ASEE AMERICAN SOCIETY FOR ENGINEERING EDUCATION

Paper ID #35166

Work-In-Progress: TriQL: A tool for learning relational, graph and document-oriented database programming.

Dr. Abdussalam Alawini, University of Illinois at Urbana-Champaign

I am a teaching assistant professor in the Department of Computer Science at The University of Illinois at Urbana-Champaign. My research interests are broadly in the field of databases, with a focus on data management. I am particularly interested in applying machine learning methods to various problems of modern data management systems. I am also interested in CS education research.

Mr. Peilin Rao, UIUC

I am a student in ECE department of UIUC

Mr. Leyao Zhou, University of Illinois at Urbana-Champaign

Undergraduate student majoring in Computer Science at Unversity of Illinois at Urbana-Champaign

Miss Lujia Kang

A senior computer science student studying at the University of Illinois at Urbana-Champaign.

Mr. PING-CHE HO, PureStorage

A graduate in the year of 2020 with 2 masters degree in Computer Science and Information Management from the University of Illinois - Urbana-Champaign, with a life long goal to pursue knowledge in the field of computer science in hopes of making a significant contribution to society through science and technology.

WIP: TriQL: A Tool for Learning Relational, Graph and Document-Oriented Database Programming

Abdussalam Alawini* alawini@illinois.edu
Ping-Che Ho* pingche2@illinois.edu
Lujia Kang* lujiak2@illinois.edu
Peilin Rao* peilinr2@illinois.edu
Leyao Zhou* leyaoz2@illinois.edu

Abstract
Databases are pervasive and vital to the use and security of sensitive data such as in medical, financial, scientific, and consumer contexts[1]. However, with the abundance of database models (types), such as the relational, graph, and document-oriented databases, learners often find it challenging to decide what database model they should learn and the trade-offs between different data models. In this paper, we introduce TriQL, a system for helping novices learn the structures (schema) and query languages of three major database systems, including MySQL (a relational, SQL-Structured Query Language, database), Neo4J (a graph database), and MongoDB (a document/collection-oriented database). TriQL offers learners a graphical user interface to design and execute a query against a generic database schema without requiring them to have any database programming experience. TriQL follows an interactive approach to learning new database models, supporting a dynamic and agile learning environment that can be easily integrated into database labs and homework assignments.

1 Introduction

With more organizations relying on data to make crucial business decisions, database systems have become essential in managing financial, medical, and scientific data. Consequently, managing databases has become a necessary skill for programmers, data analysts, and data scientists to accelerate scientific inquiry and business decision-making. However, with the abundance of database models (types), such as a relational, graph, and document-oriented databases, beginner learners often find it challenging to decide what database model they should learn. Experienced developers also struggle to understand new database models as different models have different data structures and query languages.

In response to these issues, several educational institutions have adopted a curriculum that includes relational and NoSQL (Not only SQL) databases[2,3,4,5]. However, most database courses teach each data model separately using independent labs and homework assignments without providing students with insights into the trad-offs between different data models[6,7,8]. Such

*University of Illinois at Urbana-Champaign

limitation hinders students' ability to generalize their knowledge to learning new data models.

In this paper, we introduce TriQL: Tribus linguis query, Latin for three query languages. TriQL is a system for helping novices and learners with limited database experience learn the structures (schema) and query languages of three major database systems, including MySQL (a relational, SQL-Structured Query Language, database), Neo4J (a graph database), and MongoDB (a document/collection-oriented database). Our system also helps learners explore structural and query language variations among different data models. Our system uses advanced database techniques, such as data integration and logical programming, to capture the core data operations common between the relational, graph and document-oriented data models, including selection (filtering data), projection (redefining the output schema), and grouping and aggregation. Abstracting the data operations and the schema design can help learners understand the trade-offs among different data models.

