Designing a DBMS Development Course with Automatic Assignment Evaluation

Viacheslav Galaktionov[1, 2]
[1] *JetBrains Research*
[2] *Saint-Petersburg State University*
Saint-Petersburg, Russia
viacheslav.galaktionov@gmail.com

George Chernishev[1, 2, 3]
[1] *JetBrains Research,*
[2] *National Research University Higher School of Economics,*
[3] *Saint-Petersburg State University,*
Saint-Petersburg, Russia
g.chernyshev@spbu.ru

***Abstract*—Due to the constantly growing amount of data in the world, we need better ways to process it. Conducting research and development in this area requires skilled workforce. Different universities provide different courses to prepare people for this line of work.**

In this paper we present our approach to conducting practice sessions within a DBMS development course. We describe some of the approaches implemented by other universities, outlining their advantages and disadvantages. A popular approach is to provide students with a prototype of some DBMS and let them incrementally improve it by completing certain tasks. The two most important problems in these courses are 1) choosing a DBMS (an industrial or educational one), within which students should work; 2) deciding whether to employ an automated testing system, and, if so, which one. In both cases we take a look at several options and justify the necessity to create a new one, which we then describe. In total, we have developed the following: a base prototype of a row-store query executor, an automated testing system, a set of problems along with reference solutions and test cases. Finally, we present the results of a test run involving 17 undergraduate students.

***Index Terms*—education courses, education, databases, query engine, query processing, database internals**

I. Introduction

It is well-known that the amount of data that needs to be processed is increasing with an unprecedented speed [1]. This is mostly related to the emergence of such areas as Big Data, the Internet of Things and cloud computing. The research of existing data storage and processing methods and the development of new ones are thus becoming more and more important.

Naturally, conducting said research and development requires a great amount of highly-qualified workforce. Preparing such cadres is an important task that perhaps all universities attempt to accomplish.

There is a multitude of courses aimed at improving qualifications in subjects related to databases. They can be divided into two categories:

1) Introductory courses, that explain a specific set of basic terms. Usually such courses consider the classic relational model and teach the students to apply it, but sometimes they can include information on various NoSQL systems.
2) Advanced courses, whose main task is to teach students to actually develop DBMSes. A lot of attention is paid to the internals of one or several classes of systems, as well as the most important algorithms. Student either develop their own system from scratch or modify an existing one.

Evidently, these two categories serve different purposes and as such have to be taught differently. Let us concentrate on the advanced courses in this paper. A question arises: how should one organize such courses? Clearly, just giving lectures to the student will not be enough because of the practical nature of the covered topics. The students need to be given an opportunity to apply their new knowledge in order for them to fully understand the material. This means that special attention should be given to practice sessions. In this paper we present our approach to conducting practice sessions within a DBMS development course.

The contribution of this paper is the following:

1) An overview of some of the approaches to conducting practice sessions within advanced database courses used in different universities.
2) The structure of our approach: the overall idea of the course, the used DBMS prototype, the task set, approach to testing students' solutions.
3) The results of the test run of our course, a description of our experience and the encountered issues.

This paper is structured as follows. In Section II we describe various DBMS development courses. Next, in Section III we discuss overall architecture of our approach, and in Section IV we enumerate various security measures that we undertook. The syllabus of our course and the idea of proposed tasks is described in Section V. The Section VI presents the outcome of the first test run, justifies the benefits of our approach and describes the encountered issues. The future work and conclusion are presented in Sections VII and VIII, respectively.

II. Related Work

Conducting advanced database courses is not a new problem, there are publications describing experience of many universities [2]–[6]. The referenced papers provide two different viewpoints on how such a course should be organised:

1) Students of the course described in [2] had to modify PostgreSQL, a DBMS used in the industry. However, due to complexity of PostgreSQL's architecture, only two of the tasks required students to actually modify its code.
2) On the other hand, the course described in [3] employed a DBMS developed specifically for it, SimpleDB. It was made with code clarity in mind, sacrificing performance where necessary. This allowed students, who were new to the subject, to find their bearings in the code and start modifying it. Because of this, the number of programming-related tasks in this course was 9.

