# Analyzing Patterns in Student SQL Solutions via Levenshtein Edit Distance

**Sophia Yang**
University of Illinois at Urbana-Champaign
sophiay2@illinois.edu

**Ziyuan Wei**
University of Illinois at Urbana-Champaign
ziyuanw2@illinois.edu

**Geoffrey L. Herman**
University of Illinois at Urbana-Champaign
glherman@illinois.edu

**Abdussalam Alawini**
University of Illinois at Urbana-Champaign
alawini@illinois.edu

## ABSTRACT

Structured Query Language (SQL), the standard language for relational database management systems, is an essential skill for software developers, data scientists, and professionals who need to interact with databases. SQL is highly structured and presents diverse ways for learners to acquire this skill. However, despite the significance of SQL to other related fields, little research has been done to understand how students learn SQL as they work on homework assignments. In this paper, we analyze students' SQL submissions to homework problems of the Database Systems course offered at the University of Illinois at Urbana-Champaign. For each student, we compute the Levenshtein Edit Distances between every submission and their final submission to understand how students reached their final solution and how they overcame any obstacles in their learning process. Our system visualizes the edit distances between students' submissions to a SQL problem, enabling instructors to identify interesting learning patterns and approaches. These findings will help instructors target their instruction in difficult SQL areas for the future and help students learn SQL more effectively.

### Author Keywords

SQL; database education; online assessment

### CCS Concepts

**•Applied computing → Education; •Social and professional topics → Computer science education;**

### Introduction

The Structural Query Language (SQL) is the defacto data management language that is supported by most Database Management Systems [4]. Acquiring SQL skills is vital since this declarative, highly structured query language is the dominant database language [6]. Due to its English-like syntax, learning SQL does not depend on expertise in other programming languages, making it much more accessible for beginners. However, beginner SQL learners often experience several difficulties acquiring this skill [6]. However, little research has documented examining how students learn SQL and the difficulties they face in their learning journey.

To examine how students learn SQL, database instructors often manually analyze student submissions to a given SQL problem. For instructors who use auto-graders, there is an excellent opportunity for tracing all attempts of a student to solve a problem, giving great insights on how students progressed toward the correct solution. Gaining such insights would help instructors determine what SQL concepts students struggle with and adjust their course plan to mitigate these struggles. The questions then become 1) how do students learn SQL? and 2) how can we empower instructors to teach SQL effectively and with adaptability to tailor to different students?

Our research questions originated from a desire to improve students' educational quality working on SQL in the Database Systems course offered at the University of Illinois at Urbana-Champaign. The Database Systems course has more than 400 students. Students in this class are given multiple SQL in-class exercises and a homework assignment containing 10-15 SQL problems. The instructor cannot quickly identify class-wide struggle areas in SQL or pinpoint students who are challenged. Due to the class size, the number of submissions is too large to analyze manually as, on average, a student can submit well over 20 submissions on a single problem.

Research shows that students take different learning paths, and with the instructors' ability to identify the way students learn, students' learning experience can be improved significantly [5]. To that end, this paper introduces a technique for analyzing students' progress as they solve an SQL problem. This technique uses Levenshtein Edit Distances to compute the distance between students' every submission and their final submission to understand how they reached their final solution. We examine students' submissions to homework


*L@S '21, June 22–25, 2021, Virtual Event, Germany.*

ACM ISBN 978-1-4503-8215-1/21/06.
http://dx.doi.org/10.1145/3430895.3460979

assignments offered in the Spring 2020, Summer 2020, Fall 2020, and Spring 2021 semesters. Our system visualizes the edit distances of all submissions of a student working on an SQL problem. These visualizations enable instructors to identify interesting learning patterns and approaches, which will help them target their instruction in difficult SQL areas for the future and help students learn SQL more effectively.

***Related Works.*** QueryViz is a novel visualization tool for SQL queries that reduces the time needed to read and understand existing queries [3]. However, we allow instructors to examine changes in students' solutions between submissions, identifying new approaches students used to solve a problem. The SQL-Tutor is essentially a support platform for giving students a path to the correct solution [6]. However, using this feedback system may limit the student's creativity and problem-solving skills. Our research analyzes how the students approached the SQL problem as they gradually developed the correct answer. Ahadi et al. conduct work [1] that measured difficulty by examining whether a student reached the correct solution instead of looking at how each student conquered their obstacles and made progress toward the right solution with each submission. Cagliero et al. worked on errors and difficult SQL areas that troubled the students in an aggregate form [2]. This seems helpful for instructors to have a quick overview of challenging areas for students but fails to deliver upon answering the question of how students overcame the difficulties they faced. In our research, the main difference is we compute the Levenshtein Edit Distances between each students' submission with their final submission to gain a deeper understanding of why the students made the specific amount of changes they did before their next submission.