We also discuss the integration of TriQL into database education. In particular, we discuss how TriQL can be used as part of two labs: one that focuses on the database schema and query languages, and the other lab shade light on the tradeoffs between relational, graph, and document-oriented models. By developing TriQL, we aim to influence other engineering education disciplines to develop tools for helping students explore the trade-offs between different programming, design, and development paradigms. The rest of this paper is organized as follows. In Section 2, we provide an overview of the relational (SQL), graph (Neo4J), and document (MongoDB) databases and their query languages. Then, we present TriQL's system architecture in Section 3. In Section 4, we discuss how can TriQL be integrated with a database course. We provide a literature review in Section 5. Section 6 discusses our future plans for TriQL and concludes the paper.

2 Background

In this section, we first introduce DataLog, an intermediate logical database language we use to capture the generic user query to express it in other database languages. Then, we provide a quick tutorial on the database query languages TriQL supports, including SQL, MongoDB, and Cypher.

2.1 Introduction to Datalog

Datalog is a powerful logical and declarative programming language[9]. Due to its simplicity and query expressiveness, DataLog became the standard choice for intermediate query language generated based on the user's input[10,11,12]. In Datalog, each formula is a function-free Horn clause, and every variable in the head of a clause must appear in the body of the clause. Such simple yet powerful formalization can maintain the query logic and express it in several database models, enabling our system to be flexible, easily plugable, and extendable with different database technologies.

We briefly demonstrate the syntax of Datalog queries through two examples. We use the following university database as a running example. This database has three relations (tables):

Course(course_id, Name, Instructor); **Has** (course_id, student_id) **Student** (student_id, FirstName, LastName, Age, Year, Major)

The following query (Q_1) will find the major of a student named "James Smith":

```
output(E) :- Student(A, B, C, D, E, F), B = "James", C = "Smith"
```

`Student(A, B, C, D, E, F)` is the definition of the `Student` relation with each variable corresponding to each field in the `student` relation. For example: `A` represents `student_id`, `B` represents `FirstName`, `C` represents `LastName`. This query has two conditions: `B = "James"` and `C = "Smith"` and projects (i.e., outputs) the result of `E`, which is the `Major` attribute.
A more complex query (Q_2) that finds, for each course, the number of students majoring in ECE:

```
output(B, V0) :- V0 = Course(_, B, _), COUNT(D):
{Course(A, B, C), Student(D, E, F, G, H, I), Has(J, K),
A = J, D = K, I = "ECE"}
```

This is an example of the aggregation operation. In order to find the number of ECE students in every class, we first select tuples with the ECE major. Then, we JOIN `Course` and `Has` on `course_id`, and `Student` and `Has` on `student_id`. Finally, we GROUP BY `course_name` in `Course` table. `V0` is the result of the COUNT operation, which counts the `student_id`. The output is `course_name` and `V0`.

2.2 Introduction to Relational Databases and SQL

The primary data structure in the relational model is called relation (table), each relation consists of a set of tuples (rows or records), and each tuple consists of a set of attributes (columns). Structured Query Language (SQL) is the de facto standard for querying relational databases [13]. Indeed, by some metrics, it is the most in demand programming language [13,14]. We introduce SQL queries using MySQL, an open-source implementation of the relational database that is widely used in industry and academia.

Student

student_id	FirstName	LastName	Age	Year	Major
1	"James"	"Smith"	21	"Junior"	"CS"
2	"Petter"	"Jackson"	22	"Senior"	"ECE"
3	"Mick"	"Young"	20	"Sophomore"	"CS"

Course

course_id	Name	Instructor
1	"Intro to Database Systems"	"David Young"
2	"Linear Algebra"	"Tim Miller"

HAS

course_id	student_id
1	1
1	2
2	1
2	3

Figure 1: The relational version of the university database presented in Section 2.1

