Automated Testing of Database Engines

Frequently Asked Questions (Part 2)

> **Q1.** *The project description specifies that the reducer should expose the interface `./reducer --query <query-to-minimize> --test <oracle-script>`. Should `<query-to-minimize>` be a file containing the query (as with the benchmarks), or the query text itself?*

It is a file. `<query-to-minimize>` is the path to a `.sql` file containing the SQL script you want to minimize. For example:

```
reducer --query queries/query1/original_test.sql --test queries/query1/test.sql
```

> **Q2.** *The benchmarks suggest we still only consider SQLite 3.26.0 and 3.39.4. Is the reducer allowed to invoke these binaries directly, or should it interact only with the provided test script?*

The reducer must never call SQLite directly. Only the test script (the oracle) is allowed to run SQLite and decide whether the bug still triggers. Your reducer simply produces a candidate query and invokes the oracle script (e.g., by spawning it as a separate process). This keeps the reducer independent of any particular database engine or bug type.

> **Q3.** *I ran every benchmark query against its test script. `query14` is labelled `CRASH(3.26.0)`, but it does not crash: it terminates normally with exit code 1 and the message "Error: near line 534: database disk image is malformed". It is the only benchmark that behaves this way. Is it misclassified? (It does fail if the oracle is changed to DIFF.)*

The term CRASH is somewhat overloaded in this case. It means the query should reproduce the same unexpected behavior as the original, which, for `query14`, is the specific error message shown above rather than a hard crash.

> **Q4.** *I want to be sure I understand "token" correctly in "the quality of reduction is measured by the number of tokens in the SQL query." Would `SELECT * FROM t0` count as 3 tokens or 4 (counting the `*`)? I assume literals are counted too.*

You do not need to define "token" yourself, the evaluation is based on `sqlglot`'s tokenizer. Our evaluation script counts tokens as follows:

```
import sys
import sqlglot

file = sys.argv[1]
with open(file, "r") as f:
    query = f.read()

tokenizer = sqlglot.Tokenizer()
tokens = tokenizer.tokenize(query)
print(len(tokens))
```

We strongly recommend you use this exact script to compute token counts in your own evaluation, so that your reported numbers match ours. Whether * and literals are counted is therefore decided by `sqlglot`, not by a rule in this document.

> **Q5.** *The project description says the reducer should use "an arbitrary shell script that checks whether the minimized query still triggers the bug." Is this a hard requirement? Performance-wise it seems wasteful to spawn a subprocess instead of running the test directly inside the reducer.*

Yes, it is a hard requirement, and it is a deliberate design choice rather than an oversight.

Keeping the oracle in a separate script decouples it from the reducer: the reducer stays a generic, reusable tool that knows nothing about SQLite, while the script encapsulates everything specific to a given bug. Because the script can contain arbitrary code, you can express arbitrary oracle checks through it (e.g., a particular crash signature, a specific error message, a result mismatch between two engine versions, and so on) without ever changing the reducer. This is also the convention followed by established test-case reducers such as C-Reduce and Perses, which take the oracle (the "interestingness test") as an external script for exactly these reasons.

> **Q6.** *I'm not sure how the reducer should output its final result.*

The reducer modifies the provided query file in place. At every iteration, it overwrites the file passed via `--query` with the current candidate. When reduction finishes, that same file therefore contains the final, minimized query. This in-place behavior is consistent with existing test-case reducers such as C-Reduce.

> **Q7.** *I'm not sure what to expect for the `--test` input. Is it the path to a `.sh` script, or the script as plaintext? I assume `--query` is plaintext (please correct me if I'm wrong).*

Both `--test` and `--query` are filesystem paths, not inline content. `--test` points to the oracle script on disk, and `--query` points to the `.sql` file containing the query. Your reducer reads these files from the given locations.

> **Q8.** *When running inside a container, how does the reducer access the test scripts and queries? Should those files already be in the image (e.g., copied in via the Dockerfile, or mounted)? Will the image be modified to include the grading benchmarks?*

You do not need to copy the benchmarks or test scripts into your image. During grading, we mount the benchmarks and their oracle scripts into the container ourselves when the container is created. Your Dockerfile only needs to build and install the reducer itself at `/usr/bin/reducer` and the corresponding SQLite binaries.

> **Q9.** *Are we allowed to use the following?*
>
> - `Sqlparse`, which mainly separates tokens and little else.
> - `Sqlglot`, also a parser, but with extra functionality including an optimizer that performs simplifications. We currently use only its tokenizer for our reduction. Is it acceptable to use just the tokenizing and pretty-printing features and not the optimizer?
> - Parser-provided visitors, for example the `SQLiteParserBaseVisitor<TResult>` C# class generated by ANTLR4. Are we expected to walk the AST ourselves, or is this allowed?

You may use a parsing library, including its AST-traversal mechanisms (e.g., visitors) and its pretty-printing functionality. What you may not use is any functionality that optimizes or rewrites a query for you, e.g., `sqlglot`'s optimizer. In short: parsing, traversal, and pretty-printing are fine; query optimization is not, because the reduction logic must be your own work.

> **Q10.** *Within the reducer itself (not the test scripts), must the environment variable `TEST_CASE_LOCATION` point to a file that already exists, or may the reducer create that file before invoking the test script?*

It need not point to an existing file. Your reducer is free to choose the path, create (or overwrite) the file with the current candidate query, set `TEST_CASE_LOCATION` accordingly, and only then invoke the oracle script. The only requirement is that the file exists and contains the candidate query at the moment the oracle script runs (otherwise the oracle script would fail 🙂).