Echelon: An AI Tool for Clustering Student-Written SQL Queries

Matthew Weston*, Haorong Sun*, Geoffrey L Herman*, Hisham Benotman†, Abdussalam Alawini*
*University of Illinois at Urbana-Champaign
{mweston, haorong4, glherman, alawini}@illinois.edu
†Purdue University
hbenotma@purdue.edu

***Abstract*—As part of teaching SQL, instructors often rely on auto-grading systems for marking students' assignments. However, such systems lack essential insights into the approaches students use to solve these assignments, allowing subtle flaws in student intuition to go unseen. Further, manual analysis of students' code submissions ranges from costly to impossible, depending on the assessments' frequency. In this paper, we present a system capable of extracting features that instructors deem significant from students' SQL queries and using them to generate clusters that capture the key approaches taken. To supplement this, we project the extracted information to an interactive dashboard and demonstrate its usefulness in allowing database systems professors and teaching staff to quickly identify trends in students' solutions.**

***Index Terms*—Databases, Education, Visualization**

I. Introduction

Databases are pervasive and vital to a wide range of domains, such as medical, financial, scientific, and consumer contexts [1]. They are used by software developers, researchers, and business analysts. People with data-related skills are becoming more and more in demand [2]. Structured Query Language (SQL) is the de facto standard for querying relational databases [3]. By some metrics, it is the most in-demand programming language [3], [4]. Knowledge of SQL is vital to the use of data science for accelerating scientific inquiry, business decision-making, and effective healthcare. Critically, SQL does not require prerequisite knowledge of other programming language paradigms and can be immediately useful to a wide range of professionals, providing an otherwise overlooked and underused gateway into computing disciplines.

SQL is a declarative query language, which is a completely different paradigm than other common introductory languages such as Python, Java, or C. Consequently, students learning SQL often struggle to shift from the procedural paradigm they learned in their first year of the CS curriculum to SQL's declarative paradigm [5]–[7]. Sadiq et al. stated that "the declarative nature of SQL is rather difficult for many learners to grasp. Whereas procedural languages allow students to approach complex problems in steps, SQL requires learners to think sets rather than steps" [6].

Lab coding assignments can be a practical teaching approach for bridging this gap. Working through practical problems allows students to test what they have learned. Simultaneously, the presence of an educator helps them reinforce correct assumptions, discard incorrect ones, and, more importantly, bridge the gap between the declarative and procedural paradigms. This active learning teaching methodology can increase classroom engagement and aid students in building a connection between the theory being taught and its practical implications.

Unfortunately, the automatic code grading systems typically used are often insufficient to achieve these ends [8]. Running student SQL code and providing a binary correctness score can lead to students developing bad habits, or fortify an incorrect understanding of how their SQL code operates after being compiled. In the worst-case scenario, students can repeatedly make arbitrary adjustments to their submissions until success, or even attempt to game the grading mechanism by submitting a simpler than intended SQL code that returns the same output.

Manually examining student responses is prohibitively time-consuming, especially considering that spotting bad SQL coding practice takes a trained eye and careful examination. Accordingly, the motivation exists for developing a second layer of automation capable of carrying out the most labor-intensive parts of this analysis on its own, and passing an intelligent grouping of submissions' programming styles and approaches on to the instructor, who can immediately make use of this information to provide feedback to students while their work is fresh in their memories.

As an example, consider a scenario in which a student has struggled through a programming assignment, repeatedly tweaking a poorly constructed SQL query until its output satisfies the grading criteria. Through the use of our system, the instructor can identify the disparate classes of submissions, and explain the flaws present in popular but sub-optimal approaches to the class. This provides our student with directly relevant information that can be used to build a more complete understanding of the course material.

In this paper, we present Echelon[1], an AI-based system capable of automating the most labor-intense parts of assessing student-created SQL queries. Using anonymized student data collected from two large U.S. public universities, Echelon's recurrent module learns the syntactic and semantic patterns

[1]Birds flying in a V shape create an echelon. The analogy here is that our system forms clusters (echelons) of similar students' SQL queries.

most relevant to student success, and use them to extract a set of learned features. These features, while often intractable to hard-coded, hand-designed algorithms, are essential in meaningfully classifying student responses. The output of this process is used to create a two-dimensional projection, which is then linked to a dashboard that instructors can use to rapidly assess class performance, using clustering algorithms to group student approaches into clean, intuitive categories. Instructors can then address a variety of student approaches, and thus create a more responsive classroom.

We begin by presenting a review of past literature in this area in Section 2, followed by an overview of each of our system's three core modules (Section 3). Finally, in Section 4, we present our preliminary evaluation of Echelon.

