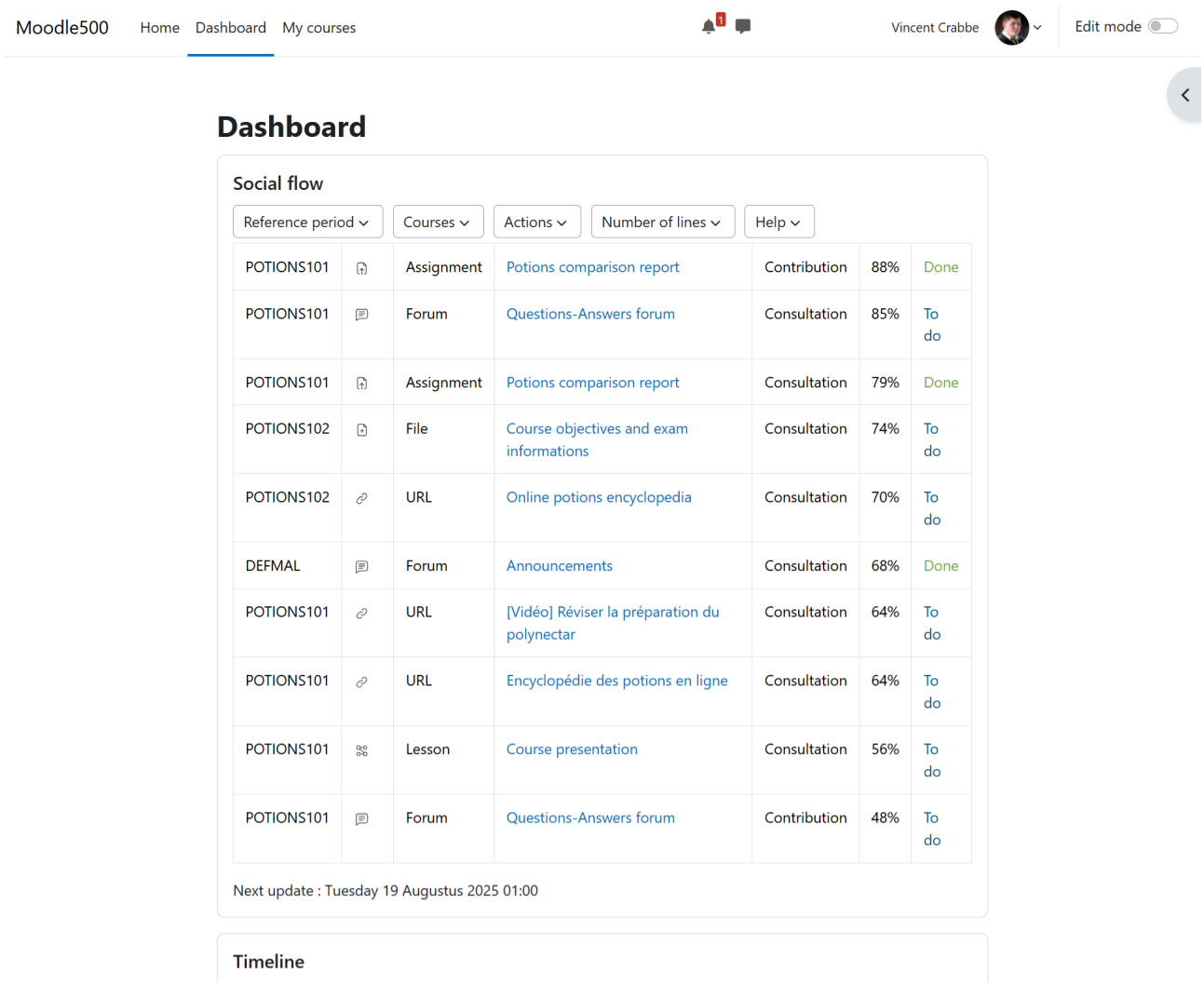



**Figure 19: Social flow block rendering on a student dashboard**

## 5.8 Social flow block rendering (version 1)

This ends the implementation of the first version of the social flow plugin. This is already a big challenge that has been achieved. The first version of the block rendering on the dashboard is presented in figure 19.

## 5.9 Remark on block plugin installation

Notice a block installation does not automatically activate the block; it has to be placed on the default dashboard for all users from admin page Administration > Site administration > Appearance > Default Dashboard page. You then have to activate the edit mode, add the block at you desired place and then click on the button 'Reset Dashboard for all users' to apply these settings to the Dashboard for everyone on the site. As an admin, it is normal the block presents no data. Connect as a student in an active course to see this block in action.

The next section proposes new evolutions of the social flow block plugin to take some data exclusions in the social flow view.

# 6 Third challenge : managing excluded activities

While testing the first version of the plugins, I found a functional limitation : some Moodle activities have a deadline and it would be better to indicate when activities have late deadline in the social flow. Activities that are hidden or in a hidden section should also be excluded from social flow. And finally activities with access restrictions conditions could also be excluded.

Taking all these exclusions into account requires new database improvements that are presented at figure 20.

## 6.1 New closing date table to manage activities with deadline

While analyzing the different Moodle activities, I found that the deadline field name may vary depending on the module (deadline, cutoffdate, limitdate, submissionend ...), and the default value when no deadline is defined may also vary and may be 0 or `null`. Assign and forum have also 2 different deadlines defined : a due date that is the announced deadline and a closing date that corresponds to the date the activity is closed. Such configuration enables teachers to accept late submissions.

The only efficient way to take these deadlines into account is therefore to create an extra table to store these information. To be able to make joins on this table for each action, the table needs to have as much lines as the hits table. The lines associated to events without deadline will therefore be filled with an infinite timestamp deadline (9999999999). Activities that support late submit will only require to show a custom message to student and will not influence the social flow query.

The `events` table (with yellow title) is augmented with 5 fields that enable us to manage activities with closing date and late submit:

- `moduletable` stores the module table name where the deadline field is stored;
- `hasclosingdate` indicates if the event has a closing date based on 0,1 value;
- `closingdatefield` records the closing date associated field name if existing or it has null value;
- `haslatesubmit` indicates if the event supports late submit based on 0,1 value;
- `latedatefield` records the late date associated field name or it has null value;

A new table is also created, `logstore_socialflow_closing`, and the tasks that builds the hits table also fills this table with as much lines as the hits table. The `execute` function of the `hit_task.php` file is augmented to fill the closing dates table. More details about this implementation may be found in Appendix E with commented code.

