Noname manuscript No.
(will be inserted by the editor)

One DB Does Not Fit It All: Teaching the Differences in Advanced Database Systems

Lena Wiese · Aboubakr Benabbas · Golnaz Elmamooz · Daniela Nicklas

the date of receipt and acceptance should be inserted later

Abstract In this article we report on our experience with teaching differences that exist between relational and non-relational data models. We present results of an evaluation of a practical course in which students are assigned 15 queries within 6 tasks that they execute on four different database systems. The pedagogical aim of this course was to show conceptual differences between data models, difficulties that can occur when trying to formulate queries in different query languages, as well as specific system behavior. We present several technical environments in which provided the database systems over several years. We evaluate the practical course based on a questionnaire that recorded the students' performance on each task for each DB system.

Keywords Database Systems · NoSQL · Teaching Concept

1 Introduction and Related Work

In standard CS curricula the relational data model is a major teaching topic. Accordingly, SQL databases are presented in depth in order to teach relevant practical database competencies. The advantage of this approach is that most human beings are used to structuring data into tables with a fixed amount of columns so that the relational data model appears somewhat intuitive; moreover SQL has the advantage that it is an all-round language to express versatile queries and that it is a mature standard. However in certain circumstances it is inconvenient to "squeeze" your data into a tabular format and – following the theoretical specification – applying database normalization steps. In particular, when applying the relational data model and SQL, a database user might run into difficulties with sparse and heterogeneous data (for example, unknown amount and type of attributes at schema creation time), complex network-structured data and nested data.

On this background we designed an Advanced Data Management lecture combined with practical exercises. Our purpose is to show pros and cons of other data models in comparison to the relational one. The practical exercises were conceptualized with the aim to give students a hands-on experience with a variety of both data models as well as database access methodologies.

Similar activities are presented in [5] on conducting a NoSQL course over three years. Two databases, MongoDB and Cassandra, were studied in detail to help students make better decisions in their industrial work life. Students were provided with individual and teamwork assignments to work on the mentioned databases from different aspects (like query-driven data modeling, sharding and replication, and geospatial related concepts). In the end, the students were asked to evaluate the whole teaching concepts and targets of the course using a general survey. Another brief use case (only employing CouchDB) is described in [2]. In comparison to these approaches, our concept covers a more comprehensive treatment of the differences of data models on CS Master level and covers a wider range of database systems comparatively. Moreover, we present the novel concept of unsolvable tasks with regard to specific data models and DBSs. This reflects the situation in which an engineer should solve a certain task with a given non-standard DB, and she does not know upfront if

L. Wiese
Goethe-Universität Frankfurt am Main
E-mail: lwiese@cs.uni-frankfurt.de

A. Benabbas, G. Elmamooz, D. Nicklas
University of Bamberg
E-mail: [firstname.lastname]@uni-bamberg.de

it can be implemented with a simple query or needs a tedious work-around in the application. This competency is rarely covered by university courses, where students can always assume that tasks are solvable when assigned in a course. However, to keep the workload bounded, we give a maximum time for each task.

In this paper we present the underlying concept (Section 2) for the Advanced Data Management course and a use case (Section 3) for practical exercises that in our opinion is ideal to show (dis-)advantages of specific categories of database systems. We compare three alternative system environments (Section 4) for setting up the database systems and enabling user access to them. We describe the special situation of a full online course in Summer Term 2020 (Section 5). Lastly we present the evaluation of student feedback (Section 6).

2 Teaching Concept

We first survey the teaching concept of the course and discuss how lecture, exercises, and exam tasks are designed based on competencies and learning outcomes.

2.1 Competencies

According to the European Qualifications Framework (EQF), a *'competence' means the proven ability to use knowledge, skills and personal, social and/or methodological abilities, in work or study situations and in professional and personal development* [1]. Following a competency-oriented design, learning outcomes are not defined based on a list of lecture contents, but by competencies that students can achieve when completing the course. As per the EQF definition, competencies depend on the situations in which they will be used. For a database course, we defined four major target situations, which can occur both in study or work and are relevant both for industry or research situations.

