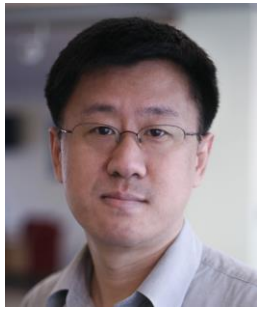
ETH zürich

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

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

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;
```

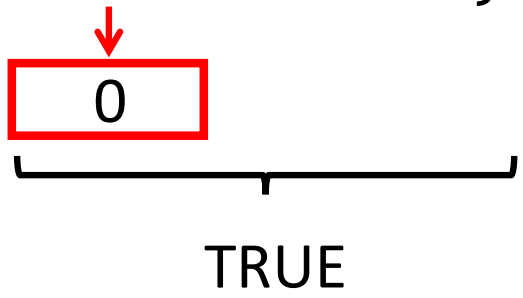
https://sqlite.org/src/tktview/80256748471a01

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;
```



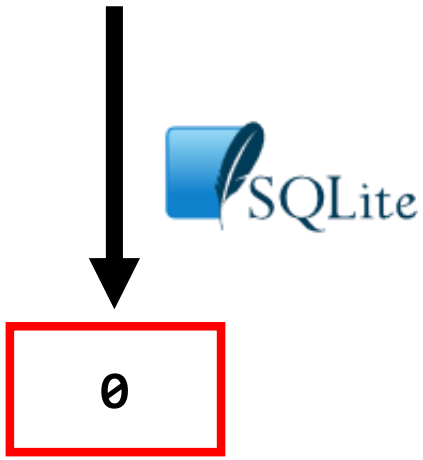
https://sqlite.org/src/tktview/80256748471a01

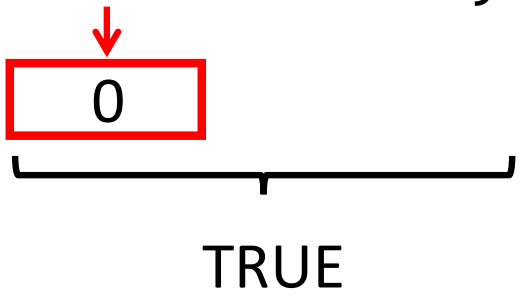
Example: SQLite3 Bug

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;
```





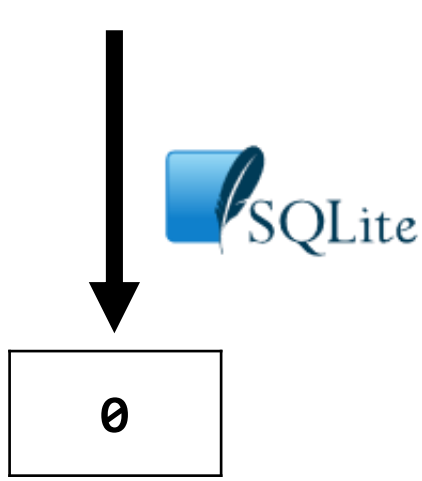
https://sqlite.org/src/tktview/80256748471a01

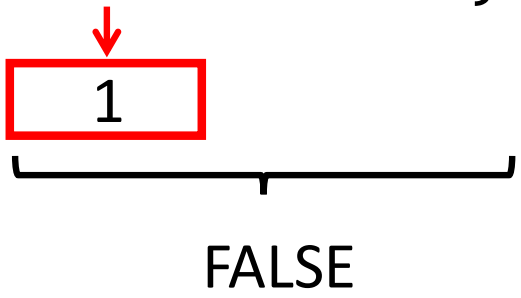
Example: SQLite3 Bug

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;
```





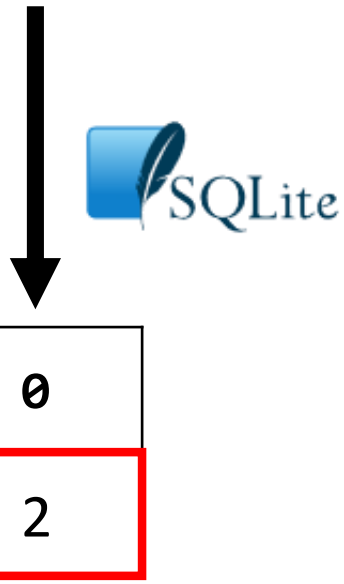
https://sqlite.org/src/tktview/80256748471a01