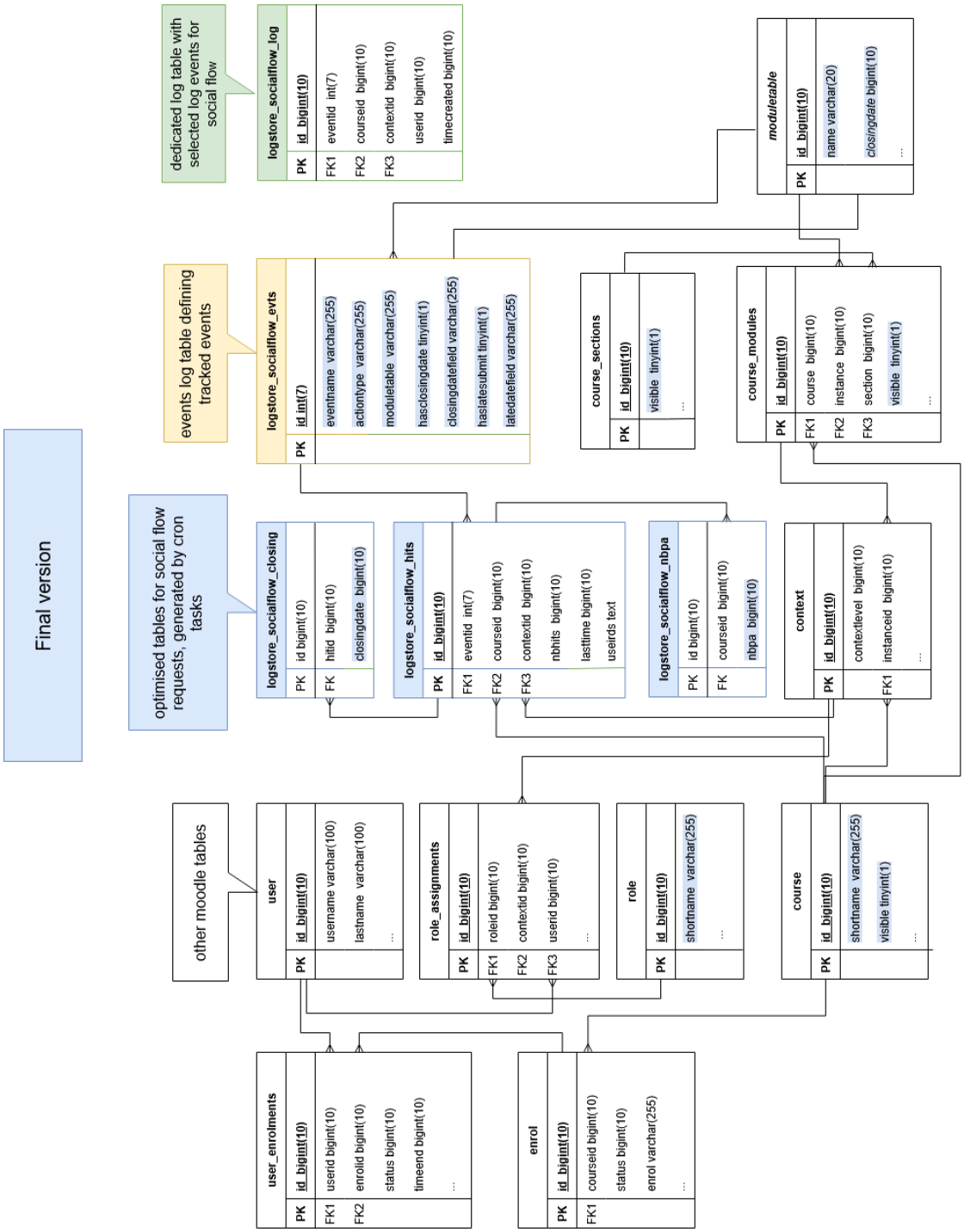



**Figure 20: Social flow plugin database structure version 3**

With this database structure, the social flow query in `block_socialflow.php` file has a new `JOIN` and `WHERE` clause at lines 4-5:

```
1  $sql = "WITH event_hits AS (
2                              SELECT h.id, h.courseid, h.contextid, h.
                                  eventid, h.nbhits, h.userids
3                              FROM {logstore_socialflow_hits} h
4                              INNER JOIN {logstore_socialflow_closing} c ON
                                   h.id = c.hitid
5                              WHERE h.courseid IN (" .
                                  $currentcoursesstring . ") AND h.lasttime
                                  > " . $loglifetime . " AND c.closingdate >
                                   (" . $now . " + 172800)
6                          )
7                          SELECT ei.id AS hitid, ei.contextid, ei.eventid, ei
                              .courseid, evts.actiontype, evts.moduletable,
                              evts.hasclosingdate, evts.haslatesubmit, evts.
                              latedatefield, c.instanceid, cm.instance, m.name
                              , ei.userids,
8                          CASE ";
9
10 foreach ($currentcourses as $id) {
11        $nbpa = $coursenbpa[$id];
12        $sql .= "WHEN ei.courseid = " . $id . " THEN ei.nbhits / " . $nbpa
              . " ";
13 }
14 $sql .= "END AS freq
15              FROM event_hits ei
16              INNER JOIN {logstore_socialflow_evts} evts ON ei.eventid =
                    evts.id
17              INNER JOIN {context} c ON ei.contextid = c.id
18              INNER JOIN {course_modules} cm ON c.instanceid = cm.id
19              INNER JOIN {modules} m ON cm.module = m.id ";
20
21 if ($currenttype != 'both') {
22       $sql .= "AND evts.actiontype = '" . $currenttype . "' ";
23 }
24
25 $sql .= "ORDER BY freq DESC LIMIT " . $currentitemnum . ";";
```

In the social flow view, an information message is also shown for activities with closing date :

- before the closing date, the closing date is reminded;
- if the closing date is passed for less than 2 days, the closing date is still reminded but the student is informed that the activity is closed;
- if the closing date is passed for more than 2 days, the actions is excluded from social flow, because such information does not stimulate engagement.

If a late submit date exists, message is also adapted to tell student its assign of forum submission will be marked as late. Screen capture presenting all exclusion results is presented at figure 21 and red messages at first and fourth line of the table render the deadlines additional information. The social flow block code adaptations are commented in Appendix F.

## 6.2 Social flow query revision to exclude hidden modules

Exclude hidden modules or modules in hidden sections only requires a new join and one new `WHERE` clause on the social flow query, at lines 18 and 20:

```
1  $sql = "WITH event_hits AS (
2                      SELECT h.id, h.courseid, h.contextid, h.eventid, h.
                          nbhits, h.userids
3                      FROM {logstore_socialflow_hits} h
4                      INNER JOIN {logstore_socialflow_closing} c ON h.id =
                           c.hitid
5                      WHERE h.courseid IN (" . $currentcoursesstring . ")
                          AND h.lasttime > " . $loglifetime . " AND c.
                          closingdate > (" . $now . " + 172800)
6                  )
7                  SELECT ei.id AS hitid, ei.contextid, ei.eventid, ei.
                       courseid, evts.actiontype, evts.moduletable, evts.
                       hasclosingdate, evts.haslatesubmit, evts.
                       latedatefield, c.instanceid, cm.instance, m.name, ei.
                       userids,
8                  CASE ";
9  foreach ($currentcourses as $id) {
10     $nbpa = $coursenbpa[$id];
11     $sql .= "WHEN ei.courseid = " . $id . " THEN ei.nbhits / " . $nbpa .
           " ";
12 }
13 $sql .= "END AS freq
14           FROM event_hits ei
15           INNER JOIN {logstore_socialflow_evts} evts ON ei.eventid = evts.
                id
16           INNER JOIN {context} c ON ei.contextid = c.id
17           INNER JOIN {course_modules} cm ON c.instanceid = cm.id
18           INNER JOIN {course_sections} cs ON cm.section = cs.id
19           INNER JOIN {modules} m ON cm.module = m.id
20           WHERE cm.visible = 1 AND cs.visible = 1 ";
21 if ($currenttype != 'both') {
22    $sql .= "AND evts.actiontype = '" . $currenttype . "' ";
23 }
24 $sql .= "ORDER BY freq DESC LIMIT " . $currentitemnum . ";";
```

## 6.3 Taking module restrictions into account : information is the only issue

Moodle implements a very complicated system of restrictions defined in a specific field of the module table in json format. It enables to define restriction based on groups, roles, date, note or profile field and even combine several types of restrictions. The easiest and fastest way to know is a student may access a resource or activity is to call function `get_fast_modinfo` as described below :

```
1                   $modinfo = get_fast_modinfo($courseid);
2                   $cm = $modinfo->get_cm($instanceid);
3                   if ($cm->uservisible) {
4                       $available=1;
5                   } else {
6                       $available=0;
7                   }
```

When a student is not authorized to access a module due to restriction, this line is show in gray in the social flow and a clear message explains he may not access the module due to restriction. We could also hide the line in the social flow, but it would be strange for students if their social flow to show 9 lines while they selected to show 10 lines ! A temporary table could not help to solve the problem in this case.

## 6.4 Social flow block rendering (final version)

This second and last version of the block rendering is presented in figure 21.

Presentation of late activities may be observed at lines 1 and 4 while presentation of an activity with access restriction is shown at line 5.

## 6.5 New queries remain efficient

On the copy of UCLouvain production database, the queries to compose the closing date table take about 1 second. And the new joins do not impact a lot the other queries execution times. Two cron tasks have finally been implemented : first to compute the number of participants in each course and second to compute the hits and the closing dates information one after the other. The hits table and closingdate table need to be recalculated one after the other because they need to be perfectly synced.

These tasks common execution times should give an idea on how often it could be ran. But from a functional view, student should be consulted to make this tool a guide without causing stress.

