Testing Database Engines via Pivoted Query Synthesis

Manuel Rigger        Zhendong Su
ETH Zurich, Switzerland

11/05/2020

@RiggerManuel @ast_eth

https://people.inf.ethz.ch/suz/
Database Management Systems (DBMSs)
“it is seems likely that there are over one trillion (1e12) SQLite databases in active use”

https://www.sqlite.org/mostdeployed.html
Database Management Systems (DBMSs)

We found 96 unique bugs in these DBMSs, 78 of which were fixed!
Goal: Find Logic Bugs

**Logic bugs:** DBMS returns an incorrect result set
Example: SQLite3 Bug

```
CREATE TABLE t0(c0);
CREATE INDEX i0 ON t0(1) WHERE c0 NOT NULL;
INSERT INTO t0 (c0) VALUES (0), (1), (2), (NULL);
SELECT c0 FROM t0 WHERE t0.c0 IS NOT 1;
```
Example: SQLite3 Bug

```
CREATE TABLE t0(c0);
CREATE INDEX i0 ON t0(1) WHERE c0 NOT NULL;
INSERT INTO t0 (c0) VALUES (0), (1), (2), (NULL);
SELECT c0 FROM t0 WHERE t0.c0 IS NOT 1;
```

<table>
<thead>
<tr>
<th>c0</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>NULL</th>
</tr>
</thead>
</table>

**IS NOT** is a “null-safe” comparison operator
Example: SQLite3 Bug

CREATE TABLE t0(c0);
CREATE INDEX i0 ON t0(1) WHERE c0 NOT NULL;
INSERT INTO t0 (c0) VALUES (0), (1), (2), (NULL);
SELECT c0 FROM t0 WHERE t0.c0 IS NOT 1;
Example: SQLite3 Bug

```
CREATE TABLE t0(c0);
CREATE INDEX i0 ON t0(1) WHERE c0 NOT NULL;
INSERT INTO t0 (c0) VALUES (0), (1), (2), (NULL);
SELECT c0 FROM t0 WHERE t0.c0 IS NOT 1;
```

```
<table>
<thead>
<tr>
<th>c0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>NULL</td>
</tr>
</tbody>
</table>
```

TRUE
Example: SQLite3 Bug

CREATE TABLE t0(c0);
CREATE INDEX i0 ON t0(1) WHERE c0 NOT NULL;
INSERT INTO t0 (c0) VALUES (0), (1), (2), (NULL);
SELECT c0 FROM t0 WHERE t0.c0 IS NOT 1;

<table>
<thead>
<tr>
<th>c0</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>NULL</td>
</tr>
</tbody>
</table>
Example: SQLite3 Bug

```
CREATE TABLE t0(c0);
CREATE INDEX i0 ON t0(1) WHERE c0 NOT NULL;
INSERT INTO t0 (c0) VALUES (0), (1), (2), (NULL);
SELECT c0 FROM t0 WHERE t0.c0 IS NOT 1;
```

<table>
<thead>
<tr>
<th>c0</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>NULL</th>
</tr>
</thead>
</table>

**Result:**

- SELECT: 0, 1, 2
- WHERE: 0, 1
- FALSE

https://sqlite.org/src/tktview/80256748471a01
Example: SQLite3 Bug

```sql
CREATE TABLE t0(c0);
CREATE INDEX i0 ON t0(1) WHERE c0 NOT NULL;
INSERT INTO t0 (c0) VALUES (0), (1), (2), (NULL);
SELECT c0 FROM t0 WHERE t0.c0 IS NOT 1;
```

![Diagram showing the table t0 with values 0, 1, 2, and NULL, and the result of the query]

https://sqlite.org/src/tktview/80256748471a01
Example: SQLite3 Bug

```sql
CREATE TABLE t0(c0);
CREATE INDEX i0 ON t0(1) WHERE c0 NOT NULL;
INSERT INTO t0 (c0) VALUES (0), (1), (2), (NULL);
SELECT c0 FROM t0 WHERE t0.c0 IS NOT 1;
```

