Project Summary: SQLGrader

Drew Skwiers-Koballa
drew.skwierskoballa@gatech.edu

***Abstract*—**There is an industry desire for job candidates with stronger knowledge of relational database concepts while grading projects focused on relational databases in massively open online courses (MOOCs) is presently not well scalable. SQLGrader, a tool to automate grading and environment management for relational databases, was developed with the aim to facilitate grading student projects at scale. Successful completion of SQLGrader core functionality (Skwiers-Koballa, 2021) established a pathway to expand the coverage of databases in computer science education. Initial implementation was completed and further improvements are achievable on the extensible and open source framework.

1 INTRODUCTION

Class sizes are expanding to commonly include 100 or more students in traditional settings at the undergraduate level for database courses (Stanger, 2018) and massively open online courses (MOOCs) introduce a new breath of students to college-level material creating class sizes of several hundred students (Fini, 2009). The use of technological innovations to improve student outcomes and create efficiencies in grading has grown alongside the growth of MOOCs, with grading automation becoming commonplace in courses with large enrollments (Atwood and Singh, 2018). An autograder applies the principles of standardized rubrics and unit tests to generate student grades and feedback with little to no human intervention. The use of an autograder can also be extended to improve student access to feedback while a course project is in progress. In application programming oriented courses, the use of an autograder improves student learning opportunities and is often able to address the complexities of a particular subject's content (Cheang et al., 2003; Helmick, 2007).

Most curriculum for undergraduate degrees in computer science specify a minimum of a single course focused on databases and for further learning multiple courses on databases are available at the graduate level (Taipalus and Seppänen, 2020). Introductory courses on databases commonly include both writing queries (data manipulation language, DML) and creating data schema objects

(data definition language, DDL). By covering both DML and DDL, students have an opportunity to gain a well-rounded understanding of a functional database. Coupled with other software development concepts, an introductory course on databases may guide learners through building a basic web or console application.

The concepts provided by secondary database courses generally focus on object-relationships, query performance, and advanced data types. The stronger cases for success came from demonstrations of industry-backed demand for skill sets such as big data or advanced data types Dietrich and Chaudhari, 2011; Villa, 2016). Especially in the context of MOOCs, where a larger segment of students are focused on gaining education with direct impact on industry skills, the connection between content and skills desired in the workplace is paramount.

Ongoing evaluations of SQL code, whether written directly by a developer or generated by an ORM, have shown that this is generally a silent cost to a project in that it introduces few bugs but has a significant performance impact (Muse et al., 2020). Parsing queries with a abstract-syntax-tree is a robust option, allowing for flexibility in engine specificity and capable of handling large workloads. This was shown in previous weeks for parsing queries as well as by Nagy and Cleve (2015). As was noted previously, a positive thread from this research into SQL development anti-patterns is that the most common items are ubiquitous across both direct SQL code as well as ORMs, such that intermediate to advanced database courses would serve students very well in the workforce to be able to combat these issues no matter the software stack they work with.

Delivering database courses with a suitable number of assignments to develop student skills in working with relational databases is challenging, especially with rising class sizes, given the time constraint in providing student feedback. In courses focused on databases the use of autograders is minimal. When an autograder is used the content of the course is limited to the capabilities of the autograder. When an autograder is not used, the content of the course is reduced to adapt to the expanding class sizes.

1.1 Related Work

Work has begun to develop more robust database autograders, however collaborative progress has not been made (Stanger, 2018; Wagner, 2020). Advances in code analysis techniques for SQL have been significant. Future opportunities

for code analysis in the context of a learning environment focused on databases include incorporation into autograder improvements and introduction as an in-cycle development tool for students (Eessaar, 2020; Yang et al., 2018). Both opportunities are pointed towards improving student short-term and long-term outcomes.