   A similar approach is being used at Harvard [4] right now. There, students implement their own main-memory column-store. They cover such topics as indexing methods optimized for main-memory and shared scan methods. During the class hours students discuss state of the art research papers.

   In Russia such courses also exist. In 2004 the South Ural State University offered a course "Parallel database systems" [7], where students had to develop their own prototype of a parallel database management system using the MPI standard. In the Computer Science Center [5] course "Software engineering for big data" students were offered to implement a distributed key-value data store and an application. The assignment encouraged team participation[1] and there were 4 tasks overall. Several years ago Innopolis Univeristy also offered [6] a three-week assignment for building a simplified relational query engine in Python. This project was aimed for team participation also. To the best of our knowledge, both these courses involved manual checking of the solutions.

The second approach appears to be more desirable, as it is both easier for the students and more saturated, i.e. instead of simply reading about different algorithms and approaches, the students will have to actually implement them.

Another advantage of the second approach is that the students are given an opportunity to improve their skills related to systems programming as well as complex system development [8], [9].

Because of the aforementioned reasons we have decided to take the second approach. However, that raises the question: which DBMS should we use? SimpleDB itself is hardly an option, since it puts a lot of attention on multi-user operation, which is not very interesting for us but complicates the code somewhat.

We have considered other systems as well: Minibase [10] and MinSQL [11]. The former provides a great set of features and comes in two version: Microbase, which is freely available to everyone but has a heavily restricted feature set, and Minibase, the full version, which is available only to teachers. However, the source code for the full version has already been published by third parties, which makes it easy for students to cheat. The latter, MinSQL, has never been made public.

Therefore, we have decided to develop our own educational DBMS.

Another important problem is grading students' work. In the case of complex systems like DBMSes it becomes too difficult to assess the code by just reading it. Some form of automated testing is required. Trusting the student to write their own tests is not an option, and handing out tests to students could lead to them writing code that's designed to pass the tests instead of actually solving the problems.

A more correct and modern approach is to allow the students to upload their code to some testing system, which runs various tests on it. Besides testing for correctness and performance, such a system can recognize different kinds of cheating: copying others' solutions, DoS attack, etc.

This approach is very popular with programming contests and online courses. In both cases there is a stream of solutions that is too large for a group of people to evaluate in a reasonable time period.

Of course, automated testing is used outside of these areas as well. Code quality is an important characteristic of any software, so it receives a lot of attention. There are industrial systems for automated code testing.

We have evaluated multiple systems from different areas. Let us summarize our findings:

1) **Programming contest platforms**. We have considered Yandex.Contest [12] and Codeforces [13], which are the most popular platforms in Russia. These systems are capable of testing the code for correctness, performance, and are also protected against DoS attacks. However, they are expect the users to provide a single source file. This becomes a problem with complex systems like DBMSes. It is possible to put the entirety of source code into one file, however, that is not something we want to teach our students.
2) **Online course platforms**. This area was represented in our research by Stepik [14] and Coursera [15]. They exhibit the same problem as the programming contest platforms. However, they have additional disadvantages. For example, to the best of our knowledge, it is impossible to create a private Coursera course. Furthermore,
3) **Industrial testing systems**. We have looked at Travis CI [16] and Jenkins [17]. While they provide exceptional testing capabilities, they are not well-suited to track students' progress.

Due to the lack of a system that would meet all our requirements, we have decided to implement our own.