1. For a given set of requirements or tasks, a database system needs to be chosen, e. g., at the beginning of a new software project or data science project.
2. A database system should be used to solve tasks which it was designed or chosen for.
3. While solving tasks with a database system, one must understand the behavior (e. g., query execution performance) based on knowledge about the internal implementation of the system.
4. A database system should be able to solve new types of tasks which were unknown when it was chosen.

Notably we use the term *task* instead of *query*: A task is the problem that should be solved supported by the database. Depending on the database system and its query capabilities, it can be more or less supported by queries; whatever is left to fully fulfil the task must be then implemented in application code.

While it may be straight-forward to define competencies for the first three of these situations for a given DB system, situation 4) needs specific consideration: When using a database system for a new type of tasks, the level of support for the developer can differ: from full support – the task can be solved by DB queries only – to almost no support – the task must be solved mostly by application code with a *workaround*. The competencies needed for this situation cannot be taught well in the lecture but we rely on the exercises for that.

Formally, the module handbook at Bamberg University contains the course *Advanced Data Management* as 6 ECTS Master level course with lecture and exercises. Along the same lines, at Frankfurt University the course *Advanced topics in the area of databases* is offered.

2.2 Lecture

The main contents of the lecture were designed along the lines of the book *Advanced Database Management* [8], complemented by selected content from *Designing Data-Intensive Applications* [6] and tutorials and papers [3,4]. For a Master level course, we assume solid knowledge of traditional, relational database systems. This also corresponds to the work or study situations in which the students will apply their competencies later: It can be assumed that colleagues and IT departments have much more experiences and knowledge about relational DBS than about the NOSQL systems we teach.

The goal of the lecture is to present and discuss the knowledge needed for the learning outcomes. This knowledge is two-fold: (1) general implementation and data modeling techniques of NOSQL DBSs, e. g., "data partitioning" or "embedding vs. referencing", and (2) specific knowledge about DBSs, like query languages. In theory, we could separate these two parts completely: First, teach all the generic concepts, then the concrete systems with references to the techniques they used. Yet, the exercises are done in parallel to the lecture such that we decided to integrate this content more closely.

The general structure of the lecture is the following:

1. (1 lecture) Introduction and motivation: Introduce learning outcomes and competencies, and define new requirements for database systems that led to the development of NOSQL systems.
2. (1 lecture) A short review of relational DBS: ER modeling, relational model, SQL, normal forms, page

buffers and indexing. We clearly state that we consider this relevant pre-knowledge for the course and only re-fresh it as a basis for further discussions.
3. (2 lectures) Distributed Data Management: General considerations and techniques, like partitioning, replication and synchronization, consistency definitions or consensus algorithms
4. (0,5 lecture) NOSQL decision tree: Based on [3]; provide mental framework so that students can classify the DB systems that are taught further on
5. (1-2 lectures each) NOSQL database systems: For each system class, we cover data model (in contrast to the relational model), query language, references to implementation techniques (if applicable, from distributed data management lecture); examples of implementations that fall into the system class.
6. (1-2 lectures) Further models and outlook. Additional topics and systems that weren't covered so far, like Geo databases or time-series databases

   In 2020, we covered the following NOSQL systems:
   - Extensible Record Stores; e.g. Cassandra, Google BigTable
   - Document Databases, including XML databases; e.g. MongoDB, XPath in Postgres
   - Graph Databases; e.g. Neo4J, Apache Tinkerpop
   - Key-value stores, Map and Reduce; e.g. Redis

2.3 Exercises

To achieve the learning outcomes, practical exercises are essential. We had two types of assignments: (1) Task solving with provided database systems, and (2) Task solving on paper as a preparation for the (written) exam. In this paper we focus to the first type of assignments; a short description on the second type (and the written exam) can be found in Section 2.4.

To teach the differences between the different NOSQL systems, we decided to use the same use case and dataset for all database systems, and *the same set of tasks*. This was carefully designed so that for each database, some tasks were solvable with a single query and clearly hit the sweet spot of that system. At the same time, for each database, some tasks were very hard or even unsolvable without significant workarounds.