Tools for manipulating SQL queries or schemas are under active development but often through isolated research teams. Whether the primary goal of the work is to perform analysis for code smells during the development cycle ((Eessaar, 2020); (Yang et al., 2018); (Emani et al., 2016); (Lyu et al., 2021)) or to assist in an educational environment ((Wagner, 2020); (Stanger, 2018); (Miao et al., 2020)), the act of providing feedback on the SQL code in an automated manner is a unified concept. Introducing this concept more formally in academia both in student workflows as well as a grading tool can impact its prevalence as a broadly used professional development practice. A broad use of code analysis for SQL antipatterns at an industry level would be ubiquitously beneficial but is not covered by SQLGrader at this time.

In industry, many of the developments in the code analysis space or development tooling space are done collaboratively through open source projects. In these instances, individuals across organizational boundaries are able to build capabilities greater than the sum of the individual expected output. Tools for specialized uses are especially popular open development patterns due to its impact on the functionality and adoption success (Arulraj, 2021; tSQLt, 2021).

2 SQLGRADER FUNCTIONALITY

The implemented system functions as a standalone course assignment management tool or as a supplement to existing course management technology (Canvas, Moodle, and others) with targeted grading services. The SQLGrader grading functionality establishes relational database environments for individual assignment questions (*items*) based on SQL data definition language scripts defined for the cohesive assignment and the individual item. Multiple specific relational database types are currently supported by SQLGrader, including Postgres, MS SQL, and MySQL. Database support and grading algorithms are exposed in customizable components to enable SQLGrader for further use cases.

Items assessing student submissions for the creation of schema items are directly

compared against the provided environment definition. Items assessing SQL query submissions are evaluated n times, where n is one or greater and populated from either provided datasets or datasets generated by SQLGrader upon request. While custom datasets provided by an instructor craft the grading results, the generated datasets show promise in detecting common query mistakes and open the door for assessing additional attributes of a student's submission such as query performance on large datasets.

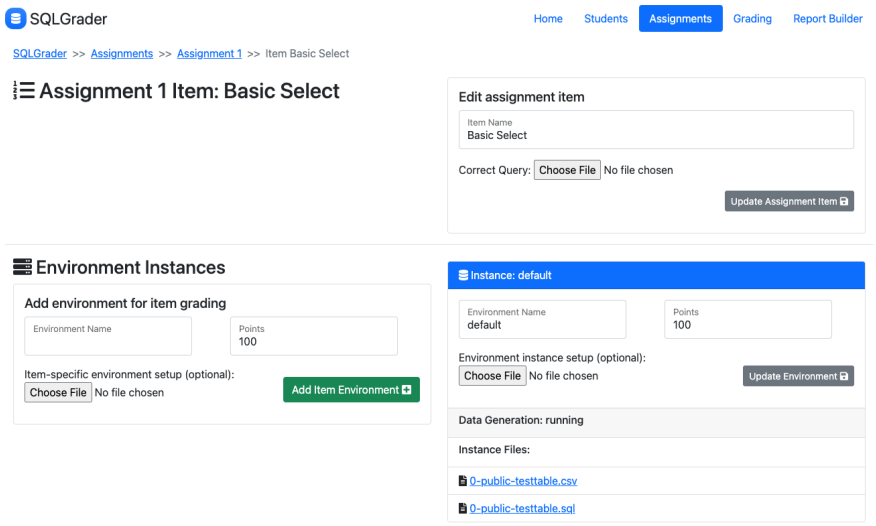

Figure 1—SQLGrader web interface

Management of SQLGrader is provided foremost through a web-based user interface while secondary HTTP and CLI APIs are exposed. Only SQL scripts are required to create one or more database environments, eliminating the requirement to administer databases in the course of grading.

2.1 Architecture

SQLGrader is self-contained in a docker-in-docker architecture, where a single container is managed by the host container runtime and starts its own container runtime. The inner container runtime orchestrates the web-based user interface and all database types requested by SQLGrader over the course of operation, as seen in figure 2. Host software requirements are minimal, where the only software requirement is support of a container runtime where the prebuilt SQLGrader image is retrieved and started. The evolving CPU architecture landscape is considered and images are available for both AMD64 and ARM64 architectures, where the primary selected relational database types provide supported

images for both architectures.

