The XDa-TA System for Automated Grading of SQL Query Assignments

Amol Bhangdiya [#], Bikash Chandra, Biplab Kar [§], Bharath Radhakrishnan,
K. V. Maheshwara Reddy [#], Shetal Shah, S. Sudarshan
IIT Bombay
{amolb12, bikash, biplabkar11, bharathrk, kvmahesh12, shetals, sudarsha}@cse.iitb.ac.in

***Abstract*—Grading of student SQL queries is usually done by executing the query on sample datasets (which may be unable to catch many errors) and/or by manually comparing/checking a student query with the correct query (which can be tedious and error prone). In this demonstration we present the XDa-TA system which can be used by instructors and TAs for grading SQL query assignments automatically. Given one or more correct queries for an SQL assignment, the tool uses the XData system to automatically generate datasets that are designed specifically to catch common errors. The grading is then done by comparing the results of student queries with those of the correct queries against these generated datasets; instructors can optionally provide additional datasets for testing. The tool can also be used in a learning mode by students, where it can provide immediate feedback with hints explaining possible reasons for erroneous output. This tool could be of great value to instructors particularly, to instructors of MOOCs.**

I. Introduction

Grading SQL queries is usually done by executing the query on sample datasets and checking the query results and/or by reading the student query and checking it manually. Comparing query results on manually created datasets or fixed sets of datasets can miss many errors in queries; in particular subtle errors can only be caught by carefully designed datasets.

Manual reading and comparing of queries is tedious since students may write queries in a variety of different ways and the grader has to spend a lot of time to check if a query is correct. This grading method is prone to errors as graders are likely to miss subtle mistakes. For example when required to write the query Q below,

```
SELECT department.dept_name, I.name
FROM department LEFT OUTER JOIN (SELECT *
FROM instructor WHERE instructor.salary > 70000) I
ON (I.dept_name=department.dept_name);
```

a student may write the following query Q_s :

```
SELECT department.dept_name, instructor.name FROM
department LEFT OUTER JOIN instructor
ON (instructor.dept_name=department.dept_name)
WHERE instructor.salary > 70000;
```

which looks sufficiently similar that the grader may miss the difference. These queries are not equivalent, since they give different results if there is a department where all instructors have salary $\leq$ 70000.

The approach of comparing the results of queries on sample datasets is widely used, and systems such as Gradience [1] help in automating such testing. However unless the sample datasets are carefully constructed there is no guarantee that they will catch all or a good number of errors. Automated generation of datasets designed to catch errors in specific queries is therefore an area of importance. The XData system [2], [3], [4] developed at IIT Bombay, generates dataset(s) that can catch commonly occurring errors in a large class of SQL queries using an approach based on query mutations.

Most incorrect queries are small syntactic deviations from a correct query and can be thought of as mutants of the correct query. Specifically, a *mutation* is single syntactic correct change of the original query; and a *mutant* is the result of one of more mutations on the original query. Running the student queries against one or more datasets and checking, if the results match with correct query works well if the datasets are designed to catch errors/mutations. The key step in this approach is to generate datasets that catch these errors. A dataset *kills* a mutant if the original query and the mutant give different results on the dataset. A test suite consisting of multiple datasets kills a mutant if at least one of the datasets kills the mutant. Section II lists the class of mutations that our system catches.

Another approach to check queries is by checking equivalence; while this approach works for simple conjunctive queries it is not decidable in general. Sufficient conditions for equivalence can be checked using transformation rules in query optimization but they often fail to detect equivalence an correct queries as shown in [4].

In this demonstration we showcase the XDa-TA system for automated grading of SQL queries submitted by students. The instructor provides XDa-TA with the schema and optionally sample data to help generate realistic datasets. The instructor then provides correct queries for the assignment questions. One or more datasets are generated for each correct query. Students can then submit queries to the system. Once the submissions are made the instructor can grade the student queries using the tool. Comparison of results of correct queries with those of student queries is used to test if the student query has an error. The system can be used to give immediate feedback to students, as well as for automated grading.

* Work partially supported by a research grant from Tata Consultancy Services, India
[#] Currently working at SAP Labs, India
[§] Currently working at Oracle India Pvt. Ltd.

In addition to walking through the demonstration of the system this paper describes user interaction by students and instructors, practical issues in grading student SQL query assignments such as security, concurrent access, and integration with course management systems.

