CS6218. Principles of Programming Languages & Software Engineering

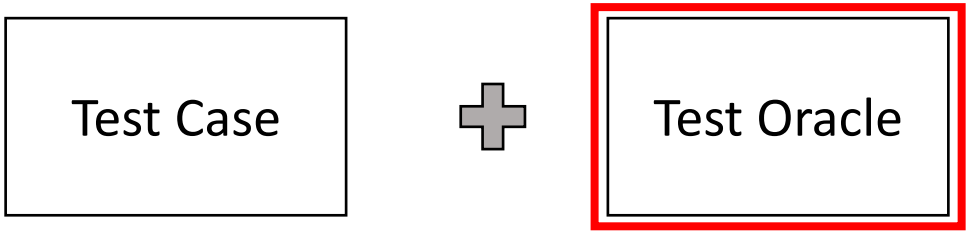
Week 3: SQLancer and test oracles for logic bugs in database systems

Last Lecture

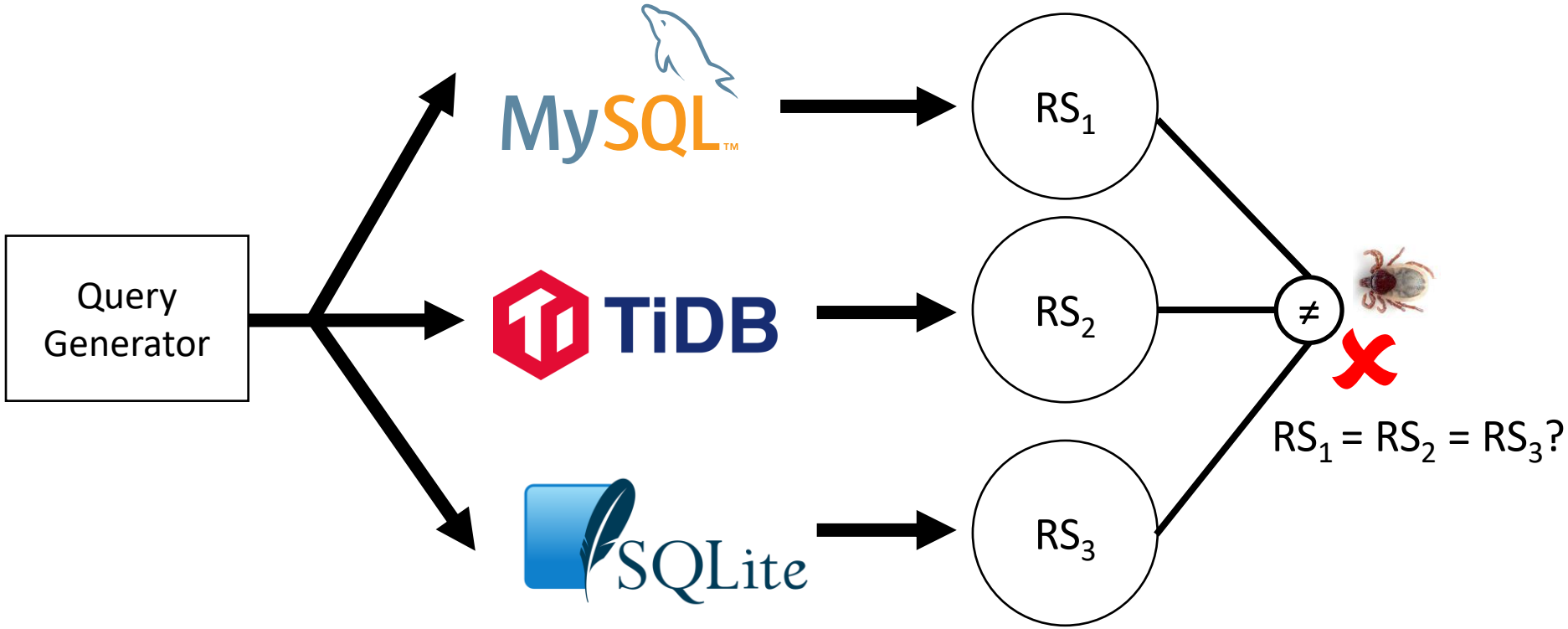
- Manual/unit testing, code coverage, mutation testing
- Test oracles: Differential testing, metamorphic testing, property-based testing, intramorphic testing
- Test case generation: generational vs. mutational, automated testing vs fuzzing
- Test case reduction and deduplication

