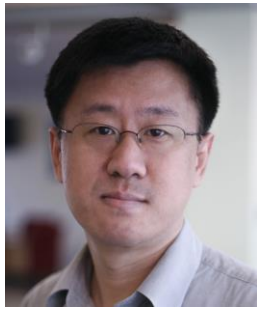
ETH zürich

Testing Database Engines via Pivoted Query Synthesis



Manuel Rigger Zhendong Su

ETH Zurich, Switzerland

11/05/2020

AST
ADVANCED SOFTWARE TECHNOLOGIES

@RiggerManuel @ast_eth

https://people.inf.ethz.ch/suz/

Database Management Systems (DBMSs)

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

Goal: Find Logic Bugs

Logic bugs: DBMS returns an incorrect result set

Example: SQLite3 Bug

t0

c0
0
1
2
NULL

```
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;
```

https://sqlite.org/src/tktview/80256748471a01

Example: SQLite3 Bug

t0

c0
0
1
2
NULL

```
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;
```

IS NOT is a “null-safe” comparison operator

https://sqlite.org/src/tktview/80256748471a01

Example: SQLite3 Bug

t0

c0
0
1
2
NULL

```
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;
```

https://sqlite.org/src/tktview/80256748471a01

Example: SQLite3 Bug

t0

c0
0
1
2
NULL

```
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;
```

0

TRUE

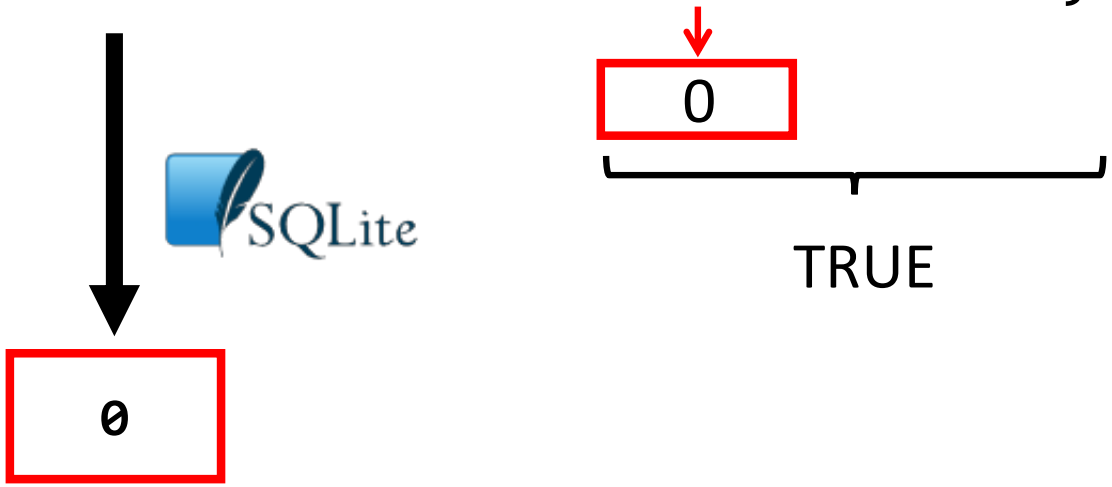
https://sqlite.org/src/tktview/80256748471a01

Example: SQLite3 Bug

t0

c0
0
1
2
NULL

```
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;
```



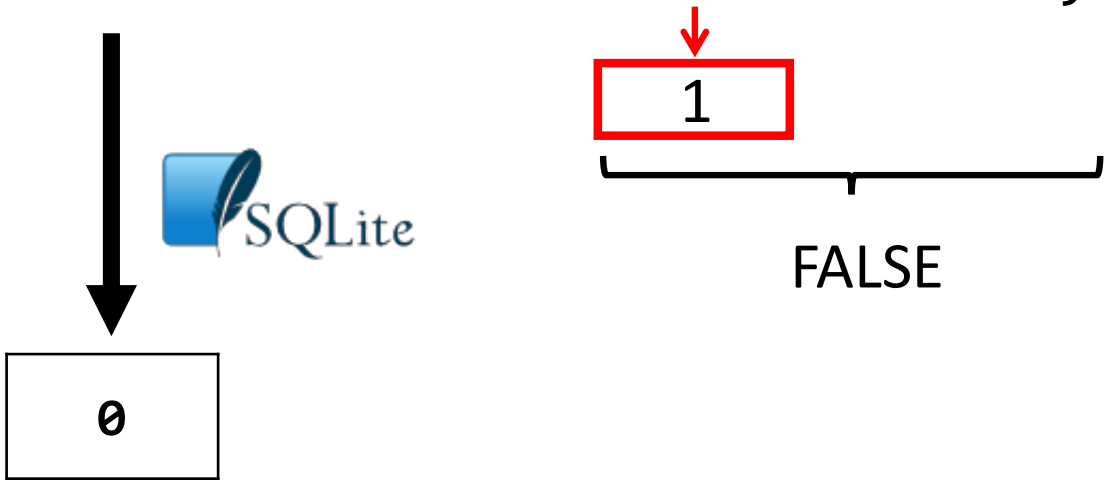
https://sqlite.org/src/tktview/80256748471a01

Example: SQLite3 Bug

t0

c0
0
1
2
NULL

```
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;
```



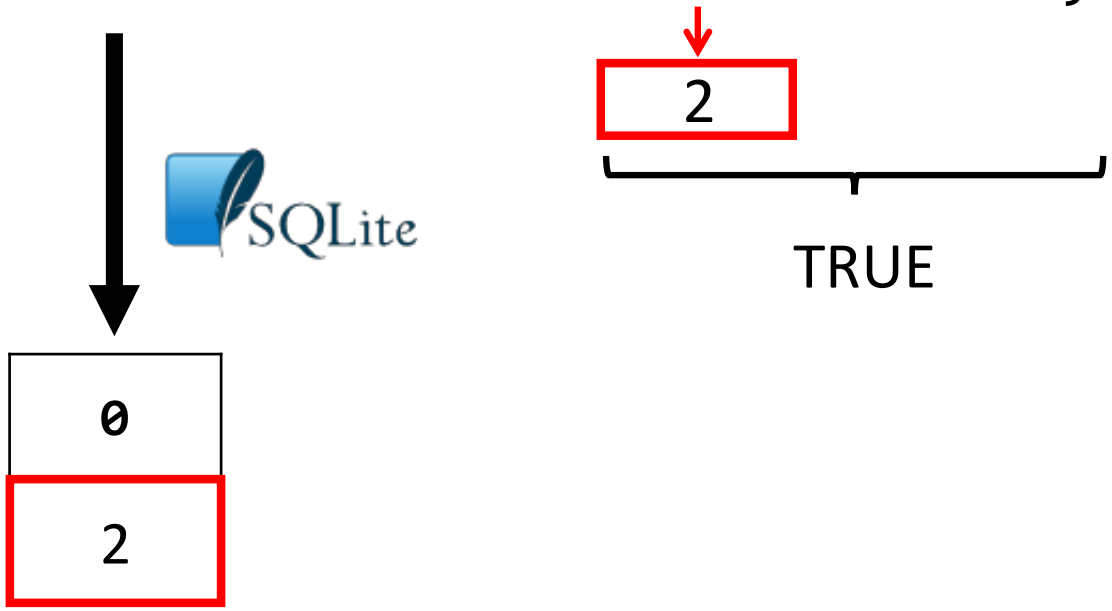
https://sqlite.org/src/tktview/80256748471a01

Example: SQLite3 Bug

t0

c0
0
1
2
NULL

```
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;
```



https://sqlite.org/src/tktview/80256748471a01

Example: SQLite3 Bug

t0

c0
0
1
2
NULL

```
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

TRUE

SQLite

0
2
NULL

https://sqlite.org/src/tktview/80256748471a01

Example: SQLite3 Bug

t0

c0
0
1
2
NULL

```
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

TRUE

SQLite

0
2
NULL

https://sqlite.org/src/tktview/80256748471a01

Example: SQLite3 Bug

t0

c0
0
1
2
NULL

```
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

TRUE

0
2

https://sqlite.org/src/tktview/80256748471a01

Example: SQLite3 Bug

t0

c0
0
1
2
NULL

```
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

TRUE

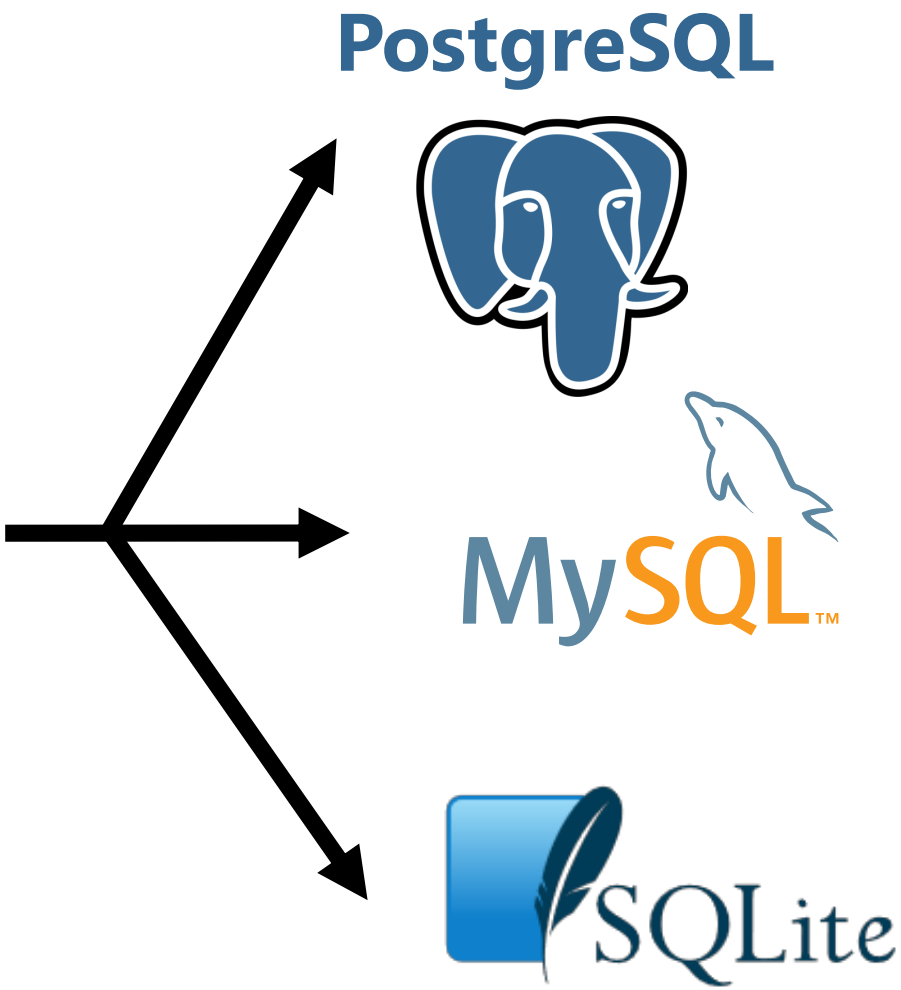
0
2

NULL was not contained in the result set!

