Using SQLancer to test ClickHouse and other database systems

Manuel Rigger

ETH zürich

@RiggerManuel

Ilya Yatsishin

Yandex

qoega

Plan

- What is ClickHous and why do we need good testing?
- How do we test ClickHouse and what problems we have to solve?
- What is SQLancer and what are the ideas behind them?
- How to add yet another DBMS support to SQLancer?

Open Source analytical DBMS for BigData with SQL interface.

- Blazingly fast
- Scalable
- Fault tolerant

https://github.com/ClickHouse/clickhouse

Why do we need good CI?

In 2020:

361	4081	261	11	<15
Contributors	Merged Pull Requests	New Features	Releases	Core Team

Pull Requests exponential growth

https://gh-api.clickhouse.tech | play

```
SELECT toStartOfQuarter(created_at) as quarter, uniq(number) as pr, bar(pr, 0, 20000)
FROM github_events
WHERE repo_name = 'ClickHouse/ClickHouse' OR repo_name = 'yandex/ClickHouse'
GROUP BY toStartOfQuarter(created_at)
```

Run (Ctrl/Cmd+Enter) ✔ Elapsed: 0.064 sec, read 4255417 rows.

quarter	pr	bar(uniq(number), 0, 20000)
2016-04-01	39	
2016-07-01	89	
2016-10-01	187	
2017-01-01	378	
2017-04-01	319	
2017-07-01	417	
2017-10-01	510	
2018-01-01	533	
2018-04-01	503	
2018-07-01	861	
2018-10-01	890	
2019-01-01	1122	
2019-04-01	1169	
2019-07-01	1641	
2019-10-01	2062	
2020-01-01	1916	
2020-04-01	2814	
2020-07-01	4036	
2020-10-01	3914	
2021-01-01	3492	

All GitHub data available in ClickHouse

https://gh-api.clickhouse.tech | play

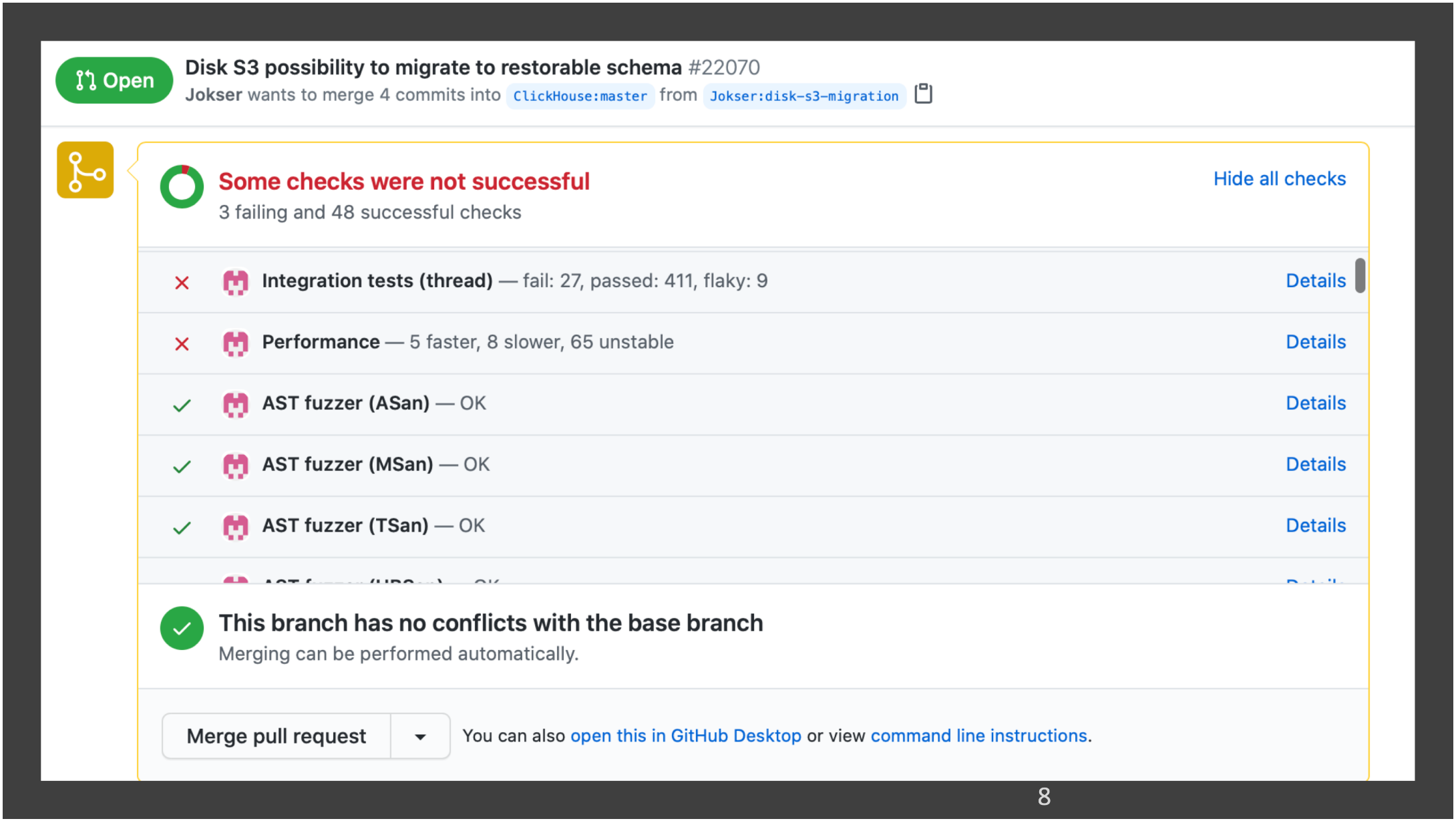
```
SELECT
    repo_name,
    uniq(actor_login),
    uniq(number) as pull_requests,
    sum(merged) as merged_prs,
    sum(state='closed' and not merged) as closed_prs,
    round(sum(merged) / uniq(number) * 100) AS merged_share
FROM github_events
WHERE repo_name = 'ClickHouse/ClickHouse' and created_at BETWEEN '2020-01-01 00:00:00' AND '2021-01-01 00:00:00'
    AND event_type = 'PullRequestEvent' AND base_ref = 'master'
GROUP BY repo_name
ORDER BY uniq(actor_login) DESC
```

Run (Ctrl/Cmd+Enter) ✔ Elapsed: 0.044 sec, read 156163 rows.

repo_name	uniq(actor_login)	pull_requests	merged_prs	closed_prs	merged_share
ClickHouse/ClickHouse	361	4711	4021	578	85

https://gh.clickhouse.tech/explorer/
https://gh-api.clickhouse.tech/play

How is ClickHouse tested?


Open
Disk S3 possibility to migrate to restorable schema #22070
Jokser wants to merge 4 commits into ClickHouse:master from Jokser:disk-s3-migration
Some checks were not successful
3 failing and 48 successful checks
Hide all checks
Integration tests (thread) — fail: 27, passed: 411, flaky: 9
Details
Performance — 5 faster, 8 slower, 65 unstable
Details
AST fuzzer (ASan) — OK
Details
AST fuzzer (MSan) — OK
Details
AST fuzzer (TSan) — OK
Details
This branch has no conflicts with the base branch
Merging can be performed automatically.
Merge pull request
You can also open this in GitHub Desktop or view command line instructions.

ClickHouse Testing

- Style
- Unit
- Functional
- Integrational
- Performance
- Stress
- Static analysis (clang-tidy, PVSStudio)
- Compatibility with OS versions

- Flaky check for new or changed tests
- All tests are run in with sanitizers (address, memory, thread, undefined behavior)
- Thread fuzzing (switch running threads randomly)
- Coverage
- Fuzzing

https://rc1a-ity5agjmuhyu6nu9.mdb.yandexcloud.net:8443

```
SELECT DISTINCT check_name
FROM "gh-data".checks
WHERE commit_sha = '0675f9403cbd8bf17c0d114113dbafb8f911be69'
```

Run (Ctrl/Cmd+Enter) ✔

check_
Compatibility check
Docs release
Fast test
Unit tests release gcc
Yandex synchronization (only for Yandex employees)
Functional stateful tests (address)
Functional stateful tests (debug)
Functional stateful tests (memory)
Functional stateful tests (release)
Functional stateful tests (release, DatabaseOrdinary)
Functional stateful tests (thread)
Functional stateful tests (ubsan)
Functional stateless tests (release, wide parts enabled)
Functional stateless tests (ubsan)
PVS check
Push to dockerhub
Push to dockerhub
SQLancer test
Split build smoke test
Stress test (debug)
Style Check
Functional stateless tests (ANTLR debug)
Unit tests ASAN
Unit tests release clang
Functional stateless tests (address)
Functional stateless tests (debug)
Functional stateless tests (memory)
Functional stateless tests (release)
Functional stateless tests (release, DatabaseOrdinary)
Functional stateless tests (thread)
Integration tests (memory)
Integration tests (release)
Stress test (address)
Stress test (memory)
Stress test (thread)
Stress test (undefined)
Testflows check
Unit tests MSAN
Unit tests TSAN
Unit tests UBSAN
Integration tests (asan)
Integration tests (thread)

100K tests is not enough

https://rc1a-ity5agjmuhyu6nu9.mdb.yandexcloud.net:8443	qoega	••••••••

```
SELECT COUNT(DISTINCT check_name), COUNT(DISTINCT (check_name, test_name))
FROM "gh-data".checks
WHERE commit_sha = '0675f9403cbd8bf17c0d114113dbafb8f911be69'
```

Run (Ctrl/Cmd+Elapsed: 2.268 sec, read 315706649 rows.

uniqExact(check_name)	uniqExact(tuple(check_name, test_name))
41	106714

Fuzzing

libFuzzer to test generic data inputs – formats, schema etc.

Thread Fuzzer – randomly switch threads to trigger races and dead locks

- https://presentations.clickhouse.tech/cpp_siberia_2021/

AST Fuzzer – mutate queries on AST level.

- Use SQL queries from all tests as input. Mix them.
- High level mutations. Change query settings
- https://clickhouse.tech/blog/en/2021/fuzzing-clickhouse/

Test development steps

Unit,
Functional,
Integration,
Stress,
Performance
etc.

Find bugs earlier.
Sanitizers:

- Address
- Memory
- Undefined Behavior
- Thread

Improve coverage.

???

In search for correctness

Test cases for correctness require developer or QA specialist to write it manually

Additional tests mostly check if DBMS crashes or stops working

It is hard to find reference system to validate results with:

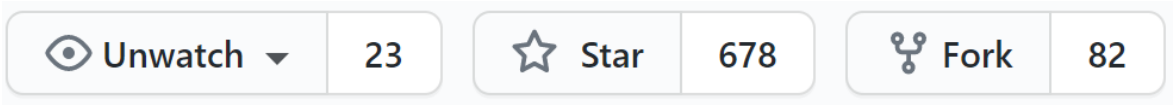
- SQL syntax differs between systems
- No SQL conformance testing suite
- All SQL standard extensions can't be tested this way

How we can check correctness?

- SQL has some invariants and basic assumptions
- DBMS can help us!

There should be some solutions that already exploit that

SQLancer



https://github.com/sqlancer

SQLancer is an effective and widely-used
automatic testing tool to find **logic bugs** in **DBMSs**

Facts

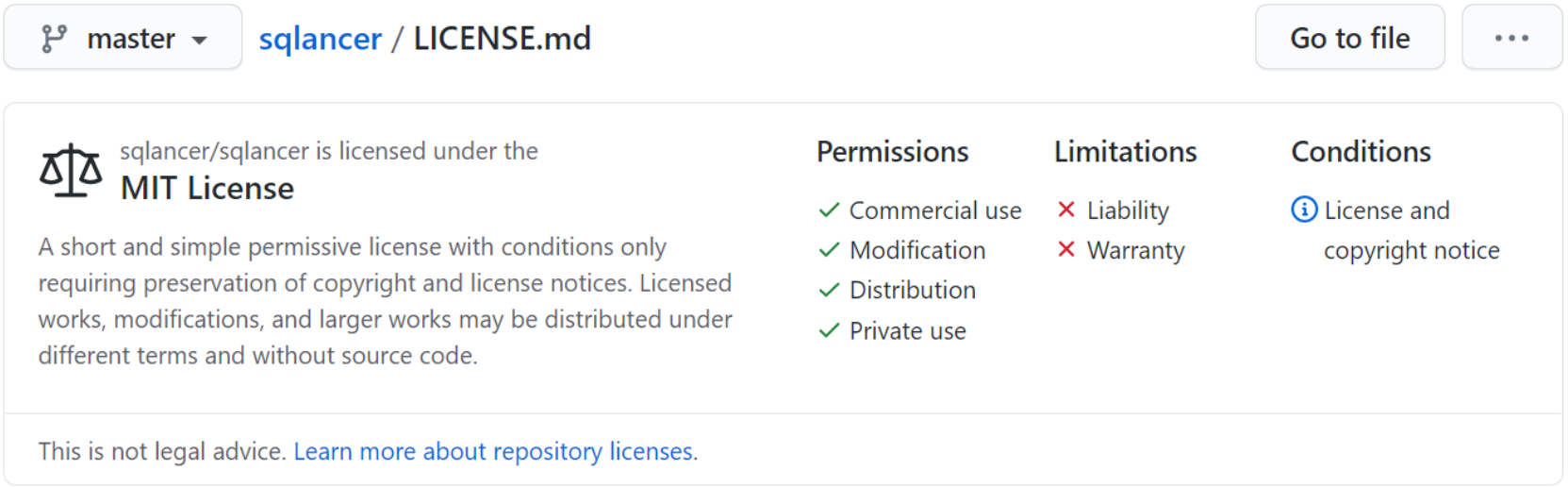
- Written in Java


```
$ git clone https://github.com/sqlancer/sqlancer
$ cd sqlancer
$ mvn package -DskipTests
$ cd target
$ java -jar sqlancer-*.jar sqlite3
```

You can quickly **try out** SQLancer on the embedded DBMSs **SQLite, H2, and DuckDB** without setting up a connection

Facts

- Written in Java
- Permissive license (MIT License)



Facts

- Written in Java
- Permissive license (MIT License)
- 43,000 LOC

Facts

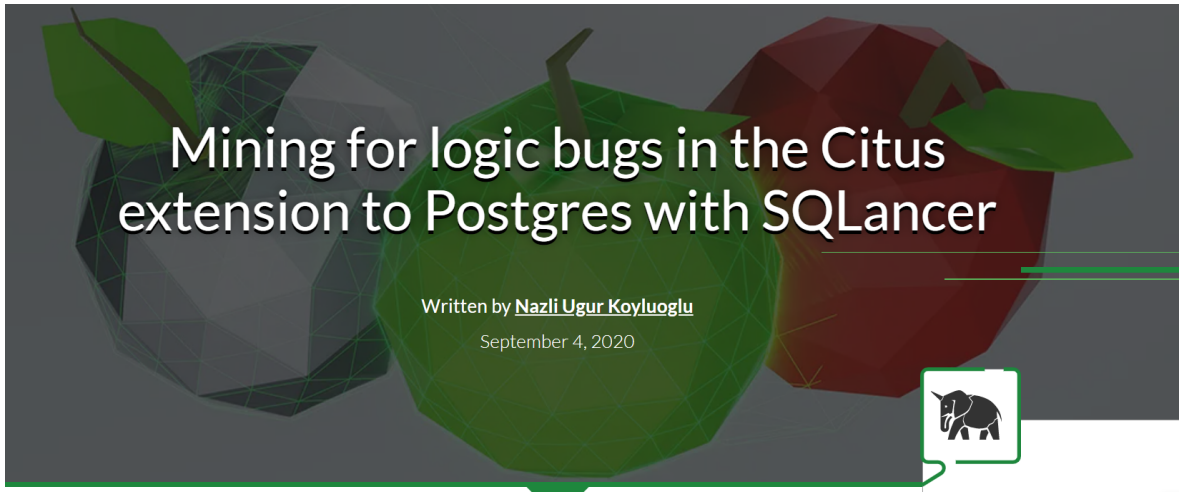
- Written in Java
- Permissive license (MIT License)
- 43,000 LOC

Facts

- Written in Java
- Permissive license (MIT License)
- 43,000 LOC
- 9 contributors

Contributors

Nazli Ugur Koyluoglu



https://www.citusdata.com/blog/2020/09/04/mining-for-logic-bugs-in-citus-with-sqlancer/

Contributors

Patrick Stäuble

ETH zürich

Contributors

Ilya Yatsishin

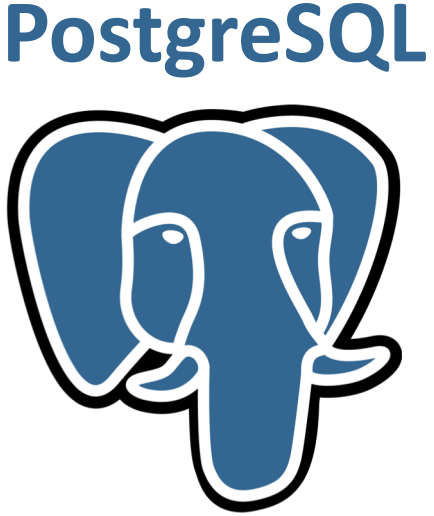
Yandex

ClickHouse

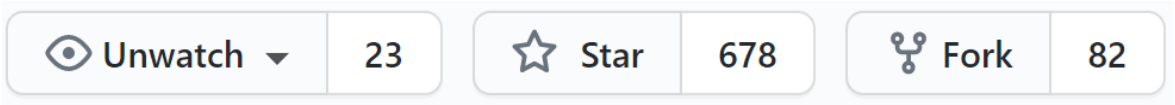
Facts

- Written in Java
- Permissive license (MIT License)
- 43,000 LOC
- 9 contributors
- >= 10 supported DBMSs

Supported DBMSs



SQLancer

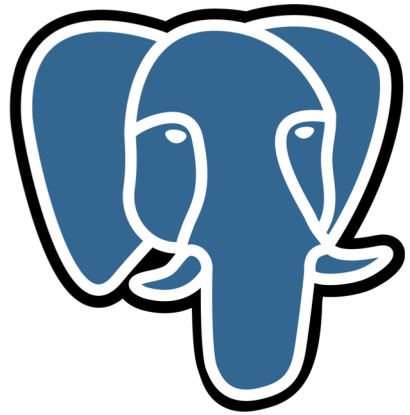


https://github.com/sqlancer

SQLancer is an effective and widely-used
automatic testing tool to find **logic bugs** in **DBMSs**

More than 450 Bugs

PostgreSQL



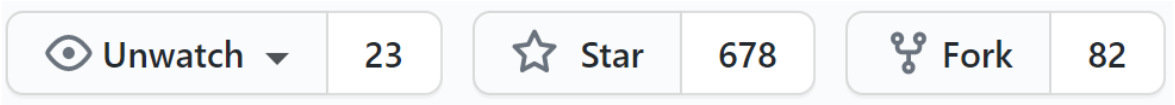
I used SQLancer to find **over 450 unique, previously unknown bugs** in widely-used DBMSs

More than 450 Bugs

	# Bugs		
DBMS	Logic	Error	Crash
CockroachDB	16	46	5
DuckDB	29	13	31
H2	2	15	1
MariaDB	5	0	1
MySQL	21	9	1
PostgreSQL	1	11	5
SQLite	93	44	42
TiDB	30	27	4
Sum	**197**	**165**	**90**

https://github.com/sqlancer/bugs

SQLancer



https://github.com/sqlancer

SQLancer is an effective and widely-used
automatic testing tool to find **logic bugs** in **DBMSs**

Adoption

Adoption

DuckDB

ClickHouse

monet

Yandex uses SQLancer to test every commit

citusdata

Adoption

DuckDB

ClickHouse

✓	# 5677.15	AMD64	Bionic	SQLancer	40 min 52 sec
✓	# 5677.16	AMD64	Bionic	SQLancer (with Address Sanitizer)	24 min 34 sec

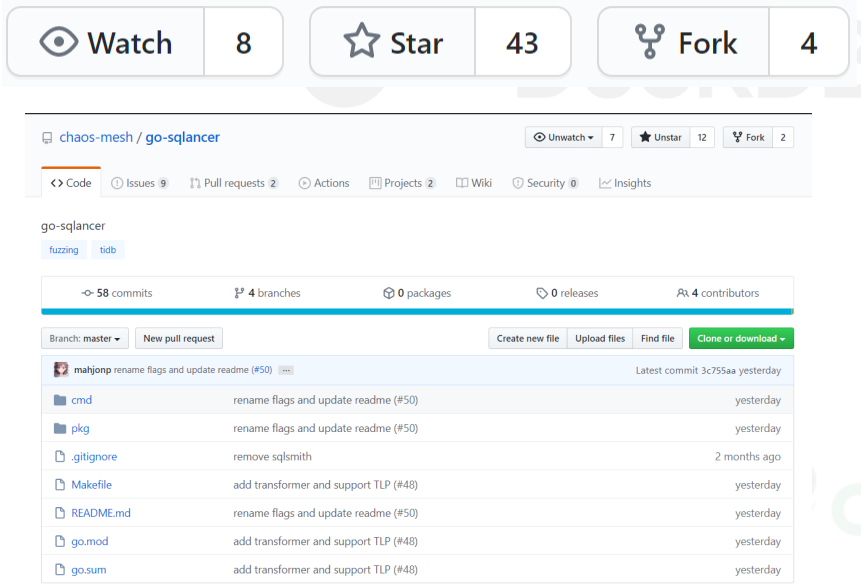
DuckDB runs SQLancer on every pull request

Adoption

"With the help of SQLancer, an automatic DBMS testing tool, we have been able to identify **>100 potential problems** in corner cases of the SQL processor."

https://www.monetdb.org/blog/faster-robuster-and-feature-richer-monetdb-in-2020-and-beyond

Adoption



TiDB

PingCAP implemented a tool go-sqlancer

https://github.com/chaos-mesh/go-sqlancer

Adoption

original query:

```
SELECT ALL t1.c1, t1.c0, t2.c1, t4.c1, t2.c0
FROM t1
CROSS JOIN t4
FULL OUTER JOIN t2 ON (((((((('[832125354,1134163512)'::int4range)*('(0,2106623281-…
```

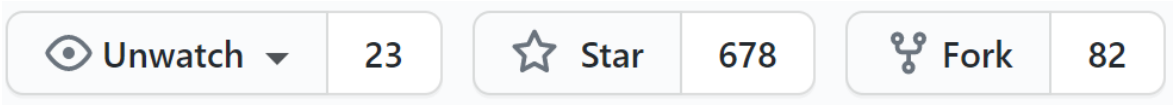
semantically equivalent to partitioned query:

```
SELECT ALL t1.c1, t1.c0, t2.c1, t4.c1, t2.c0
FROM t1
CROSS JOIN t4
FULL OUTER JOIN t2 ON (((((((('[832125354,1134163512)'::int4range)*('(0,2106623281-…
WHERE ((t4.c0)<(CAST(CAST(upper(t1.c0) AS VARCHAR(893)) AS VARCHAR)))

UNION ALL SELECT t1.c1, t1.c0, t2.c1, t4.c1, t2.c0
FROM t1
CROSS JOIN t4
FULL OUTER JOIN t2 ON (((((((('[832125354,1134163512)'::int4range)*('(0,2106623281-…
WHERE NOT (((t4.c0)<(CAST(CAST(upper(t1.c0) AS VARCHAR(893)) AS VARCHAR))))

UNION ALL SELECT ALL t1.c1, t1.c0, t2.c1, t4.c1, t2.c0
FROM t1
CROSS JOIN t4
FULL OUTER JOIN t2 ON (((((((('[832125354,1134163512)'::int4range)*('(0,2106623281-…
WHERE (((t4.c0)<(CAST((upper(t1.c0))::VARCHAR(893) AS VARCHAR)))) ISNULL
```

citusdata

https://www.citusdata.com/blog/2020/09/04/mining-for-logic-bugs-in-citus-with-sqlancer/

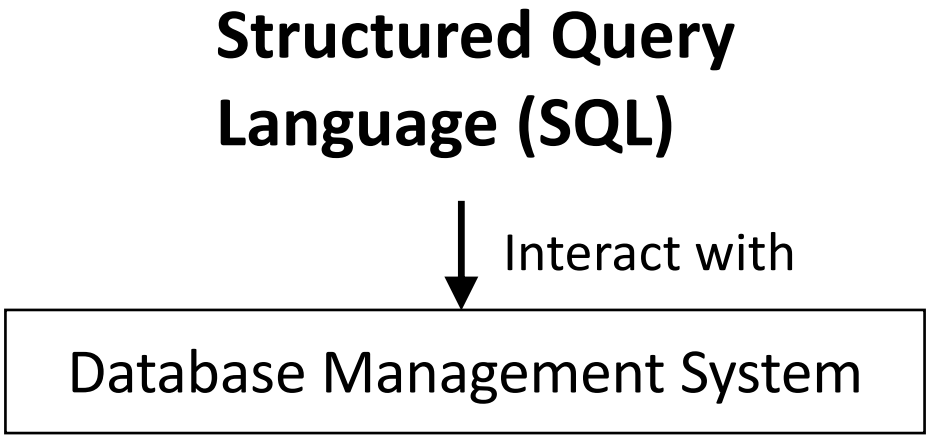
SQLancer



https://github.com/sqlancer

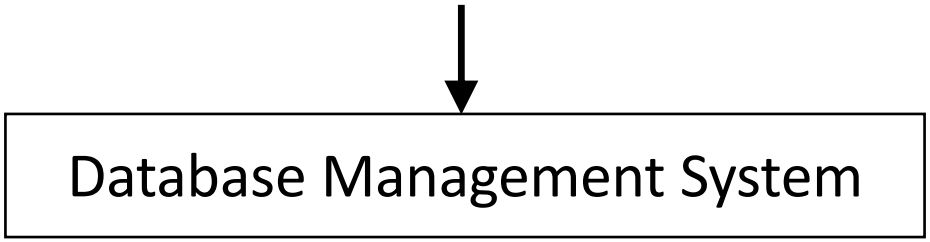
SQLancer is an effective and widely-used **automatic testing tool** to find **logic bugs** in **DBMSs**

Database Management Systems (DBMS)

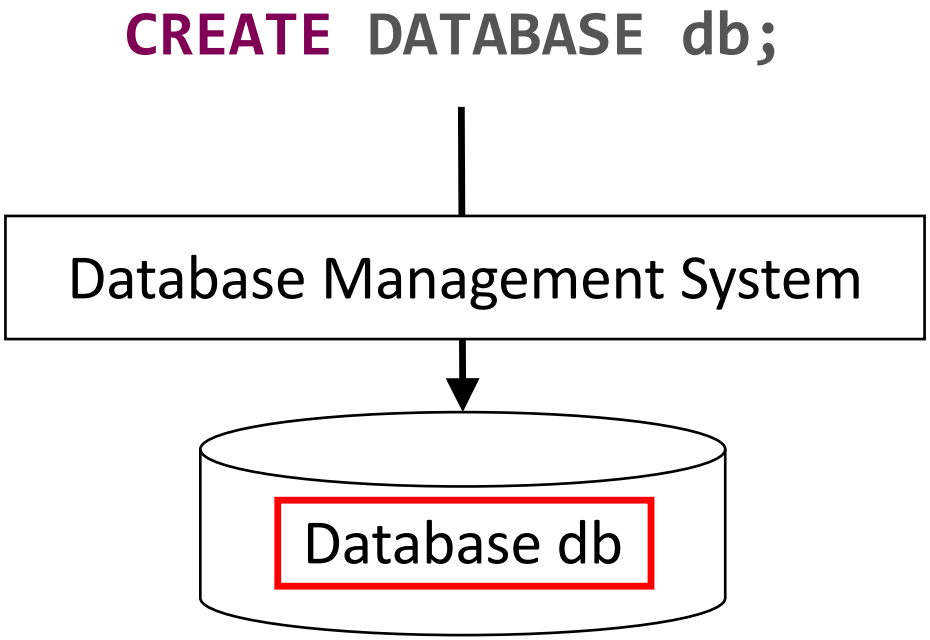


Database Management Systems (DBMS)

```
CREATE DATABASE db;
```

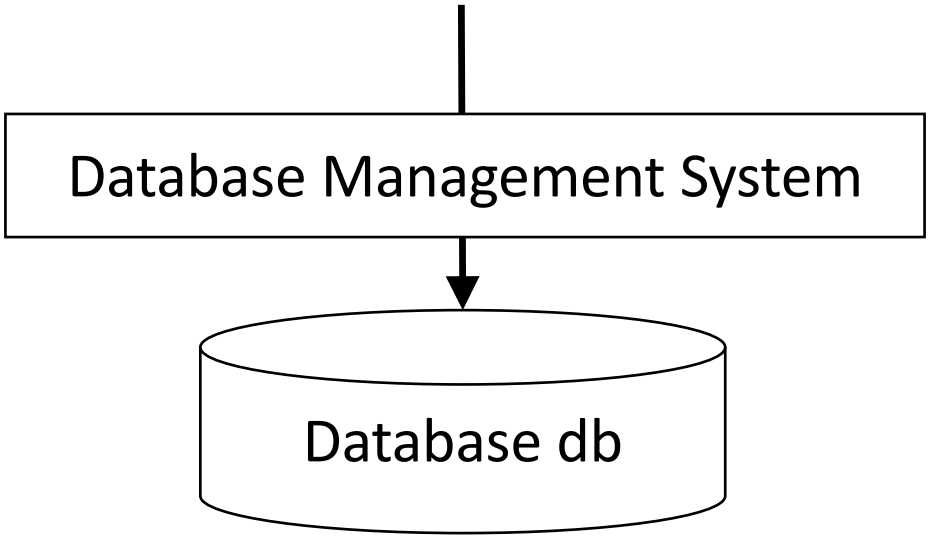


Database Management Systems (DBMS)