For solvable tasks, we expect queries as a result. Yet, for unsolvable tasks, students need to write application code. Only if it is required in the exercise sheet to use a user-defined function for a task, we expect the submission of application code; otherwise we expect descriptions on how application code could solve the task not solvable by queries. These *unsolvable tasks* are the key contribution of our teaching concept. They correspond to the mentioned situation that a new task should be solved with a given database system and it is unknown whether the DBS is suited for this task or not. Of course, we did not inform the student up-front whether a task was solvable or not; however, we gave a maximum time for each task (solvable or not) and added the following information to the task sheet:

> *Please do not work longer than the specified time on a task! If you think that you will not be able to finish the task in the given maximum time, stop working on it 15 minutes before the end, and provide an explanation containing the following information: Whether you think that the task is solvable with the current system at all, and why? If you think that is solvable with more time: which approach would you try out next?*

2.4 Exam

It is always a challenge to map the targeted work or study situations to exam tasks. For the given competencies, different exam types are possible; so far, we did oral exams and written exams. Since some of the competencies can be shown best in the exercises, we rewarded the active participation with bonus points that have an effect on the final grade.

In designing the exam tasks, we aim for providing the context or situation in which the task should be solved, e.g., by describing a certain situation with a customer. It is particularly hard to test the competency to write queries for a given database system, since this often relies on several rounds of try and error, which cannot be provided in a written exam. In addition, if a solution is provided which is not correct, it is very hard to judge how it corresponds to the general understanding of the student. Hence, for queries, we often rely on the competence to spot errors or missing parts in queries by providing partial or erroneous query code that should be commented or completed.

Another important competency is the ability to find out the criteria that are important for the customer. This can be much better tested in an oral exam, where the examiner can play the role of a non-expert who just provides a starting problem, and the student being the expert who will find the best solution by asking questions and following up based on the given answers.

3 Use Case for Tasks

As a common dataset, we used a subset of the Enron corpus [7] (a large set of email messages). Figure 1 shows the data model that we provided to the students with arrows denoting foreign key references.

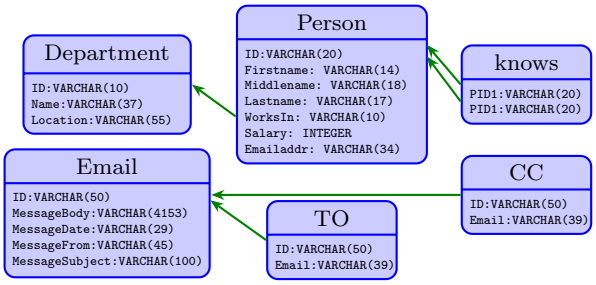

Fig. 1 Common data model for exercises

Our set of tasks on these data was the following:

1. System preparation: This task is DBS specific. It describes how to set up the specific environment needed to solve the tasks.
2. Equi Join: Three different tasks where data needed to be combined based on equality of certain attributes, like *List of people with their department.*
3. Theta Join: Combination of data based on a comparison that is not an equality, e.g. between salary and the number of emails: *Do people that earn more than the average salary in their department write more emails than those who don't?*
4. Schema Evolution: Add an attribute to a table, and add information to an entity set with a default value.
5. Missing values: *Find missing values for each attribute of the e-mails. Which attribute has the most missing values?*
6. Network analysis: Investigating the social structure of the Enron company based on the email corpus. Each employee can be seen as a node in a network where edges are represented by the emails sent from one employee to another or by the "knows" relationship between two employees. The tasks here are to find out network size by the *knows* relation, to find out which people are in the 2-hop email network, and to find out who sent emails to exact 7 TO-recipients (count outgoing edges).
7. User defined function: Creating a word count example as a UDF and add it to the database.

