A Problem on Benchmarking SQL-analysis tools

We are currently facing a seemingly fundamental problem on benchamrking *one of the contributions* of the paper. Ideally, in this paper, we are going to present a new notion of application correctness (**ES**), and *emprically* show that it is sound compared to real-world application requirements and additionally find examples where **ES** catches anomalies that are arguably problematic but are not specifically asked by developers (e.g. maybe they forgot them when designing their application). In other words, we must show that ES is neither too strong nor too weak.

However, in reality, the problem is that there does not exist many real-world applications that use SQL queries (the input language in many other analysis techniques e.g. [3, 4]) directly. In other words, real database applications are usually implemented using high-level interfaces such as ORMs and do not directly use SQL. For example, Rubi on Rails applications, which offer a vast set of database-backed web applications (used for benchmarking in [1]) cannot be trivially turned into underlying SQL queries that they operationally make use of. As a result, analysis tools which work directly on SQL (such as ours), are forced to either:

1. Take real-world applications and *manually* export them to their version of SQL language (The approach in [4]). Using this approach we can compare ES to the ones that such applications specify and thus offer evidence for its usefulness. However, this naturally raises the question of soundness of the rewritten application, in addition to the tediousness of doing such a task manually.

2. Use (a relatively small number of) benchmark applications (such as TPC-C and another handful presented in [2]). These applications although are based on real use cases and their detailed explanations do exist, unfortunately, lack any notion of high-level correctness requirements and are written assuming serializability (with the exception of TPC-C). Taking this approach, we will be forced to manually extract such high-level invariants for the applications, which is not very satisfying, because we would not be able to strongly argue about the practicality of our results (maybe we deliberately didn't specify some invariants compared to ES, why should the reviewer believe us?)

Now, what I am looking for is an approach offering the best of both worlds: The set of benchmark applications should be large and ideally derived from

real-world code bases such as github and any manual transformation on them should be done minimally.

References

[1] P. Bailis, A. Fekete, M. J. Franklin, A. Ghodsi, J. M. Hellerstein, and I. Stoica. Feral concurrency control: An empirical investigation of modern application integrity. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 1327–1342, New York, NY, USA, 2015. ACM.

[2] D. E. Difallah, A. Pavlo, C. Curino, and P. Cudre-Mauroux. Oltp-bench: An extensible testbed for benchmarking relational databases. *Proc. VLDB Endow.*, 7(4):277–288, Dec. 2013.

[3] G. Kaki, K. Nagar, M. Najafzadeh, and S. Jagannathan. Alone together: Compositional reasoning and inference for weak isolation. *Proc. ACM Program. Lang.*, 2(POPL):27:1–27:34, Dec. 2017.

[4] Y. Wang, I. Dillig, S. K. Lahiri, and W. R. Cook. Verifying equivalence of database-driven applications. *Proc. ACM Program. Lang.*, 2(POPL):56:1–56:29, Dec. 2017.