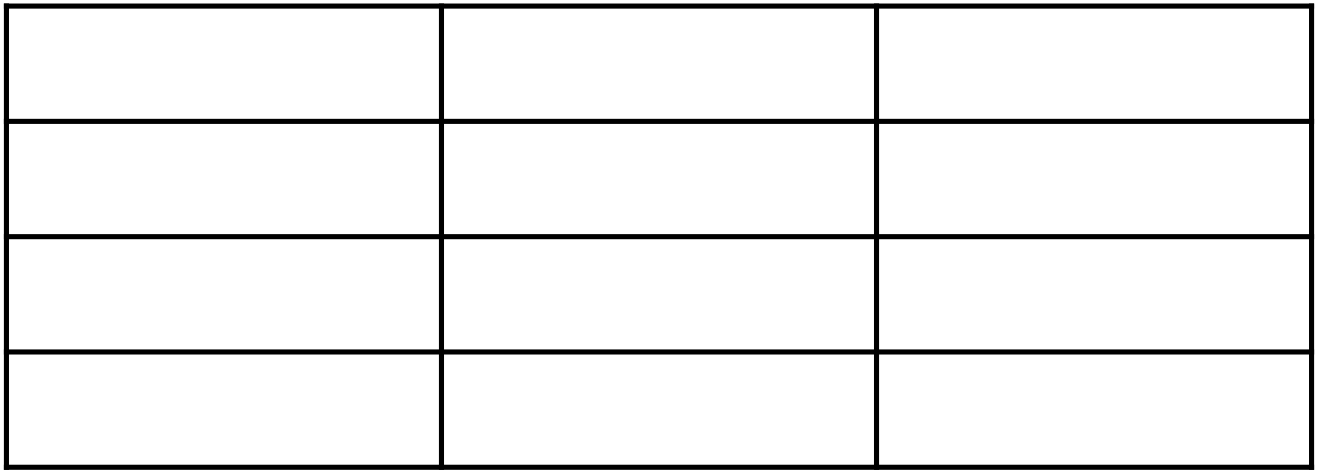


Database Management Systems (DBMS)

```
CREATE TABLE t0(c0 INTEGER);
```

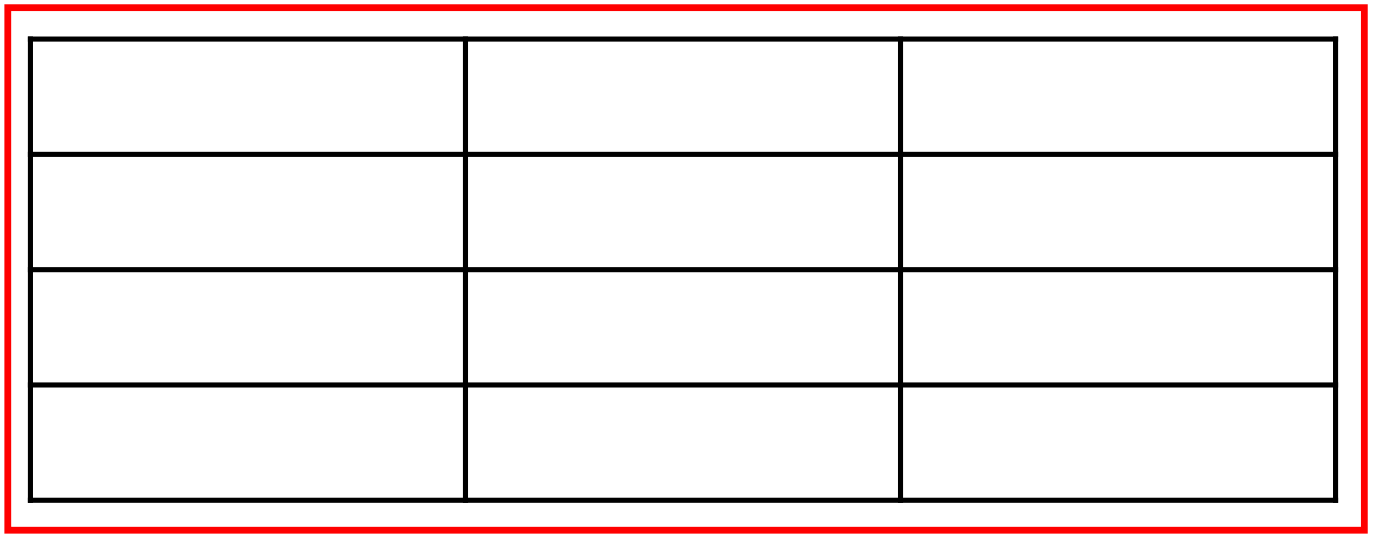


Relational Model



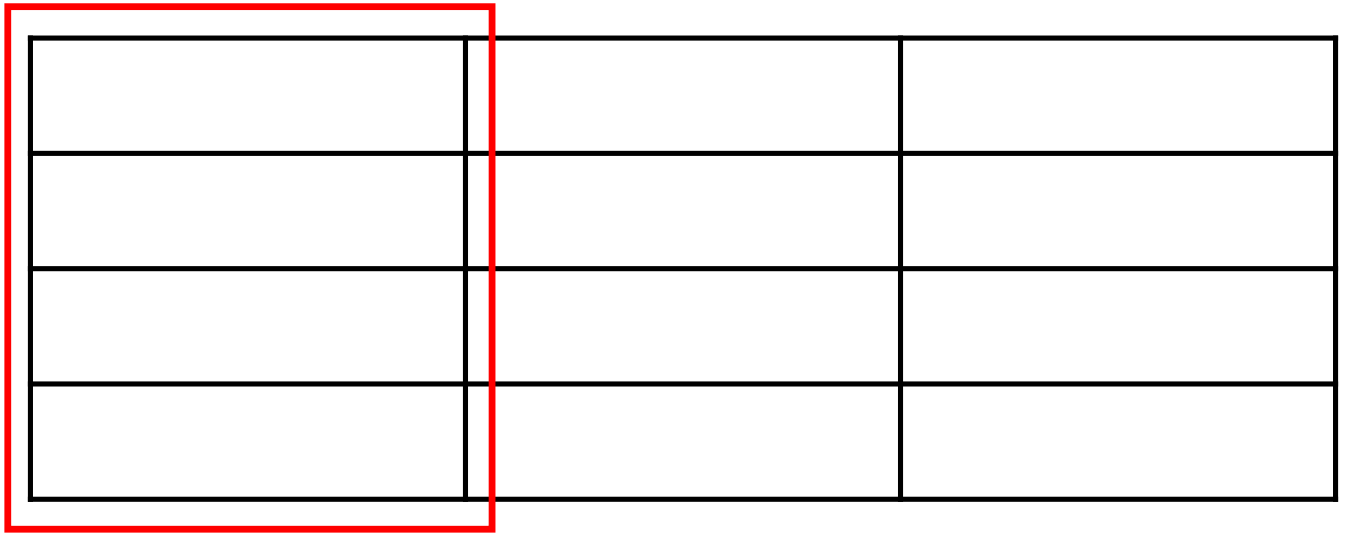
Relational Model

Table

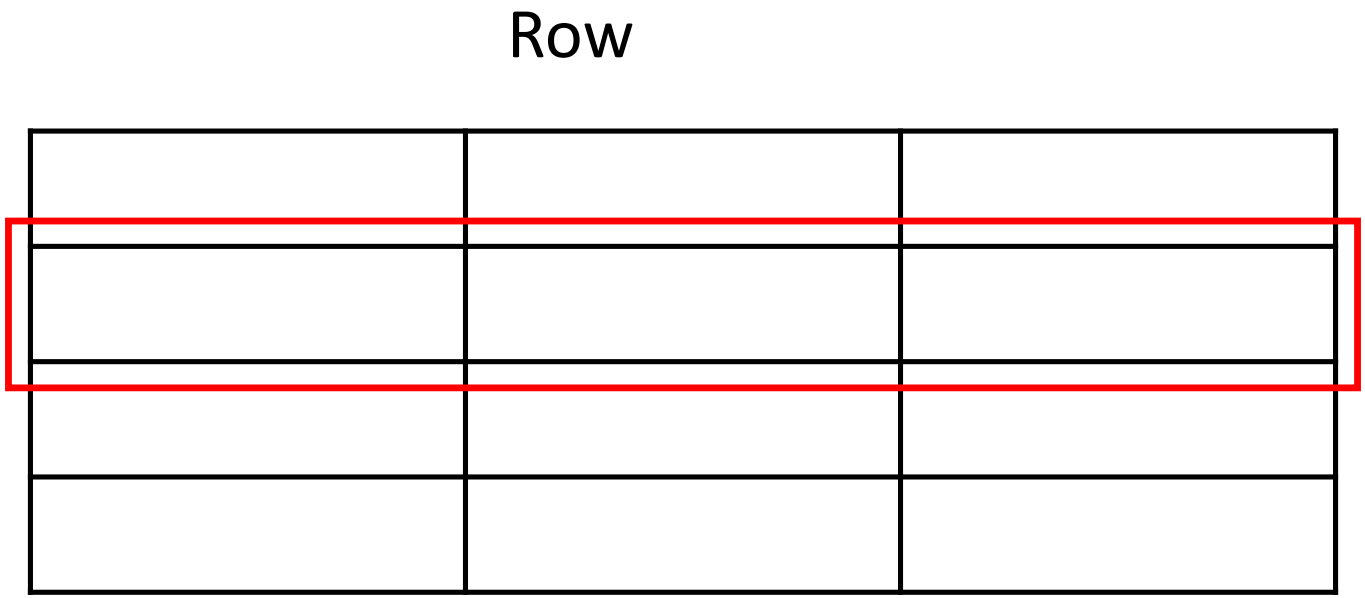


Relational Model

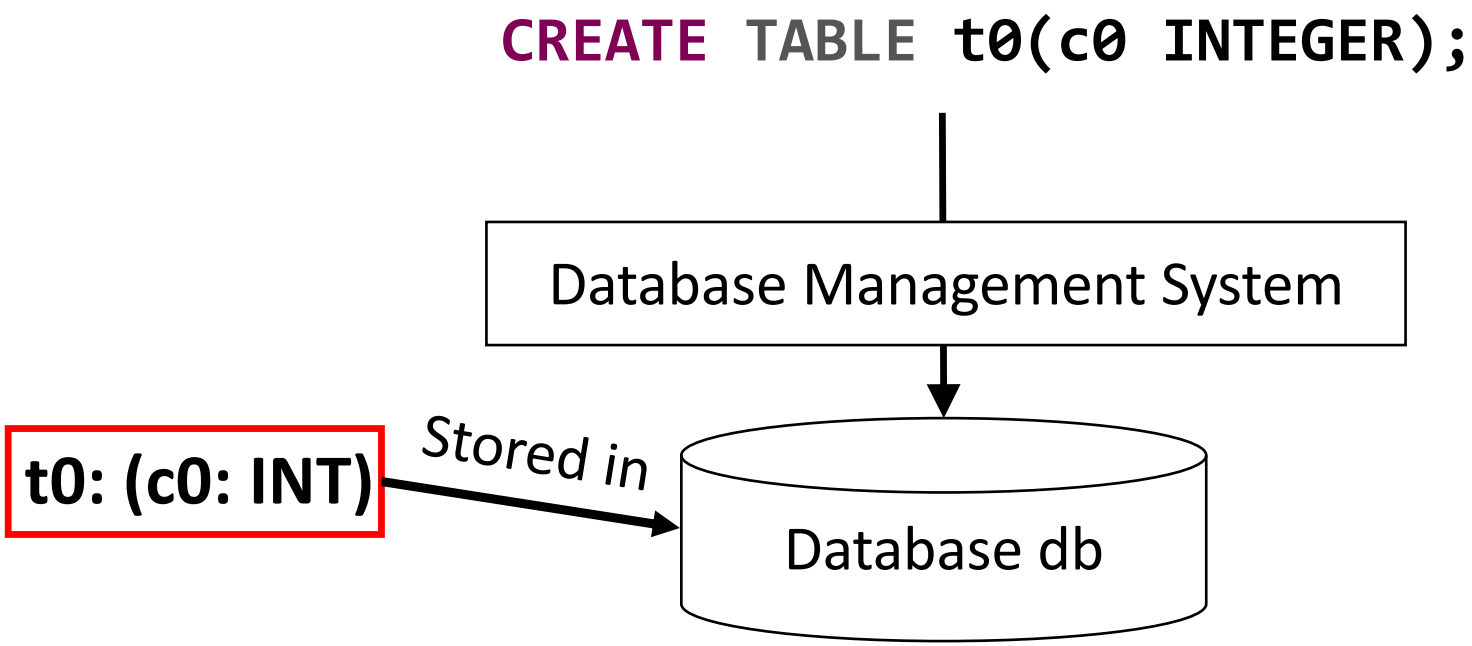
Column



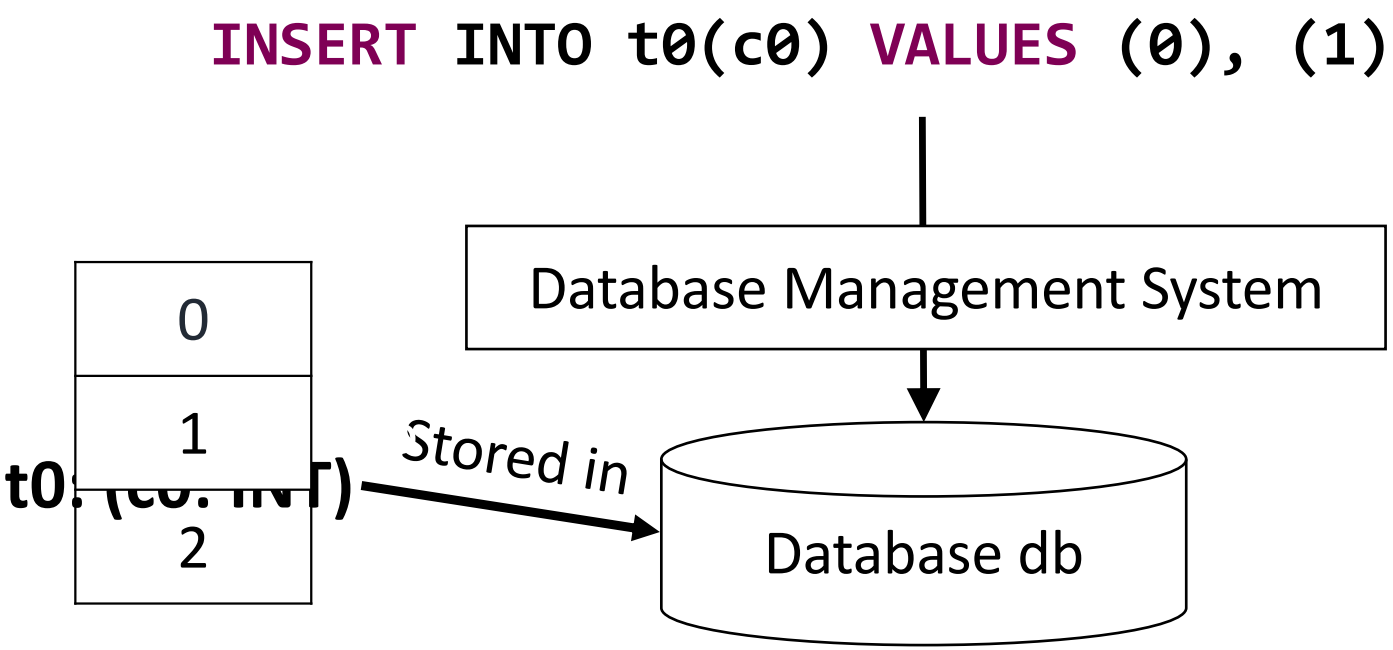
Relational Model



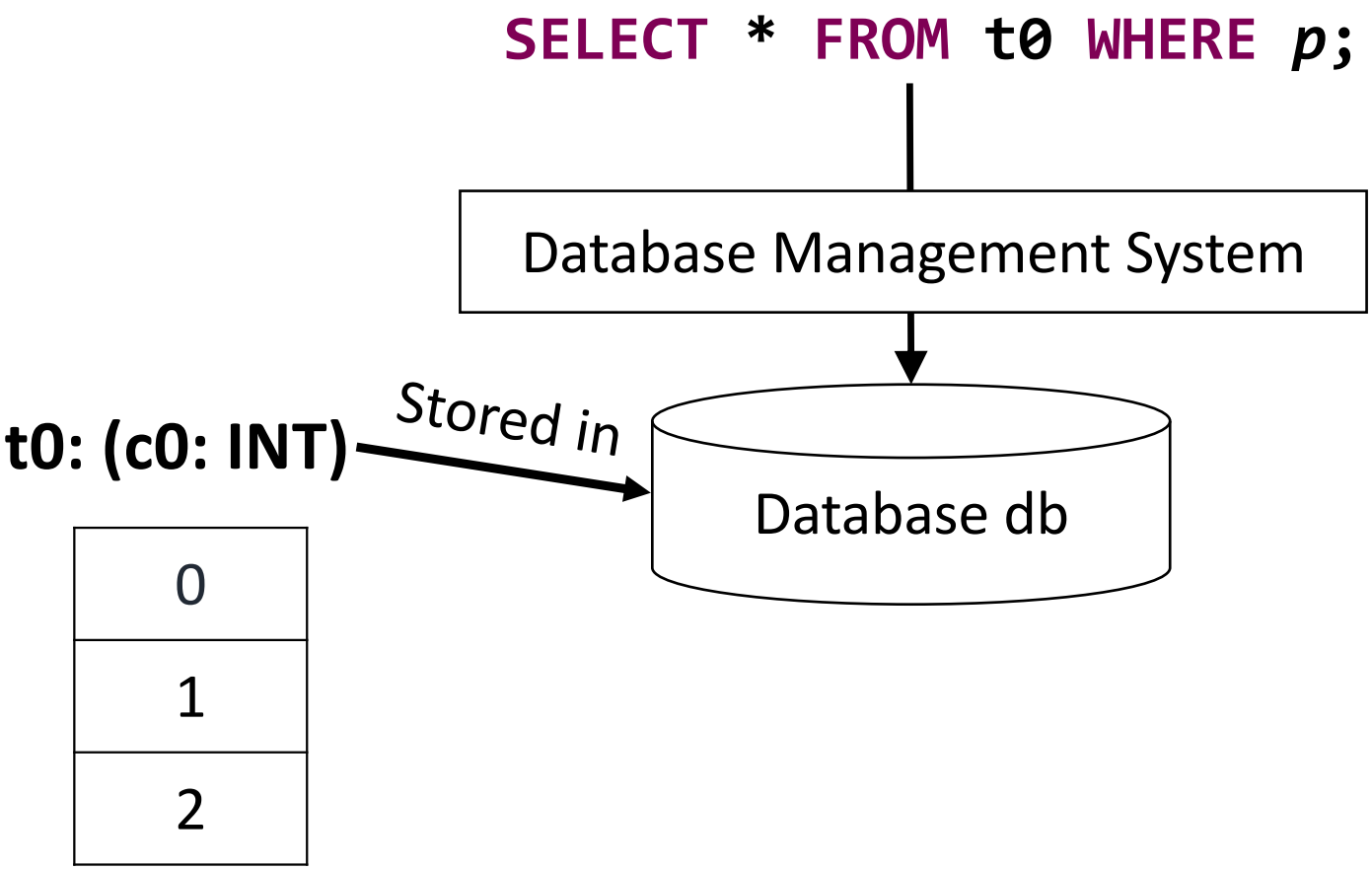
Database Management Systems (DBMS)



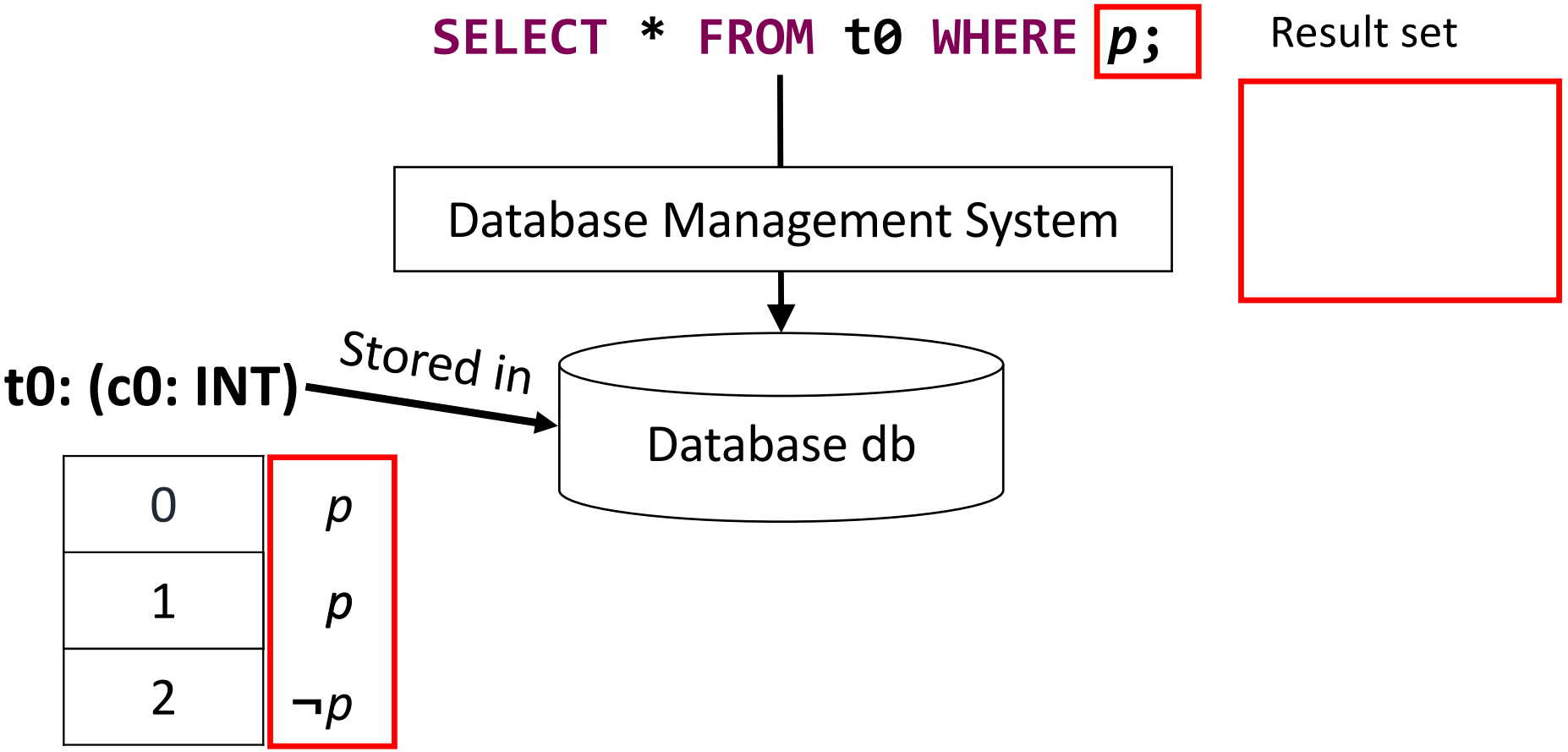
Database Management Systems (DBMS)



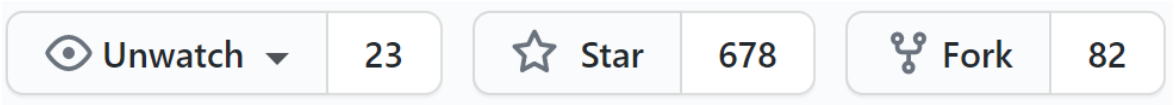
Database Management Systems (DBMS)



Database Management Systems (DBMS)



SQLancer



https://github.com/sqlancer

SQLancer is an effective and widely-used
automatic testing tool to find **logic bugs** in **DBMSs**

Goal: Find Logic Bugs

Bugs that cause the DBMS to return an **incorrect result set**

Motivating Example

t0

c0
0.0

t1

c0
-0.0


```
SELECT * FROM t0, t1
WHERE t0.c0 = t1.c0;
```

→ ?

It might seem **disputable** whether the predicate should evaluate to `true`

Motivating Example

t0

c0
0.0

t1

c0
-0.0

false?

0	0	0	0	0	…	0	0	0	0

0

1	0	0	0	0	…	0	0	0	0

-0

Different binary representation

```
SELECT * FROM t0, t1
WHERE t0.c0 = t1.c0;
```

t0.c0	t1.c0

Motivating Example

t0

c0
0.0

t1

c0
-0.0

true?

