USING ORACLE QUERY PLAN FOR AUTOMATED ASSESSMENT OF SQL

D. Nelson, K. McGarry, M. Proud

University of Sunderland (UNITED KINGDOM)

Abstract

In our second-year undergraduate database foundations module students are exposed to using the Oracle cloud environment for developing SQL (Structured Query Language) queries. Their first summative assessment for the module is of a problem-solving nature, whereby they are expected to repair a partially developed SQL schema for a given scenario, before subsequently developing SQL queries based on the scenario. Marking this work can be laborious, error prone and time consuming, as well as requiring multiple sample solutions for each query which cater for all potential solutions that the students could produce. The aim of this paper is therefore to describe an early-stage prototype automated assessment system for marking the SQL queries developed by the students.

Keywords: SQL, automated marking, assessment.

# INTRODUCTION

At the university our Computing students take a second-year undergraduate database module in which they are taught fundamentals of database systems, and practice their SQL development skills using the Oracle Cloud environment . The module contains two practical, summative assessments, and their first assessment is of a problem-solving nature, whereby they are expected to demonstrate their fundamental SQL skills by repairing a partially developed SQL schema for a given scenario, and then developing a set of SQL queries to solve various questions expressed in natural language.

Students submit a complete SQL script file containing the corrected database schema and their SQL solutions to the given queries. This is then marked by the module tutors, with the SQL code being marked by hand, which can be laborious, error prone and time consuming. Student SQL solutions can be written in a variety of methods which may all be correct, so sample solutions need to be developed which cater for all potential correct solutions, especially when there are multiple markers on the module.

Our proposed solution is to investigate the use of the query optimisation tool in Oracle, alongside various machine learning techniques, to enable automated marking and grading of ad-hoc SQL queries against one sample solution to improve accuracy of automated grading.

The aim of this paper is therefore to describe a prototype automated assessment system for marking the SQL queries developed by the students and outline future work. The paper will therefore discuss and provide examples of SQL queries where we are able to match student solutions using real world student solutions, discuss issues which are currently being enhanced, before moving on to investigate how machine learning and natural language processing could be effectively developed as future work to allow for partial grading of student solutions which are not one hundred percent correct according to the sample solution provided by the module tutor. It will outline an undergraduate final year project which is in the early stages of investigating these technologies and how they could be implemented.

# related work

Several previous authors have worked primarily in automated assessment in, for example, programming and marking of ER Diagrams, but there is limited evidence of successful automated assessment of SQL queries. The XData system [1] focuses on generating query specific test cases to catch common errors in queries, whereas Gradience [2] uses fixed query independent or manually constructed datasets in automated assessment of SQL queries. Both have issues in identifying all potential solutions for an SQL query, especially in finding solutions which are close but not one hundred percent correct, requiring a number of sample solutions to be provided for each SQL query.

Many other systems consider only the result of a query, disregarding the correctness of the query itself. SQL Tester [3] and ASQLAG [4] are two such systems. SQL Tester requires the submitted query result to be identical, and in row order to the reference answer (i.e., sample solution) table. ASQLAG would consider a result correct if both the reference table and query result table were empty (i.e., returned no rows) which in no way guarantees that the submitted query was correct or valid.

Further advances have been made by Chu et al. in the form of Cosette [5], a system that determines if queries are logically equivalent by converting them into logical expressions. The latest version requires all possible query answers be included in an answer key for comparison, significantly increasing the complexity of the system. Along with the advances by Chu et al., research by Rivas and Schwartz [6] into the use of neural networks has shown some improvements over both static and dynamic comparison systems, however information on how the system is implemented is limited, and the system is yet to be made publicly available.

While research has been abundant into the automation of SQL marking, much of it has remained theoretical, or the developed systems are now obsolete, discontinued or unlocatable.

# oracle query plan

When writing SQL queries, these are normally optimised by the DBMS to ensure that the final query ran by Oracle is efficient. The query is optimised based on several parameters, including expression transformation rules, query cost and statistics gained from previous executions of the query. Query plans are normally tree based.

In Oracle DBMS’s SQL Developer application, we are able to view the predicted query plan for an SQL query, in a tree-based representation [6]. For example, for the following SQL query, which is based on a golfing hotel scenario, the SQL query shown below in Fig. 1 would generate the query plan shown in Fig. 2.



Figure 1. SQL query to retrieve golfer and booking details

Graphical user interface, table

Description automatically generated

Figure 2. Oracle Query Plan for SQL query in Fig. 1

We can use this query plan to compare two queries if they are equivalent, i.e., they produce the same result even when some operations are different. For example, the query in Fig. 3 gives the same result as the query in Fig. 1, however the attributes are returned in a different order, and we use cartesian product and a restriction rather than using the join operation of SQL. We have also added the optional ASC keyword to the ORDER BY statement.



Figure 3. Modified SQL query

Generating the query plan for the SQL query in Fig. 3 produces the (partial) query plan shown in Fig. 4. Notice it is identical to the query plan shown in Fig. 2. Our proposed solution is therefore to be able to generate and use these query plans in automated assessment of student solutions against tutor provided sample solutions.

