Corrector, a web interface for practice session with immediate feedback
Rapport A/377/CRI

Fabien Coelho
fabien.coelho@ensmp.fr
Centre de Recherche en Informatique, École des mines de Paris
35, rue Saint-Honoré, 77305 Fontainebleau cedex, France

19 avril 2006

Résumé

Corrector est une application Web client-serveur qui donne à l'étudiant en séance pratique des retours immédiats en validant automatiquement les réponses proposées. L'enseignant n'a plus besoin d'intervenir que pour résoudre les problèmes de compréhension rencontrés avec des explications limpides. Corrector est particulièrement adapté à l'enseignement du langage SQL utilisé dans les bases de données relationnelles. En effet, certaines questions admettent de nombreuses requêtes SQL équivalentes comme réponses valides. La correction automatique résout ce problème en comparant le résultat de l'exécution de la requête à celui d'une requête référence. D'autres modes de correction sont proposés qui abordent la modification des données ou du schéma de la base, utilisent des preuves de succès cryptographiques ou des expressions régulières. Cet article décrit l'interface et son fonctionnement, puis rapporte et discute le résultat de deux ans d'expériences auprès de plus de 150 étudiants avec Corrector.

Abstract

Corrector is a client-server web application which provides immediate feedback to students during practice sessions by automatically validating the proposed answers. The teacher needs only focus on solving comprehension problems with clever explanations. Corrector is especially well fitted to help teaching SQL, the language used in relational database systems. Indeed, some questions admit many equivalent SQL queries as valid answers. The automatic correction solves this issue by comparing the result of the student query execution to the result of a reference query. Other correction modes are available that deal with modifying data contents or structures, use cryptographic

```sql
-- number of books per collection
SELECT cnom, COUNT(*) AS nb
FROM collection
JOIN oeuvre USING(cid)
GROUP BY cnom
UNION
SELECT cnom, 0 AS nb
FROM collection
LEFT JOIN oeuvre USING(cid)
WHERE titre IS NULL
ORDER BY nb DESC, cnom ASC;
```

```sql
-- number of books per collection
-- with COUNT(NULL) = 0 extension
SELECT cnom, COUNT(fid)
FROM collection
LEFT JOIN oeuvre USING(pid)
GROUP BY cnom
ORDER BY COUNT(fid) DESC, cnom ASC;
```

Figure 1: Two equivalent queries with PostgreSQL

proof-of-success tokens or regular expressions. This paper describes the interface and its internals, then reports and discusses the result of two years of experiments with Corrector and more than 150 students.

1 Introduction

Teaching computer science becomes harder as self-motivation seems to drive less students to such technical fields. The western society highlights successes quickly acquired through business or arts, often making science and technical subjects not so attractive to students. Improving the attractivity of our courses is thus necessary.

Relational database [3] and SQL [4], the *Structured Query Language* designed to manipulate them, is such a technical subject with an algebraic background which requires abstraction efforts from the students. SQL allows data to be selected, inserted, updated or deleted into relational databases, as well as structures to be created, dropped or altered and access rights to be granted or revoked.

Teaching SQL involves presenting the underlying relational model and the language syntax. A simple traditional class organization is to alternate theoretical blackboard (or white board) presentations with practice sessions hands-on to solve problems with the tools discussed. During practice sessions, students are asked to develop queries to answer variously challenging questions on a database. In the relational model, many queries are equivalent. For instance, Figure 1 shows two widely differing queries that computes

the number of books for all collections. Moreover is it often unclear to a student whether a query gives the expected answer, especially for advanced questions. Thus either the students ask for a direct validation from the teacher during the session, what is time consuming and makes less time available for helping them understand the subtilities involved, or the validation is postponed to a later stage but then students may miss the point and only be told so much later, after the teacher has processed hundreds of answers which look right but may not be valid.

Corrector is a web interface based on the PostgreSQL database. It validates proposed answer interactively, thus students do not need any help from the teacher for this purpose. All queries are executed on the database by the tool. For selects, the result is compared to the result of a reference query. For inserts, updates, deletes, creates, drops and alters, transactional database features are used to rollback the student changes at the end of the validation. The validation itself is performed by specially tailored queries to check whether the intended transformations were performed, before cancelling them. Performances issues are also addressed, as a badly crafted student request can easily submerge the database or web server. This tool enables teachers to focus on comprehension problems encountered by students, and students to have direct feedback about their progress during the sessions.