Last Lecture: Automated Testing

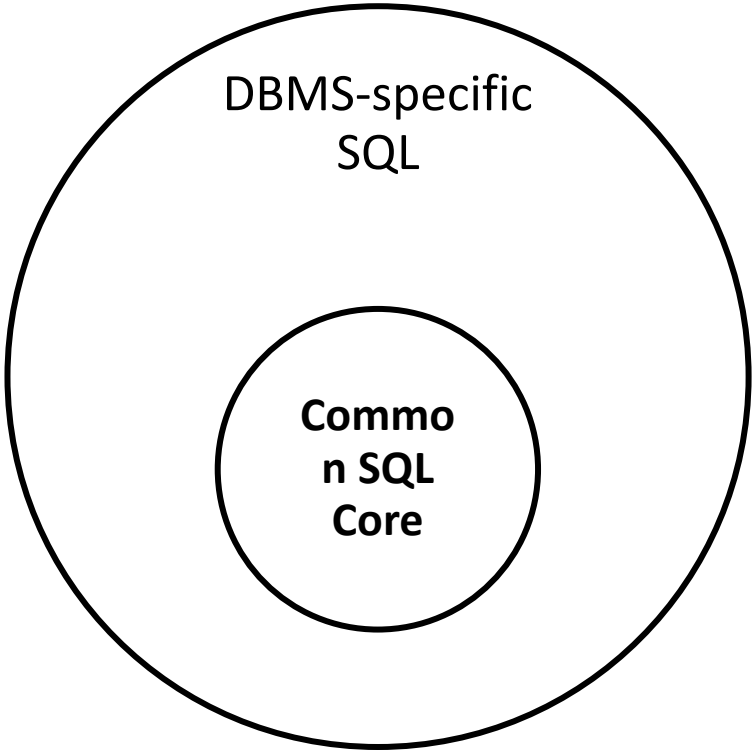
- Automation of test case generation and the test oracle



Last Lecture: Differential Testing

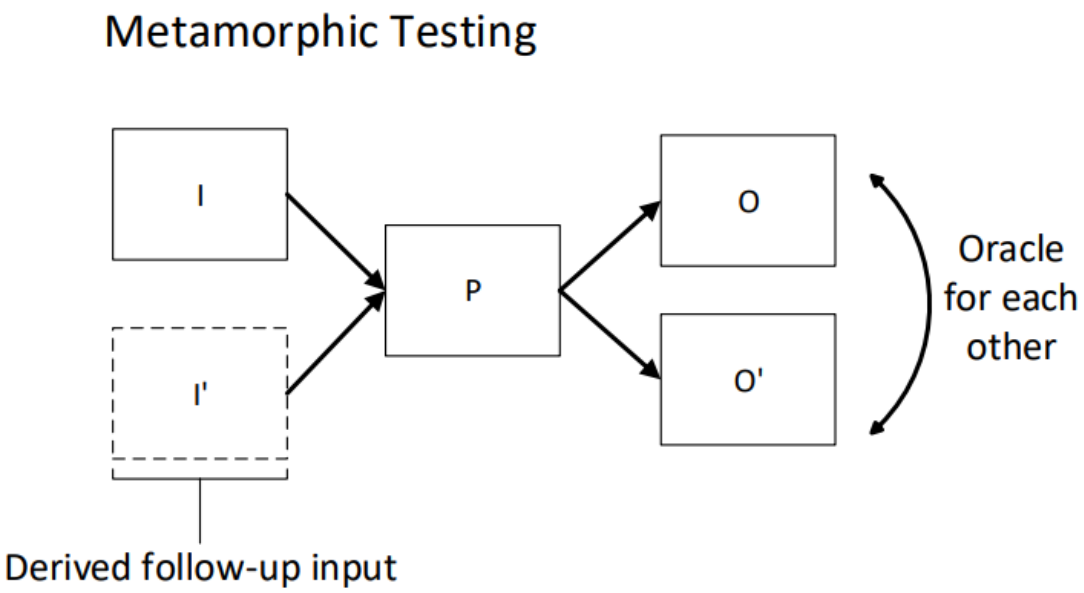


Last Lecture: Differential Testing



"[…] proved to be extremely useful, but only for the **small set of common SQL**"

Last Lecture: Metamorphic Testing



This Lecture

- State-of-the-art techniques implemented in SQLancer
- Focus: test oracles for finding logic/correctness bugs in database systems
  - Can we do better than differential testing?