III. Testing System Architecture

Our system is organized as a set of Docker containers. The system provides a web interface, where the students can submit their code for testing and the teacher can track their progress. The main parts of the system are as follows:

1) **Web server**. The system has two kinds of user accounts: for students and for teachers. The interface is structured

[1] https://github.com/alesavin/csc-bdse

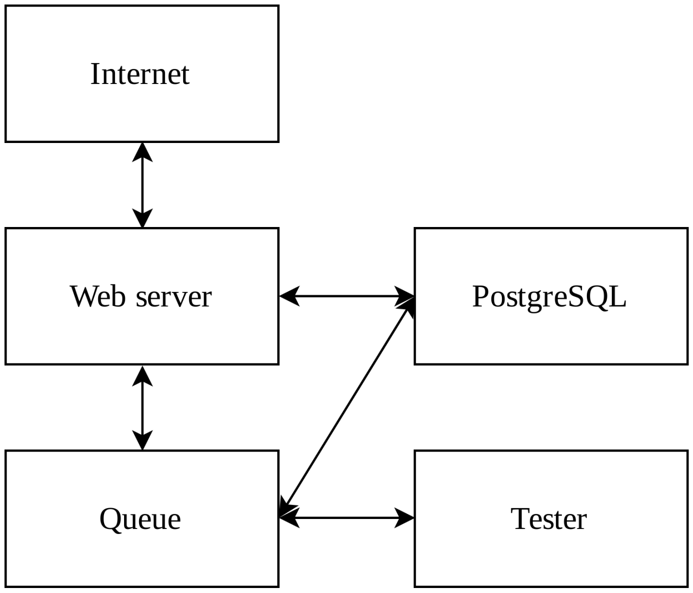

Figure 1. Interaction between different parts of the system

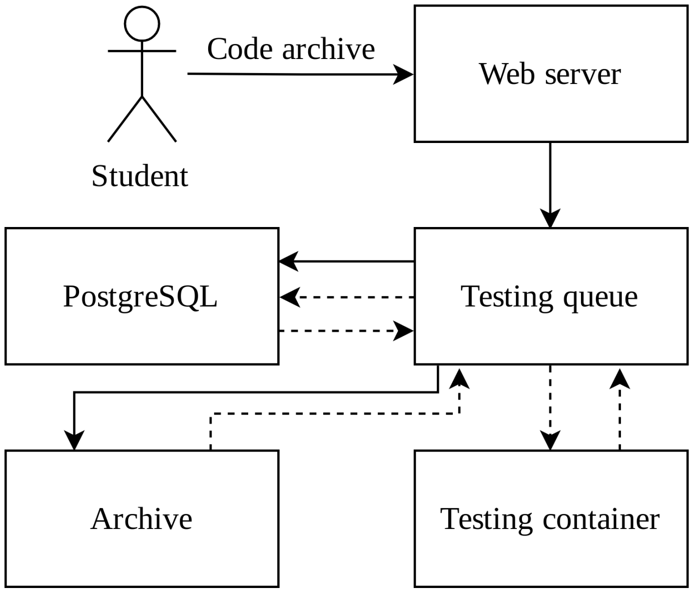

Figure 2. The solution checking process

differently depending on who the user is. The student will see a list of tasks, some of which will be marked as completed if they have submitted a solution that passed all tests successfully. The teacher can add and remove students from the course, make various changes to the problem set, and track the students' progress.

2) **Database**. We use PostgreSQL to store the information on students, tasks, and submissions. We also use it to organize the testing queue, which will be described later.
3) **Testing queue**. This module acts as a mediator between the web server and the testing container. It is also responsible for storing the uploaded solutions on disk. The testing queue provides an HTTP API that is used by the web server whenever a student uploads a new solution or whenever someone wants to download a previously submitted solution.
4) **Testing container**. Completely isolated from the rest of the network, this module encapsulates building and running the code. The entrypoint of that container is a script that will unpack and compile the code and then run our testing program. This program will run a set of queries and check their results, meanwhile providing a log that can help the students understand how and why their code failed. However, this log is very concise and does not provide the student with all information about errors in order to keep the test data secret.

The diagram of interaction between different parts of the system is presented in Figure 1. Note that only the web server has access to the Internet.

The testing system was written in the Go programming language. The net/http and html/template packages from its standard library were used to implement the web interface and the testing queue logic.