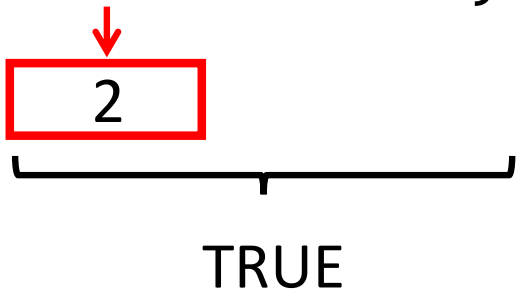
Example: SQLite3 Bug

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;
```





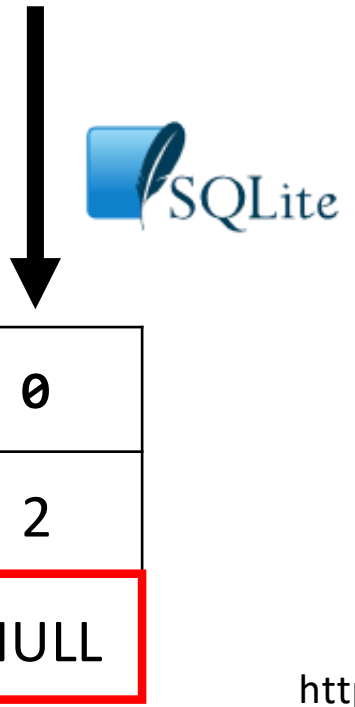
https://sqlite.org/src/tktview/80256748471a01

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

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

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

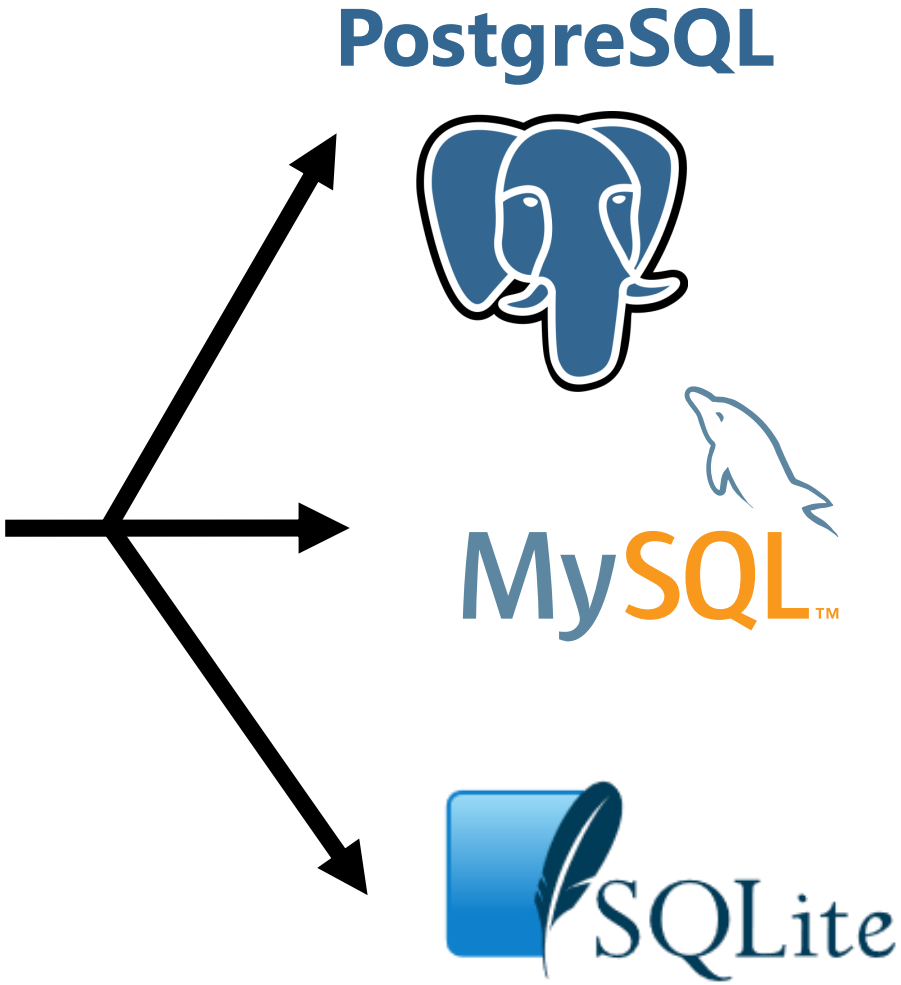
0
2

NULL was not contained in the result set!

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

Background: Differential Testing

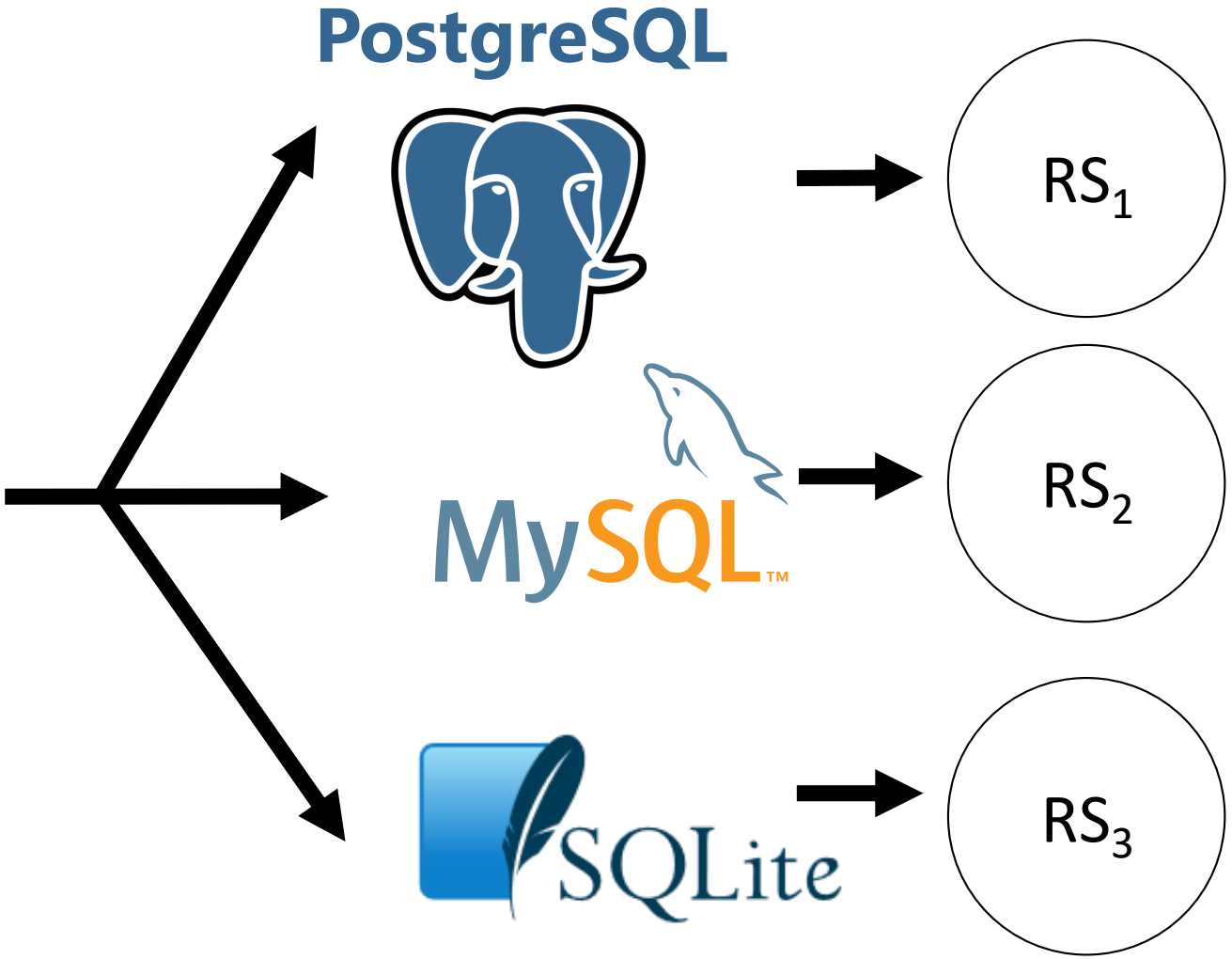
```
SELECT c0 FROM t0
WHERE t0.c0 IS NOT 1;
```



Massive Stochastic Testing of SQL by Slutz, 1998.

Background: Differential Testing

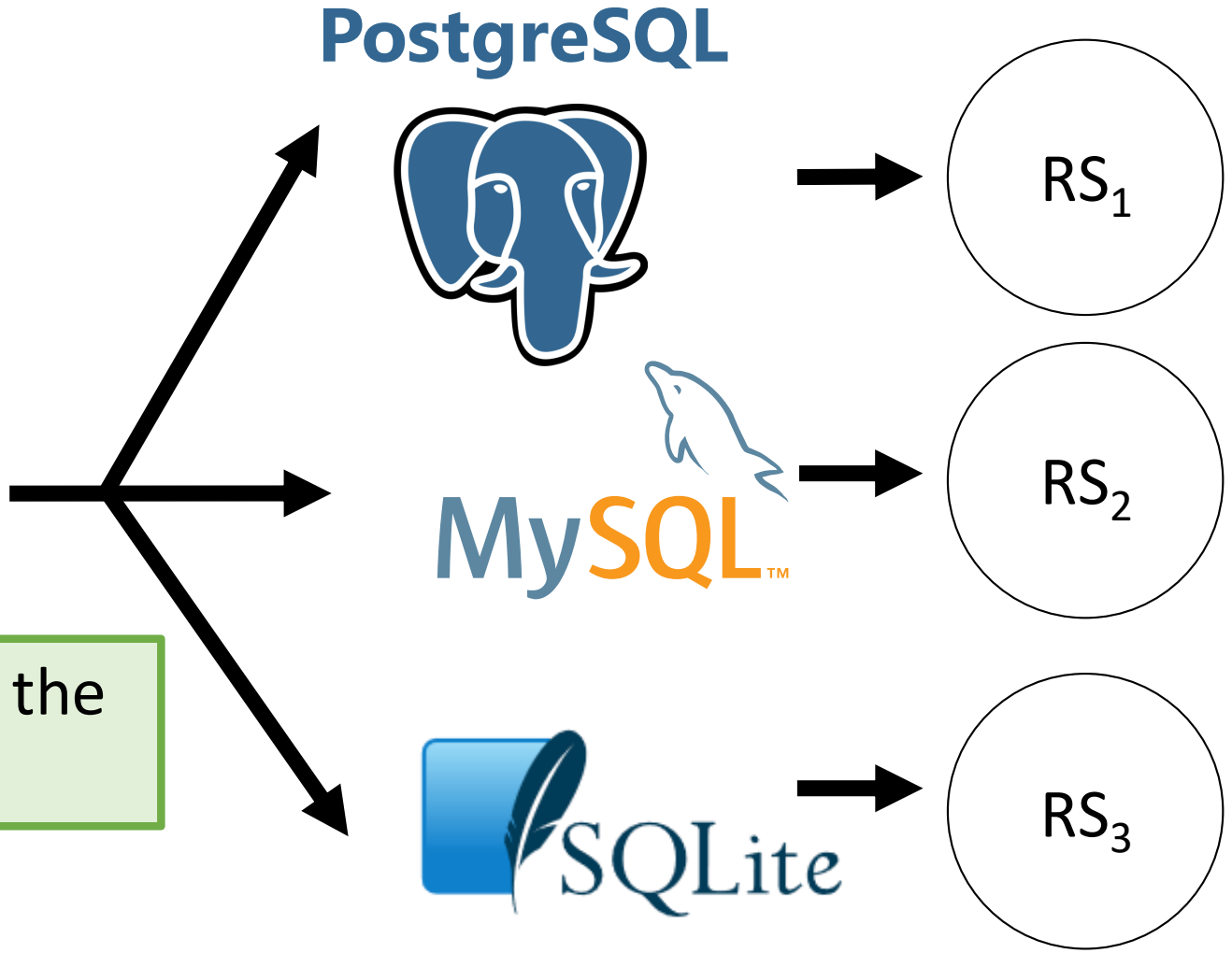
```
SELECT c0 FROM t0
WHERE t0.c0 IS NOT 1;
```



Massive Stochastic Testing of SQL by Slutz, 1998.

Background: Differential Testing

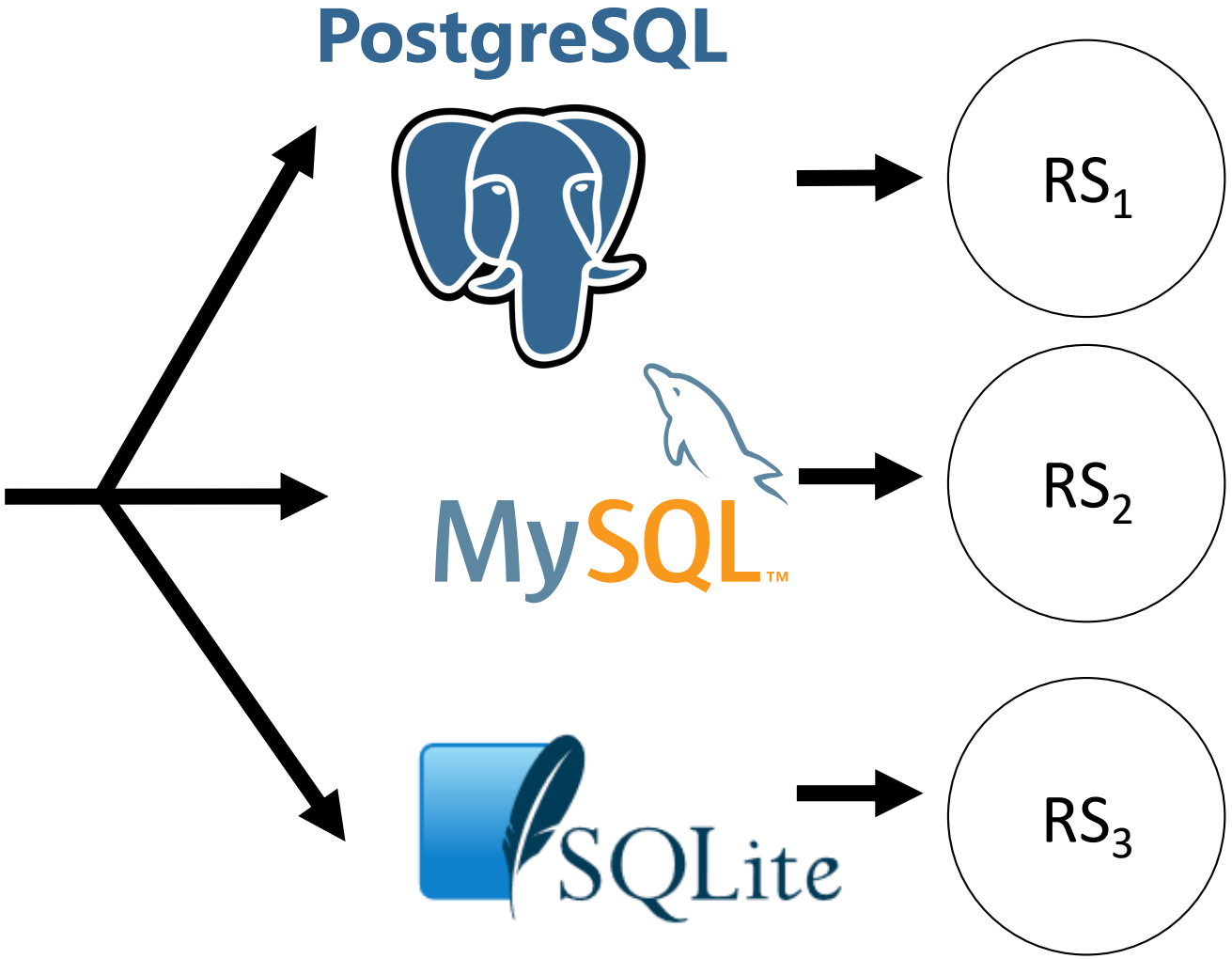
```
SELECT c0 FROM t0
WHERE t0.c0 IS NOT 1;
```



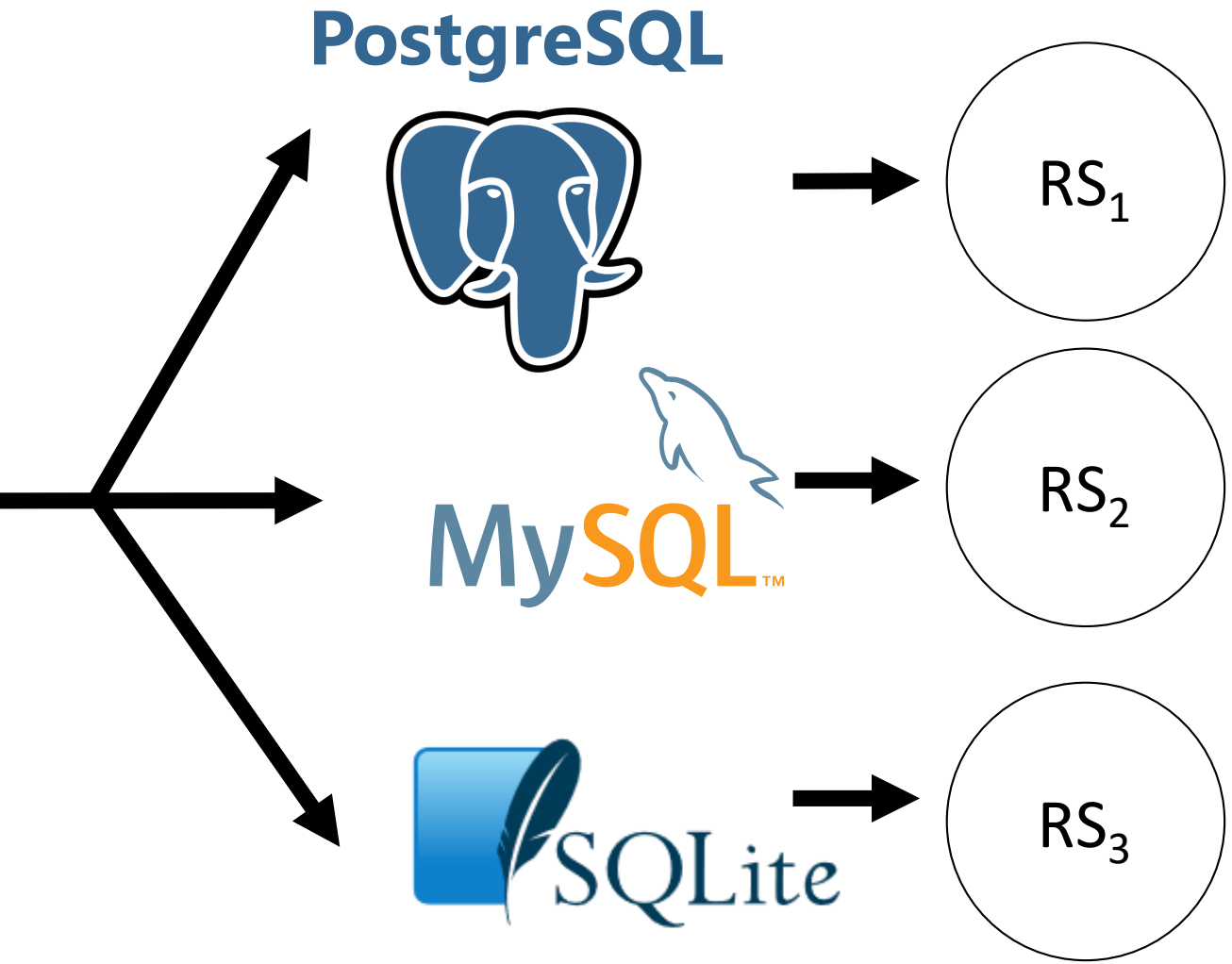
Check that all DBMSs compute the **same result ($RS_1 = RS_2 = RS_3$)**

Massive Stochastic Testing of SQL by Slutz, 1998.

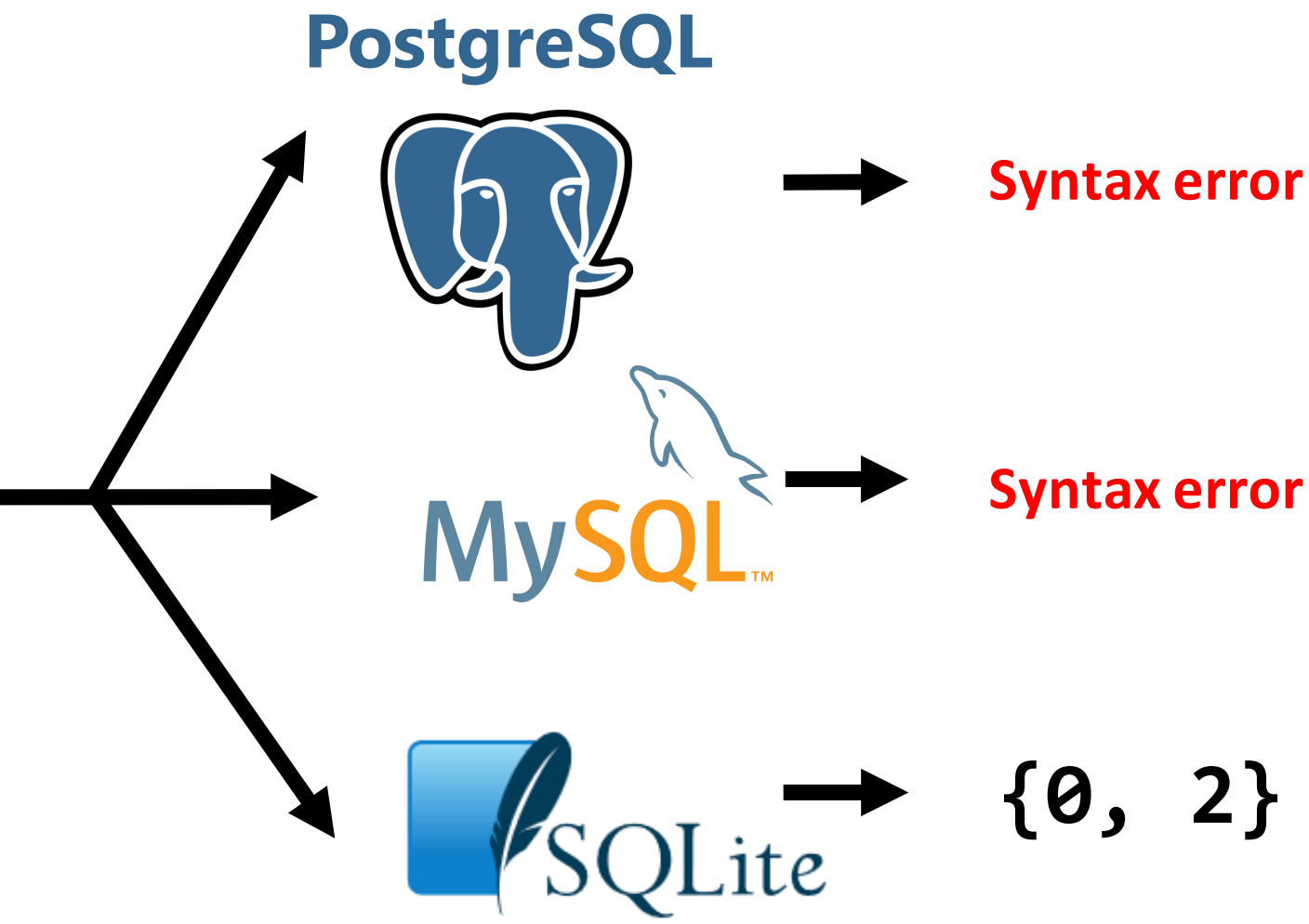