https://sqlite.org/src/tktview/80256748471a01

Background: Differential Testing

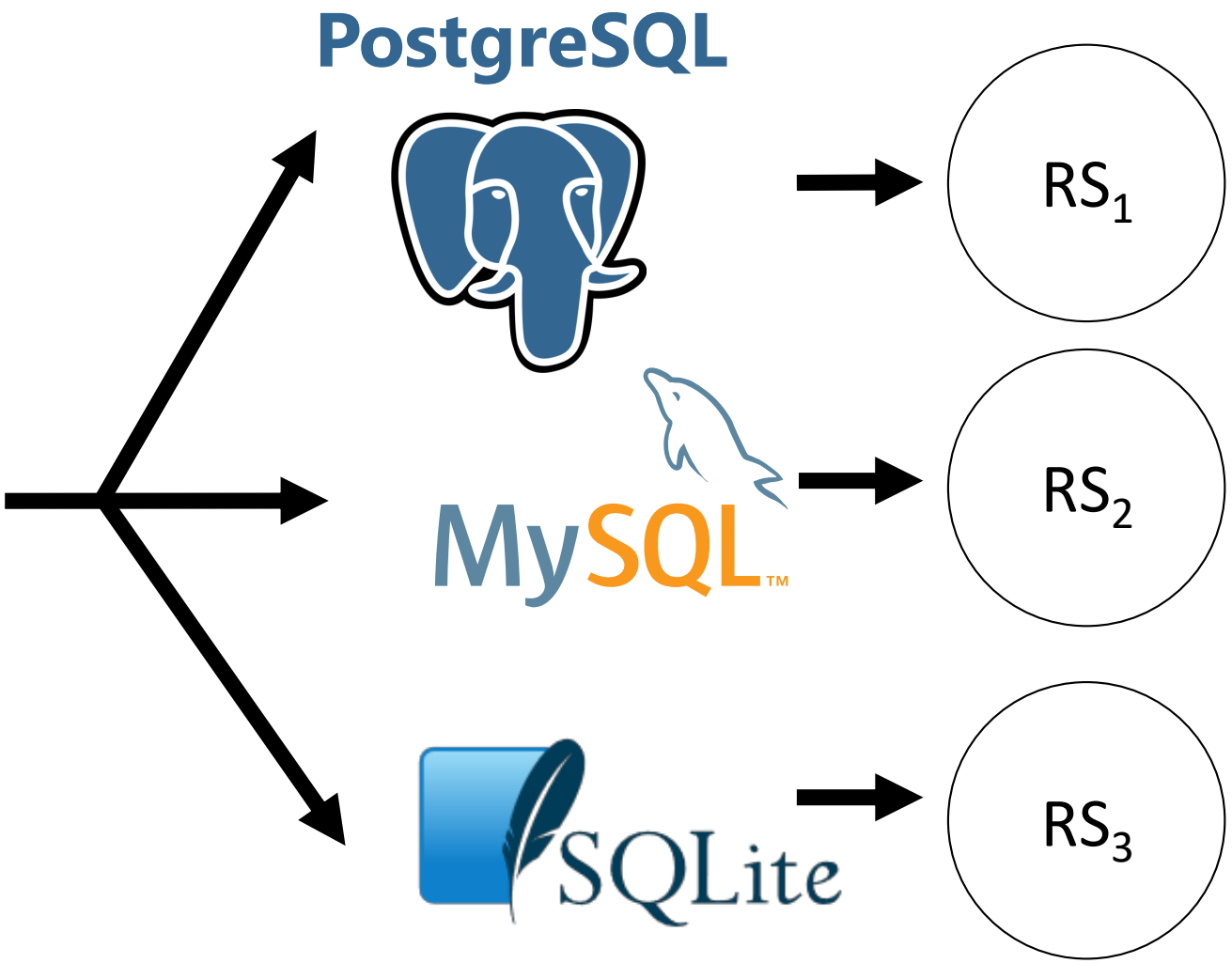
```
SELECT c0 FROM t0
WHERE t0.c0 IS NOT 1;
```



Massive Stochastic Testing of SQL by Slutz, 1998.

Background: Differential Testing

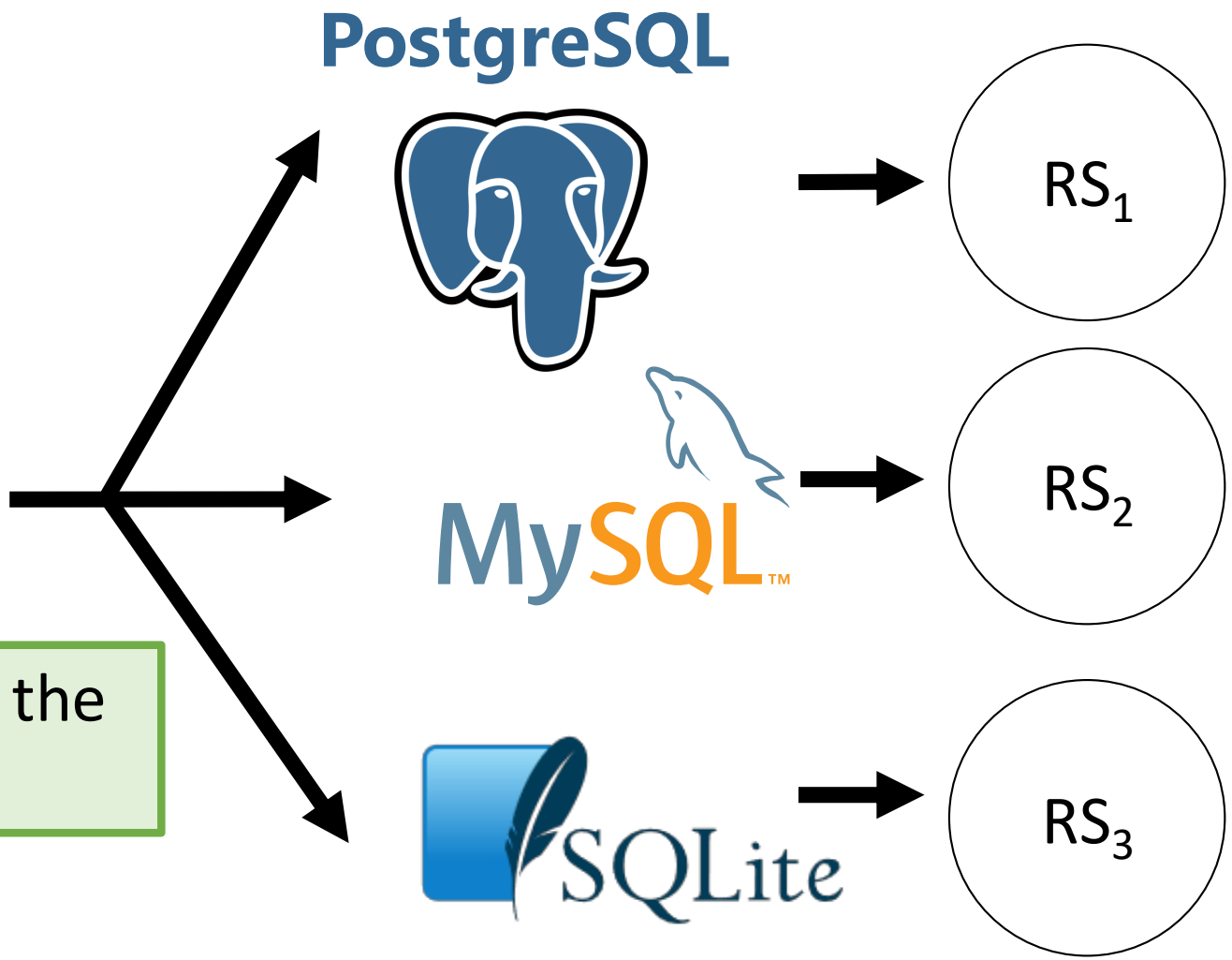
```
SELECT c0 FROM t0
WHERE t0.c0 IS NOT 1;
```



Massive Stochastic Testing of SQL by Slutz, 1998.