Figure 1, shows the relational database schema of our running example presented in Section 2.1. we show the SQL query corresponding to Q_1 and Q_2 in Figure 2. Figure 2(a) shows the SQL version of Q_1. To find the major of a student, we `SELECT` (output) the `Major` column `FROM` the `Student` table *WHERE* the `name` is 'James Smith'. Figure 2(b) shows the SQL query of Q_2, which is slightly more complex SQL query than Q_1 as it involves aggregation and joining of

several tables. We first join `Student`, with `Has` and `Courses` to find students and courses they are taking. Then, we use the `WHERE` clause to select only ECE students. To count the number of students in each course, we `GROUP BY` course name use the `COUNT` function to count the number of ECE students in each course.

```
SELECT s.Major
FROM Student AS s
WHERE s.FirstName = "James"
      AND s.LastName = "Smith"
```

(a) Q_1: Find the major of a student named "James Smith"

```
SELECT c.name, COUNT(s.student_id) AS count_student
FROM Course AS c
    JOIN Student AS s
    JOIN HAS ON (c.course_id = HAS.course_id
        AND s.student_id = HAS.student_id)
WHERE s.Major = "ECE"
GROUP BY c.name
```

(b) Q_2: for each course, find the number of students majoring in ECE

Figure 2: The SQL queries corresponding to the DataLog queries presented in Section 2.1

2.3 Introduction to Neo4J and Cypher

Cypher is the query language used to query Neo4j databases. It has a similar syntax to SQL, with declarative pattern-matching features added for querying graph relationships. As one of the most popular graph databases, Neo4j stands out among the current graph models for its performance, simplicity, and powerful query language[15,16]. We briefly demonstrate the syntax of Neo4j structure and queries. Figure 3 shows the Neo4J database equivalent to the database from our running example. It has two types of nodes: `Course` and `Student`, and one relationship `HAS`. Notice that the `HAS` relationship was represented as a relation (table) in the relational database.

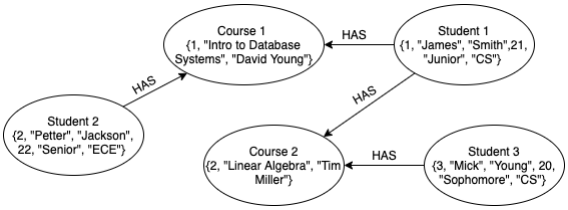


Figure 3: The Neo4J (graph) version of the university database presented in Section 2.1

Figure 4 shows the Cypher query for Q_1 and Q_2. The Cypher version of Q_1 (Figure 4(a)) is similar to Q_1's SQL version. We first find the `Student` nodes and use the `WHERE` clause to select students with the name 'James Smith'. Then, we `RETURN` (output) the `Major` property of the student. Figure 4(b) shows the Cypher query equivalent to Q_2 and demonstrates Neo4J's power in querying interconnected data. The graph pattern matching clause shown in the `MATCH` finds all `Students` measuring ECE and taking a class (represented by the relationship `Has`). The `RETURN` clause group the result by course name and COUNT `student_ids` per course name. Notice that Cypher does not use explicit GROUP BY cluase. If the `RETURN` clause contains any aggregate function (such as `COUNT()`), it will group by all listed attributes in the `RETURN`.

```
MATCH (s:Student)
WHERE s.FirstName = "James"
    AND s.LastName ="Smith"
RETURN s.Major
```

(a) Q_1: Find the major of a student named "James Smith"

```
MATCH (s:Student {Major:"ECE"})-[:HAS]->(c:Course)
RETURN c.name, COUNT(s.student_id) AS count_student
```

(b) Q_2: for each course, find the number of students majoring in ECE

Figure 4: The Cypher queries corresponding to the DataLog queries presented in Section 2.1

2.4 Introduction to MongoDB

MongoDB[17] is a document-oriented NoSQL database that stores JSON-like data in documents with flexible schema, removing the need for pre-defining the structure before inserting the data into documents. A MongoDB consists of a set of `Collections`, which consists of a set of `Documents`, which consists of a set of key-value pairs. We show examples of MongoDB documents and code snippets that exhibit the basic syntax of MongoDB. Note that the `_id` attribute is an indexed attribute that must be included in every object. Figure 5 shows a MongoDB database equivalent to our running example's database.

```
Course:
{
    "course_id": 1,
    "Name": "Intro to Database System",
    "Instructor": "David Young"
}
```

```
Student:
{
   "student_id": 2,
    "FirstName": "Peter",
    "LastName": "Jackson",
    "Age": 22,
    "Year": "Senior",
     "Major": "ECE"
}
```

```
Has:
{
    "course_id": 1,
    "student_id": 2
}
```