Section 2 briefly describes the choice of tool and exercises for our course. Sections 3 and 4 outlines the web interface and its internal organization. Section 5 details the automatic correction modes available. Section 6 discussed various aspects of the interface before leading to the conclusion in Section 7.

2 Software and Exercises

The PostgreSQL [1] free software is an object-relational database management system, with many interfaces and high extensibility. It includes advanced features uniquely available for a software in its price category, such as transaction with MVCC, roles, triggers, integrity constraints, and an SQL implementation very close to the standard. It is *the* database of choice for a serious course.

Having a good exercise base on top of a good tool is also paramount to help students learn quickly and effectively new concepts. As far as SQL is concerned, this requires a set of questions to be answered which cover the various aspects of the language and the difficulty of which is progressive. However, this is not enough.

Figure 2 shows the database of my comics used for practicing SQL. This very deep and introspective subject helps to motivate students. Indeed, although queries to answer *what is the average salary in the marketing de-*

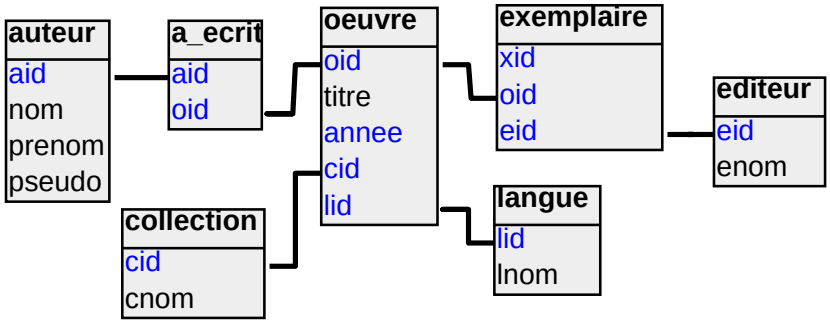

Figure 2: Comics Database Outline

partment and *what is the average number of pages of comics published by Delcourt editions* may have both the very same structure, the later question is more attractive to students, even at a late stage of their studies.

The first practices use a simplified view which collapses actual books *(exemplaire)* that I own and conceptual books *(œuvre)* that I know exists but that are not in my library. Such distinction is only useful for advanced questions with SQL which require relational operators such as union, intersection or difference.

3 Web Interface

A simple prototype web client-server architecture based on standard free (open source and inexpensive) softwares and interfaces such pas HTML [5], Apache [2], CGI [6], Perl [10, 11], PostgreSQL [1] has been developed.

Figure 3 outlines the student and teacher interfaces: each page is generated by a script. It presents a dynamically generated contents based on the user identity and additional parameters provided by the urls followed by the user. Users are authenticated through the standard HTTP basic login password scheme by apache, and the identity is passed to all pages.

Students are restricted to the upper interface. They can access their list of active practice sessions. Each session points to its list of questions for which the student can suggest answers. The list of all questions can be detailed with the answer already provided by the student.

The teacher and administrator interface includes many more pages to manage the application. New users can be added and managed in classes. New exercises can be created, new sessions and questions added to an exercise. The management of a session allows to define its schedule and status: when is the access is open or closed, whether it is visible, and so on. All data can be modified if the user has enough privileges in the application. Finally, a teacher can check students answers and alter the points automatically