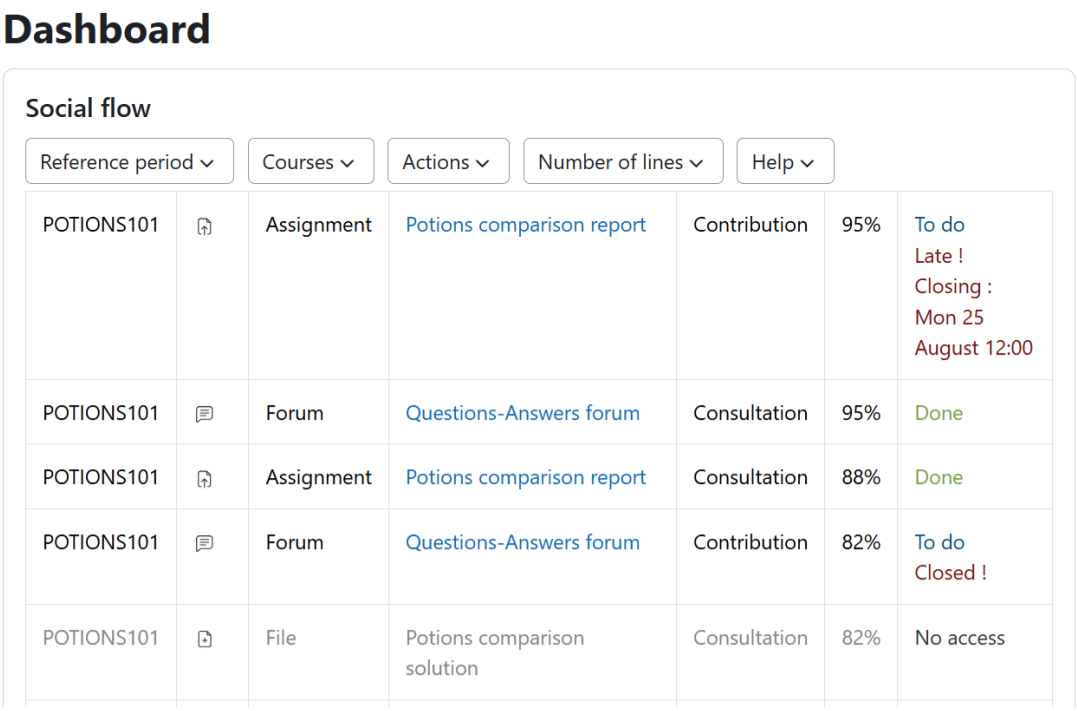



**Figure 21: Social flow block rendering with closing dates and restrictions information**

## 6.6 Informing students about the stored data and treatments

An information section has been integrated in a specific tab (see the capture at figure 22) to describe how to use the social flow block and how data is treated.

This text tries to face all the recommendations given in the subsection 3.3.3.

## 6.7 The final plugin versions to be tested

The final version of social flow plugins may be found on my github account :
https://github.com/zabellemotte/moodle-socialflow-log
https://github.com/zabellemotte/moodle-socialflow-block

Note that in this text, the queries are described only in MySQL but in final code version, queries have been implemented in the four database systems languages supported by Moodle, with the help of ChatGPT.

A function to import data from an existing default log table to the social flow log table has also been implemented in `cli/import.php`.

A view for teachers has also been implemented and to rapidly distinct student and teacher on Moodle dashboard, the `has_capability` function has been tested on `'mod/assign:submit'` permission (tip found in Moodle forums).

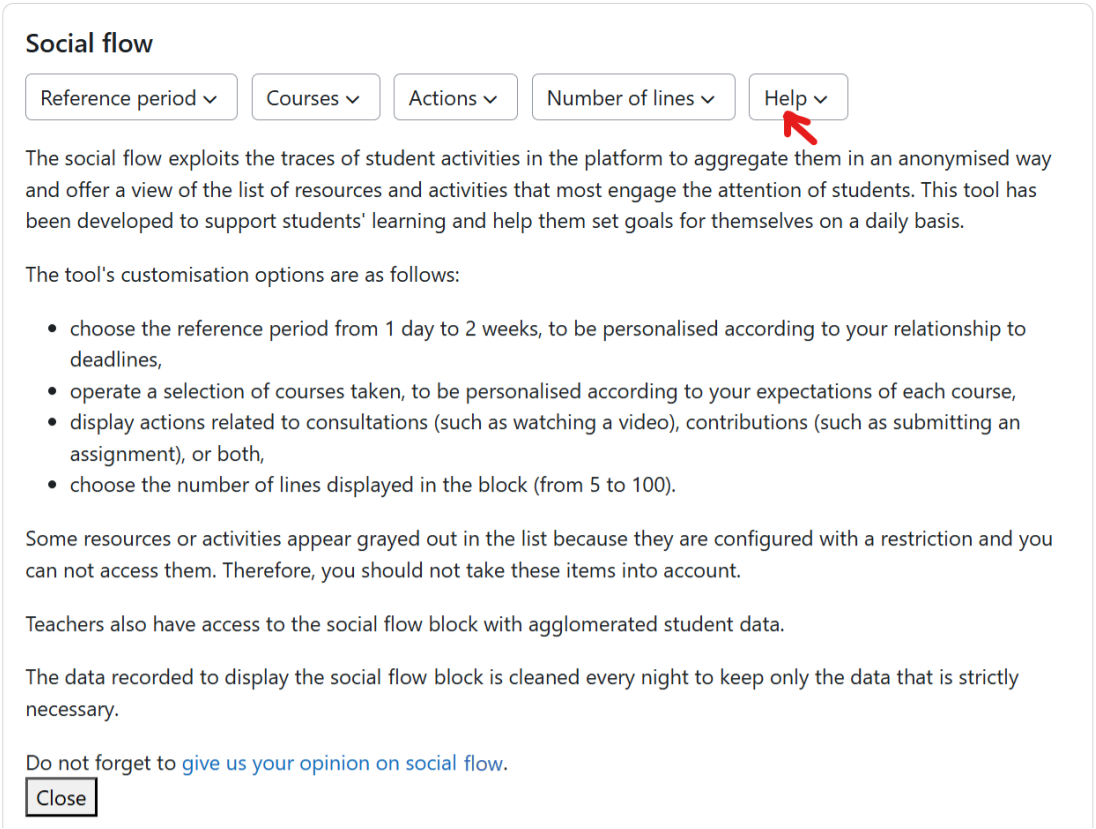



**Figure 22: Social flow block help information**

The plugins have been tested at small scale on Moodle 4 an 5 under MySQL and PostgreSQL database systems. All database queries have been tested on a copy of a big production Moodle. The plugins are ready to be tested on a larger scale. The log plugin may be installed silently and associated cron tasks may be ran manually at off-peak times to verify time execution. The cron tasks should take few minutes to execute, and then all data is ready to be shown in the social flow block. The log plugin may also be tested on a selection of courses, defined in the associated administration parameter.

These plugins should be completed with a report plugin that shows a social flow report in the course.

# 7 Evaluation form to get feedback about these plugins

An evaluation form has been built to collect data about the following points:

- Student profile : level of study, age bracket, gender, guiding preferences, ...
- Use of the social flow : preferred filters configuration, problems, ideal information frequency of updating, ...
- Usability of the tool : usability questions are based on the usability UEQ+ form, which

is grounded in usability research [15]
- Legal assessment : student attitude towards the use of their personnal data
- Added educational value : measures guiding effect on the different compounds of student engagement.

Due to a professional reorientation, I will not be able to evaluate the social flow with students, but do not hesitate to install the plugins and evaluate their interest with students.

The full evaluation form may be found in Appendix G.

# Conclusion

This report presents a built-in approach to extract data from the log table in an efficient way.

The social flow plugins development has been optimized with several approaches :

- storing only useful data in integer format in a dedicated log table,
- using intermediate tables that are updated by a cron task to avoid some elements to be regenerated at each student view load,
- make intermediate tables update fast while regenerating their complete data via a temporary table,
- when creating a table with counted elements, it is more efficient to store the concatenated list of the elements directly in an associated text field for fast retrieval,
- database queries optimization also helped a lot, with common tables expressions, intelligent joins and optimized indexes.