Out of these tasks, some are straight-forward for a relational database system (like the JOIN-operations or the schema evolution), while the network analysis would require many self-joins with an unknown number. When designing these tasks, we had six DBS categories in mind (Relational Databases, Extensible Record Stores, Analytical Databases/Column Stores, Graph Databases, Key-Value Stores, Document Databases). We categorized the tasks into three levels of difficulty (easy, medium, hard). In addition, some tasks are solvable and some tasks are unsolvable or too complex (which we consider unsolvable). Based on these categories, a task can be classified into six classes like: easy/solvable, easy/non-solvable, and so on up to hard/non-solvable. To assign the maximum time for each task we considered the level of difficulty and solvability of the tasks.

With some databases (in particular, Graph DB or Document Databases) students need time to get familiar with the query language. Therefore, a medium level of difficulty is considered for the first task. Yet, in other databases (like Relational Databases) the first task with an Equi-Join can be considered easy/solvable. In a database in which the operator *join* is not supported (like Cassandra) it takes some time for the students to realize this deficiency. Hence, the first task in Cassandra is considered as hard/non-solvable. Subsequently, the next tasks that need a join are considered as easy/non-solvable. Another example considered as a hard/non-solvable task is Network Analysis in a Document Database like MongoDB – although still some clever students provided some ingenious solutions by going over the data in the Enron dataset and gaining insight into the network. In contrast, Network Analysis is an easy/solvable task in a Graph Database.

4 System Environment

In order for us to provide a suitable environment for the students to solve the assignments using the proposed systems, we needed to answer the following questions:

- Where does the server run? (Cloud, On-premise server, Virtual Environment on Client Computer)
- What client application will be used for running queries? (Native Admin Interface or terminal with text-only access)
- Where does the client run? (Within a browser or on a client computer)
- What client computers are used? (Standardized, CIP pool / Student's Computers)
- How does the authentication work? (Accounts)
- How big is the dataset? How can it be inserted into the DBS?

We offered the course at two different universities and over the run of three years. This resulted in different system environments that we set up for each edition of the course. We discuss the setups in detail and provide a comparison matrix in Table 1.

Cloud Platform: In the first edition of the course in 2015 we cooperated with a company, who provided us with access to their (back then) recently published cloud platform in which the database systems were offered. As academic institutions we were using a free plan. Yet each individual student had to acquire an access key

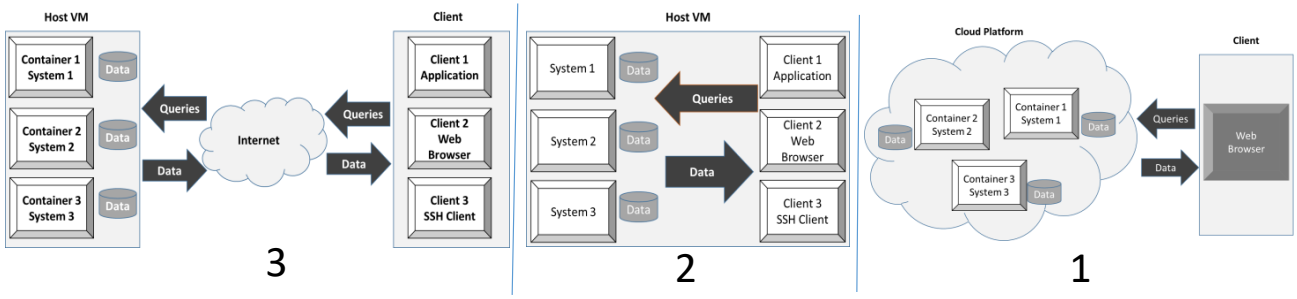

Fig. 2 The different environment setups for the hands-on exercises

that was valid only for a limited duration. In this platform, all the server systems ran in the cloud as depicted in Fig. 2 (1). The client application was a web interface that could be used from anywhere (students' own computers or thin clients in CIP Pool). The students got a free account for one month as a part of a trial plan with a limit on the bandwidth, data storage and the number of processors allocated for data processing.
This cloud solution neither requires database installation nor user management from our side; yet, we neither had options for database configurations. As the system was new back then, the web services for the database systems all had a different look-and-feel, although the platform had a common portal for login. We also experienced some system outages and connectivity issues (delay in response, etc.). Before the students could start using the systems, they had to import the data individually into the target database systems by uploading the data files. The trial plan imposed also a limitation on the data storage, which meant that only a part of the data could be stored. This architecture is mostly suitable for tutorials over a short period of time (as long as the trial plan), where the teaching goal is geared towards introducing the systems and getting a first hands-on experience. This form is unsuitable for usage periods that span over the whole course period, a complex set of task, and large data sets. A more mature cloud platform and a paid usage plan however might lead to a quite different teaching experience.