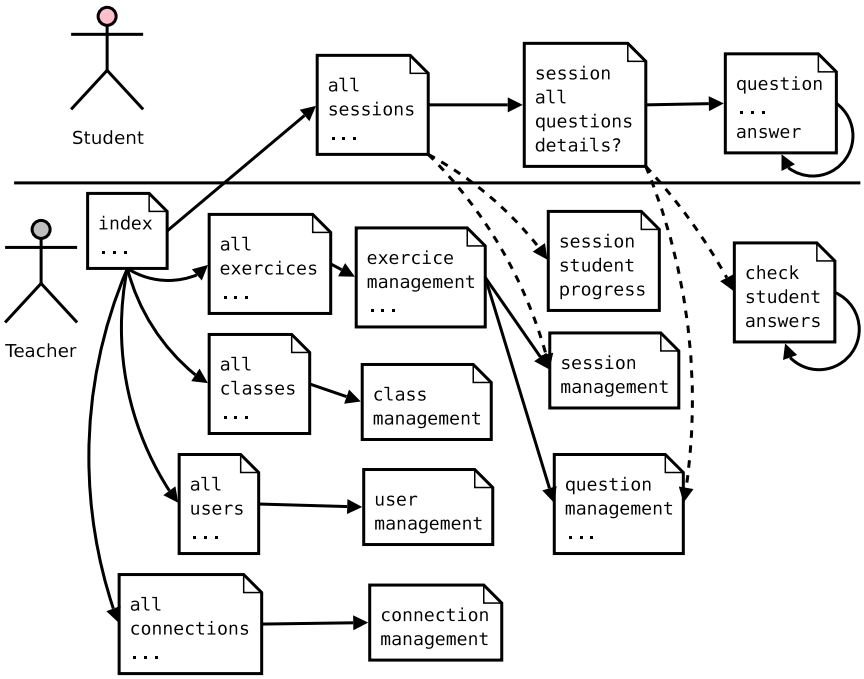

Figure 3: Interface architecture

given to the student by the correction system.

4 Corrector Internals

Figure 4 outlines the corrector internal database schema accessed by the pages. Users may belong to classes. Exercises are composed of questions, which may be corrected by different means and using different database connections. A session for an exercise is open to users, and they can provide answers to questions, although only the last one given is important. The interface is optimized for the common case, for instance by automatically switching to the next question on success, and asking for another answer on failure.

The student answer table quickly becomes the largest of all application tables as users keep suggesting new answers till they are given the pass mark and go on to the next question. It is critical that the queries used in the corresponding page are executed efficiently. A PostgreSQL partial index is very useful in this context, so as to index only the last answer for every question and every user.

Access control is implemented both by apache and the page scripts. The apache side requires only a valid user to be able to access a page. The authentication password may be fetched directly from the corrector database

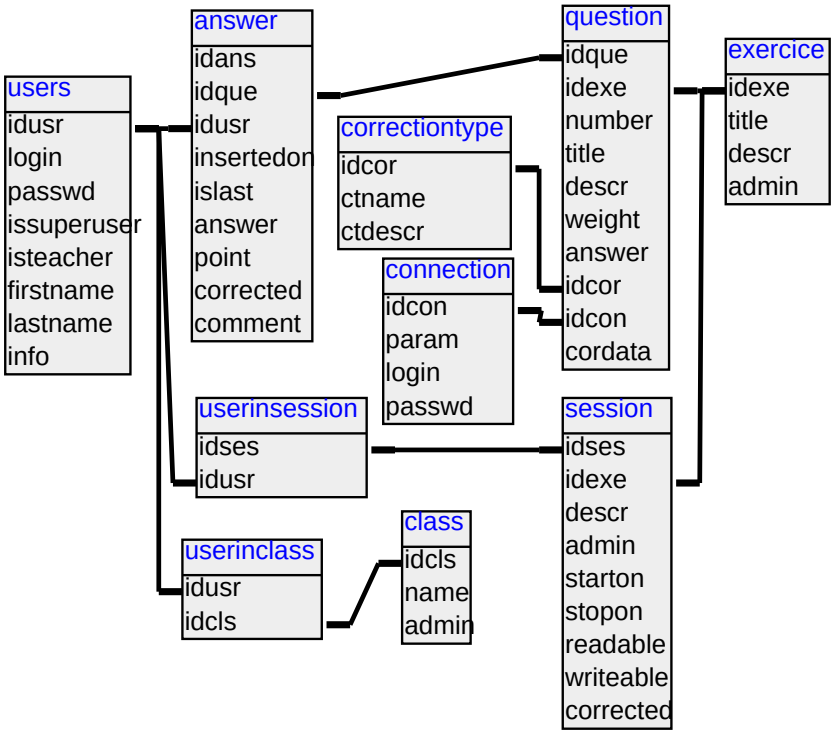

Figure 4: Corrector Data

by the used of the *mod_auth_pgsql* [7] module or stored anywhere apache can get it. At the beginning of each page, depending on the provided user identity, the available rights are assessed. It is necessary to do so because what can or cannot be done by the user is defined dynamically within the application itself: what sessions are currently open to a student, who is a teacher, and so on. The security enforcement is programmed rather than declared, because the available security models in apache do not allow the fine grain adjustments necessary to the application.

All database interactions issues such as connections and SQL queries are encapsulated in a common module shared by all pages. The needed configurations such as database host, login, password and so are managed from the apache configuration file and passed by the mean of environment variables. Thus it is not necessary to edit the page sources to configure the application.

The application is internationalized [9], that is all messages displayed to the user can be adapted to any language, thus it can be localized for any language if a translation is provided into a standard PO (Portable Object) file. At the time, both English and French are available. Handling the localization choices in the dynamic pages require some tweaking: indeed, the choice is not fixed and can be altered by the student. This can be done by several mean: the user may specify a preferred language in its browser configuration transported by the HTTP accept-language header; However very few users know about these configuration details and use it, so the application can also be localized by clicking on a link which sets a cookie holding the user preference.