I hope the detailed description of optimization approaches used in the building of social flow plugins will inspire other learning analytics projects.

# Acknowledgments

The initial idea of the social flow learning analytics project was inspired of the readings proposed by Mikaël De Clercq, educational researcher.

My review of the existing analytics plugins has been helped by a discussion with Luiggi Sansonetti, Moodle expert.

The technical evolutions have been supported by discussions with Freddy Gridelet and Raoul De Guchteneere and this final report was improved by a careful review performed by Professor Charles Pecheur.

Thank you all for the questioning and enriching exchanges.

# References


[1] L. M. Nkomo, B. K. Daniel, and R. Butson, "Synthesis of student engagement with digital technologies: a systematic review of the literature," *International Journal of Educational Technology in Higher Education*, vol. 18, 2021.

[2] A. Namoun and A. Alshanqiti, "Predicting student performance using data mining and learning analytics techniques: A systematic literature review," *Applied Sciences*, vol. 11, no. 1, 2021.

[3] D. Ifenthaler and J. Y.-K. Yau, "Utilising learning analytics to support study success in higher education: a systematic review," *Educational Technology Research and Development*, vol. 68, no. 4, pp. 1961–1990, 2020.

[4] J. Wong, M. Baars, B. B. de Koning, T. van der Zee, D. Davis, M. Khalil, G.-J. Houben, and F. Paas, *Educational Theories and Learning Analytics: From Data to Knowledge*, pp. 3–25. Cham: Springer International Publishing, 2019.

[5] M. Siadaty, D. Gašević, and M. Hatala, "Associations between technological scaffolding and micro-level processes of self-regulated learning: A workplace study," *Computers in Human Behavior*, vol. 55, pp. 1007–1019, 2016.

[6] M. Siadaty, D. Gašević, and M. Hatala, "Measuring the impact of technological scaffolding interventions on micro-level processes of self-regulated workplace learning," *Computers in Human Behavior*, vol. 59, pp. 469–482, 2016.

[7] N. Milikić, D. Gašević, and J. Jovanović, "Measuring effects of technology-enabled mirroring scaffolds on self-regulated learning," *IEEE Transactions on Learning Technologies*, vol. 13, no. 1, pp. 150–163, 2020.

[8] W. Bonesire, "Suivi de l'étudiant : analyse et implémentation dans moodle," mémoire de master, Université catholique de Louvain, Louvain-la-Neuve, Belgique, juin 2017.

[9] E. Dalton, "Introduction to moodle learning analytics for designers." https://www.youtube.com/watch?v=kLZfFtYed7c, March 2017. Video, Moodle HQ.

[10] M. HQ, "Moodle lms 4.0 - new features overview." https://www.youtube.com/watch?v=6FVYgJl-79w, April 2022. Video, Moodle HQ.

[11] T. Dondorf, *Learning Analytics for Moodle: Facilitating the Adoption of Data Privacy Friendly Learning Analytics in Higher Education*. Dissertation, Rheinisch-Westfälische Technische Hochschule Aachen, Aachen, 2022.

[12] H. Drachsler and W. Greller, "Privacy and analytics – it's a delicate issue. a checklist for trusted learning analytics.," 04 2016.

[13] D. Ifenthaler and C. Schumacher, "Student perceptions of privacy principles for learning analytics," *Educational Technology Research and Development*, vol. 64, no. 5, pp. 923–938, 2016.

[14] J.-L. Hainaut, *Bases de données: concepts, utilisation et développement*. Paris, France: Dunod, 5 ed., july 2022. 5e édition.

[15] M. Schrepp and J. Thomaschewski, "Design and validation of a framework for the creation of user experience questionnaires," *International Journal of Interactive Multimedia and Artificial Intelligence*, vol. 5, pp. 88–95, 12/2019 2019.

# Appendices

# A Customprefs plugin code commented

Here is the associated `lib.php` file content with default preferences values :

```
1  $preferences['customprefs_optionchoice'] = array(
2          'null' => NULL_NOT_ALLOWED,
3          'default' => 0,
4          'type' => PARAM_ALPHA,
5          'choices' => array(0,1,2),
6          'permissioncallback' => [core_user::class, 'is_current_user'],
7      );
8
9      $preferences['customprefs_courseschoice'] = array(
10         'null' => NULL_ALLOWED,
11         'default' => null,
12         'type' => PARAM_RAW,
13         'permissioncallback' => [core_user::class, 'is_current_user'],
14     );
15
16     return $preferences;
```

The `block_customprefs.php` file fetches the chosen options at lines 9 and 28, and this data is stored with the `set_user_preference` function at lines 11 and 32. These preferences are retrieved with `get_user_preference` at lines 14 and 36. The html code to show the different options is displayed using the `html_writer` Moodle function :

```
1  function get_content() {
2          global $USER;
3          // get list of possible options
4          $options=[];
5          $options[0]=get_string('option1', 'block_customprefs');
6          $options[1]=get_string('option2', 'block_customprefs');
7          $options[2]=get_string('option3', 'block_customprefs');
8          // get list of chosen options
9          if( isset( $_POST['customprefs_optionchoice'] ) ){
10             $currentchoice = $_POST['customprefs_optionchoice'];
11             set_user_preference('customprefs_optionchoice', $currentchoice
                   );
12         }
13         else {
14             $currentchoice=get_user_preferences('customprefs_optionchoice'
                   );
15         }
16         // build user courses list
```

```
17          $courselistarray=[];
18          $allcourses=[];
19          if ($courses = enrol_get_my_courses()) {
20                  foreach ($courses as $course) {
21                      $courseid= $course->id;
22                      $courseshortname=$course->shortname;
23                      $courselistarray[$courseid] =$courseshortname;
24                      array_push($allcourses,$courseid);
25                  }
26           }
27          // get list of chosen courses or select all
28          if (isset( $_POST['customprefs_courseschoice'] ) )
29          {
30          $currentcourses = $_POST['customprefs_courseschoice'];
31          $currentcoursesstring= implode(',', $currentcourses);
32          set_user_preference('customprefs_courseschoice',
               $currentcoursesstring);
33          }
34          elseif ( !is_null( get_user_preferences('customprefs_courseschoice
               ') ) )
35          {
36          $currentcoursesstring = get_user_preferences('
               customprefs_courseschoice');
37          $currentcourses=explode(',', $currentcoursesstring);
38          }
39          else
40          $currentcourses=$allcourses;
41
42          $this->content->text="";
43          // button to open option selection form
44          $this->content->text .=html_writer::start_tag('div', array('class'
               =>'customprefs_optionselectopener','id'=>'
               customprefs_optionselectopener'));
45          $this->content->text .=get_string('osotext','block_customprefs');
46          $this->content->text .="<span class='expanded-icon'><i class='icon
               fa fa-chevron-down fa-fw'></i></span>";
47          $this->content->text .= html_writer::end_tag('div');
48            // button to open course selection form
49          $this->content->text .=html_writer::start_tag('div', array('class'
               =>'customprefs_courseselectopener','id'=>'
               customprefs_courseselectopener'));
50          $this->content->text .=get_string('csotext','block_customprefs');
51          $this->content->text .="<span class='expanded-icon'><i class='icon
               fa fa-chevron-down fa-fw'></i></span>";
52          $this->content->text .= html_writer::end_tag('div');
53
54          // block with option selection form
55          $this->content->text .=html_writer::start_tag('div', array('class'
               =>'customprefs_optionselectblock','id'=>'
```

```
        custompref s_optionselectblock'));
56      $this->content->text .= html_writer::start_tag('form', array('
        action' => '','method' => 'post','class' => '
        custompref s_optionselectform'));
57      // radio boxes with list of options
58      for ($i=0; $i<sizeof($options);$i++) {
59          if ($i==$currentchoice)
60              $checked='checked';
61          else $checked='';
62          $this->content->text .= "<input type='radio' name='
              custompref s_optionchoice' value='".$i."' ".$checked."/>";
63          $this->content->text .="<label>".$options[$i]."</label>";
64      }
65      // submit button for option selection form
66       $this->content->text .=html_writer::start_tag('button',array('
          class'=>'custompref s_optionselectbutton','onclick'=>'document.
          getElementById("custompref s_optionselectblock").submit()'));
67       $this->content->text .=get_string("save","block_custompref s");
68       $this->content->text .=html_writer::end_tag('button');
69      //$this->content->text .= html_writer::select($options, '
        custompref s_choice',$currentchoice,false, array('onchange'=>'
        this.form.submit()'));
70      $this->content->text .= html_writer::end_tag('form');
71      $this->content->text .= html_writer::end_tag('div');
72
73
74          // block with course selection form
75      $this->content->text .=html_writer::start_tag('div', array('class'
        =>'custompref s_courseselectblock', 'id'=>'
        custompref s_courseselectblock'));
76      $this->content->text .= html_writer::start_tag('form', array('
        action' => '','method' => 'post','class' => '
        custompref s_courseselectform'));
77      // checkboxes with list of courses
78      foreach ($courses as $course) {
79                  $courseid= $course->id;
80                  if (in_array($courseid,$currentcourses))
81                      $checked=true;
82                  else $checked=false;
83                  $courseshortname=$course->shortname;
84                  $this->content->text .= html_writer::checkbox('
                      custompref s_courseschoice[]',$courseid,$checked,
                      $courseshortname);
85      }
86       // submit button for course selection form
87       $this->content->text .=html_writer::start_tag('button',array('
          class'=>'custompref s_courseselectbutton','onclick'=>'document.
          getElementById("custompref s_courseselectblock").submit()'));
88       $this->content->text .=get_string("save","block_custompref s");
```

```
89          $this->content->text .=html_writer::end_tag('button');
90
91         $this->content->text .= html_writer::end_tag('form');
92          $this->content->text .= html_writer::end_tag('div');
93
94
95         // result of the different selections
96         $this->content->text.="<hr><div class='customprefs_option'> ".
              get_string('ocitext','block_customprefs').$options[
              $currentchoice]."</div>";
97         $this->content->text.="<div class='customprefs_course'>".
              get_string('ccitext','block_customprefs');
98         foreach ($currentcourses as $one){
99               $this->content->text.=$courselistarray[$one].", ";
100        }
101        $this->content->text=rtrim($this->content->text,', ');
102        $this->content->text.="</div>";
103
104         return $this->content;
105     }
106 }
```

