Teaching about Data and Databases: Why, What, How?

Alan D. Fekete
University of Sydney
alan.fekete@sydney.edu.au

Uwe Röhm
University of Sydney
uwe.roehm@sydney.edu.au

ABSTRACT

The panel on data(base) education at VLDB2021 [13] drew attention to important challenges in choosing how database classes are constructed for students in a world where data is being used in novel and impactful settings. This paper aims to present one view of a process for making these pedagogy decisions. We don't aim to present a best-possible design of the subject, rather we want to illuminate the space of possibilities, to encourage reasoned choices rather than simply teaching the subject as it was previously offered, or spending time on the latest innovations without considering the "opportunity cost" of doing so. We hope to guide the perplexed instructor or departmental curriculum committee.

1. THE CHALLENGE

Any computing education will include material about data management, but there are many alternative ways a class on this topic might be structured. There are so many topics already that seem valuable, and new ones are gaining importance, but a degree would not be long enough to learn it all. The standard textbooks like [30, 25, 12, 23, 20] are each over 800 pages. Class time and student attention are scarce resources. So there are hard choices facing the instructor or curriculum committee who want to decide what to teach, and how to teach it. If we include the alternative approaches to computing distributed joins, what would we leave out to make room? Which of the sophisticated aspects of SQL will be taught and assessed? Which (if any) normal forms are crucial? Which (if any) concurrency control mechanisms are covered? Should we replace XML by graph data and its query language? Do students need to know how to code insertion into a B+-tree, or to use an object-relational mapping framework? Is blockchain ready to be a core topic?

This paper aims to characterize some of the design space of data management teaching, and we advocate making *principled* decisions within the space. Our approach comes from *constructive alignment*, where the learning objectives are chosen (and announced to students), and then one picks activities for students to carry out, and especially the assessment tasks, so these will help the students achieve the intended outcomes. We go a step further, and argue that the choice of detailed learning outcomes, should be guided by an overall purpose for the class, which takes account of career paths students will be prepared for, and other wider curricular goals.

2. CHOOSE THE PURPOSE(S)

Why would a student take a data-related class, and why would a department offer it? Just as there is too much known about databases, to fit it all in one class (or even two), so there are too many topics in computer science, for a student to learn them all. One justification of devoting some of the scarce teaching attention to data management may simply be accreditation: some coverage of this is required for all students by standard curricula such as ACM Computer Science [7] and ACM Data Science [28]. But the curricula aren't detailed enough determine the focus of a data-related subject.

For most faculty, their teaching is driven by the goal to contribute to students' intellectual growth. This view would tend to favor using the available time for fewer topics in depth, rather than a broad and shallow introduction, but again, how should we choose which aspect to explore deeply? One consideration might be if there is a crucial style of thinking that is neglected elsewhere in the curriculum, and we can then pick the topics within data management that develop students' aptitudes in this important degree-wide outcome. For example, the database subject has often served as an exemplar of larger-scale software design, teaching students the way a complex system can be structured in layers and components with clean interfaces. A quite different role for the database subject, can be as the main exposure for technically-oriented students, to awareness of business needs and organisational is-

sues. Coverage of normalisation can be a vehicle demonstrating the capacity of theory to have practical impact. In some curricula, SQL may be the best hope to awaken students to the power of declarative programming.

While faculty can take a larger view of their purpose, students often engage better when they see how the skills and knowledge contribute to their professional future. Many career paths are available from a computing education, and knowing which paths are most common for the students in a class can help identify which topics are most worth including in the limited time available for a subject. Even for the traditional relational database class, some topics were most natural for the future developer of database-backed applications, others for someone who will become a database administrator, and it is different again for an aspiring data analyst, or for the potential implementor of a platform. Now, with the spread of data science, there are extra paths such as data engineer (creating and tuning the analysis pipelines), or even the domain expert from other disciplines who will need to manage their own datasets and explore them. Because many smaller organisations can't afford a whole team for data science, so a single student may graduate into a role where they have to clean and integrate datasets, write data analysis programs, install and tune data storage, explain the results, etc. As we discuss in the next section, the students' intended path can shape the amount to content on different topics, as well as the approach taken to lectures and assessments. The choice of career focus in a subject may be influenced by local industry needs, and also by the alternatives offered nearby: for example, if a Business School is producing strong Information Systems graduates for the local region, then a CS department has less reason to stress data modelling for its students.