Background: Differential Testing

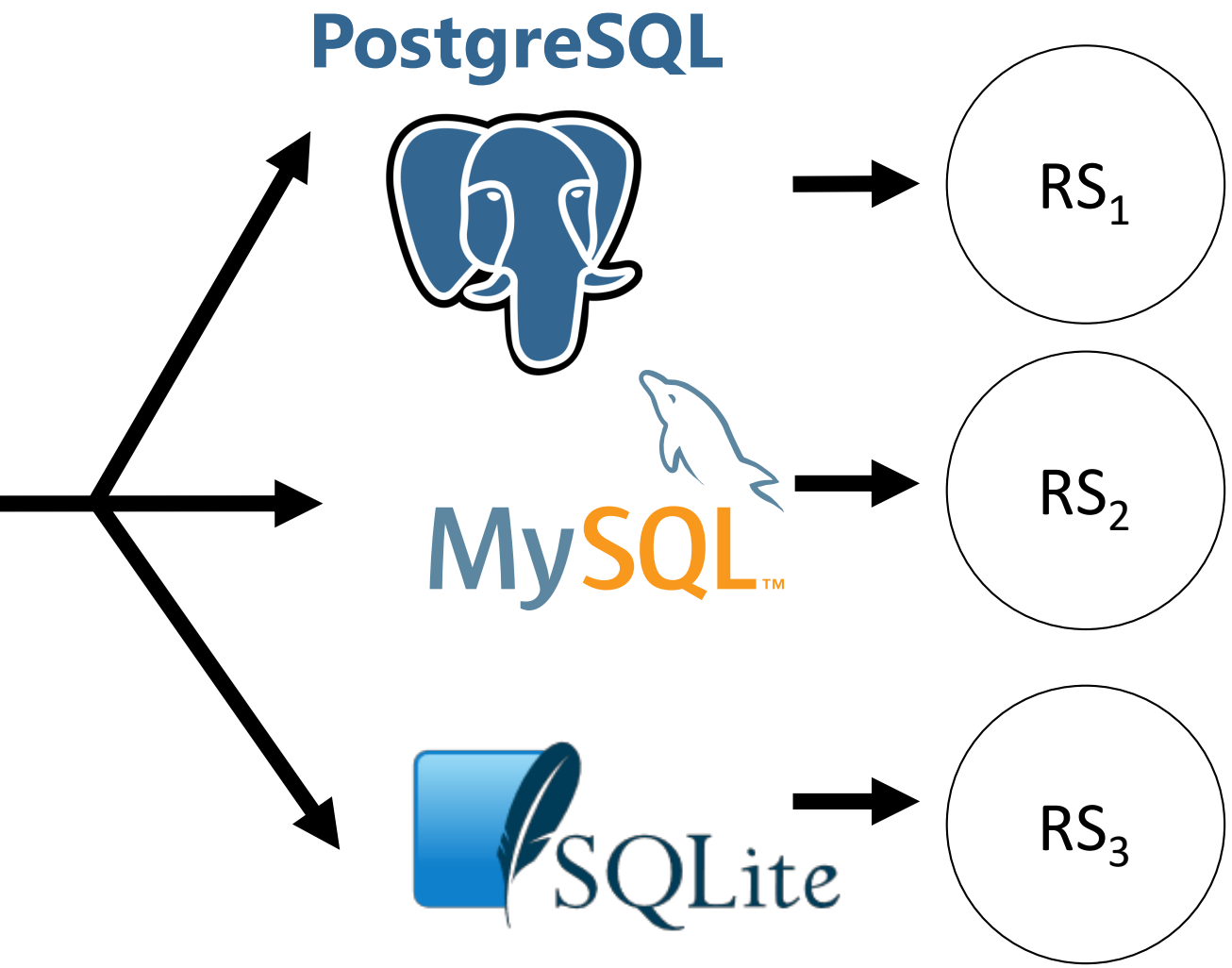
```
SELECT c0 FROM t0
WHERE t0.c0 IS NOT 1;
```



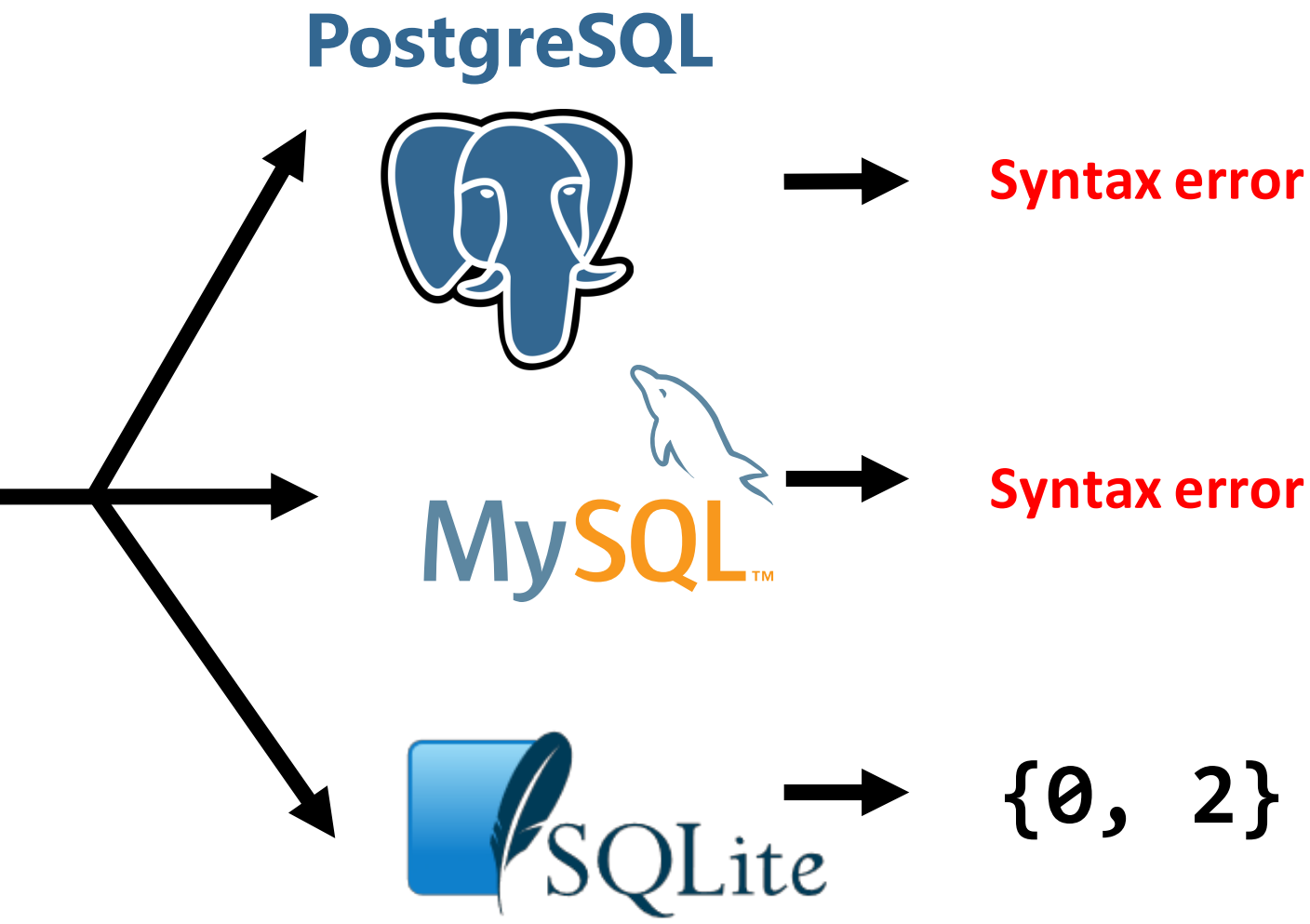
Check that all DBMSs compute the **same result ($RS_1 = RS_2 = RS_3$)**

Massive Stochastic Testing of SQL by Slutz, 1998.

Background: Differential Testing



Background: Differential Testing



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;
```

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;
```

MySQL and PostgreSQL require a data type definition

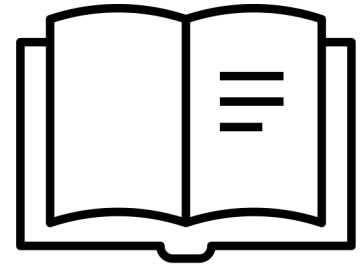
Background: Differential Testing

```
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;
```

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

Pivoted Query Synthesis (PQS)

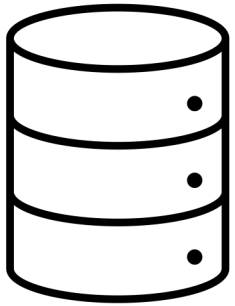
```
CREATE TABLE t0(c0);
CREATE INDEX i0 ON t0(1) WHERE c0 NOT NULL;
INSERT INTO t0 (c0) VALUES (0), (1), (2), (NULL);
SELECT c0 FROM t0 WHERE t0.c0 IS NOT 1;
```



IS NOT is a “null-safe”…

t0

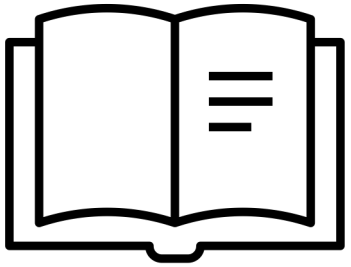
c0
0
1
2
NULL



0
2

Pivoted Query Synthesis (PQS)

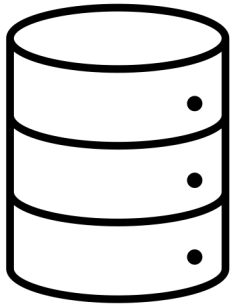
```
CREATE TABLE t0(c0);
CREATE INDEX i0 ON t0(1) WHERE c0 NOT NULL;
INSERT INTO t0 (c0) VALUES (0), (1), (2), (NULL);
SELECT c0 FROM t0 WHERE t0.c0 IS NOT 1;
```