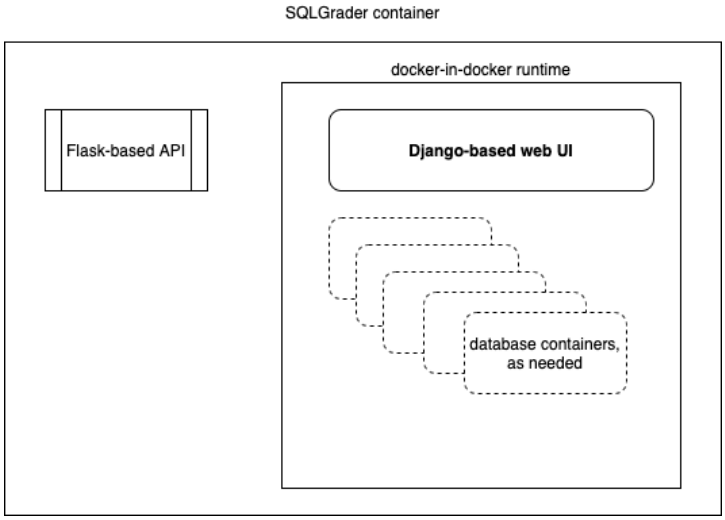

Figure 2—Architecture diagram

User-facing infrastructure is minimized in this architecture while flexibility is maintained to dynamically scale components. While the SQLGrader application can be run from a local workstation, the ubiquity of the containerd runtime presents opportunities to independently host an instance from a cloud or local datacenter.

A prime benefit of the flexibility of the docker-in-docker architecture is support for assessing multiple database types without fundamental changes. Because the database engines are setup and removed on-demand, a singular SQLGrader instance can assess assignments for Postgres, MS SQL, and MySQL synchronously.

2.2 Schema Grading

While ANSI standardization of databases does not enforce strict syntax interchangability, the use of *INFORMATION SCHEMA* tables by ANSI-compliant database systems provides a suitably uniform surface area to collect information about the schema-based structures in a database (PostgresSQL, 2021; Microsoft, 2021; Oracle, 2021). Queries on *INFORMATION SCHEMA* are used by SQLGrader to describe the submitted schema model in memory for comparison with the assignment item's correct schema.

Syntactic differences between database types inform a variation on the same query for each engine. For example, the syntax for data type casting in Postgres

is `column::text` while in MS SQL it is `try_cast(column as varchar(5)`. While the query input varies, the columns output is consistent across such that the same comparison and grading algorithm can be used for any ANSI-compliant database type.

2.3 Query Grading

Determining query equivalence can be accomplished through algorithmic means with success, as shown in Chandra et al. (2019). SQLGrader does not incorporate those approaches and instead establishes query equivalence through query results comparison over a number of datasets. The datasets can be provided by the user or generated by SQLGrader, with a combination of both options as the most efficient approach to