Even on a local network with controlled students the scripts potential security issues. As the automatic correction will execute student provided queries to check for their validity, the student may break something: for instance a massive cross join can put down the database server or the machine running the page script. In order to avoid such issues, a timeout is set to the database for the student queries, and result sizes are also limited. The queries provided must be carefully escaped before being passed to the database. Within it the student can do whatever is available in the database. This last point is not significant an issue as the student already has a direct access to the database during the practice, so there is no more harm that what can be done directly.

5 Automatic Correction

A key point of the corrector application, after which it was named, is its ability to provide an immediate feedback to students. This greatly improves the learning experience of the students as well as the life of their teacher, as direct intervention during the practice sessions is only required to solve

comprehension or technical problems encountered by the students, and not to validate good answers. In this section, we describe the different basic and advanced correction modes available, their internal working and their pitfalls...

First, the application allows a question to be simply stored into the system to be looked at later by the teacher and be corrected manually. Thus, no feedback at all is provided. One of the teacher page shows all answers to a question for a given session, which can be corrected at once in a single step. The manual correction involves giving a mark and writing a comment, which in our experience is very rarely if ever looked at by the students.

Second, a set of correction modes compares the provided answer with a target string. The several comparison variants include regular expression matches, possible case insensitiveness, prior answer string case and space normalization, and whether the comparison is performed for part of the answer or for the whole. These corrections modes have been added as their need arise for new practice sessions. They have been proven hard to be highly effective, as discussed in the next section.

Third, the *select* correction mode allows to compare a select SQL query given by a student to a reference query provided by their mighty teacher. As the relational algebra and many SQL functions makes many queries equivalent, it is not possible to compare them directly. Exploring the class of algebraic equivalence would be difficult if not impossible, as semantic details can contribute to the equivalence of queries in a given schema. Thus a simple and effective approach is followed, which consists of comparing the relations resulting from the execution of both queries. If they match in size and contents, then it is okay. Otherwise, the first difference, whether the number of tuples or attributes or the differing item is reported to the student as a clue.

Fourth, three correction modes rely on a select query the result of which must be either empty or non empty, or lead to a boolean true response after the execution of the student response. This allows to test working inserts and updates for the first form, and deletes for the second one, and for the last the existence of some objects and their properties, for instance to check that a view is defined by the student as requested.

Fifth, a cryptographic token-based shared secret correction mode is also provided. The idea is shown in Figure 5. The student interacts with a helper application, say a web server in a practice about the HTTP protocol. When the interaction succeeds, the helper application releases a proof of success token which involves a public information, such as the login name of the student obtained by the ident protocol, a description of the exercise undertaken or the day and time of the event, and the cryptographic hash of

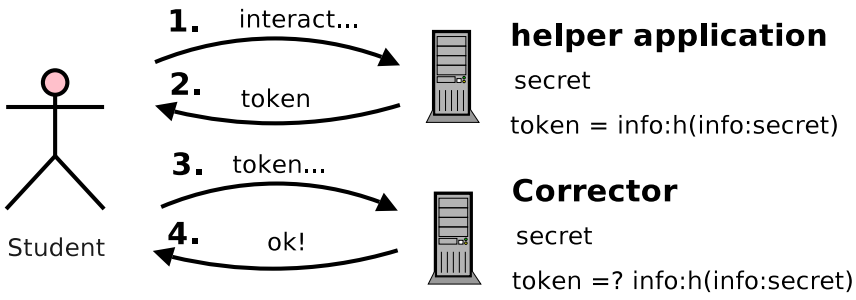

Figure 5: Cryptographic validation

this public information and a question specific secret.

$$\text{token} = \text{info:}h(\text{info:secret})$$

When receiving the token, the correction modes recomputes the hash from the provided information and the secret it knows in order to validate it. Later the teacher can check that the information provided to identify students that would use credentials obtained by another student in its response...

6 Discussion

A common issue of the SQL oriented correction modes is that they involve executing a query on the database. These queries may modify the data, either by accident or by intent when insertions, updates or deletions are asked for. A possible solution would be to use a read-only connection. Although this is fine for SQL selects, it would fail for more advanced commands which must modify the contents or even the schema. Thus all queries performed by the corrector application are enclosed within a transaction which is rollbacked at the end instead of being committed. The isolation level provided by the MVCC (Multi View Concurrency Control) available in PostgreSQL insures that simultaneous connections do not interfere. As all control, definition and manipulation commands are subject to being rollbacked by a transaction, this approach works as well with schema alteration or granted privileges.