II. Test Data Generation

A key component of our grading tool is automated generation of test cases specific to each query. Our system handles a wide variety of SQL features including selections, joins, aggregates, subqueries and set operators. The test cases are designed to catch common errors made in writing SQL queries, which are modeled as query mutations. The type of mutations considered includes join type mutations (inner/outer), join condition mutations, selection condition mutations, aggregate operator mutations, group by attribute mutations, mutations in string patterns, like clause mutations, distinct clause mutations, subquery connective mutations and set operator mutations, amongst others. The data generation technique to kill each mutation is specific to the mutation type considered.

Details of data generation techniques in the initial version of the XData system can be found in [2]. Some extensions and experimental results on student queries are described in [3]. Further details of test data generation in the XData system including details of handling string predicates, constrained aggregation, subqueries, NULLs, disjunctions, set predicates, group by attribute mutation, join condition mutation and distinct mutation are described in [4]. Results of grading student assignments in an actual database course using XDa-TA versus TAs or fixed datasets are given in [3], [4] which show that XDa-TA significantly outperforms fixed datasets, as well as manual grading by TAs in terms of catching errors in student queries.

For every correct query Q, XData generates multiple datasets. The first dataset ensures a non-empty result for Q (which itself kills several mutations that would generate an empty result on that dataset). Each of the remaining datasets is targeted to kill one or more mutations of the query; i.e. on each dataset the given query returns a result that is different from those returned by each of the mutations targeted by that dataset. The number of possible mutations is very large, but the number of datasets generated to kill these mutations is relatively small.

To generate a particular dataset, XData generates a set of constraints, where each tuple in the target dataset is represented by a tuple of constraint variables. XData then invokes a constraint (SMT) solver, CVC3 [5], to solve the constraints; the solution defines a dataset on which the query is to be tested.

As an example, consider data generation for the query Q_s in Section I. To generate the first dataset XData adds CVC constraints to specify that both the instructor and department relation contain 1 tuple each and that the dept_name for both the tuples are the same. Constraints in CVC are also added to ensure that the tuple in the instructor table has salary $>$ 70000. These constraints ensure that the dataset generated will produce non-empty result on the query.

In order to kill the mutation of INNER JOIN to LEFT OUTER JOIN XData creates a dataset with constraints to ensure that department.dept_name does not match any value in instructor.dept_name along with the constraint that salary $>$ 70000. For the dataset generated using these constraints the INNER JOIN query produces an empty result while the LEFT OUTER JOIN query produces non empty results.

III. Grading tool

The XDa-TA grading tool supports the Learning Tools Interoperability (LTI) standard, allowing it to interface with course management systems such as Moodle or Blackboard. Users can login on the course management system and follow a link to XDa-TA. LTI integration allows user authentication and course information to be transmitted securely to XDa-TA. LTI also allows marks generated by XDa-TA to be uploaded to the course management system.

The initial setup of the XDa-TA tool involves linking of the tool with a course management system and creation of site authentication information required for LTI.

A. Instructor Mode

The instructor mode provides several features for creating, updating and grading assignments. The instructor can add database connection information to the tool by providing JDBC URLs along with database usernames and passwords. For every assignment the instructor can then choose which database instance to use for evaluating student SQL queries. Our system currently supports PostgreSQL but we are working on supporting Oracle and MySQL also.

The instructor first uploads the schema and optionally small sample tables, by providing SQL script files. The schema provides information about the tables - name and datatype of columns, columns that are nullable and those that are not, primary key constraints and foreign key constraints - that are required for dataset generation. The sample tables help in generating realistic values for the database. Without the sample tables the value of credits for a course may be 1000, when it should have been less than 8, or the course_id of a course in the Biology department can be 'abc', when it should have been something like 'BIO-301'. All `CREATE TABLE` statements in the SQL script file are modified to `CREATE TEMPORARY TABLE` so that the table remains live only in the session. More importantly, if multiple queries are executed from different database connections using the same database user_id they will not conflict with each other.

The instructor can then upload the assignment questions along with the correct SQL queries for the questions. For each correct query the tool generates datasets, as discussed in Section II. Each dataset is tagged with a label indicating what kind of mutation the dataset was designed to kill.

For some assignments it may be possible to write correct queries using several very different approaches. Datasets generated for a correct query are designed to kill mutations of

that query, but may or may not succeed in killing mutations of a different formulation of the query. It could also happen that a question set by the instructor is ambiguous and there are multiple ways of interpreting it. For these cases the instructor mode allows multiple correct queries to be uploaded. Datasets generated from all the correct queries are used while evaluating student queries. The instructor may choose whether datasets of all the queries need to be passed, or datasets generated for any query needs to be passed depending on the need. The instructor can optionally provide additional datasets for testing student queries.