Virtual Environment: In a second edition of the course we installed VirtualBox in a PC pool. We then created images of virtual machines containing preconfigured database system installations that the students could load and then run on their PCs. In this setup – depicted in Fig. 2 (2) – the server side of all the database systems runs on the virtual machine, the client side of each database systems is also used from within the virtual machine. This setup offers more flexibility in the systems configuration and removes the constraints of connectivity and processing, since the students could use their own computers (or CIP pool accounts) for the tasks. This solution however assumes, that each student has access to an appropriate computer, which could exclude some students (who cannot afford computers – in particular in full online editions as described in the upcoming section). Yet, this solution guarantees that each user's data is completely shielded from other students. However, when the data to be used is too large, the image can become very big, which makes this solution less attractive. We encourage the use of this solution for tutorials that span a period of one semester or more, where the number of the participants is small (less than one hundred) and the data size is manageable (in the area of hundreds of megabytes).

On-premise Centralized Database Server: In other editions at the University of Bamberg, we set up a virtual machine on one of our servers – removing performance bottlenecks of individual PCs. All the database systems had their server sides installed on the virtual machine. All the server sides were installed in a docker container each for the convenience, isolation and ease of exposure to the outside that containers offer. Use of the systems follows one of the methods portrayed in Fig 2 (3):

1. Installation of a client application to establish a connection with the servers
2. Access the system through an included administrative web-based application
3. Direct access to the host container using SSH

While this solution enables the students to use all systems without having a powerful machine, the administrators have to take care of the authentication of the students (user creation and access management). It incurs more overhead for system preparation, but offers the benefit of relieving the students from the chores of system installation and data importing. This option is advised for tutorials that last a semester or longer, for large data processing tasks, and are destined for a very large number of students.

	Configurability	Self-contained-ness	Ease of use	Data Volume
Cloud (free plan) (1)	–	–	0	0
Virtualized (2)	+	+	0	–
Centralized (3)	+	0	+	+

Table 1 Comparison of system environments

5 The Summer Term 2020 Course

In summer term 2020, the whole course concept was turned into a full online course due to the Corona situation. In this section, we present how we conducted the course in this special semester at the University of Bamberg. The exam was done as a usual written exam.

5.1 Lecture

We pre-recorded videos on the lecture contents which were between 10 minutes and 40 minutes long with a clearly defined topic. To provide some structure, we kept the original lecture times (weekly) and used that data to live-stream the videos (called "watch party") with a parallel live chat (on-premise Rocket.Chat installation) for questions. Between two videos, there was a short pause for Q+A in the chat. In the announced Q+A chat meetings, the aim is to answer students' questions in the chat quickly. Later on the same day, the videos were available for asynchronous watching. The live chat was available during the whole course so that students could ask questions whenever they watched videos. Besides, students could use the forum and email asynchronously to contact supervisors. The last session was done as a live event with no recording. Students could either join the Q+A session at MS Teams with their university account, or watch the live stream of this session and ask questions over Rocket.Chat.

In contrast to a parallel Bachelor course, the interest of attending the live streaming sessions dropped significantly after some weeks from 80 attendees to 20 attendees, so that we changed that to video provisioning only after week 7. In addition to our own content, we also included a public virtual guest lecture (over YouTube) in the course, where we could invite students from several courses and universities, which would not have been possible in a physical setting.