SQLancer

968 stars 148 forks

https://github.com/sqlancer

@sqlancer_dbms

SQLancer

```
$ 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

Adoption

DuckDB

ClickHouse

monetdb

TiDB

citusdata

Adoption

Watch 70 | Unstar 2.5k | Fork 161

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 + TLP on every pull request

Adoption

Watch 605 | Star 14.5k | Fork 2.9k

ClickHouse

Yandex contributed a TLP implementation to SQLancer

Adoption

ClickHouse Sqlancer Test for master

test_run.txt.out.log | master | Commit | TLPWhere.err | TLPWhere.out | TLPGroupBy.err | TLPGroupBy.out | TLPHaving.err | TLPHaving.out | TLPWhereGroupBy.err | TLPWhereGroupBy.out | TLPDistinct.err | TLPDistinct.out | TLPAggregate.err | TLPAggregate.out | logs.tar.gz | stdout.log | stderr.log | Help | Task (private network)

Test name	Test status	Test time, sec.
TLPWhere	OK	
TLPGroupBy	OK	
TLPHaving	OK	
TLPWhereGroupBy	OK	
TLPDistinct	OK	
TLPAggregate	OK	

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

chaos-mesh / go-sqlancer

Unwatch 7 | Unstar 12 | Fork 2

Code | Issues 9 | Pull requests 2 | Actions | Projects 2 | Wiki | Security 0 | Insights

go-sqlancer

fuzzing tidb

58 commits | 4 branches | 0 packages | 0 releases | 4 contributors

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

mahjonp rename flags and update readme (#50) ... Latest commit 3c755aa yesterday

cmd	rename flags and update readme (#50)	yesterday
pkg	rename flags and update readme (#50)	yesterday
.gitignore	remove sqlsmith	2 months ago
Makefile	add transformer and support TLP (#48)	yesterday
README.md	rename flags and update readme (#50)	yesterday
go.mod	add transformer and support TLP (#48)	yesterday