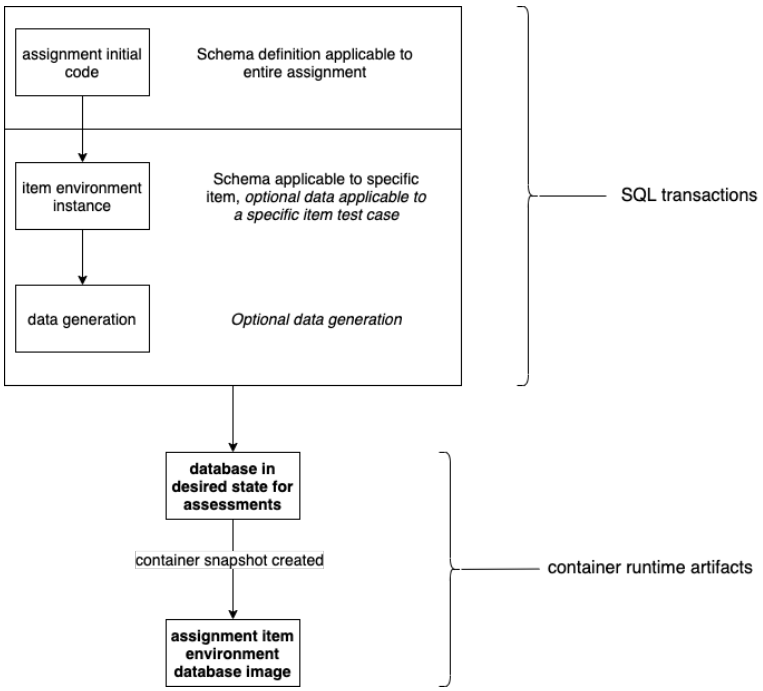

Figure 3—Assignment item assessment environment setup

Hardware efficiency is maximized by dynamically creating container images for the database environment specified for each assignment item, reducing the database engine executions for establishing the baseline environment to a single time regardless of the number of student submissions. The process is demonstrated in figure 3. Large and realistic datasets are readily handled through the container image capture process, as dataset duplication is file IO during container

creation and removal without the overhead of database engine operations.

3 DISCUSSION

SQLGrader is the only public tool for automating grading of both query and schema objects with generated datasets and support for multiple database types. Selected comparative tools are summarized in table 1. The foundational implementation provides distinctive functionality beyond currently available solutions and presents a pathway to additional capabilities through extensible design and a welcoming open source presence. Beyond capabilities as a autograder, SQLGrader presents a unique approach to managing connectivity to multiple databases of different types with managed connections and environment initialization.

Table 1—Database assessment tool summary

Project	**Grade Queries**	**Grade Schema Objects**	**DB types**	**Code avail.**	**Further adoption**
SQLFE (Wagner, 2020)	Against single instructor-provided dataset		1	Yes	None
XData (Chandra et al., 2019)	Generates datasets to test based on abstract syntax tree		1	Yes	None
LearnSQL (Abelló et al., 2016)	Requires instructor interaction	Extensive support, non-automated	1	No	None
CS121 Tool (Gong, 2015)		Tables and columns	1	Yes	None
SQL Schema Assessment (Stanger, 2018)		Tables, columns, primary and foreign keys	1	No	None
SQLGrader (Skwiers-Koballa, 2021)	Against one or more instructor or randomly generated datasets	Tables, columns, and foreign keys	3	Yes	To be determined

3.1 Extensibility

When designing SQLGrader, architectural decisions were deliberately made to promote ease of code customization and extensibility. By design, it is not expected

that a user would need to understand a significant portion of the SQLGrader code base to modify a specific behavior to their need.

For example, the comparison of schema objects is done on a case-insensitive basis by default. This functionality is surfaced in an API component for schema grading interactions. While customizations can be implemented in a live SQLGrader instance, it is recommended to directly build a container image to preserve changes.

SQLGrader system data is extensibly accessed in multiple interfaces. The default Django admin interface is secured and accessible by the environment administrator. Certain objects expected to be present in large volume, such as students, can be directly imported in the SQLGrader web interface from a comma separated value file. All system data can be exported through customizable reports in a complete reporting portal in SQLGrader.

3.2 Open Source

The distinction between extensibility and an open code base is blurry but the significance of an open source project is not consumed by extensibility. Fundamentally, code being publicly available and permissively licensed constitutes open source. The health of open source projects hinges on much more than a code base and requires documentation for functionality and development.

A network of dependencies forms between open source projects, encouraging developers to engage at the level where they are comfortable and in a subject area where they have interest. SQLGrader is build in an open source language (Python) on open source frameworks (Django, Flask) and deploys on an open source runtime (containerd). Additional smaller open source components are incorporated to enhance specific areas of functionality of SQLGrader, including the data generation feature.