Graphical user interface, text, application, table

Description automatically generated

Figure 4. Oracle Query Plan for SQL query in figure 3

# Proposed Solution

In this paper we therefore outline the currently developed solution, which is implemented in python, whereby we have been able to successfully extract the predicted query plans from Oracle to enable direct comparison of a collection of sample SQL queries against student solutions.

In the previous section we saw how it is possible to generate an execution plan for an SQL query in the SQL Developer tool. However, we need to be able to extract this query plan from the Oracle DBMS directly into python. When Oracle query plans are generated, they are stored within a plan table [6] which can then be queried in the same way as any other table in the current schema. This can be done by prepending the SQL query with the EXPLAIN PLAN statement, for example, for the query in Fig. 1 we would write:



Figure 5. SQL Query to generate Query Plan

In the example, we also give the statement an identifier, e.g., Q2, so that in python we can then extract, into a Pandas data frame, the required query plan by the statement id.

By using Oracle’s query plan, which produces the predicted query plan for an ad-hoc SQL query, it allows us to compare queries where, for example, join operations, use of subqueries instead of joins, use of different comparison operators (e.g., LIKE instead of ‘=’), etc., are classified as correct and therefore allow us to identify correct student solutions without the requirement of having to produce all possible permutations of correct solutions.

For example, for the queries in Fig. 1 and Fig. 3, the data frames generated in python are shown below in Fig. 6. The two data frames are identical, showing that the highlighted statement at the bottom indicates that the queries match. Even though the queries are syntactically different, they are semantically the same and reducing both queries to their predicted query plan allows us to compare queries without having to provide multiple sample solutions.

Calendar

Description automatically generated

Figure 6. Pandas data frames for two semantically equivalent SQL queries

Our second examples show that we can also successfully compare some sub-queries with join queries. For example, the queries in Fig. 7 are a simplified version of the queries from figure 6, where we have created a semantically equivalent query using a sub-query and a join. Because the Oracle query plan recognizes that the sub query is more optimal if replaced by a join operation, we get the same query plan for both SQL queries, allowing us to match some SQL queries in this way.

Table

Description automatically generated

Figure 7. Comparing a Join and a subquery operation in SQL

There are currently some minor issues in how the query plan, for example, specifies table and attribute names when, for example, table aliases are used, which we are currently working on. Currently this information (the ‘projection’ column) is removed when comparing the queries to allow us to return a match as shown in figure 6.

The query in Fig. 7 can also be problematic, for example when using table aliases or specifying the ‘join’ where clause with different ordering then causes the match to fail, as illustrated in Fig. 8. We are currently investigating the use of natural language operations, e.g., Levenstein distance, to deal with some of these outstanding issues.

Table

Description automatically generated

Figure 8. Switching the order of the tables in the WHERE clause

# Conclusions and Future Work

This paper has shown some early-stage work in automated marking of SQL queries. Although it has demonstrated that the Oracle query plan can be used for matching queries which are semantically equivalent. However, it has illustrated that there are a number of current issues which need resolved, for example, whether the query plan includes full table names or includes aliases, or the ordering of the tables in the query expression. Current work is therefore investigating how these can be resolved to correctly match more types of queries.

We are also currently working on a project to investigate how machine learning techniques could be applied to enable marking of SQL queries which are partially correct, and how these techniques could be used to give partial scores for student solutions where they do not completely match the sample solution.

REFERENCES

[1] B. Chandra, M. Joseph, B. Radhakrishnan, S. Acharya, and S. Sudarshan, ‘Partial marking for automated grading of SQL queries’, *Proc. VLDB Endow.*, vol. 9, no. 13, pp. 1541–1544, Sep. 2016, doi: 10.14778/3007263.3007304.

[2] ‘Gradiance: The Gradiance Service for Database Systems’. https://www.gradiance.com/db.html (accessed Nov. 17, 2022).

[3] A. Kleerekoper and A. Schofield, ‘SQL Tester: An Online SQL Assessment Tool and Its Impact’, in *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, New York, NY, USA, 2018, pp. 87–92. doi: 10.1145/3197091.3197124.

[4] P. Singporn, P. Vichianroj, and A. Trongratsameethong, ‘ASQLAG - Automated SQL Assignment Grading System for Multiple DBMSs’, *J. Technol. Innov. Tert. Educ.*, vol. 1, no. 1, pp. 41–59, 2018, doi: 10.14456/jti.2018.4.

[5] S. Chu, D. Li, C. Wang, A. Cheung, and D. Suciu, ‘Demonstration of the Cosette Automated SQL Prover’, in *Proceedings of the 2017 ACM International Conference on Management of Data*, New York, NY, USA, 2017, pp. 1591–1594. doi: 10.1145/3035918.3058728.

[6] H. Baer *et al.*, ‘Explaining and Displaying Execution Plans’, *Oracle Help Center*. https://docs.oracle.com/en/database/oracle/oracle-database/19/tgsql/generating-and-displaying-execution-plans.html (accessed Dec. 07, 2022).