Detecting Logical Bugs of DBMS with Coverage-based Guidance

Yu Liang       Song Liu       Hong Hu
Memory Bugs in DBMS: Well Studied

- **Generation-based testing**
  - *SQLsmith, QAGen [SIGMOD’07], QGEN [VLDB’04] ...*

- **Mutation-based fuzzing**
  - *Squirrel [CCS’20], PolyGlot [Oakland’21], RATEL [ICSE-SEIP’21] ...*
Logical Bugs in DBMS: Limited Exploration

CREATE...  
SELECT...  

DBMS  

{  

incorrect results

| DISCARD TEMP results in "ERROR: cache lookup failed for type 0" |
| COLLATE nocase index on a WITHOUT ROWID table malfunctions |
| Title: Incorrect result on a table scan of a partial index |

| MariaDB Server / MDEV-21065 |
| UNIQUE constraint causes a query with string comparison to omit a row in the result set |

| Double negation causes incorrect result #15725 |
| Bug #95889 | Functional index seems to malfunction with UNSIGNED column |
Existing Works: Differential Testing

CREATE...
SELECT...

SQLite

MySQl

PostgreSQl

Result A

= ?

Result B

= ?

Result C
Existing Works: Differential Testing

- SQLite dialects
  - without rowid; fts5; ...
- PostgreSQL dialects
  - pg_catalog; integer[]; ...
- MySQL dialects
  - datetime; json_set(); ...

- Limited common syntax
- Low coverage
- Various dialects/features
- Low correctness rate (validity)
Existing Works: SQLancer

- Use oracles to find logical bugs
  - compare results from function-equivalent queries

- Cons: rely on rule-based query generator
  - limited to explore deep program logic

```
rule-based generator  ➔  SELECT...
                      ➔  DBMS
                      ➔  Results
                      ➔  oracles
                      ➔  SELECT...
                      ➔  = ?
                      ➔  Results
```
Contributions

• SQLRight: a general platform to test DBMS logical bugs
  o coverage-guided fuzzing
  o validity-oriented mutation
  o general interfaces for DBMS oracles
• Found 18 logical bugs in SQLite and MySQL
• https://github.com/psu-security-universe/sqlright
Motivating Example (SQLite)

```
CREATE TABLE person (pid INT);
INSERT INTO person VALUES (1), (10), (10);
CREATE UNIQUE INDEX idx ON person (pid) WHERE pid=1;
SELECT DISTINCT pid FROM person WHERE pid=10;
```
Challenges to Detect Logical Bugs

• Generating valid queries
  - invalid queries cannot trigger logical bugs
Challenges to Detect Logical Bugs

• Implementing DBMS oracles
  o no platform for easy oracle development
  o no easy integration with existing techniques
• SQLancer: non-trivial manual efforts
Validity-oriented Query Mutation

- Dedicated Parsing

```
SELECT...
```

[Diagram with steps:
- SELECT...
- DBMS Bison Parser
- DBMS Unique Structure
- Query Processing
- Translator
- SQLRight Parser
- SQLRight General IR
- Fuzzing Mutation]
Validity-oriented Query Mutation

- Dedicated Parsing

SELECT...

DBMS Bison Parser → DBMS Unique Structure → Query Processing

original grammar rule

SQLRight Parser → SQLRight General IR → Fuzzing Mutation
Validity-oriented Query Mutation

CREATE TABLE x ( x INT, x INT, x INT);
INSERT INTO x VALUES (x), (x), (x);
ALTER TABLE x RENAME TO x;
SELECT x FROM x WHERE x = x;

- Context-based IR Instantiation
  - fill in concrete query operands
Validity-oriented Query Mutation

```sql
CREATE TABLE v0 ( c1 INT, c2 INT, c3 INT);
INSERT INTO v0 VALUES (0), (10), (10);
ALTER TABLE v0 RENAME c3 TO c4;
SELECT * FROM v0 WHERE c1 = c4;
```

- Context-based IR Instantiation
  - fill in concrete query operands

**SOLUTION 1**
- rename c3 to c4
- use c4 not c3
Validity-oriented Query Mutation

- Two other techniques (details in paper)
  - cooperative mutation
  - non-deterministic behaviors removal
General Interfaces for DBMS Oracles

- Easy development for new oracles
- Four general APIs
General Interfaces for DBMS Oracles

- remove improper queries

CREATE TABLE person (pid INT);
INSERT INTO person VALUES (1), (10), (10);
CREATE UNIQUE INDEX idx ON person (pid) WHERE pid=1;
INSERT INTO person VALUES (RANDOM());

random results
General Interfaces for DBMS Oracles

- remove improper queries

```sql
CREATE TABLE person (pid INT);
INSERT INTO person VALUES (1), (10), (10);
CREATE UNIQUE INDEX idx ON person (pid) WHERE pid=1;
INSERT INTO person VALUES (RANDOM());
```
General Interfaces for DBMS Oracles