`t0`

<table>
<thead>
<tr>
<th>c0</th>
<th>0</th>
<th>1</th>
<th>NULL</th>
</tr>
</thead>
</table>

SQLite

```
0
2
NULL
```

TRUE

https://sqlite.org/src/tktview/80256748471a01
Example: SQLite3 Bug

```
CREATE TABLE t0(c0);
CREATE INDEX i0 ON t0(1) WHERE c0 NOT NULL;
INSERT INTO t0 (c0) VALUES (0), (1), (2), (NULL);
SELECT c0 FROM t0 WHERE t0.c0 IS NOT 1;
```

**Table t0**

<table>
<thead>
<tr>
<th>c0</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>NULL</td>
<td></td>
</tr>
</tbody>
</table>

**Diagram**

```
0
2
NULL
```

**Output**

```
0
2
NULL
```

---

https://sqlite.org/src/tktview/80256748471a01
Example: SQLite3 Bug

```
CREATE TABLE t0(c0);
CREATE INDEX i0 ON t0(1) WHERE c0 NOT NULL;
INSERT INTO t0 (c0) VALUES (0), (1), (2), (NULL);
SELECT c0 FROM t0 WHERE t0.c0 IS NOT 1;
```

```
<table>
<thead>
<tr>
<th>c0</th>
<th>Null</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NULL</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
```

SQLite

TRUE

https://sqlite.org/src/tktview/80256748471a01
Example: SQLite3 Bug

```
CREATE TABLE t0(c0);
CREATE INDEX i0 ON t0(1) WHERE c0 NOT NULL;
INSERT INTO t0 (c0) VALUES (0), (1), (2), (NULL);
SELECT c0 FROM t0 WHERE t0.c0 IS NOT 1;
```

NULL was not contained in the result set!

https://sqlite.org/src/tktview/80256748471a01
Background: Differential Testing

SELECT c0 FROM t0
WHERE t0.c0 IS NOT 1;

Background: Differential Testing

SELECT c0 FROM t0
WHERE t0.c0 IS NOT 1;

Background: Differential Testing

SELECT c0 FROM t0
WHERE t0.c0 IS NOT 1;

Check that all DBMSs compute the same result \((RS_1 = RS_2 = RS_3)\)

Background: Differential Testing

PostgreSQL

MySQL

SQLite

RS₁

RS₂

RS₃
Background: Differential Testing

PostgreSQL

MySQL

SQLite

RS₁

RS₂

RS₃
Background: Differential Testing

- PostgreSQL
- MySQL
- SQLite

Syntax error

\{0, 2\}
Background: Differential Testing

CREATE TABLE t0(c0);
CREATE INDEX i0 ON t0(1) WHERE c0 NOT NULL;
INSERT INTO t0 (c0) VALUES (0), (1), (2), (3), (NULL);
SELECT c0 FROM t0 WHERE t0.c0 IS NOT 1;
CREATE TABLE t0(c0);
CREATE INDEX i0 ON t0(1) WHERE c0 NOT NULL;
INSERT INTO t0 (c0) VALUES (0), (1), (2), (3), (NULL);
SELECT c0 FROM t0 WHERE t0.c0 IS NOT 1;

MySQL and PostgreSQL require a data type definition
Background: Differential Testing

```sql
CREATE TABLE t0(c0);
CREATE INDEX i0 ON t0(1) WHERE c0 NOT NULL;
INSERT INTO t0 (c0) VALUES (0), (1), (2), (3), (NULL);
SELECT c0 FROM t0 WHERE t0.c0 IS NOT 1;
```

PostgreSQL provides an IS DISTINCT FROM operator, and MySQL a <=> null-safe comparison operator.
Idea: PQS

Pivoted Query Synthesis (PQS): *Divide-and-conquer* approach for testing DBMSs
**PQS Idea**

```
CREATE TABLE t0(c0);
CREATE INDEX i0 ON t0(1) WHERE c0 NOT NULL;
INSERT INTO t0 (c0) VALUES (0), (1), (2), (3), (NULL);
SELECT c0 FROM t0 WHERE t0.c0 IS NOT 1;
```