Evaluates to `true` for **most programming languages**

```
SELECT * FROM t0, t1
WHERE t0.c0 = t1.c0;
```

t0.c0	t1.c0
0.0	-0.0

Motivating Example

t0

c0
0.0

t1

c0
-0.0

```
SELECT * FROM t0, t1
WHERE t0.c0 = t1.c0;
```

t0.c0	t1.c0
0.0	-0.0

Motivating Example

t0

c0
0.0

t1

c0
-0.0

The latest version of MySQL that we tested failed to fetch the row


```
SELECT * FROM t0, t1
WHERE t0.c0 = t1.c0;
```

t0.c0	t1.c0

https://bugs.mysql.com/bug.php?id=99122

Motivating Example

t0

c0
0.0

t1

c0
-0.0

[3 Apr 2020 13:07] Jon Stephens

Documented fix as follows in the MySQL 8.0.21 changelog:

A query whose predicate compared 0 with -0 where at least one of these was a flaoting-point value returned incorrect results.


```
SELECT * FROM t0, t1
WHERE t0.c0 = t1.c0;
```

t0.c0	t1.c0

https://bugs.mysql.com/bug.php?id=99122

Motivating Example

t0

c0
0.0

t1

c0
-0.0

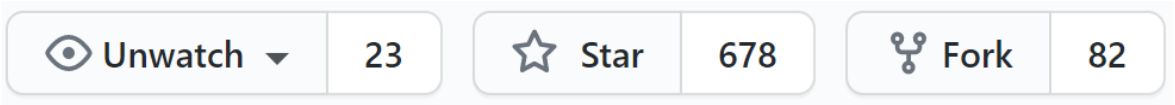
We could automatically **find the bug without having an accurate understanding** ourselves


```
SELECT * FROM t0, t1
WHERE t0.c0 = t1.c0;
```

t0.c0	t1.c0

https://bugs.mysql.com/bug.php?id=99122

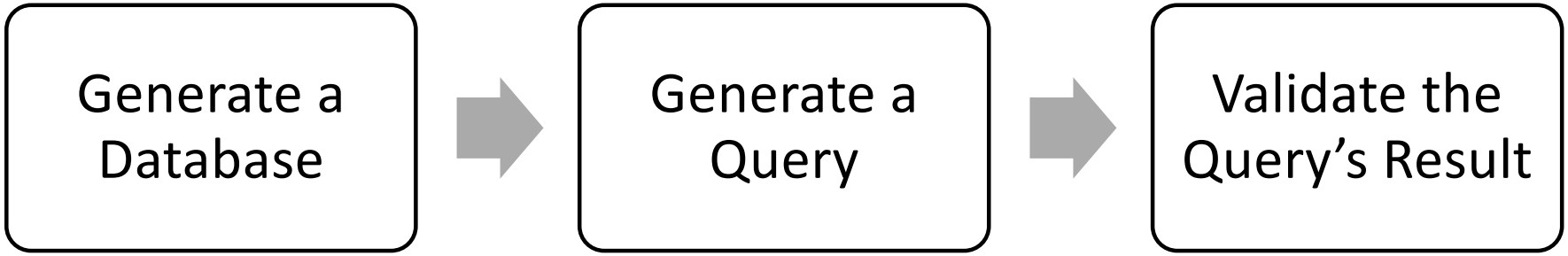
SQLancer



https://github.com/sqlancer

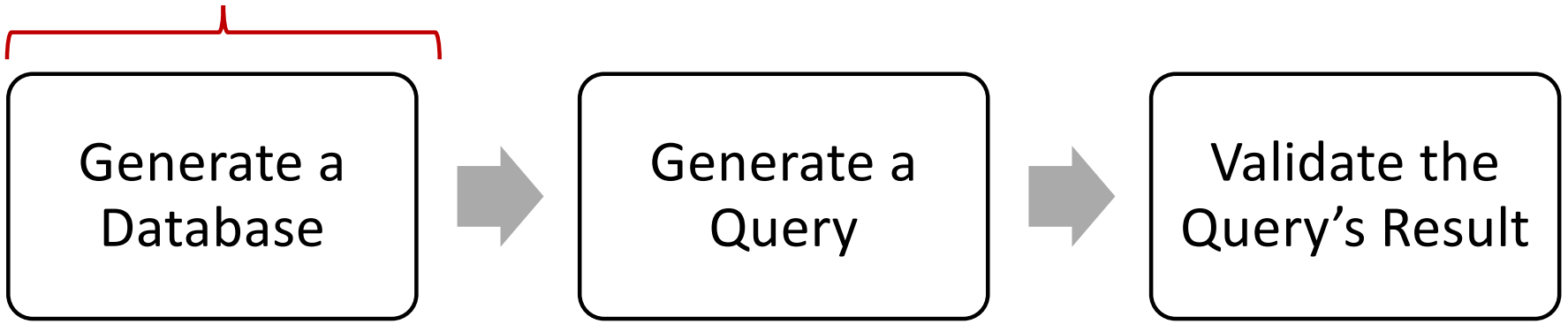
SQLancer is an effective and widely-used **automatic testing tool** to find **logic bugs** in **DBMSs**

Automatic Testing Core Challenges



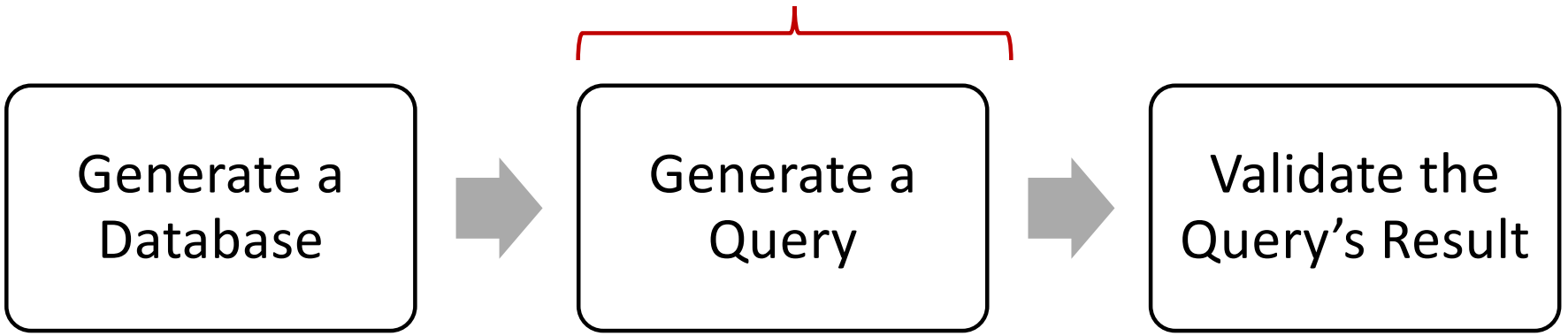
Automatic Testing Core Challenges

```
CREATE TABLE t0(c0 DOUBLE);
CREATE TABLE t1(c0 DOUBLE);
INSERT …
```

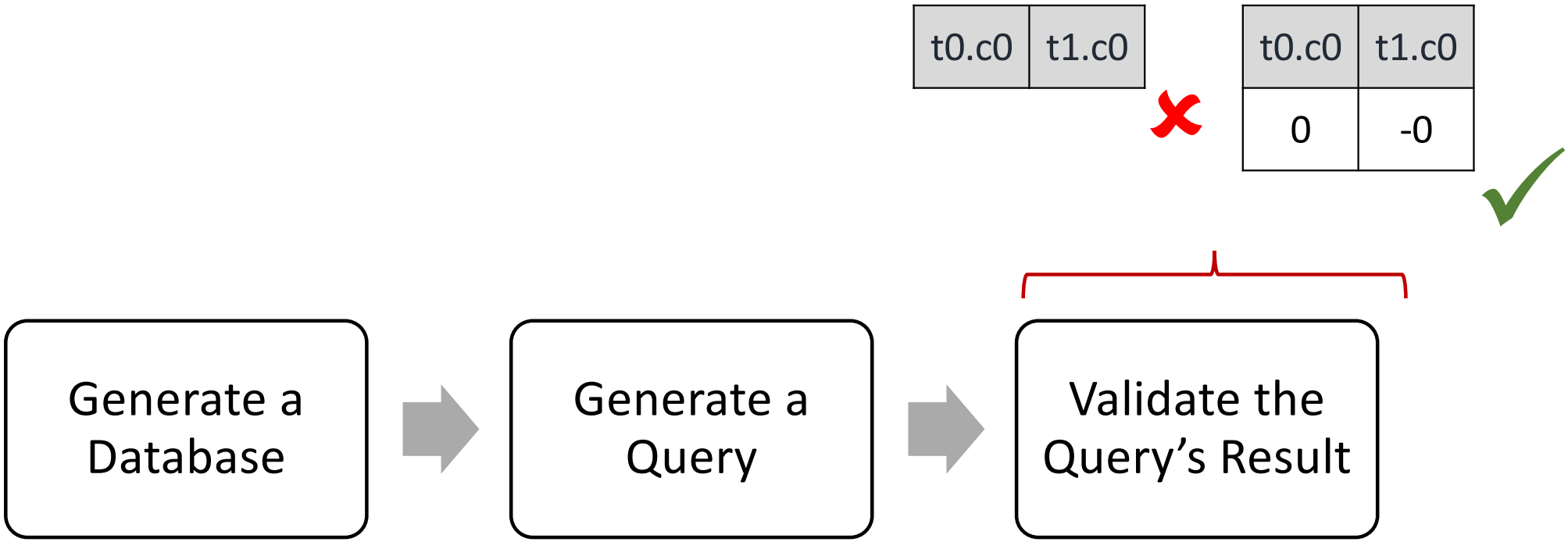


Automatic Testing Core Challenges

```
SELECT * FROM t0, t1
WHERE t0.c0 = t1.c0;
```

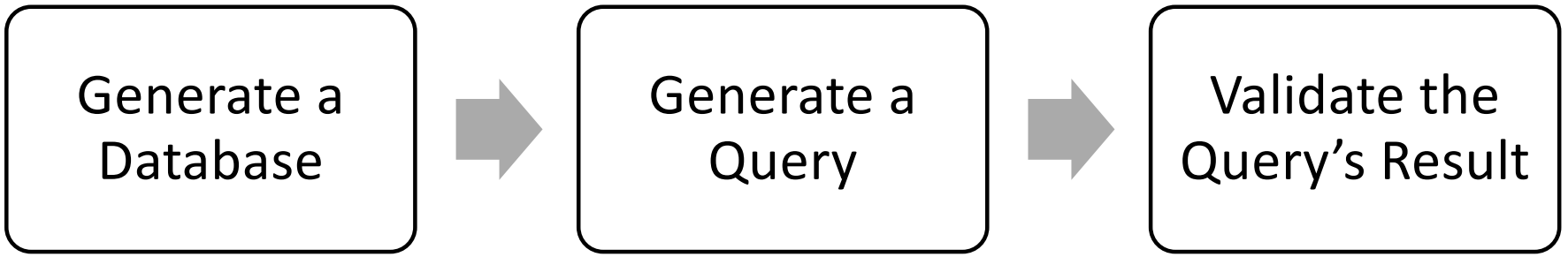


Automatic Testing Core Challenges



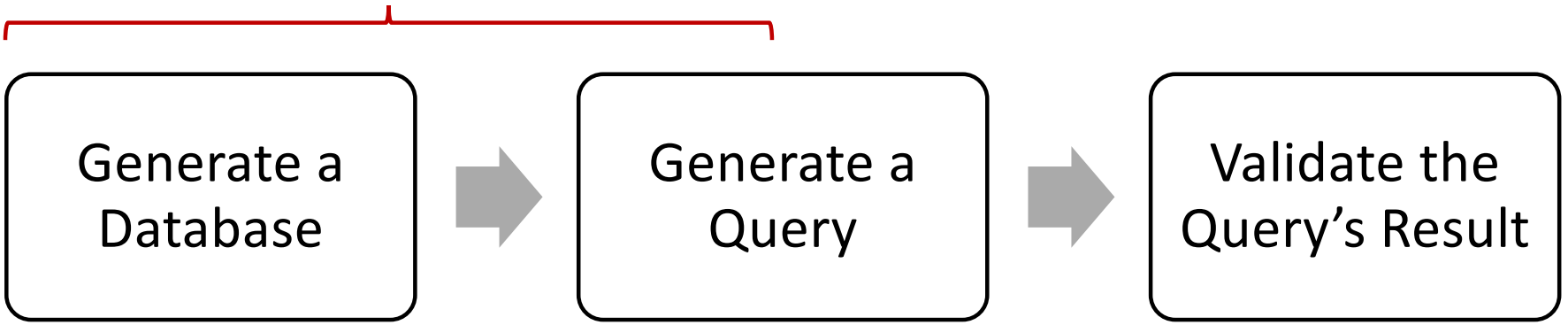
Automatic Testing Core Challenges

SQLancer **does not require any user interaction** for any of the steps



Automatic Testing Core Challenges

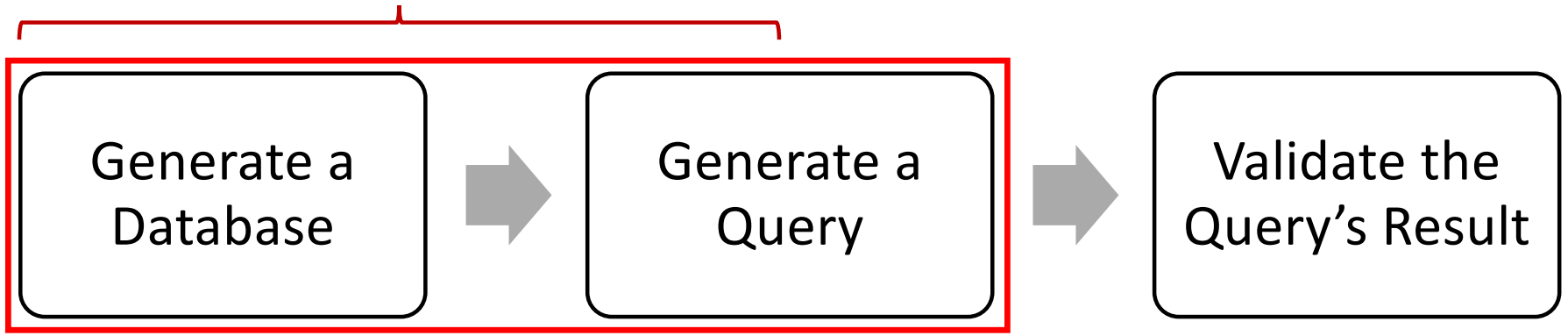
1. Effective test case



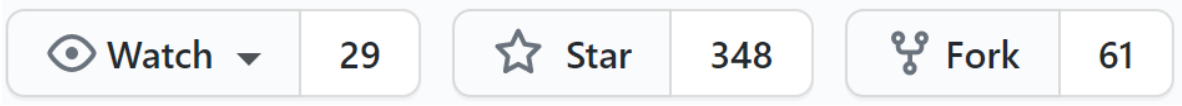
Automatic Testing Core Challenges

Manually implemented heuristic database and query generators

1. Effective test case



SQLsmith



SQLsmith

```
<mba> "I love the smell of coredumps in the morning"
```

Description

SQLsmith is a random SQL query generator. Its paragon is Csmith, which proved valuable for quality assurance in C compilers.

It currently supports generating queries for PostgreSQL, SQLite 3 and MonetDB. To add support for another RDBMS, you need to implement two classes providing schema information about and connectivity to the device under test.

Besides developers of the RDBMS products, users developing extensions might also be interested in exposing their code to SQLsmith's random workload.

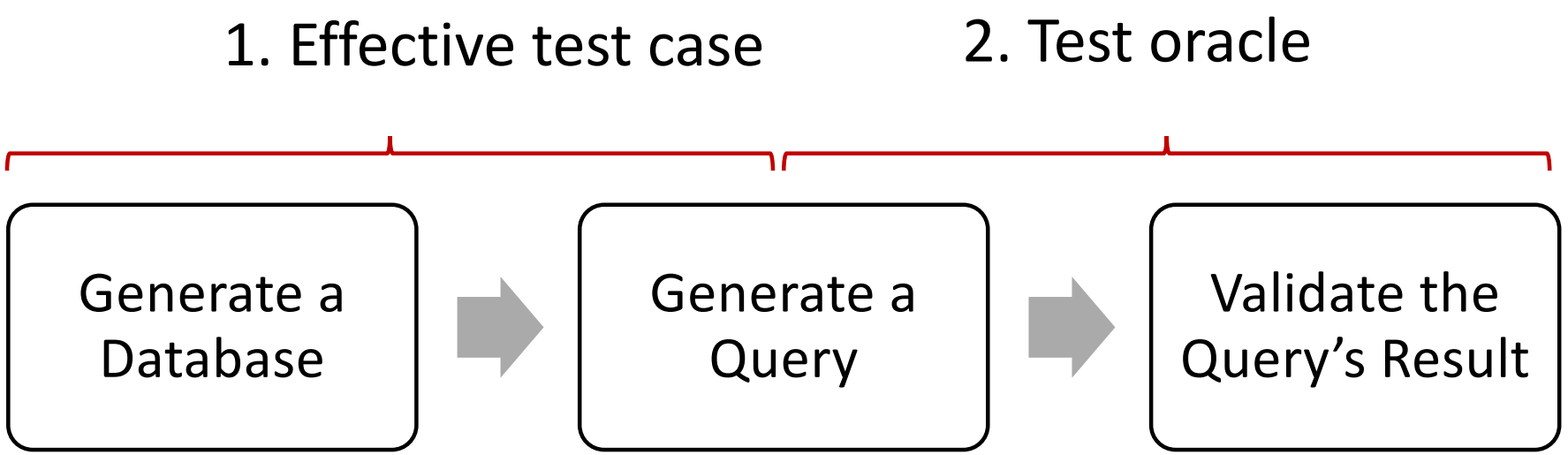
Since 2015, it found 118 bugs in alphas, betas and releases in the aforementioned products, including security vulnerabilities in released versions. Addtional bugs were squashed in extensions and libraries such as orafce and glibc.

https://github.com/anse1/sqlsmith/wiki#score-list

SQLancer's random database and query generation works **similar to existing tools**

https://github.com/anse1/sqlsmith

Automatic Testing Core Challenges

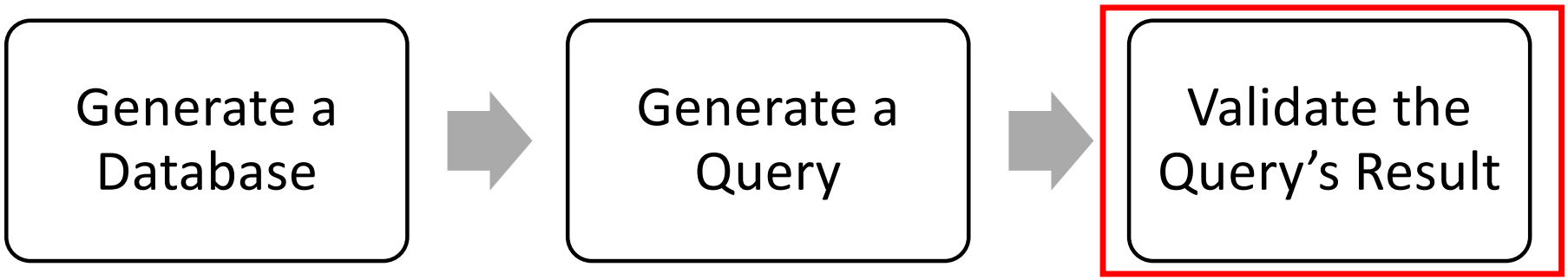


Automatic Testing Core Challenges

Test oracles are the “secret sauce” of SQLancer

1. Effective test case

2. Test oracle



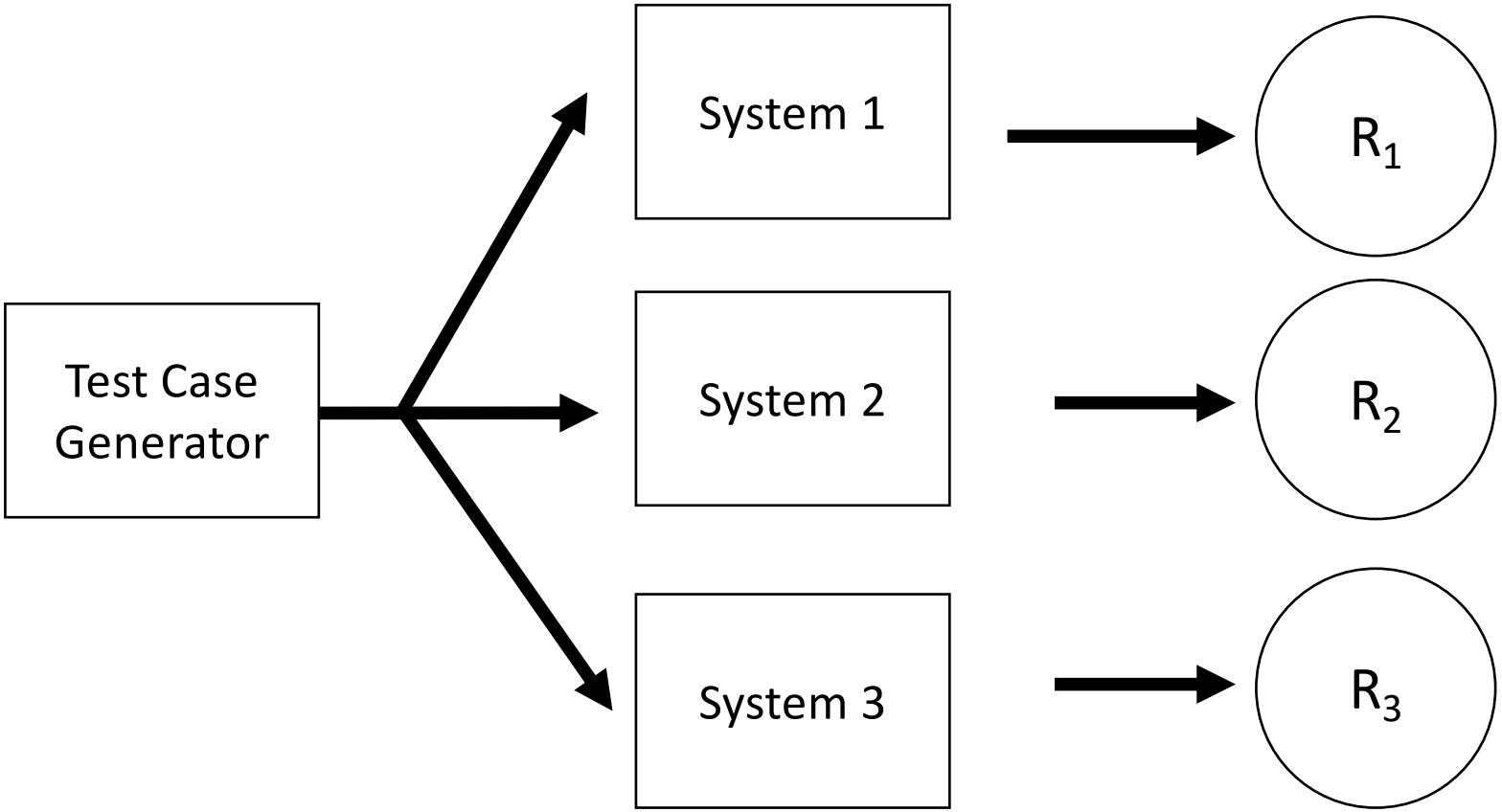
Test Oracle

“a test oracle (or just oracle) is a mechanism for determining whether a test has passed or failed”

https://en.wikipedia.org/wiki/Test_oracle

What existing test oracles could we use to find bugs in DBMSs?

Differential Testing



$R_1 = R_2 = R_3$?

Differential Testing: RAGS (Slutz, VLDB 1998)

Massive Stochastic Testing of SQL

Don Slutz
Microsoft Research
dslutz@Microsoft.com

Abstract

Deterministic testing of SQL database systems is human intensive and cannot adequately cover the SQL input domain. A system (RAGS), was built to stochastically generate valid SQL statements 1 million times faster than a human and execute them.

1 Testing SQL is Hard

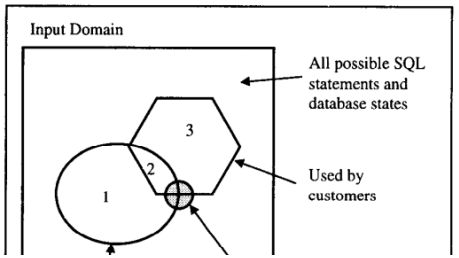
Good test coverage for commercial SQL database systems is very hard. The *input domain*, all SQL statements, from any number of users, combined with all states of the database, is gigantic. It is also difficult to verify output for positive tests because the semantics of SQL are complicated.

Software engineering technology exists to predictably improve quality ([Bei90] for example). The techniques involve a software development process including unit tests and final system validation tests (to verify the absence of bugs). This process requires a substantial investment so commercial SQL vendors with tight schedules tend to use a more ad hoc proc-

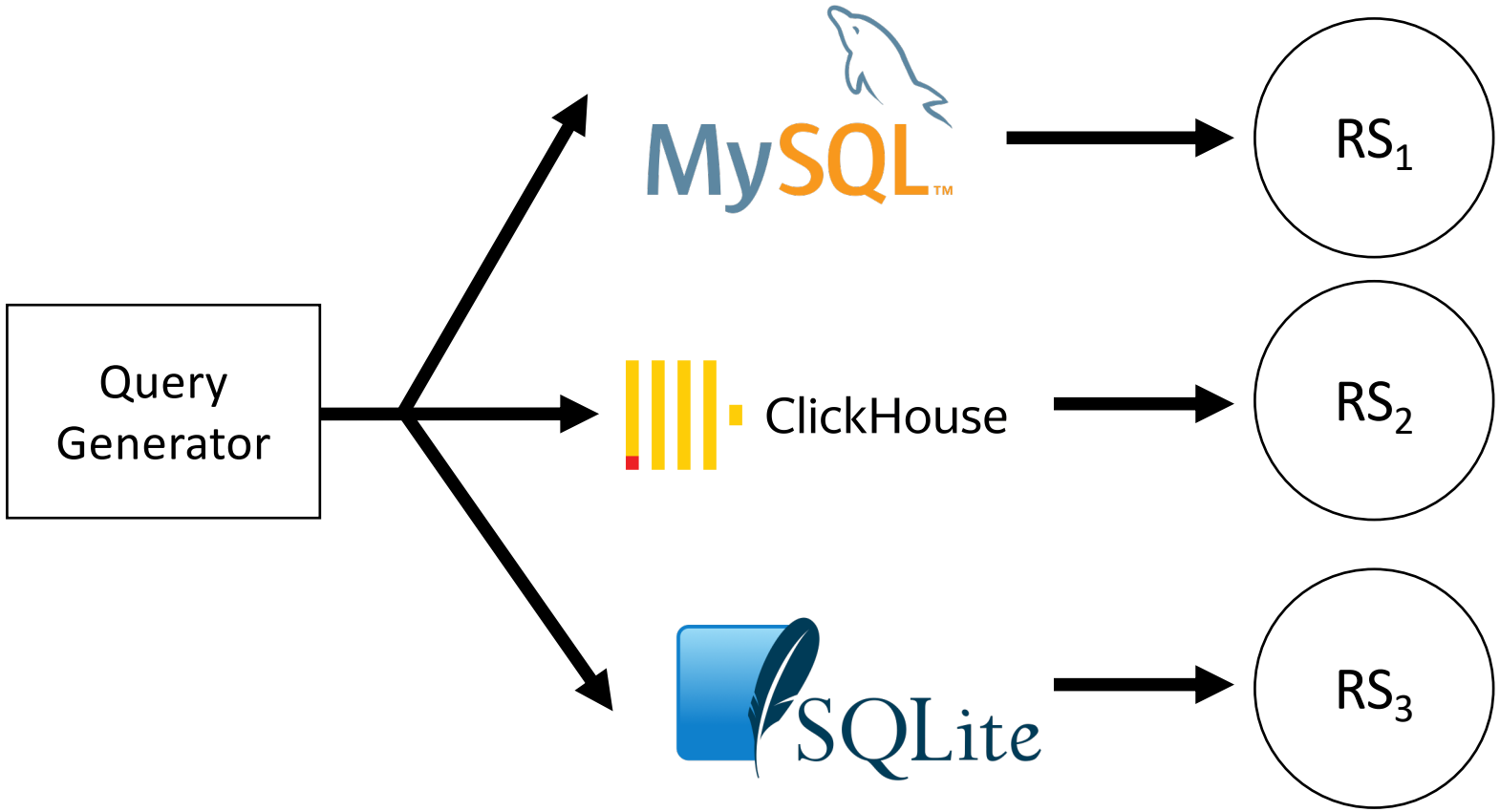
distribute the SQL statements in useful regions of the input domain. If the distribution is adequate, stochastic testing has the advantage that the quality of the tests improves as the test size increases [TFW93].

A system called RAGS (Random Generation of SQL) was built to explore automated testing. RAGS is currently used by the Microsoft SQL Server [MSS98] testing group. This paper describes RAGS and some illustrative test results.

Figure 1 illustrates the test coverage problem. Customers use the hexagon, bugs are in the oval, and the test libraries cover the shaded circle.

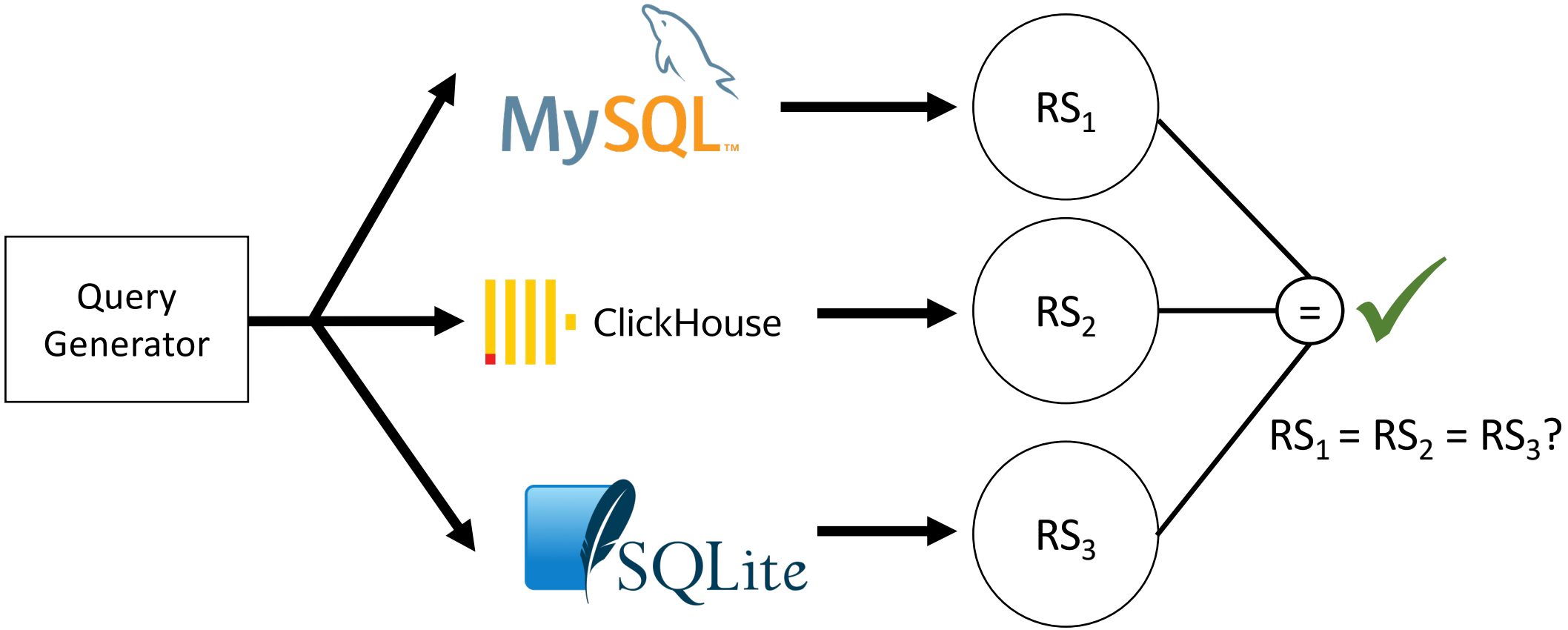


Differential Testing: RAGS (Slutz, VLDB 1998)

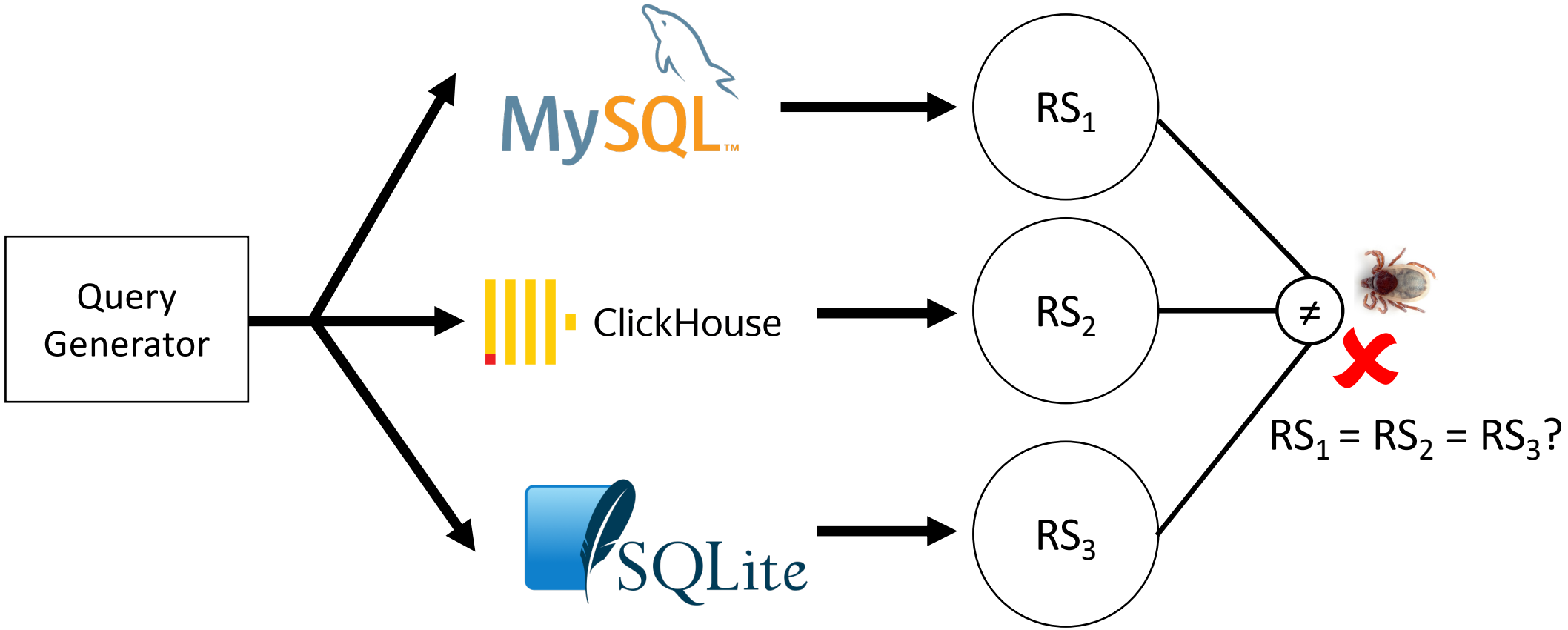


$RS_1 = RS_2 = RS_3$?