go.sum	add transformer and support TLP (#48)	yesterday

ClickHouse

Watch 1.4k | Unstar 26.8k | Fork 4.2k

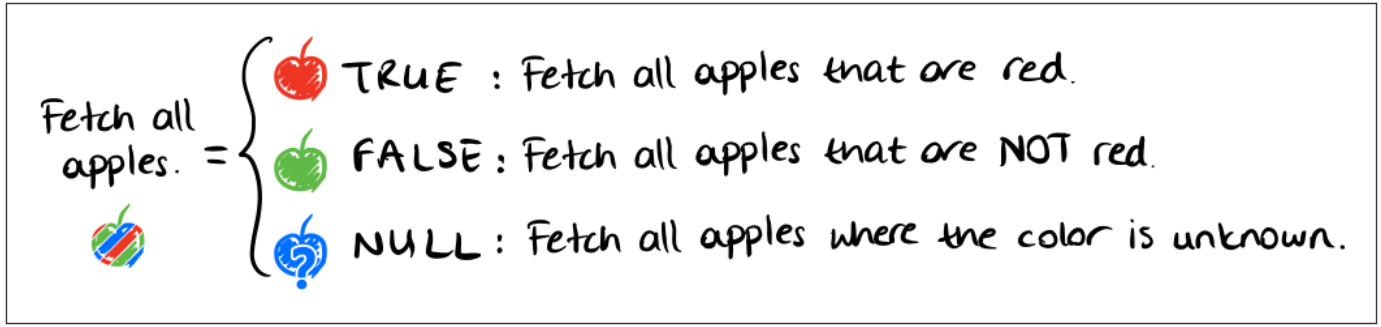
TiDB

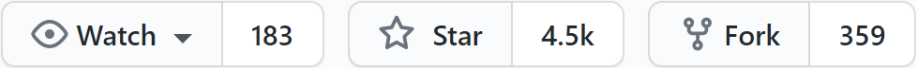
PingCAP implemented TLP as a tool go-sqlancer

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

Adoption

Mining for logic bugs in the Citus extension to Postgres with SQLancer by Nazli Ugur Koyluoglu



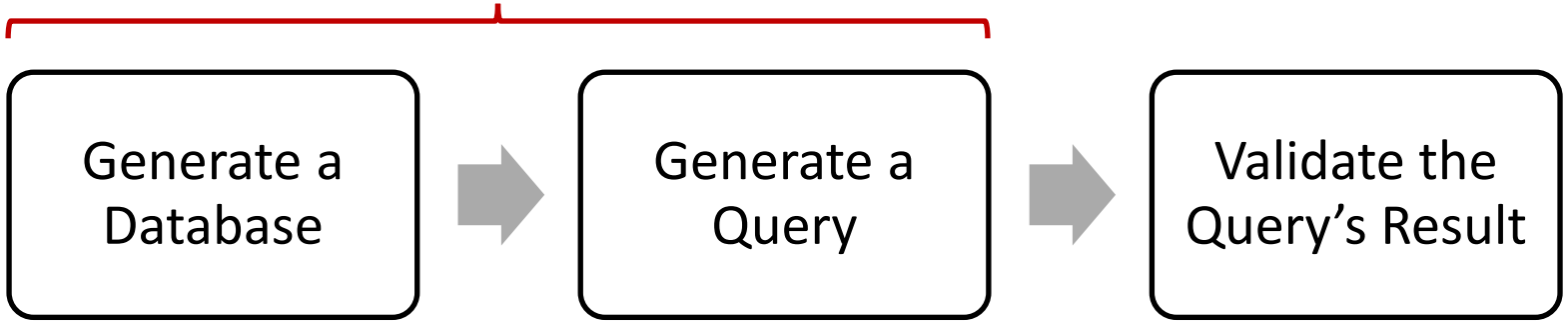


citusdata

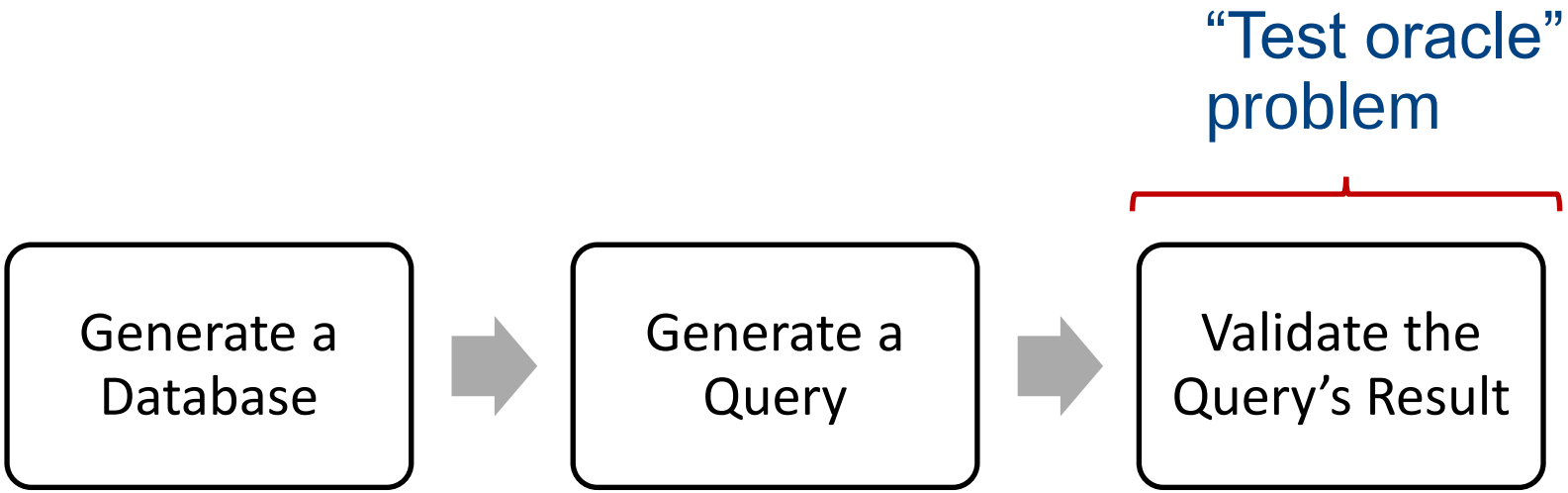
https://www.citusdata.com/blog/2020/09/04/mining-for-logic-bugs-in-citus-with-sqlancer/

Automatic Testing Core Challenges

SQLancer uses a random-generation approach to automatically generate tests



Automatic Testing Core Challenges



Finding Logic Bugs in DBMSs

Ternary Logic Partitioning

OOPSLA ‘20

Non-optimizing Reference Engine Construction

ESEC/FSE ‘20

Pivoted Query Synthesis

OSDI ‘20

Finding Logic Bugs in DBMSs

Ternary Logic Partitioning

OOPSLA '20

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

Non-optimizing Reference Engine Construction

ESEC/FSE '20

Pivoted Query Synthesis

OSDI '20

Finding Logic Bugs in DBMSs

Ternary Logic Partitioning

OOPSLA ‘20

Partition a query’s result set

Non-optimizing Reference Engine Construction

ESEC/FSE ‘20

Pivoted Query Synthesis

OSDI ‘20

Finding Logic Bugs in DBMSs

Ternary Logic Partitioning

OOPSLA '20

Generate a query for which it is **ensured** that a randomly-selected row, the **pivot row**, **is fetched**

Non-optimizing Reference Engine Construction

ESEC/FSE '20

Pivoted Query Synthesis

OSDI '20

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 Optimization 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 '-*';
```