II. Related Work

A great deal of work has been done in automating the analysis of student code, and in developing machine learning systems capable of extracting meaning from input code. In this section, we present a concise overview of the research we have built upon.

In pursuit of a more responsive set of automated systems, numerous researchers and educators have developed systems for automatically assessing student-written code. Brusilovsky [9] et al., Welty [10], and Poulsen et al. [11] each discuss strategies for evaluating students' approaches to SQL programming, though each of these authors focus on non-time-sensitive analysis whereas Echelon is designed for live use. Alhadi et al. [12] propose a similar system, though it is designed around the analysis of output vectors rather than student approaches.

Several authors have made use of automated techniques to perform abstract code analysis. In the domain of security, Li et al. [13] make use of long short-term memory units to more effectively detect SQL injection attacks, demonstrating that machine learning techniques can be useful in analyzing abstract features in sections of code. In the education domain, Huang et al. [14] make use of a rule based classifier to guage the viability of a student's approach to a given task, and Macnish [15] makes use of directed graphs and various clustering algorithms to categorize student errors in Java programming.

We have also analyzed prior learning tools and teaching methodologies in order to verify that Echelon serves a valuable role. Renaud et. al [16] compare two models for teaching SQL, and discuss the tools that would be most valuable in light of their analysis. They consider the importance of teaching concepts in a way that allows them to be built upon, rather than focusing on immediate success, which dovetails with Echelon's purpose: providing feedback based on correctness alone is insufficient to teach a student strong intuition. The authors go on to underscore that any tool used should not encourage students to frame their model of SQL around use of the tool, lest they become dependent on it. Echelon works exclusively by displaying different student approaches, and avoids encouraging any one approach over another. Rather than encouraging a certain paradigm for learning SQL, Echelon allows instructors to discuss the mental models employed by students, encouraging them to adjust errors in intuition as soon as they become apparent.

SQLator, proposed by Sadiq et. al [6], is an SQL learning tool with several qualitative similarities to Echelon, in that its key function attempts to provide an ability to evaluate student queries. It compares SQL queries to plain English requests, and, with a high success rate, determines whether the two match in order to verify whether a student-written query is correct. This allows for rapid formulation of practice assignments, but is largely aimed at easily confirming correctness rather than analyzing flaws in a student's approach. Unlike Echelon, SQLator bases this capability out of a 'workbench' of tools, including a multimedia tutorial and a collection of practice databases. While this provides students with additional resources, it also limits flexibility in a way that Echelon, which is capable of aiding analysis of any arbitrary set of queries, does not.

III. System Overview

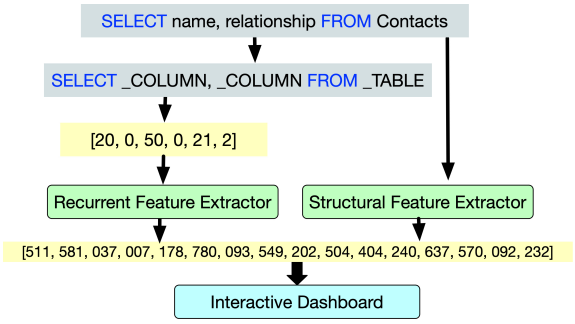


Fig. 1. Echelon's System Architecture: Echelon's takes students' submitted queries and presents clusters of students' queries in an interactive dashboard

Our system consists of three core parts. A recurrent feature extractor is used to provide an abstract notion of query quality to Echelon's model, a bag-of-words model is used to provide an overall structure, and an output analysis system uses the features extracted by these modules to generate a clear, intuitive visualization for use by a course instructor. In this section, we explore the functionality of each of these.

A. Recurrent Feature Extractor

Recurrent neural networks (RNNs) [17], which iteratively process a sequence of inputs by taking in both the next input in the sequence and the network's previous output, are popular for a wide variety of tasks, from literary sentiment analysis to music generation. Long Short Term Memory networks [18], or LSTMs, are a subcategory of recurrent neural networks that specialize in maintaining information across long sequences, and perform well in most applications where RNNs are typically used.

We extract meaning from a set of tokenized queries using a recurrent neural network. The sections below describe the process used to create our feature extractor and its role in our complete system.

1) Generalization and Tokenization: In order to prepare data for the recurrent feature extractor, we first generalize and tokenize each SQL query. Generalization is the process by which question or student-specific keywords, such as table and column names, are removed from the data being fed to the system, allowing it to learn a general representation of query structures and how they relate to student success. SQLParse, a non-validating SQL-parsing library, is used for this purpose. The generalized queries are then tokenized, with each word in the vocabulary being converted to an integer with one-to-one correspondence. These integers are referred to as tokens.