## Data Collection

We collected data from CS 411, a Database Systems course offered at the University of Illinois at Urbana-Champaign. Due to the COVID-19 pandemic, instruction for Spring 2020, Summer 2020, Fall 2020, and Spring 2021 semesters have been delivered remotely following a flipped-classroom model. Students review prerecorded lectures and answer a short quiz about the lecture. Students work on group exercises during the class meeting time to solidify their understanding of the prerecorded lectures. They also worked on a week-long period to complete the relevant homework assignment. Course enrollment was as follows: 303 Spring 2020, 223 Summer 2020, 403 Fall 2020, and 417 in Spring 2021.

***Description of Homework Assignments.*** We collected our data in an online learning management system that auto-grades code and immediately gave students feedback regarding their submission. The online learning management system compares the student solution query's output datasets with the expected solution's outputs to validate the students' solutions on a binary grading scale. If the student query passes some test cases, students are notified that their work was partially correct. However, the students would not yet receive credit for their work. The student can see both the actual data table produced by their own submitted query and the desired data table output. If the students' solution was incorrect, the student might provide a different query until the deadline, until they answer the question correctly, or until they choose to move on. The students may re-answer any questions they have gotten credit for to test if a different solution query that outputs the same data table result exists. An example of an SQL problem (Spring 2021) and its instructor solution is shown below.

> *Write an SQL query that returns the ProductName of each product made by the brand 'Samsung' and the number of customers who purchased that product. Only count customers who have purchased more than 1 Samsung product. Order the results in descending order of the number of customers and in descending order of ProductName.*

```
SELECT Pr1.ProductName, COUNT(C1.CustomerId) as
    numCustomers
FROM Products Pr1 NATURAL JOIN Purchases Pu1 NATURAL JOIN
    Customers C1
WHERE Pr1.BrandName = 'Samsung'
    AND C1.CustomerId IN (
        SELECT C2.CustomerId
        FROM Customers C2 NATURAL JOIN Purchases Pu2
            NATURAL JOIN Products Pr2
        WHERE Pr2.BrandName = 'Samsung'
        GROUP BY C2.CustomerId
        HAVING COUNT(C2.CustomerId) > 1
    )
GROUP BY Pr1.ProductName
ORDER BY numCustomers DESC, Pr1.ProductName DESC;
```

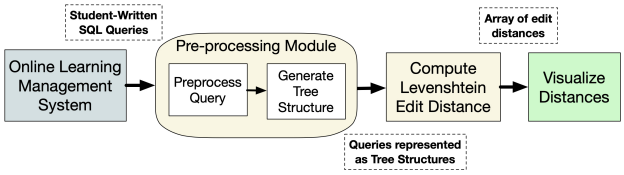


**Figure 1. System Overview Diagram**

## System Overview

Figure 1 shows an overview of our system, which is mainly composed of two parts. First, the SQL Analyzer, which preprocesses students' submissions and computes the edit distance. Second, the Submissions Visualizer, which visualizes each student based on their Levenshtein Edit Distances between their submissions. The SQL Analyzer's components are the preprocessing module and the Levenshtein Edit Distance computation module. We have preprocessed SQL queries to remove insignificant parts in the preprocessing module, such as comments and redundant punctuation, including new-line characters and white spaces. The results are then stored in a tree-like structure. We calculate the Levenshtein Edit Distances between the students' current submissions and the students' final submissions by comparing the tree data structure containing the SQL query submissions. The Submissions Visualizer constructs each individual's plot using Levenshtein Edit Distance, produced by the SQL Analyzer, providing insights regarding the students' learning process.

***Computing Levenshtein Edit Distance.*** We compute the difference between each student's current submission and the final submission by calculating the Levenshtein Edit Distances between the two SQL queries' tree structures from the previous preprocessing module. To accomplish this part, we apply the edit distance algorithm to compute the slightest change in

the components inside the tree structures, which includes any additions, deletions, and modifications. The SQL statement components are separated into two groups for the computation: 1) SQL keywords (such as "SELECT" or "WHERE") and 2) other attribute names (such as table names, aliases, and data schema attributes). The two groups are then iteratively computed for the Levenshtein Edit Distance. The sums are added together to achieve the final edit distance between that submission and the final submission.

***Visualization and Findings.*** We use Python's matplotlib library to visualize the edit distances' between each student submission for each student. The plot's x-axis represents the student submission number for that particular problem (sorted in chronological order), and the y-axis of the plot represents the Levenshtein Edit Distance between the student's current submission and the final submission. The last submission usually indicates the correct solution, and the edit distance between the last submission and itself is always zero, which explains the dip at the end of the graph. The graph shows that the major turning points in student submissions are reflected by the major hikes/dips in the graph. By comparing the edit distance graphs of each student with their SQL submissions, we can see that students commonly use a trial-and-error or divide-and-conquer approach, or a combination of both. For the former approach, minor changes in the edit distance values are detected consecutively and are usually a result of syntax errors. For the latter approach, more significant changes in the edit distance values are detected and are commonly a result of semantic errors, which require a change in the approach taken. These modifications more easily allow for more considerable edit distances to appear.