- append oracle-compatible SELECT statements

CREATE TABLE person (pid INT);
INSERT INTO person VALUES (1), (10), (10);
CREATE UNIQUE INDEX idx ON person (pid) WHERE pid=1;
INSERT INTO person VALUES (RANDOM());
SELECT DISTINCT COUNT(*) FROM person WHERE pid=10;
General Interfaces for DBMS Oracles

- transform query to functional equivalent forms

```
CREATE TABLE person (pid INT);
INSERT INTO person VALUES (1), (10), (10);
CREATE UNIQUE INDEX idx ON person (pid) WHERE pid=1;
INSERT INTO person VALUES (RANDOM());
SELECT DISTINCT COUNT(*) FROM person WHERE pid=10;
SELECT DISTINCT pid=10 FROM person;
```
SELECT DISTINCT COUNT(*) FROM person WHERE pid=10;
SELECT DISTINCT pid=10 FROM person;

○ comparison method to identify unexpected result

CREATE TABLE person (pid INT);
INSERT INTO person VALUES (1), (10), (10);
CREATE UNIQUE INDEX idx ON person (pid) WHERE pid=1;
INSERT INTO person VALUES (RANDOM());

SELECT DISTINCT COUNT(*) FROM person WHERE pid=10;
SELECT DISTINCT pid=10 FROM person;

matched
res: {1}
res: {0, 1}
General Interfaces for DBMS Oracles

- comparison method to identify unexpected result

CREATE TABLE person (pid INT);
INSERT INTO person VALUES (1), (10), (10);
CREATE UNIQUE INDEX idx ON person (pid) WHERE pid=1;
INSERT INTO person VALUES (RANDOM());

SELECT DISTINCT COUNT(*) FROM person WHERE pid=10;
SELECT DISTINCT pid=10 FROM person;

---

not matched

res: {2}
res: {0, 1}
SQLRight Overview

- Coverage-guided fuzzer
- Validity-oriented mutation
- General interfaces for oracles
Evaluation

- Can SQLRight detect real-world logical bugs?
- Can SQLRight find more bugs than existing tools?
- Contribution of different SQLRight components?
Detect Real-world Logical bugs

<table>
<thead>
<tr>
<th>DBMS</th>
<th>SQLITE</th>
<th>MYSQL</th>
<th>PostgreSQL</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NoREC</td>
<td>11</td>
<td>3</td>
<td>0</td>
<td>14</td>
</tr>
<tr>
<td>TLP</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>ROWID</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>INDEX</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>TOTAL</td>
<td>14</td>
<td>4</td>
<td>0</td>
<td>18</td>
</tr>
</tbody>
</table>

- **18 logical bugs**
  - 14 SQLite
  - 4 MySQL
- **15 bugs fixed**
- **2 from new oracles**
Comparison regarding **Detected Bugs** (NoREC)

- **SQLite**

![Graph showing bug detection comparison for SQLite, SQLRight, Squirrel + oracle, and SQLancer.](image)

- **SQLRight**: 6 bugs
- **Squirrel + oracle**: 1 bug
- **SQLancer**: 0 bug

Squirrel: mem corrupt detector
Comparison regarding **Branch Coverage** (NoREC)

- SQLite

**Chart:**
- SQLRight: 40.6K
- Squirrel_{oracle}: 35.9K
- SQLancer: 34.2K

12.97% more coverage than best baseline
Comparison regarding **Query Validity** (NoREC)

- **SQLite**

![Graph showing Query Validity comparison]

- SQLRight: 40.0%
- Squirrel+_oracle: 32.0%
- SQLancer: 92.0% rule-based
Contribution of *Coverage Feedback* (NoREC)

- **SQLite**

![Graph showing bug contributions for SQLite](image)

- **SQLRight**: 4 bugs
- **SQLRight\_drop**: 2 bugs
- **SQLRight\_save**: 2 bugs
- **SQLRight\_rand**: 1 bug
Contribution of **Validity** (NoREC)

- **SQLite**

![Graph showing bug contributions for SQLite databases with different configurations.](image)

- SQLRight: 4 bugs
- SQLRight\_ctx\_valid: 1 bug
- Squirrel\_oracle: 1 bug
- SQLRight\_db\_par\&ctx\_valid: 0 bug
More Evaluations in the Paper

NoREC

TLP
Conclusion

• SQLRight: a general platform to test DBMS logical bugs
  o coverage-guided fuzzing
  o validity-oriented mutation
  o general interfaces for DBMS oracles
• Found 18 logical bugs in SQLite and MySQL
• https://github.com/psu-security-universe/sqlright
Thank You

Question?

yuliang@psu.edu