<table>
<thead>
<tr>
<th>c0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>NULL</td>
</tr>
</tbody>
</table>

Validate the result set based on one randomly-selected row.
PQS Idea

CREATE TABLE t0(c0);
CREATE INDEX i0 ON t0(1) WHERE c0 NOT NULL;
INSERT INTO t0 (c0) VALUES (0), (1), (2), (3), (NULL);
SELECT c0 FROM t0 WHERE t0.c0 IS NOT 1;

Validate the result set based on one randomly-selected row
PQS Idea

Create a query that is guaranteed to at least fetch the pivot row.
PQS Idea

CREATE TABLE t0(c0);
CREATE INDEX i0 ON t0(1) WHERE c0 NOT NULL;
INSERT INTO t0 (c0) VALUES (0), (1), (2), (3), (NULL);
SELECT c0 FROM t0 WHERE t0.c0 IS NOT 1;

If the pivot row is missing from the result set a bug has been detected.
Approach

Randomly generate database → Select pivot row → Generate query for the pivot row → Validate that the pivot row is contained
Approach

Randomly generate database → Select pivot row → Generate query for the pivot row → Validate that the pivot row is contained

CREATE TABLE t0(c0);
CREATE INDEX i0 ON t0(1) WHERE c0 NOT NULL;
INSERT INTO t0 (c0) VALUES (0), (1), (2), (3), (NULL);

Statements are **heuristically generated** based on the DBMS’ SQL dialect
Approach

Randomly generate database → Select pivot row → Generate query for the pivot row → Validate that the pivot row is contained

One random row from multiple tables and views
Approach

Randomly generate database

Select pivot row

Generate query for the pivot row

Validate that the pivot row is contained

\[
\text{SELECT } c0 \text{ FROM } t0 \\
\text{WHERE }
\]

Generate \textit{predicates} that \textbf{evaluate to TRUE} for the pivot row and use them in JOIN and WHERE clauses
Random Expression Generation

- Randomly generate database
- Select pivot row
- Generate query for the pivot row
- Validate that the pivot row is contained

```
t0.c0 IS NOT 1;
```
Random Expression Generation

Randomly generate database

Select pivot row

Generate query for the pivot row

Validate that the pivot row is contained

\[ t0.c0 \text{ IS NOT } 1; \]

We implemented an expression evaluator for each node
Random Expression Generation

1. Randomly generate database
2. Select pivot row
3. Generate query for the pivot row
4. Validate that the pivot row is contained

**Evaluate** the tree based on the **pivot row**

```
IS
NOT
```

```
t0.c0 1
```

```
t0
c0
0
1
2
NULL
```
Random Expression Generation

- Randomly generate database
- Select pivot row
- Generate query for the pivot row
- Validate that the pivot row is contained

**Column references** return the values from the **pivot row**

```
<table>
<thead>
<tr>
<th>c0</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>NULL</th>
</tr>
</thead>
<tbody>
<tr>
<td>t0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
IS NOT
```
```
t0.c0 1
```
Random Expression Generation

Randomly generate database -> Select pivot row -> Generate query for the pivot row -> Validate that the pivot row is contained

Column references return the values from the pivot row

<table>
<thead>
<tr>
<th>t0</th>
<th>c0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>NULL</td>
</tr>
</tbody>
</table>
Random Expression Generation

Randomly generate database → Select pivot row → Generate query for the pivot row → Validate that the pivot row is contained

Constant nodes return their assigned literal values

- t0
  - c0
    - 0
    - 1
    - 2
    - NULL

- t0.c0
  - NULL
  - 1
Random Expression Generation

Randomly generate database

Select pivot row

Generate query for the pivot row

Validate that the pivot row is contained

Constant nodes return their assigned literal values

Randomly generate database

Select pivot row

Generate query for the pivot row

Validate that the pivot row is contained

Constant nodes return their assigned literal values

Randomly generate database

Select pivot row

Generate query for the pivot row

Validate that the pivot row is contained

Constant nodes return their assigned literal values

Randomly generate database

Select pivot row

Generate query for the pivot row

Validate that the pivot row is contained

Constant nodes return their assigned literal values
Random Expression Generation

Randomly generate database → Select pivot row → Generate query for the pivot row → Validate that the pivot row is contained