Background: Differential Testing



Background: Differential Testing



Background: Differential Testing



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

Idea: PQS

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

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

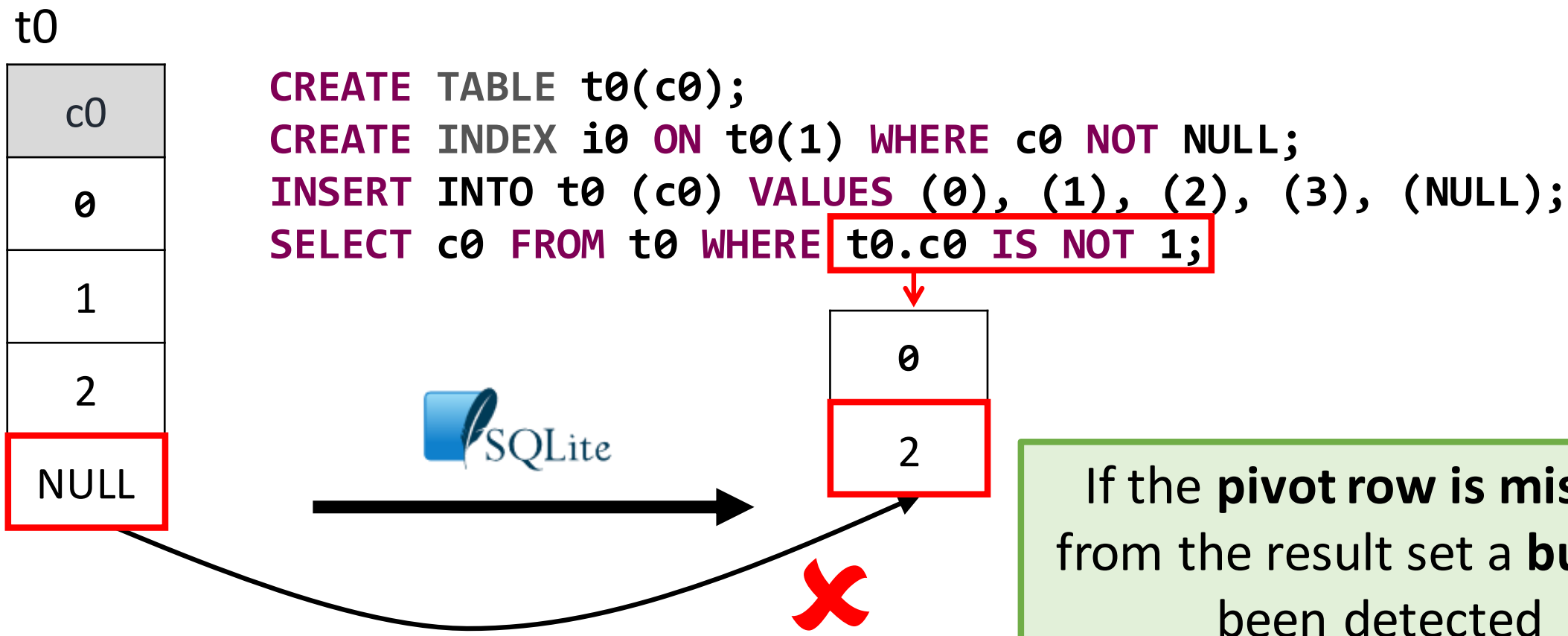
```
CREATE TABLE t0(c0);
CREATE INDEX i0 ON t0(1) WHERE c0 NOT NULL;
INSERT INTO t0 (c0) VALUES (0), (1), (2), (3), (NULL);
SELECT c0 FROM t0 WHERE t0.c0 IS NOT 1;
```

NULL

TRUE

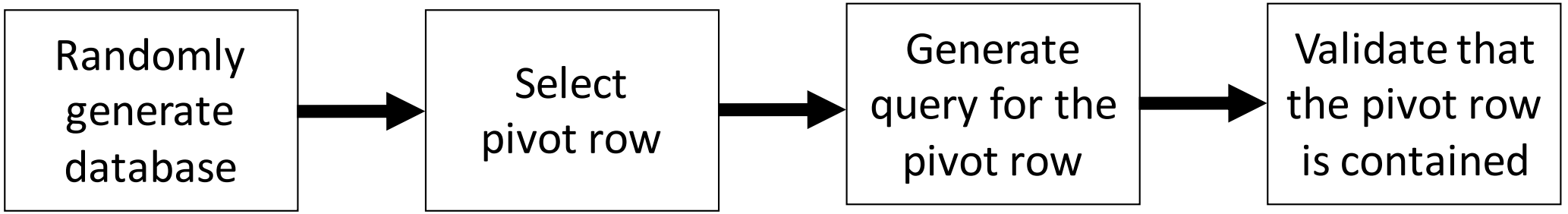
Generate a query that is guaranteed to at least fetch the pivot row

PQS Idea

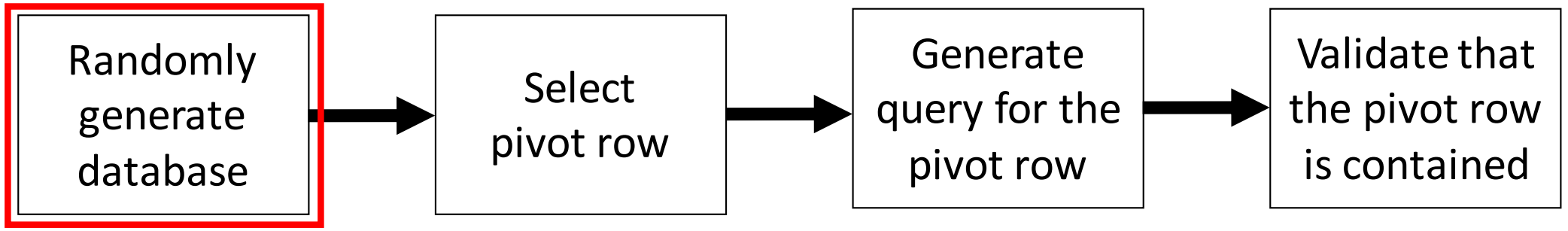


If the **pivot row is missing** from the result set a **bug** has been detected

Approach


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

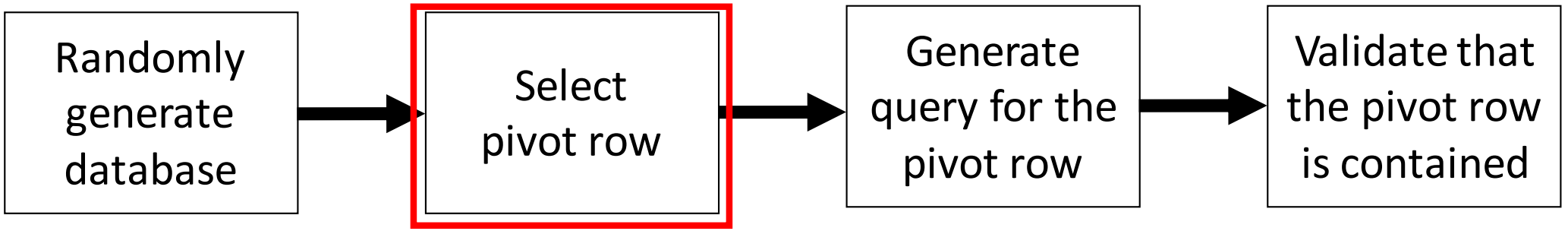
Approach




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

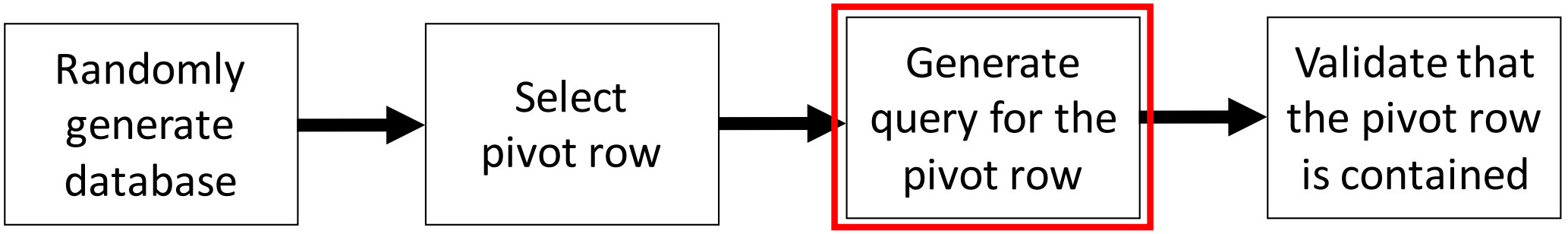
Statements are **heuristically generated** based on the DBMS' SQL dialect

Approach



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

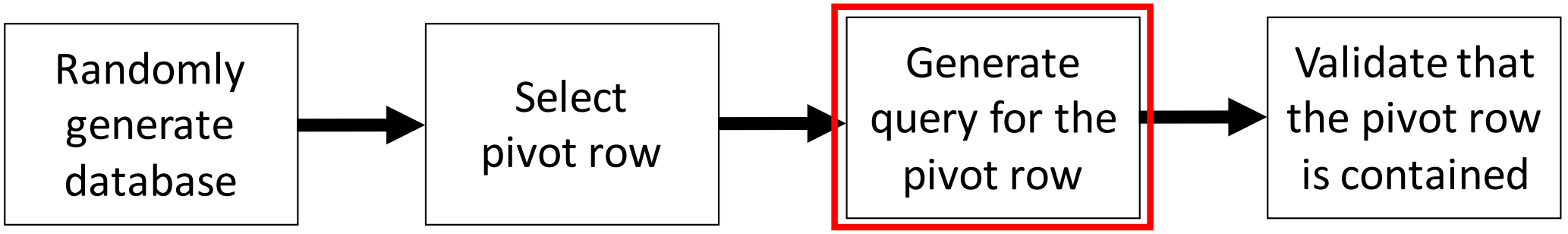
Approach

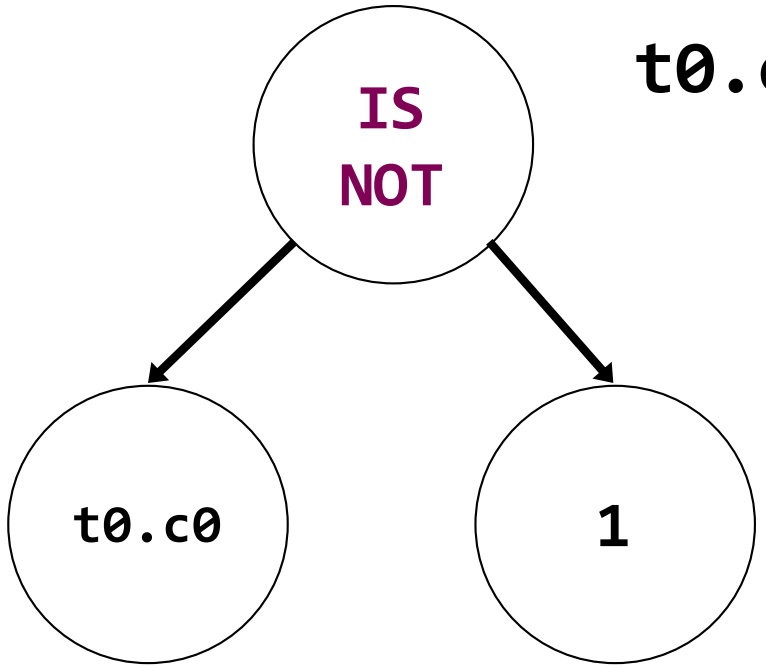