5.2 Exercises

The main goal of online exercise slots was to help students start working on the assignments. We introduced each database and provided the database schema and information for system preparation. An important step is to provide a database schema for the students that represents the structure of data in the database. In order to make each database efficient and show the characteristics of each database, we needed to design different structures for different databases. For example, a *join* operation is not supported in Cassandra and not efficient in MongoDB. Yet, we can use data duplication to create an efficient and fast data access for both Cassandra and MongoDB. To emphasize that, we designed suitable keyspaces or collections considering the tasks in the assignments. For the system preparation in the first edition of the course, as mentioned before, students just needed an access key for the IBM cloud platform. In the exercise edition with our centralized database server, a system preparation on the client side was needed. The appropriate information regarding the suitable client side preparation was provided by us. For example, to execute queries against Postgres, students had to install Pgadmin on their computers.

6 Evaluation

We present an evaluation of the exercises from the Summer term 2020 course (see Section 5). Students were provided with evaluation forms that they had to fill in for each database system. The evaluation was not only for grading but also estimating our success in conducting the course.

Evaluation Setup. To evaluate the solutions of students participating in the exercises, students were asked to enter their executable queries, partial queries or descriptions in a solution sheet. The evaluation and grading of the solution sheet was be done automatically. In addition to the solution of tasks, students were asked to enter more information for the evaluation: Students were asked to record the actual amount of time that he/she spent on each task, as well as a self-assessment whether the solution was a correct one and what was their perceived level of difficulty of the task. The evaluation forms were submitted as CSV or Excel files. We set up a KNIME workflow that reads in the files, does the appropriate data cleaning (like conversion of the durations into minutes), calculates the intended statistics forming the input for the presented PGFPlots.

Results. As a first indicator of the attractiveness of the course we measured the overall participation in the Summer 2020 course. It turned out that roughly 55% of the students who participated at the beginning stayed until the end of the course and submitted result sheets:

In total, 120 students registered for the exam, 94 attended the exam. 90 students participated in the practical exercises and gained at least some bonus points. Yet we experienced the usual effect of decreasing participation during the course with 87 submitted (and evaluable) result sheets for Postgres, 56 for Cassandra and 47 and 46 for MongoDB and Neo4J, respectively.

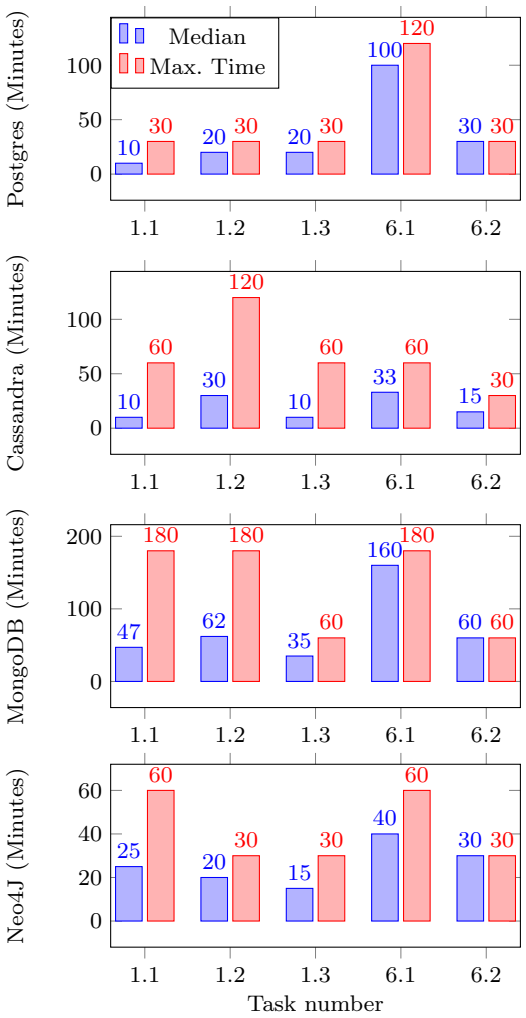

Fig. 3 Working times (Median) for selected subtasks

Due to space limitations we cannot evaluate all tasks here but focus on four exemplary subtasks as shown in Table 2. First we considered the actual time students spent on each task in comparison to the maximum time allowed. Figure 3 shows the median of the working time reported by the students. We can observe that students mostly stayed well in the limits of the assigned time and hence our judgement of the maximum time was justified. Only in the network analysis Subtask 6.2 several students tried to find a solution until the maximum