A start time and end time can be set for an assignment. The instructor can set each SQL assignment as a graded assignment or a learning assignment. The instructor may also choose to ignore the presence of duplicate tuples in the result of a question. Student queries can be bulk-loaded from a file by the instructor if desired.

B. Student Mode

Students can login and attempt the assignments within the deadline set by the instructor. Queries written by students sometimes include a schema name (e.g. `schema1.table1`) which will not execute correctly on the test database. To catch this and other minor errors such as output columns in the wrong order, we provide a dataset that produces non empty results for the student query. When the student submits a SQL query, this dataset could be used to alert the student that the query submitted is incorrect and give a chance to correct it without any penalty.

The instructor may have marked some assignments as learning assignments. For these assignments once the student submits a query, the tool checks its correctness as shown in Section IV-A. Instead of assigning marks to students the interface displays the tag of the dataset for which the query fails (this can be done incrementally, one failed dataset at a time). It would make sense to order the datasets starting from the most common errors, for example start with selection conditions, followed by join conditions and join type errors etc. The student can use this feedback and correct the mistakes.

IV. Evaluating queries

Once the students have submitted the assignments the instructor can grade the student queries when desired.

A. Checking for correctness

Let $Q_{i,j}$ denote the j^{th} student's query submission for question i. As discussed in Section III-A for every question the instructor may provide multiple correct queries. Let $CQ_{i,m}$ denote the m^{th} correct query for question i and $D_{i,m,k}$ be the k^{th} dataset for the correct query $CQ_{i,m}$

To evaluate student queries for a given correct query $CQ_{i,m}$, for each corresponding dataset $D_{i,m,k}$, the tool first uploads the dataset to the database, creating appropriate tables. The tables created for this purpose are temporary tables whose visibility is limited for only a session, to ensure that there are no conflicts in case multiple student queries are being evaluated simultaneously.

Next to compare the result of each student query $Q_{i,j}$ with that obtained by the correct query, $CQ_{i,m}$, the tool executes an SQL query of the form

$(Q_{i,j}$ `EXCEPT ALL` $CQ_{i,m})$ `UNION` $(CQ_{i,m}$ `EXCEPT ALL` $Q_{i,j})$

on the temporary tables.

If the result of the above query is non-empty for any dataset $D_{i,m,k}$, the student query $Q_{i,j}$ is marked as incorrect. The datasets along with the tags indicating which errors they were designed to catch along with the expected result and actual results are made available to the students as shown in Figure 2.

If the results of the above query are empty for *all* datasets, query $Q_{i,j}$ is deemed correct for the purpose of grading; our system cannot in general guarantee its correctness, unless we are able to establish query equivalence. (As described in [4] we tried a sufficient condition for query equivalence, namely that both $CQ_{i,m}$ and $Q_{i,j}$ generate the same optimal query plan, but the results of experiments in [4] show that this approach is often unable to establish equivalence of correct queries.)

If the student query is able to pass all desired datasets full marks are awarded. However if for one or more datasets the student query fails, the query is incorrect and no marks are awarded. Once the assignments are evaluated the students can see the result of evaluation as shown in Figure 1.

Our approach for checking correctness of query relies on killing mutations of the correct query and not of the student query. As a result we may not catch erroneous student queries that have extra conditions, since these conditions were not taken into account for data generation. One way to deal with this is to generate datasets based on mutations of the student query as well and use these also in grading. Since this requires a lot of overhead including constraint generation, constraint solving etc. for all the student queries, we do not implement this currently.

Correctness of a query with respect to result ordering cannot be checked by comparing results since the sort order may be a result of a chosen plan rather than due to an ORDER BY clause. Comparing the list of ORDER BY columns in the student query and the correct query is not sufficient in general since ORDER BY clauses may be equivalent even if they differ in the list of attributes, due to functional dependencies, selections conditions that force an attribute to be single valued and join conditions that equate attributes. Checking for equivalence of ORDER BY clauses is an area of future work.