Now that we've covered the structure of our system, let us describe how it operates when a student submits a solution. First of all, the web server will receive the code archive and pass it to the testing queue via its API. Then, the testing queue will save the archive on disk, and then add two records to the database: one that describes the submission itself, and one that describes the testing result.

The testing queue process contains a set of coroutines, each of which periodically queries the database for submissions that need to be tested. If there is a solution that is ready to be tested, the coroutine will get a lock on the corresponding database record, and start a testing container. This container will be limited in its resources such as the amount of RAM or execution time. These limits are configurable via the teacher's web interface on a per-problem basis.

Once the testing container stops working, the coroutine will determine why it stopped and assign the submission one of the following grades: accepted, invalid answer, runtime error, timeout, compilation error. After that both the grade and the log are saved in the database and the testing queue starts waiting for a new task.

This process is depicted in Figure 2. Solid arrows represent the path that a student's code archive follows within our testing system when it is uploaded. Dash arrows represent the behaviour of a coroutine that is responsible for checking this solution.

IV. Security Measures

Any user facing system should be sufficiently protected from different kinds of attacks. Of course, our system is no exception. A lot of attention has been given to protecting the system from malicious behaviour that can be exhibited by some students. Let us go over some of the precautions taken:

1) Our system works only over HTTPS. This helps us prevent traffic-sniffing in order to steal passwords, for example.
2) Testing containers are limited in their resources: RAM, the number of PIDs, disk space. These limits help us prevent fork-bombs and memory floods.
3) By leveraging access rights we prevent the student's code from being able to write into the log generated by the testing program, thus preventing data leaks.
4) To prevent data loss, backup copies of all essential files were made daily.

V. Syllabus

The course was designed with undergraduate students in mind. Thus, it does not require any specific knowledge on the students' part: they should be experienced programmers with a basic understanding of the C++ programming language and be familiar with the UNIX environment. They should also understand what a database management system, a relational data model, and SQL are.

At the beginning of the course, all students are provided a prototype of a relational DBMS with minimal functionality:

- parsing queries written in a subset of SQL;
- a small set of physical operators: data source, filtering, and nested-loop join;
- building a simple query plan that can have at most one join operation;
- reading table data and printing query results in the CSV format.

The prototype is written in the C++ programming language. Architecture-wise it is a row-store that follows the Volcano query processing model. It was provided to the students through a GitHub repository [18].

A total of 8 tasks have been prepared, each of them aimed at expanding the prototype's functionality. The main topics were query optimization and execution with read-only workloads.

Along with the tasks we have written a reference solution to each task, using the simplest approaches. This reference solution was used during test generation to keep track of the expected execution time.

At the end of the course students who have completed all tasks will have developed a DBMS with the following features:

- block-oriented data processing;
- a larger set of physical operators: cross product, merge join and hash join, projection, multiple implementations of duplicate removal;
- an optimizer that can optimally select physical operators and the order of joins given the available RAM amount;
- a rewriter able to simplify the predicates and recognize the inconsistent ones.

The students were supposed to work on their own. Weekly seminars were held so that they could consult the instructor regarding task formulations and any technical problems they encountered. In order to check that there was no cheating, we froze the submission three days before the final exam to look through the code.

Table I
Project Details

Type of work	Lines of code	Man-hours
Testing system	2900	20
Test cases	—	30
Base prototype	1200	5
Reference solution	800	15
Source code post-review	—	6
All components	4900	76

VI. Course Test Run

This course has been conducted at the Higher School of Economics in Saint Petersburg for a group of 17 students. All of them were taught this course in the same manner, there was no division into experimental and control groups. The test run was approved by the administration of the St. Petersburg School of Mathematics, Physics and Computer Science. No personal information was ever retrieved, and students' behavior within the course was not matched to students' university record or any other personal data. The testing system was deployed on the server provided to us by the institution.