**Compound nodes compute their result based on their children**

**Diagram:**
- **t0**
  - c0
    - 0
    - 1
    - 2
    - NULL
- **IS NOT**
- **t0.c0** → 1
- NULL → 1
Random Expression Generation

Randomly generate database → Select pivot row → Generate query for the pivot row → Validate that the pivot row is contained

**Compound nodes compute their result based on their children**

- **t0**
  - **c0**
  - 0
  - 1
  - 2
  - NULL

**Diagrams:**
- Diagram showing the process flow.
- Diagram illustrating compound nodes and their evaluation.
Query Synthesis

Randomly generate database → Select pivot row → Generate query for the pivot row → Validate that the pivot row is contained

```
SELECT c0 c0 FROM t0
WHERE t0.c0 IS NOT 1;
```
Query Synthesis

Randomly generate database → Select pivot row → Generate query for the pivot row → Validate that the pivot row is contained

```
SELECT c0 c0 FROM t0
WHERE t0.c0 IS NOT 1;
```

What if the expression does not evaluate to TRUE?
Random Expression Rectification

```java
switch (result) {
    case TRUE:
        result = randexpr;
    case FALSE:
        result = NOT randexpr;
    case NULL:
        result = randexpr IS NULL;
}
```
Random Expression Rectification

Randomly generate database

Select pivot row

Generate query for the pivot row

Validate that the pivot row is contained

\[
\text{switch (result) } \{
\text{\hspace{1em} case TRUE:} \\
\hspace{2em} \text{result} = \text{randexpr}; \\
\text{\hspace{1em} case FALSE:} \\
\hspace{2em} \text{result} = \text{NOT randexpr}; \\
\text{\hspace{1em} case NULL:} \\
\hspace{2em} \text{result} = \text{randexpr IS NULL}; \\
\}\n\]

Alternatively, we could validate that the pivot row is expectedly not fetched.

Randomly generate database

Select pivot row

Generate query for the pivot row

Validate that the pivot row is contained

\[
\text{switch (result) } \{
\text{\hspace{1em} case TRUE:} \\
\hspace{2em} \text{result} = \text{randexpr}; \\
\text{\hspace{1em} case FALSE:} \\
\hspace{2em} \text{result} = \text{NOT randexpr}; \\
\text{\hspace{1em} case NULL:} \\
\hspace{2em} \text{result} = \text{randexpr IS NULL}; \\
\}\n\]

Alternatively, we could validate that the pivot row is expectedly not fetched.
Random Expression Rectification

Randomly generate database

Select pivot row

Generate query for the pivot row

Validate that the pivot row is contained

• DISTINCT clauses
• ORDER BY clauses
• DBMS-specific clauses (e.g., FOR UPDATE)
Approach

Randomly generate database → Select pivot row → Generate query for the pivot row → Validate that the pivot row is contained

**SELECT (NULL) INTERSECT**

**SELECT c0 FROM t0 WHERE NULL IS NOT 1;**

Rely on the DBMS to check whether the row is contained
Approach

Randomly generate database → Select pivot row → Generate query for the pivot row → Validate that the pivot row is contained
Approach

1. Randomly generate database
2. Select pivot row
3. Generate query for the pivot row
4. Validate that the pivot row is contained
Approach

Randomly generate database → Select pivot row → Generate query for the pivot row → Validate that the pivot row is contained

We generate 100,000 queries for each generated database
Implementation

https://github.com/sqlancer
## Bugs Overview

<table>
<thead>
<tr>
<th>DBMS</th>
<th>Fixed</th>
<th>Verified</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLite</td>
<td>64</td>
<td>0</td>
</tr>
<tr>
<td>MySQL</td>
<td>17</td>
<td>7</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>
## Bugs Overview

