Bug Squashing with SQLsmith

andreas.seltenreich@credativ.de

October 25, 2018
Outline

Motivation

Testing Methodology

Analysis of Bugs Uncovered

Design

Future Work
Motivation: My Story

- Inspired by Csmith, a random C program generator
- While working on a C compiler, I learned that one can never have enough testing
  - Regression tests, unit tests and testbenches were all green
  - Csmith made assertions fail in my optimization phase
- Me in 2015: We need an SQLsmith!
- In total, it found:
  - 71 Bugs in PostgreSQL
  - 3 in SQLite
  - 50 in MonetDB
  - 6 in various libraries (even glibc!)
Motivation: Who’s it for?

- Developers of SQL speaking databases
- Extension writers
- Reviewers of submitted patches
- Security auditors
- Indirectly, all users profit from additional quality assurance
On Fuzz Testing: Bit-Level

Bit-Level fuzzers (e.g. AFL, libFuzzer)

- Only applicable when information density is very high
- Do not grasp high-level concepts such as syntax, schema, catalog, identifiers or scope
- Works ok for fuzzing Postgres’ regexp parser
- Need ages to find the first trivial syntactically correct query
- Need eons to find a hit in the catalog/schema
On Fuzz Testing: Domain-Aware

Domain-aware fuzzers (Csmith, SQLsmith)

- Generate syntactically and semantically valid input at high speed
- Still cannot interpret the result semantically
- Semantics may be verified indirectly
Prior work

- **CSmith by utah.edu (since 2011)**
  - Found over 400 bugs in various compilers
  - Finds bugs in optimizations, code generators, register allocators, etc. despite fuzzing the parser
  - BSD-style license

- **RAGS by Microsoft (conference paper from 1998)**
  - They implemented differential testing
  - Queries look similar to SQLsmith’s, albeit smaller
  - No code available
Running SQLsmith

- Just tell it the target database

$ sqlsmith --target="host=/tmp port=65432 dbname=regression"
$ sqlsmith --sqlite="file:~/.firefox/places.sqlite?mode=ro"
$ sqlsmith --monetdb="mapi:monetdb://localhost:50000/smith"

- Using --verbose, it prints a character for each query

<table>
<thead>
<tr>
<th>symbol</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>ok</td>
</tr>
<tr>
<td>S</td>
<td>syntax error</td>
</tr>
<tr>
<td>t</td>
<td>timeout</td>
</tr>
<tr>
<td>C</td>
<td>broken connection</td>
</tr>
<tr>
<td>e</td>
<td>other error</td>
</tr>
</tbody>
</table>
Advanced Options

--log-to=connstr  log errors to postgres database
--seed=int        seed RNG with specified int instead of PID
--dump-all-graphs dump generated ASTs
--dump-all-queries print queries as they are generated
--dry-run         print queries instead of executing them
--exclude-catalog don’t generate queries using catalog relations
--max-queries=long terminate after generating this many queries
--rng-state=string deserialize dumped rng state
How to Hunt Bugs

Watch out for symptoms like:

- Core dumping due to failed assertions, PANICs
- Outlandish error messages or warnings
  - Log them into a database to allow filtering
  - Analysis of historical data may also give insights
- Processes bloating, hogging CPU
  - Need to monitor system load to find these bugs
Nature of Bugs Found: Crashes

postgres=# select bit '1' >> (-2^-31)::int;

LOG: server process (PID 15838) was terminated by signal 11: Segmentation fault
LOG: terminating any other active server processes
LOG: database system was not properly shut down; automatic recovery in progress
LOG: redo is not required
LOG: database system is ready to accept connections
Datum bitshiftleft(PG_FUNCTION_ARGS)
{
    VarBit *arg = PG_GETARG_VARBIT_P(0);
    int32 shft = PG_GETARG_INT32(1);

    /* Negative shift is a shift to the right */
    if (shft < 0)
        PG_RETURN_DATUM(DirectFunctionCall2(
            bitshiftright,
            VarBitPGetDatum(arg),
            Int32GetDatum(-shft)));

    /* do bitshift left for positive arguments */
}
Nature of Bugs Found: PANICs

postgres=# update brintest
    set oidcol = coalesce(brintest.oidcol, pg_my_temp_schema()),
        timestamptzcol = clock_timestamp(), uuidcol = null
    returning brintest.byteacol;
WARNING: specified item offset is too large
PANIC: failed to add BRIN tuple
server closed the connection unexpectedly
Nature of Bugs Found: Failed Assertions