## Results

In this section, we present our findings and insights from the visualizations generated by our system. We focus on a subset of students who worked on the question featured in the Data Collection Section. In Figure 2, we have the Levenshtein Edit Distance graph that includes the edit distances between each submission for homework question 14 and the student's final submission. We will analyze this case study to validate our findings. In the first 14 submissions, the student's SQL query had syntax errors and would not run properly. The queries terminated with an error message each time, targeted at helping the student identify the issue's source. An example of an error that the student had received is shown here:

> 1052 (23000): Column 'BrandName' in field list is ambiguous

The error had resulted from the query shown here:

```
Select prn, id
From (Select ProductName as prn, BrandName as br, Count(
    CustomerId) as id
From Purchases natural join Products join Brands on Brands
    .BrandName = Products.BrandName
Group By prn) T
Where id > 1 AND br = 'Samsung'
Order by id desc, prn asc;
```

The source of the error is in the second clause of the query, where "From" is located. Because both the Products table and the Brands table have BrandName as an attribute, it is unclear from which table the student's query is calling the BrandName attribute from. One way to eliminate this error involves explicitly calling the table name, such as Brands.BrandName or Products.BrandName.

From submission 15, the student was able to run the query without any syntax issues. However, the resulting dataset that was queried did not match the correct solution (semantic error). With the higher edit distance jump from submission 14 to 15, we detected that the student took a different approach to the homework problem. The previous query was commented out, and a shorter, simpler query was submitted for submissions 15 through 17. Here is submission 15:

```
Select pr.ProductName as prn, pr.BrandName as br, pur.
    CustomerId as id
From Purchases pur natural join Products pr join Brands b
    on b.BrandName = pr.BrandName
```

Submissions 16 and 17 have a very small edit distance compared with submission 15, and we can see that 16 and 17 are queries that build upon the approach that was taken in submission 15. Submission 16 appends the following clause to submission 15:

```
Where br = 'Samsung'
```

Submission 17 appends this clause instead:

```
Where b.BrandName = 'Samsung'
```

The dip in the graph (Figure 2) at submission 18 indicates that the edit distance fell back to a similar level as with submissions 1-14. With a look into the student's solution query, we can see that the previously commented-out approach was readopted with a slight change that resulted in a SQL syntax error. The student again abandoned the approach at submission 19 and commented out the old approach, using the approach seen in submissions 15-17. Using the automated grading system's feedback, the student consistently makes minor changes to the query through submissions 19-32.

At submission 33, the spike in the Levenshtein Edit Distance graph is explained by the student adopting a third approach while commenting out the previous two approaches; the third approach is highlighted here:

```
Select pur.CustomerId, pr.ProductId, br.BrandName
From Purchases pur natural join Products pr join Brands br
     on br.BrandName = pr.BrandName;
```

The student attempts to build upon this third approach, trying out different combinations of clauses previously demonstrated in the first and second approaches until submission 47.

During submission 47, the student starts again with a fresh, straightforward approach and cleans up the solution query, leaving out the commented portions as shown here:

```
Select CustomerId, ProductName, BrandName
From Purchases natural join Products;
```

The student drastically adds clauses into the query at submission 49 and tries to merge clauses from earlier approaches into this clause through a trial-and-error process. Submission 49 uses submission 47 as a sub-query:

```
Select CustomerId, ProductName, Count(ProductName) as nump
From (
Submission 47
Where BrandName = 'Samsung') T
Group by CustomerId;
```

At this point, the student appears to construct and test the sub-query of its final solution. The query used by the student in submissions 49-58 appears to be the sub-query in submission 59, hence the significant drop in edit distance in the submission graph. Submission 58 is exhibited below:

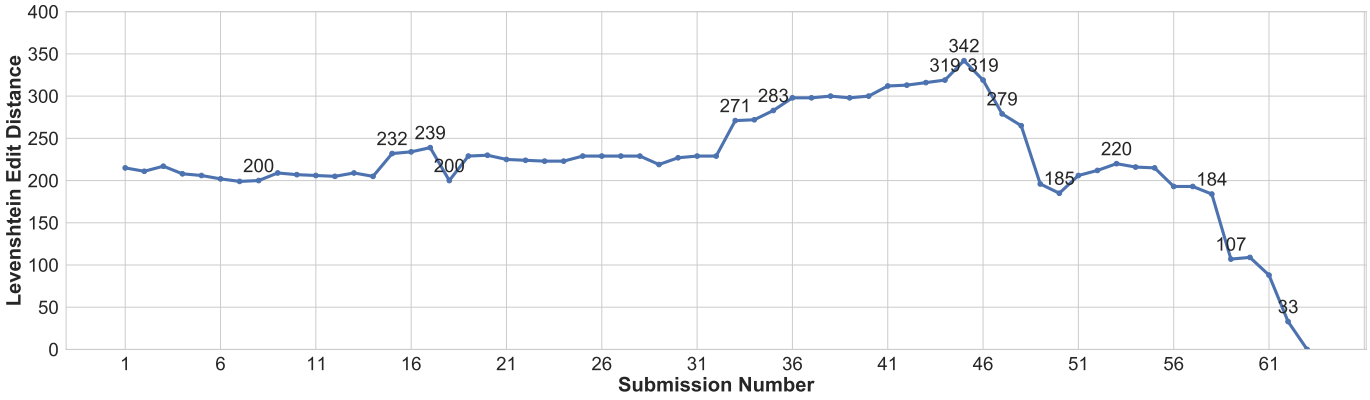


**Figure 2. User 277's visualization of SQL submissions for question 14**

```
Select CustomerId
From (
Submission 49
) T1
Where nump >= 2;
```

The student makes a few minor edits in the query between submissions 59 and 61. Submission 59 is as follows:

```
Select CustomerId, ProductName
From Purchases natural join Products
Where CustomerId IN (
Submission 58
) T2;
```

Finally, we see the last significant drop in edit distance in the submission graph between submissions 61 and 63. This was a result of a structural change in the query. The approaches taken in submissions 59-61 serve as the sub-query for the final submission, which is displayed here:

```
Select ProductName, Count(*) as count2
From (
Approach from submissions 59-61
) T2
Group By ProductName
Order by count2 desc, ProductName desc;
```

By validating our visualizations against the student submission queries, we can see that there are a few trends students take to solving the homework problem. The first type builds upon their solution from the earlier approaches through a combination of divide-and-conquer and trial-and-error. Thus, they change their approaches throughout their submission history and become increasingly closer to a form of the correct solution. We also detect another type of trend when there are the same consecutive edit distance values on the submission graph. These changes usually either identify students cleaning up their query (removing commented-out sections, since commented content does not count towards edit distance) or students becoming frustrated at the SQL problem and are attempting to resubmit the same incorrect solution in hopes of passing the auto-grader tests. With these trends in mind, we have made it easier for instructors to target tricky areas for students through the visualizations and the learning path they may take in working with SQL queries.

## Conclusion and Future Work

We have presented a novel system for analyzing students' SQL submissions by visualizing the Levenshtein Edit Distances between their submission entries and their final submission. Such visuals are revolutionary to instructors for pinpointing where students changed their approach through sharp turns in the submission graph, since it's unrealistic for instructors to manually sift through all the submissions in a large class. Building upon this system, we plan to utilize global sequence alignment algorithms within a two-dimensional dynamic programming array; this will explicitly highlight the path regarding how the student constructed their solution and provide deeper insights of an individual student's thinking patterns. In addition, we also plan to extend this system by building a framework using hierarchical clustering; this will give instructors insights on a class-wide level, leveraging the submissions in an aggregate format. By comparing and clustering the final submissions of students' SQL queries, we will be able to identify common patterns or approaches taken by students. This will enable instructors to familiarize with how their students are learning or using SQL, and may include features such as an advanced plagiarism detection aid, all in a more time-efficient format.

## References


[1] A. Ahadi, J. Prior, V. Behbood, and R. Lister. 2015. A Quantitative Study of the Relative Difficulty for Novices of Writing Seven Different Types of SQL Queries. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '15)*. ACM, New York, NY, USA, 201–206.

[2] L. Cagliero, L. De Russis, L. Farinetti, and T. Montanaro. 2018. Improving the Effectiveness of SQL Learning Practice: A Data-Driven Approach. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 01. 980–989.

[3] J. Danaparamita and W. Gatterbauer. 2011. QueryViz: Helping Users Understand SQL Queries and Their Patterns. In *Proceedings of the 14th International Conference on Extending Database Technology (EDBT/ICDT '11)*. ACM, New York, NY, USA.

[4] Ashley DiFranza. 2020. 5 Reasons SQL is the Need-to-Know Skill for Data Analysts. (2020).

[5] Robert C. Jinkens. 2009. Nontraditional Students: Who Are They? *SIGCSE Bull.* 43, 4 (Dec. 2009), 979–987.

[6] A. Mitrovic. 1998. Learning SQL with a Computerized Tutor. In *Proceedings of the Twenty-Ninth SIGCSE Technical Symposium on Computer Science Education (SIGCSE '98)*. ACM, New York, NY, USA, 307–311.