A. Quantitative analysis

First of all, let us estimate various metrics related to all components of our system. They are presented in Table I. Here, we try to show the effort it took to develop each of the components in lines of code (all involved programming languages combined) and man-hours. The first line describes testing system itself, which was the largest in terms of lines of code. However, in terms of man-hours, the development of test cases and more importantly calculating their time limits were significantly more demanding.

The next three rows describe the properties of the prototype that was handed out to students, our reference implementation, and an averaged solution. Please note that the reference solution was built on top of the base prototype. The last line describes our effort to conduct a source code post-review to check for cheating and other possible problems.

Our estimates (from previous years, where manual checking was performed) show that each task requires at least two attempts, where each attempt lasts about 10 minutes. Therefore, the overall time required to check all problems that are offered in our new course would be approximately $10\,minutes * 2 * 8 * 17 = 45\,hours$. Each practice class session lasts 90 minutes (once a week) and there are $\approx$ 14 such sessions in a semester, which gives us 21 hours in total. Therefore, it would be impossible to check all these solutions without involving additional reviewers.

Moreover, this number of additional man-hours required to prepare this course has been compensated by the following:

1) **Flexibility.** Students can check their solutions at any arbitrary time and can work at their own pace. Further-

more, they can attend class sessions only if they have questions.

2) **Quality.** We have improved the quality of the course by introducing precooked test queries and answers. This removes the possibility of instructors forgetting to check some particular test cases.
3) **Reusability.** The testing framework we have developed can be reused during the next iterations of the course.
4) **Scalability.** Using an automated testing systems makes it significantly easier to increase the number of students in the future runs.

B. Qualitative analysis

The course has received good feedback from the students. However, we have found some problems in the way our course was organized:

- We have simplified the code for tuples too much, so in order to make block-oriented processing beneficial, the students would have to rework a significant part of the system, especially since this task was given late in the course.
- Easy access to automated testing prevented students from writing their own tests. This has reflected on the total number of submissions, which has exceeded 1000. Perhaps we should have provided students with a data generator instead of relying on them to find or develop one.
- Testing correctness by running a query and checking its result can be excessive, especially when testing such features as query rewriting. Furthermore, writing tests becomes unnecessarily hard in such cases, since the only criterion for success is whether the queries can be executed within a given time limit. A better approach would be to check the query plan directly.
- At the earlier stages of the course (before implementing a full-blown query optimizer) query execution time depended heavily on the order of joins. This made it impossible for one of the students to pass the tests despite having a perfectly working solution.
- Due to the fact that each run of the students' programs corresponded to running a single query, some students made questionable design choices. For example, two students updated catalog information during query rewriting instead of relying on temporary data structures.
- There are some things that cannot be tested in an automated manner. The greatest example of that is student's understanding of the code they have written. There was an exam at the end of the course, however, it was theoretical in nature and therefore there was no place for code-related questions. Conducting code-review sessions during practice lessons would be greatly beneficial.

VII. Future Work

This was just the first iteration of the course. We hope to improve our testing system and additional materials. In addition to fixing the problems mentioned in Section VI, we plan to do the following:

- Improve the user interface of our testing system, both in terms of functionality and visual design. As an example of missing functionality, the system might benefit from having an option to let the students ask questions so that the teaching assistants could answer them.
- Transfer the code submission process from an archive-based one to a git-based one. We expect that to bring multiple improvements. First of all, the process should become easier for students. Secondly, experience shows that many students will use git anyway. We have noticed that almost a half of our students have made their own publicly available forks of our GitHub repository. These forks included their solutions to the problems, which is undesirable, as it makes it easy for future students to cheat.
- Add means to check if a solution was copied from some other student. This time we had to dedicate three days before the exam in order to manually check all submissions that were accepted by the testing system.
- Make building and running the code separate isolated stages. This would allow us to keep full compilation logs and show them to students in case their code fails to build, as well as measure the actual query processing time. The latter would in turn allow us to range the students based on their code's performance, thus giving them an incentive to find better solutions.
- Provide students with tools to test their code locally: a data generator; some simple logging facility; and a verbose version of our testing program, which would give the user detailed error messages without being afraid to leak test data.