From: Andreas Seltenreich <seltenreich(at)gmx(dot)de>
To: pgsq1-hackers(at)postgresql(dot)org
Subject: [sqlsmith] Failed assertion in postgres_fdw/deparse.c:1116

Creating some foreign tables via postgres_fdw in the regression db of master as of de33af8, sqlsmith triggers the following assertion:

    TRAP: FailedAssertion("!(((((const Node*)(var))->type) == T_Var))", File: "deparse.c", Line: 1116)

gdb says var is holding a T_PlaceHolderVar instead.
Nature of Bugs Found: Internal ERRORS

ERROR: failed to build any 8-way joins
ERROR: could not devise a query plan for the given query
ERROR: plan should not reference subplan’s variable
ERROR: failed to assign all NestLoopParams to plan nodes
ERROR: could not find pathkey item to sort
ERROR: too late to create a new PlaceHolderInfo
Nature of Bugs Found: Other ERRORs

From: Andreas Seltenreich <seltenreich(at)gmx(dot)de>
To: pgsq1-hackers(at)postgresql(dot)org
Subject: [sqlsmith] Missing CHECK_FOR_INTERRUPTS in tsquery_rewrite

[...]
testing with sqlsmith yielded an uncancellable backend hogging CPU time.
[...]
select ts_rewrite(
(select string_agg(i::text, '&')::tsquery from generate_series(1,32) g(i)
(select string_agg(i::text, '&')::tsquery from generate_series(1,19) g(i)
'foo');
How to Hunt Bugs (cont.)

- Complicate DUT configuration (replication, non-default settings)
- Make interesting objects or values available to sqlsmith
  - Use a regression DB as a starting point
  - Add Foreign Tables
  - Have infinity, NaN, $2^{31}-1$, etc around in your database
- Use additional tools
  - low-memory/libfailmalloc
  - ASAN
  - valgrind
  - trap on division by zero
My Testing Rig

- Cluster of cheap surplus Sandy Bridge quad-cores in my apartment
- Ansible to put testing arrangements on machines
- `gdb` scripts to harvest backtraces from appearing coredumps
- `sinfod`
  - Broadcasts system load on the network
  - Yields a real-time view on the entire cluster load
  - Many failure modes are readily identifiable
### BUGs by Nature over Modules

<table>
<thead>
<tr>
<th></th>
<th>Plan</th>
<th>Exec</th>
<th>Access</th>
<th>TX</th>
<th>Oper</th>
<th>Contrib</th>
<th>ADT</th>
<th>∑</th>
</tr>
</thead>
<tbody>
<tr>
<td>Segfault</td>
<td>2</td>
<td>6</td>
<td>1</td>
<td>3</td>
<td>8</td>
<td>1</td>
<td>1</td>
<td>21</td>
</tr>
<tr>
<td>PANIC</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>TRAP</td>
<td>11</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td></td>
<td>25</td>
</tr>
<tr>
<td>ERROR</td>
<td>10</td>
<td>4</td>
<td></td>
<td></td>
<td>4</td>
<td>1</td>
<td></td>
<td>15</td>
</tr>
<tr>
<td>÷ 0</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>other</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td><strong>∑</strong></td>
<td>26</td>
<td>11</td>
<td>6</td>
<td>4</td>
<td>19</td>
<td>3</td>
<td>2</td>
<td>71</td>
</tr>
</tbody>
</table>

Regarding SQLite3, all three bugs were failed assertions in the planner and executor.
Test Coverage

```
src/postgres$ ./configure --enable-coverage
```

<table>
<thead>
<tr>
<th>test load</th>
<th>overall</th>
<th>parser</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqlsmith</td>
<td>39.8</td>
<td>30.3</td>
</tr>
<tr>
<td>make check</td>
<td>62</td>
<td>80.2</td>
</tr>
<tr>
<td>sqlsmith+make check</td>
<td>65.1</td>
<td>80.4</td>
</tr>
</tbody>
</table>

Numbers generated using sqlsmith commit 7ffac2d, running 4 instances w/25000 queries each. Postgres code for the analysis was from master branch at around the same time.
Project Goals for SQLsmith

Inspired by Csmith, the following goals were set

▶ Be product-agnostic
▶ No requirement for templates/user-provided grammar/etc
  \(\sim\) The language is the limit
▶ Deterministic generation for reproducability/benchmarking
▶ Speed: The bottleneck should always be the database under test (DUT)
Language Choice: C++11