A final consideration at this level of the design process, is how the subject fits into the overall curriculum, and any institutional constraints. If a subject is counted in several majors, there is usually a struggle to ensure that it works equally well for the different cohorts. One important issue is the expected background knowledge. If a class is intended to accept data science students, we should probably assume less experience with computer organisation, and perhaps they haven't knowledge of object-oriented programming. If all students have already learned first-order logic, the way we teach query language can exploit it; similarly conceptual data model topics might be abbreviated or eliminated when students already have done object-oriented domain models and UML.

Of course, some departments have the resources to offer more than one database course; this raises really interesting choices in how to divide the material. One common pattern is to have a lower division subject focused for application developers, and follow in higher years with a class on system internals, where the future database implementer, or data engineer is the target. At some institutions, there are different introductory classes in different study paths, for example, one with a business focus in the information systems curriculum, a more technical one for computer science, and/or one where data science students learn how to scale analytic tasks for data beyond the in-memory processing of their favorite language.

Our intention is not to argue that one purpose is better or worse, merely that the instructor needs to understand the purpose and ensure that the way they teach is well-matched to it. It is motivating for students if they understand why a subject is offered, and where different topics fit into their growth as a professional, providing both career skills and also more general aptitudes.

3. SELECT THE TOPICS AND DEPTH OF TREATMENT

There are strongly held views among faculty and employers, on what students need to know; unfortunately, there is insufficient time in a degree to meet all the requests for content. We look here at several of the main facets of the subject, within which choices must be made. In each case, we expect that any class will cover the concepts at some level, but there is large variation possible: for example, a learning outcome "Students will recognize when a schema involves redundancy and possible inconsistency in the data storage" could be covered in much less time (and with much less assumed math background) than "Students can use inference rules to deduce consequences of a set of functional and multi-valued dependencies"; in between might be "Students can produce a BCNF schema for a collection of attributes related by functional dependencies". Deeper coverage of one direction will inevitably leave less time for another.

3.1 Diverse data models and data types

The clean, 1NF, relational model has a magnificent theory and is a joy to teach. It is naturally at the core of any data management class no matter the expected career path: much traditional enterprise data is structured this way, and many data science datasets come in csv format with a tabu-

lar structure, and are processed with dataframes for which the relational theory fits fairly well. But there are many other *NoSQL* platforms; at the least every graduate needs to know enough to have sensible conversations with enthusiasts for graph databases, document stores, and key-access column-family stores. In some settings, some students will need to work with, or develop the internals of, one of these style of system. Whether to merely show the concepts, or to give practical experience with systems, is a hard decision for the course designer, as reaching real understanding the strengths and weaknesses of each approach takes a lot of class time.

Future platform developers may well work on systems supporting these diverse models, but the main ideas of system internals seem likely to transfer well from relational systems, so these careers may not need much class focus on the new models. In contrast, data engineer and science careers are likely to need skills with stores that support these models, while for enterprise application developers the focus may instead be on how to shoehorn data like this into a relational representation. Many relational platforms have added support for columns declared as json or graph. Unfortunately, these extensions are often proprietary, and so education about the details may not transfer well across the diverse platforms students will need to use in later careers. and also the query facilities for a typed column are usually much weaker than in a dedicated store for the type. Thus, it is worth discussing how to shred the data into a standard 1NF relation, and write SQL queries to recover some useful information.

As well as json and graph, there are many significant data types beyond string, integer, float! Working with time and date data is crucial for any career path, and any likely data set of interest will include temporal aspects, so covering these seems inevitable. Students need to know the distinction between time instants and time periods, different ways of dealing with current data (a period whose end has not yet been determined), how to query the state as-of a given time, and a variety of temporal joins.