<table>
<thead>
<tr>
<th>DBMS</th>
<th>Fixed</th>
<th>Verified</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLite</td>
<td>64</td>
<td>0</td>
</tr>
<tr>
<td>MySQL</td>
<td>17</td>
<td>7</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>

**96 bugs** were unique, previously unknown ones.
Bugs Overview

<table>
<thead>
<tr>
<th>DBMS</th>
<th>Fixed</th>
<th>Verified</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLite</td>
<td>64</td>
<td>0</td>
</tr>
<tr>
<td>MySQL</td>
<td>17</td>
<td>7</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>

The SQLite developers **quickly responded** to all our bug reports → we focused on this DBMS.
Oracles

<table>
<thead>
<tr>
<th>DBMS</th>
<th>Logic</th>
<th>Error</th>
<th>Crash</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLite</td>
<td>46</td>
<td>17</td>
<td>2</td>
</tr>
<tr>
<td>MySQL</td>
<td>14</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>1</td>
<td>7</td>
<td>1</td>
</tr>
</tbody>
</table>

61 were logic bugs
Example: SQLite

CREATE TABLE t1(c1, c2, c3, c4, PRIMARY KEY (c4, c3));
INSERT INTO t1(c3) VALUES (0), (0), (0), (0), (0), (0), (0), (0), (0), (0), (NULL), (1), (0);
UPDATE t1 SET c2 = 0;
INSERT INTO t1(c1) VALUES (0), (0), (NULL), (0), (0);
ANALYZE t1;
UPDATE t1 SET c3 = 1;
SELECT DISTINCT * FROM t1 WHERE t1.c3 = 1;

https://www.sqlite.org/src/tktview?name=ced41c7c7d
Example: SQLite

```sql
CREATE TABLE t1(c1, c2, c3, c4, PRIMARY KEY (c4, c3));
INSERT INTO t1(c3) VALUES (0), (0), (0), (0), (0), (0), (0), (0), (0), (0), (NULL), (1), (0);
UPDATE t1 SET c2 = 0;
INSERT INTO t1(c1) VALUES (0), (0), (NULL), (0), (0);
ANALYZE t1;
UPDATE t1 SET c3 = 1;
SELECT DISTINCT * FROM t1 WHERE t1.c3 = 1;
```

ANALYZE gathers **statistics about tables**, which are then used for query planning.

https://www.sqlite.org/src/tktview?name=ced41c7c7d
Example: SQLite

```sql
CREATE TABLE t1(c1, c2, c3, c4, PRIMARY KEY (c4, c3));
INSERT INTO t1(c3) VALUES (0), (0), (0), (0), (0), (0), (0), (0), (0), (0), (NULL), (1), (0);
UPDATE t1 SET c2 = 0;
INSERT INTO t1(c1) VALUES (0), (0), (NULL), (0), (0);
ANALYZE t1;
UPDATE t1 SET c3 = 1;
SELECT DISTINCT * FROM t1 WHERE t1.c3 = 1;
```

<table>
<thead>
<tr>
<th>NULL</th>
<th>0</th>
<th>1</th>
<th>NULL</th>
</tr>
</thead>
</table>

https://www.sqlite.org/src/tktview?name=ced41c7c7d
Example: SQLite

CREATE TABLE t1(c1, c2, c3, c4, PRIMARY KEY (c4, c3));
INSERT INTO t1(c3) VALUES (0), (0), (0), (0), (0), (0), (0), (0), (0), (0), (NULL), (1), (0);
UPDATE t1 SET c2 = 0;
INSERT INTO t1(c1) VALUES (0), (0), (NULL), (0), (0);
ANALYZE t1;
UPDATE t1 SET c3 = 1;
SELECT DISTINCT * FROM t1 WHERE t1.c3 = 1;