- OO design well-suited for AST construction
- Absolute type safety
- Implicit memory management
- Standardized multi-threading
- Exceptions, also employed for backtracking in AST generation
- Speed
Product Abstraction

Two front-end classes provide product-agnostic access to the DUT

- Schema class
- DUT class

Implemented for:

- PostgreSQL 9.1 or later
- SQLite 3
- MonetDB (contributed by cwi.nl)
- Various forks on github
Auxiliary Modules

▶ Class logger
  ▶ Invoked for generation and result
  ▶ Implementations for
    ▶ logging to stderr (with primitive analysis)
    ▶ logging into a database (allows filtering)
    ▶ collecting statistics

▶ Impedance matching module
  ▶ Allows to adapt grammar to the DUT
  ▶ Productions consistently leading to errors are blacklisted
Grammar Production Class Hierarchy

- Base class `prod` for all grammar productions
- Instantiation yields a random object of the respective production
- Some members of `prod` subclasses are derived from `prod` as well, forming the AST
- Visitor design pattern allows walking this AST
- `operator«` emits SQL for a production
- Productions are instantiated speculatively
  - Constructors throw exception when there is no way to create a valid instance in the context
  - Backtracking to higher AST levels to get out of dead ends
So far we can generate syntactically correct queries.

Modelling of scope needed to generate semantically correct ones.

Scope class models column and relation visibility.

Production constructors take a scope object where they pick their references from.

Productions may create a different scope to pass to their members.
sqltype Class

- Type matching among columns, operators, functions
- Originally, SQLsmith did only consider type equality
  \[\sim\] turned out too primitive
- Now there's a method: `sqltype::consistent(sqltype*)`
- Schema classes may fill the scope with a derived `sqltype` to adapt the grammar to a product-specific type model
On Randomness

- Csmith uses a sophisticated stochastical model
- SQLsmith uses
  - dice throws when the grammar allows alternatives
  - $d_6(), d_{12}(), d_{20}()$ calls in factories
  - random_pick<>() from container when the schema/scope allows alternatives
- All of them use an instance of C++11's std::mt19937_64
Extending SQLsmith

- Add support for a new RDBMs
  - Implement a Schema and DUT class for your product
- Extend the grammar
  - Derive something from the `prod` class
  - Extend a factory to return instances
- Doxygen-documentation available to visualize the class hierarchy and their collaboration diagrams
Future Work: Differential Testing

Product-differential testing:

- Microsoft did it with RAGS. Summary:
  - pro: "output validation proved to be extremely useful"
  - con: "the common SQL subset is relatively small"
  - con: "proportionality issues are problematic"

- Further: Deterministic queries are not enough, also need deterministic results. E.g. ...join pg_stat_activity ... where t > CURRENT_TIME ...

Setting-differential testing:

- Repeat queries with idempotent GUC settings

Version-differential testing:

- Allow spotting regressions wrt. semantics
Future Work: Statement Simplification

- Generated statements are largish and it takes effort to reduce them to an often simple testcase
- This can be automated by cutting things from the AST while maintaining the failure mode
- Creduce is the solution for Csmith, implementing a SQLreduce is a natural step
- Postgres’ parser has been factored out for stand-alone use, that’s a good starting point
- Microsoft also did it for RAGS (no code available)
Future Work: Miscellaneous

- Multithreading
- Support more products
- Add more grammar productions
- Improve SQLsmith’s primitive type system
  - About 25% of the queries currently result in type errors
- Extend Postgres with a compiled regexp type to improve filtering performance
- Log SQLSTATE
  - Need to fix libpqxx or use libpq instead
Selected ERRORS of the Day:

▶ value for domain things violates check constraint "meow"
▶ link of phone to hub does not make sense
▶ time zone "Bruce Momjian" not recognized
▶ return type mismatch in function declared to return things
▶ dimension mismatch in "+" operation: "6 Gy", "173.505 kmol"
▶ Lost connection to MySQL server during query
Bibliography

AFL: http://lcamtuf.coredump.cx/afl/
Creduce: https://embed.cs.utah.edu/creduce/
Csmith: https://embed.cs.utah.edu/csmith/
  massive-stochastic-testing-of-sql/
SQLsmith: https://github.com/anse1/sqlsmith
SQLsmith score list maintained by users:
  https://github.com/anse1/sqlsmith/wiki
libFuzzer: http://llvm.org/docs/LibFuzzer.html
sinfod: http://www.ant.uni-bremen.de/whomes/rinas/
  sinfo/man_sinfod.html
TODO: github-link to factored-out parser