Figure 5: The MongoDB version of the university database presented in Section 2.1

MongoDB databases can be queried from many different programming languages, but the most straightforward interface is through MongoDB's JavaScript shell. The MongoDB's version of Q_1 is shown below.

```
db.Student.find(
    { FirstName: "James", LastName: "Smith" },
    { _id: 0, Major: 1})
```

The find operation takes two arguments (selection and projection) and returns documents in a collection or view, and returns a cursor to the selected documents. In the query above (Q_1), the second line represents the selection part of the query. The selection condition finds a document with `FirstName` 'James' and `LastName` 'Smith'. The third line (`{_id:0, Major:1}`) returns `Major` and hides (`_id:0`) the _id property.

Figure 6 shows the MongoDB version of Q_2. This query uses the `aggregate` pipeline on the `Course` collection. The `$lookup` operation finds documents in the `Has` collection that match on the `course_id`. Then, the `$unwind` creates a document for each element in the newly constructed array `has`. The same two operations will be repeated again to connect `has` to `Student` based on the `student_id` property. Next, we use the `$MATCH` operator to select students majoring 'ECE'. Then, the `$group` operator groups documents by the course name; and finally the `$project` operator outputs the course name and the size of the student array.

```
db.Course.aggregate([
    {$lookup:
        {   from: "HAS",
            local_field: "course_id",
            foreign_field: "course_id",
            as:"has" }
    },
    {$unwind: "$has"},
    {$lookup:
        {   from: "Student",
            local_field: "has.student_id",
            foreign_field: "student_id",
            as:"student" }
    },
    {$unwind: "$student"},
    {$match:
        { Student.Major: "ECE" }
    },
    {$group:
        {   _id: {name: "name"},
            students: {$push: "$student.student_id"} }
    },
    {$project: {_id: 1, count_student: {$size: "$students"}}}
])
```

Figure 6: MongoDB version of Q_2: for each course, find the number of students majoring in ECE

3 System Overview

Figure 7 shows the main components of TriQL. Once a user submits a generic query (represented as a JSON, JavaScript Object Notation, structure) via the Query Builder Interface (QBI), the Intermediate Query Generator (IQG) converts the user query into DataLog. Then, the Schema and Query Translator (SQT) 1) transforms the generic database (JSON) schema shown to the user into MySQL, Neo4J, and MongoDB, and 2) concurrently converts the DataLog query into SQL, Cypher, and MongoDB. Finally, TriQL executes each generated query on its corresponding database engine and shows native result to the user on the Query Result Interface. The user can then examine the three generated queries along with their outputs. They can also modify their query using TriQL's QBI and resubmit it again to see the effects of their changes. Such an interactive approach is beneficial for users to learn by examples in a dynamic and agile fashion.