IS NOT is a “null-safe”…

t0

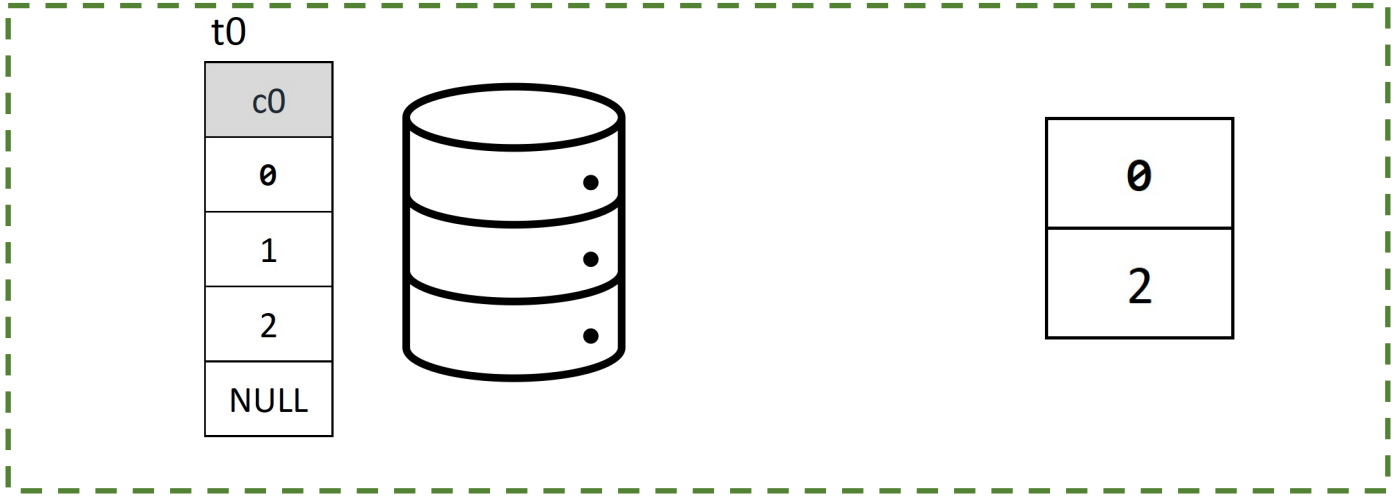
c0
0
1
2
NULL



0
2

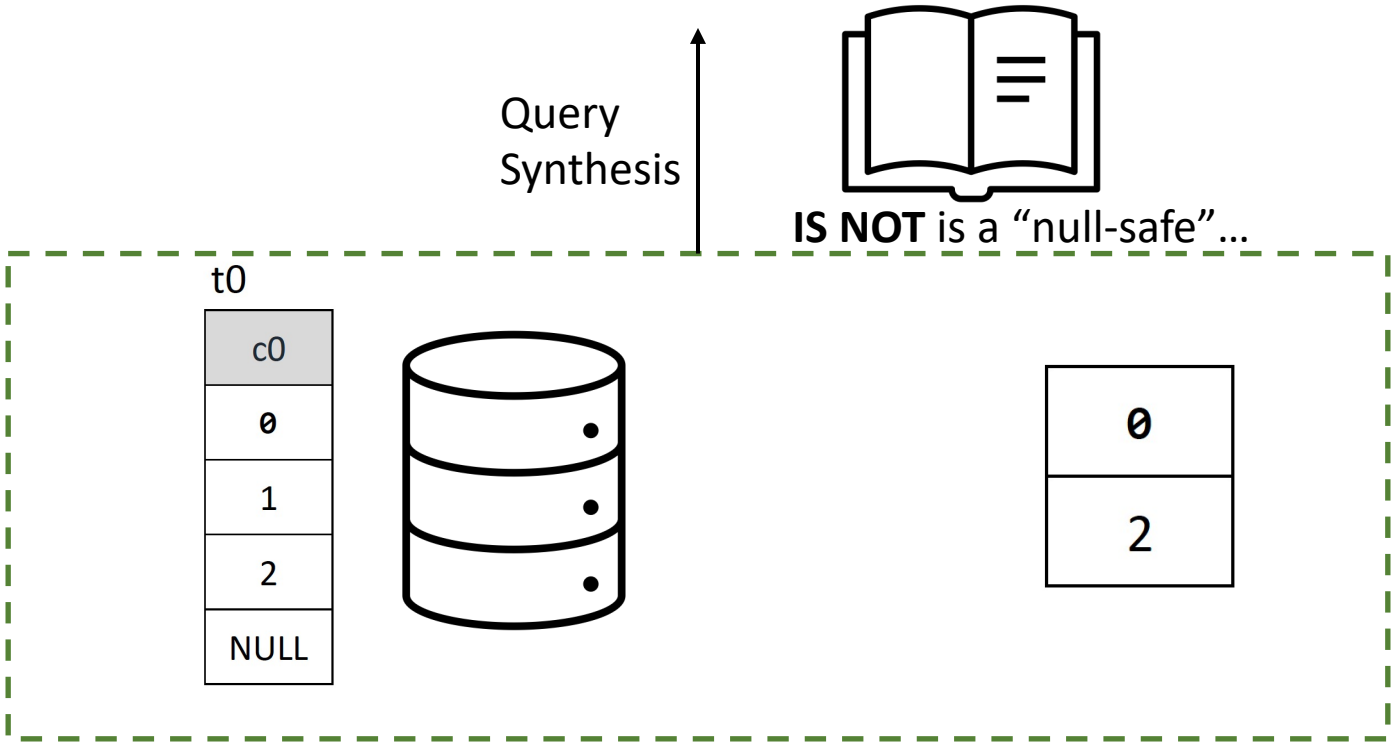
Correct or not?

Pivoted Query Synthesis (PQS)



Pivoted Query Synthesis (PQS)

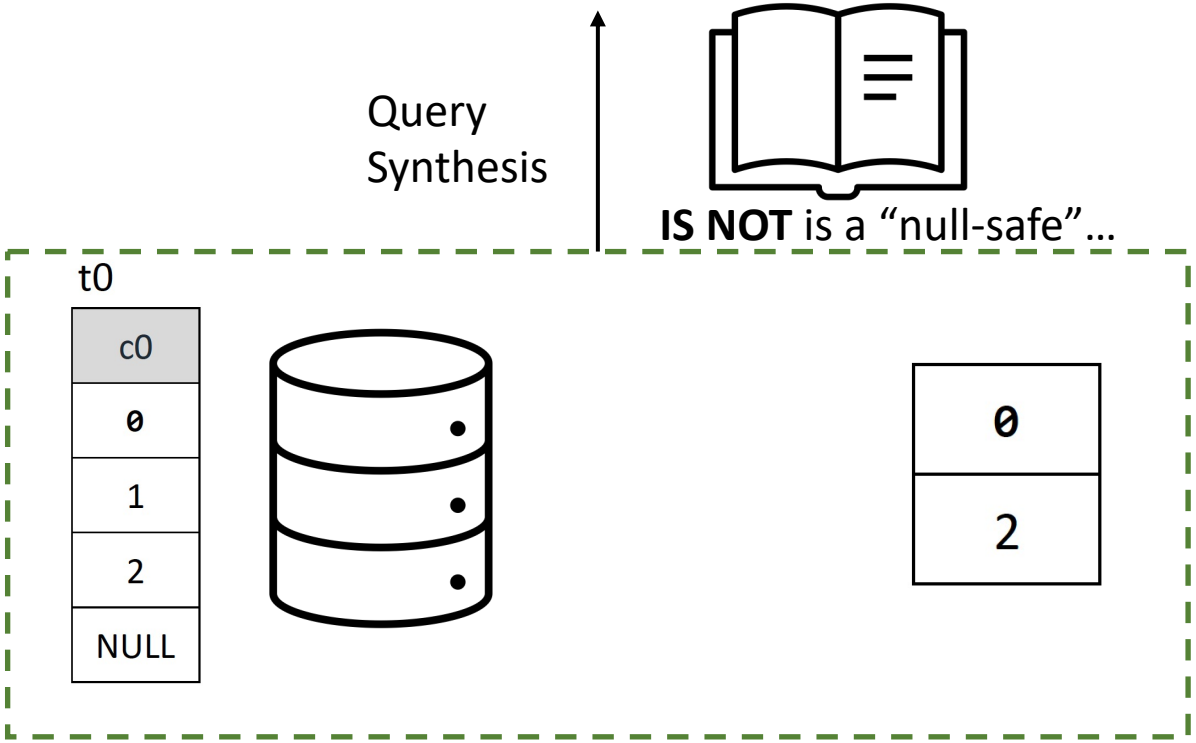
```
CREATE TABLE t0(c0);
CREATE INDEX i0 ON t0(1) WHERE c0 NOT NULL;
INSERT INTO t0 (c0) VALUES (0), (1), (2), (NULL);
SELECT c0 FROM t0 WHERE t0.c0 IS NOT 1;
```



Pivoted Query Synthesis (PQS)

Infeasible to synthesize exact query for the output

```
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;
```

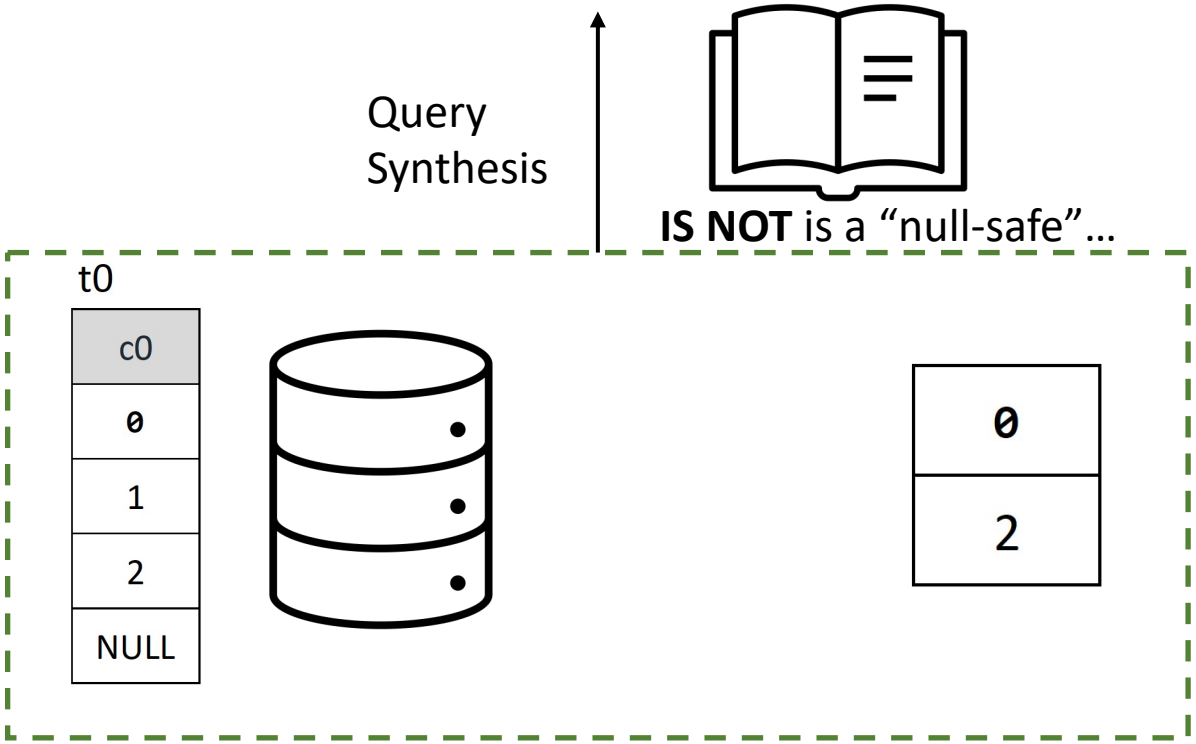


Pivoted Query Synthesis (PQS)

Infeasible to synthesize exact query for the output

Let's relax it: Synthesize a query producing partial output

```
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;
```