A class aimed at data science careers will also want good coverage of geo-spatial data. For example, in our university, the data science curriculum core class (DATA2001) presents conceptual material on spatial data. We stress the differences between points, regions, paths, and trajectories (and different approaches to represent them), and also the importance of the coordinate system and of topological relationships between spatial objects. We introduce some of the operations and functions which are meaningful on spatial types, especially drawing attention to the high query costs even when indices are present, and the distinction between spatial join and a join on non-spatial attributes. The students get practice in labs, working with both PostGIS and Python geopandas; their project work also involves spatial data (Section 4.2)

3.2 Query languages

No matter what career path students will follow, they should have learnt enough about SQL or a similar declarative way to express queries. Even for data science students who may not be keeping data in a relational platform, most of the concepts of where-conditions, joins, group-by, etc, are found in dataframe processing in Python, or in systems like Apache Hive above HDFS for distributed analytics. However, even standard SQL has much more complexity than the basic SELECT-FROM-WHERE-GROUPBY-HAVING, INSERT, DELETE, UPDATE, handling of NULL (including outer join variants), and nested subqueries, that all textbooks cover. Beyond the standard, each platform's SQL variant has its own extensions and restrictions. An instructor needs to decide how much of this students see, and how much of that to stress in assessment. Future application developers will need to know many of the tricks covered in books like [5], such as how to do running totals or recursive aggregation for bill-of-materials; but these may not be important for other careers. Other parts of SQL such as table declaration and modification, constraints, triggers and access control, are often introduced to various depth, in connection with topics on schema design, security etc.

Data science frameworks like pandas have many of the concepts taught in SQL, such as filtering by a boolean predicate, grouping, joins. While the style of code, with an explicitly sequence of operators, is less declarative than SQL, it is close enough that students seem likely to transfer SQL skills to these frameworks without difficulty. Alternatively, students can be taught to program with a dataframe library directly, either a standard one or a simplified one for novices, as done in the famous Berkeley Data8 class [1] see `http://data8.org`. Another category of query language topics is for non-relational data models and advanced data types, as mentioned above.

What of the theoretical query languages? Relational algebra is a powerful notation for discussing even the simplest ideas of query processing, so we expect most classes to introduce this. It is hard to see career utility for relational calculus, but this

may be valued by instructors who want to stress the connections between theory and practice (for example, it can motivate some of the SQL tricks for universal quantifiers via nested NOT EXISTS clauses), or as a way to teach some logic in a curriculum where other subjects do not.

3.3 Internals

In many departments, the database class is presented as a systems experience, with lots of coverage of the algorithms and structures inside the engine, such as indexing, query processing and optimisation, transaction management, replication. For students who may work as platform developers, database administrators, data engineers, this is vital content, and it may be covered as a way of building general skills in this direction (if the rest of the curriculum does not do so). However, for students aiming to become application developers, this material may feel unmotivated; also it usually requires knowledge of low-level coding (eg C or C++), a skill which may not be produced in previous subjects. In our experience, one can find a nice subset of topics about internals, which are meaningful for application programmer careers, by looking for cases where the user's satisfaction is shaped by choices available at the application. For example, declaring an index can greatly affect application performance. The choice of class orientation shapes how time is spent and especially how the students are assessed. Teaching index structures for platform developers may lead to exam questions to trace the execution of an insertion, or estimate storage used in a structure. But for application developers, one might ask which index to create to improve performance of a given workload.

Concurrency control and recovery algorithms are sometimes excluded entirely from a first data class with a focus for application developers (eg in [34]). However, some knowledge of transaction concepts must be provided for these students, because enforcing isolation impacts the tradeoff between integrity and throughput [9]. We believe that a crucial skill for application coders, is identifying what steps need to be combined into a transaction. Developers need to be aware of the kinds of integrity violation that can arise when their code runs at default isolation (which is Read Committed rather than Serializable, in most engines).

Data science work often does not use a platform where an index or isolation level can be set declaratively, and where query optimization is done by the engine; instead the programmer will code support for these aspects in the application. For these careers, it will be important to know at least the simplest techniques and how they influence performance at scale. Replica coordination and distributed processing are of particular importance in a class for these students.

3.4 Application structure

If a class is a path to application developer careers, it needs to give students awareness and practice of some ways that a data-backed application can be coded and deployed. An important skill for these graduates will be to recognize the tradeoffs between different architectural approaches, such as the explicit writing of SQL or having it generated through an ORM such as Rails, separation of the code into tiers, placement of tiers across a network (including whether to use cloud-hosting, and differences between self-managed databases versus cloud-provided data services), and where and how to maintain state. This is a topic where the technology changes rapidly (and often a programmer's options are constrained by organisational imperatives). This makes it hard to choose core concepts that will still hold their value for the students when they reach the workforce. It is a challenge to find teaching resources, as textbooks are often obsolete, and online resources are rarely written with a broad and platform-independent viewpoint. It may be that the best approach is to provide students with a sample codebase to explore, and to discuss some alternatives as a way to model the architecture decision.