Here is a javascript script that displays or hides the options based on buttons activations. When the first option button is activated, the course selection options are hidden and vice versa.

```
1  define(['jquery','block_customprefs/optionselection'], function($,
       myscript) {
2      return {
3          init: function() {
4              let mytogg0 = document.getElementById('
                  customprefs_optionselectopener');
5              let mytogg1 = document.getElementById('
                  customprefs_courseselectopener');
6              let myd0=document.getElementById('
                  customprefs_optionselectblock');
7              let myd1=document.getElementById('
                  customprefs_courseselectblock');
8              mytogg0.addEventListener('click', () => {
9                  if(getComputedStyle(myd0).display != 'none'){
10                     myd0.style.display = 'none';
11                 } else {
12                     myd1.style.display = 'none';
13                     myd0.style.display = 'block';
14                 }
15             });
16             mytogg1.addEventListener('click', () => {
17                 if(getComputedStyle(myd1).display != 'none'){
```

```
18                      myd1.style.display = 'none';
19                  } else {
20                      myd0.style.display = 'none';
21                      myd1.style.display = 'block';
22                  }
23              });
24          }
25      }
26  });
```

This file is stored in `amd/src` and has to be named for example `optionselection.js`. It has to be minified and placed in `amd/build` with the name `optionselection.min.js`.

# B Log store plugin files to store number of participants

To store number of course participants, our log plugin needs a new task in the `classes/task` folder, defined in `nbpa_task.php`, which extracts the list of courses where at least on action took place and stores this list with computed number of participants.

To update information in the intermediate nbpa table, I have noticed it was much faster to completely regenerate all the table data than to adjust table content. A new temporary table is therefore created to store the new data and the table content is finally replaced and temporary table dropped.

The main action depends on the `execute` function that

- creates a temporary table with exactly the same structure (lines 10 to 16),
- extracts the list of active courses with `$sql2` query at line 20,
- computes the number of participants for each of these courses with `$sql3` query at lines 31 to 41,
- stores it in the temporary table (at line 47),
- and finally replaces the existing `logstore_socialflow_nbpa` table with the temporary one (at lines 56-57).

```
1  public function execute() {
2        global $CFG, $DB;
3        // Include the XMLDB library
4        require_once($CFG->libdir . '/ddllib.php');
5        mtrace("Number of participants computations begin ...");
6        // Load the DDL manager and xmldb API.
7        $dbman = $DB->get_manager();
8
9        // temporary table to store number of active participants
             information
```

```
10        $temp_soci_nbpa = new \xmldb_table('logstore_socialflow_nbpa_temp')
              ;
11        if($dbman->table_exists($temp_soci_nbpa)) $dbman->drop_table(
              $temp_soci_nbpa);
12        $temp_soci_nbpa->add_field('id', XMLDB_TYPE_INTEGER, '10', null,
              XMLDB_NOTNULL, XMLDB_SEQUENCE, null);
13        $temp_soci_nbpa->add_field('courseid', XMLDB_TYPE_INTEGER, '10',
              null, XMLDB_NOTNULL, null, null);
14        $temp_soci_nbpa->add_field('nbpa', XMLDB_TYPE_INTEGER, '10', null,
              XMLDB_NOTNULL, null, null);
15        $temp_soci_nbpa->add_key('primary', XMLDB_KEY_PRIMARY, ['id']);
16        $dbman->create_table($temp_soci_nbpa);
17        if (!$temp_soci_nbpa) die("temporary nbpa table creation impossible
              ");
18
19        // get list of all courses in log table
20        $sql2 ="SELECT DISTINCT(courseid) FROM mdl_logstore_socialflow_log"
              ;
21        $result2 = $DB->get_records_sql($sql2);
22        if (!$result2)  die("no data to treat");
23        $now=time();
24        // get number of active participants in the course
25        // number of active participants needs a heavy computation
26        // (verifications : active user, active enrollment method, active
              enrollment, student role)
27        // it is sufficient to compute it 1 time a day, so it is stored in
              a dedicated table
28        // that is updated via the nbpa crontask
29        foreach ($result2 as $row) {
30              $courseid=$row->courseid;
31              $sql3 ="SELECT COUNT(DISTINCT(u.id)) AS nbpa
32                                    FROM {user} u
33                                    INNER JOIN {user_enrollments} ue ON ue.
                                          userid = u.id
34                                    INNER JOIN {enrol} e ON e.id = ue.enrolid
35                                    INNER JOIN {role_assignments} ra ON ra.
                                          userid = u.id
36                                    INNER JOIN {context} ct ON ct.id = ra.
                                          contextid AND ct.contextlevel = 50
37                                    INNER JOIN {course} c ON c.id = ct.
                                          instanceid AND e.courseid = c.id
38                                    INNER JOIN {role} r ON r.id = ra.roleid
                                          AND r.shortname = 'student'
39                                    WHERE e.status = 0 AND u.suspended = 0 AND
                                           u.deleted = 0 AND (ue.timeend = 0 OR
                                          ue.timeend > $now)
40                                    AND ue.status = 0 AND c.id = $courseid";
41              $result3 = $DB->get_record_sql($sql3);
42              if(!$result3) die("nbpa computation impossible");
```

```
43              $nbpa=$result3->nbpa;
44              $data= new \stdClass();
45              $data->courseid=$courseid;
46              $data->nbpa=$nbpa;
47              $result4 = $DB->insert_record('logstore_socialflow_nbpa_temp'
                    ,$data);
48              if (!$result4) die("insert new nbpa data impossible");
49      }
50      mtrace("Computations completed, data replacement begins ...");
51
52      // nbpa table replacement and temporary table dropping
53      // no generic truncate function in moodle data api,
54      // so it is faster to drop table and recreate it, rather than
            deleting all records
55      $soci_nbpa = new \xmldb_table('logstore_socialflow_nbpa');
56      $dbman->rename_table($soci_nbpa,'logstore_socialflow_nbpa_old');
57      $dbman->rename_table($temp_soci_nbpa,'logstore_socialflow_nbpa');
58      $soci_nbpa_old = new \xmldb_table('logstore_socialflow_nbpa_old');
59      $dbman->drop_table($soci_nbpa_old);
60      mtrace("Nbpa information updated");
61  }
```