SQLite

-1

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 '-*';
```

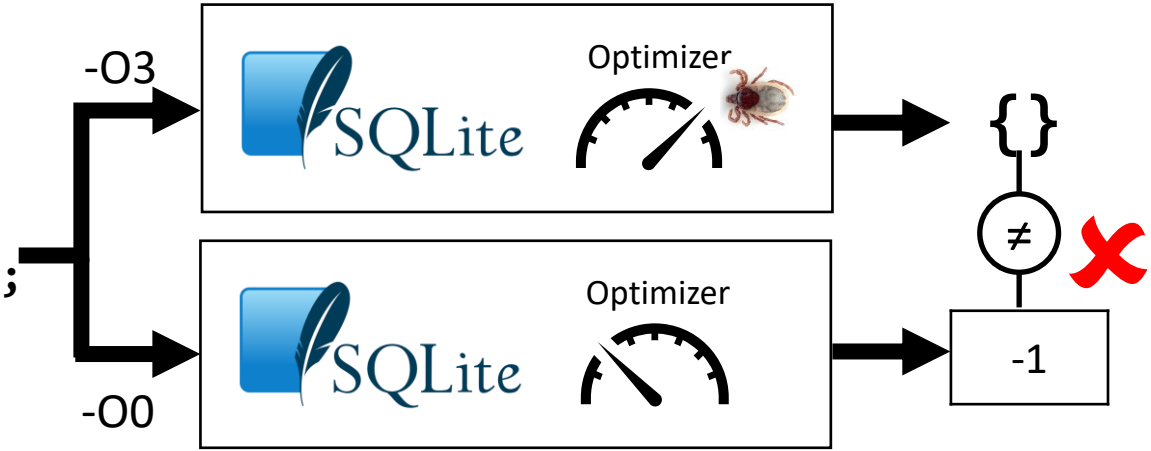
The *LIKE optimization* malfunctioned for non-text columns and a pattern prefix of “-”

{}

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
check
collati
compile_options
~~count_changes[1]~~
~~data_store_direct~~
data_version
database_list
~~default_cache_siz~~
defer_foreign_key
~~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
legacy_alter_table
legacy_file_format
page_count
page_size
parser_trace[2]
pragma_list
query_only
quick_check
read_uncommitted