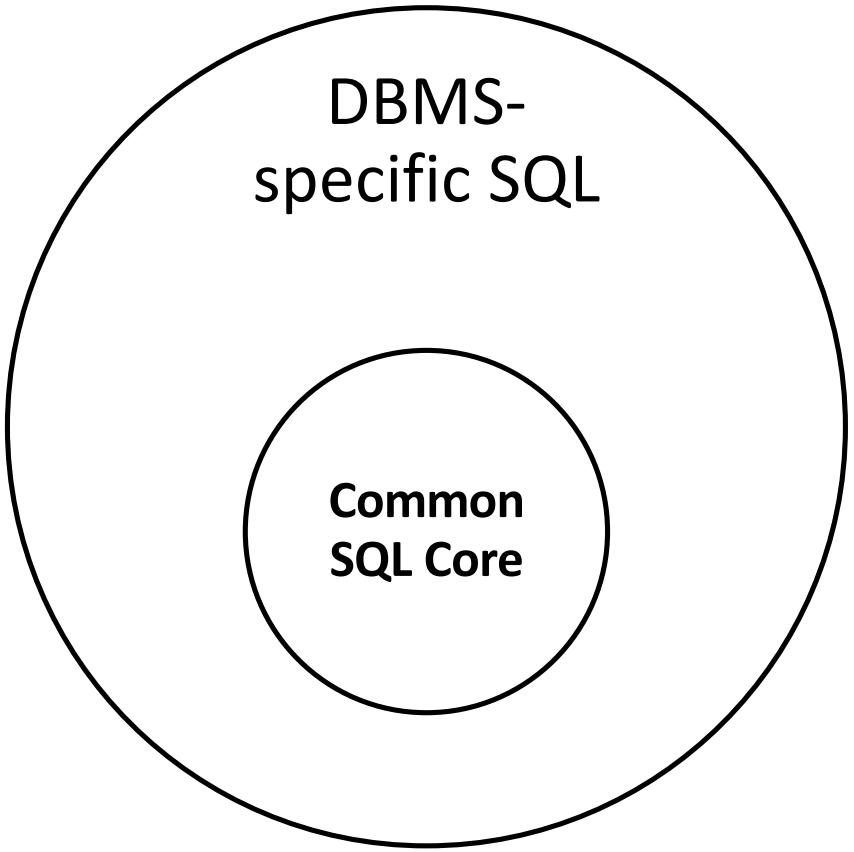
Differential Testing: RAGS (Slutz, VLDB 1998)



Differential Testing: RAGS (Slutz, VLDB 1998)



Differential Testing: RAGS (Slutz, VLDB 1998)



"We are **unable to use Postgres** as an oracle because CockroachDB has **slightly different semantics** and SQL support, and generating queries that execute identically on both is tricky […]." – Cockroach Labs

What oracles were introduced in SQLancer?

Finding Logic Bugs in DBMSs

Ternary Logic Partitioning

OOPSLA ‘20

SQLancer provides three test oracles

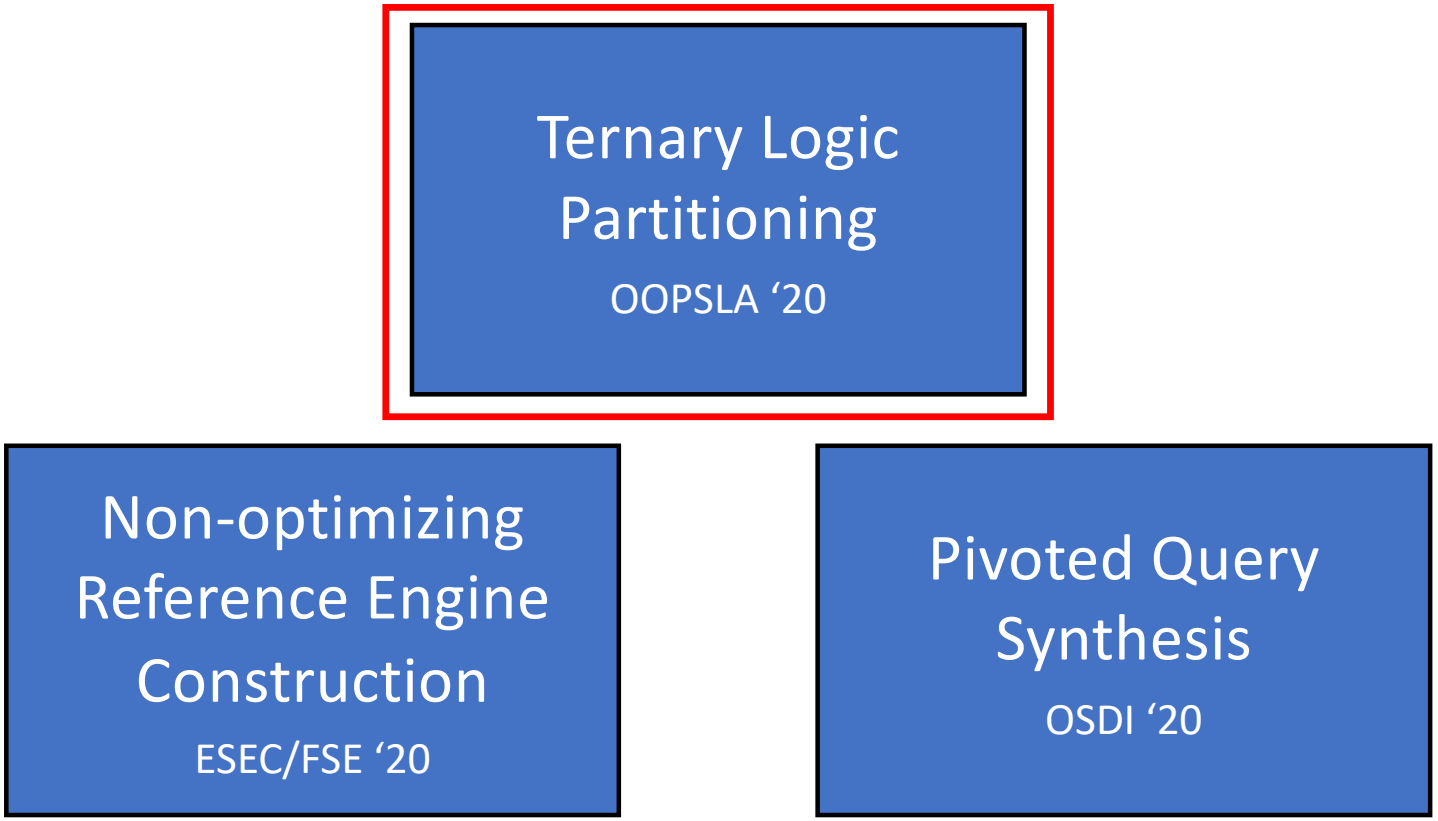
Non-optimizing Reference Engine Construction

ESEC/FSE ‘20

Pivoted Query Synthesis

OSDI ‘20

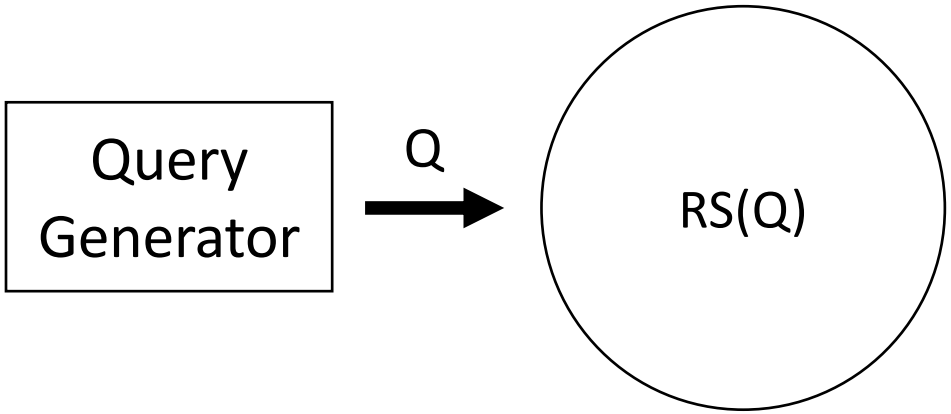
Finding Logic Bugs in DBMSs



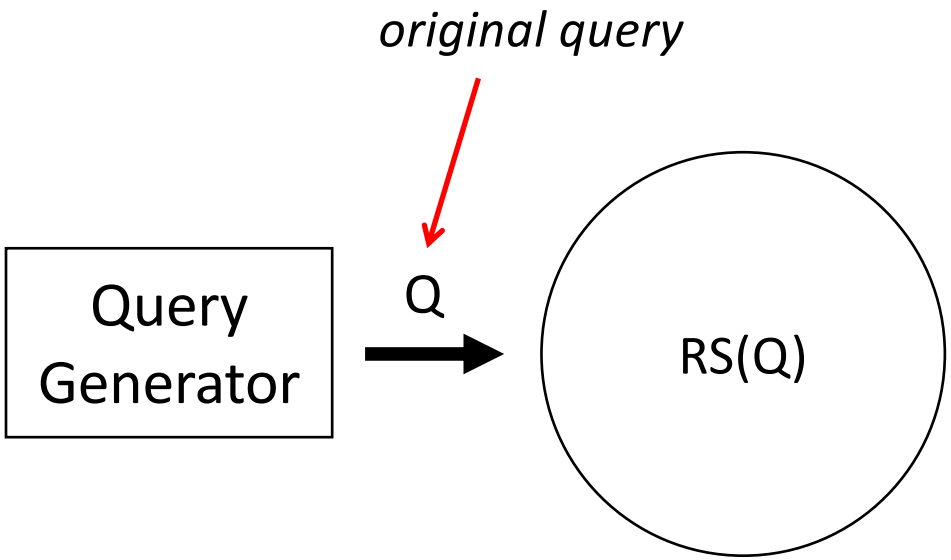
Query Partitioning

Ternary Logic Partitioning (TLP) is based on a conceptual framework called **Query Partitioning**

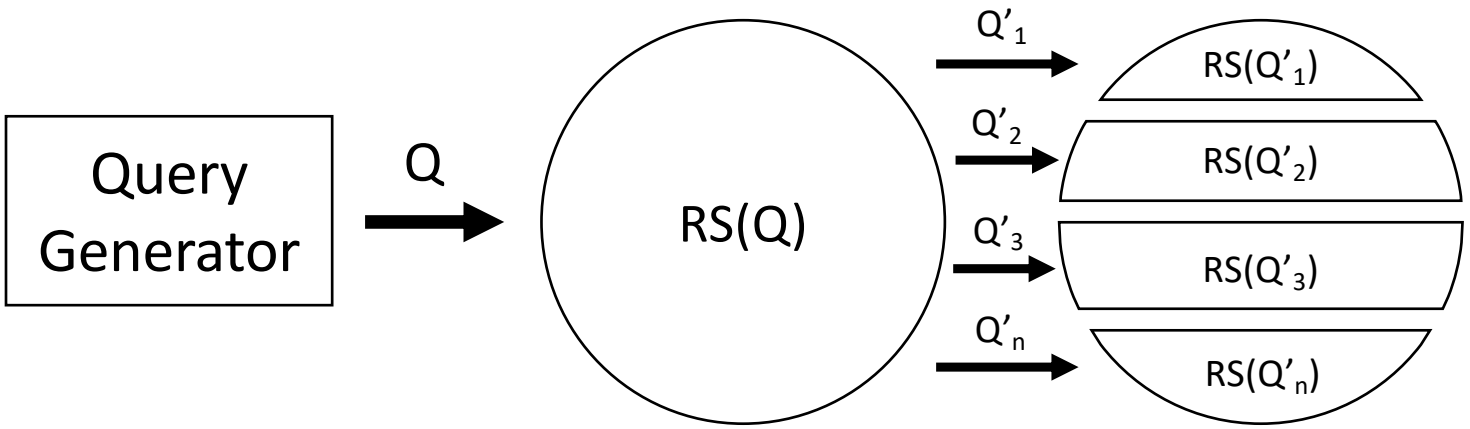
Query Partitioning



Query Partitioning

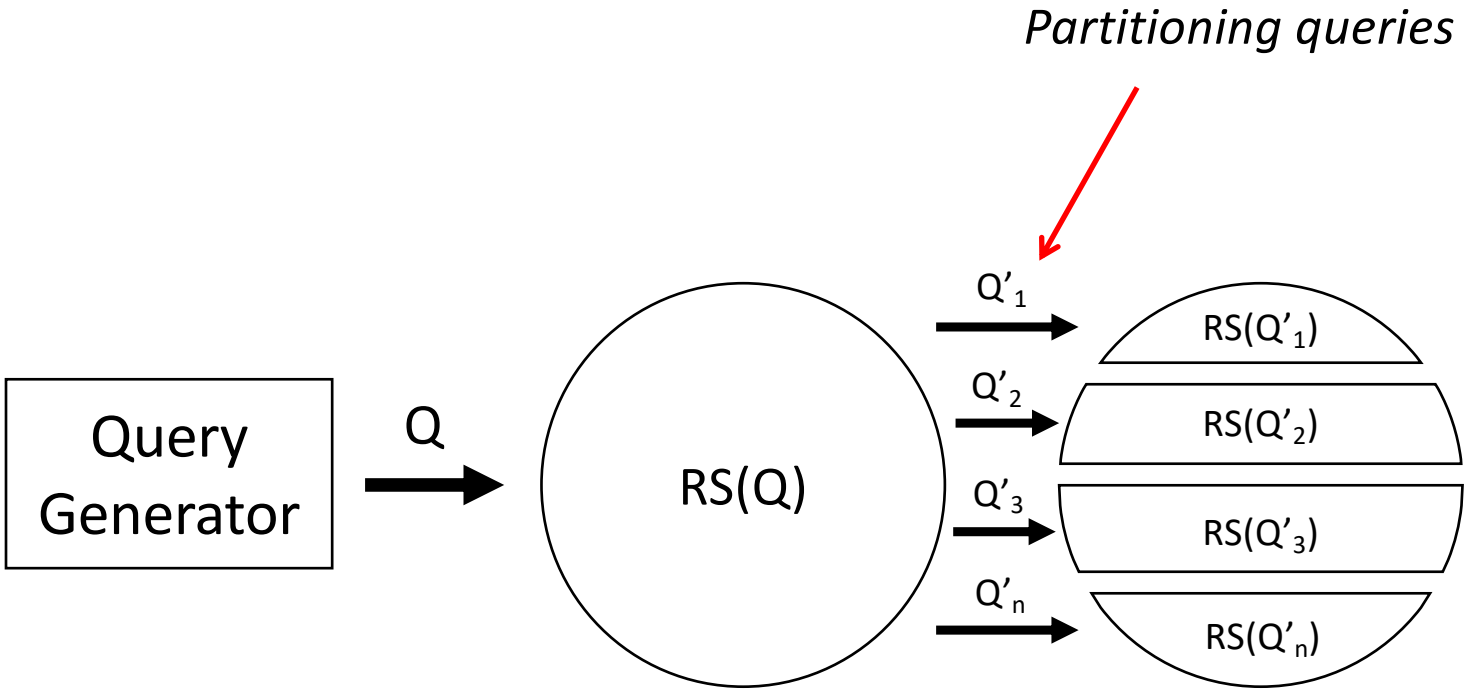


Query Partitioning

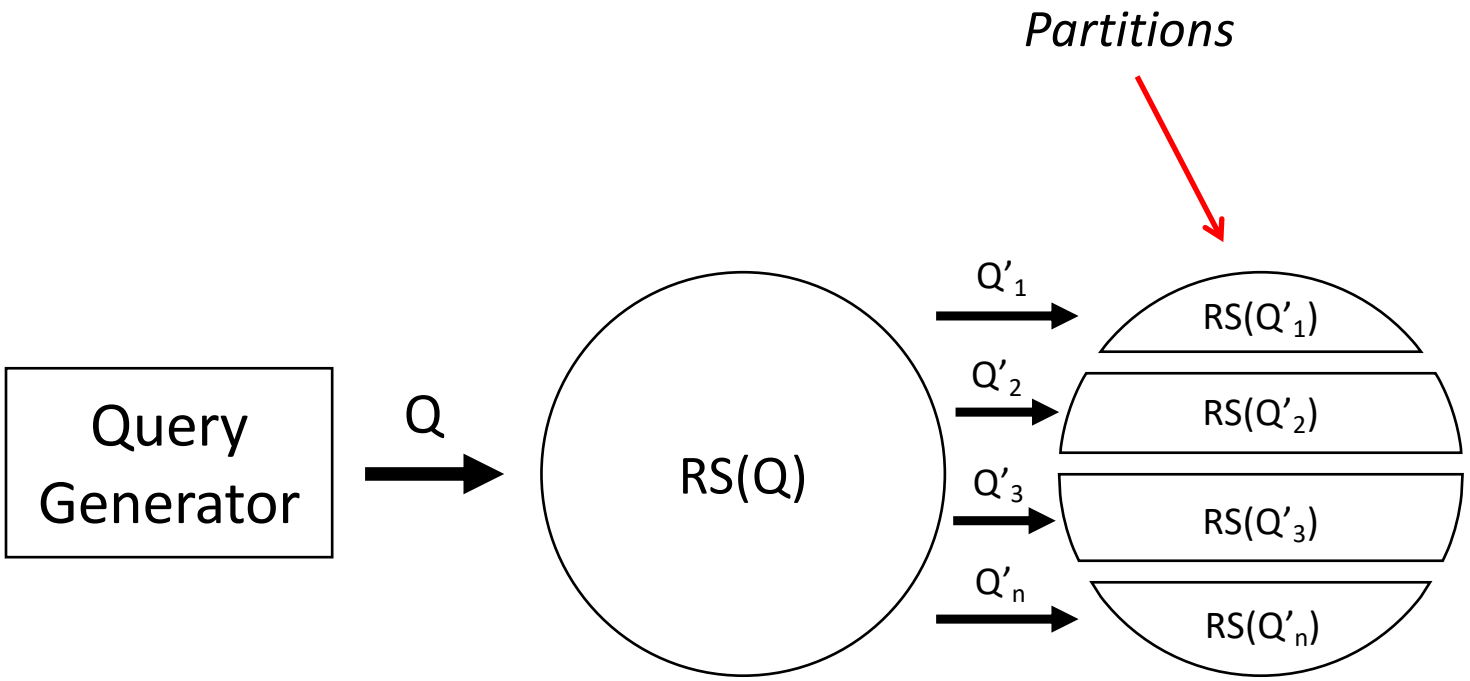


Partition the result set

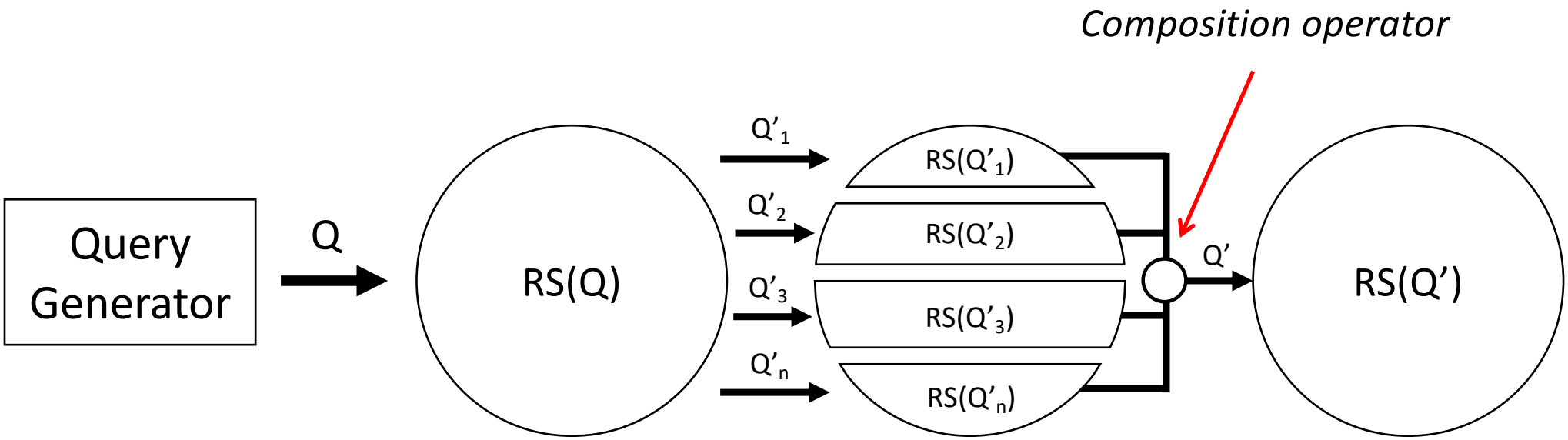
Query Partitioning



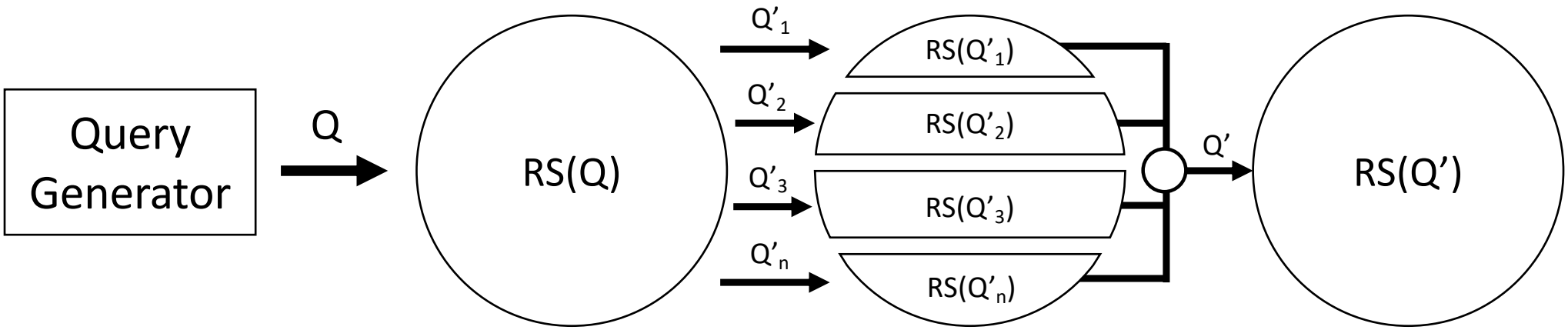
Query Partitioning



Query Partitioning

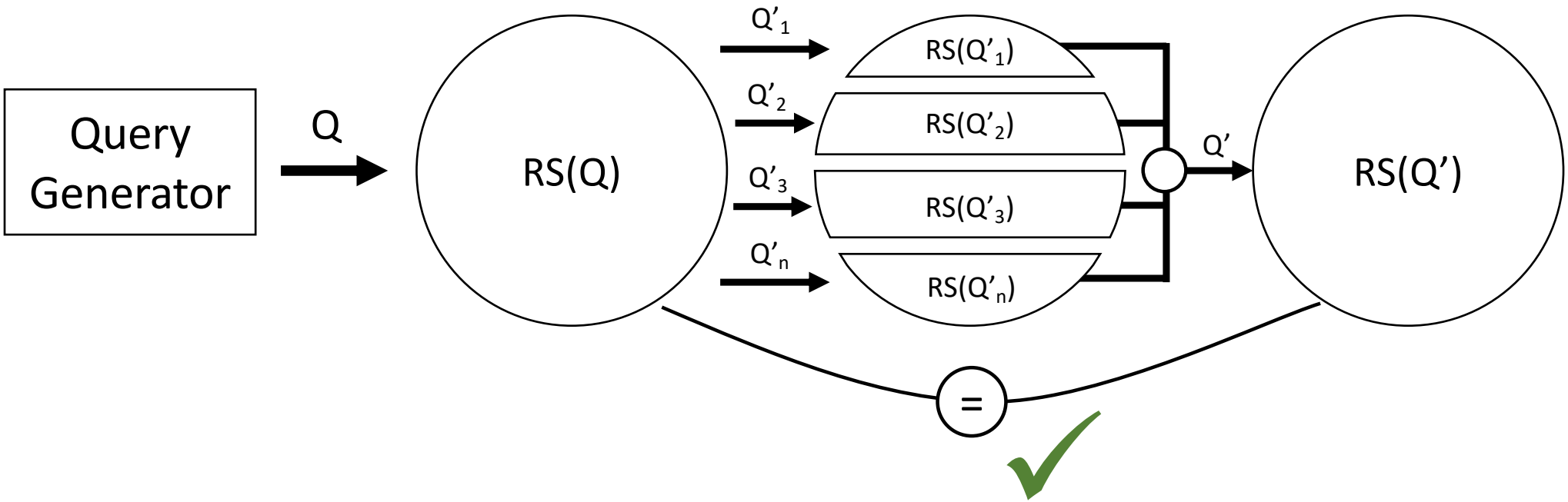


Query Partitioning

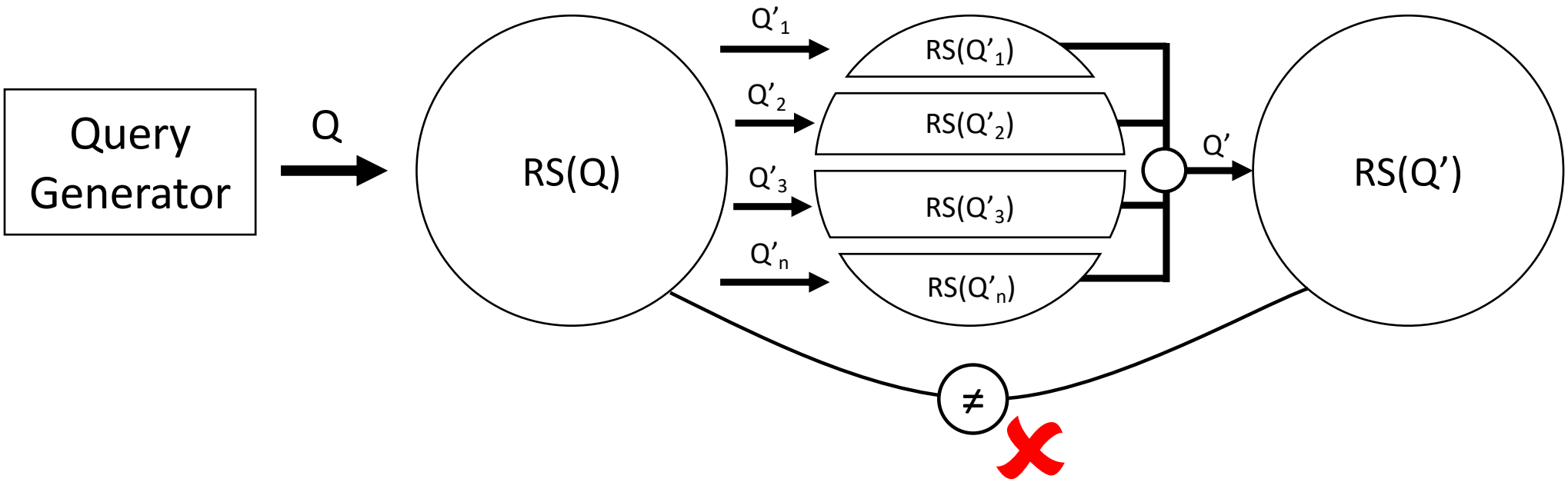


Combine the results so that RS(Q)=RS(Q')

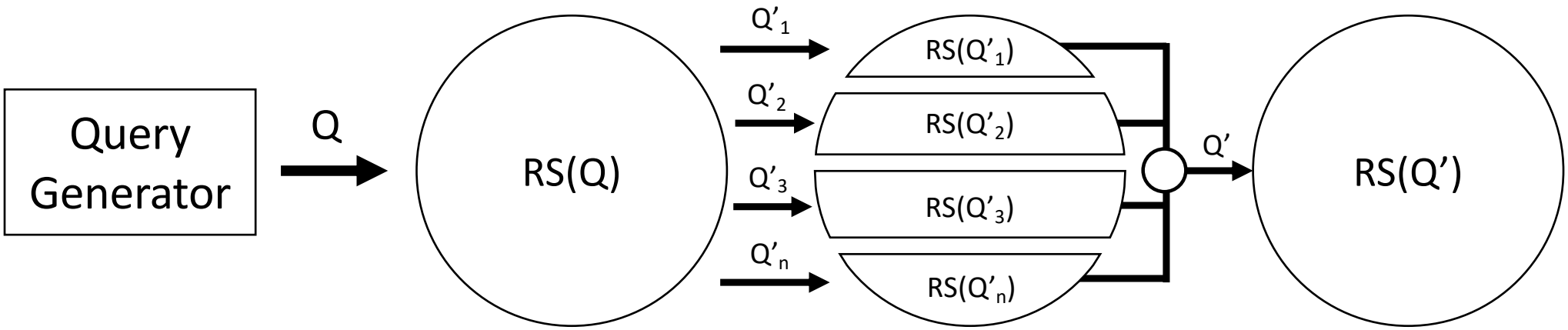
Query Partitioning



Query Partitioning



Query Partitioning



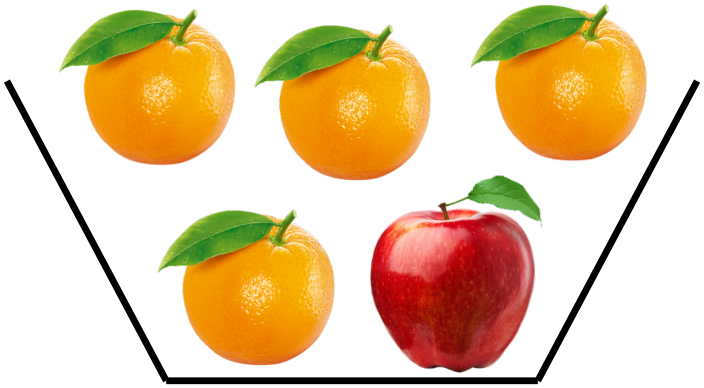
In contrast to differential testing, we need **only a single DBMS**

How to Realize This Idea?