# C Log store plugin files to compute hits

The `classes/task/hits_task.php` function performs the following tasks :

- temporary table creation to store the new hits data at line 10 to 20;
- hit computation for each action is directly retrieved and stored in the temporary table with an imbricated query (lines 36 to 42);
- finally, we replace the hits table content with the temporary table content (lines 52 and 53).

```
1  public function execute() {
2       global $CFG, $DB;
3       // Include the XMLDB library
4       require_once($CFG->libdir . '/ddllib.php');
5       mtrace("Hits table update begins ...");
6       // Load the DDL manager and xmldb API.
7       $dbman = $DB->get_manager();
8
9       // create temporary table to store hits information
10      $temp_soci_hits = new \xmldb_table('logstore_socialflow_hits_temp')
            ;
11      if($dbman->table_exists($temp_soci_hits)) $dbman->drop_table(
            $temp_soci_hits);
12      $temp_soci_hits->add_field('id', XMLDB_TYPE_INTEGER, '10', null,
```

```
            XMLDB_NOTNULL, XMLDB_SEQUENCE, null);
13          $temp_soci_hits->add_field('eventid', XMLDB_TYPE_INTEGER, '10',
               null, XMLDB_NOTNULL, null, null);
14          $temp_soci_hits->add_field('courseid', XMLDB_TYPE_INTEGER, '10',
               null, XMLDB_NOTNULL, null, null);
15          $temp_soci_hits->add_field('contextid', XMLDB_TYPE_INTEGER, '10',
               null, XMLDB_NOTNULL, null, null);
16          $temp_soci_hits->add_field('nbhits', XMLDB_TYPE_INTEGER, '10', null
               , XMLDB_NOTNULL, null, null);
17          $temp_soci_hits->add_field('lasttime', XMLDB_TYPE_INTEGER, '10',
               null, XMLDB_NOTNULL, null, null);
18          $temp_soci_hits->add_field('userids', XMLDB_TYPE_TEXT, null, null,
               XMLDB_NOTNULL, null, null);
19          $temp_soci_hits->add_key('primary', XMLDB_KEY_PRIMARY, ['id']);
20          $dbman->create_table($temp_soci_hits);
21          if (!$temp_soci_hits) die("temporary hits table creation impossible
               ");
22
23          $loglifetime = (int) get_config('logstore_socialflow', 'loglifetime
               ');
24
25          if (empty($loglifetime) || $loglifetime < 0) {
26                $loglifetime=14;
27          }
28
29          $loglifetime = time() - ($loglifetime * 3600 * 24); // Value in
               days.
30
31          // hit query depend on the SGBD, thanks to Chatgpt for the
               conversion ;-)
32          // storing userids in one table field is the fastest way to store
               and access this information
33          // as far as no select action is made on this field this does not
               raise performance problem
34          $dbtype=$CFG->dbtype;
35
36          $sql1 = "INSERT INTO {logstore_socialflow_hits_temp} (contextid,
               eventid, courseid, nbhits, lasttime, userids)
37                                       SELECT log.contextid, log.eventid,
                                             log.courseid,
38                                           COUNT(DISTINCT log.userid) AS
                                             nbhits,
39                                           MAX(log.timecreated) AS
                                             lasttime,
40                                           GROUP_CONCAT(DISTINCT log.
                                             userid ORDER BY log.userid)
                                              AS userids
41                                       FROM {logstore_socialflow_log} log
42                                       GROUP BY log.contextid, log.
```

```
                                                eventid, log.courseid";
43
44          $result1=$DB->execute($sql1);
45          if (!$result1)  die("no data to treat");
46
47
48      mtrace("information stored in temporary tables, data replacement
            begins ...");
49      // hits table replacement and temporary table dropping
50      // no generic truncate function in moodle data api, so it is faster
            to drop table and recreate it, rather than deleting all records
51      $soci_hits = new \xmldb_table('logstore_socialflow_hits');
52      $dbman->rename_table($soci_hits,'logstore_socialflow_hits_old');
53      $dbman->rename_table($temp_soci_hits,'logstore_socialflow_hits');
54      $soci_hits_old = new \xmldb_table('logstore_socialflow_hits_old');
55      $dbman->drop_table($soci_hits_old);
56      mtrace("Hits information updated");
57  }
```

# D Fetching all information for social flow view

All the social flow view needed data is fetched in the `block_socialflow.php` main function as follows :

- the course title is retrieved at line 9 in `$shortname` variable, with a link to the course based on the `$courseid`;
- the event action type ('consult' or 'contrib'), stored in variable `$actiontype` and is associated to the pertinent lang string in lines 10-15 ;
- the action frequency is stored in `$freq` at line 19 and formatted as a percentage at line 21;
- the module name (assign, forum, ...) is extracted in the correct language at line 23;
- the module icon is stored in `$moduleicon` variable, which appears to contain the `img` tag, that is retrieved at line 24 on the basis of module name in `modulename` variable;
- the module instance name (name of the activity defined by teacher) is stored in variable `$title` at line 30 based on a single query on `modulename` table at line 27; it will be presented in a tag with a direct link to the activity thanks to the `$moduleid`;
- the 'Done, Not done' indicator for the users, stored in `$done` variable that is generated at lines 31-37.

All these data is formatted in one array line at line 38.

```
1   function get_content() {
2               // … (code omitted)
3                   $this->content->text.="<div id='socialflowdata'> <table
                        id='socialflow_table' class='generaltable'>";
```

```
4               foreach ($result as $row) {
5                   $hitid=$row->hitid;
6                   $contextid=$row->contextid;
7                   $eventid=$row->eventid;
8                   $courseid=$row->courseid;
9                   $shortname=$courselistarray[$courseid];
10                  $actiontype=$row->actiontype;
11                  if ($actiontype=='contrib') {
12                      $actiontypelang=get_string('contrib','
                            block_socialflow');
13                  } else {
14                      $actiontypelang=get_string('consult','
                            block_socialflow');
15                  }
16                  $instanceid=$row->instanceid;
17                  $instance=$row->instance;
18                  $modulename=$row->name;
19                  $freq=$row->freq;
20                          // express frequency in percent
21                          $freqp=round(100*$freq,0,PHP_ROUND_HALF_UP);
22                  $moduletable=$row->moduletable;
23                  $modulenameinlang=get_string("modulename","$moduletable")
                        ;
24                  $moduleicon = $OUTPUT->pix_icon('icon', $modulenameinlang
                        , $modulename, array('class' => 'iconlarge
                        activityicon'));
25                  $moduleurl=new moodle_url('/mod/'.$modulename."/view.php"
                        , array('id' => $instanceid));
26                  // get the activity or resource title
27                  $sql5="SELECT name FROM {".$modulename."} WHERE id=
                        $instance";
28                  $result5= $DB->get_record_sql($sql5);
29                  if (!$result5) continue;
30                  $title=$result5->name;
31                  $userids_string=$row->userids;
32                  $userids = explode(',', $userids_string);
33                  if (in_array($cuserid, $userids)){
34                      $done='<span class="socialflow_todo">'.get_string('
                            todo','block_socialflow').'</span>';
35                  } else {
36                      $done='<span class="socialflow_done">'.get_string('
                            done','block_socialflow').'</span>';
37                  }
38                          $this->content->text.="<tr><td>$shortname</td><td
                            >$moduleicon</td><td>$modulenameinlang</td> <
                            td><a href='".$moduleurl."'>$title</a></td><td
                            >$actiontypelang </td> <td>$freqp\%<td> $done
                            </td></tr>";
39              }
```

```
40 $this->content->text.="</table></div>";
41 // … (code omitted)
42 }
```

# E Closing date table building

The new table `logstore_socialflow_closing` is created by the log plugin to store closing dates information and the tasks that builds the hits table also fills this table with as much lines as the hits table. The `execute` function of the `hit_task.php` file is augmented with the lines below.