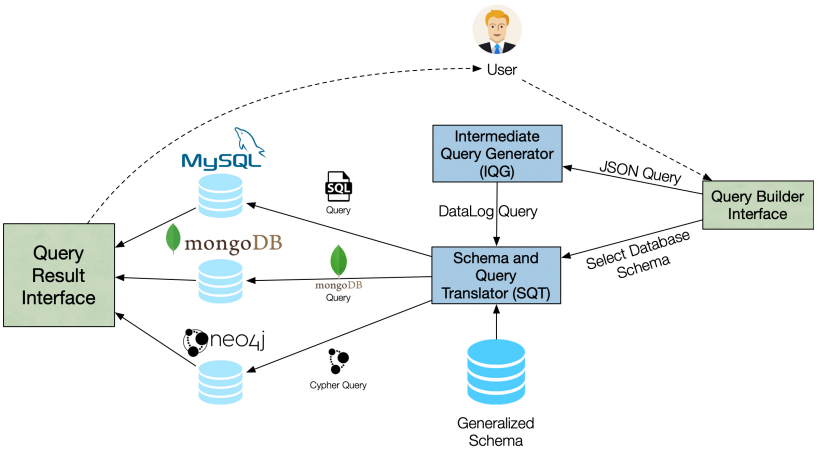


Figure 7: TriQL System Architecture: The Query Builder for building queries using a GUI; the Intermediate Query Generator converts user queries to DataLog; The Schema and Query Translator generates the schema of the three database and coverts the DataLog query into SQL, Cypher and MongoDB

3.1 Generalized Database Schema

The database schema describes data entities and their relationships based on the real-world application's underlying business logic. Database systems vary significantly in how they represent data. For example, the relational database has a fixed schema where users must define the data structure before uploading it into the database. In contrast, other databases, such as MongoDB

and Neo4J, have a more flexible schema where users can upload the data without worrying about its structure. Such variations in database structures make it challenging for novices to learn database programming.

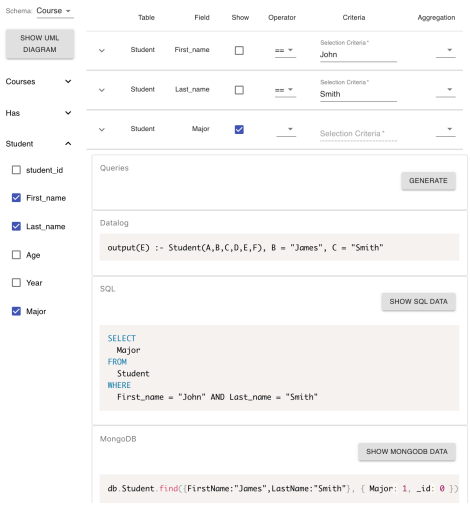
In this paper, we introduce our generalized JSON-based (JavaScript Object Notation) schema (GS), capable of capturing properties of relational (SQL) and NoSQL databases. We choose JSON to represent our generalized schema because of its expressive and straightforward structure. JSON is also widely supported by many programming languages.

The ability to generalize the structure of databases has several key learning outcomes: 1) learners can easily understand the data entities and how they are connected without having to learn complex data definition languages, 2) because we capture various database properties in GS, TriQL can easily transform GS into other database structures, and 3) our GS provides learners with the ability to examine the properties of relational, graph and document databases.

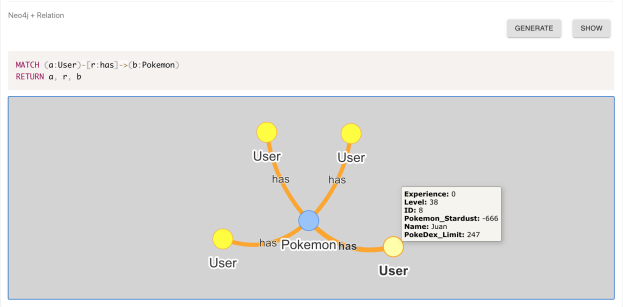
The general schema stores information of the underlying data models using two primary substructures: ENTITY and RELATION. Entities contain attributes that describe real-world objects; relationships capture connections between these entities. Each relationship captures the *relationship cardinality* (i.e., one-to-one, one-to-many, or many-to-many), and the *relationship direction* for directed relationships. Such a simple yet powerful representation of structure allows learners to quickly identify entities and their attributes and relationships between these entities. Additionally, capturing information such as relationship direction and cardinality allows our system to transform this schema into relational, graph, or document databases without human intervention.