Tokens are provided to an embedding network, which outputs a vector of values representing their significance to the set of all SQL queries used to train the network. A recurrent neural network module then uses each embedding in an input sequence to iteratively update its internal state using a series of differentiable mathematical functions. Lastly, this internal state is directed to a simple, feed-forward neural network, which is used to extract a final set of features.

2) Architecture and Training: Our recurrent model consists of a four layer bidirectional LSTM with hidden layers of size 64, and a dropout rate of 0.4. We selected these parameters based on the model's performance on the classification task used to train the feature extractor. The LSTM feeds its output into a feature extractor composed of two fully-connected, or 'dense' layers, followed by a classification module made up of a third fully-connected layer.

During training, the complete model is trained through backpropagation to predict whether the entries in a set of student-written queries obtained from a large Database Systems course are correct or incorrect. When the system is in use, the classification layer is ignored, and the output of the feature extractor is collected. In essence, the core process is the same as that employed in some implementations of Faster-RCNN, in which the earlier layers of a network used to extract features for one purpose are reused for a second task, for which training is less tractable. While we ultimately have no interest in predicting the correctness of a given query, a classifier that learns to do so tends to learn to extract significant features that correlate with query quality in the process.

3) Usage: Once the set of responses for a given query has been generalized and converted to tokens, it is batched and fed into the feature extractor. The feature extractor's outputs for each batch are concatenated vertically into a single two-dimensional array. These features are then horizontally concatenated with those provided by the structural feature extractor, and the resulting array is passed, alongside the raw queries, to the output analysis module. Should the provided queries be labeled as correct or incorrect, this information will also be forwarded to the user interface for the purpose of color-coding.

B. Structural Feature Extractor

In addition to the quality-based Recurrent Feature Extractor, we make use of a bag-of-words model to provide a notion of structure to the feature set, augmenting the quality-based features. This is done by gathering the counts of a set of words determined to be common in student-written SQL queries, and concatenating it with the previous set of features. This ensures that the complete set of features can be used to distinguish clear differences in query approach along with more subtle ones. As an example, queries featuring multiple subqueries will be immediately identified by the structural feature extractor, allowing differences determined by the recurrent feature extractor to be used for finer distinctions. We combine this relatively simple approach with the more complex approach above to develop a system that can cleanly make a wide variety of useful distinctions.

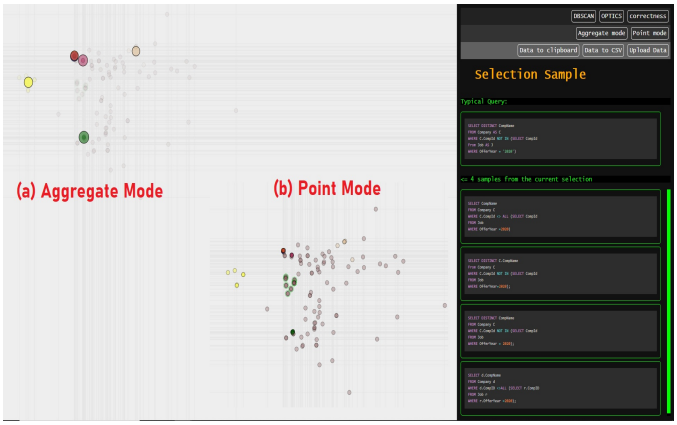


Fig. 2. Echelon's UI: (a) shows the Aggregate Mode with multiple clusters of similar queries (b) shows an excerpt of the plot part of the UI on the same question using the Point Mode

C. Output Analysis and User Interface

Once a set of meaningful features has been extracted from each response associated with a given question, an instructor can make use of our interface to interpret the data, identifying the main approaches taken by students and adjusting lecture content accordingly.

In order to intuitively display the full set of queries to the instructor, principal component analysis is used to project the extracted features down to two dimensions while maximizing the amount of information maintained. The resulting scatterplot, shown in figure 2, allows for groups of similar SQL-writing approaches to be readily identified. Selecting a cluster, or a set of responses (Figure 2(a)), displays a random, anonymized sample of student submissions from the selection in the sidebar. When selecting a cluster, a process described below, the response that is closest to the cluster center will be displayed above this sample, serving as an immediate representation of the cluster's meaning.