```
SELECT c0 FROM t0
WHERE
```

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

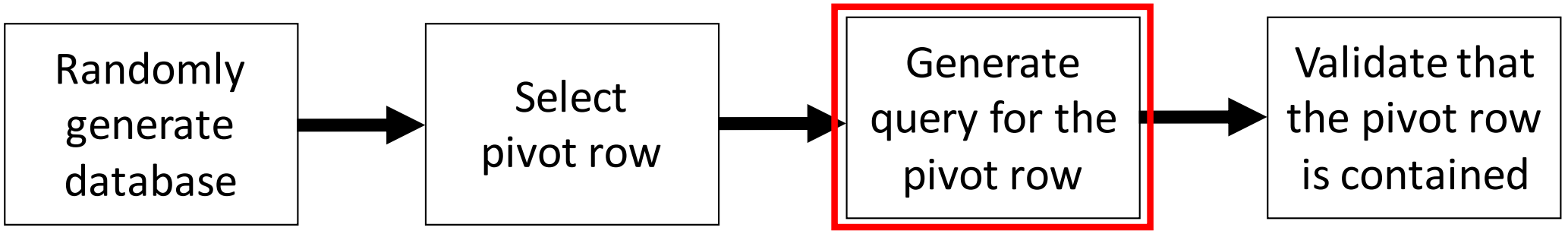
Random Expression Generation

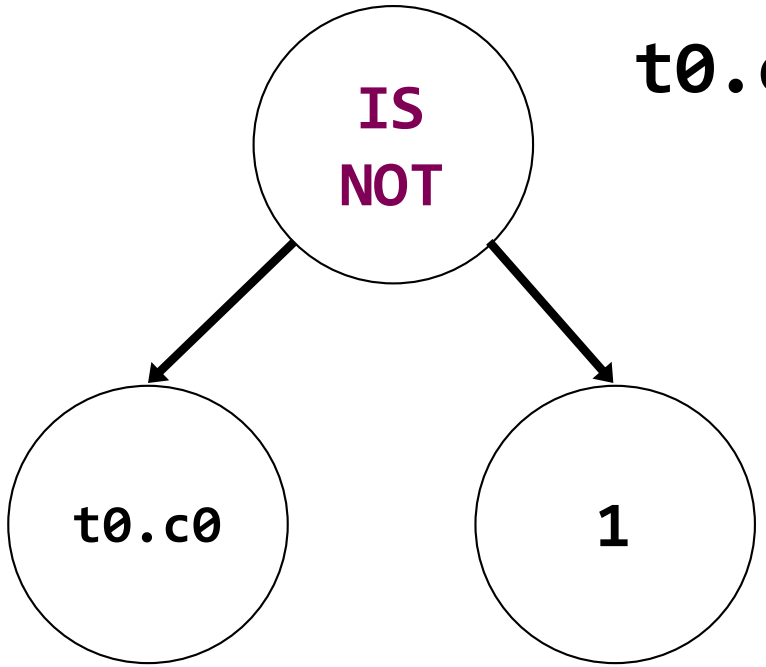





```
t0.c0 IS NOT 1;
```

Random Expression Generation

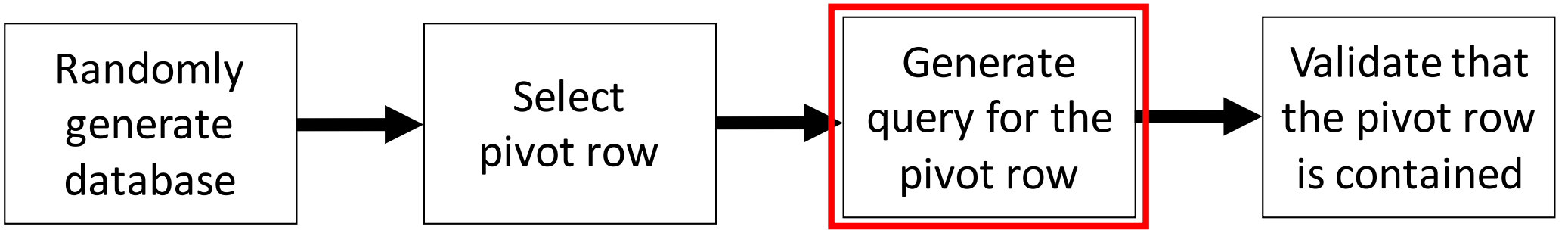