3.2 Query Builder and Result Interface

TriQL's Query Builder Interface (QBI) allows users to construct queries using a user-friendly Graphical User Interface (GUI). The GUI allows users to select a database schema from a set of preloaded databases. Users can examine the conceptual design by clicking the "show UML diagram" button, which displays the design as a Unified Modeling Language (UML) diagram. Once the user is comfortable with the database schema, they can use the QBI to query the database. They can select entities and attributes (fields), define selection criteria over attributes, and decide whether to return an attribute with the output. The user must use the "Show" checkbox to choose the queries' attribute or aggregations output. They can also define grouping and aggregation functions. After clicking the "Generate" button, the QBI sends the user query as a JSON structure to TriQL's intermediate (DataLog) query generator. The generated Datalog query is then translated into SQL, Mongo, and Neo4j queries, displayed in the Query Result Interface (see Figure 8). To see each translated query's output, users can click the "show data" buttons to see the query result in its native structure. Figure 8(b) shows an example of TriQL's Query Result Interface showing a Cypher query in its native Neo4J database. Users can interact with the graph and visualize the properties of the nodes and edges.



(a) TriQL's Query Builder and Query Result Interfaces. The QBI allows users to construct the query using a user-friendly GUI and the Query Result Interface shows the query result in its native database.



(b) An example of TriQL's Query Result Interface showing a Cypher query in its native Neo4J database. Users can interact with the graph and visualize the properties of the nodes and edges.

Figure 8: TriQL's Query Builder and Results Interfaces

3.3 Intermediate Query Generator

The Intermediate Query Generator (IQG) translates the JSON query received from the query builder into DataLog. The process starts with converting entities into Datalog relations. For instance, the user input from Figure 2 will output relation **tmp**(Major). Next, the IQG creates a mapping between entities' attributes and the Datalog variable. Using the relations and mappings, the IQG can now generate a Datalog query that matches the user query. Here is the Datalog query for Q_1:

```
output(E) :- Student(A, B, C, D, E, F), B = "James", C = "Smith"
```

The head of the DataLog query (`output(E)`) specifies the attribute(s) that the user would like to output. The query body contains the input relation (`Student`) and the conditions for filtering the tuples (records). The query, the generated intermediate relations, and dictionary are passed to the Schema and Query Translator (SQT), which converts the DataLog to the corresponding SQL, Cypher, and MongoDB queries.

3.4 Schema and Query Translator

The Schema and Query Translator (SQT) has two subsystems: Schema Generator and Query Translators. The schema generator converts the JSON-based Generalized Schema of the selected database and converts it into relational (MySQL), graph (Neo4J), and document (MongoDB) databases. The Query translators subsystem receives the DataLog and translates it into into the equivalent SQL, Cypher, and MongoDB queries.

The Query Translator uses predefined rules to convert a DataLog query into SQL, Cypher and MongoDB queries (See Figure 8). The generated queries can capture the primary data querying

operations, including *selection*, *projection*, *grouping*, and *aggregation*. Once the three queries are generated, each query will be executed in its corresponding database and the data will be returned to the user in its native structure.

4 Integrating TriQL in Database Curriculum

Several database courses have adopted a curriculum that includes relational and NoSQL models. For example, Portland State University offered Neo4J and MongoDB databases in its 'Data Management in the Cloud' course as early as 2014[6]; the University of Pennsylvania's Database and Information Systems course has also offered SQL, and NoSQL databases since 2017[7]; our university's 'Database Systems' course (CS 411), taught by the first author, also covers NoSQL databases since 2018[8]. While these courses teach multiple data models, they do not offer labs that provide students with insights into the tradeoffs between different data models. Such teaching methodology hinders students' ability to generalize the database knowledge to learning new data models. The primary motivation for developing TriQL is to bridge this gap. Thus, in this section, we discuss how we can integrate TriQL labs into a database curriculum, using CS 411 as an example.