Clustering on the extracted features from students' submissions provides a bird's-eye view of the approaches students use to solve a problem, allowing the instructor to make use of information that a two-dimensional projection is unable to completely convey. Selecting a clustering algorithm from the upper toolbar recolors the display according to the clusters identified. To supplement this, the lower toolbar allows the

instructor to display the submissions as a set of aggregates (Figure 2(b)) rather than individual points, allowing complex, high-dimensional categorizations of the data to be easily viewed and interacted with. An instructor can zoom in or out via the scroll wheel to focus on smaller groups of queries, and can move the viewpoint by right-clicking and dragging. In addition to this, the instructor can download the raw data, including the clusters and each query's proximity to the cluster center, enabling further analysis. These features allow Echelon to be applied to both small and large datasets in an intuitive, user-friendly manner. To facilitate widespread adoption, we have designed the interface to rely entirely on universal concepts such as clicking and dragging to scroll and select, and scrolling a mouse wheel to zoom in and out. Accordingly, professors and teaching staff can integrate Echelon into their course structure without any prerequisites.

IV. Instructor Experience Report

To evaluate Echelon's usefulness, we asked two database instructors (both co-authors in this paper) from two universities to use the tool to analyze students' SQL submissions of their homework assignments. The goal of this evaluation is to gather information on the following topics:

1) (**T1**) Do the clusters that Echelon shows represent different student approaches in a way that enhances an instructor's ability to teach?
2) (**T2**) can the instructors identify useful learning tips and insights from the different approaches shown in Echelon's clusters?

In this section, we start by describing the datasets and our evaluation set-up before discussing our preliminary evaluation results.

A. Data Sources

We collect two datasets from database courses offered at two different research-intensive institutions. We will refer to these institutions as UX and UY for anonymization purposes. UX's Database Systems is an elective course taken primarily by upper-level undergraduate and graduate students, while UY's database class is only offered for upper-level undergraduates. The two courses cover SQL, but the UX course focuses on database implementation (systems internals), while the UY course focuses more on the database application aspect.

Data was gathered from the Fall 2020 offering of the two courses. The UX dataset included 4093 "best submissions" (i.e., submissions that received the highest personal score) from 434 students. These submissions were obtained as part of an assignment that included ten questions about SQL queries covering Joins (Inner, Outer joins), Subqueries (Nested queries), Correlated Subqueries, Grouping and Aggregation, Set Operations, Triggers, and Stored Procedures. The UY dataset included 2458 submissions from 167 students in response to 16 questions that covered the following SQL topics: Joins, Grouping (Group By and Having), Aggregate Functions, and Subqueries.

The two courses differ in the way students submit their solutions. For UX, students submit their assignments on an online homework and exam platform, which we will call OHEP for anonymization purposes. The students are allowed an unlimited number of submissions for homework problems without penalties, and the highest score of all submissions are recorded as the final score. Students can also solve the problems in any order and return to any of the problems before the deadline for submissions. Students taking UY's course received a database instance to test their queries before submitting their final solutions to Brightspace, a learning management system. Students' queries are then auto-graded using a Python script and a slightly modified database.

B. Experience Report Set-up

Our preliminary evaluation of Echelon consists of the following steps: (1) Each instructor downloads their students' SQL submissions for each question, (2) uploads each question's submissions to Echelon, and examines the resulting clusters, recording sample queries in each of these clusters. (3) The instructors then analyze the clusters associated with questions covering various SQL topics relevant to their respective courses, and (4) meet to compare their results and discuss their conclusions.

Due to page-limit constraints, we only discuss the largest clusters (or clusters with interesting approaches) and discuss relevant query parts in each of these clusters. However, we plan to make Echelon system, our datasets, and results publicly available after the work is published.