A bug in the skip-scan optimization caused this logic bug

https://www.sqlite.org/src/tktview?name=ced41c7c7d
Example: SQLite

```
CREATE TABLE t1(c1, c2, c3, c4, PRIMARY KEY (c4, c3));
INSERT INTO t1(c3) VALUES (0), (0), (0), (0), (0), (0),
(0), (0), (0), (0), (NULL), (1), (0);
UPDATE t1 SET c2 = 0;
INSERT INTO t1(c1) VALUES (0), (0), (NULL), (0), (0);
ANALYZE t1;
UPDATE t1 SET c3 = 1;
SELECT DISTINCT * FROM t1 WHERE t1.c3 = 1;
```

```
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>NULL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NULL</td>
<td>1</td>
<td>NULL</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>1</td>
<td>NULL</td>
</tr>
</tbody>
</table>
```

https://www.sqlite.org/src/tktview?name=ced41c7c7d
Example: SQLite

CREATE TABLE t1(c1, c2, c3, c4, PRIMARY KEY (c4, c3));
INSERT INTO t1(c3) VALUES (0), (0), (0), (0), (0), (0), (0), (0), (0), (0), (NULL), (1), (0);
UPDATE t1 SET c2 = 0;
INSERT INTO t1(c1) VALUES (0), (0), (NULL), (0), (0);
ANALYZE t1;
UPDATE t1 SET c3 = 1;
SELECT DISTINCT * FROM t1 WHERE t1.c3 = 1;

The bug was classified as “Severe” and quickly fixed

https://www.sqlite.org/src/tktview?name=ced41c7c7d
Result: Bug in PostgreSQL

CREATE TABLE t0(c0 INT PRIMARY KEY, c1 INT);
CREATE TABLE t1(c0 INT) INHERITS (t0);
INSERT INTO t0(c0, c1) VALUES(0, 0);

https://www.postgresql.org/message-id/CA%2Bu7OA7VLKf_vEr6kLF3MnWSA9LToJYncgpNX2tQ-oWzYCBQAw%40mail.gmail.com
Result: Bug in PostgreSQL

CREATE TABLE t0(c0 INT PRIMARY KEY, c1 INT);
CREATE TABLE t1(c0 INT) INHERITS (t0);
INSERT INTO t0(c0, c1) VALUES(0, 0);
INSERT INTO t1(c0, c1) VALUES(0, 1);

https://www.postgresql.org/message-id/CA%2Bu7OA7VLKf_vEr6kLF3MnWSA9LT0JYncgpNX2tQ-oWzYCBQAw%40mail.gmail.com
Result: Bug in PostgreSQL

The inheritance relationship causes the row to be **inserted both in t0 and t1**

---

https://www.postgresql.org/message-id/CA%2Bu7OA7VLKf_vEr6kLF3MnWSA9LTpYlYncgpNX2tQ-oWzYCBQAw%40mail.gmail.com
Result: Bug in PostgreSQL

CREATE TABLE t0(c0 INT PRIMARY KEY, c1 INT);
CREATE TABLE t1(c0 INT) INHERITS (t0);
INSERT INTO t0(c0, c1) VALUES(0, 0);
INSERT INTO t1(c0, c1) VALUES(0, 1);
SELECT c0, c1 FROM t0 GROUP BY c0, c1;

https://www.postgresql.org/message-id/CA%2Bu7OA7VLKf_vEr6kLF3MnWSA9LTjoJYncgpNX2tQ-oWzYCBQAw%40mail.gmail.com
Result: Bug in PostgreSQL

<table>
<thead>
<tr>
<th>t0</th>
<th>c0</th>
<th>c1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

```
CREATE TABLE t0(c0 INT PRIMARY KEY, c1 INT);
CREATE TABLE t1(c0 INT) INHERITS (t0);
INSERT INTO t0(c0, c1) VALUES(0, 0);
INSERT INTO t1(c0, c1) VALUES(0, 1);
SELECT c0, c1 FROM t0 GROUP BY c0, c1;
```

Neutral

https://www.postgresql.org/message-id/CA%2Bu7OA7VLKf_vEr6kLF3MnWSA9LT0JYncgpNX2tQ-oWzYCBQAw%40mail.gmail.com
Result: Bug in PostgreSQL

<table>
<thead>
<tr>
<th></th>
<th>t0</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>c0</td>
<td>c1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