PQS Idea

t0

c0
0
1
2
NULL

```
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

t0

c0
0
1
2
NULL

Pivot row

```
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

t0

c0
0
1
2
NULL

```
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;
```

NULL

TRUE

Generate a query that is guaranteed to at least fetch the pivot row

PQS Idea

t0

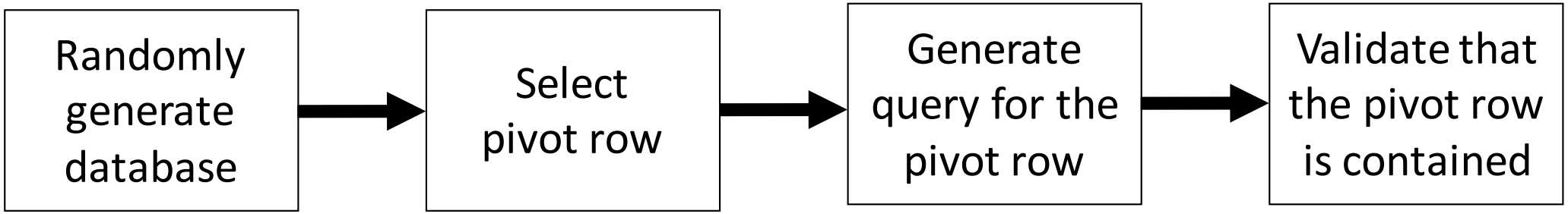
c0
0
1
2
NULL

```
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;
```

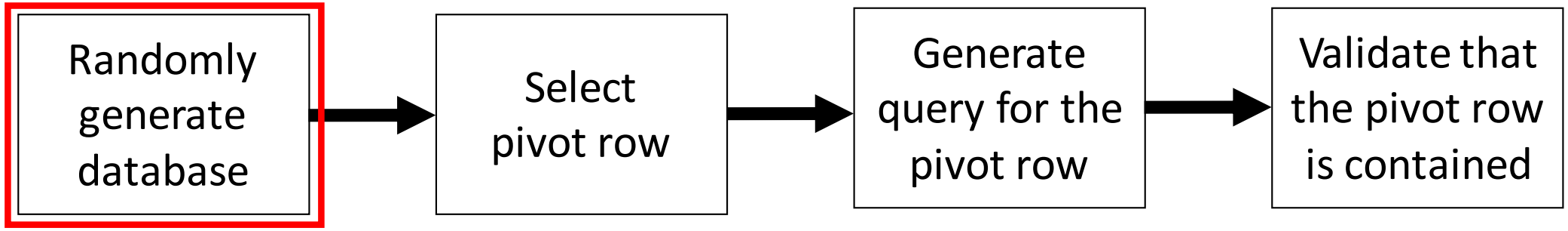
0
2

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

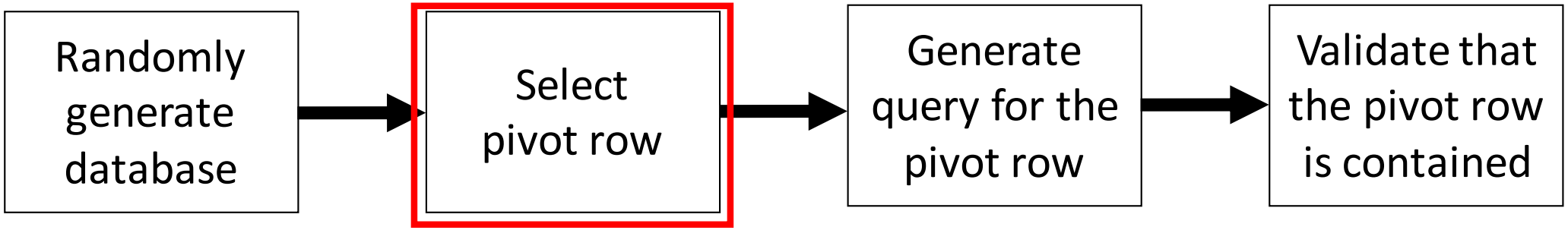
Approach




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

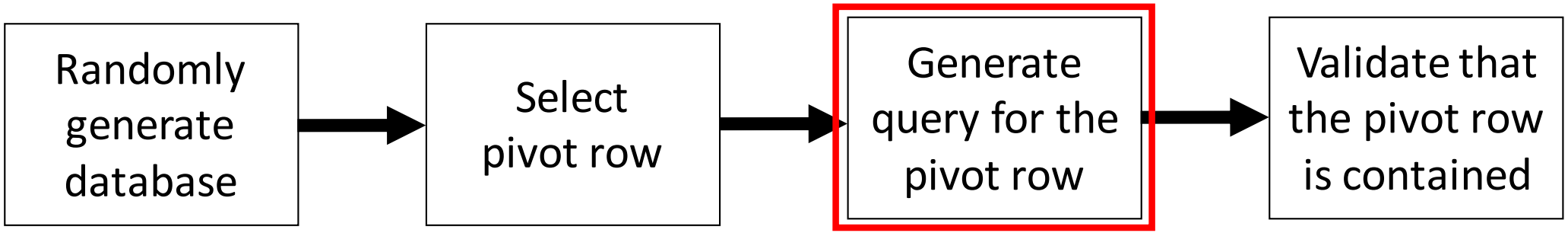
Statements are **heuristically generated** based on the DBMS' SQL dialect

Approach



One **random row** from multiple tables and views

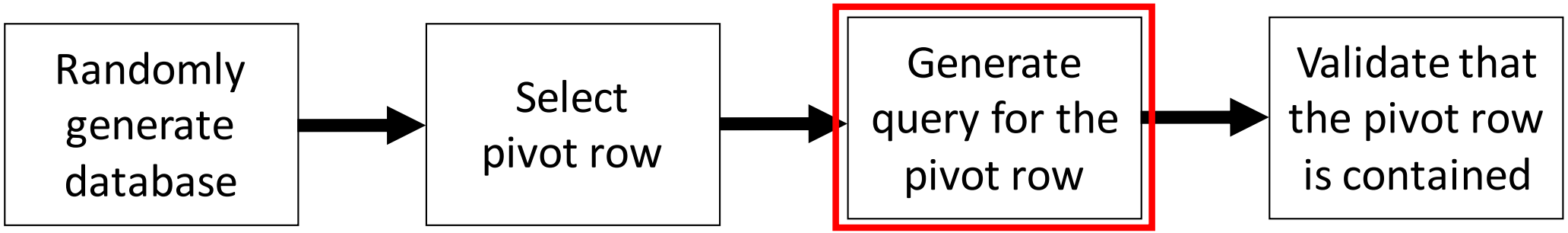
Approach

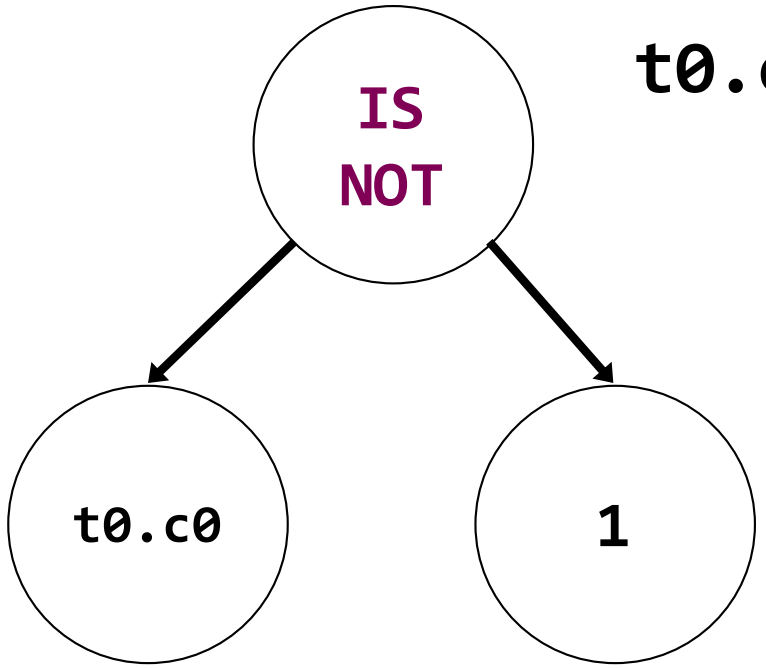



```
SELECT c0 FROM t0
WHERE [                    ]
```

Generate **predicates** that **evaluate to TRUE** for the pivot row and use them in JOIN and WHERE clauses