C. Results and Discussion

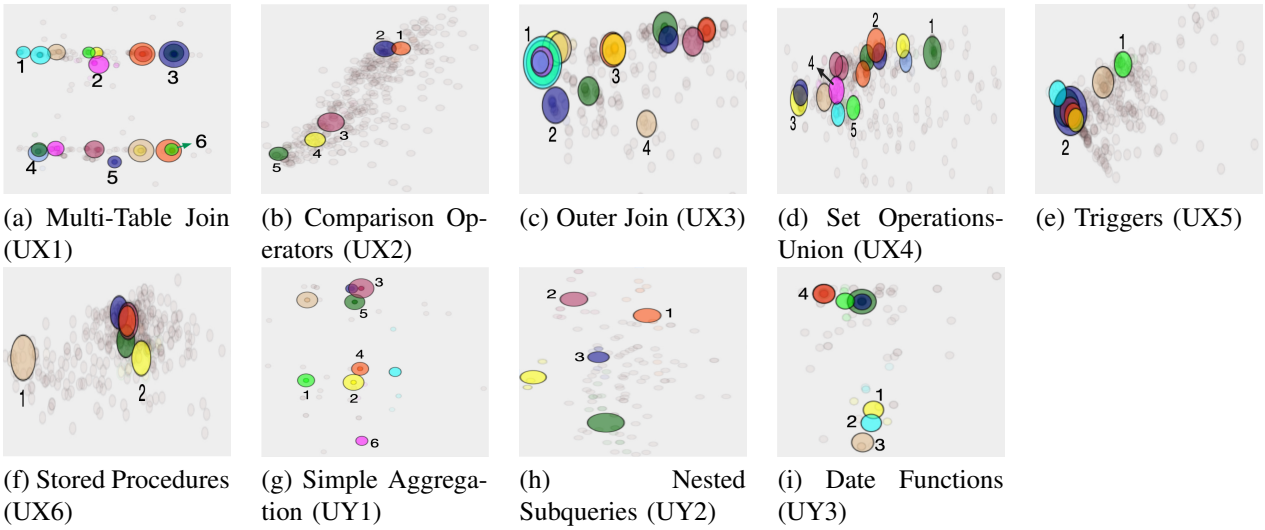
We present the cluster analysis (step 3) results for UX and UY in sections 4.3.1 and 4.3.2, respectively. We discuss the combined analysis of the instructors (step 4) in section 4.3.3.

1) UX Results.: Out of the ten analyzed questions, we select four questions covering Multi-table Joins, Comparison Operators, Outer Joins, and Set Operations (Union). We also briefly discuss some unexpectedly interesting results from the Triggers and Stored Procedures questions. To familiarize the reader with each question, we provide the question prompt along with an example solution query. Please note that each question's prompt shows the schema for each table in the database (omitted here due to space limitations).

Multi-Table Join Question (UX1):

```
Write one SQL query to list the students whose
FirstName OR LastName starts with 'A' and
their instructors' name also starts with 'A'.
Return the students' FirstName, LastName, and
Instructor.
```

```
SELECT Students.FirstName, Students.LastName,
    Courses.Instructor
FROM Students INNER JOIN Enrollments ON Students.
    NetId=Enrollments.NetId INNER JOIN Courses ON
    Enrollments.CRN=Courses.CRN
WHERE (Students.FirstName LIKE 'A%' OR Students.
    LastName LIKE 'A%') AND (Courses.Instructor LIKE
     'A%');
```



(a) Multi-Table Join (UX1) (b) Comparison Operators (UX2) (c) Outer Join (UX3) (d) Set Operations-Union (UX4) (e) Triggers (UX5)

(f) Stored Procedures (UX6) (g) Simple Aggregation (UY1) (h) Nested Subqueries (UY2) (i) Date Functions (UY3)

Fig. 3. Echelon Clustering of the UX and UY Datasets

Figure 3(a) shows the clusters of students' solutions to this question. The largest cluster (cluster 3) contains queries similar to the one provided above, showing that many students approach the question as intended. Cluster six groups students who did not use the Join keyword but instead join their tables by using cross-product, listing the join condition in the WHERE clause (e.g., FROM Students s, Enrollments e, Courses c WHERE s.NetId= e.NetId and e.CRN= c.CRN). It is interesting to see that cluster one groups queries with subqueries in the FROM clause, while cluster five shows queries that make use of the LEFT JOIN operator. Going over this information with students in a timely manner would help them avoid such over-complicated (or incorrect) approaches.

Comparison Operator Question (UX2):

Write one SQL query that returns the NetId, FirstName and LastName of 'CS' students who received the lowest score in any 'ECE' courses they have taken. Make sure to remove duplicates from the query result. Return the results in ascending order by student LastName and descending order by student NetId.

```
SELECT DISTINCT S.NetId, S.FirstName, S.LastName
FROM   Students S, Enrollments E, Courses C
WHERE  S.NetId = E.NetId AND E.CRN = C.CRN AND S.
    Department = 'CS' AND C.Department = 'ECE'
        AND E.Score <=  All (SELECT E2.Score
                             FROM   Enrollments E2
                             WHERE  E2.CRN = E.CRN)
ORDER BY S.LastName, S.NetId DESC;
```

The clusters of UX2 submissions are shown in Figure 3(b). Cluster one groups students who follow an approach similar to the solution above, making use of the comparison operator ALL. Clusters three, four, and five grouped submissions that used the IN operator to answer this question. These clusters differ in the type of JOIN students use. For instance, cluster three contains queries that used NATURAL JOIN and JOIN in one query, where cluster five only lists queries that exclusively used NATURAL JOIN.