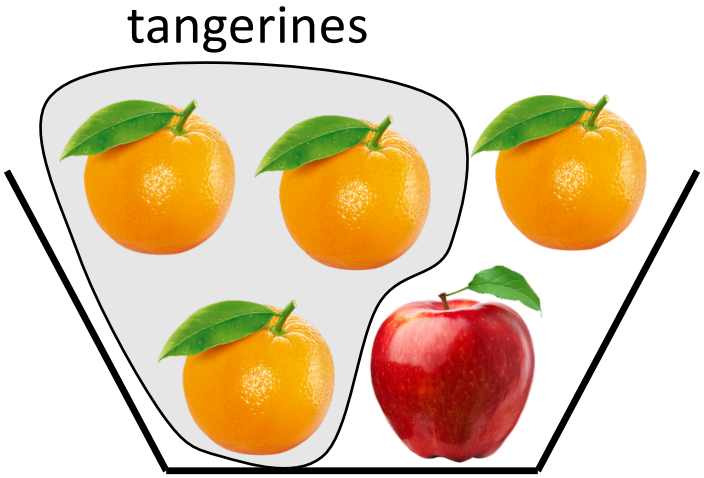
How can we partition the result set?

Scenario: Coffee Kitchen

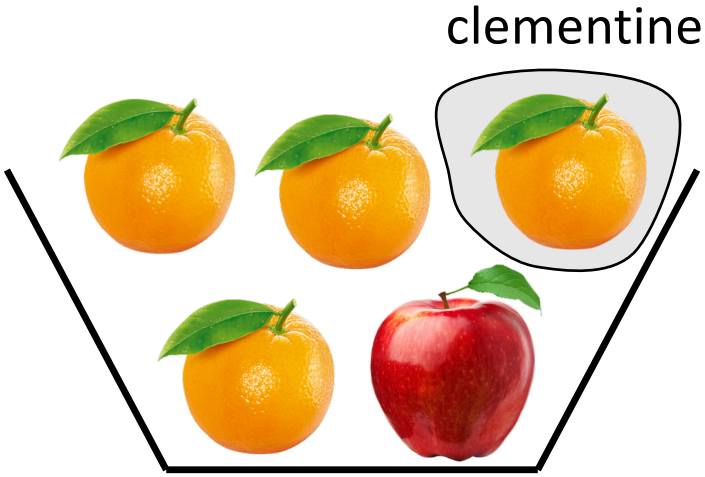
Tangerines vs. Clementines



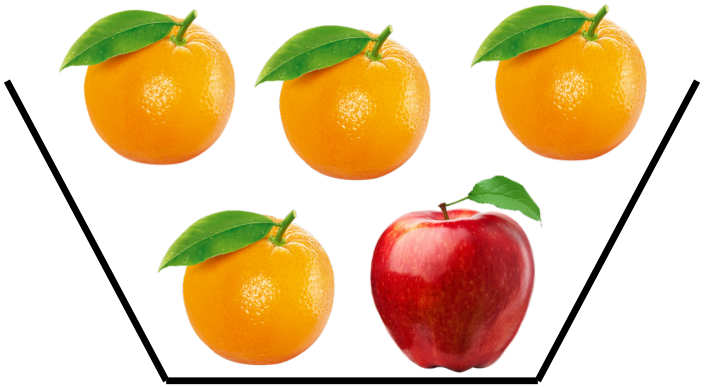
Tangerines vs. Clementines



Tangerines vs. Clementines

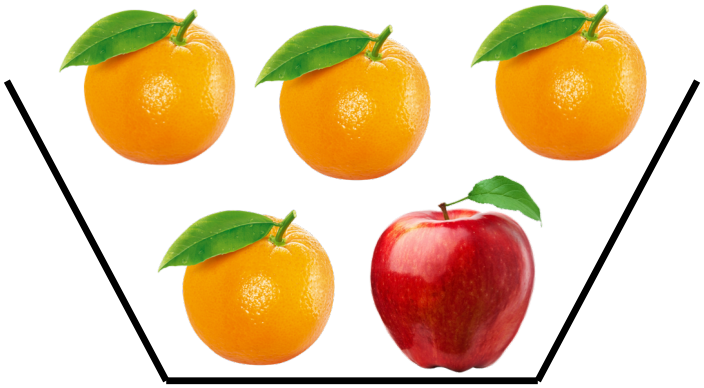


Tangerines vs. Clementines

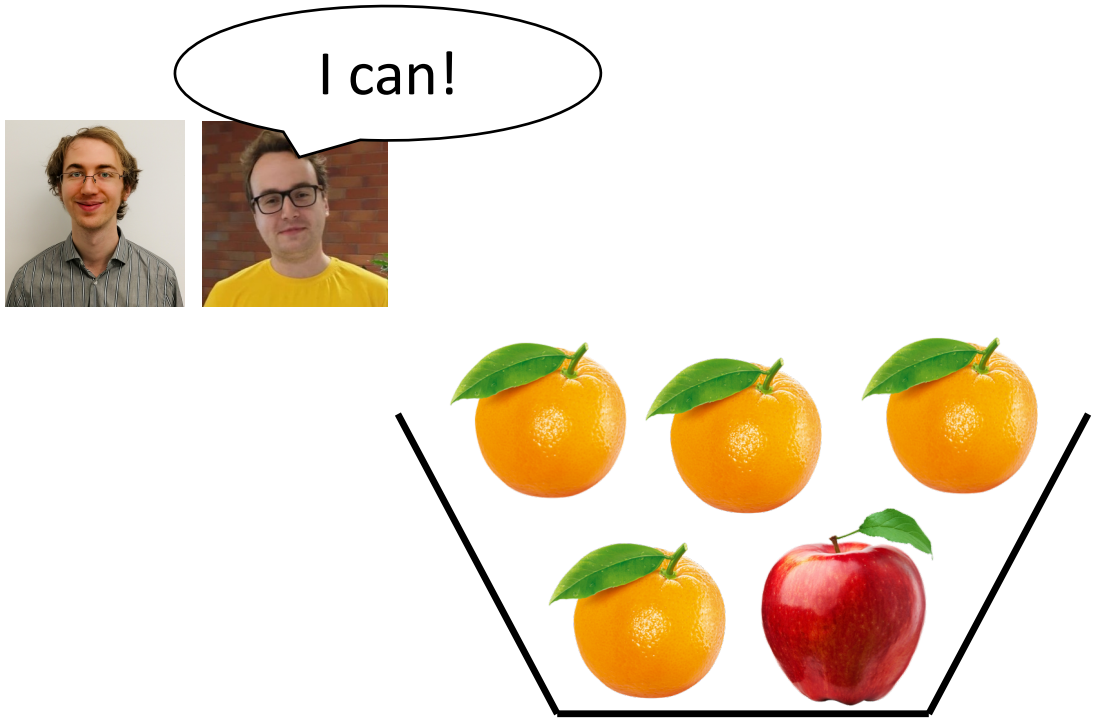


Tangerines vs. Clementines

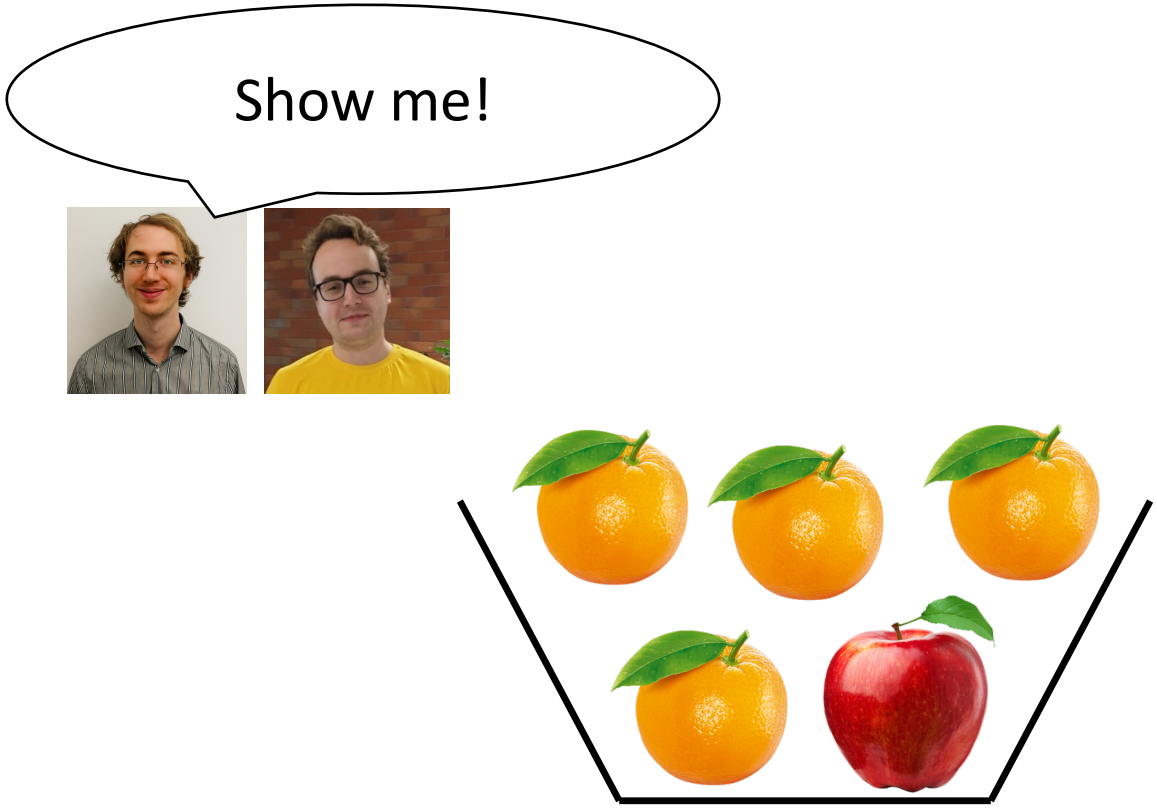
I can never tell different citrus fruits apart



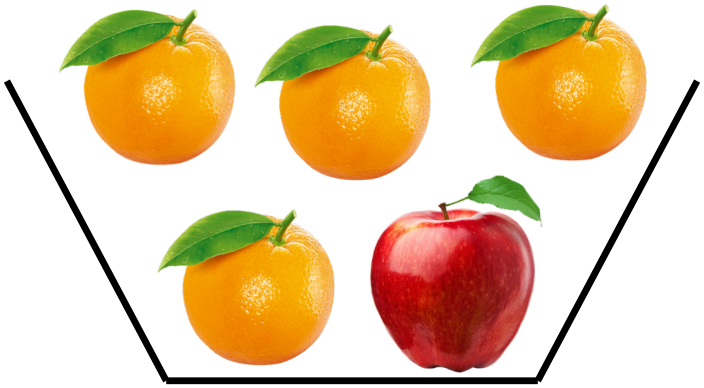
Tangerines vs. Clementines



Tangerines vs. Clementines



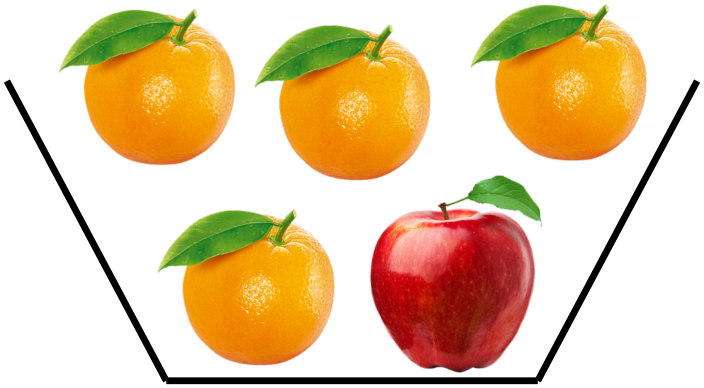
How I test Ilya's understanding without knowing the differences myself?



Tangerines vs. Clementines

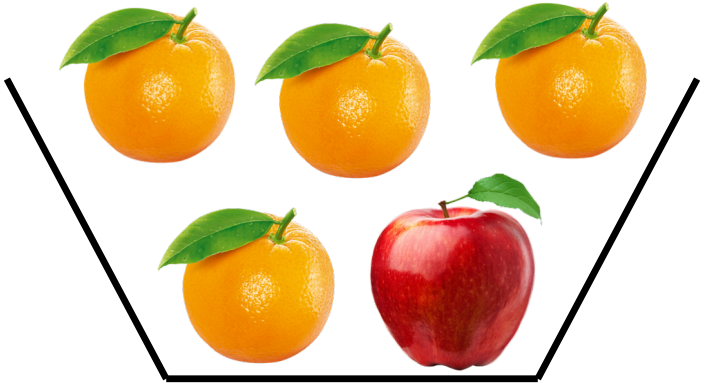
Please bring me all clementines

2 fruits



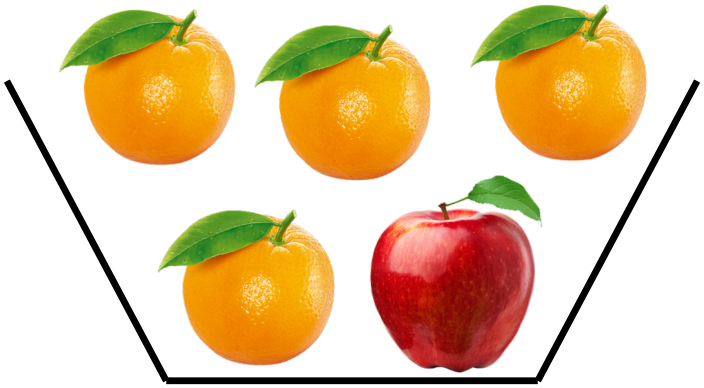
Tangerines vs. Clementines

Please bring me all clementines



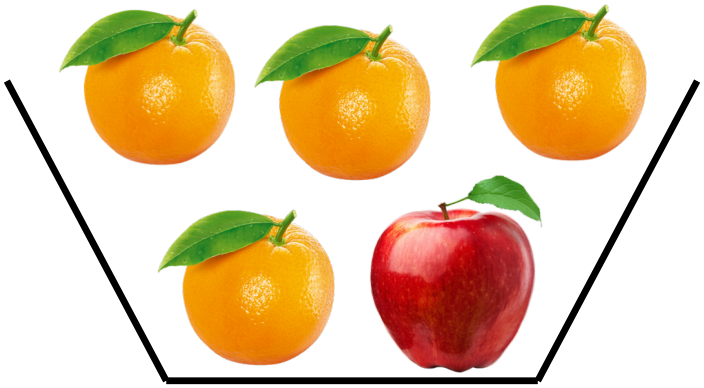
Tangerines vs. Clementines

Please bring me all fruits that are **not clementines**



4 fruits

Tangerines vs. Clementines



2 fruits

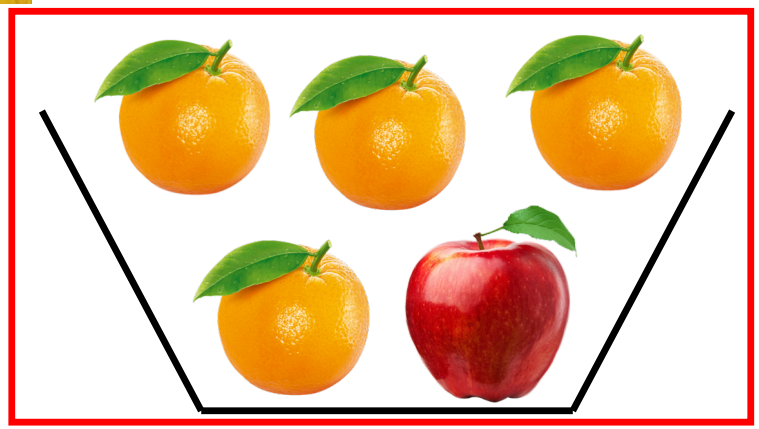
4 fruits

6 fruits

Tangerines vs. Clementines

You likely classified a fruit as **both** a tangerine and a clementine!


5 fruits



2 fruits
4 fruits
6 fruits

Insight

Insight: Every object in a (mathematical) universe is either **a tangerine** or **not a tangerine**

How can we apply this idea to find bugs in DBMSs?

Ternary Logic

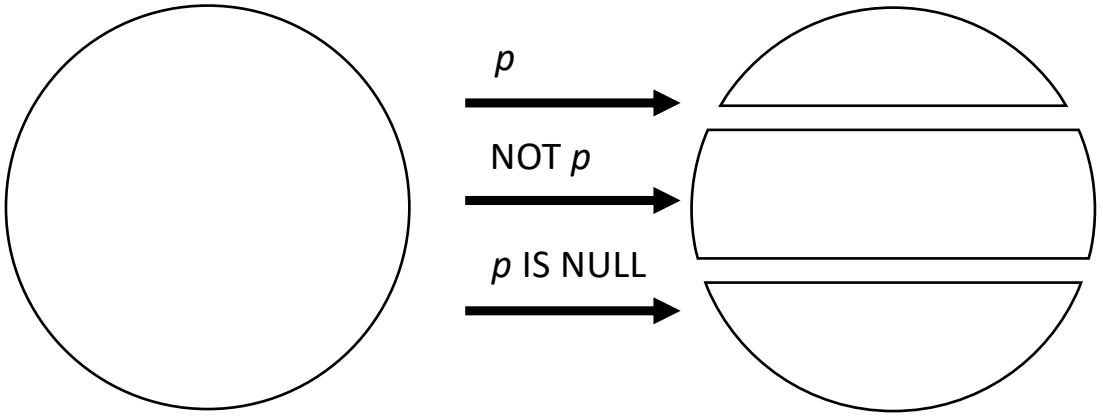
Consider a predicate *p* and a given row r.
Exactly **one** of the following must hold:

- *p*
- NOT *p*
- *p* IS NULL

Ternary Logic

Consider a predicate *p* and a given row r.
Exactly **one** of the following must hold:

- *p*
- NOT *p*
- *p* IS NULL



Motivating Example

t0

c0
0

t1

c0
-0

How did this insight allow us to detect this bug?

```
SELECT * FROM t0, t1
WHERE t0.c0 = t1.c0;
```

t0.c0	t1.c0

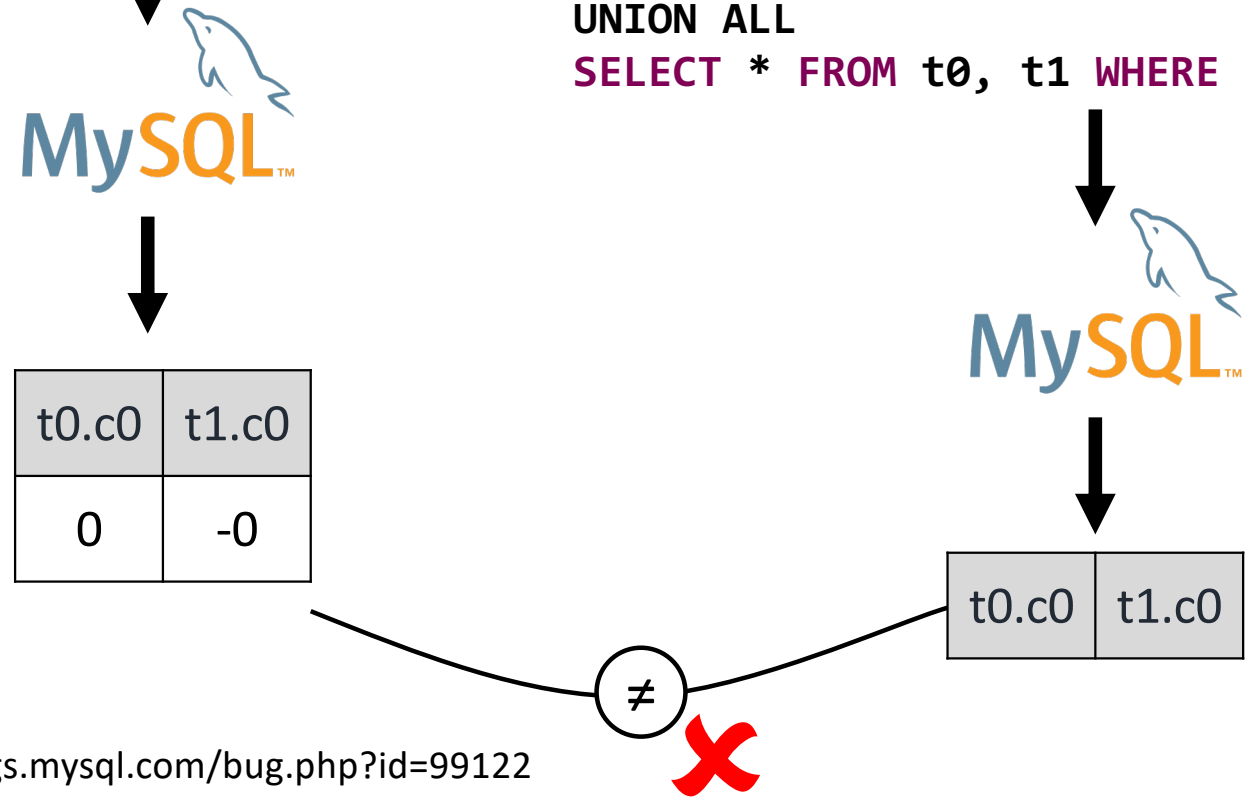
https://bugs.mysql.com/bug.php?id=99122

Example: MySQL

p

```
SELECT * FROM t0, t1;
```

```
SELECT * FROM t0, t1 WHERE t0.c0=t1.c0
UNION ALL
SELECT * FROM t0, t1 WHERE NOT (t0.c0=t1.c0)
UNION ALL
SELECT * FROM t0, t1 WHERE (t0.c0=t1.c0) IS NULL;
```



https://bugs.mysql.com/bug.php?id=99122

Insight

The DBMS is **more likely** to process the **partitioning queries incorrectly** due to their higher complexity

What kind of features we can apply Query Partitioning testing oracle?