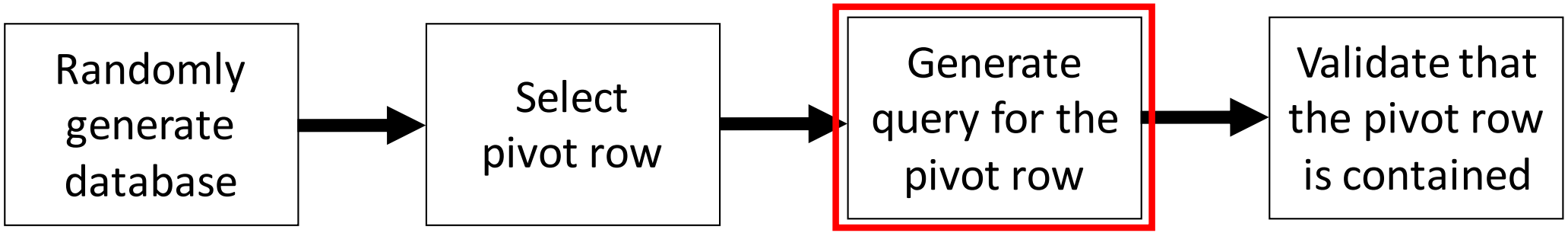
Random Expression Generation

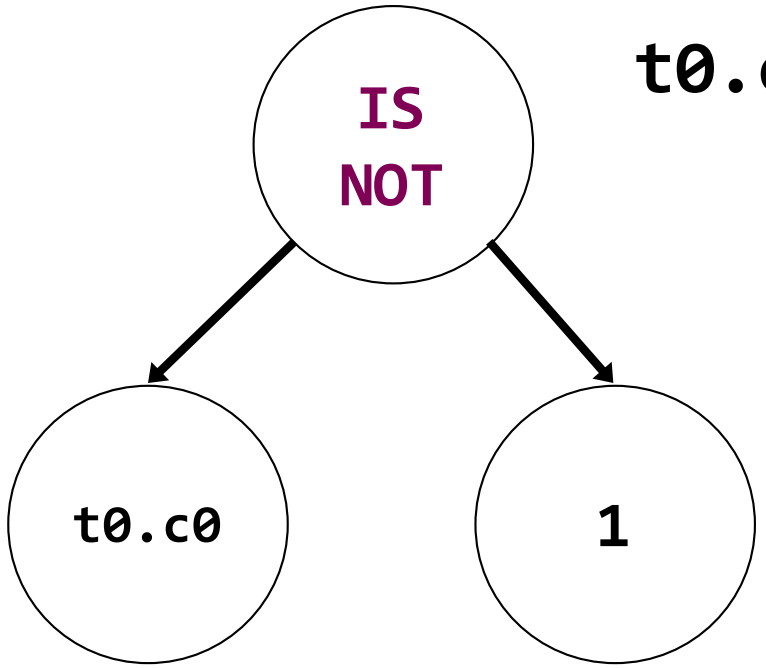





```
t0.c0 IS NOT 1;
```

Random Expression Generation

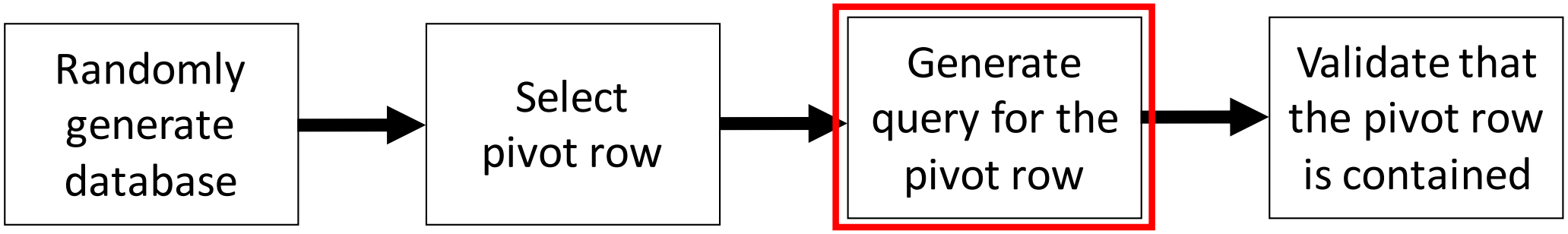


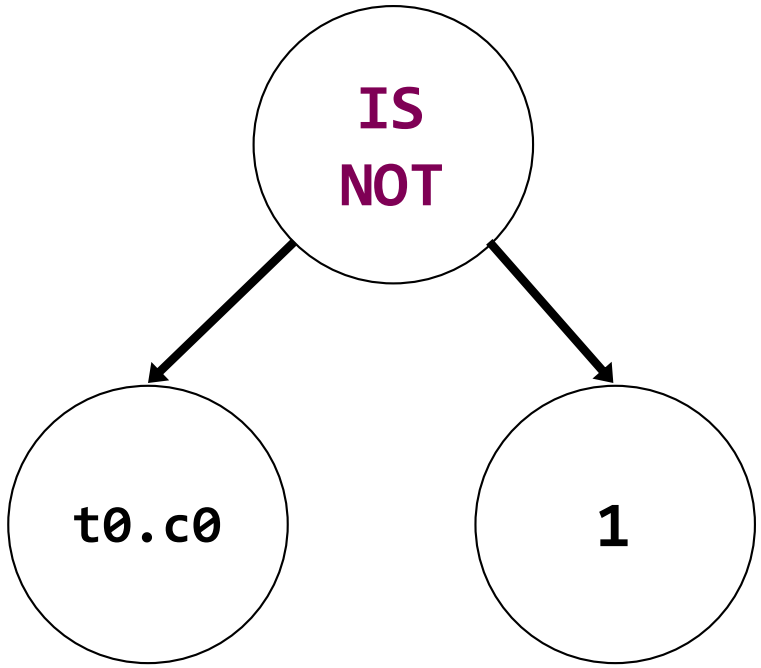



```
t0.c0 IS NOT 1;
```

We implemented an **expression evaluator** for each node

Random Expression Generation



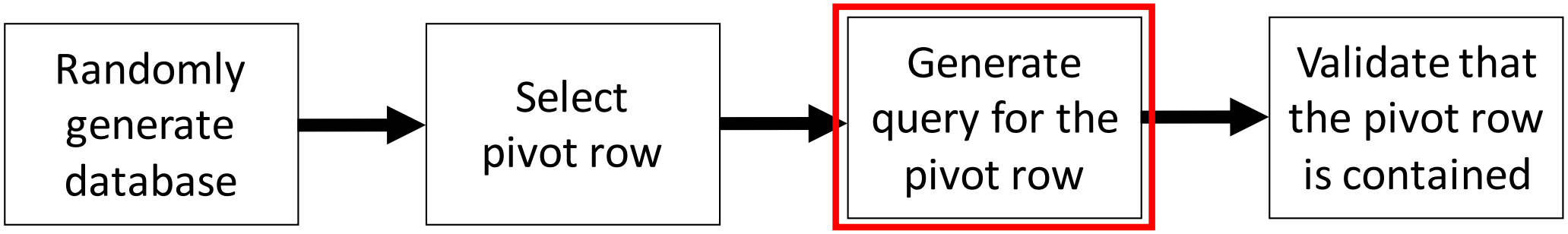


Evaluate the tree based on the **pivot row**

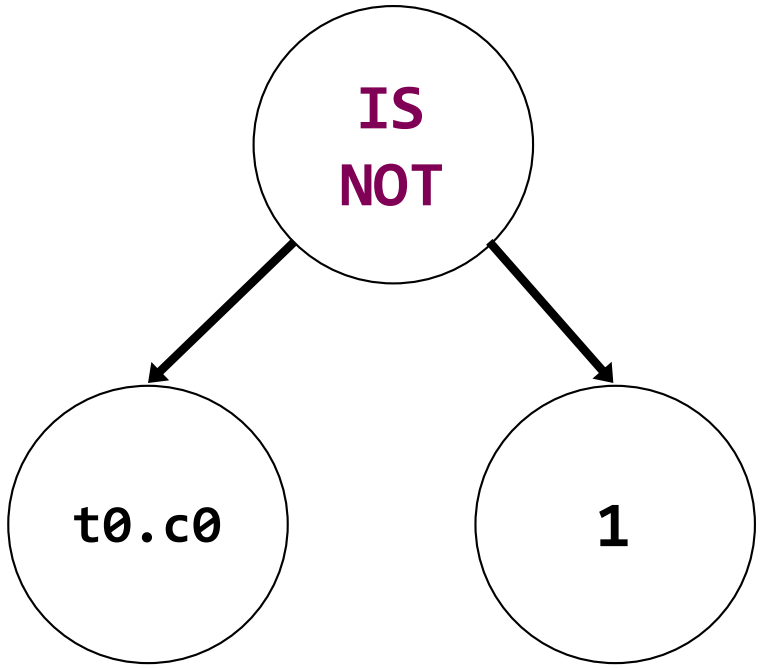
t0

c0
0
1
2
NULL

Random Expression Generation



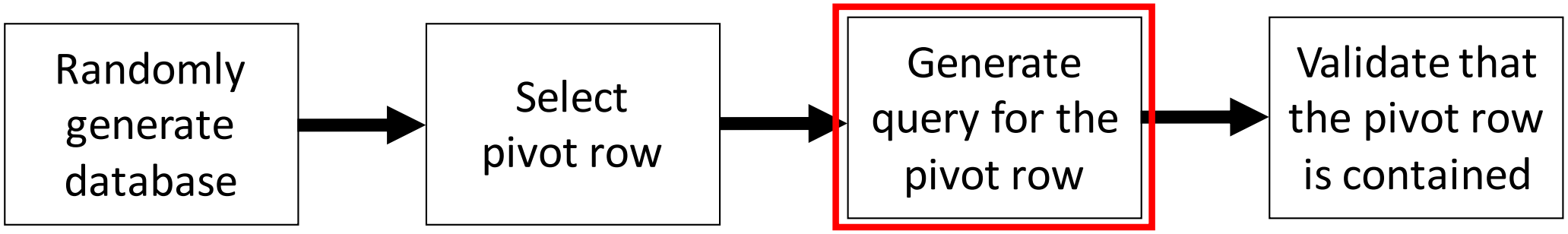
Column references return the **values** from the **pivot row**



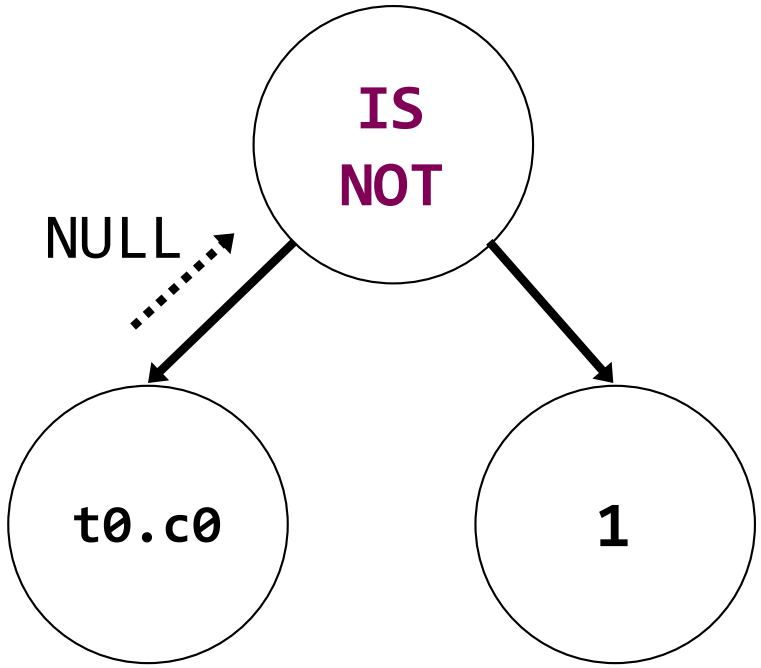
t0

c0
0
1
2
NULL

Random Expression Generation



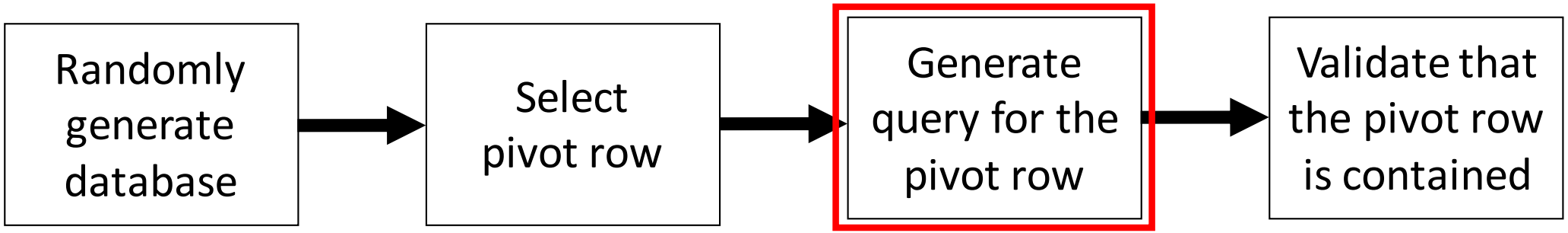
Column references return the **values** from the **pivot row**