```
t0.c0 IS NOT 1;
```

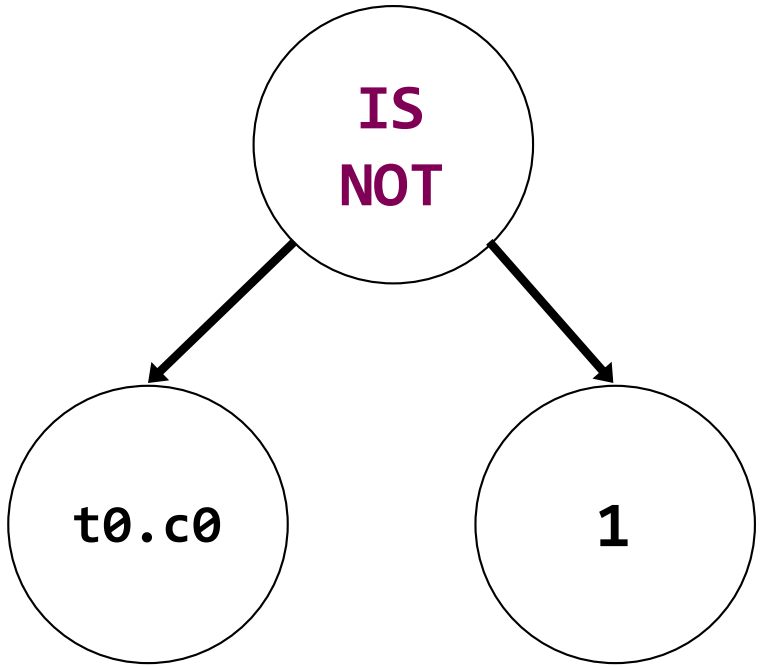
We implemented an **expression evaluator** for each node

Random Expression Generation



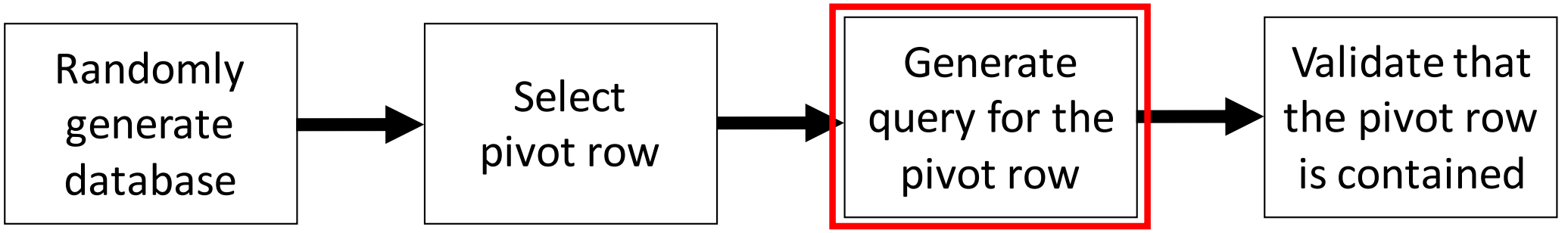
t0

c0
0
1
2
NULL



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

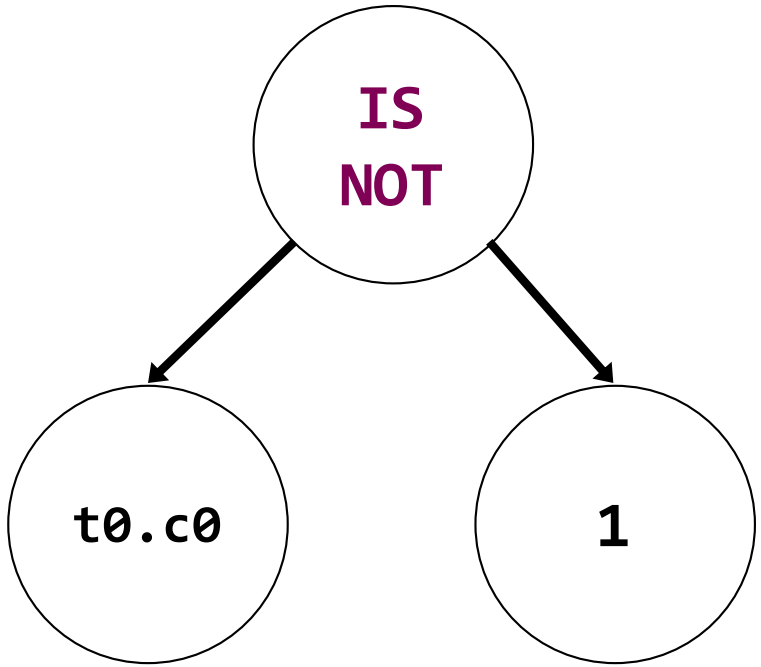
Random Expression Generation



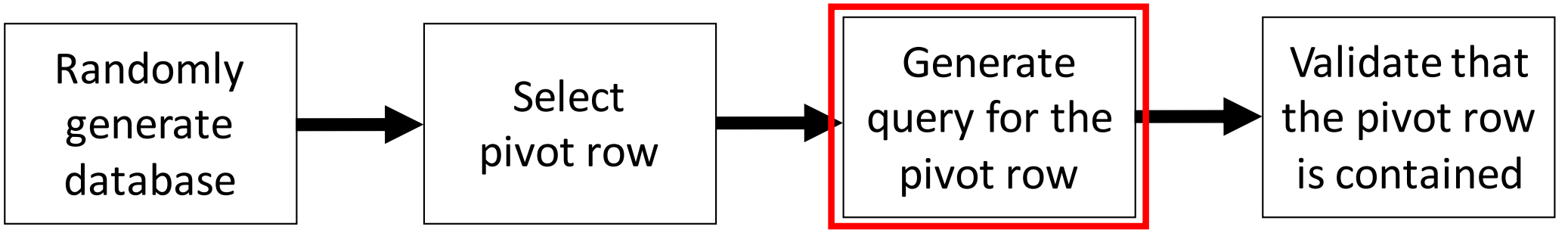
t0

c0
0
1
2
NULL

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



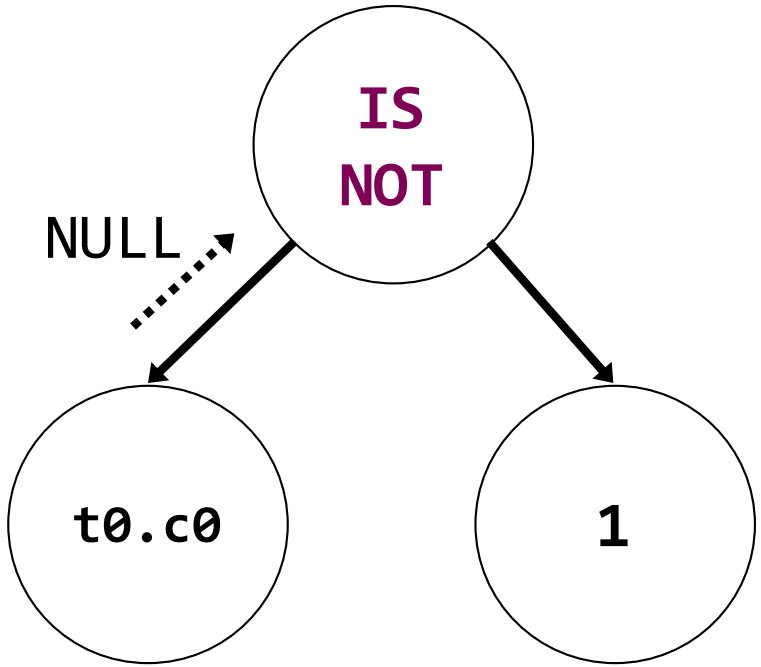
Random Expression Generation



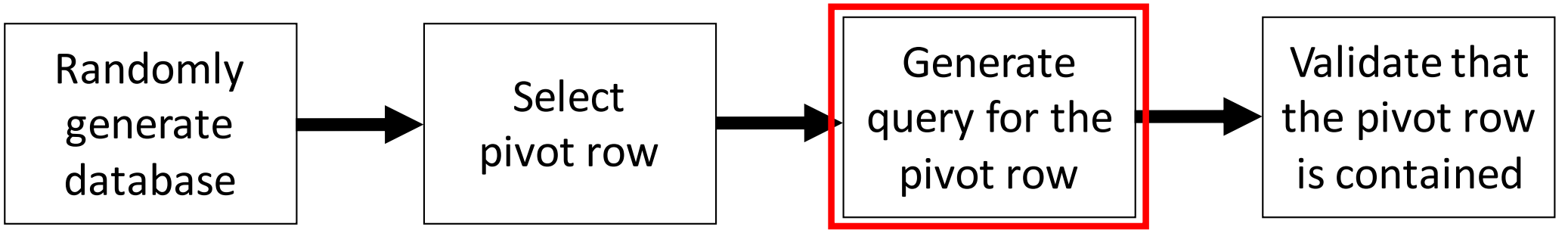
t0

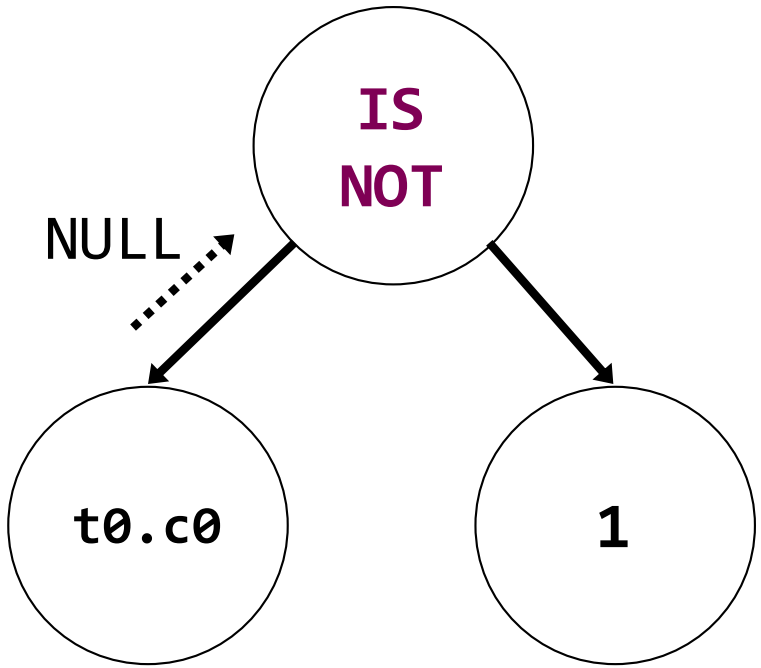
c0
0
1
2
NULL

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



Random Expression Generation



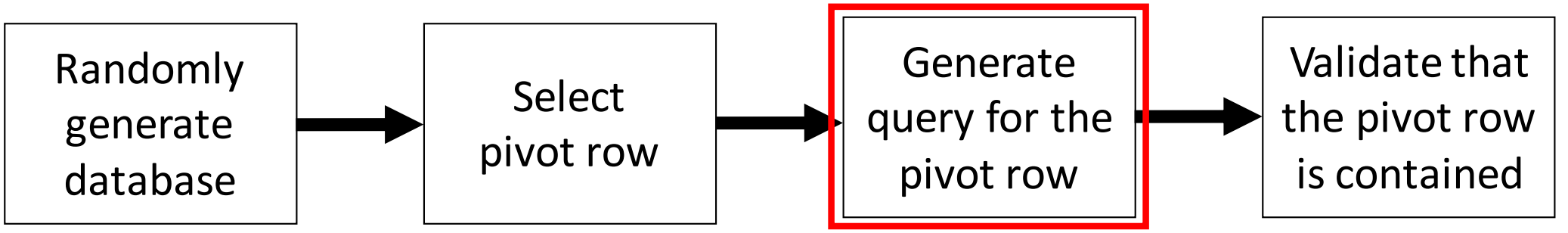


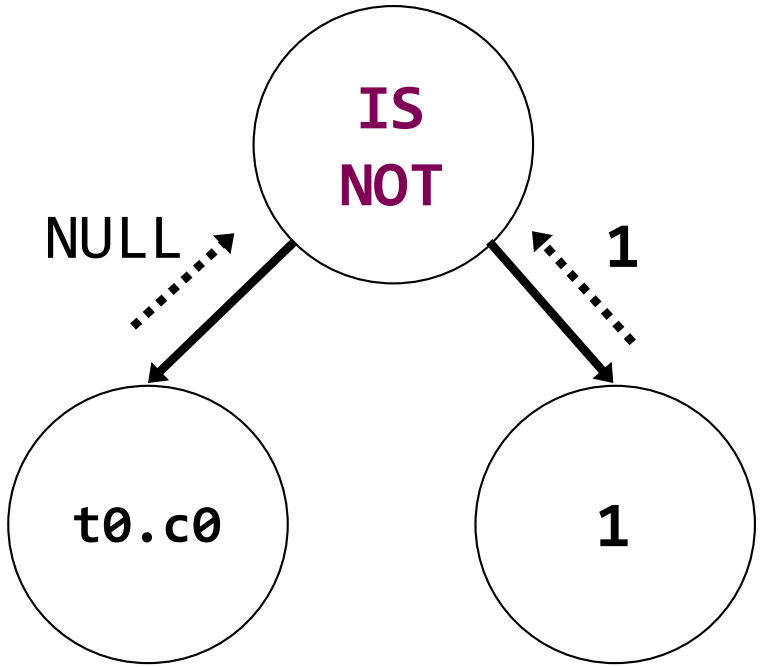
Constant nodes return their assigned **literal values**

t0

c0
0
1
2
NULL

Random Expression Generation



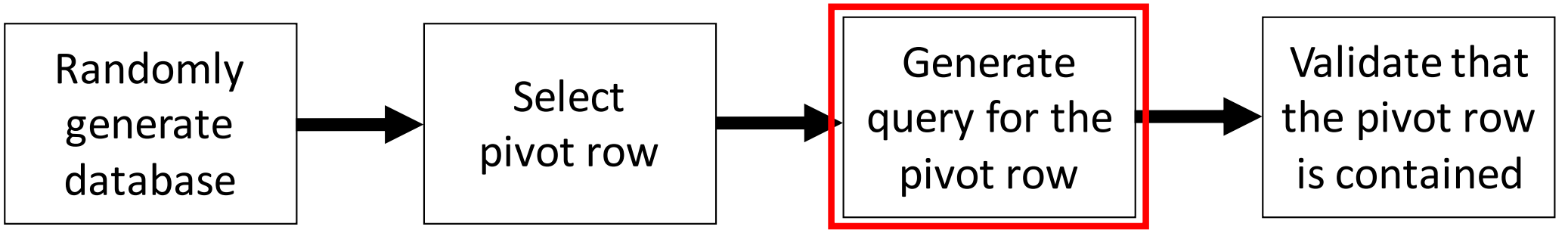


Constant nodes return their assigned **literal values**

t0

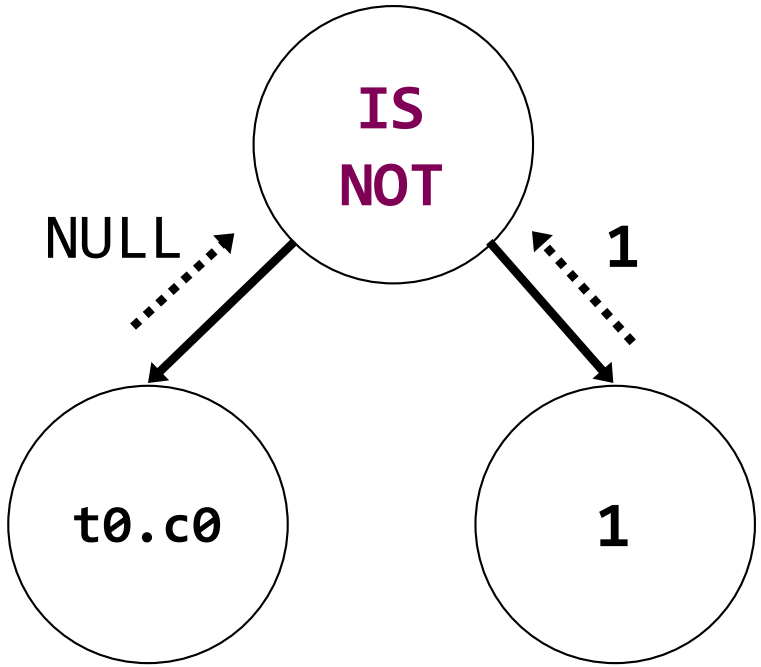
c0
0
1
2
NULL

Random Expression Generation



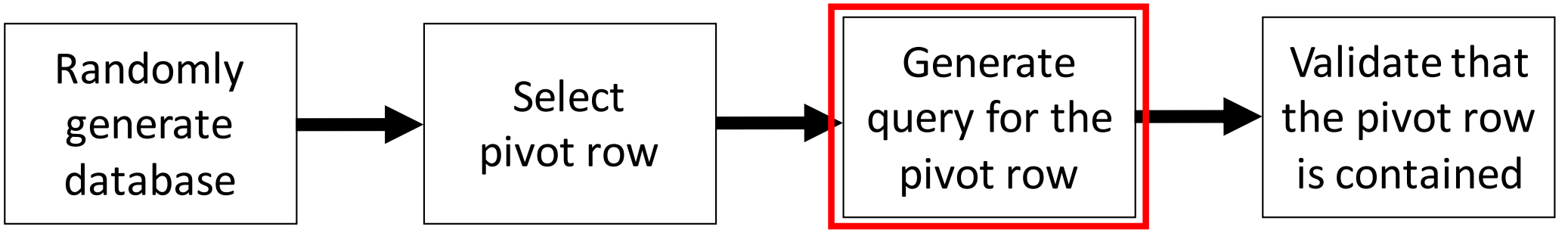