CS 411 is an elective course taken primarily by graduate students or undergraduates nearing the end of their degree, with pre-requisites, including introduction to programming and data structures. The course is structured to cover three main units: *data models and query languages* (relational model: SQL, graph model: Neo4j and cypher and document-oriented model: MongoDB), *database design* (conceptual design and normal forms) and *database implementation* (storage and indexing, query optimization, concurrency control). The course spends five lectures on SQL, two on MongoDB and two on Neo4J. As summative assessments, CS 411 offers homework assignments and exams. For formative assessments, it offers group-based lab assignments hosted on PrairieLearn, an online system for problem-driven learning[18]. Five labs dedicated to SQL, two labs to MongoDB, and two to Neo4J. For each lab, we provide students with a general description of the database and a set of questions. Students have a text editor where they write their queries and *save and grade* or only *save* if they prefer to grade them later. Prairielearn provides instant feedback from auto-graders on each submitted query.

We plan to integrate TriQL in two additional lab assignments. TriQL lab 1, which precedes the SQL, MongoDB, and Neo4J labs, introduces students to the generalized schema and the query builder interface. The first part of this experimental lab would help students explore the generalized schema of a real-world application. Examining a generic schema helps students understand the data entities and their connections without worrying about understanding any particular database's structure. The second part of this lab focuses on teaching students the principal data querying operations, including selection, projection, grouping, and aggregation. Using the TriQL QB interface, students can immediately query the database without any prior knowledge of any database programming language.

TriQL lab 2, which will succeed all SQL, MongoDB, and Neo4J labs, will include open-ended questions that encourage students to use TriQL to solve problems and reflect on the differences between the relational, graph, and document-oriented models and their query languages. We will design this lab to showcase the advantages and disadvantages of each data model. For example, students can work on a scenario in which data entities are highly connected. Cypher (graph) and

the graph model of Neo4J would be more effective in such a scenario compared to SQL or MongoDB.

By developing TriQL, we aim to influence other engineering courses and education disciplines. Computer Science introductory programming courses introduce students to several programming paradigms, such as object-oriented, functional, and imperative. Electrical Engineering courses that teach chip and circuit design can deploy our methodology by exposing students to hardware programming languages, such as Verilog, VHDL, and C. These courses can use the exploratory methodology we implemented in TriQL to help students learn the trade-offs between different programming paradigms.

5 Related Work

While there is not much research on students' difficulties while learning different database models, there are multiple reports of instructors including other database models into their database courses[2,3,4,5]. Mohan reported experiences of a database education curriculum that incorporated NoSQL[4]. In Mohan's work, students were exposed to several NoSQL paradigms and had a set of projects, lab and research assignments to complete using the knowledge they gained during the course. However, their course did not provide labs for exploring the trade-offs between different database models. Other NoSQL databases have also been incorporated into university curricula. For example, Fowler et al. reported their experience in two database courses with teaching CouchDB, a NoSQL data management system that uses JavaScript as its query language [5]. They mainly focused on measuring students' improvement of understanding NoSQL systems.

Researchers have proposed several tools for teaching databases. SQLator, proposed by Sadiq et al.[19], is an SQL learning tool that attempts to evaluate student queries. It compares SQL queries to plain English prompts to verify whether a student-written query is correct. SQLator uses a 'workbench' of tools, including a multimedia tutorial and a collection of practice databases. While this provides students with additional resources, it is focused on the relational database and SQL, and thus inadequate for teaching the trade-offs of the relational, graph, and document-oriented databases. Other database learning tools include WebSQL[20], an interactive system for executing SQL queries, and MDB[21], a tool for teaching MongoDB. These tools also focus on helping students learn a particular database system, and none of these tools combines relational and NoSQL. TriQL is different as it allows students to learn query languages, focusing on teaching students the trade-offs between the various data models and query languages.

6 Future work and Conclusions

We developed TriQL, a system for helping novices learn three major database systems, including relational (MySQL), graph (Neo4J), and document-oriented (MongoDB), and their query languages. Learners can explore the trade-offs among data models and the different querying paradigms, including the abstract declarative paradigm of SQL and Cypher and the imperative paradigms of MongoDB shell. We also discussed how TriQL can be integrated into an introductory database curriculum as part of the database programming lab assignments.