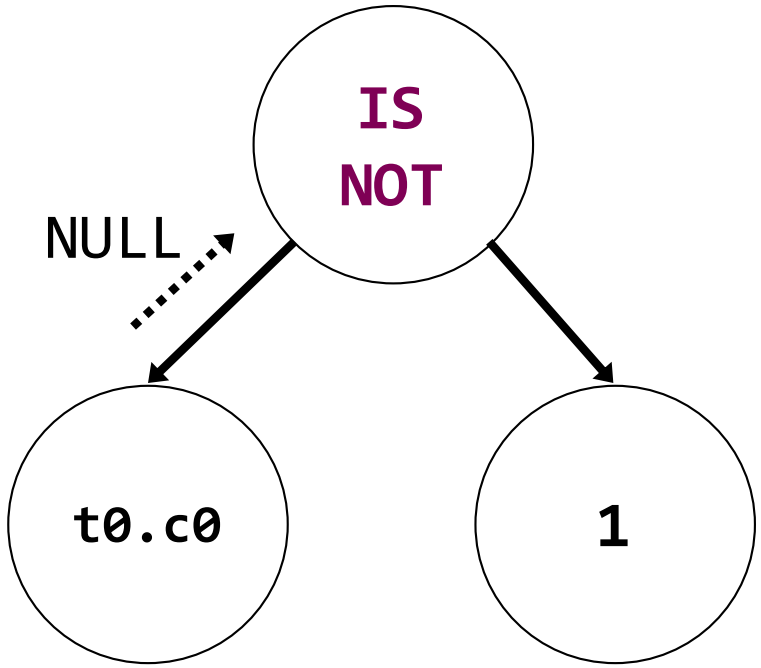


t0

c0
0
1
2
NULL

Random Expression Generation



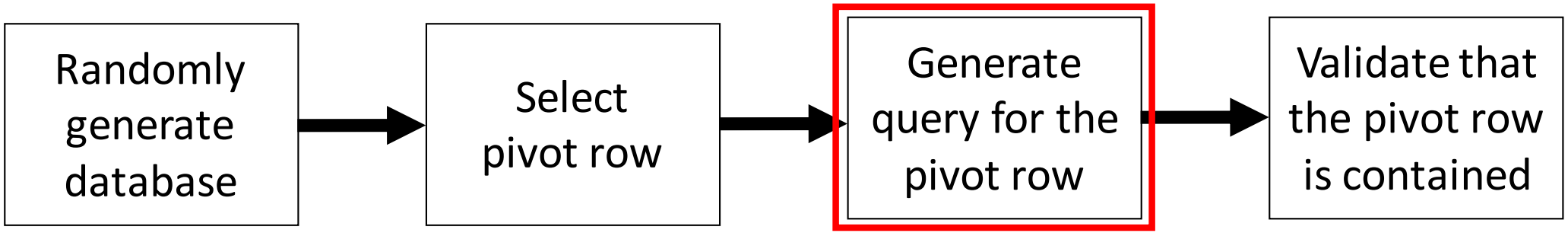


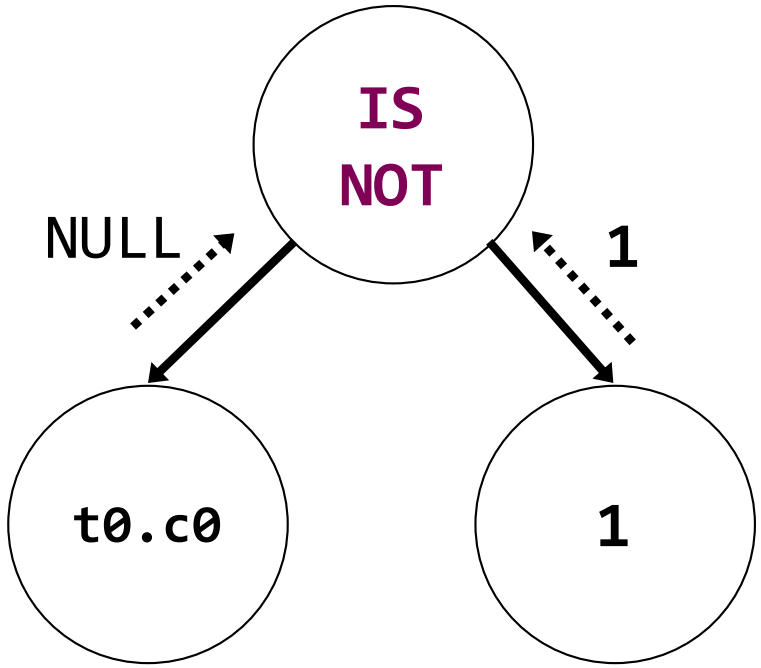
Constant nodes return their assigned **literal values**

t0

c0
0
1
2
NULL

Random Expression Generation



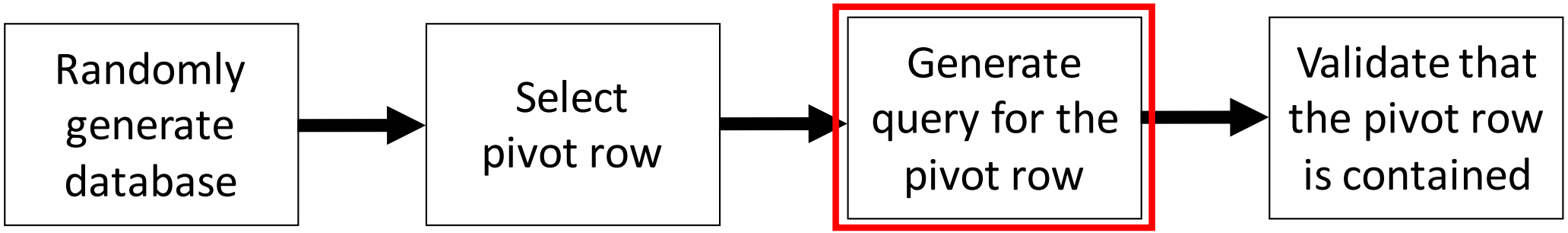


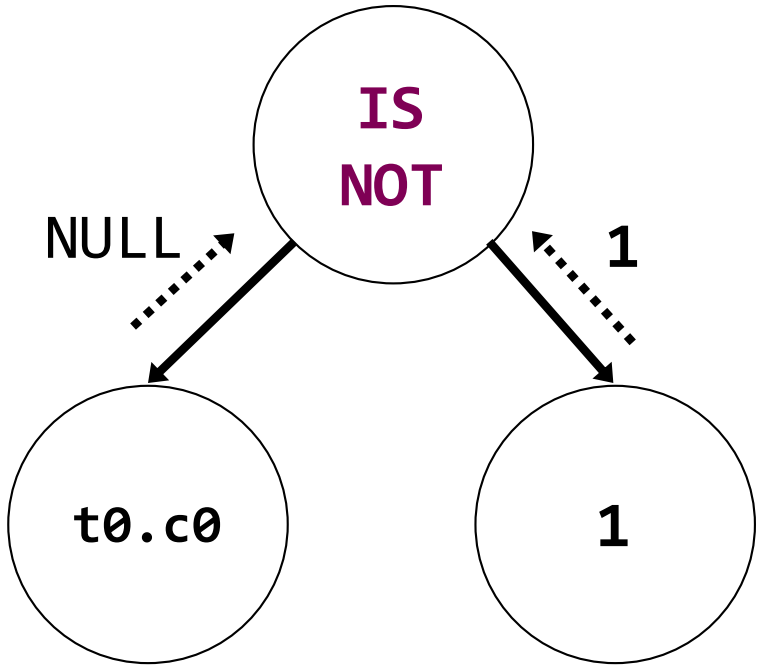
Constant nodes return their assigned **literal values**