3.5 Conceptual models and schema design

Learning to create an E-R diagram for a domain, and how to achieve normal forms for schema, have long been core topics in a database class. The relevance to many future enterprise-centred careers for these seems hard to see in current times, when little work is done in a greenfields setting; rather, application developers and DBAs and data scientists typically work with data whose storage and schema are already established. One approach we have found useful, is to motivate in the context of expanding an existing design to include extra data for extra functionality. Normalisation can also be a wonderful topic for building students skills with logic and discrete math, though perhaps one might limit discussion to one or two normal forms.

In a class for data science, there are topics related to logical design that are important, but are not part of classic relational design. One issue that is crucial for working with charting libraries, and machine learning, is the different ways to treat related

attributes. In standard theory, attribute names are atomic, but real-world data often arrives with attributes whose name includes the year, location, category etc (eg"temperature-Sydney" or "responses-2020"). For a dataframe, there are alternatives between wide schema (with a family of related attribute names) or long schema (where the varying aspect is made a separate attribute), and libraries like pandas have functions to transform data between these formats.

3.6 Security, integrity, and resilience

The traditional database curriculum includes a scattering of topics such as views, access control, integrity constraints, triggers, recovery, SQL injection threats; more recently, differential privacy and blockchain have emerged as headline topics. These can feel to students like minor and unrelated technical details. We suggest that they can be motivated when brought together under the heading of data quality and use, and combined with discussion of the ethical decisions. One important distinction when teaching students whose career may lie in data science, is that the management of these properties is not typically provided by a platform, but instead needs careful implementation in code and file settings, etc.

4. CREATE LEARNING EXPERIENCES

Pedagogy theory tells us that active learning is effective. So even a traditional class with content-delivery lectures, will include time for tasks where students perform actively, perhaps during small group discussions with a TA, or as homework. The choice of which activities to assign, has a huge impact on what the students learn in the subject. We next discuss two categories of active tasks: writing queries, and completing a semester-scale project.

4.1 Query writing

As mentioned above, a first data management subject will teach SQL or a similar declarative approach to queries. As when mastering any programming skill, there is no effective substitute for hands-on experience. A very common style of task which students are told to do for practice, and for assessment, is to write the SQL to calculate some result, given the natural language description of the purpose. For example,"Here is a schema .... Write a SQL query to show the employees by name, and their department, where the employee's salary is more than 10% of the highest salary paid in the department."

It is natural to set query-writing tasks in domains that are relevant to students, and also where they have background knowledge to help them understand the data and the intention behind each query. So a remarkably common choice is a student-record domain, with tables of student, subject, grade, instructor, etc. However, a class centered on business applications can use tables like Employer, Customer, etc; or data science students may work with real datasets (weather, census, movies, etc).

For effective pedagogy, the query-writing tasks should be ordered over at least several weeks, so that students are gradually exposed to more complex SQL constructs and concepts, after they have mastered easier ideas. Experiences from teaching programming shows that combining constructs is itself a complex step, and is better done after the students are comfortable with each of the sub-techniques being combined. So, one should give practice with simple aggregates over a table, and practice with simple filtering of a table to find the rows that satisfy a condition, before students are asked to write queries that aggregate over a subset of rows. Natural joins, and select from a single table, should both come before queries that join and also select. There are however constructs which seem independent of each other, and whose comparative difficulty is not so different, and thus there is a choice in the ordering. Should aggregation over a table come before, or after, selecting rows from a table based on a simple condition? What is the better sequence, for learning correlated subqueries and grouping?

A central challenge in giving students programming practice, is how to provide feedback. Rapid indication of their mistakes is invaluable, and students also gain from helpful advice to undo any misconceptions revealed by their errors. In smaller classes, a teaching assistant can provide this face-to-face, but support from teaching software has advantages of accessibility on-demand, allows students to practice as much as they wish, and scales well as student numbers grow. We now discuss some of the types of tool that have been proposed in both database and computing education communities.