Opening the SQLGrader project to additional developers is accomplished through contributing guidelines, developer documentation, and development environment images. Departure from common collaborative antipatterns seen in educational technology is intentional to improve the cumulative outcome from this project, consistent with the model described by Khatri et al. (2016).

The results from the open source structures around SQLGrader is a developer with minimal knowledge of any related technologies can participate in a mini-

Table 2—Collaborative focus on open source in SQLGrader compared to common antipatterns of open code observed.

Collaboration Barrier	SQLGrader	Antipatterns
Code access	pull git repository	download code in zip
Getting started with development	attach VS Code IDE to preconfigured environment	manually install prerequisites
Development documentation	visible code history, open documentation	static documentation website
Communication	unified code and communication platform	email, none specified

mal number of actions and test the entire SQLGrader application locally. Users of SQLGrader benefit from the short cycle for introducing improvements from open source contributors back into the project, which is amplified by its use of prevalent technologies and availability on a popular site (GitHub).

3.3 Limitations

The grading capabilities of SQLGrader can be further expanded to cover table indexes, query performance, stored procedures, and other database components. Currently the implementation is foundational and leaves opportunity for expansion.

Deployment of SQLGrader to scaling instances in Kubernetes clusters is not described and it should be considered a singular node application for use by a single instructor or instructional team at a time.

3.4 Future Work

The focus of future work is on improving the capabilities of the query and schema assessment and adding surfaces for directly interacting with assessment and managed development database containers. Inspired by the robust query correctness capabilities of Chandra et al. (2019), the query assessment capability of SQLGrader can be expanded to shape the generated datasets based on static values found in the correct query script. To expand the impact of SQLGrader, the query assessment capabilities can be expanded to also report on query performance based on query plan operators and data access statistics.

Surfaces for interacting with the assessment database container instances would

allow for further customization by instructors to augment the implemented assessment capabilities with changes specific to their curriculum. In alignment with the difficulties described by Randolph (2003) for students and instructors alike in setting up database environments, SQLGrader can expose managed database container instances for course development purposes.

While SQLGrader already demonstrates unique capabilities in the database autograder landscape, these proposed improvements tightly align with difficulties encountered in database education. In addition to the mentioned intentions, future work will be shaped by user feedback and requests submitted at the code repository. Open source mechanisms for managing future work are discussed in appendix 6.2.

4 CONCLUSION

The SQLGrader tool provides an interface for automating grading for one or more database courses and supports best practices of database education, including realistic dataset sizes and multiple ANSI-standard database types. With a combination of both query and schema object grading capabilities, SQLGrader is capable of providing autograder capabilities for the foundational database concepts common in database education (Taipalus and Seppänen, 2020) and can combat the reduction in student outcomes by creating efficiencies in the instructor grading workload. As described by Wolff (2001), students benefit from exposure to multiple database types to deepen their understanding of SQL syntax and the differing strengths of the alternative implementations.

In following published best practices for the long-term success of innovation in educational (Taylor et al., 2019) and with open source software (Eghbal, 2020), SQLGrader will continue to gain distinctive functionality proven advantageous to anyone instructing a database course. The most significant characteristic departure between SQLGrader and existing database assessment tools is the focus on reaching multiple institutions by supporting adoption and encouraging those who might otherwise start anew to collaborate on the tool.

5 REFERENCES

[1] Abelló, Alberto, Burgués, Xavier, Casany, M.J., Martín, Carme, Quer, Carme, Rodríguez, M., Romero, Oscar, and Urpí, Toni (June 2016). "A Software Tool

for E-Assessment of Relational Database Skills". In: *International Journal of Engineering Education*.

[2] Arulraj, Joy (Nov. 2021). *jarulraj/sqlcheck*. URL: https://github.com/jarulraj/sqlcheck.