CREATE TABLE t0(c0 INT PRIMARY KEY, c1 INT);
CREATE TABLE t1(c0 INT) INHERITS (t0);
INSERT INTO t0(c0, c1) VALUES(0, 0);
INSERT INTO t1(c0, c1) VALUES(0, 1);
SELECT c0, c1 FROM t0 GROUP BY c0, c1;

https://www.postgresql.org/message-id/CA%2Bu7OA7VLKf_vEr6kLF3MnWSA9LT0JYnCgpNX2tQ-oWzYCBQAw%40mail.gmail.com
Result: Bug in MySQL

CREATE TABLE t0(c0 INT);
INSERT INTO t0(c0) VALUES (1);
SELECT * FROM t0 WHERE 123 != (NOT (NOT 123));

Result: Bug in MySQL

```
CREATE TABLE t0(c0 INT);
INSERT INTO t0(c0) VALUES (1);
SELECT * FROM t0 WHERE 123 != (NOT (NOT 123));
```

The double negation cannot be removed due to MySQL’s flexible type system

Result: Bug in MySQL

CREATE TABLE t0(c0 INT);
INSERT INTO t0(c0) VALUES (1);
SELECT * FROM t0 WHERE 123 != (NOT (NOT 123));

### Oracles

<table>
<thead>
<tr>
<th>DBMS</th>
<th>Logic</th>
<th>Error</th>
<th>Crash</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLite</td>
<td>46</td>
<td>17</td>
<td>2</td>
</tr>
<tr>
<td>MySQL</td>
<td>14</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>1</td>
<td>7</td>
<td>1</td>
</tr>
</tbody>
</table>

Error bugs are due to unexpected (internal) errors.
Example: SQLite3 Bug

**Example SQL Statements:**

```sql
CREATE TABLE t0(c0, c1 REAL PRIMARY KEY);
INSERT INTO t0(c0, c1) VALUES
(TRUE, 9223372036854775807), (TRUE, 0);
UPDATE t0 SET c0 = NULL;
UPDATE OR REPLACE t0 SET c1 = 1;
SELECT DISTINCT * FROM t0 WHERE (t0.c0 IS NULL);
```

**Issue:**

Database disk image is malformed

---

**Note:**

This example illustrates a bug in SQLite3 where updating a table with a PRIMARY KEY and setting a column to NULL can lead to a corrupted database image.
Discussion: Implementation Effort

• **Literal evaluator**
  • Simpler than PL AST Interpreters → No mutable state
  • Simpler than query engines → only a single row needs to be considered
Discussion: Implementation Effort

• **Literal evaluator**
  - Simpler than PL AST Interpreters → No mutable state
  - Simpler than query engines → only a single row needs to be considered

• Operators are **implemented naively**
  - The performance of the DBMS is the bottleneck
Discussion: Implementation Effort

• **Literal evaluator**
  - Simpler than PL AST Interpreters → No mutable state
  - Simpler than query engines → only a single row needs to be considered

• **Operators are implemented naively**
  - The performance of the DBMS is the bottleneck

• Higher implementation effort for functions (e.g. `printf`) and complex operators
Discussion: Limitations

- Requires understanding of the SQL semantics
- Aggregate and window functions
- Ordering
- Duplicate rows
Discussion: Bug Importance

CREATE TABLE t0 (c0);
CREATE TABLE t1 (c1);
INSERT INTO t0 VALUES (1);
SELECT c0 FROM t0 LEFT JOIN t1 ON c1=c0 WHERE NOT (c1 IS NOT NULL AND c1=2);

https://www.mail-archive.com/sqlite-users@mailinglists.sqlite.org/msg117440.html
Discussion: Bug Importance

CREATE TABLE t0 (c0);
CREATE TABLE t1 (c1);
INSERT INTO t0 VALUES (1);
SELECT c0 FROM t0 LEFT JOIN t1 ON c1=c0 WHERE NOT (c1 IS NOT NULL AND c1=2);

This is a cut-down example, right? You can't possibly mean to do that WHERE clause in production code.
Discussion: Bug Importance

CREATE TABLE t0 (c0);
CREATE TABLE t1 (c1);
INSERT INTO t0 VALUES (1);
SELECT c0 FROM t0 LEFT JOIN t1 ON c1=c0 WHERE NOT (c1 IS NOT NULL AND c1=2);

This is a cut-down example, right? You can't possibly mean to do that WHERE clause in production code.