Early work [22, 27] was based on the concept of an Intelligent Tutoring System which takes the student through a sequence of tasks for which they write SQL queries. It provides guidance, with error messages that help a novice understand why their answer is wrong (eg "This task requires a GROUP BY clause" or "you do not mention the correct attribute"). The next task given to a user is often personalised based on a learner model, built up from the previous mistakes and correct answers, that reflects which topics need practice. Another kind of

tool derives from algorithm animation. Here the goal is to help a student develop their understanding of SQL semantics, but showing step-by-step, how the output of a query is determined from the database contents [15]. The pedagogy assumption is that students will write queries more sensibly, if they understand the "notional machine" and therefore can predict what a query text is calculating. A similar system `https://pandastutor.com` has been produced for pandas dataframe manipulations.

Another type of tool, especially driven by MOOCs, has built systems for automated grading of SQL queries. These can check a query by comparing the output to what is required; often the query is run against a variety of database states, to pick up cases where an incorrect formulation happens to give the same answer in a particular state [18, 26]. Our experience though is that it is actually quite hard to phrase a question so the meaning is unambiguous, and no reasonable interpretation of the wording will be marked incorrect. An interesting proposal [8] combines automated checks with peer review. Some systems such as Ullman's Gradiance `https://www.gradiance.com/STwelcomeAWS.html` generate personalised question wording for each student (but testing the same constructs), by filling in a template with randomly chosen elements. An interesting recent enhancement of grading systems has been to automatically generate input database instances, specifically chosen so that mutated queries will give different output than the correct query text [4].

4.2 Project work

Due to strong practical relevance, it is widespread that each computing class (except the most theoretical) will have a significant project as a large component in the assessment. For a data-management class, there are two quite different ways this happens: project work in the platform implementation, or project work in building a data-backed application. While some very intensive subjects may give both projects, more often, an instructor chooses between these directions, and the decision naturally would impact the overall career focus of the subject. However, there are plenty of axes of choice within each style of project.

For a project looking at platform internals, it is usual to work with an existing platform, and in a sequence of tasks, students either rewrite some component, or extend that component (eg provide an alternative page replacement policy, or include a new index type or join algorithm). This style of project was famous from the Carey and Ramakrishnan Minibase code from Wisconsin, written in C++ and supporting a subset of SQL; this was provided with [25] (and inspired by an older Minirel system from Dave DeWitt). Alternative simple pedagogic platforms have been produced in different languages, with different query interfaces [24, 32, 29]. Ailamaki and Hellerstein [2] argued for instead having project work that uses a full-power open-source DBMS such as PostgreSQL. Broadly speaking, projects based on a simplified pedagogic platform pose less cognitive load, with cleaner interfaces (and less interdependency) between components, but they can be less motivating, and also they provide less preparation for future internship and career activities. A different approach has students implement all components of a simple dbms from scratch [31].

With an application development project, one essential difference is between a greenfields work, doing the whole application from scratch, versus extending a code-base that is provided to the students (for example, creating code for additional use-cases, which involve modifying the schema, creating appropriate extra tables and indices, and writing new transactions). In either case, there are choices for an instructor, in the technology used for the development. There are of course different options for the data management platform, and also choices for the application development tools, such as an object-relational mapping layer, a focus on putting the business logic in stored procedures, or use of explicit code working with cursors. The instructor also needs to decide which stages of the life-cycle to emphasise in a project. If the class has a focus to application development careers, more emphasis is often given to requirements elicitation; while educating future DBAs might devote significant work for conceptual modelling and schema improvement, then physical design.

What would be suitable project work in a class where data science is an important focus? It is likely to focus more on complex analytics, and less on transactions and updates. Also, a lot depends on the decisions about specific content topics that are covered. For example, if the class has looked at a variety of data types, then it is natural to include those in the project work. Here is one way, that we use at our institution. Our Data Science major contains a (compulsory) second-year course on Big Data and Data Variety (DATA2001). A central part of this course is a project-based assignment where students develop a complete Data Science pipeline from data ingestion, data transformation and cleaning, to data analysis and visualisation. We provide students with a real-world scenario including

a spatial-temporal dataset which students have to combine with other datasets, typically at least one CSV and some JSON/XML data via a Web API. The scenarios are inspired by local circumstances, so for example in our case a bushfire risk analysis for Greater Sydney, or the computation of a cyclability score for different suburbs of the city. In order to teach students the declarative power of SQL, and also the scalability benefit of appropriate indexes, these scenarios cover at least one spatial join. Students develop their pipeline with Jupiter notebooks in Python, but are explicitly encouraged to keep the spatial data in PostgreSQL and to use its PostGIS extension for the spatial join(s). With a spatial index, this approach is typically much more scalable than using a pure in-memory Python implementation with GeoPandas.