Crafting questions suitable to the corrector interface is not straightforward. There is a communication problem, as there must be only one possible response from the student. Indeed, the automatic correction cannot deal with good alternate answers to ambiguous questions and attribute points to them. Thus all SQL query questions must tell precisely the attributes expected as well as the order in which the results must be presented, thus requiring full *order by* clauses in all answers.

Students often use the web interface as an interface to the database, not bothering to try the queries first on the database before submitting them to corrector. Although this works well for simple queries, the interface is not as friendly as the different text or graphical database interfaces when dealing with user syntax or semantical errors. Moreover, the HTML text input is very basic: no syntax highlighting or automatic indentation is provided which helps them for advanced questions.

Most students never get back to the interface to have a look at the expected answers after the practice, once the corrections are available. Thus entering comments to explain why their answer was not right and what misunderstanding lays behind looks like a loss of time. Some approach should be devised that would push the relevant information to them, possibly through a simple mail pointing to a web summary page for instance. Student time is a scarce resource.

Some students tend to develop clever strategies to get rid of the practice as quickly as possible while still getting the positive feedback from the interface. A yes/no answer would often result in trying both answers without giving a thought to the question. For a number, they would try all integers from 0 till they get the point. Keyword oriented corrections may result in large responses provided by copy-pasting a dictionary or a large file instead of extracting the answer from the raw data. Some SQL query questions result in empty result sets: At the first error the student would figure that out from the 0 row size hinted by the interface, and the next answer may be whatever query returns an empty result. Some students take their teacher for a low-power light bulb.

Keyword or regular expression automatic corrections do not work well. Students are often much more imaginative than foreseen when it comes to providing the right answer to very simple questions: an American student would keep responding in English, because the interface is localized in his language, but unluckily the automatic correction is not! A number could be provided in numeral 1 or with letters *one*...

Although a teacher interface is available to create new exercises directly, it is seldom used. Indeed, it is often useful to have a paper version of the exercise to distribute. For that purpose, an importation script takes a LaTeX file with the exercise and special comments to feed it to the Corrector system. This allow to have both paper and online versions very easily. On the other hand, the teacher interface is often used to fix on the fly questions and correction parameters during the session when students stumble upon problems or ambiguities.

7 Conclusion

More than 150 students have passed through the Corrector prototype on lessons about databases, network protocols and cryptography. Its constant feedback has helped them to reach a better comprehension of the subjects involved, and their teacher to focus on giving explanations that help them towards this goal.

The teacher's time spent on preparing and performing lessons has not been tremendously reduce with Corrector. However the quality of the student experience is greatly improved, as well as the teacher motivation. Crawling through bunches of student SQL queries after the practice session to check them was very unattractive. More time now is spend on the preparation and the configuration, and less time on the correction phase as most of the work is already done.

I intend to distribute Corrector as a free software at some stage. However it requires a precise documentation, especially on the system installation issues which involve careful database and web server configuration, and which is yet to be written. Future works also involve new features, such as an examination mode which does not report the points to the student. A redevelopment in python [8] is also considered.

References

[1] PostgreSQL. `www.postgresql.org`, 1996–2006.

[2] Apache Software Foundation. Apache. `httpd.apache.org`, 1995–2006.

[3] Edgar F. Codd. A relational model for large shared databanks. *Communications of the ACM*, 13(6):377–387, June 1970.

[4] ISO/IEC. Information technology - database languages - SQL, 2003. Standard 9075.

[5] Dave Raggett, Arnaud Le Hors, and Ian Jacobs. HTML 4.01 Specification. W3C Recommendation on `www.w3c.org`, December 1999.

[6] David R. T. Robinson and Ken A. L. Coar. Common Gateway Interface version 1.1. W3C RFC 3875, October 2004.

[7] Adam Sussman, Matthias Eckermann, and Giuseppe Tanzilli. Apache pgsql authentication module. `www.giuseppetanzilli.it`, 1996–2005.

[8] Guido van Rossum. Python programming language. `www.python.org`, 1991–2006.

[9] Phillip Vandry. Locale::gettext perl module. `www.cpan.org`.

[10] Larry Wall. Perl. `www.perl.org`, 1987–2006.

[11] Larry Wall, Jon Orwant, and Tom Christiansen. *Programming Perl*. O'Reilly, July 2000.

typeset with LaTeX, document revision 946