Two temporary tables are used (lines 1-20) because it is impossible to proceed an `INSERT` action on a table while also proceeding to a `SELECT` action on this table and the different steps are those :

- the `$sql3` query at line 32 retrieves all tracked events that have closing dates;
- for each event with closing date, the `$sql4` query (lines 41-47) extracts hits lines and retrieves the associated closing dates in the right table and right field, and the result is directly stored in the temporary closing date table;
- the `$sql5` query (lines 60-63) selects all lines that are not referred in the temporary closing date table and stores it in the second temporary table with an infinite value for the associated closing date;
- the last lines (66-80) proceed closing date table content replacement and drop temporary tables.

```
1  //create temporary table to store closing dates information
2  $temp_soci_closing = new \xmldb_table('logstore_socialflow_closing_temp')
       ;
3  if($dbman->table_exists($temp_soci_closing)) $dbman->drop_table(
       $temp_soci_closing);
4  $temp_soci_closing->add_field('id', XMLDB_TYPE_INTEGER, '10', null,
       XMLDB_NOTNULL, XMLDB_SEQUENCE, null);
5  $temp_soci_closing->add_field('hitid', XMLDB_TYPE_INTEGER, '10', null,
       XMLDB_NOTNULL, null, null);
6  $temp_soci_closing->add_field('closingdate', XMLDB_TYPE_INTEGER, '10',
       null, XMLDB_NOTNULL, null, null);
7  $temp_soci_closing->add_key('primary', XMLDB_KEY_PRIMARY, ['id']);
8  $dbman->create_table($temp_soci_closing);
9  if (!$temp_soci_closing) die("temporary closing table creation impossible
       ");
10
11 // create second temporary table to store closing date information
12 // while avoiding an imbricated query with SELECT and INSERT on the same
       table
13 $temp_soci_closing2 = new \xmldb_table('logstore_socialflow_closing_temp2
```

```
    ');
if($dbman->table_exists($temp_soci_closing2)) $dbman->drop_table(
    $temp_soci_closing2);
$temp_soci_closing2->add_field('id', XMLDB_TYPE_INTEGER, '10', null,
    XMLDB_NOTNULL, XMLDB_SEQUENCE, null);
$temp_soci_closing2->add_field('hitid', XMLDB_TYPE_INTEGER, '10', null,
    XMLDB_NOTNULL, null, null);
$temp_soci_closing2->add_field('closingdate', XMLDB_TYPE_INTEGER, '10',
    null, XMLDB_NOTNULL, null, null);
$temp_soci_closing2->add_key('primary', XMLDB_KEY_PRIMARY, ['id']);
$dbman->create_table($temp_soci_closing2);
if (!$temp_soci_closing2) die("temporary closing table 2 creation
    impossible");


$loglifetime = (int) get_config('logstore_socialflow', 'loglifetime');


if (empty($loglifetime) || $loglifetime < 0) {
      $loglifetime=14;
}

$loglifetime = time() - ($loglifetime * 3600 * 24); // Value in days.

$sql3="SELECT id, moduletable, closingdatefield FROM {
    logstore_socialflow_evts} WHERE hasclosingdate>0";
$result3=$DB->get_records_sql($sql3);
if ($result3) {
      foreach ($result3 as $row){
            $eventid=$row->id;
            $moduletable=$row->moduletable;
            if ($dbman->table_exists($moduletable)) {
                  $closingdatefield=$row->closingdatefield;
                   // in core plugins, the default value for closing date
                       field is zero but in additional plugins, default
                      value is sometimes null
                   $sql4="INSERT INTO {logstore_socialflow_closing_temp}
                      (hitid,closingdate)
                                    SELECT DISTINCT h.id AS hitid,mt.".
                                       $closingdatefield." AS
                                       closingdate FROM {
                                       logstore_socialflow_hits} h
                                       INNER JOIN {
                                          logstore_socialflow_evts} evts
                                           ON h.eventid =".$eventid.
                                         " INNER JOIN {context} c ON h.
                                            contextid = c.id
                                         INNER JOIN {course_modules} cm
                                            ON c.instanceid = cm.id
```

```
                                                INNER JOIN {".$moduletable."}
                                                   mt ON cm.instance=mt.id
                                                WHERE (mt.".$closingdatefield."
                                                    IS NOT NULL) AND (mt.".
                                                   $closingdatefield." > 0)";
                                $result4=$DB->execute($sql4);
                                if (!$result4)  die("error on temp closing table
                                    insert");

            }
        }
}
// To make it possible to exclude actions while closingdate is passed,
// I need to have a closing date defined for each event in the hits table
    .
// The queries below complete the closing table so that all hits without
    closing date
// has an infinite value for closing date so 9999999999.
// This operations requires 2 temporary tables because it is impossible
    to play
// an imbricated query with SELECT and INSERT on the same table
$sql5="INSERT INTO {logstore_socialflow_closing_temp2} (hitid,
    closingdate)
                                SELECT h.id AS hitid, 9999999999 AS
                                   closingdate
                                FROM {logstore_socialflow_hits} h
                                WHERE h.id NOT IN (SELECT hitid FROM {
                                   logstore_socialflow_closing_temp})";
$result5=$DB->execute($sql5);
if (!$result5)  die("error on temp closing table insert");
$sql6="INSERT INTO {logstore_socialflow_closing_temp} (hitid, closingdate
    )
                                SELECT hitid, closingdate FROM {
                                   logstore_socialflow_closing_temp2}";
$result6=$DB->execute($sql6);
if (!$result6)  die("error on temp closing table insert");

mtrace("information stored in temporary tables, data replacement begins
    ...");
// closing table replacement and temporary tables dropping
// no generic truncate function in moodle data api, so it is faster to
    drop table and recreate it, rather than deleting all records
$soci_closing = new \xmldb_table('logstore_socialflow_closing');
$dbman->rename_table($soci_closing,'logstore_socialflow_closing_old');
$dbman->rename_table($temp_soci_closing,'logstore_socialflow_closing');
$soci_closing_old = new \xmldb_table('logstore_socialflow_closing_old');
$dbman->drop_table($soci_closing_old);
$soci_closing_temp2 = new \xmldb_table('logstore_socialflow_closing_temp2
    ');
```

```
80 $dbman->drop_table($soci_closing_temp2);
```

# F Social flow block code to retrieve closing date

The closing dates information are retrieved in social flow block with the lines below.

The main steps of this implementation are the following:

- the closing date timestamp is retrieved at lines 15-27 and the infinite timestamp are converted in null value;
- the late date timestamp is retrieved at lines 29 to 44 and the 0 values are converted in null values;
- when the late data and closing date are defined, a subtle imbrication of conditions (lines 45-52) enables us to show
  - an information message with the late date when not passed;
  - a critical information message when the late date is passed and the closing date is in the future;
  - a more critical information message when the closing date is passed (and the event will remain in the social flow for a 48h delay);
- if there is only a closing date, the message changes when the closing date is in the future or in the past (lines 54-57).

```
1  // When action not done, a custom comment is added to done/to do
       information
2  // when a closing date or late date exists.
3  // Some activities (assign and forum) allow a late date, that does not
       close the activity,
4  // but activities after the late date are noticed as late.
5  // Latedate or closingdate are integrated to the comment when not in the
       past.
6  // When latedate and closingdate are in the past, they are not shown and
       activity is noticed as closed.
7  // Closed activities are excluded from social flow when 48 hours passed
       after closing date.
8  // This is done to avoid student despondency, while viewing closed
       activities for 1 or 2 weeks.
9  if ($isdone==0) {
10     $comment="";
11     $hasclosingdate=$row->hasclosingdate;
12     $haslatesubmit=$row->haslatesubmit;
13     if ($hasclosingdate>0){
14           // get the closing date information
15           $sql6="SELECT closingdate FROM {logstore_socialflow_closing}
                  WHERE hitid=".$hitid;
16           $result6= $DB->get_record_sql($sql6);
```

```
17            // error on $result6 should not arise has far as all hits have
                 a closingdate completed, but let make sure it works whatever
18            if ($result6) {
19                  $closingdatet=$result6->closingdate;
20                  // when closing date is undefined, the closingdate value
                       is 9999999999
21                  if ($closingdatet==9999999999) {
22                        $closingdatet=null;
23                  }
24                  else {
25                        $closingdate = strftime('%a %d %B %H:%M',
                             $closingdatet);
26                  }
27            }
28            if($haslatesubmit>0){
29                 $latedatefield=$row->latedatefield;
30                 $sql7="SELECT ".$latedatefield." FROM {".$moduletable. "}
                      WHERE id=".$instance;
31                 // in core plugins, the default value for late date field
                      is 0
32                 //but in some additional plugins default value is null
33                  $result7= $DB->get_record_sql($sql7);
34                  if (($result7)&&($result7!==null)) {
35                        $latedatet=$result7->$latedatefield;
36                        // in additional plugins, it may arise that
                             undefined latedate correspond to value 0
37                        if($latedatet==0) {
38                              $latedatet=null;
39                        } else {
40                        $latedate = strftime('%a %d %B %H:%M', $latedatet);
41                        }
42                    }
43                    // if no late date defined, $latedate and $latedatet
                        are not defined
44            }
45            if (isset($latedate)){
46                  if ($now<$latedatet){
47                        $comment="<br/><span class='socialflow_comment'>".
                             get_string('limitdate','block_socialflow')."<br
                             />".$latedate."</span>";
48                  } elseif ($now<$closingdatet){
49                        $comment="<br/><span class='socialflow_comment
                             socialflow_critical'>".get_string('latedate','
                             block_socialflow')."<br/>".$closingdate."</span>
                             ";
50                  } else {
51                        $comment="<br/><span class='socialflow_comment
                             socialflow_critical'>".get_string('closed','
                             block_socialflow')."</span>";
```

```
52                  }
53            } elseif (isset($closingdate)) {
54                  if($now<$closingdatet) {
55                        $comment="<br/><span class='socialflow_comment'>".
                             get_string('limitdate','block_socialflow')."<br/>
                             ".$closingdate."</span>";
56                  } else {
57                        $comment="<br/><span class='socialflow_comment
                             socialflow_critical'>".get_string('closed','
                             block_socialflow')."</span>";
58                  }
59            } else $comment="";
60      }
61      $done='<span class="socialflow_todo">'.get_string('todo','
           block_socialflow').'</span>'.$comment;
62  } else {
63      $done='<span class="socialflow_done">'.get_string('done','
           block_socialflow').'</span>';
64  }
```

# G Evaluation form

**Your experience of using Moodle's integrated social flow block**
This evaluation form is divided into 5 parts and mainly comprises questions with options and statements associated with a rate of agreement. You are free to fill in open-ended comments to clarify your answers.

## You profile

### 1.What is your level of study?

You can select several options if you are halfway between two years of study.
(Mandatory.Multiple choice.)
bac 1
bac 2
bac 3
master 1
master 2

### 2.What age bracket do you fall into?

(Mandatory. Unique choice.)
18-20
20-22
22-25

25-30
30-40
+40

**3.What gender are you ?**

(Mandatory. Unique choice.)
woman
men
non binary

**4.I would like to draw your attention to a particular feature of my profile which I feel is important for analyzing my answers (dyslexia, disability, etc.).**

(Mandatory. Unique choice.)
yes
no

**(If answer yes to question 4)**
**5.Describe the elements of your profile that we need to take into account when analyzing your answers.**

(Open answer)

**6.Specify your level of agreement with the following statements**

(Mandatory. Likert.)
Disagree - Rather disagree - Indifferent - Rather agree - Agree
I appreciate having support tools to guide my learning.
I feel comfortable organizing my work and setting my priorities.
I tend to work at the last minute.
I'm more motivated when I feel part of a learning community.

## Your use of the social flow

In this section, you will be asked to describe the configuration of the social flow block that you found most practical, and you will have the opportunity to describe other configurations used at the end of the questionnaire.

**7.Have you used the social flow block for at least 3 days?**

(Mandatory. Unique choice)
yes
no

**(If answer no to question 7)**
**8.Please describe the reasons why you are not using the social flow block.**

(Open answer)

**9.The social flow block allows you to filter the most frequent actions by reference period. Which configuration did you find most practical?**

(Mandatory.Unique Choice)
Reference period = 2 weeks
Reference period = 1 week
Reference period = 3 days
Reference period = 1 day

**10.Note on your choice of reference period filter (optional)**

(Open answer)

**11.You can also select the courses targeted by the social flow block. Describe the criteria that determined your course selection.**

(Mandatory. Open answer)

**12.The social flow block also lets you filter the type of action displayed: consultation, contribution or both. Which configuration did you find most practical?**

(Mandatory. Unique choice)
Consultation
Contribution
Both

**13.Note on your choice for the type of action (optional)**

(Open answer)

**14.You were finally able to select the number of lines displayed in the social flow. Which configuration did you find most practical?**

(Mandatory. Unique choice)
show 5 lines
show 10 lines
show 15 lines
show 20 lines
show 30 lines
show 50 lines
show 100 lines

**15.Comments on the number of lines displayed in the social flow block (optional)**

(Open text)

**16.Do you think there should be other filter options for displaying data from the social flow ?**

(Mandatory.Unique choice)
yes
no

**(If answer yes to question 16)**
**17.What other filter(s) would you suggest we integrate?**

(Open answer)

**18.If you found it interesting to use the social flow block with other filter configurations, please describe which other configuration(s) you used and why (optional).**

(Open text)

**19.Have you noticed any problems with the quality of the information in the social flow (incorrect, missing or erroneous information, etc.)?**

(Mandatory. Unique choice)
yes
no

**(If answer yes to question 19)**
**20.Describe the type of irrelevant information you observed in the social flow block**

(Open text)

**21.Have you encountered any problems using the social flow block (e.g. slowness)?**

(Mandatory. Unique choice)
yes
no

**(If answer yes to question 21)**
**22.Describe in as much detail as possible the difficulties encountered, when and in what configuration of the block the problems arose.**

(Open answer)

**23.The social flow data is updated every night. But this data could be updated more or less often. What do you think is the ideal frequency for updating the information in the social flow so that the tool is motivating without being stressful?**

(Mandatory. Unique choice)
every hour
2 times a day
1 time day (like now)
3 times a week
1 time a week

**24.Do you think it would be interesting to offer a social flow report within each course, in addition to the general social flow block on the dashboard?**

(Mandatory. Unique choice)
yes
no

**(If answer yes to question 24)**
**25.Comment on the benefits of including a social flow report in each course (optional).**

(Open answer)

**26.Other suggestion(s) regarding the display of social flow data (optional)**

(Open answer)

## Usability of the tool

**27.When you use a new tool to support your learning, how important are the following issues to you in general?**

(Unique choice. Likert)
Not important - Not very important - Indifferent - Important - Very important
The tool must be of benefit to me.
The tool must be easy to use.
The tool must be customizable.
The tool must be stimulating.
The tool must be intuitive.
The tool must give me confidence in the way it handles my personal data.

**28.How much do you agree with the following statements?**

(Unique choice. Likert)
Disagree - Rather disagree - Indifferent - Rather agree - Agree

I find the possibility of using the social flow beneficial.
I find it easy to use the social flow.
I find that the social flow is a customizable tool.
I find working with the social flow stimulating.
I find the tool intuitive to use.
As far as the use of my personal information and data is concerned, the product is reliable.

## Legal assessment

### 29.How much do you agree with this statement?

(Unique choice. Likert)
Disagree - Rather disagree - Indifferent - Rather agree - Agree
I don’t think that using student data to generate the social flow report requires any particular legal precautions.

### 30.Justify your answer (optional)

(Open answer)

## Added educational value

### 31.How much do you agree with the following statements?

(Unique choice. Likert)
Disagree - Rather disagree - Indifferent - Rather agree - Agree
The social flow makes it easier for me to set goals for my work.
The social flow helps me to organize myself and set my priorities.
The social flow ensures that I don’t miss any important deadlines.
The social flow allows me to feel more integrated into the university community.
The social flow stresses me out because there’s too much to do every day.

### 32.Comments on the educational value or otherwise of the tool (optional)

(Open answer)

### 33.Sum up your opinion of the social flow in a few lines.

(Mandatory.Open answer)