As explained in Section 3.2, a background on databases and declarative data processing with SQL is also very helpful for understanding the challenges of big data processing. We found this enables us teach high-achieving students to scale-out their data analysis using distributed data processing platforms, such as Spark, and to gain some first experiences with scalability evaluations.

Whichever choices are made, it is vital that the project work be well supported, as any problems with server capacity, security restrictions on access, etc, will distract the students from the learning goals, and also add a lot of unnecessary time demands on them.

5. RELATED WORK

The central conferences of our community such as SIGMOD, VLDB, do sometimes include work about teaching [33, 4, 26]. Another source for ideas and information comes from the Computing Education community, and its main conferences such as SIGCSE and ITiCSE. Some papers here are in the style of education research, but the majority are *experience* papers that contribute to a "community of practice", where the author describes an innovation in teaching, and reflects on lessons learned that others could benefit from. Papers about database and data science education have appeared regularly in these forums. Here we draw attention to a few recent ones, beyond those we already cited. In [10] we show one way data management concepts can be taught in a sequence of subjects in a data science major. Including NoSQL concepts into a database class is discussed [16]. Using data about errors made in student tasks, SQL language topics were characterised [21], and similarly for document store queries [3] and graph data queries [6]. A tool was created for grading data designs (in UML) [11]. The impact of automated assessment tools on student learning has been studied [17]. A flipped approach to teaching an upper-division class on internals was evaluated [19], as was use of group exams in an introductory class [14].

6. REFERENCES

[1] A. Adhikari, J. DeNero, and M. I. Jordan. Interleaving computational and inferential thinking: Data science for undergraduates at berkeley. *Harvard Data Science Review*, 4 2021. https://hdsr.mitpress.mit.edu/pub/e69066t4.

[2] A. Ailamaki and J. M. Hellerstein. Exposing undergraduate students to database system internals. *SIGMOD Rec.*, 32(3):18–20, 2003.

[3] R. Alkhabaz, S. Poulsen, M. Chen, and A. Alawini. Insights from student solutions to mongodb homework problems. In *ITiCSE 2021: 26th ACM Conference on Innovation and Technology in Computer Science Education*, pages 276–282, 2021.

[4] A. Bhangdiya, B. Chandra, B. Kar, B. Radhakrishnan, K. V. M. Reddy, S. Shah, and S. Sudarshan. The xda-ta system for automated grading of SQL query assignments. In *31st IEEE International Conference on Data Engineering, ICDE 2015*, pages 1468–1471, 2015.

[5] J. Celko. *SQL for Smarties: Advanced SQL Programming, (5. ed.)*. Morgan Kaufmann, 2014.

[6] M. Chen, S. Poulsen, R. Alkhabaz, and A. Alawini. A quantitative analysis of student solutions to graph database problems. In *ITiCSE 2021: 26th ACM Conference on Innovation and Technology in Computer Science Education*, pages 283–289, 2021.

[7] A. Danyluk and P. Leidig. Computer science curricula 2013. `https://www.acm.org/binaries/content/assets/education/cs2013_web_final.pdf`.

[8] M. de Raadt, S. Dekeyser, and T. Y. Lee. Do students *SQLify*? improving learning outcomes with peer review and enhanced computer assisted assessment of querying skills. In *6th Baltic Sea Conference on Computing Education Research, Koli Calling'06,*, pages 101–108, 2006.

[9] A. D. Fekete. Teaching transaction management with SQL examples. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in*

Computer Science Education, ITiCSE 2005, pages 163–167, 2005.

[10] A. D. Fekete, J. Kay, and U. Röhm. A data-centric computing curriculum for a data science major. In *SIGCSE '21: The 52nd ACM Technical Symposium on Computer Science Education*, pages 865–871, 2021.

[11] S. Foss, T. Urazova, and R. Lawrence. Automatic generation and marking of UML database design diagrams. In *SIGCSE 2022: The 53rd ACM Technical Symposium on Computer Science Education, Providence, RI, USA, March 3-5, 2022, Volume 1*, pages 626–632, 2022.