I might not spell it like that myself, but a code generator would do it (and much worse!). This example was simplified from a query generated by a Django ORM queryset using .exclude(nullable_joined_table__column=1), for instance.

https://www.mail-archive.com/sqlite-users@mailinglists.sqlite.org/msg117440.html
Discussion: Bug Importance

CREATE TABLE t0 (c0);
CREATE TABLE t1 (c1);
INSERT INTO t0 VALUES (1);
SELECT c0 FROM t0 LEFT JOIN t1 ON c1=c0  WHERE NOT (c1 IS NOT NULL AND c1=2);

This is a cut-down example, right? You can't possibly mean to do that WHERE clause in production code.

I might not spell it like that myself, but a code generator would do it (and much worse!). This example was simplified from a query generated by a Django ORM queryset using .exclude(nullable_joined_table__column=1), for instance.

Even “obscure” bugs might affect users

https://www.mail-archive.com/sqlite-users@mailinglists.sqlite.org/msg117440.html
Overview

Pivoted Query Synthesis (PQS)

Non-optimizing Reference Engine Construction (NoREC)

Ternary Logic Query Partitioning (TLP)
Overview

Pivoted Query Synthesis (PQS)

Detecting optimization bugs by rewriting the query so that it cannot be optimized

Non-optimizing Reference Engine Construction (NoREC)

Ternary Logic Query Partitioning (TLP)

>150 bugs
Overview

Pivoted Query Synthesis (PQS)

Non-optimizing Reference Engine Construction (NoREC)

Ternary Logic Query Partitioning (TLP)

**Partition** the query into several **partitioning queries**, which is applicable to test various features

>150 bugs
SQLancer: Supported DBMSs

- TiDB
- H2
- DuckDB
- ClickHouse
- MariaDB
- SQLite
- CockroachDB
- PostgreSQL
Summary

@RiggerManuel  manuel.rigger@inf.ethz.ch

Goal: Detect logic bugs

Example: SQLite3 Bug

| 10 | CREATE TABLE t0(c0); |
| 1  | CREATE INDEX c0 ON t0(1) WHERE c0 NOT NULL; |
| 2  | INSERT INTO t0 (c0) VALUES (0), (1), (2), (3), (NULL); |
| 0  | SELECT c0 FROM t0 WHERE t0.c0 IS NOT 1; |

NULL was not contained in the result set!

PQS randomly selects a pivot row

PQS Idea

| 10 | CREATE TABLE t0(c0); |
| 1  | CREATE INDEX c0 ON t0(1) WHERE c0 NOT NULL; |
| 2  | INSERT INTO t0 (c0) VALUES (0), (1), (2), (3), (NULL); |
| 0  | SELECT c0 FROM t0 WHERE t0.c0 IS NOT 1; |

Pivot row

Validate the result set based on one randomly-selected row

Rectify a random expression

Random Expression Generation

Compound nodes compute their result based on their children

| 10 | CREATE TABLE t0(c0); |
| 1  | CREATE INDEX c0 ON t0(1) WHERE c0 NOT NULL; |
| 2  | INSERT INTO t0 (c0) VALUES (0), (1), (2), (3), (NULL); |
| 0  | SELECT c0 FROM t0 WHERE t0.c0 IS NOT 1; |

Pivot row

Validate the result set based on one randomly-selected row

Evaluation: Close to 100 bugs in DBMSs

Bugs Overview

<table>
<thead>
<tr>
<th>DBMS</th>
<th>Fixed</th>
<th>Verified</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLite</td>
<td>64</td>
<td>0</td>
</tr>
<tr>
<td>MySQL</td>
<td>17</td>
<td>7</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>

96 bugs were unique, previously unknown ones

ETH zurich