Scope

- WHERE
- GROUP BY
- HAVING
- DISTINCT queries
- Aggregate functions

Testing WHERE Clauses

Q	Q'_{ptern}	Composition Operator
SELECT <columns> **FROM <tables>** **[<joins>]**	**SELECT <columns>** **FROM <tables>** **[<joins>]** **WHERE p_{tern}**	UNION ALL

Testing WHERE Clauses

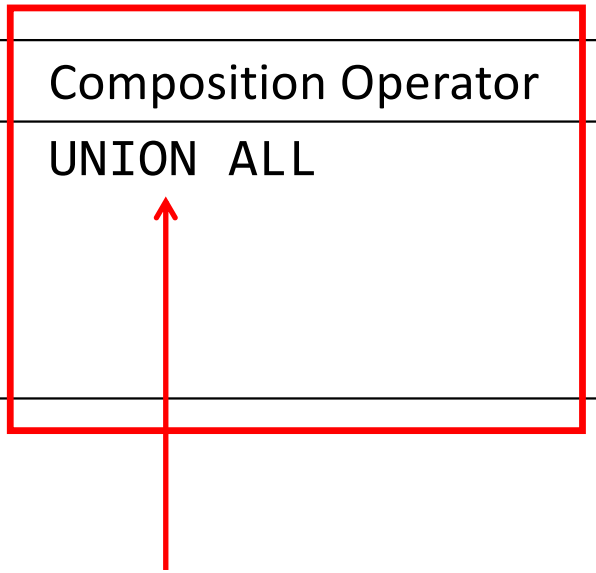
Q	Q'_{ptern}	Composition Operator
SELECT <columns> **FROM <tables>** **[<joins>]**	**SELECT <columns>** **FROM <tables>** **[<joins>]** **WHERE p_{tern}**	UNION ALL

Testing WHERE Clauses

Q	Q'_{ptern}	Composition Operator
SELECT <columns> **FROM <tables>** **[<joins>]**	**SELECT <columns>** **FROM <tables>** **[<joins>]** **WHERE p_{tern}**	UNION ALL

Testing WHERE Clauses

Q	Q'_{ptern}	Composition Operator
SELECT <columns> **FROM <tables>** **[<joins>]**	**SELECT <columns>** **FROM <tables>** **[<joins>]** **WHERE p_{tern}**	UNION ALL



UNION ALL keeps duplicate rows

Scope

- WHERE
- GROUP BY
- HAVING
- DISTINCT queries
- Aggregate functions

Testing DISTINCT Clauses

Q	Q'_{ptern}	Composition operator
SELECT DISTINCT **<columns>** **FROM <tables>** **[<joins>]**	**SELECT** **<columns>** **FROM <tables>** **[<joins>]** **WHERE $p_{tern;}$**	UNION

Testing DISTINCT Clauses

Q	Q'_{ptern}	Composition operator
SELECT DISTINCT **<columns>** **FROM <tables>** **[<joins>]**	**SELECT** **<columns>** **FROM <tables>** **[<joins>]** **WHERE $p_{tern;}$**	UNION

UNION removes duplicate rows

Scope

- WHERE
- GROUP BY
- HAVING
- DISTINCT queries
- Aggregate functions

Testing Self-decomposable Aggregate Functions

Q	Q'_{ptern}	Composition operator

`SELECT MAX(<e>)`
`FROM <tables>`
`[<joins>]`
	`SELECT MAX(<e>)`
	`FROM <tables>`
	`[<joins>]`
	`WHERE` $p_{tern};$
		MAX

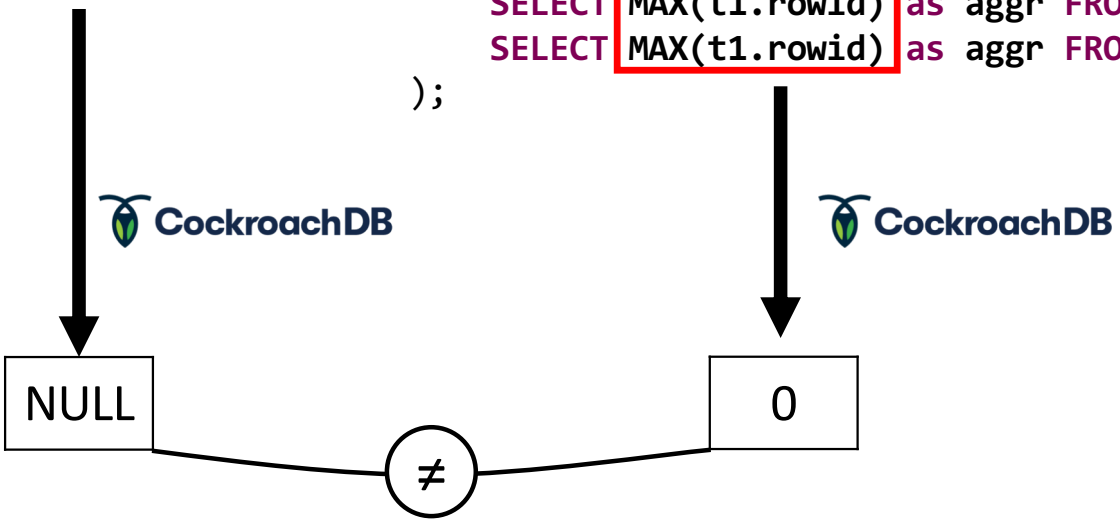
A partition is an **intermediate result**, rather than a subset of the result set

Bug Example: CockroachDB

```
SET vectorize=experimental_on;
CREATE TABLE t0(c0 INT);
CREATE TABLE t1(c0 BOOL) INTERLEAVE IN PARENT t0(rowid);
INSERT INTO t0(c0) VALUES (0);
INSERT INTO t1(rowid, c0) VALUES(0, TRUE);
```

```
SELECT MAX(t1.rowid)
FROM t1;
```

```
SELECT MAX(aggr) FROM (
    SELECT MAX(t1.rowid) as aggr FROM t1 WHERE '+' >= t1.c0 UNION ALL
    SELECT MAX(t1.rowid) as aggr FROM t1 WHERE NOT('+' >= t1.c0) UNION ALL
    SELECT MAX(t1.rowid) as aggr FROM t1 WHERE ('+' >= t1.c0) IS NULL
);
```

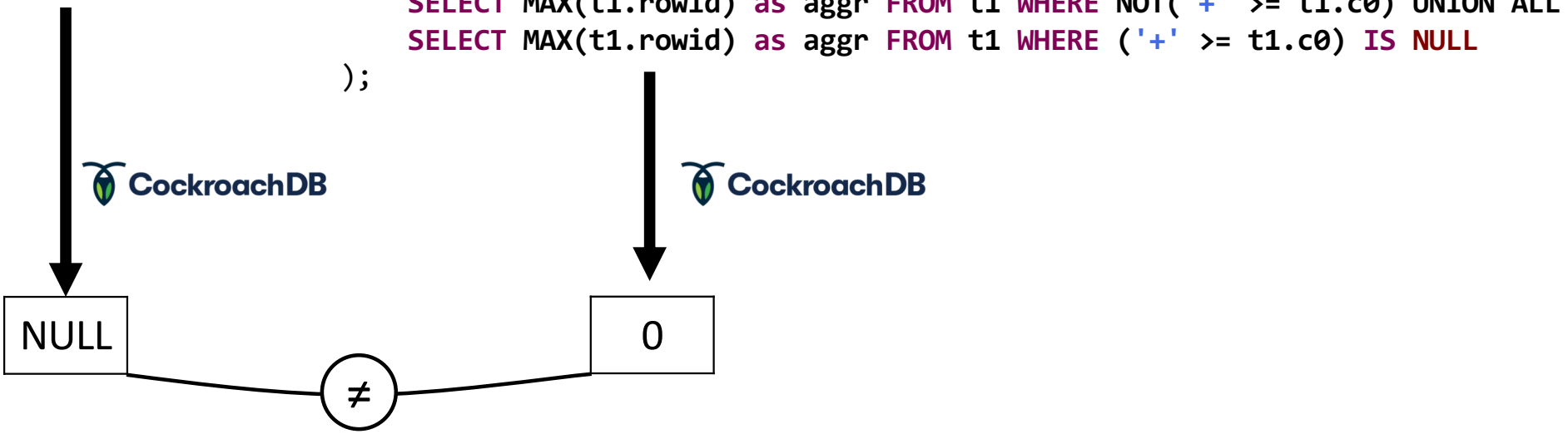


Bug Example: CockroachDB

```
SET vectorize=experimental_on;
CREATE TABLE t0(c0 INT);
CREATE TABLE t1(c0 BOOL) INTERLEAVE IN PARENT t0(rowid);
INSERT INTO t0(c0) VALUES (0);
INSERT INTO t1(rowid, c0) VALUES(0, TRUE);
```

```
SELECT MAX(t1.rowid)
FROM t1;
```

```
SELECT MAX(aggr) FROM (
    SELECT MAX(t1.rowid) as aggr FROM t1 WHERE '+' >= t1.c0 UNION ALL
    SELECT MAX(t1.rowid) as aggr FROM t1 WHERE NOT('+' >= t1.c0) UNION ALL
    SELECT MAX(t1.rowid) as aggr FROM t1 WHERE ('+' >= t1.c0) IS NULL
);
```



Testing Decomposable Aggregate Functions

Q	Q'_{ptern}	Composition operator

`SELECT AVG(<e>)`
`FROM <tables>`
`[<joins>];`
	`SELECT SUM(<e>) as s,`
	`COUNT(<e>) as c`
	`FROM <tables>`
	`[<joins>];`
		$\frac{SUM(s)}{SUM(c)}$

A **single value** to represent a partition is **insufficient**

Bug example: ClickHouse

```
CREATE TABLE t3 (`c0` Int32, `c1` Int32, `c2` String) ENGINE = Log()
INSERT INTO t3(c0,c1,c2) VALUES (1,10,'1'), (1,0,'2');

SELECT *
FROM t3
```

```
┌─c0─┬─c1─┬─c2─┐
│  1 │ 10 │ 1  │
│  1 │  0 │ 2  │
└────┴────┴────┘
```

```
SELECT MIN(c2)
FROM t3
GROUP BY c0
```

```
┌─MIN(c2)─┐
│ 1       │
└─────────┘
```

```
SELECT MIN(t3.c2)
FROM t3
GROUP BY t3.c0
HAVING NOT t3.c1
UNION ALL
SELECT MIN(t3.c2)
FROM t3
GROUP BY t3.c0
HAVING NOT (NOT t3.c1)
UNION ALL
SELECT MIN(t3.c2)
FROM t3
GROUP BY t3.c0
HAVING isNull(NOT t3.c1)
```

```
┌─MIN(c2)─┐
│ 2       │
│ 1       │
└─────────┘
```

Bug example: ClickHouse

```
SELECT MIN(t3.c2)
FROM t3
GROUP BY t3.c0
HAVING NOT t3.c1
```

Incorrect Query should return an error, but not incorrect answer.

Each column reference directly contained in the search condition shall be one of the following:
a) An unambiguous reference to a column that is functionally dependent on the set consisting of every column referenced by a column reference contained in group by clause.
...

How general is the technique?
Can I apply it to other domains?

Metamorphic Testing

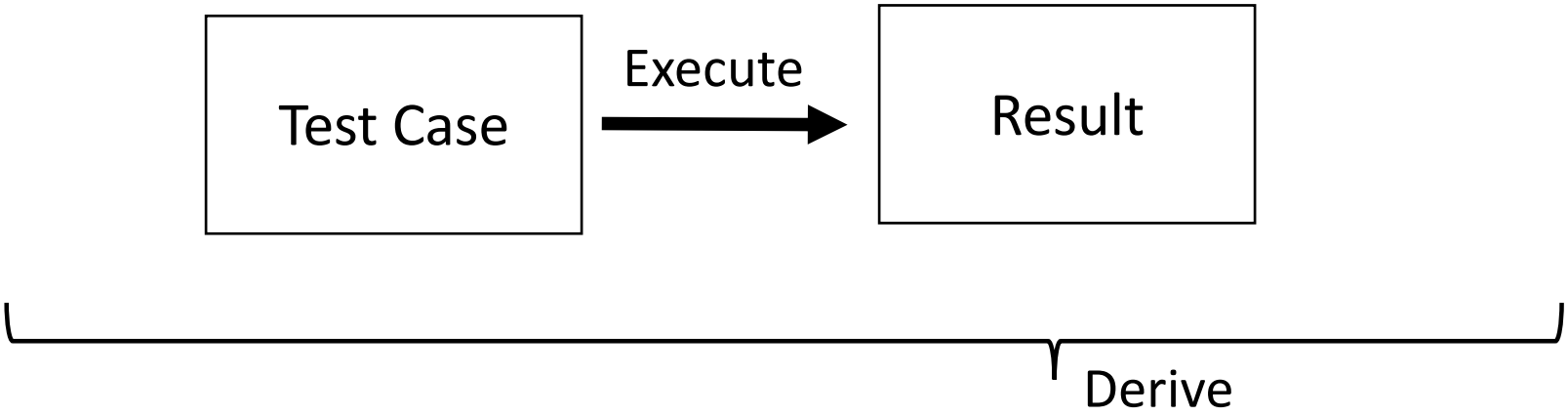
```
SELECT * FROM t0, t1
WHERE t0.c0 = t1.c0;
```

t0.c0	t1.c0
0.0	-0.0

Derive

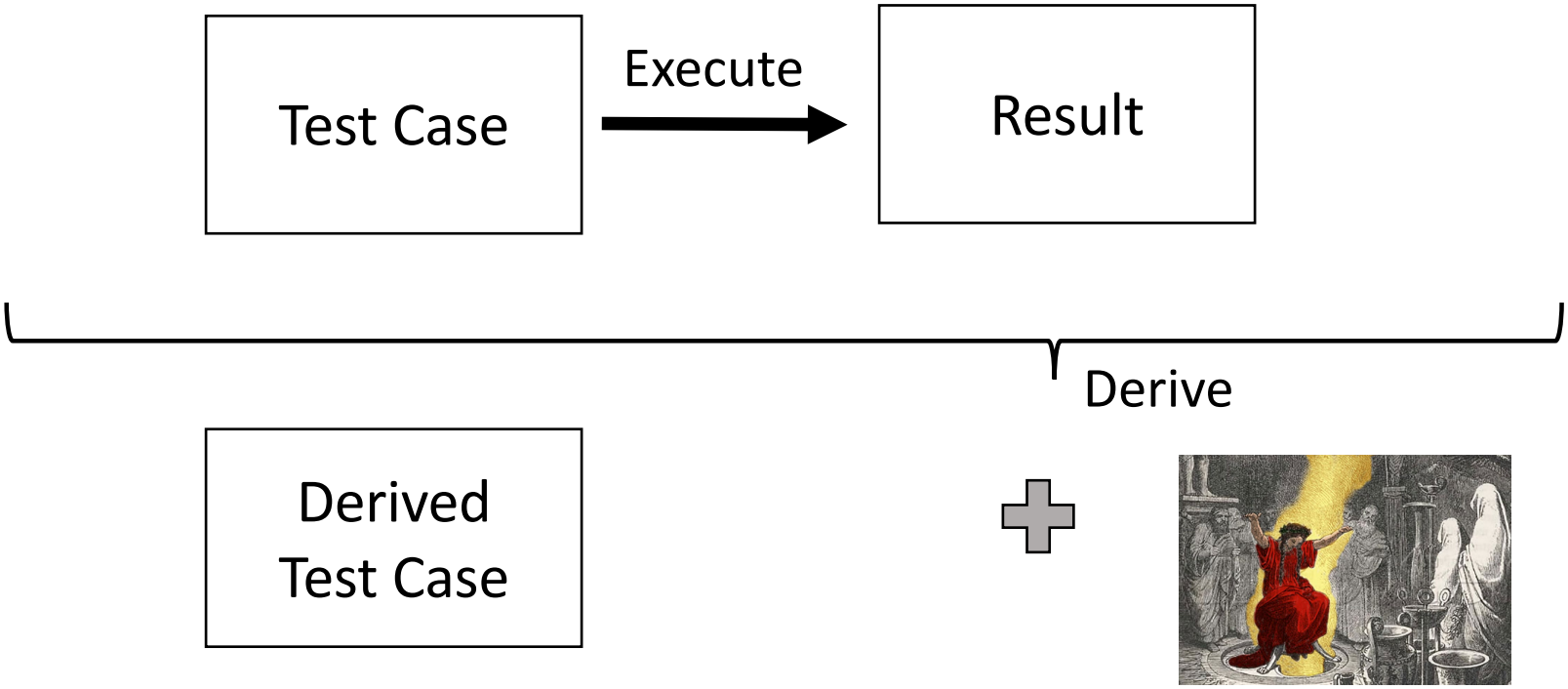
```
SELECT * FROM t0, t1 WHERE t0.c0=t1.c0
UNION ALL
SELECT * FROM t0, t1 WHERE NOT (t0.c0=t1.c0)
UNION ALL
SELECT * FROM t0, t1 WHERE (t0.c0=t1.c0) IS NULL;
```

Metamorphic Testing



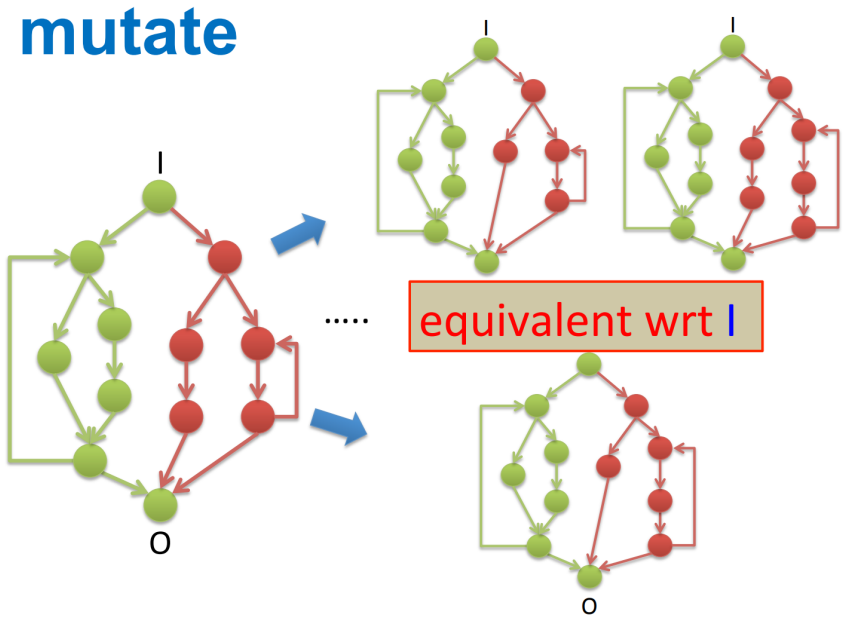

```
SELECT * FROM t0, t1 WHERE t0.c0=t1.c0
UNION ALL
SELECT * FROM t0, t1 WHERE NOT (t0.c0=t1.c0)
UNION ALL
SELECT * FROM t0, t1 WHERE (t0.c0=t1.c0) IS NULL;
```

Metamorphic Testing



This technique is known as metamorphic testing

Equivalence Modulo Inputs (EMI) for Testing Compilers



EMI's idea is to create programs that produce **the same output for a given input**

https://people.inf.ethz.ch/suz/emi/index.html

Can we test not only DBMS?

Approach is quite generic.

We can try to use it in

1. Generic API with filtering or grouping
2. Regular expressions library
3. Event processing
4. Image recognition
5. Your ideas

What about other metamorphic test oracles for DBMSs?

Finding Logic Bugs in DBMSs

Ternary Logic Partitioning

OOPSLA ‘20

Non-optimizing Reference Engine Construction

ESEC/FSE ‘20

Pivoted Query Synthesis

OSDI ‘20

Goal: Find Logic Bugs

Optimization bugs: logic bugs in the query optimizer

Motivating Example

t0

c0
-1

```
CREATE TABLE t0(c0 UNIQUE);
INSERT INTO t0 VALUES (-1) ;
SELECT * FROM t0 WHERE t0.c0 GLOB '-*';
```

https://www.sqlite.org/src/tktview?name=0f0428096f

Motivating Example

t0

c0
-1

```
CREATE TABLE t0(c0 UNIQUE);
INSERT INTO t0 VALUES (-1) ;
SELECT * FROM t0 WHERE t0.c0 GLOB '-*';
```

https://www.sqlite.org/src/tktview?name=0f0428096f

Motivating Example

t0

c0
-1

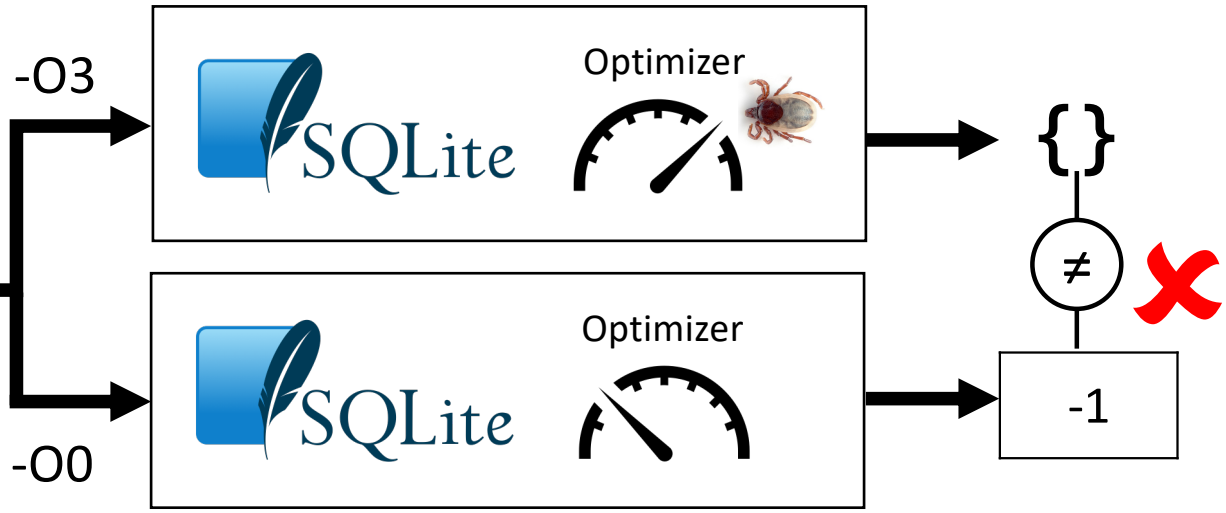
```
CREATE TABLE t0(c0 UNIQUE);
INSERT INTO t0 VALUES (-1) ;
SELECT * FROM t0 WHERE t0.c0 GLOB '-*';
```

The *LIKE optimization* malfunctioned for non-text columns and a pattern prefix of “-”

https://www.sqlite.org/src/tktview?name=0f0428096f

Differential Testing

```
SELECT * FROM t0
WHERE t0.c0 GLOB '-*';
```



Differential Testing

List Of PRAGMAs

analysis_limit
application_id
auto_vacuum
automatic_index
busy_timeout
cache_size
cache_spill
case_sensitive_like
cell_size_check
checkpoint_fullfsync
collation_list
compile_options
~~count_changes[1]~~
~~data_store_directory[1]~~
data_version
database_list
~~default_cache_size[1]~~
defer_foreign_keys
~~empty_result_callbacks[1]~~
encoding
foreign_key_check
foreign_key_list
foreign_keys
freelist_count
~~full_column_names[1]~~
fullfsync
function_list
hard_heap_limit
ignore_check_constraints
incremental_vacuum
index_info
index_list
index_xinfo
integrity_check
journal_mode
journal_size_limit
legacy_alter_table
legacy_file_format
locking_mode
max_page_count
mmap_size
module_list
optimize
page_count
page_size
parser_trace[2]
pragma_list
query_only
quick_check
read_uncommitted
recursive_triggers
reverse_unordered_selects
schema_version[3]
secure_delete
~~short_column_names[1]~~
shrink_memory
soft_heap_limit
stats[3]
synchronous
table_info
table_xinfo
temp_store
~~temp_store_directory[1]~~
threads
trusted_schema
user_version
vdbe_addoptrace[2]
vdbe_debug[2]
vdbe_listing[2]
vdbe_trace[2]
wal_autocheckpoint
wal_checkpoint
writable_schema[3]

https://www.sqlite.org/pragma.html

Differential Testing

List Of PRAGMAs

analysis_limit
application_id
auto_vacuum
automatic_index
busy_timeout
cache_size
cache_spill
case_sensitive_like
cell_size_check
checkpoint_fullfsync
collation_list
compile_options
count_changes[1]
data_store_directory[1]
data_version
database_list
default_cache_size[1]
defer_foreign_keys
empty_result_callbacks[1]
encoding
foreign_key_check
foreign_key_list
foreign_keys
freelist_count
full_column_names[1]
fullfsync
function_list
hard_heap_limit
ignore_check_constraints
incremental_vacuum
index_info
index_list
index_xinfo
integrity_check
journal_mode
journal_size_limit
legacy_alter_table
legacy_file_format
locking_mode
max_page_count
mmap_size
page_count
page_size
parser_trace[2]
pragma_list
query_only
quick_check
read_uncommitted
recursive_triggers
reverse_unordered_selects
schema_version[3]
secure_delete
short_column_names[1]
shrink_memory
soft_heap_limit
stats[3]
synchronous
table_info
temp_store
temp_store_directory[1]
threads
user_version
vdbe_listing[2]
vdbe_trace[2]
wal_autocheckpoint
wal_checkpoint
writable_schema[3]

PRAGMA case_sensitive_like = *boolean*;

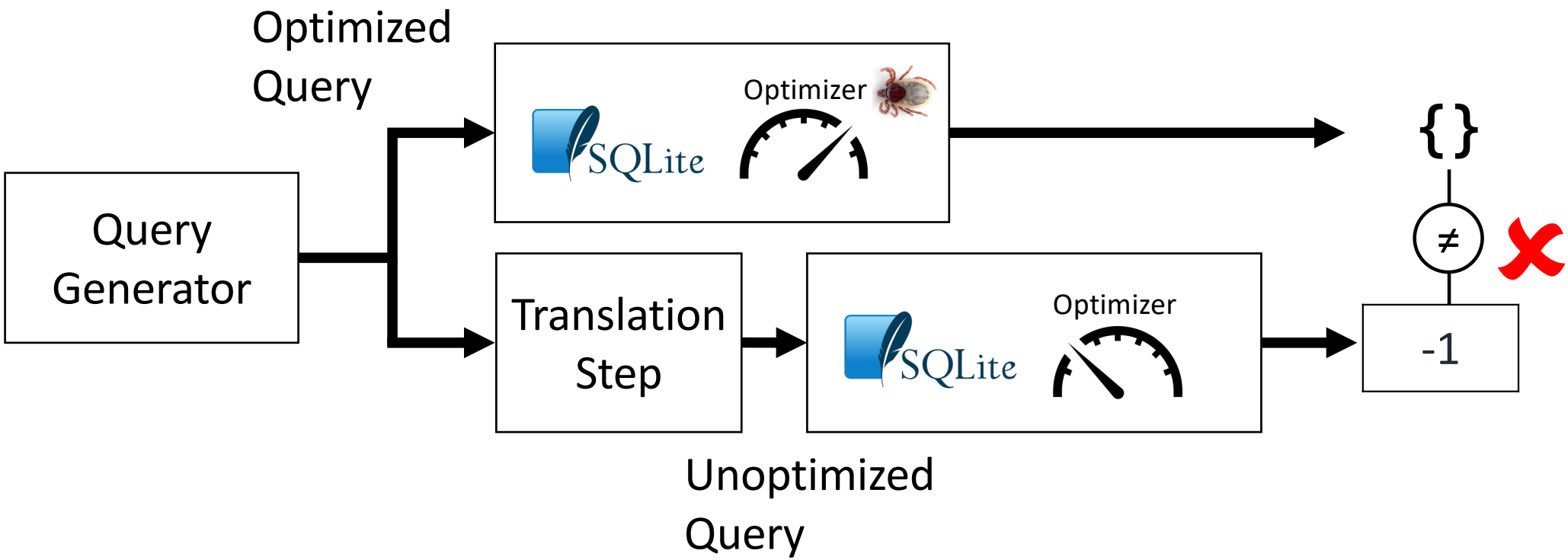
DBMSs typically provide only very **limited control** over optimizations

https://www.sqlite.org/pragma.html

NoREC

Idea: **Rewrite** the query so that the DBMS **cannot optimize it**

Idea



We want to create a “non-optimizing reference engine”

Given Query

Consider the following format for the optimized query:

```
SELECT * FROM t0
WHERE p;
```

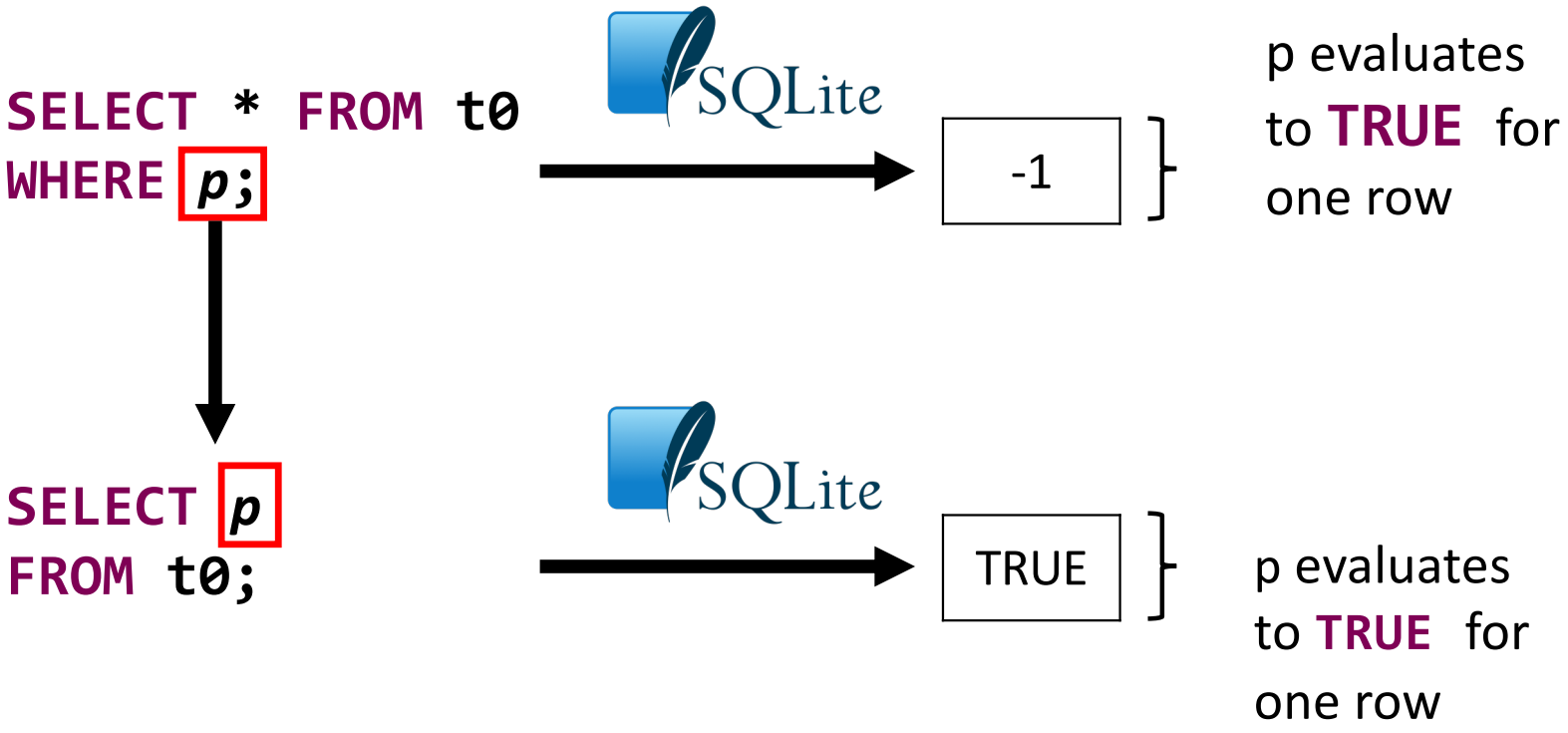
```
t0.c0 GLOB '-*'
```

It is unobvious how we could derive an unoptimized query

Insight

First Insight: The predicate ***p*** must **always evaluate to the same value**, irrespective of its context