In the future, we plan to improve TriQL's functionality by allowing users to submit queries in a database programming language (without using the GUI) and see the equivalent query in another database programming (query) languages. We also plan to develop an API (Application

Programming Interface) to integrate TriQL with online assessment tools, such as PrairieLearn. Most importantly, we plan to use TriQL in our database course (CS 411) to evaluate its learning effectiveness. We will have students use TriQL as part of lab and homework assignments. We will then conduct quantitative and qualitative analysis to measure the impact of TriQL in students' understanding of database schema and query languages.

References

[1] H. Garcia-Molina, *Database systems: the complete book*. Chennai, Tamil Nadu, India: Pearson Education India, 2008.

[2] M. Guo, K. Qian, and L. Yang, "Hands-on labs for learning mobile and nosql database security," in *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2, pp. 606–607, IEEE, 2016.

[3] L. Li, K. Qian, Q. Chen, R. Hasan, and G. Shao, "Developing hands-on labware for emerging database security," in *Proceedings of the 17th Annual Conference on Information Technology Education*, SIGITE '16, (New York, NY, USA), p. 60–64, Association for Computing Machinery, 2016.

[4] S. Mohan, "Teaching nosql databases to undergraduate students: A novel approach," in *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, SIGCSE '18, (New York, NY, USA), p. 314–319, Association for Computing Machinery, 2018.

[5] B. Fowler, J. Godin, and M. Geddy, "Teaching case: Introduction to nosql in a traditional database course."

[6] D. M. Kristin Tufte, "Data management in the cloud," 2014.

[7] S. Davidson, "Data management in the cloud," 2020.

[8] Anonymous, "Database systems," 2018.

[9] J. Minker, "Logic and databases: Past, present, and future," *AI Mag.*, vol. 18, pp. 21–47, 1997.

[10] Y. Wu, A. Alawini, D. Deutch, T. Milo, and S. Davidson, "Provcite: Provenance-based data citation," *Proc. VLDB Endow.*, vol. 12, p. 738–751, Mar. 2019.

[11] Y. Wu, A. Alawini, S. B. Davidson, and G. Silvello, "Data citation: Giving credit where credit is due," in *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, (New York, NY, USA), p. 99–114, Association for Computing Machinery, 2018.

[12] A. Alawini, S. B. Davidson, G. Silvello, V. Tannen, and Y. Wu, "Data citation: A new provenance challenge.," 2018.

[13] S. Overflow, "Stack overflow developer survey 2019," 2019. [Online; accessed 10-January-2020].

[14] J. Patel, "The 9 most in-demand programming languages of 2017," 2017. [Online; accessed 10-January-2020].

[15] J. Guia, V. G. Soares, and J. Bernardino, "Graph databases: Neo4j analysis.," in *ICEIS (1)*, pp. 351–356, 2017.

[16] D. Fernandes and J. Bernardino, "Graph databases comparison: Allegrograph, arangodb, infinitegraph, neo4j, and orientdb.," in *DATA*, pp. 373–380, 2018.

[17] MongoDB, Inc., "Mongodb," 2019.

[18] M. West, G. L. Herman, and C. Zilles, "Prairielearn: Mastery-based online problem solving with adaptive scoring and recommendations driven by machine learning," in *2015 ASEE Annual Conference & Exposition*,

no. 10.18260/p.24575, (Seattle, Washington), pp. 26.1238.1–26.1238.14, ASEE Conferences, June 2015. https://peer.asee.org/24575.

[19] S. Sadiq, M. Orlowska, W. Sadiq, and J. Lin, "Sqlator: An online sql learning workbench," *SIGCSE Bull.*, vol. 36, p. 223–227, June 2004.

[20] G. N. Allen, "Websql: An interactive web tool for teaching structured query language," *AMCIS 2000 Proceedings*, p. 384, 2000.

[21] M. M. Hilles and S. S. A. Naser, "Knowledge-based intelligent tutoring system for teaching mongo database," 2017.