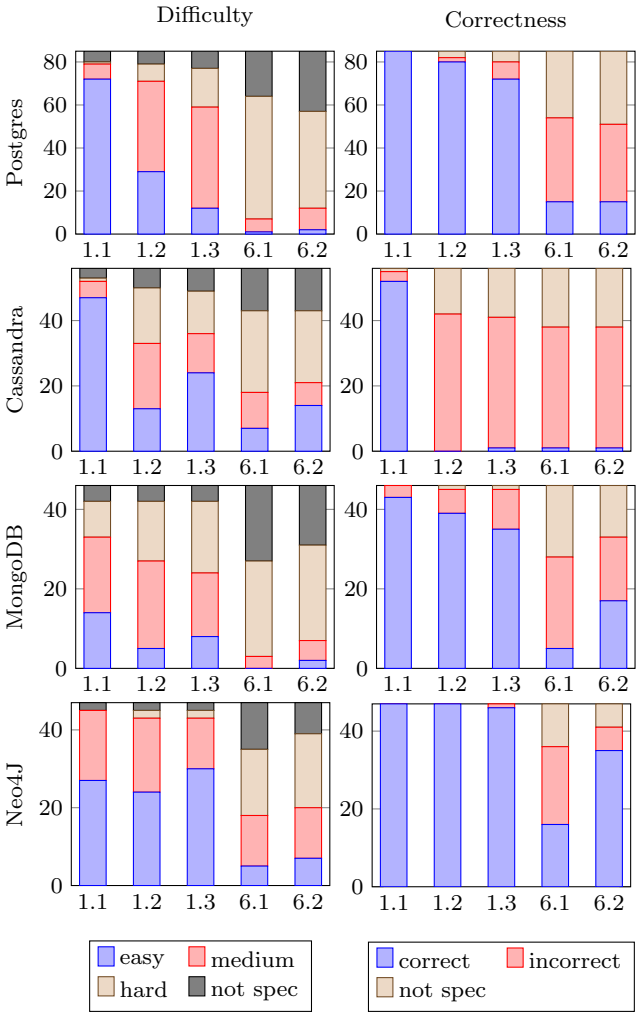

Fig. 4 Difficulty levels (left) and correctness (right) in total number of submission for selected subtasks

time – notably, even for Neo4J for which we considered network analysis an easy and solvable task.

Next we compare the self-assessment of the students (in terms of level of difficulty and assumed correctness of the result) with our prior assumptions in Figure 4. We can observe that for Subtask 1.1 our intention coincides with the students assessment as most students reported “easy” for Postgres/Cassandra while most reported easy or medium for MongoDB/Neo4J and nearly all students reported correct results (hence showing solvability of the task). Regarding Subtask 1.2 opposed to our prior judgment several students reported “medium” even on Postgres showing that our assumption of a decent knowledge of SQL was not met by some students. For Cassandra, Subtasks 1.2, 1.3, 6.1, and 6.2 are considered as hard due to absence of the join operator. However, this reasoning is the same for all the subtasks. Hence, once the student learns this in Subtask

Subtask#	Task	Postgres	Cassandra	MongoDB	Neo4j
1.1	Equi join	easy/solvable	easy/solvable	medium/solvable	medium/solvable
1.2	Equi join	easy/solvable	hard/not solvable	medium/solvable	easy/solvable
1.3	Equi join	medium/solvable	hard*/not solvable	hard/solvable	easy/solvable
6.1	Network analysis	hard/not solvable	hard*/not solvable	hard/not solvable	easy/solvable
6.2	Network analysis	hard/not solvable	hard*/not solvable	hard/not solvable	easy/solvable

Table 2 Subtasks selected for evaluation; *: easy to see unsolvability by analogy (absence of join in Cassandra)