t0

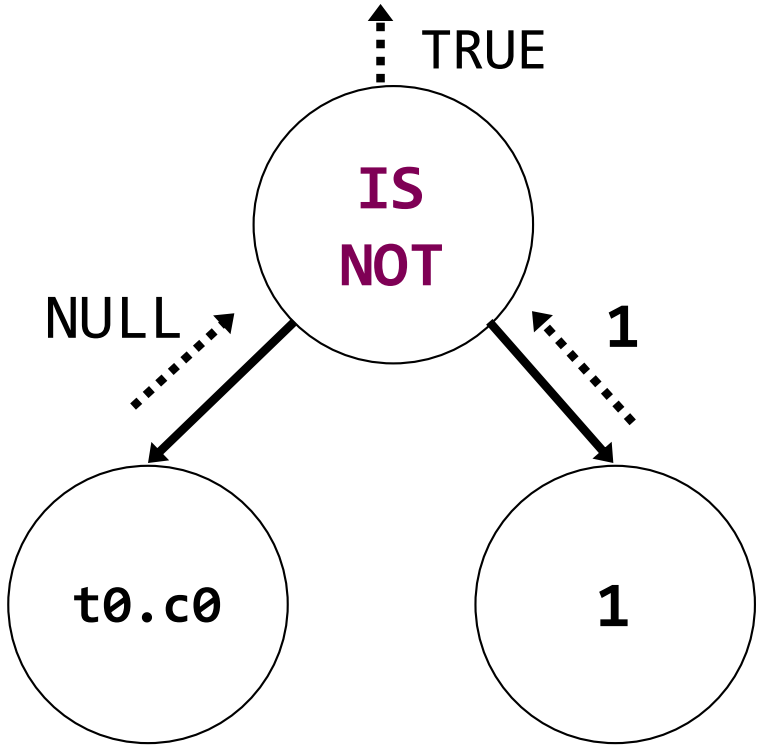
c0
0
1
2
NULL



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

Random Expression Generation



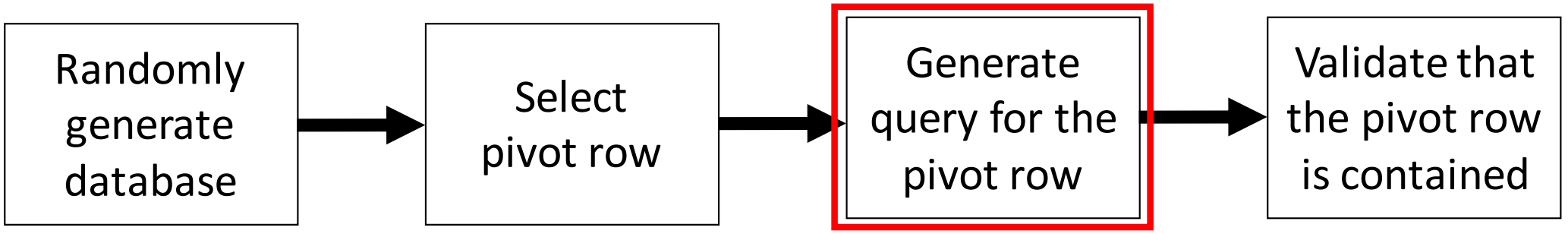


Compound nodes compute their result **based on their children**

t0

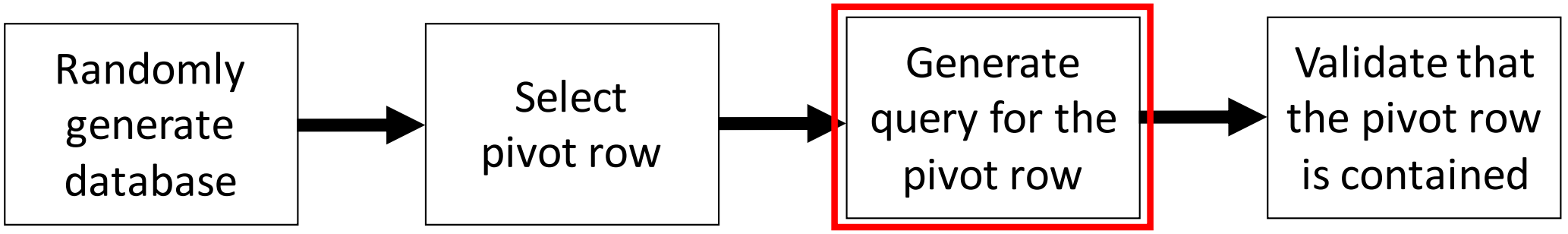
c0
0
1
2
NULL

Query Synthesis




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

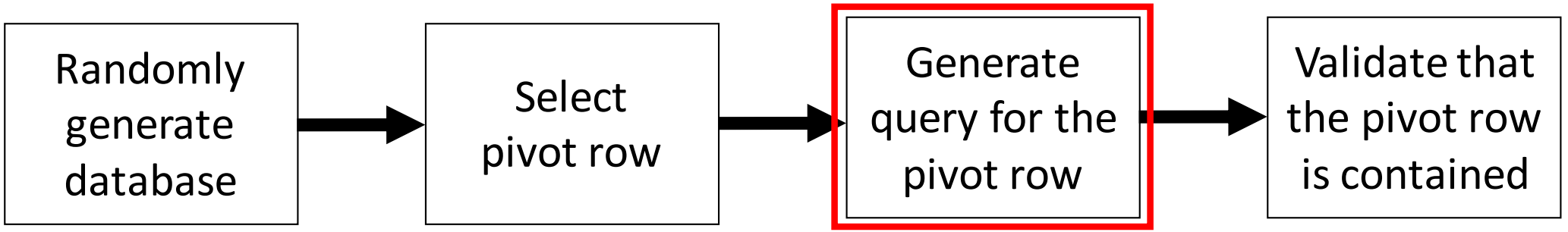
Query Synthesis




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

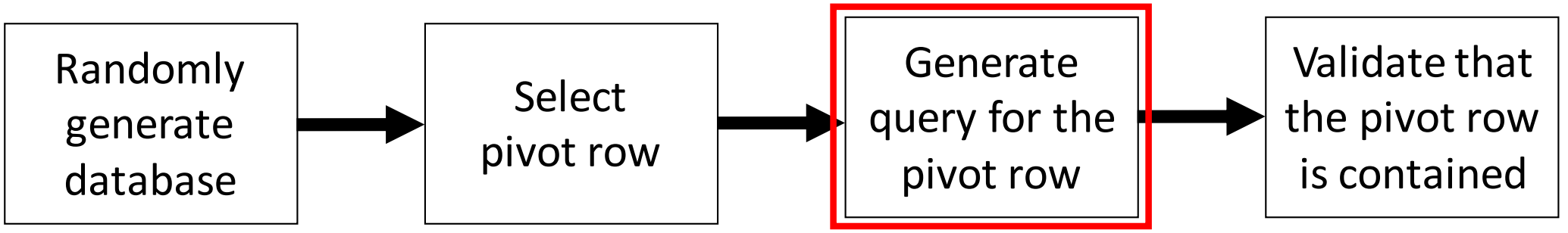
What if the expression **does not evaluate to TRUE**?

Random Expression Rectification




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

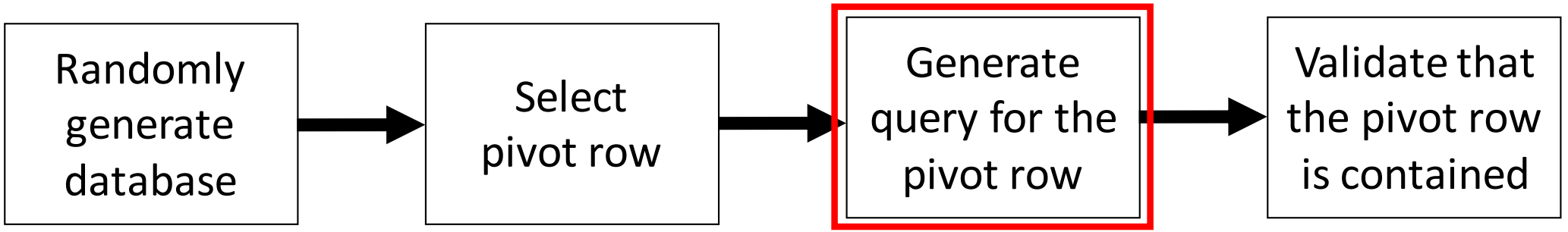
Random Expression Rectification




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

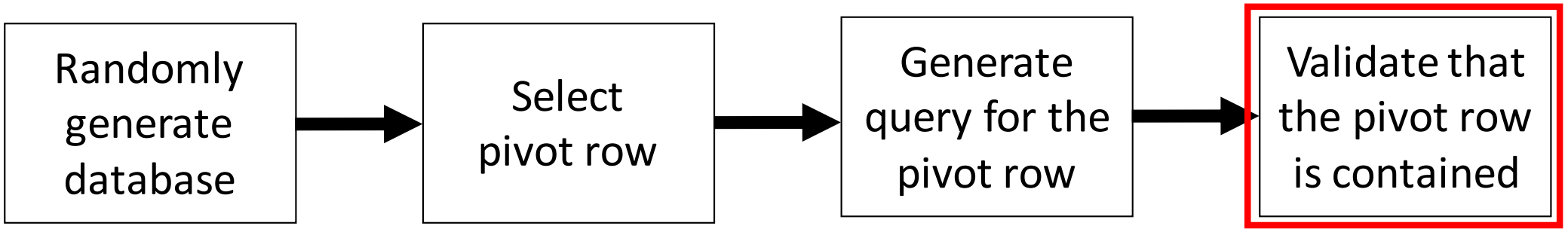
Alternatively, we could validate that the pivot row is expectedly **not fetched**

Random Expression Rectification



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

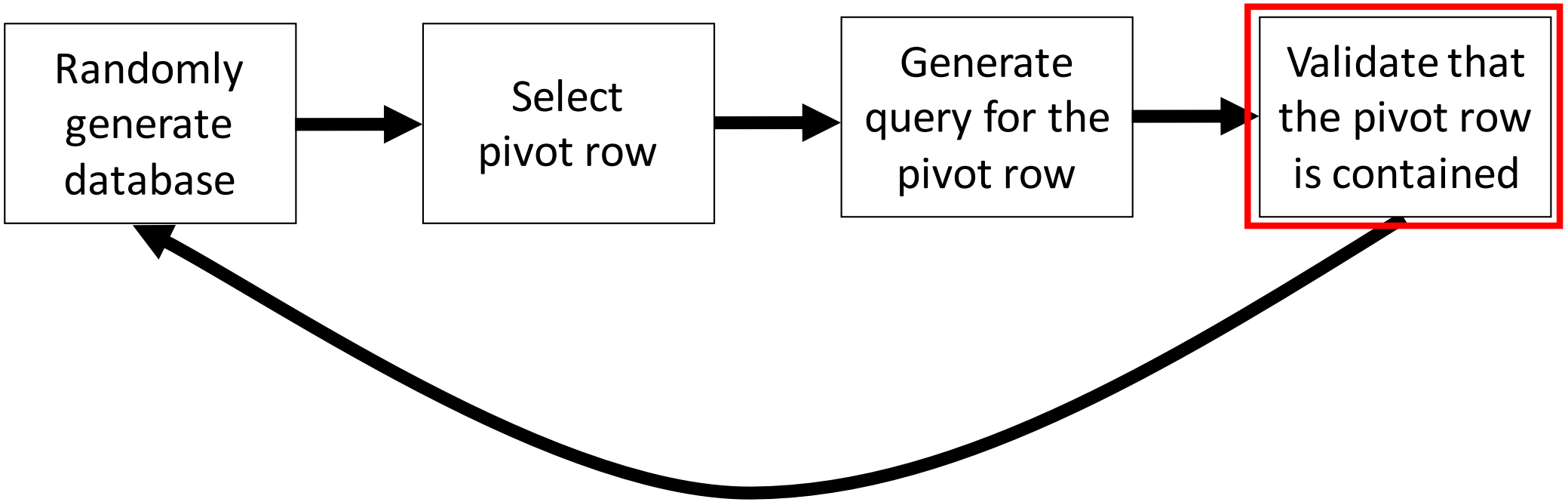
Approach



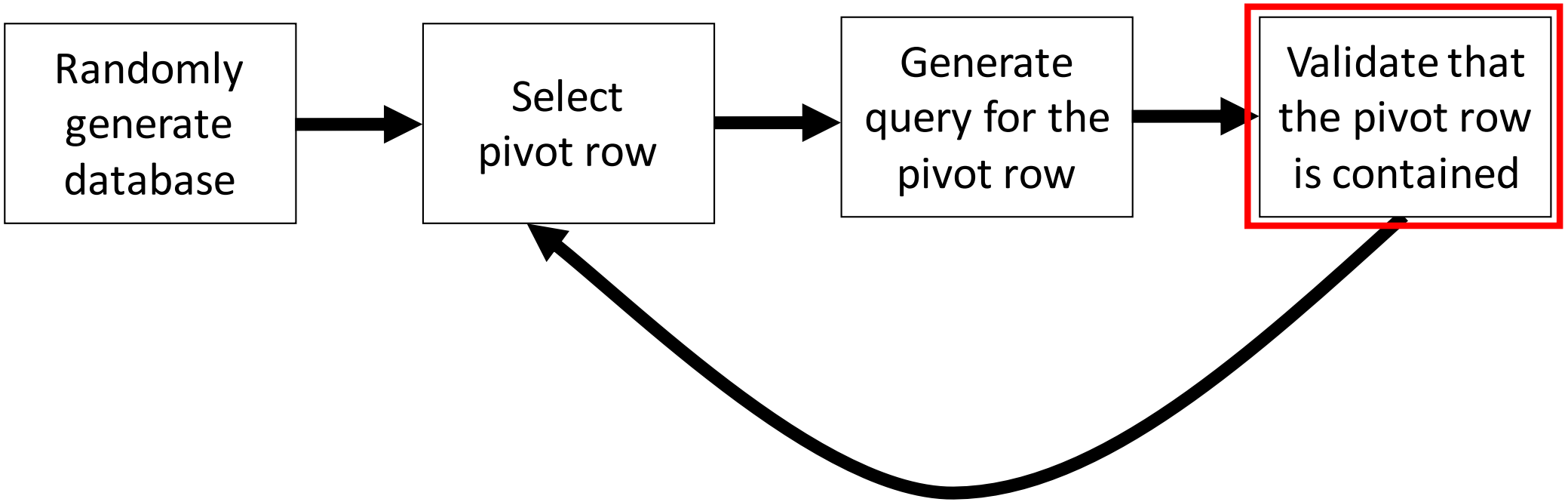