[3] Atwood, Sara A. and Singh, Arjun (June 2018). "Improved Pedagogy Enabled by Assessment Using Gradescope". In: URL: https://peer.asee.org/improved-pedagogy-enabled-by-assessment-using-gradescope.

[4] Chandra, Bikash, Banerjee, Ananyo, Hazra, Udbhas, Joseph, Mathew, and Sudarshan, S. (Apr. 2019). "Automated Grading of SQL Queries". In: *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pp. 1630–1633. DOI: 10.1109/ICDE.2019.00159.

[5] Cheang, Brenda, Kurnia, Andy, Lim, Andrew, and Oon, Wee-Chong (Sept. 2003). "On automated grading of programming assignments in an academic institution". In: *Computers and Education* 41.2, pp. 121–131. ISSN: 0360-1315. DOI: 10.1016/S0360-1315(03)00030-7.

[6] Dietrich, Suzanne W. and Chaudhari, Mahesh (Mar. 2011). "LINQ ROX! integrating LINQ into the database curriculum". In: *Proceedings of the 42nd ACM technical symposium on Computer science education*. SIGCSE '11. Association for Computing Machinery, pp. 293–298. ISBN: 978-1-4503-0500-6. DOI: 10.1145/1953163.1953251. URL: https://doi.org/10.1145/1953163.1953251.

[7] Eessaar, Erki (2020). "Automating Detection of Occurrences of PostgreSQL Database Design Problems". In: *Databases and Information Systems*. Ed. by Tarmo Robal, Hele-Mai Haav, Jaan Penjam, and Raimundas Matulevičius. Communications in Computer and Information Science. Springer International Publishing, pp. 176–189. ISBN: 978-3-030-57672-1. DOI: 10.1007/978-3-030-57672-1_14.

[8] Eghbal, Nadia (Aug. 2020). *Working in Public: The Making and Maintenance of Open Source Software*. Stripe Press.

[9] Emani, K. Venkatesh, Ramachandra, Karthik, Bhattacharya, Subhro, and Sudarshan, S. (June 2016). "Extracting Equivalent SQL from Imperative Code in Database Applications". In: *Proceedings of the 2016 International Conference on Management of Data*. SIGMOD '16. Association for Computing Machinery, pp. 1781–1796. ISBN: 978-1-4503-3531-7. DOI: 10.1145/2882903.2882926. URL: https://doi.org/10.1145/2882903.2882926.

[10] Fini, Antonio (Nov. 2009). "The Technological Dimension of a Massive Open Online Course: The Case of the CCK08 Course Tools". In: *International Review of Research in Open and Distance Learning* 10. DOI: 10.19173/irrodl.v10i5.643.

[11] Gong, Angela (Oct. 2015). *CS 121 Automation Tool*. URL: https://github.com/anjoola/cs12x-automate.

[12] Helmick, Michael T. (June 2007). "Interface-based programming assignments and automatic grading of java programs". In: *Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education*. ITiCSE '07. Association for Computing Machinery, pp. 63–67. ISBN: 978-1-59593-610-3. DOI: 10.1145/1268784.1268805. URL: https://doi.org/10.1145/1268784.1268805.

[13] Khatri, Raina, Henderson, Charles, Cole, Renée, Froyd, Jeffrey E., Friedrichsen, Debra, and Stanford, Courtney (Feb. 2016). "Designing for sustained adoption: A model of developing educational innovations for successful propagation". In: *Physical Review Physics Education Research* 12.1, p. 010112. DOI: 10.1103/PhysRevPhysEducRes.12.010112.

[14] Lyu, Yingjun, Volokh, Sasha, Halfond, William G. J., and Tripp, Omer (July 2021). "SAND: a static analysis approach for detecting SQL antipatterns". In: *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*. ISSTA 2021. Association for Computing Machinery, pp. 270–282. ISBN: 978-1-4503-8459-9. DOI: 10.1145/3460319.3464818. URL: https://doi.org/10.1145/3460319.3464818.