t0

c0
0
1
2
NULL

Random Expression Generation



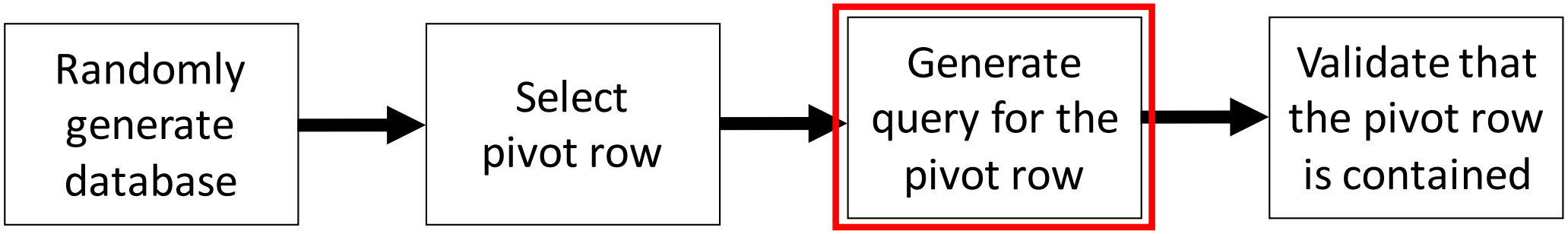


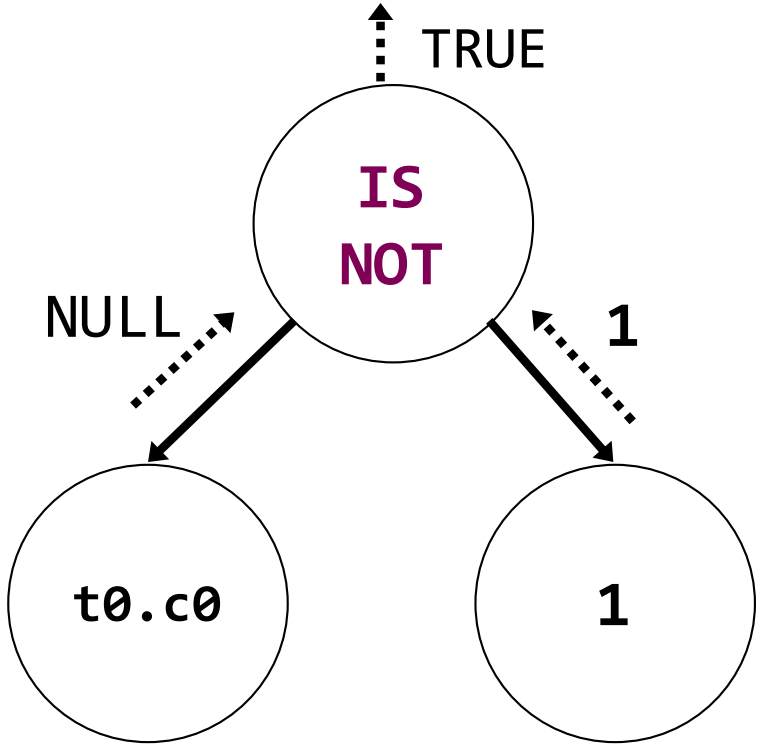
Compound nodes compute their result **based on their children**

t0

c0
0
1
2
NULL

Random Expression Generation



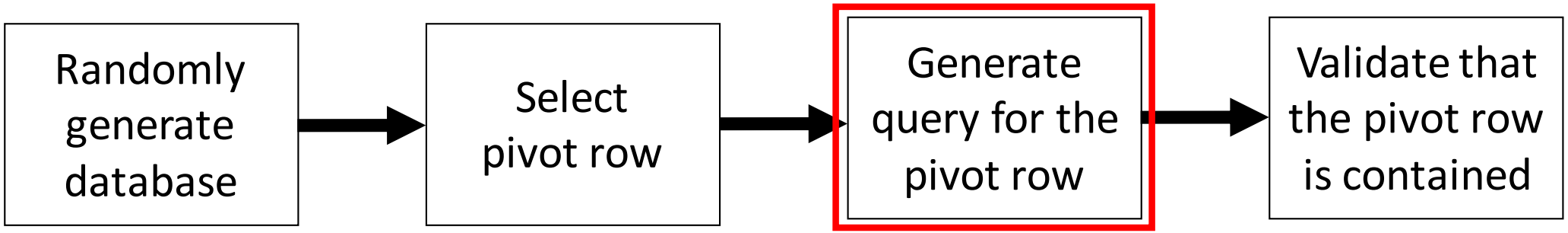


Compound nodes compute their result **based on their children**

t0

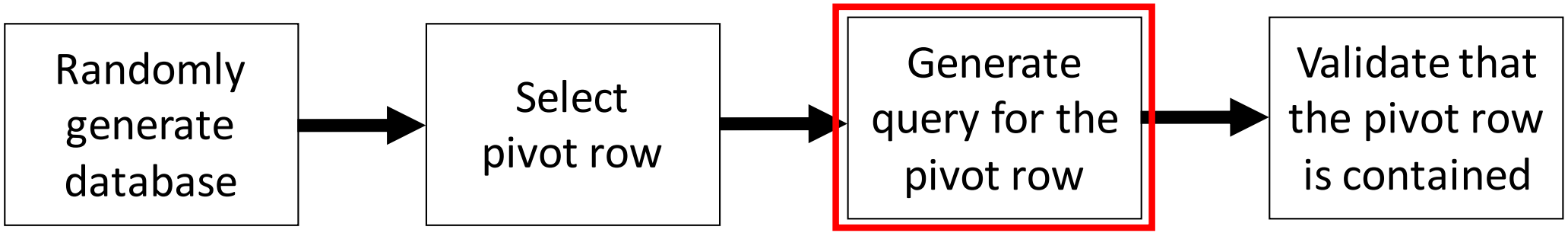
c0
0
1
2
NULL

Query Synthesis




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

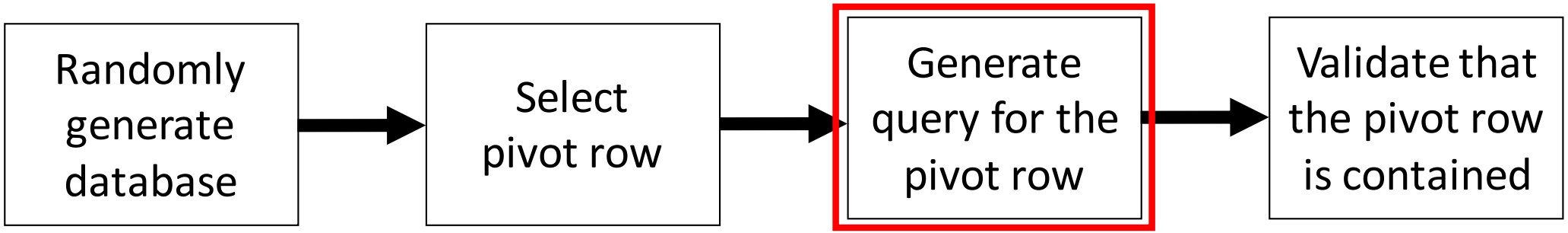
Query Synthesis




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

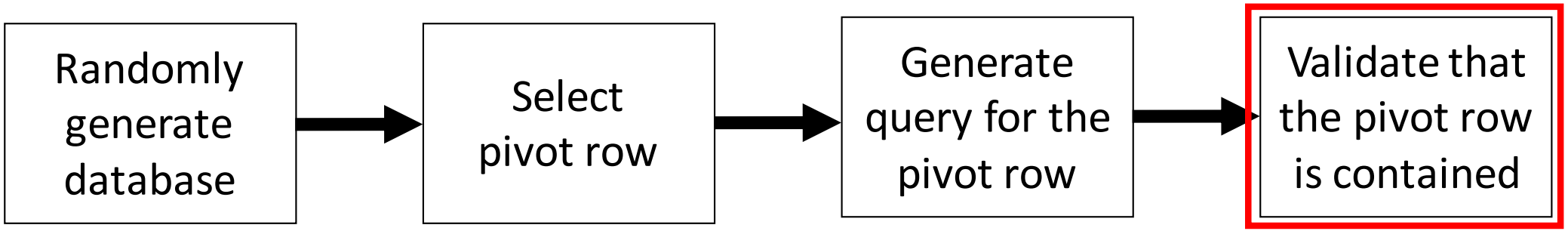
What if the expression **does not evaluate to TRUE**?

Random Expression Rectification




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

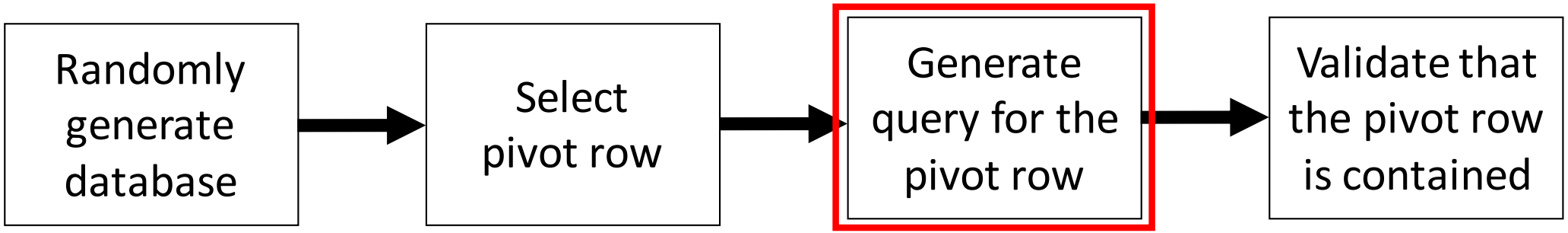
Approach




```
SELECT (NULL) INTERSECT
  SELECT c0 FROM t0 WHERE NULL IS NOT 1;
```

Rely on the DBMS to check whether the row is contained

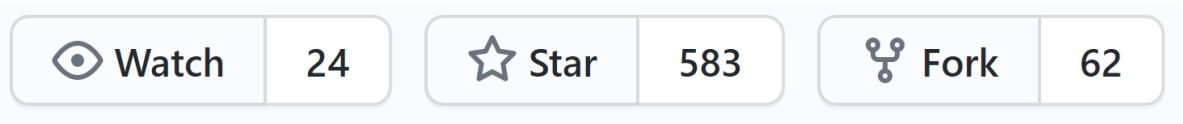
Random Expression Rectification




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

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

Implementation



https://github.com/sqlancer

Bugs Overview

DBMS	Fixed	Verified
SQLite	64	0
MySQL	17	7
PostgreSQL	5	3

Bugs Overview

DBMS	Fixed	Verified
SQLite	64	0
MySQL	17	7
PostgreSQL	5	3

96 bugs were unique, previously unknown ones

Oracles

DBMS	Logic	Error	Crash
SQLite	46	17	2
MySQL	14	10	1
PostgreSQL	1	7	1

61 were logic bugs

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.

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.*

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

Discussion: Limitations

- Implementation effort for complex operations
- Requires understanding of the SQL semantics
- Aggregate and window functions
- Ordering
- Duplicate rows

Discussion

- Does not explain the bugs well
  - Nature of black-box testing
- SQLancer detects bugs that expected results not outputted (containment bugs?)
  - Unexpected results can rarely be outputted
  - Ensure the completeness of results? Hard!
- SQLancer mostly tests the boolearn evaluator within DBMS
  - The WHERE and JOIN clauses are evaluated correctly or not
  - Reminds me of YinYang paper (PLDI ’20)

Example: SQLite3 Bug

t0

c0
0
1
2
NULL

```
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

TRUE

0
2

NULL was not contained in the result set!

https://sqlite.org/src/tktview/80256748471a01