Outer Join Question (UX3):

Write one SQL query that returns customers' FirstName and LastName and the count of their purchases, if any. Return the results in ascending order of FirstName and descending order of LastName.

```
SELECT FirstName, LastName, COUNT(PurchaseId) AS cnt
FROM Customers LEFT JOIN Purchases USING (CustomerId
    )
GROUP BY FirstName, LastName
ORDER BY FirstName, LastName DESC;
```

It is interesting to see that the largest cluster (cluster 1) in Figure 3(c) groups queries that use NATURAL JOIN instead of LEFT JOIN, indicating that a large number of students do not understand this part of the question. Students who use the correct approach are grouped in clusters three and four. The main difference between clusters three and four is that many queries in the latter explicitly use the OUTER keyword. Cluster two contains queries that (unnecessarily) use a subquery in the FROM clause to perform the join.

Set Operations Question (UX4):

Write a SQL query that returns the NetID, FirstName and LastName of 'CS' students who have scored more than 80 in 'CS' courses and 'ECE' students who have scored more than 70 in 'CS' courses. Return the results in ascending order of NetId.

```
(SELECT S.NetID, S.FirstName, S.LastName
FROM Students S NATURAL JOIN Enrollments E JOIN
    Courses C USING (CRN)
WHERE S.Department = 'CS' AND C.Department = 'CS'
    AND E.Score  > 80)
UNION
(SELECT S1.NetID, S1.FirstName, S1.LastName
FROM Students S1 NATURAL JOIN Enrollments E1 JOIN
    Courses C1 USING (CRN)
```

```
WHERE S1.Department = 'ECE' AND C1.Department = 'CS'
     AND E1.Score  > 70)
order by NetId;
```

The largest cluster (cluster 1) in Figure 3(d) represents the correct submissions similar to the solution provided above. Cluster two groups students who answered the question correctly using the OR operator in the WHERE clause (instead of UNION), but most students' queries in this cluster do not involve subqueries in the WHERE clause. We notice the use of subqueries with the OR operator in the WHERE clause in clusters three, four and five. The main difference between the last three clusters is the type of join (i.e., NATURAL, INNER, or Cross Product) used in the queries. An example of a query from cluster three is shown below:

```
SELECT NetID, FirstName, LastName
FROM Students
WHERE  Department = 'CS'
and NetID in (SELECT NetID
              FROM Enrollments NATURAL JOIN Courses
              WHERE Department = 'CS' and Score >
                  80)
    OR Department = 'ECE'
            and NetID in (SELECT NetID
                          FROM Enrollments NATURAL
                              JOIN Courses
                          WHERE Department = 'CS'
                              and Score > 70)
ORDER by NetID ASC
```

Triggers (UX5) and Stored Procedures (UX6)[2]:
Triggers and Stored Procedures vary in syntax from one database implementation to the other, as they are not part of standard SQL. Consequently, we decided to exclude keywords related to Triggers and SPs from our model. However, we are very impressed with Echelon's results on questions covering these topics. As shown in Figure 3(e), Echelon is able to identify clusters with different approaches to this problem, as Triggers contain SQL queries within their definition.

We also find similar trends in the SP question (see Figure 3(f)). For instance, the tool groups students who use the IF statement inside the INSERT command in Cluster one and two students who use the IF statement first followed by an INSERT command in Cluster two.

2) UY Results.: Out of the 16 queries, we discuss three queries that cover aggregation, subqueries, and Date functions.

Simple aggregation (UY1):

```
Write one SQL query to find the number of
jobs.
```

```
SELECT COUNT(JobId) as NumberOfJobs FROM Job;
```

Figure 3(g) shows the clusters for this question. Echelon reveals two interesting patterns for this question (a third pattern/cluster contained queries similar to the solution). The main query in clusters one and two is "Select count(*) ...", which is different but still equivalent to the solution. A follow-up lesson may discuss situations where "count(*)" may not work, such as when having null values in the column to be counted (in this case, count(col) and count(*) may return different values). The characteristic of clusters three and four is the use of the DISTINCT keyword either before or within the count function. The DISTINCT keyword has no effect for this table (since jobID is a key). Providing feedback to students about the use of this keyword is an example of an Echelon-motivated change in course material that is likely to help students better understand SQL.