[12] H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database systems - the complete book (2. ed.)*. Pearson Education, 2009.

[13] Z. Ives, J. Gehrke, J. Giceva, A. Kumar, and R. Pottinger. VLDB panel summary: "the future of data(base) education: Is the cow book dead?". *SIGMOD Rec.*, 50(3):23–26, 2021.

[14] J. Kawash, T. N. Jarada, and M. Moshirpour. Group exams as learning tools: Evidence from an undergraduate database course. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education, SIGCSE 2020*, pages 626–632, 2020.

[15] R. Kearns, S. Shead, and A. D. Fekete. A teaching system for SQL. In *Proceedings of the ACM SIGCSE 2nd Australasian Conference on Computer Science Education, ACSE 1997*, pages 224–231, 1997.

[16] S. Kim. Seamless integration of nosql class into the database curriculum. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE 2020*, pages 314–320, 2020.

[17] A. Kleerekoper and A. Schofield. SQL tester: an online SQL assessment tool and its impact. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE 2018*, pages 87–92, 2018.

[18] C. Kleiner, C. Tebbe, and F. Heine. Automated grading and tutoring of SQL statements to improve student learning. In *13th Koli Calling International Conference on Computing Education Research, Koli Calling '13*, pages 161–168, 2013.

[19] E. M. Knorr. Worked examples, cognitive load, and exam assessments in a senior database course. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education, SIGCSE 2020*, pages 612–618, 2020.

[20] P. M. Lewis, A. J. Bernstein, and M. Kifer. *Database Systems: An Application Oriented Approach, Compete Version, (2. ed.)*. Addison-Wesley, 2006.

[21] A. Migler and A. Dekhtyar. Mapping the SQL learning process in introductory database courses. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education, SIGCSE 2020*, pages 619–625, 2020.

[22] A. Mitrovic. Learning SQL with a computerized tutor. In *Proceedings of the 29th SIGCSE Technical Symposium on Computer Science Education, SIGCSE 1998*, pages 307–311, 1998.

[23] P. E. O'Neil and E. J. O'Neil. *Database: Principles, Programming, and Performance, (2. ed.)*. Morgan Kaufmann, 2000.

[24] M. P. Papazoglou and W. Valder. *Relational database management - a systems programming approach*. Prentice Hall, 1989.

[25] R. Ramakrishnan and J. Gehrke. *Database management systems (3. ed.)*. McGraw-Hill, 2003.

[26] U. Röhm, L. Brent, T. Dawborn, and B. Jeffries. Sql for data scientists: Designing sql tutorials for scalable online teaching. *Proceedings of the VLDB (PVLDB)*, 13(12):2989–2992, 2020.

[27] S. W. Sadiq, M. E. Orlowska, W. Sadiq, and J. Y. Lin. Sqlator: an online SQL learning workbench. In *Proceedings of the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, ITiCSE 2004*, pages 223–227, 2004.

[28] M. Sahami and S. Roach. Computing competencies for undergraduate data science curricula. `https://dstf.acm.org/DSTF_Final_Report.pdf`, 2021.

[29] E. Sciore. Simpledb: a simple java-based multiuser syst for teaching database internals. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education, SIGCSE 2007, Covington, Kentucky, USA, March 7-11, 2007*, pages 561–565, 2007.

[30] A. Silberschatz, H. F. Korth, and S. Sudarshan. *Database System Concepts, Seventh Edition*. McGraw-Hill Book Company, 2020.

[31] B. Sotomayor and A. Shaw. chidb: Building a

simple relational database system from scratch. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education, SIGCSE 2016, Memphis, TN, USA, March 02 - 05, 2016*, pages 407–412, 2016.

[32] G. Swart. Minsql: a simple componentized database for the classroom. In *Proceedings of the 2nd International Symposium on Principles and Practice of Programming in Java, PPPJ 2003, Kilkenny City, Ireland, June 16-18, 2003*, pages 129–132, 2003.

[33] J. D. Ullman. Improving the efficiency of database-system teaching. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 1–3, 2003.

[34] J. D. Ullman and J. Widom. *A first course in database systems (2. ed.)*. Prentice Hall, 2002.