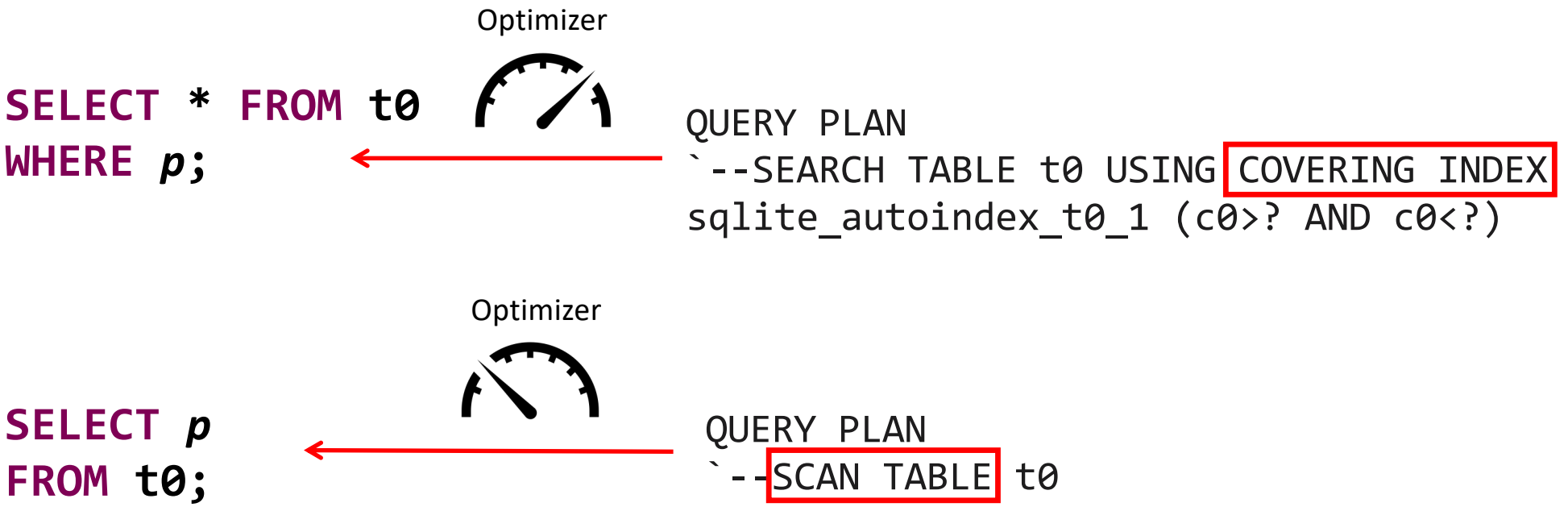
Translation Step (Correct Case)



Insights

Second Insight: DBMSs focus their optimizations on reducing the amount of data that is processed

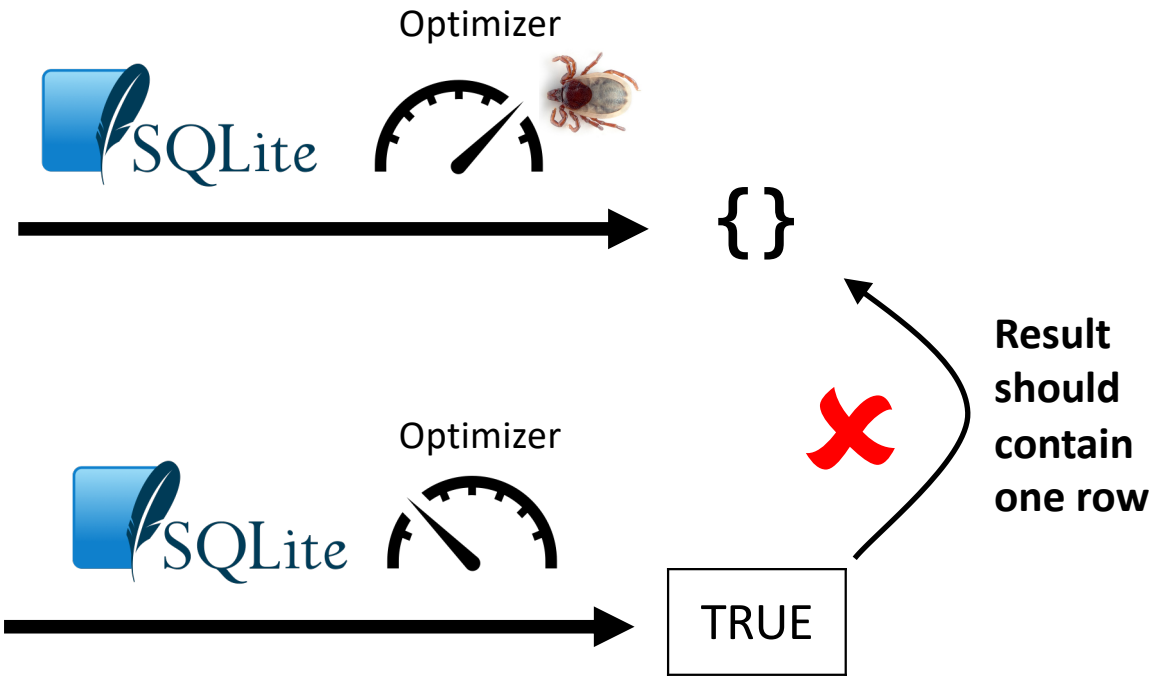
Translation Step



Translation Step

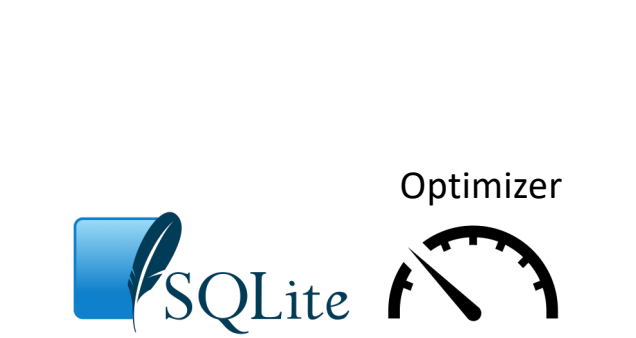
```
SELECT * FROM t0
WHERE p;
```

```
SELECT p
FROM t0;
```

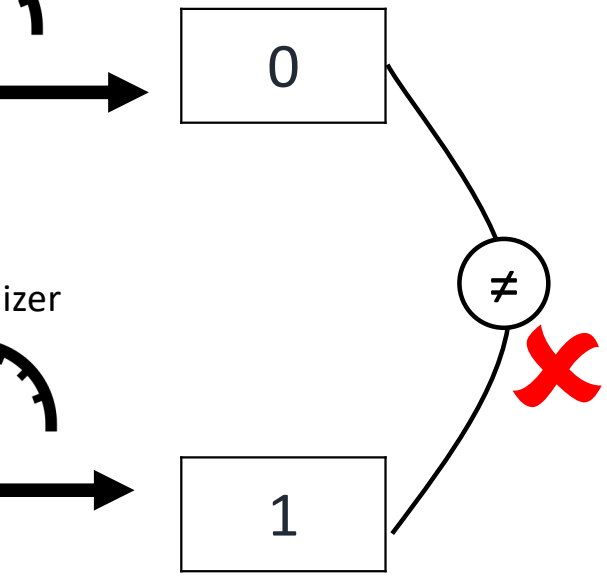


Counting Implementation

```
SELECT COUNT(*)
FROM …
WHERE p
```




```
SELECT SUM(count) FROM (
      SELECT p IS TRUE
        as count
      FROM <tables>
);
```



What about other, non-metamorphic test oracles?

Finding Logic Bugs in DBMSs

Ternary Logic Partitioning

OOPSLA ‘20

Non-optimizing Reference Engine Construction

ESEC/FSE ‘20

Pivoted Query Synthesis

OSDI ‘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;
```

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

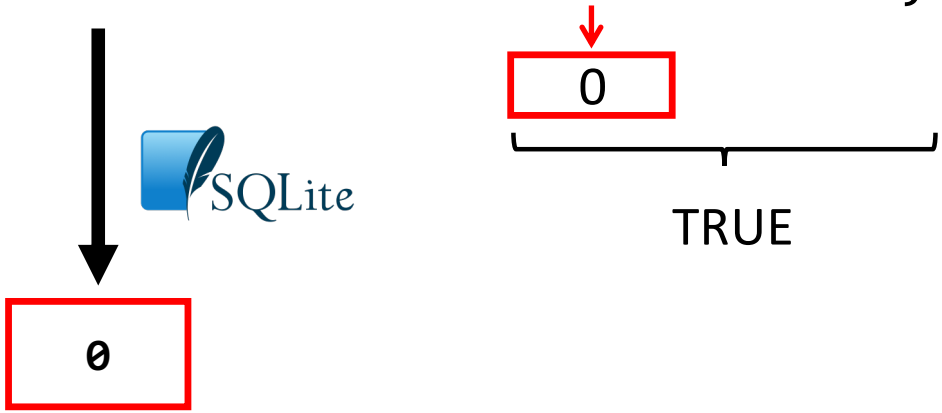
https://sqlite.org/src/tktview/80256748471a01

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;
```



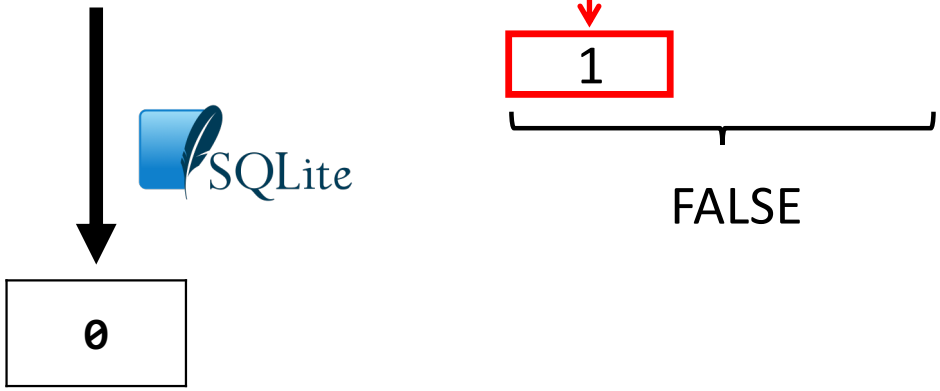
https://sqlite.org/src/tktview/80256748471a01

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;
```



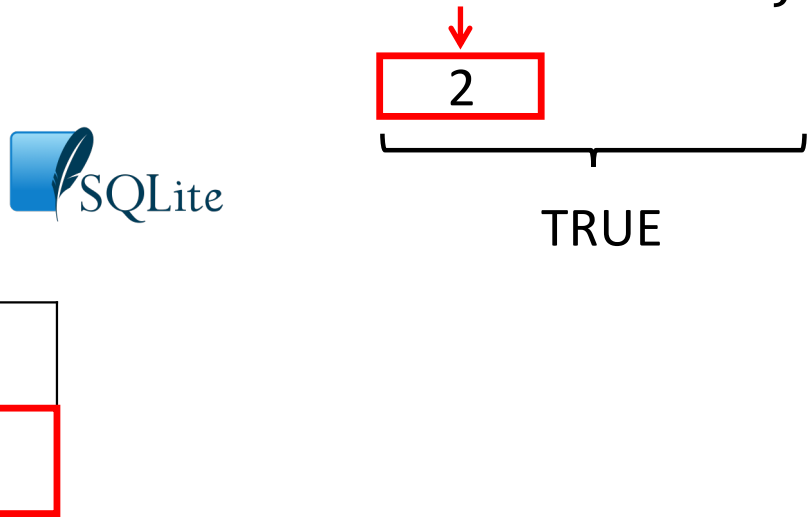
https://sqlite.org/src/tktview/80256748471a01

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;
```

NULL

TRUE

0
2

NULL was not contained in the result set!

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

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

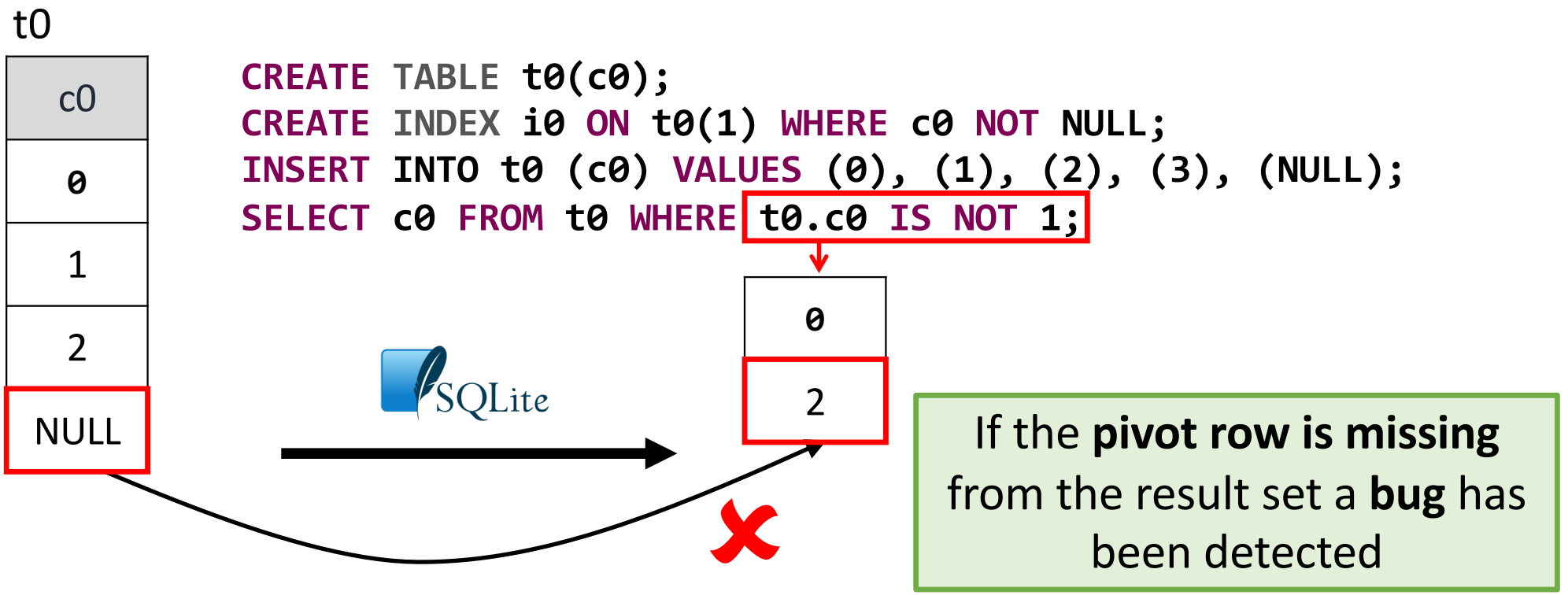
```
CREATE TABLE t0(c0);
CREATE INDEX i0 ON t0(1) WHERE c0 NOT NULL;
INSERT INTO t0 (c0) VALUES (0), (1), (2), (3), (NULL);
SELECT c0 FROM t0 WHERE t0.c0 IS NOT 1;
```

NULL

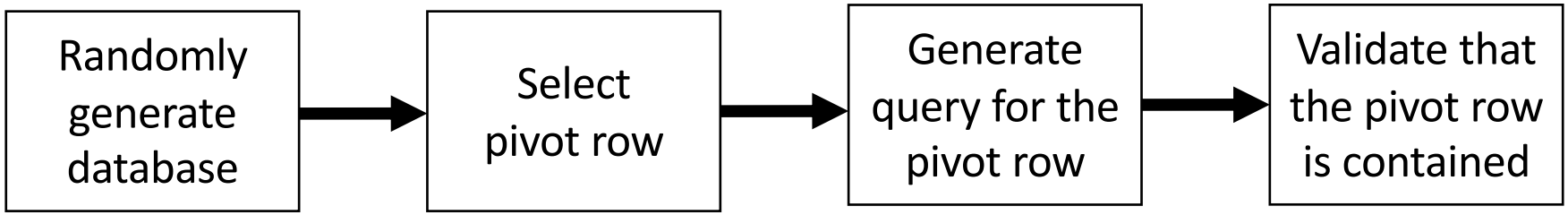
TRUE

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

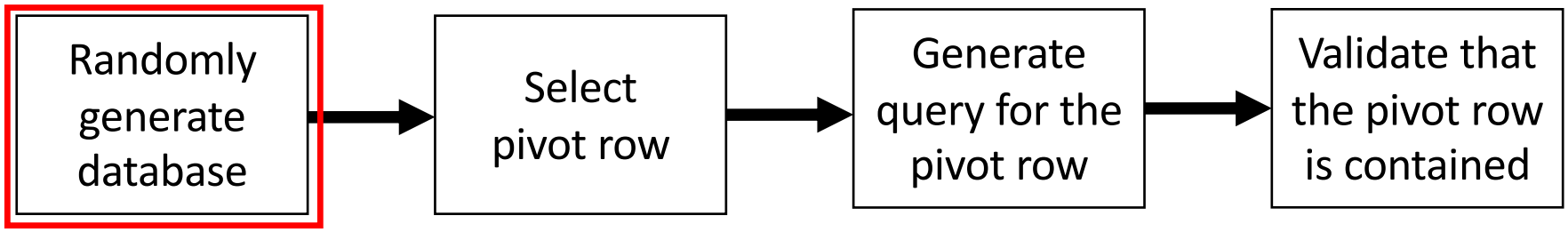
PQS Idea


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;
SQLite
0
2
If the **pivot row is missing** from the result set a **bug** has been detected

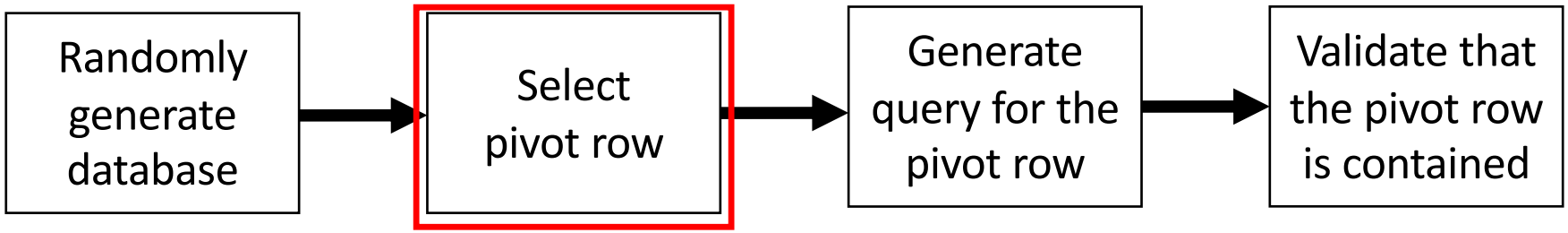
Approach


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

Approach

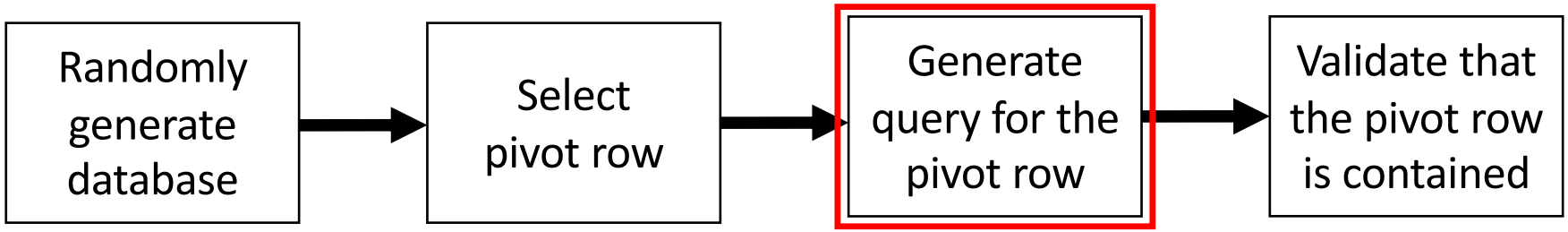

Randomly generate database
Select pivot row
Generate query for the pivot row
Validate that the pivot row is contained

Approach



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

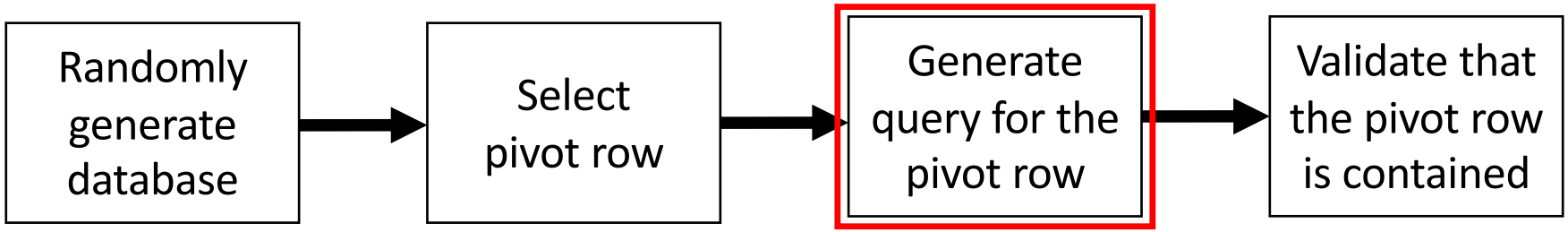
Approach

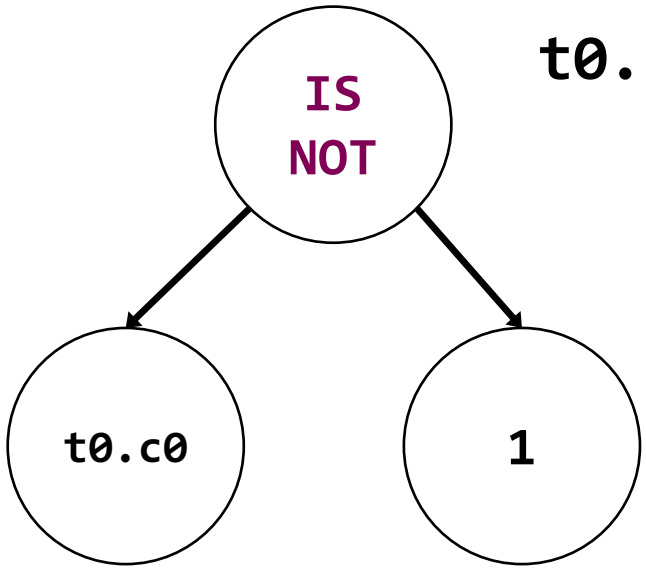



```
SELECT c0 FROM t0
WHERE
```

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

Random Expression Generation

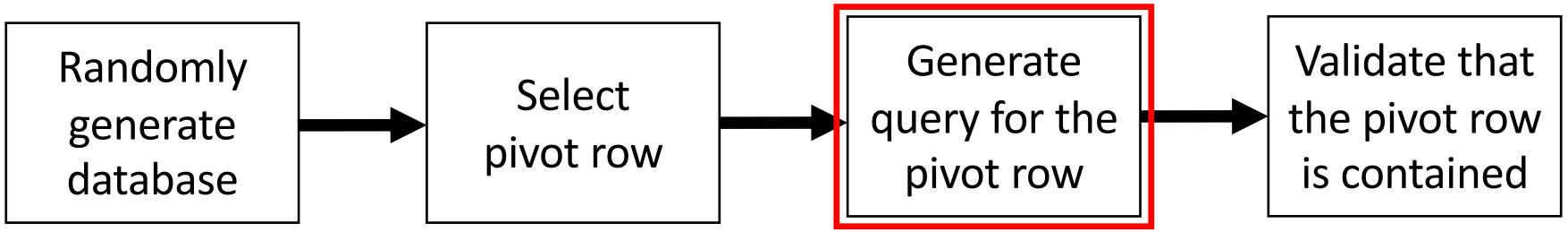





```
t0.c0 IS NOT 1;
```

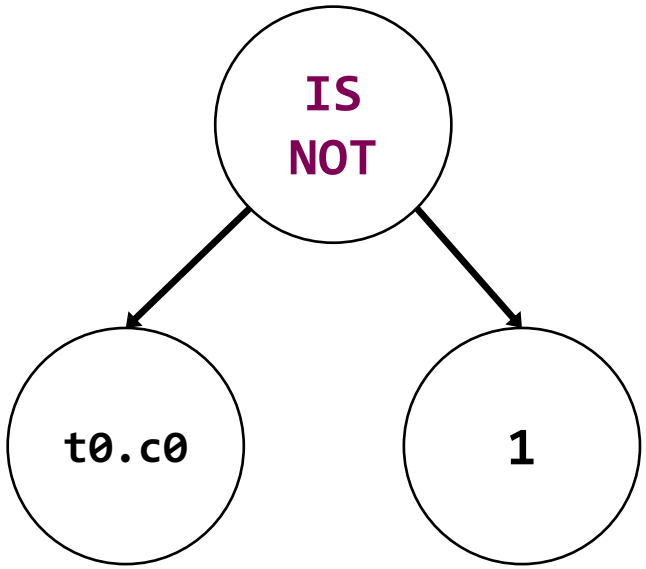
We implemented an **expression evaluator** for each node