Nested Subqueries (UY2):

```
List companies that received applications
from bright students. A student is considered
bright if they have the highest grade (GPA)
in their school. Eliminate duplicates in your
result.
```

```
SELECT DISTINCT C.CompName
FROM Company C JOIN Job J on C.compid = J.compid
JOIN JobApplication JA on J.jobid = JA.jobid
WHERE JA.studentid in
    (SELECT studentid from Student
     WHERE (schoolid, grade) IN
        (SELECT sc.schoolid, max(grade)
         FROM Student st JOIN School sc on st.
             schoolid = sc.schoolid
         GROUP BY sc.schoolid));
```

We discuss clusters one, two, and three in Figure 3(h). The sample queries we examine in cluster one (five queries) contain the same structure, nesting five queries in the From clause (an excerpt is shown below). The approach is more complicated than the one used in the solution, which nests three queries in the Where clause. In addition, the subquery in the last line presents an incorrect (and common) method, using an attribute (StudentId in this case) that is not aggregated nor included in the Group-By list of attributes.

```
SELECT DISTINCT c.CompName
FROM Company c INNER JOIN
    (SELECT j.CompId FROM Job j INNER JOIN
      (SELECT ja.JobId FROM JobApplication ja INNER
          JOIN
        (SELECT s.StudentId FROM Student s INNER JOIN
          (SELECT SchoolId, StudentId, MAX(Grade) as
              MaxGrade FROM Student GROUP BY SchoolID
              ) AS ...
```

All five queries in cluster two contain the subquery "S.Grade IN (SELECT MAX(Grade) from student GROUP BY SchoolId)". One requirement in the question is to find any top student whose grade (i.e., GPA) is the highest grade in their school. The interesting subquery incorrectly compares a student grade to a set of highest grades (one from each school).

In cluster 3, students use an incomplete Having or Where clause, featuring either "HAVING MAX(Student.Grade)" or "WHERE (SELECT MAX(Grade) FROM Student)" at the end of their queries. Both of these clauses lack a comparison operator.

Because of question complexity, we notice less similarity among the queries in each cluster (compared to easier questions). However, the tool allows us to discover common incorrect blocks included in each cluster's queries. These errors can be further discussed in class lectures to better shape

[2]UX5 and UX6 question prompts and solutions are omitted due to space limitation

course material towards the difficulties faced by students.

Date Functions (UY3):

```
List names of students born in 1992 or 1993.
```

```
SELECT StudentName, EXTRACT(YEAR FROM BirthD) As
    BirthYear FROM Student
WHERE EXTRACT(YEAR FROM BirthD) IN ('1992', '1993')
```

For this question, the tool shows different types of functions the students used to work with the BirthD attribute. Four clusters, shown in Figure 3(i), present distinct approaches, including the use of the EXTRACT function (i.e., EXTRACT(YEAR FROM BirthD)) in cluster 4, the YEAR function (i.e., Year(BirthD)) in cluster 3, the SUBSTRING function (i.e., SUBSTRING(BirthD, 1, 4)) in cluster 2, and the BETWEEN operator (i.e., BETWEEN 1992 AND 1993) in cluster 1. Using the SUBSTRING function in this question was totally unexpected.

3) Discussion Summary: We summarize our separate analysis of the two data sets. We also respond to the initial topics, and claim that Echelon can be useful in an educational setting. **T1**: We found that the system breaks down the different student approaches in a useful manner for most SQL questions. We are able to spot interesting and sometimes unexpected answers or SQL blocks in both simple and complex questions spanning the two data sets. Our results show that Echelon can quickly and competently process SQL-homework submissions, providing an interpretable visualization of interesting variations that can allow instructors to better direct class lectures or online forums. Following from this, an ideal use case involves Echelon being paired with a brief in-class coding assignment, which makes use of any of the questions covered above. The instructor could iterate over the most important clusters, explaining the nature, advantages, and disadvantages of each common approach. **T2**: We also find that analysing homework submissions in Echelon can help instructors provide useful feedback to students. This feedback can range from discussing equivalent queries to reminding students about incorrect and undesirably complicated approaches. The tool allows for rapid analysis of homework, exam, and in-class activity performance that instructors are likely to deem useful in providing rapid, highly relevant feedback to students, which is known to improve student engagement and lead to better educational outcomes [19].

V. Conclusion

We have presented a system capable of processing and extracting meaning from SQL queries written by students, and of conveying the extracted information to an instructor in an interactive visualization. By converting hundreds of multi-line queries into a projection that can be explored in real-time, we allow for quick, comprehensive in-class feedback that takes into account the most common errors and the possibility of multiple correct approaches of varying desirability. Moreover, we present a preliminary experience report demonstrating the usefulness of this system in improving an instructor's ability to examine and address the set of responses to a variety of SQL programming assignments. As an instructor-facing tool, it allows a professor unprecedented agility, creating the potential for real-time lesson plan adjustment. Future work could include the evaluation of clustering systems as a student-facing self-evaluation method, the integration of online clustering into Echelon [20], and methods of automatically generating useful high-level labels for clusters to be displayed directly on the plot.