```
SELECT (NULL) INTERSECT
  SELECT c0 FROM t0 WHERE NULL IS NOT 1;
```

Rely on the DBMS to check whether the row is contained

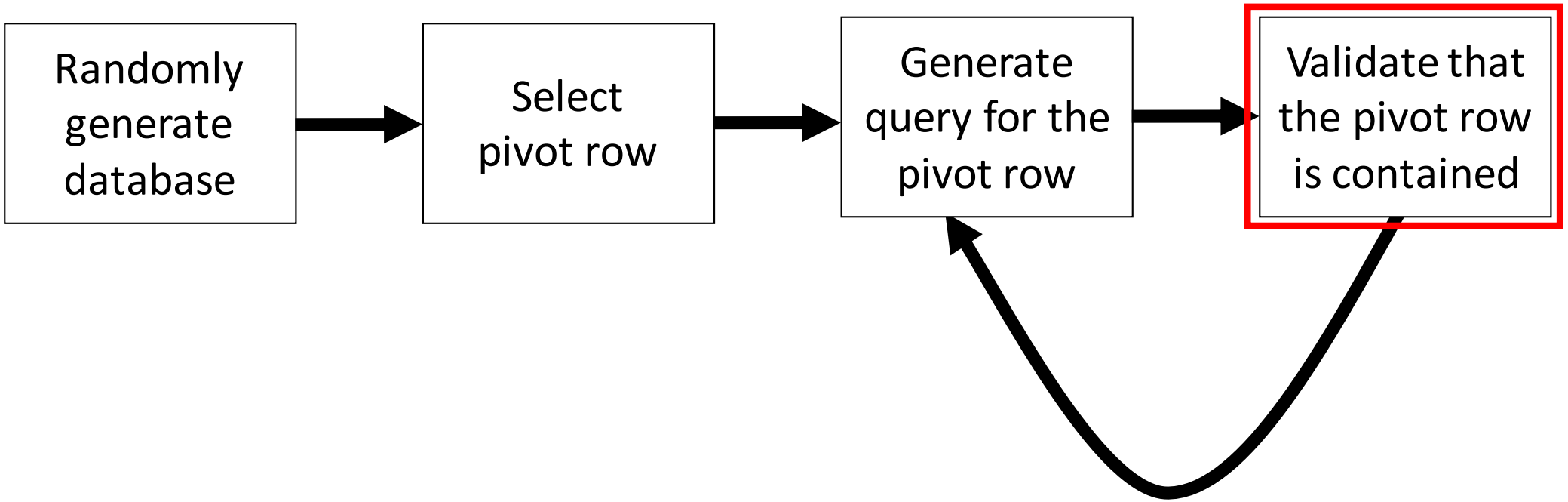
Approach


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

Approach

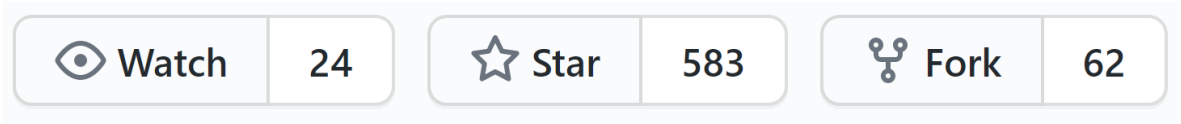

Randomly generate database
Select pivot row
Generate query for the pivot row
Validate that the pivot row is contained

Approach



We generate 100,000 queries for each generated database

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

Bugs Overview

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

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

Oracles

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

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

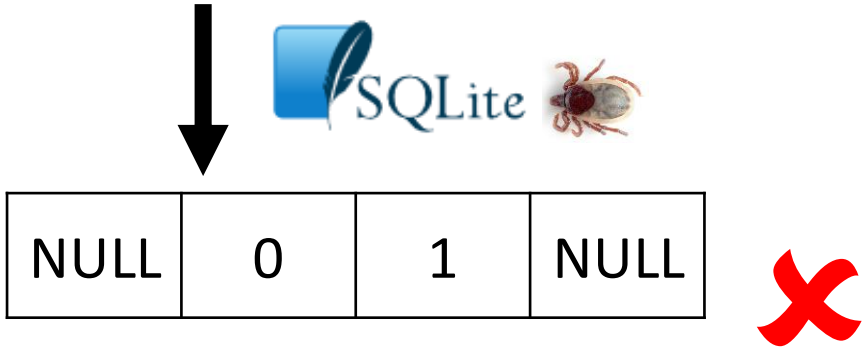
```
CREATE TABLE t1(c1, c2, c3, c4, PRIMARY KEY (c4, c3));
INSERT INTO t1(c3) VALUES (0), (0), (0), (0), (0), (0),
       (0), (0), (0), (0), (NULL), (1), (0);
UPDATE t1 SET c2 = 0;
INSERT INTO t1(c1) VALUES (0), (0), (NULL), (0), (0);
ANALYZE t1;
UPDATE t1 SET c3 = 1;
SELECT DISTINCT * FROM t1 WHERE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

```
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