[15] Miao, Zhengjie, Chen, Tiangang, Bendeck, lexander, Day, Kevin, Roy, Sudeepa, and Yang, Jun (Aug. 2020). "I-Rex: an interactive relational query explainer for SQL". In: *Proceedings of the VLDB Endowment* 13.12, pp. 2997–3000. ISSN: 2150-8097. DOI: 10.14778/3415478.3415528.

[16] Microsoft (Jan. 2021). *System Information Schema Views (Transact-SQL) - SQL Server*. URL: https://docs.microsoft.com/en-us/sql/relational-databases/system-information-schema-views/system-information-schema-views-transact-sql.

[17] Muse, Biruk Asmare, Rahman, Mohammad Masudur, Nagy, Csaba, Cleve, Anthony, Khomh, Foutse, and Antoniol, Giuliano (June 2020). "On the Prevalence, Impact, and Evolution of SQL Code Smells in Data-Intensive Systems". In: *Proceedings of the 17th International Conference on Mining Software Repositories*. MSR '20. Association for Computing Machinery, pp. 327–

338. ISBN: 978-1-4503-7517-7. DOI: 10.1145/3379597.3387467. URL: https://doi.org/10.1145/3379597.3387467.

[18] Nagy, Csaba and Cleve, Anthony (Sept. 2015). "Mining Stack Overflow for discovering error patterns in SQL queries". In: *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, pp. 516–520. ISBN: 978-1-4673-7532-0. DOI: 10.1109/ICSM.2015.7332505. URL: http://ieeexplore.ieee.org/document/7332505/.

[19] Oracle (2021). *MySQL 8.0 Reference Manual: 26 INFORMATION SCHEMA Tables*. URL: https://dev.mysql.com/doc/refman/8.0/en/information-schema.html.

[20] PostgresSQL (Nov. 2021). *Chapter 37. The Information Schema*. URL: https://www.postgresql.org/docs/14/information-schema.html.

[21] Randolph, Gary B. (Oct. 2003). "The forest and the trees: using oracle and SQL server together to teach ANSI-standard SQL". In: *Proceedings of the 4th conference on Information technology curriculum*. CITC4 '03. Association for Computing Machinery, pp. 234–236. ISBN: 978-1-58113-770-5. DOI: 10.1145/947121.947174. URL: https://doi.org/10.1145/947121.947174.

[22] Skwiers-Koballa, Drew (Dec. 2021). *SQLGrader*. SQLgrader. URL: https://github.com/robertdroptablestudents/sqlgrader.

[23] Stanger, Nigel (2018). "Semi-Automated Assessment of SQL Schemas via Database Unit Testing". In: p. 10.

[24] Taipalus, Toni and Seppänen, Ville (Aug. 2020). "SQL Education: A Systematic Mapping Study and Future Research Agenda". In: *ACM Transactions on Computing Education* 20.3, 20:1–20:33. DOI: 10.1145/3398377.

[25] Taylor, Cynthia, Spacco, Jaime, Bunde, David, Butler, Zack, Bort, Heather, Hovey, Christopher, Maiorana, Francesco, and Zeume, Thomas (Feb. 2019). "Propagating Educational Innovations". In: pp. 167–168. DOI: 10.1145/3287324.3287526.

[26] tSQLt (Dec. 2021). tSQLt.org. URL: https://github.com/tSQLt-org/tSQLt.

[27] Villa, Adam H. (Jan. 2016). "Big data: motivating the development of an advanced database systems course". In: *Journal of Computing Sciences in Colleges* 31.3, pp. 119–128. ISSN: 1937-4771.

[28] Wagner, Paul J. (Feb. 2020). "The SQL File Evaluation (SQLFE) Tool: A Flexible and Extendible System for Evaluation of SQL Queries". In: *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*.

SIGCSE '20. Association for Computing Machinery, p. 1334. ISBN: 978-1-4503-6793-6. DOI: 10.1145/3328778.3372599. URL: https://doi.org/10.1145/3328778.3372599.