VI. Acknowledgment

This work is funded by NSF. Award Number: 2021499.

References

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

[2] C. Wills, "Analysis of current and future computer science needs via advertised faculty searches for 2020," 2020, [Online; accessed 13-January-2020]. [Online]. Available: https://cra.org/analysis-of-current-and-future-computer-science-needs-via-advertised-faculty-searches-for-2020/

[3] S. Overflow, "Stack overflow developer survey 2019," 2019, [Online; accessed 10-January-2020]. [Online]. Available: https://insights.stackoverflow.com/survey/2019/

[4] J. Patel, "The 9 most in-demand programming languages of 2017," 2017, [Online; accessed 10-January-2020]. [Online]. Available: https://www.codingdojo.com/blog/9-most-in-demand-programming-languages-of-2017

[5] K. A. T. Folland, "visqlizer: An interactive visualizer for learning sql," Master's thesis, Norwegian University of Science and Technology, 2016.

[6] S. Sadiq, M. Orlowska, W. Sadiq, and J. Lin, "Sqlator: An online sql learning workbench," *SIGCSE Bull.*, vol. 36, no. 3, p. 223–227, Jun. 2004. [Online]. Available: https://doi.org/10.1145/1026487.1008055

[7] C. Welty and D. W. Stemple, "Human factors comparison of a procedural and a nonprocedural query language," *ACM Trans. Database Syst.*, vol. 6, no. 4, p. 626–649, Dec. 1981. [Online]. Available: https://doi.org/10.1145/319628.319656

[8] J. B. Moghadam, R. R. Choudhury, H. Yin, and A. Fox, "Autostyle: Toward coding style feedback at scale," in *Proceedings of the Second (2015) ACM Conference on Learning @ Scale*, ser. L@S '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 261–266. [Online]. Available: https://doi.org/10.1145/2724660.2728672

[9] P. Brusilovsky, S. Sosnovsky, M. V. Yudelson, D. H. Lee, V. Zadorozhny, and X. Zhou, "Learning sql programming with interactive tools: From integration to personalization," *ACM Trans. Comput. Educ.*, vol. 9, no. 4, pp. 19:1–19:15, Jan. 2010. [Online]. Available: http://doi.acm.org/10.1145/1656255.1656257

[10] C. Welty, "Correcting user errors in sql," *International Journal of Man-Machine Studies*, vol. 22, no. 4, pp. 463 – 477, 1985. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020737385800511

[11] S. Poulsen, L. Butler, A. Alawini, and G. L. Herman, "Insights from student solutions to sql homework problems," in *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, 2020, pp. 404–410.

[12] A. Ahadi, V. Behbood, A. Vihavainen, J. Prior, and R. Lister, "Students' syntactic mistakes in writing seven different types of sql queries and its application to predicting students' success," in *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, ser. SIGCSE '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 401–406. [Online]. Available: https://doi.org/10.1145/2839509.2844640

[13] Q. Li, F. Wang, J. Wang, and W. Li, "Lstm-based sql injection detection method for intelligent transportation system," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 5, pp. 4182–4191, May 2019.

[14] J. Huang, C. Piech, A. Nguyen, and L. Guibas, "Syntactic and functional variability of a million code submissions in a machine learning mooc," *AIED 2013 Workshops Proceedings Volume*, vol. 1009, p. 25, 01 2013.

[15] C. MacNish, "Machine learning and visualisation techniques for inferring logical errors in student code submissions," in *ICALT-2002: Proc. 2nd IEEE Int. Conf. on Advanced Learning Technologies*, 2002, pp. 317–321.

[16] R. Kearns, S. Shead, and A. Fekete, "A teaching system for sql," in *Proceedings of the 2nd Australasian Conference on Computer Science Education*, ser. ACSE '97. New York, NY, USA: Association for Computing Machinery, 1997, p. 224–231. [Online]. Available: https://doi.org/10.1145/299359.299391

[17] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning Internal Representations by Error Propagation*. Cambridge, MA, USA: MIT Press, 1986, p. 318–362.

[18] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, p. 1735–1780, Nov. 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[19] S. Cotner, B. Fall, S. Wick, J. Walker, and P. Baepler, "Rapid feedback assessment methods: Can we improve engagement and preparation for exams in large-enrollment courses?" *Journal of Science Education and Technology*, vol. 17, pp. 437–443, 10 2008.

[20] W. BARBAKH and C. FYFE, "Online clustering algorithms," *International Journal of Neural Systems*, vol. 18, no. 03, pp. 185–194, 2008, pMID: 18595148. [Online]. Available: https://doi.org/10.1142/S0129065708001518