1.2, the rest of the subtasks can be considered easy/not solvable by analogy. A slight mismatch with our judgment was also the case for Neo4J where several students reported "medium" which shows that students needed some more effort to get used to Cypher and Neo4J functionality. In terms of solvability the results matched our assumption: with a majority reporting "correct" for Postgres, MongoDB and Neo4J (solvable) but a majority reporting "incorrect" for Cassandra (not solvable). Regarding the other subtasks for Postgres, Cassandra and MongoDB the results mostly matched our expectations – although some students reported correct results for MongoDB for Subtasks 6.1 and 6.2 (which we considered unsolvable) which may be due to the fact that students developed complex workarounds. Yet opposed to our expectation, for Neo4J where we considered the network analysis tasks as "easy", we can observe a significant number of students reporting a level of "medium" or even "hard" and – in particular for Subtask 6.1. – many students reporting incorrect results. We aim to investigate whether these problems lie in understanding the concepts of graph databases or in a technical difficulty with Neo4J.

In future editions of the course, we will extend these evaluations. We plan to find out more about the actual prior knowledge of students in terms of SQL/Postgres by asking students to provide a self-assessment before and after the exercises with the SQL DBS. In the same manner, we can assess the learning curve of students in terms of NoSQL DBSs by asking for a self-assessment before and after the exercises with the other databases. This will also enable us to analyze an effect of a good prior SQL knowledge on the learnability of NoSQL systems.

7 Conclusion

We presented the concept and evaluation of a comprehensive Master degree course that teaches the theoretical differences between diverse data models and compares database systems from different categories. We received in general a very positive feedback from the students welcoming the comprehensive overview over DB solutions and the in-depth practical experience. We aim to continue the evaluations over the upcoming editions of the course. We may even include other database systems in our portfolio and obtain a comparison to the previously used ones.

Acknowledgements This course was further supported by Nasr Kasrin, Stefan Schwarz, Tim Waage, Ingmar Wiese, and many student teaching assistants. We'd further like to thank all students for their participation in the evaluation and their valuable feedback.

References

1. Council of the European Union: Council Recommendation, of 22 May 2017, on the European Qualifications Framework for Lifelong Learning and Repealing the Recommendation of the European Parliament and of the Council of 23 April 2008 on the Establishment of the European Qualifications Framework for Lifelong Learning. Official Journal of the European Union pp. 15–28 (2017)
2. Fowler, B., Godin, J., Geddy, M.: Teaching Case: Introduction to NoSQL in a Traditional Database Course. Journal of Information Systems Education **27**(2), 99 (2016)
3. Gessert, F., Wingerath, W., Friedrich, S., Ritter, N.: NoSQL database systems: a survey and decision guidance. Computer Science - Research and Development **32**(3-4), 353–365 (2017). DOI 10.1007/s00450-016-0334-3
4. Gessert, F., Wingerath, W., Ritter, N.: Scalable Data Management: An In-Depth Tutorial on NoSQL Data Stores. In: B. Mitschang, N. Ritter, H. Schwarz, M. Klettke, A. Thor, O. Kopp, M. Wieland (eds.) Datenbanksysteme für Business, Technologie und Web (BTW 2017), 17. Fachtagung des GI-Fachbereichs ,,Datenbanken und Informationssysteme" (DBIS), 6.-10. März 2017, Stuttgart, Germany, Workshopband, *LNI*, vol. P-266, pp. 399–402. GI (2017). URL https://dl.gi.de/20.500.12116/937
5. Kim, S.: Seamless Integration of NoSQL class into the Database Curriculum. In: M.N. Giannakos, G. Sindre, A. Luxton-Reilly, M. Divitini (eds.) Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE 2020, Trondheim, Norway, June 15-19, 2020, pp. 314–320. ACM (2020). DOI 10.1145/3341525.3387399
6. Kleppmann, M.: Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. O'Reilly (2016)
7. Klimt, B., Yang, Y.: Introducing the Enron Corpus. In: CEAS 2004 - First Conference on Email and Anti-Spam, July 30-31, 2004, Mountain View, California, USA (2004). URL http://www.ceas.cc/papers-2004/168.pdf
8. Wiese, L.: Advanced Data Management, For SQL, NoSQL, Cloud and Distributed Databases. De Gruyter, Berlin, Boston (2015)