Semantic Errors in SQL Queries: A Quite Complete List

Christian Goldberg, Stefan Brass
Martin-Luther-Universität Halle-Wittenberg
{goldberg,brass}@informatik.uni-halle.de

Abstract

We investigate classes of SQL queries which are syntactically correct, but certainly not intended, no matter for which task the query was written. For instance, queries that are contradictory, i.e. always return the empty set, are obviously not intended. Current database management systems, e.g. Oracle, execute such queries without any warning. In this paper, we try to give a complete list of such errors. Of course, questions like the satisfiability are in general undecidable, but a significant subset of SQL queries can actually be checked. This also applies to the other errors explained in this paper. We believe that future database management systems will perform such checks and that the generated warnings will help to develop code with fewer bugs in less time.

1 Introduction

Errors in SQL queries can be classified into syntactic errors and semantic errors. A syntactic error means that the entered character string is not a valid SQL query. In this case, any DBMS will print an error message because it cannot execute the query. Thus, the error is certainly detected and usually easy to correct.

A semantic error means that a legal SQL query was entered, but the query does not or not always produce the intended results, and is therefore incorrect for the given task. Semantic errors can be further classified into cases where the task must be known in order to detect that the query is incorrect, and cases where there is sufficient evidence that the query is incorrect no matter what the task is. Our focus in this paper is on this latter class.

For instance, consider the following query:

```
SELECT *
FROM   EMP
WHERE  JOB = 'CLERK' AND JOB = 'MANAGER'
```

This is a legal SQL query, and it is executed e.g. in the Oracle8i DBMS without any warning. However, the condition is actually inconsistent, so the query result will be always empty. Since nobody would use a database in order to get an always empty result, we can state that this query is incorrect without actually knowing what the task of the query was. Such cases do happen, e.g. in the last exam we analyzed, 18 out of 148 students wrote an inconsistent condition.

It is well known that the consistency of formulas is undecidable, and that this applies also to database queries. However, although the task is in general undecidable, many cases that occur in practice can be detected with relatively simple algorithms.

We believe that a tool for finding semantic errors in SQL-statements (like the program `lint` for "C") would be useful not only in teaching, but also in application software development. At least, a good error message could speed up the debugging process.

While our experience so far has only been with errors made by students, not professional programmers, most of the students will become programmers, and they will not immediately make fewer errors.

The main contribution of this paper is a list of semantic errors that represents years of experience while correcting hundreds of exams that contained SQL queries. However, we have also tried to explain the general principles from which these errors can be derived (as far as possible; see technical report [1]). Therefore, it is not simply by chance whether an error appears on our list, but the list has a certain degree of completeness. For a list of style check suggestions, see also [1].

The paper is structured by general reasons why SQL queries can be considered suspicious: Unnecessary complications (Section 2), inefficient formulations (Section 3), violations of standard patterns (Section 4), many duplicates (Section 5), and the possibility of runtime errors (Section 6). Related work is discussed in Section 7.

2 Unnecessary Complications

Of course, in general it is difficult to state that a syntactically correct query is semantically wrong if one does not know the task for which the query was written. However, queries can be considered as "probably not intended" when they are unnecessarily complicated. Suppose the user wrote a query Q, and there is an equivalent query Q' that is significantly simpler, and basically can be derived from Q by deleting certain parts. There might be several reasons why the user did not write Q', for example, the user knew that Q' is not a correct formulation of the task at hand (in this case Q is of course also not correct), or the user did not know that Q' is equivalent.

Actually, "equivalence" in the sense of requiring exactly the same query result in all database states would make the condition still too strict. First, we not only want to minimize the query, but also the query result. Furthermore, it is better to exclude certain unusual states when we require that the result of both queries (Q and Q') is the same. Therefore, we will require the equivalence only for states in which all relations are non-empty. It might even be possible to assume that all columns contain at least two different values.

Some types of errors produce many duplicates. More powerful query simplifications can be used if these duplicates are not considered as important for the equivalence (at least if the simpler query Q' produces less duplicates than the more difficult query Q).

Now we give a list of all cases in which a query can be obviously simplified under this slightly weakened notion of equivalence. In each of these cases, a warning should be given to the user.

2.1 Entire Query Unnecessary

- Error 1: Inconsistent conditions.

2.2 Unnecessarily Complicated SELECT Clause

- Error 2: Unnecessary duplicate elimination.
- Error 3: Constant output column.
- Error 4: Duplicate output column.

2.3 Unnecessary Complications in the FROM Clause

The next three errors are cases where tuple variables are declared under `FROM` that are not really necessary.

- Error 5: Unused tuple variables.
- Error 6: Unnecessary joins.

- Error 7: Tuple variables that are always identical.

2.4 Unnecessary Complications in the WHERE Clause

- Error 8: Implied, tautological or inconsistent subconditions.
- Error 9: Unnecessarily general comparison operator.
- Error 10: Unnecessary SELECT arguments in EXISTS-subqueries.
- Error 11: IN/EXISTS condition can be replaced by comparison.

2.5 Unnecessary Complications in Aggregation Functions

- Error 12: Unnecessary DISTINCT in aggregations.
- Error 13: Unnecessary argument of COUNT.

2.6 Unnecessary Complications in the GROUP BY Clause

- Error 14: GROUP BY with singleton groups.
- Error 15: GROUP BY with only a single group.
- Error 16: Unnecessary GROUP BY attributes.