VIII. Conclusion

In this paper we have described our experience of organizing practice lessons for a DBMS development course. We have outlined some of the approaches that have been taken by others before us and outlined their advantages and disadvantages.

In order to conduct the course we have developed a prototype of a simple DBMS and put together a set of problems for the students, along with a reference solution. To simplify the testing process for all parties involved, we have developed a system for automated testing.

We also describe the problems we have encountered during this course. Most of them have to do with the fact that this was the first iteration of this course and we lacked the time to plan and implement every desirable feature.

However, we believe our approach is viable and therefore intend to continue conducting courses in this manner. There is much to improve in our course besides dealing with the aforementioned problems. Our plans for future work are also described in the paper.

References

[1] A. Oussous, F.-Z. Benjelloun, A. A. Lahcen, and S. Belfkih, "Big data technologies: A survey," *Journal of King Saud University - Computer*

and Information Sciences, vol. 30, no. 4, pp. 431 – 448, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1319157817300034

[2] A. Ailamaki and J. M. Hellerstein, "Exposing undergraduate students to database system internals," *SIGMOD Rec.*, vol. 32, no. 3, pp. 18–20, Sep. 2003. [Online]. Available: http://doi.acm.org/10.1145/945721.945725

[3] E. Sciore, "Simpledb: A simple java-based multiuser system for teaching database internals," 01 2007, pp. 561–565.

[4] "Harvard CS165: Data Systems," http://daslab.seas.harvard.edu/classes/cs165/, [Online; accessed 06-February-2019].

[5] "Программная инженерия больших данных," https://compscicenter.ru/courses/big-data-software-engineering/2018-spring/, [Online; accessed 06-February-2019].

[6] "Своя СУБД за 3 недели. Нужно всего лишь каждый день немного времени..." https://habr.com/ru/post/347274/, [Online; accessed 06-February-2019].

[7] М. Цымблер, Л. Соколинский, А. Лепихов, "Прототипирование параллельной СУБД как основа учебного курса по параллельным системам баз данных," *Суперкомпьютерные системы и их применение*, 2004, с. 212–217.

[8] K. Smirnov and G. Chernishev, "ACM SIGMOD Programming Contest: an opportunity to study distinguished aspects of database systems and software engineering," *Computer Tools in Education*, no. 5, 2014. [Online]. Available: http://cte.eltech.ru/ojs/index.php/kio/article/view/1320

[9] C. Genzmer, V. Hudlet, H. Park, D. Schall, and P. Senellart, "The SIGMOD 2010 programming contest a distributed query engine," *SIGMOD Record*, vol. 39, no. 2, pp. 61–64, 2010. [Online]. Available: https://doi.org/10.1145/1893173.1893185

[10] R. Ramakrishnan and J. Gehrke, *Database Management Systems*, 3rd ed. New York, NY, USA: McGraw-Hill, Inc., 2003.

[11] G. Swart, "MinSQL: A simple componentized database for the classroom," in *Proceedings of the 2nd International Conference on Principles and Practice of Programming in Java*, ser. PPPJ '03. New York, NY, USA: Computer Science Press, Inc., 2003, pp. 129–132. [Online]. Available: http://dl.acm.org/citation.cfm?id=957289.957328

[12] "Yandex.Contest," https://contest.yandex.ru/, [Online; accessed 06-February-2019].

[13] "Codeforces," https://codeforces.com/, [Online; accessed 06-February-2019].

[14] "Stepik," https://stepik.org/, [Online; accessed 06-February-2019].

[15] "Coursera," https://www.coursera.org/, [Online; accessed 06-February-2019].

[16] "Travis CI," https://travis-ci.org/, [Online; accessed 06-February-2019].

[17] "Jenkins," https://jenkins.io/, [Online; accessed 06-February-2019].

[18] "ToyDBMS GitHub repository," https://github.com/chernishev/ToyDBMS, [Online; accessed 06-February-2019].