[29] Wolff, David A. (Dec. 2001). "MySQL, PostgreSQL, and PHP: open source technologies for a database management course". In: *Journal of Computing Sciences in Colleges* 17.2, pp. 91–92. ISSN: 1937-4771.

[30] Yang, Junwen, Yan, Cong, Subramaniam, Pranav, Lu, Shan, and Cheung, Alvin (Oct. 2018). "PowerStation: automatically detecting and fixing inefficiencies of database-backed web applications in IDE". In: *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, pp. 884–887. ISBN: 978-1-4503-5573-5. DOI: 10.1145/3236024.3264589. URL: https://dl.acm.org/doi/10.1145/3236024.3264589.

6 APPENDICES

6.1 SQLGrader associated materials

1. https://robertdroptablestudents.github.io/: project documentation
2. https://github.com/robertdroptablestudents/sqlgrader: code

6.2 SQLGrader Development Plan

Future development is planned based on open issues (features, bugs) at the SQLGrader code repository. A snapshot of the current project board is in figure 4. Issues are categorized by priority and type with high-visibility labels.

Contributions to the project from additional developers are welcome. Code is added via "pull request" to incorporate line-by-line changes to existing or new files.

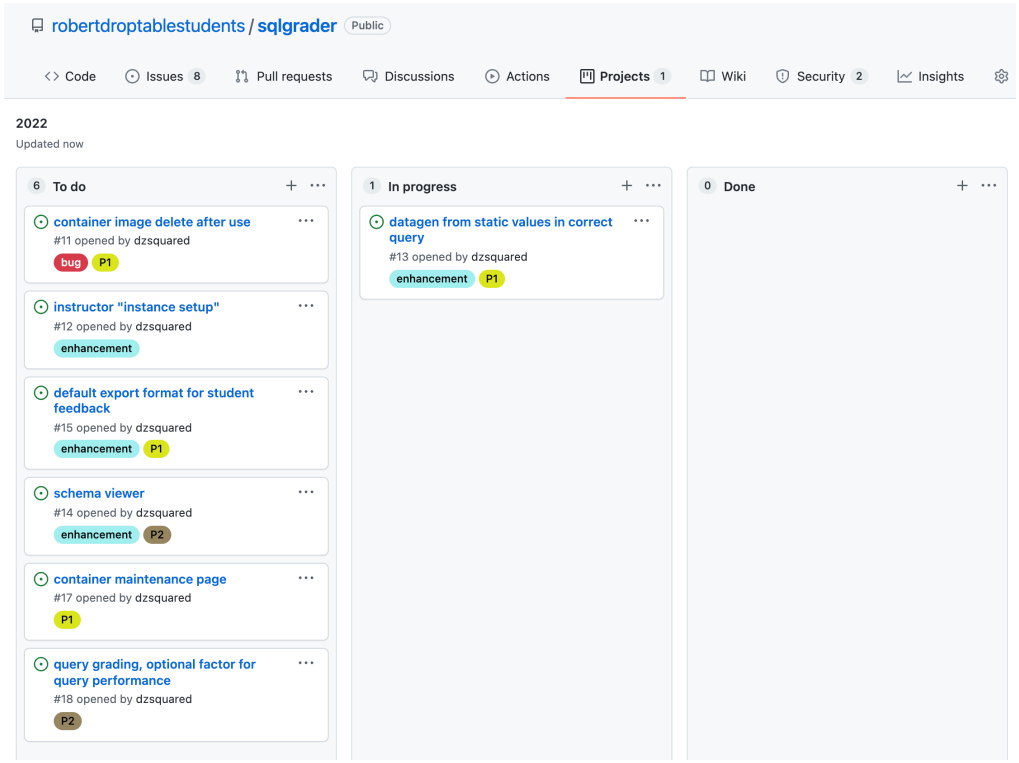

Figure 4—SQLGrader project planning board, December 2021