2.7 Unnecessary Complications in the HAVING Clause

In the `HAVING`-clause, the same errors as in the `WHERE`-clause are possible. In addition, conditions that are possible under `WHERE` are better written there (see Error 18 below).

2.8 Unnecessary Complications in the ORDER BY Clause

- Error 17: Unnecessary ORDER BY terms.

3 Inefficient Formulations

Although SQL is a declarative language, the programmer should help the system to execute the query efficiently. Errors 2 and 12 (`DISTINCT` when no duplicates are possible) also fall in this category. However, in the following two cases the query does not get shorter by choosing the more efficient formulation.

- Error 18: Inefficient HAVING (conditions without aggregation function).
- Error 19: Inefficient UNION (instead of UNION ALL).

4 Violations of Standard Patterns

Another indicator for possible errors is the violation of standard patterns.

- Error 20: Missing join conditions.
- Error 21: Uncorrelated EXISTS-subqueries.
- Error 22: SELECT-clause of subquery uses no tuple variable from the subquery.

- Error 23: Conditions in the subquery that can be moved up.
- Error 24: Comparison between different domains.
- Error 25: Strange HAVING (without GROUP BY).
- Error 26: Wildcards without LIKE.

5 Duplicates

Query results that contain many duplicates are difficult to read. It is unlikely that such a query is really intended. Furthermore, duplicates are often an indication for another error, e.g. missing join conditions. Of course, if we could give a more specific warning, that would be preferable.

- Error 27: Many duplicates.

6 Possible Runtime Errors

In C programs, it sometimes happens that a NIL-pointer is dereferenced, and the program crashes. Actually, such runtime errors are also possible in SQL, and one should try to verify that they cannot occur. Since these problems depend on the database state, they are not easily found during testing.

- Error 28: Subqueries that must not return more than one tuple.
- Error 29: No indicator variable for arguments that might be null.
- Error 30: Difficult type conversions
- Error 31: Possible runtime errors in datatype functions.

7 Related Work

It seems that the general question of detecting semantic errors in SQL queries (as defined above) is new. However, the question is strongly related to the two fields of semantic query optimization and cooperative answering.

Semantic query optimization (see e.g. [2, 5]) also tries to find unnecessary complications in the query, but otherwise the goals are different. As far as we know, DB2 contains some semantic query optimzation, but prints no warning message if the optimizations are "too good to be true". Also the effort for query optimization must be amortized when the query is executed, whereas for error detection, we would be willing to spend more time. Finally, soft constraints (that can have exceptions) can be used for generating warnings about possible errors. but not for query optimization.

Our work is also related to the field of cooperative query answering (see, e.g., [4, 3]). However, the emphasis is there more on the dialogue between DBMS and user. As far as we know, a question like the possibility of runtime errors in the query is not asked. Also, there is usually a database state given, whereas we do not assume any particular state. For instance, the CoBase system would try to weaken the query condition if the query returns no answers. It would not notice that the condition is inconsistent and thus would not give a clear error message. However, the results obtained there might help to suggest corrections for a query that contains this type of semantic error.

Actually, Oracle's precompiler for Embedded SQL (Pro*C/C++) has an option for semantic checking, but this means only that it checks whether tables and columns exist and that the types match.

The SQL Tutor system described in [6] discovers semantic errors, too, but it has knowledge about the task that has to be solved (in form of a correct query). In contrast, our approach assumes no such knowledge, which makes it applicable also for software development, not only for teaching.

8 Conclusions

There is a large class of SQL queries that are syntactically correct, but nevertheless certainly not intended, no matter what the task of the query might be. One could expect that a good DBMS prints a warning for such queries, but, as far as we know, no DBMS does this yet.

Our goal is to develop a tool "`sqllint`" for finding semantic errors in SQL queries. The list of error types contained in this paper can serve as a specification of the task of this tool. We have algorithms for all of the error types (for a suitable SQL subset), but for space reasons, we could not present them here (see, however, the technical report [1]). The error checks can often be reduced to a consistency test. While the undecidability in the general case remains, we can at least use the mature methods of automated theorem proving. We also have simpler, direct sufficient conditions for some of the errors. The current state of the project is reported at:

`http://www.informatik.uni-halle.de/~brass/sqllint/`.

This page contains a prototype of the consistency test. Also some detailed evaluations of SQL exams (with statistics about the frequency of errors) are available on this page.

References

[1] Stefan Brass, Christian Goldberg. *Detecting Logical Errors in SQL Queries.* Technical Report, University of Halle, 2004.

[2] U. S. Chakravarthy, J. Grant, and J. Minker. Logic-based approach to semantic query optimization. *ACM Transactions on Database Systems*, 15:162–207, 1990.

[3] Wesley W. Chu, Hua Yang, Kuorong Chiang, Michael Minock, Gladys Chow and Chris Larson. Cobase: A scalable and extensible cooperative information system. *Journal of Intelligent Information Systems*, 1996.

[4] Terry Gaasterland, Parke Godfrey and Jack Minker. An Overview of Cooperative Answering. Journal of Intelligent Information Systems 21:2, 123–157, 1992.

[5] Chun-Nan Hsu and Craig A. Knoblock. Using inductive learning to generate rules for semantic query optimization. In *Advances in Knowledge Discovery and Data Mining*, pages 425–445. AAAI/MIT Press, 1996.

[6] A. Mitrovic. A knowledge-based teaching system for SQL. In *ED-MEDIA 98*, pages 1027–1032, 1998.

[7] C. Welty. Correcting user errors in SQL. *International Journal of Man-Machine Studies 22:4*, 463-477, 1985.