Random Expression Generation



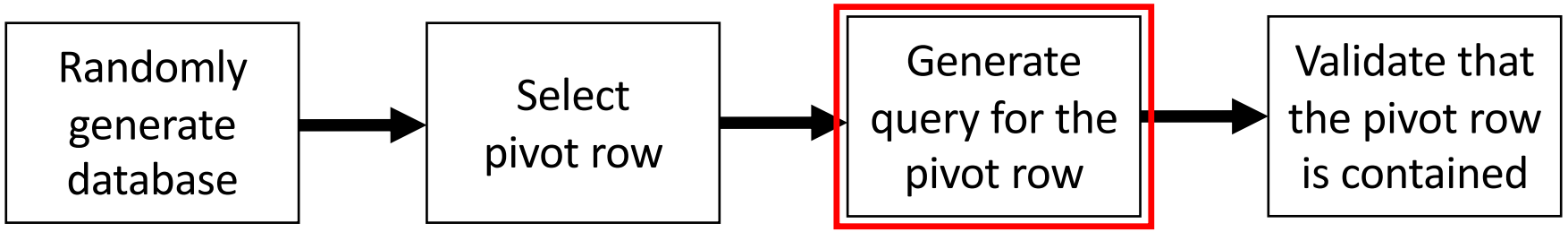
t0

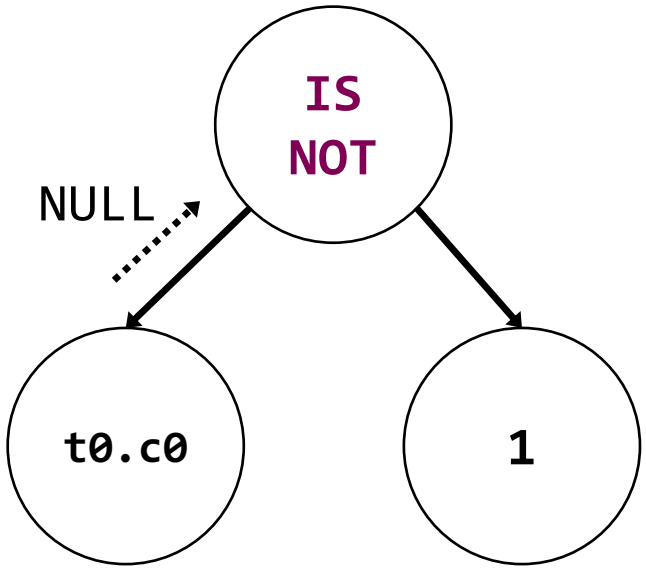
c0
0
1
2
NULL



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

Random Expression Generation



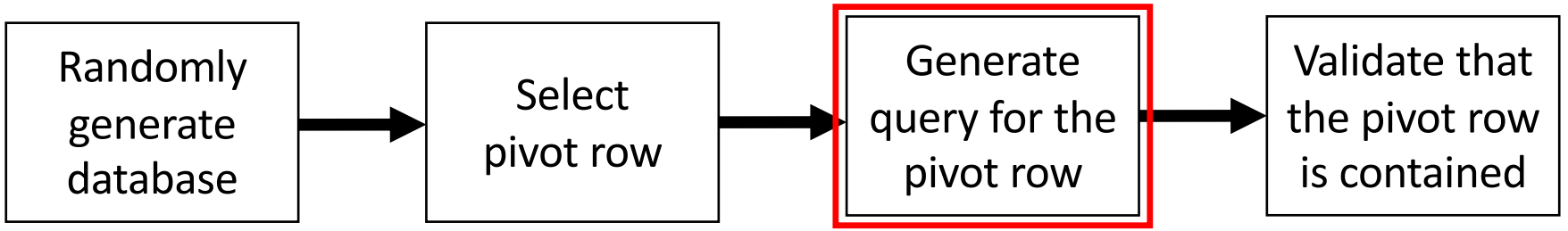


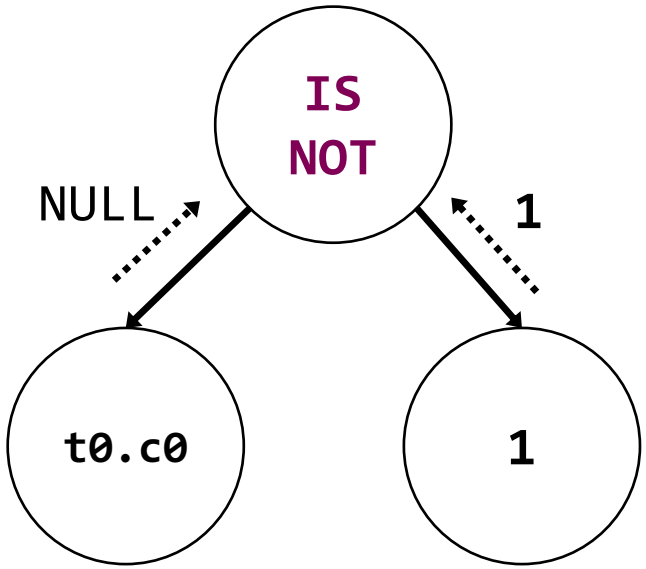
Column references return the **values** from the **pivot row**

t0

c0
0
1
2
NULL

Random Expression Generation



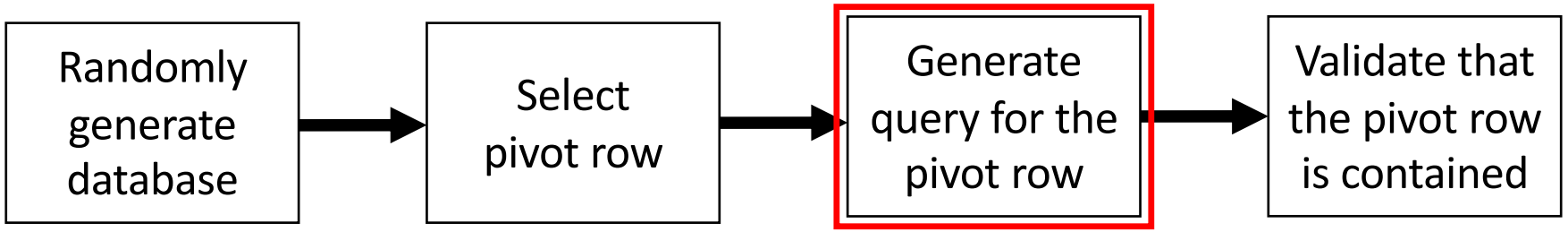


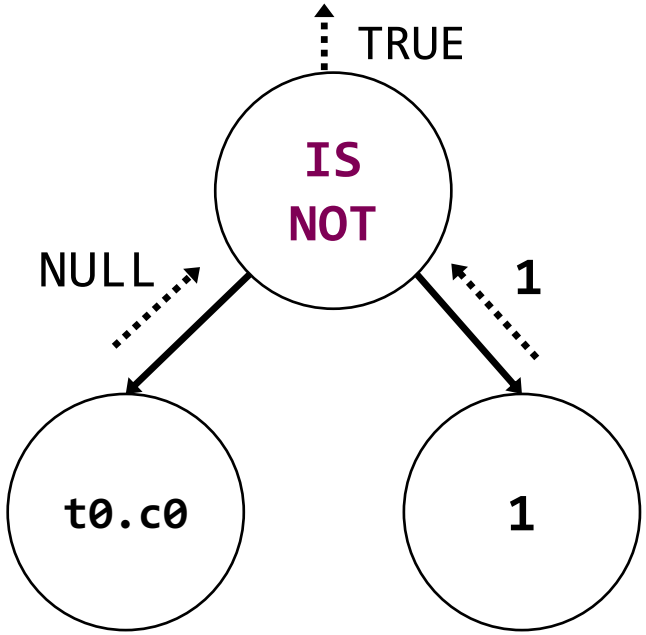
Constant nodes return their assigned **literal values**

t0

c0
0
1
2
NULL

Random Expression Generation



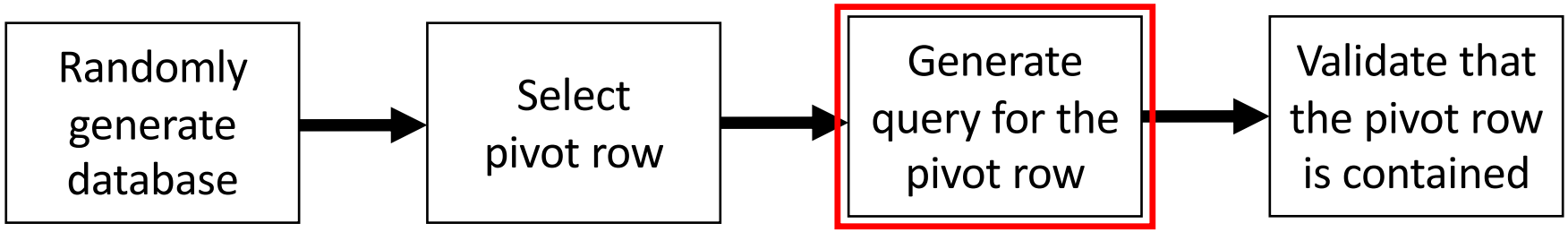


Compound nodes
compute their result
based on their children

t0

c0
0
1
2
NULL

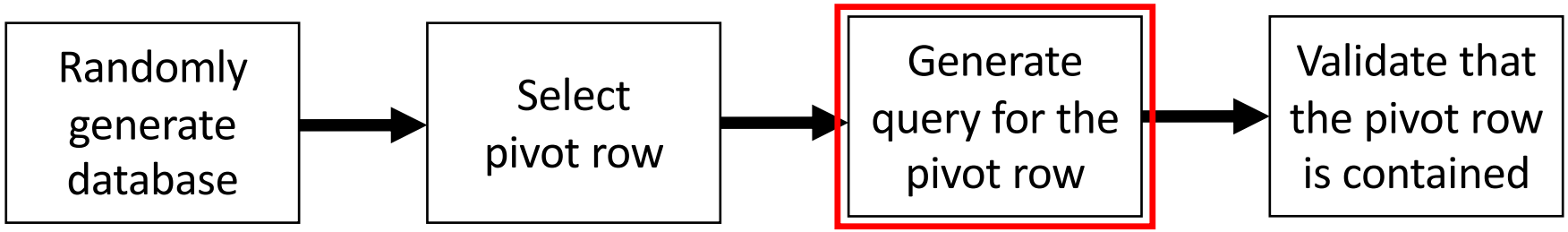
Query Synthesis




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

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

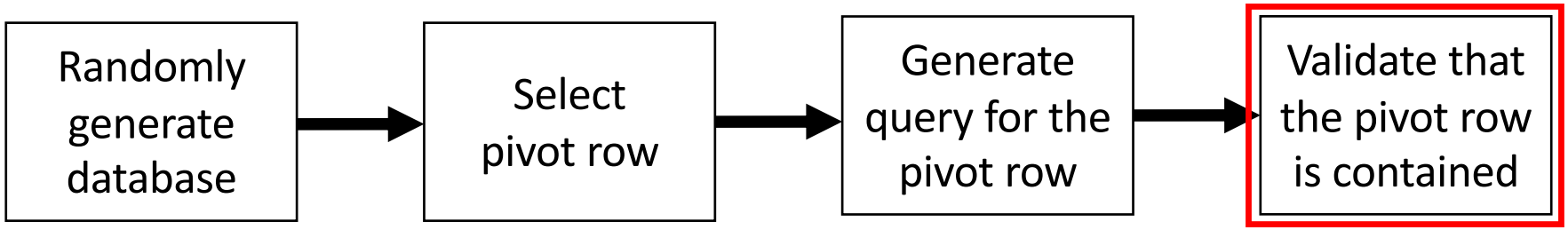
Random Expression Rectification




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

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

Approach




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

Rely on the DBMS to check whether the row is contained

How do the techniques compare to each other?

Comparison

Property	PQS	NoREC	TLP
WHERE	✓	✓	✓
Additional SQL features	✗	✗	✓
Ground truth	✓	✗	✗
No domain knowledge required	✗	✓	✓
Implementation effort	Moderate	Very Low	Low

Comparison

Property	PQS	NoREC	TLP
WHERE	✓	✓	✓
Additional SQL features	✗	✗	✓
Ground truth	✓	✗	✗
No domain knowledge required	✗	✓	✓
Implementation effort	Moderate	Very Low	Low

TLP is applicable to testing a wider range of features

Comparison

Property	PQS	NoREC	TLP
WHERE	✓	✓	✓
Additional SQL features	✗	✗	✓
Ground truth	✓	✗	✗
No domain knowledge required	✗	✓	✓
Implementation effort	Moderate	Very Low	Low

Both NoREC and TLP are metamorphic testing approaches

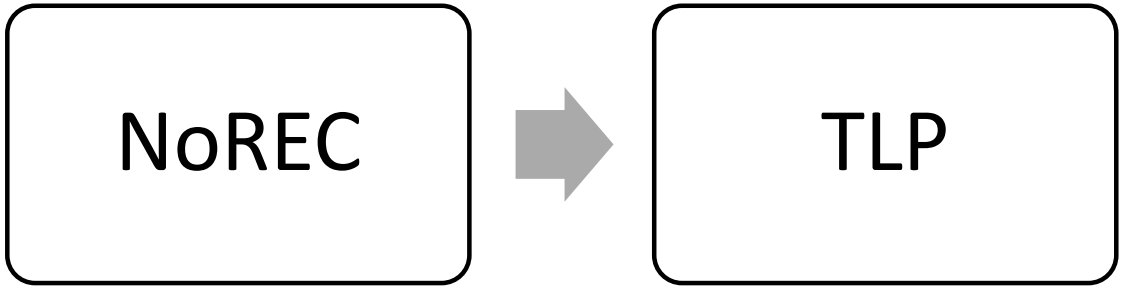
Proposed Testing Strategy

Quickly find the optimization bugs

NoREC

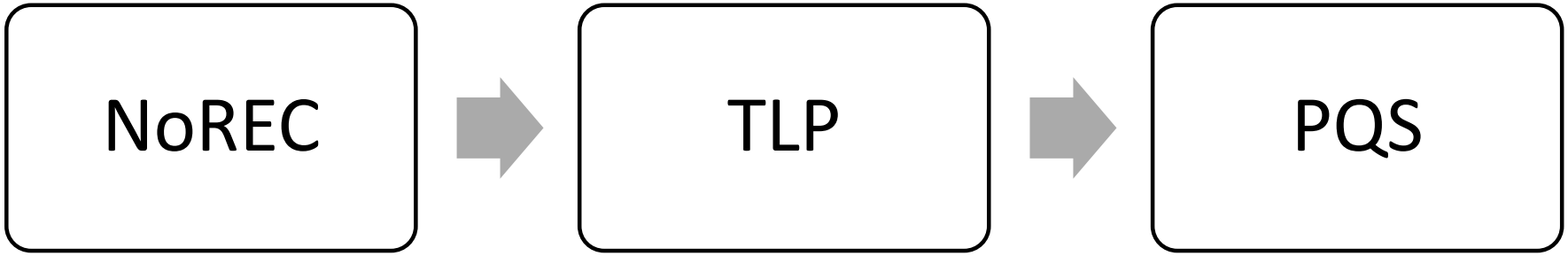
Proposed Testing Strategy

Test a wider range of features



Proposed Testing Strategy

Comprehensively test the DBMS' core functionality



What implementation strategy did you use for ClickHouse?

SQLancer from developer point of view

- Natural and intuitive approach
- Fuzzing meets correctness check
- Pluggable(but you need to write code on Java)
- You can incorporate approach in other software

How to make integration with your DBMS

1. Import Java or ODBC driver if you have one
2. Teach SQLancer to make connection, create database and make a generic query
3. A bit more to prepare: Create table with schema and insert generator
4. Implement expression generator
5. Implement oracle
6. Expected error handling
7. RUN!

```
254         <dependency>
255           <groupId>org.slf4j</groupId>
256           <artifactId> slf4j-simple</artifactId>
257           <version>1.7.30</version>
258         </dependency>
259         <dependency>
260           <groupId>ru.yandex.clickhouse</groupId>
261           <artifactId>clickhouse-jdbc</artifactId>
262           <version>0.2.5</version>
263         </dependency>
264         <dependency>
265           <groupId>com.h2database</groupId>
266           <artifactId>h2</artifactId>
267           <version>1.4.200</version>
268         </dependency>
269       </dependencies>
```

Add your driver

```
@   public SQLConnection createDatabase(ClickHouseGlobalState globalState) throws SQLExce
        ClickHouseOptions clickHouseOptions = globalState.getDmbsSpecificOptions();
        globalState.setClickHouseOptions(clickHouseOptions);
        String url = "jdbc:clickhouse://localhost:8123/default";
        String databaseName = globalState.getDatabaseName();
        Connection con = DriverManager.getConnection(url, globalState.getOptions().getUse
                globalState.getOptions().getPassword());
        String dropDatabaseCommand = "DROP DATABASE IF EXISTS " + databaseName;
        globalState.getState().logStatement(dropDatabaseCommand);
        String createDatabaseCommand = "CREATE DATABASE IF NOT EXISTS " + databaseName;
        globalState.getState().logStatement(createDatabaseCommand);
        try (Statement s = con.createStatement()) {
            s.execute(dropDatabaseCommand);
            Thread.sleep( millis: 1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        try (Statement s = con.createStatement()) {
            s.execute(createDatabaseCommand);
            Thread.sleep( millis: 1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
```

Make connection, database and query

```
210         List<ClickHouseColumn> columns = new ArrayList<>();
211         try (Statement s = con.createStatement()) {
212             try (ResultSet rs = s.executeQuery( sql: "DESCRIBE " + tableName)) {
213                 while (rs.next()) {
214                     String columnName = rs.getString( columnLabel: "name");
215                     String dataType = rs.getString( columnLabel: "type");
216                     String defaultType = rs.getString( columnLabel: "default_type");
217                     boolean isAlias = "ALIAS".compareTo(defaultType) == 0;
218                     boolean isMaterialized = "MATERIALIZED".compareTo(defaultType) == 0;
219                     ClickHouseColumn c = new ClickHouseColumn(columnName, getColumnType(dataType)
220                             isMaterialized);
221                     columns.add(c);
222                 }
223             }
```

Create table with schema

Your DBMS can has specific SQL extensions.
Implement subset of them

```
@Override
protected ClickHouseExpression generateExpression(ClickHouseLancerDataType type, int de
    if (allowAggregateFunctions && Randomly.getBoolean()) {
        return generateAggregate();
    }
    if (depth >= globalState.getOptions().getMaxExpressionDepth() || Randomly.getBoolea
        return generateLeafNode(type);
    }
    Expression expr = Randomly.fromOptions(Expression.values());
    ClickHouseLancerDataType leftLeafType = ClickHouseLancerDataType.getRandom();
    ClickHouseLancerDataType rightLeafType = ClickHouseLancerDataType.getRandom();
    if (Randomly.getBoolean()) {
        rightLeafType = leftLeafType;
    }

    switch (expr) {
    case UNARY_PREFIX:
        return new ClickHouseUnaryPrefixOperation(generateExpression(leftLeafType, dept
                ClickHouseUnaryPrefixOperation.ClickHouseUnaryPrefixOperator.getRandom(
    case UNARY_POSTFIX:
        return new ClickHouseUnaryPostfixOperation(generateExpression(leftLeafType, dep
                ClickHouseUnaryPostfixOperation.ClickHouseUnaryPostfixOperator.getRando
```

Expression generator

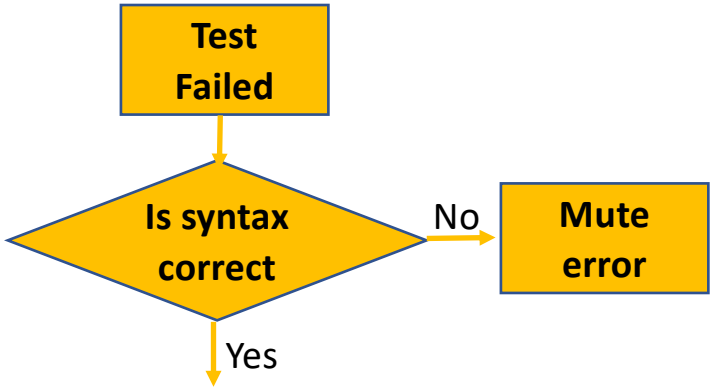
Implement Oracle

Almost the same for different DBMS

```
13    public class ClickHouseTLPWhereOracle extends ClickHouseTLPBase {
14
15        public ClickHouseTLPWhereOracle(ClickHouseProvider.ClickHouseGlobalState
20
21        @Override
22        public void check() throws SQLException {
23            super.check();
24            if (Randomly.getBooleanWithRatherLowProbability()) {
25                select.setOrderByExpressions(gen.generateOrderBys());
26            }
27            String originalQueryString = ClickHouseVisitor.asString(select);
28            List<String> resultSet = ComparatorHelper.getResultSetFirstColumnAsS
29
30            boolean orderBy = Randomly.getBooleanWithRatherLowProbability();
31            if (orderBy) {
32                select.setOrderByExpressions(gen.generateOrderBys());
33            }
34            select.setWhereClause(predicate);
35            String firstQueryString = ClickHouseVisitor.asString(select);
36            select.setWhereClause(negatedPredicate);
37            String secondQueryString = ClickHouseVisitor.asString(select);
38            select.setWhereClause(isNullPredicate);
39            String thirdQueryString = ClickHouseVisitor.asString(select);
40            List<String> combinedString = new ArrayList<>();
41            List<String> secondResultSet = ComparatorHelper.getCombinedResultSet
42                    thirdQueryString, combinedString, !orderBy, state, errors);
43            ComparatorHelper.assumeResultSetsAreEqual(resultSet, secondResultSet
44                    state);
45        }
```

How to deal with results?

- Query can be incorrect



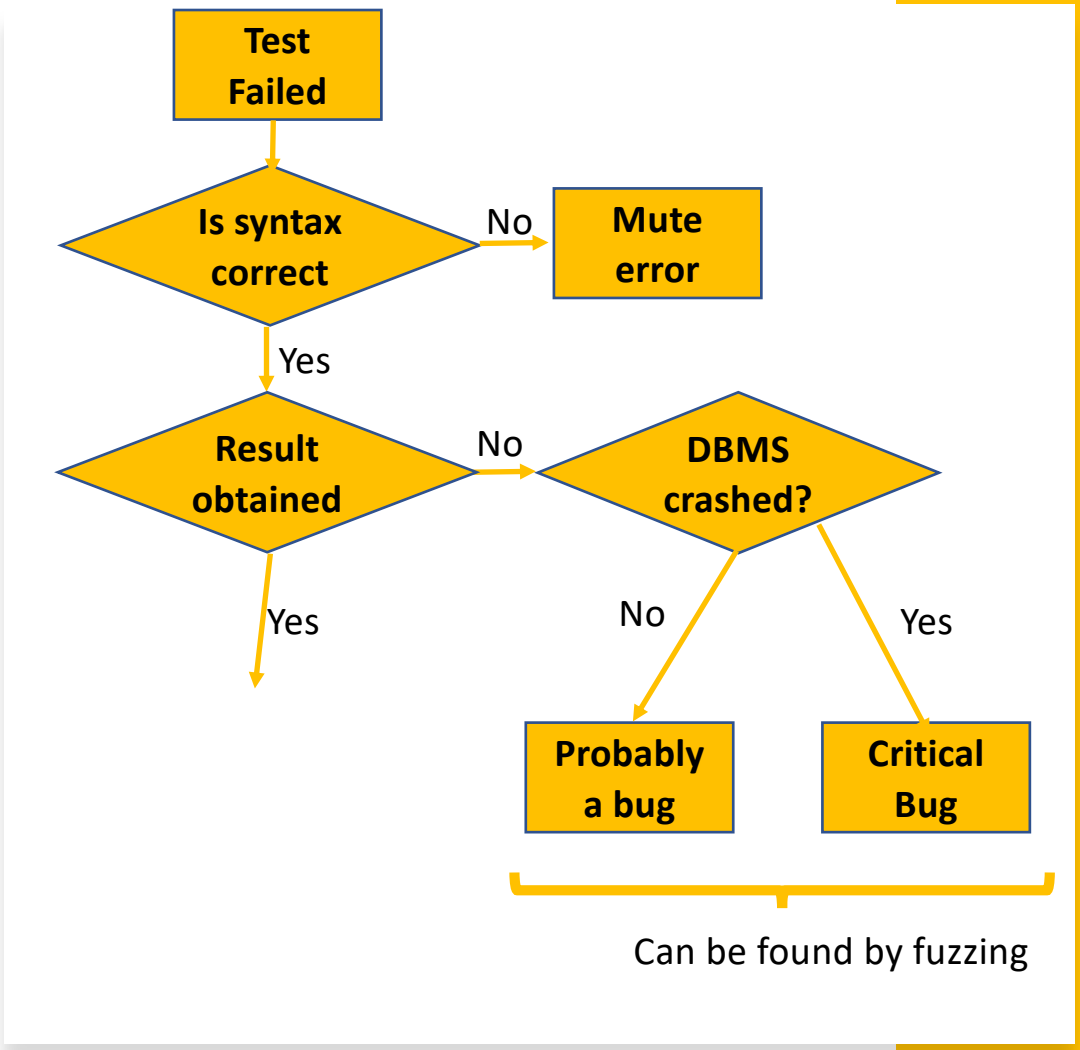
Mute errors

- You don't need to implement ideal query generator – bad queries also test your system
- You can mute known bugs and find more

```java
package sqlancer.clickhouse;

import sqlancer.common.query.ExpectedErrors;

public final class ClickHouseErrors {

    private ClickHouseErrors() {
    }

    public static void addExpectedExpressionErrors(ExpectedErrors errors) {
        errors.add("Illegal type");
        errors.add("Argument at index 1 for function like must be constant");
        errors.add("Argument at index 1 for function notLike must be constant");
        errors.add("does not return a value of type UInt8");
        errors.add("invalid escape sequence");
        errors.add("invalid character class range");
        errors.add("Memory limit");
        errors.add("There is no supertype for types");
        errors.add("Bad get: has Int64, requested UInt64");
        errors.add("Cannot convert string");
```

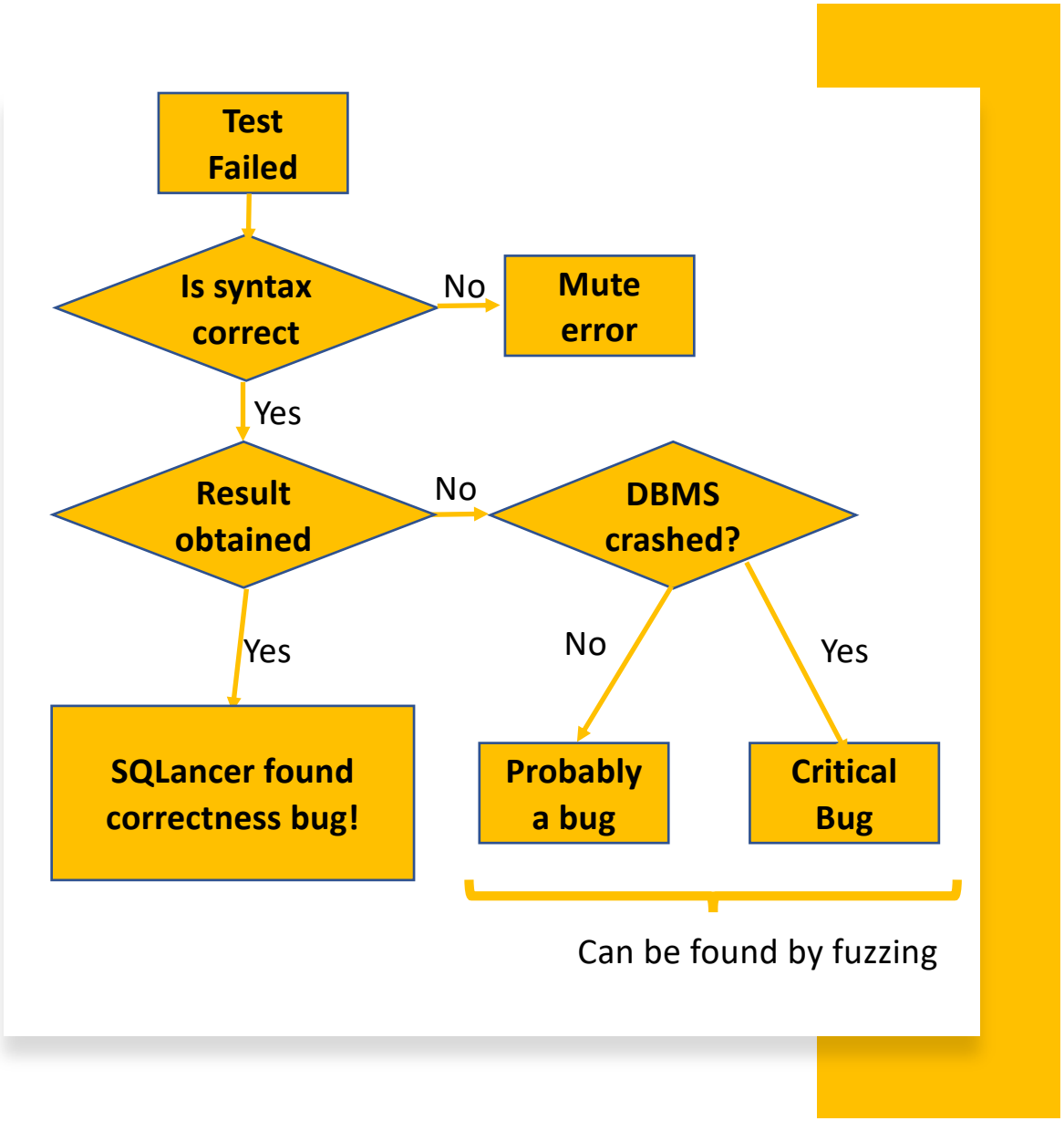
How to deal with results?

- Query can be incorrect
- DMBS can crash



How to deal with results?

- Query can be incorrect
- DMBS can crash
- If results differ – you found a correctness bug



Add Oracles to your Fuzzer

If you already have a query fuzzer or expression gererator you can implement oracles trivially.

You can try implementing NoREC or TLP Where logic by yourself:

1) Take ***expr*** from your generator

2) Add some logic to compare results:

```
SELECT * FROM t0
WHERE expr;
```

```
SELECT expr
FROM t0;
```

```
SELECT * FROM t;
```

```
SELECT * FROM t WHERE expr
UNION ALL
SELECT * FROM t WHERE NOT expr
UNION ALL
SELECT * FROM t WHERE expr IS NULL;
```

How to contribute?

- Add a new DBMS
- Automatic reduction of test cases (#333)
- Plugin system for DBMS support (#8)
- New test oracles
- Blog posts, tutorials, …

Summary

ClickHouse

Open Source analytical DBMS for BigData with SQL interface.

- Blazingly fast
- Scalable
- Fault tolerant

2013 Project started → 2016 Open Sourced → 2021 ★15K GitHub

2

SQLancer

Unwatch 23 | Star 678 | Fork 82

SQLANCER

https://github.com/sqlancer

SQLancer is an effective and widely-used **automatic testing tool** to find **logic bugs** in **DBMSs**

13

Finding Logic Bugs in DBMSs

Ternary Logic Partitioning
OOPSLA '20

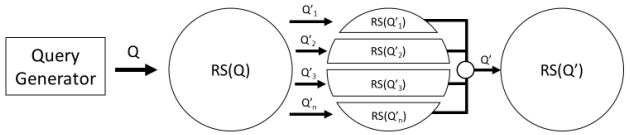
SQLancer provides three test oracles

Non-optimizing Reference Engine Construction
ESEC/FSE '20

Pivoted Query Synthesis
OSDI '20

64

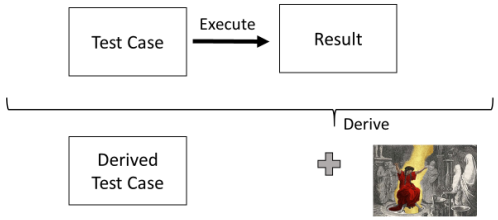
Query Partitioning



In contrast to differential testing, we need **only a single DBMS**

76

Metamorphic Testing



This technique is known as metamorphic testing

115

How to make integration with your DBMS

1. Import Java or ODBC driver if you have one
2. Teach SQLancer to make connection, create database and make a generic query
3. A bit more to prepare: Create table with schema and insert generator
4. Implement expression generator
5. Implement oracle
6. Expected error handling
7. RUN!

163

Q&A Session

- By the way, you can contribute to SQLancer and ClickHouse

https://github.com/sqlancer/sqlancer

https://github.com/ClickHouse/ClickHouse