```
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;
```

NULL	0	1	NULL

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;
```

NULL	0	1	NULL
0	NULL	1	NULL
NULL	NULL	1	NULL

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;
```

NULL	0	1	NULL
0	NULL	1	NULL
NULL	NULL	1	NULL

The bug was classified as "Severe" and quickly fixed

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

Result: Bug in PostgreSQL

t0

c0	c1
0	0

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

t1

c0	c1

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

Result: Bug in PostgreSQL

t0

c0	c1
0	0
0	1

```
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);
```

t1

c0	c1
0	1

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

Result: Bug in PostgreSQL

t0

c0	c1
0	0
0	1

```
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);
```

t1

c0	c1
0	1

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

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

Result: Bug in PostgreSQL

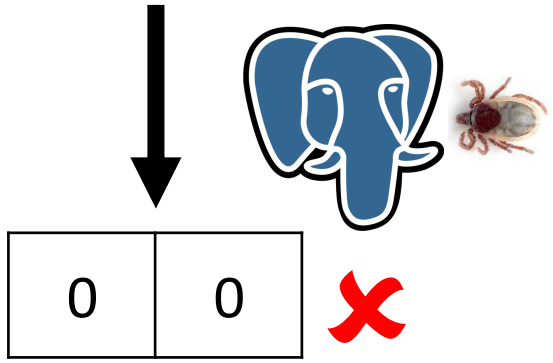
t0

c0	c1
0	0
0	1

t1

c0	c1
0	1

```
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;
```



0	0

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

Result: Bug in PostgreSQL

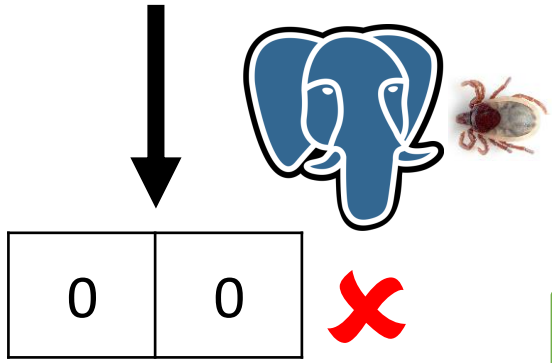
t0

c0	c1
0	0
0	1

t1

c0	c1
0	1

```
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;
```



0	0

An optimization incorrectly simplified the GROUP BY clause

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

Result: Bug in PostgreSQL

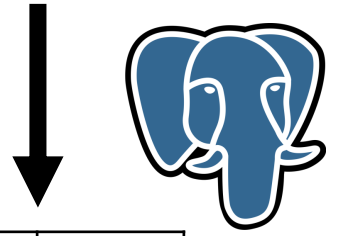
t0

c0	c1
0	0
0	1

t1

c0	c1
0	1

```
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;
```



0	0
0	1

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

Result: Bug in MySQL

t0

c0
1

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

https://bugs.mysql.com/bug.php?id=95900

Result: Bug in MySQL

t0

c0
1

```
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

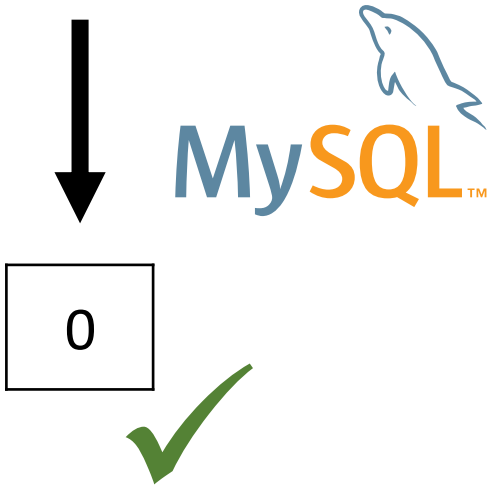
https://bugs.mysql.com/bug.php?id=95900

Result: Bug in MySQL

t0

c0
1

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



https://bugs.mysql.com/bug.php?id=95900

Oracles

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

Error bugs are due to unexpected (internal) errors

Example: SQLite3 Bug

```
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);
```

Database disk image is malformed

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.

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

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)

>150 bugs

Ternary Logic Query Partitioning (TLP)

Overview

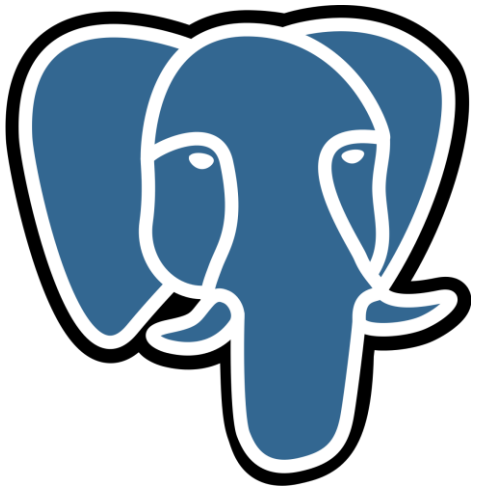
Pivoted Query Synthesis (PQS)

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

Non-optimizing Reference Engine Construction (NoREC)

Ternary Logic Query Partitioning (TLP)

SQLancer: Supported DBMSs



Summary

@RiggerManuel manuel.rigger@inf.ethz.ch

Goal: Detect logic bugs

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), (3), (NULL);
SELECT c0 FROM t0 WHERE t0.c0 IS NOT 1;
```

SQLite

NULL

TRUE

0
2

NULL was not contained in the result set!

ETH zürich 12

PQS randomly selects a 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;
```

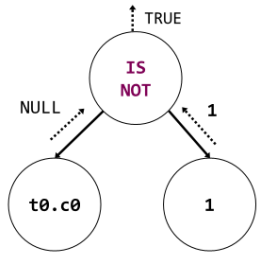
Pivot row

Validate the result set based on **one randomly-selected row**

ETH zürich 20

Rectify a random expression

Random Expression Generation



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

t0

c0
0
1
2
3
NULL

ETH zürich 31

Evaluation: Close to 100 bugs in DBMSs

Bugs Overview

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

96 bugs were unique, previously unknown ones

ETH zürich 37

SQLANCER