B. Security

We need to safeguard against student queries altering other tables or running DDL queries that may delete or create tables. Datasets used for evaluating student queries are populated in a schema called *Grading* using a database user $grader$. The query for evaluating correctness in the *Grading* schema is then run under a different database user context, $tester$. The tables created in the *Grading* schema are temporary tables that are visible only in a session. The user $tester$ has permission only to read the tables in the *Grading* schema, ensuring student

Assignment: 10

Question Number	Question Description	Your Answer	Status	Failed Test Cases
Q1	Find all department names along with the name of the instructors who have a salary of more than 7000	SELECT department.dept_name, instructor.name FROM department LEFT OUTER JOIN instructor ON (instructor.dept_name = department.dept_name) WHERE instructor.salary > 70000	Incorrect	Test Cases
Q2	Find the sum of total credits of students from Physics department	select sum(tot_cred) from student where dept_name = 'Physics'	Ok	

Fig. 1. Assignment evaluation

Question: 1

Your Answer : SELECT department.dept_name, instructor.name FROM department LEFT OUTER JOIN instructor ON (instructor.dept_name = department.dept_name) WHERE instructor.salary > 70000

Instructor's Answer : SELECT department.dept_name, I.name FROM department LEFT OUTER JOIN (SELECT * from instructor WHERE instructor.salary > 70000) I ON (I.dept_name=department.dept_name)

Status: Incorrect

Message: Your query failed to pass the datasets shown below.

DATASET TO KILL JOINS MUTATIONS

Your Result

dept_name	name
Music	Crick

Expected Result

dept_name	name
Music	Crick
Elec. Eng.	null

Hide Dataset

INSTRUCTOR

id	name	dept_name	salary
83821	Crick	Music	70000.25

DEPARTMENT

dept_name	building	budget
Music	Watson	70000.25
Elec. Eng.	Watson	70000.25

Fig. 2. Student feedback

queries do not modify the schema or data. Queries which involve UPDATE are transformed to select queries for testing as mentioned in [3]. Hence *tester* does not need any right to alter or create tables.

V. Demonstration

We will demonstrate the system in various scenarios using the University schema from [6] on the PostgreSQL database. First we will present the basic principles of our XData system including mutation testing, SQL mutants, mutation killing and test data generation. This will be followed by a description of how XDa-TA uses the generated test data for automated grading. The tables of the University database along with some sample values for the tables will be preloaded and users would be able to interact with the XDa-TA grading tool on a first-hand basis in the following manners

- **Instructor mode** : This part of the demo is aimed to provide the instructor experience to the users. Users can create their own questions and provide one or more correct SQL queries. The users will be able to generate datasets based on the queries they have provided and be able to see the datasets generated along with the targeted mutants for each dataset. Once we get a few submissions for those questions the submissions will be evaluated, and correct and incorrect submissions can be seen along with the failed datasets.
- **Student submission mode** : This part of the demo illustrates the student mode when submitting assignments. For this purpose a predefined set of assignment questions along with questions submitted in the first scenario can be used. Users will be provided feedback based on their SQL query passing the datasets provided.
- **Student interactive mode** : This part of the demo will enable the users to try the interactive student mode. A set of questions will be prepared for this and users would be able to submit their queries. The users would then get immediate feedback about the correctness of their queries and the failed cases.

An online demonstrataion of the XDa-TA system has been setup at http://www.cse.iitb.ac.in/infolab/xdata/demo.

VI. Conclusions

The XDa-TA tool has great potential for easing the life of database course instructors and teaching assistants. In case of a database course run as a MOOC, manual grading of SQL queries is not an option and a tool such as XDa-TA is of great importance. We have successfully used the grading tool in a UG database course at IIT Bombay to correct student queries. We plan to release XDa-TA in open source for use by course instructors. As with most automated grading, our tool currently provides full or no marks. How to do partial marking in a way that reflects how close the student query is to some correct query is an area of future work.

References

[1] "Gradience: The gradiance service for database systems," http://www.gradiance.com/db.html.

[2] S. Shah, S. Sudarshan, S. Kajbaje, S. Patidar, B. P. Gupta, and D. Vira, "Generating test data for killing SQL mutants: A constraint-based approach," in *ICDE*, 2011.

[3] B. Chandra, B. Chawda, S. Shah, S. Sudarshan, and A. Shah, "Extending XData to Kill SQL Query Mutants in the Wild," in *Proc. of the 6th Int'l Workshop on Testing Database Systems*, ser. DBTest '13, held in conjunction with ACM SIGMOD, 2013.

[4] B. Chandra, A. Bhangdiya, B. Chawda, B. Kar, K. V. M. Reddy, S. Shah, and S. Sudarshan", "Data Generation for Testing and Grading SQL Queries." *CoRR*, vol. abs/1411.6704, 2014.

[5] C. Barrett and C. Tinelli, "CVC3," in *Computer Aided Verification (CAV)*, 2007, pp. 298–302.

[6] A. Silberschatz, H. F. Korth, and S. Sudarshan, *Database System Concepts*, 6th ed. McGraw Hill, 2010.