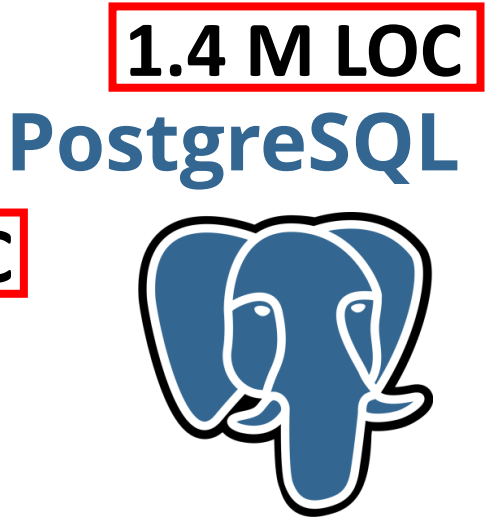
recursive_triggers
reverse_unordered_selects
schema_version[3]
secure_delete
~~short_column_names[1]~~
shrink_memory
soft_heap_limit
stats[3]
synchronous
temp_store
~~temp_store_directory[1]~~
vdbe_listing[2]
vdbe_trace[2]
wal_autocheckpoint
wal_checkpoint
writable_schema[3]

PRAGMA case_sensitive_like = *boolean*;

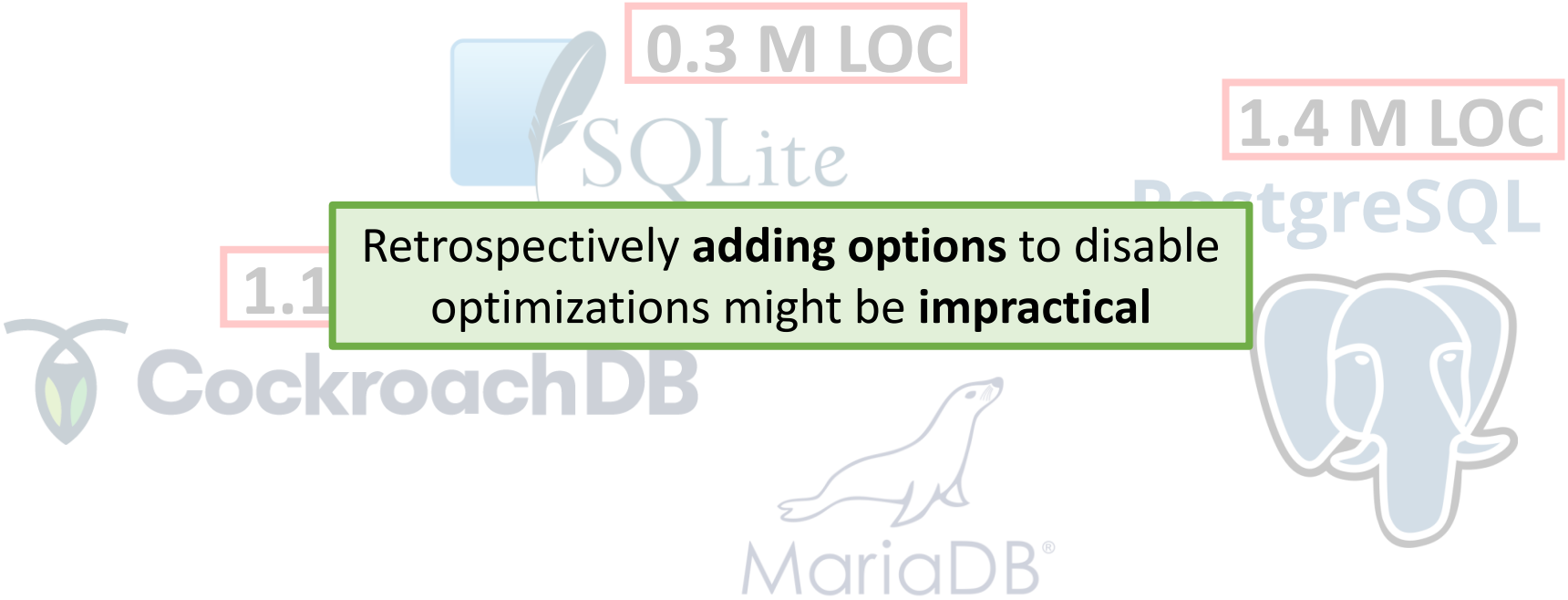
DBMSs typically provide only very **limited control** over optimizations

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

Background: Lines of Code (LOC)



Background: Lines of Code (LOC)

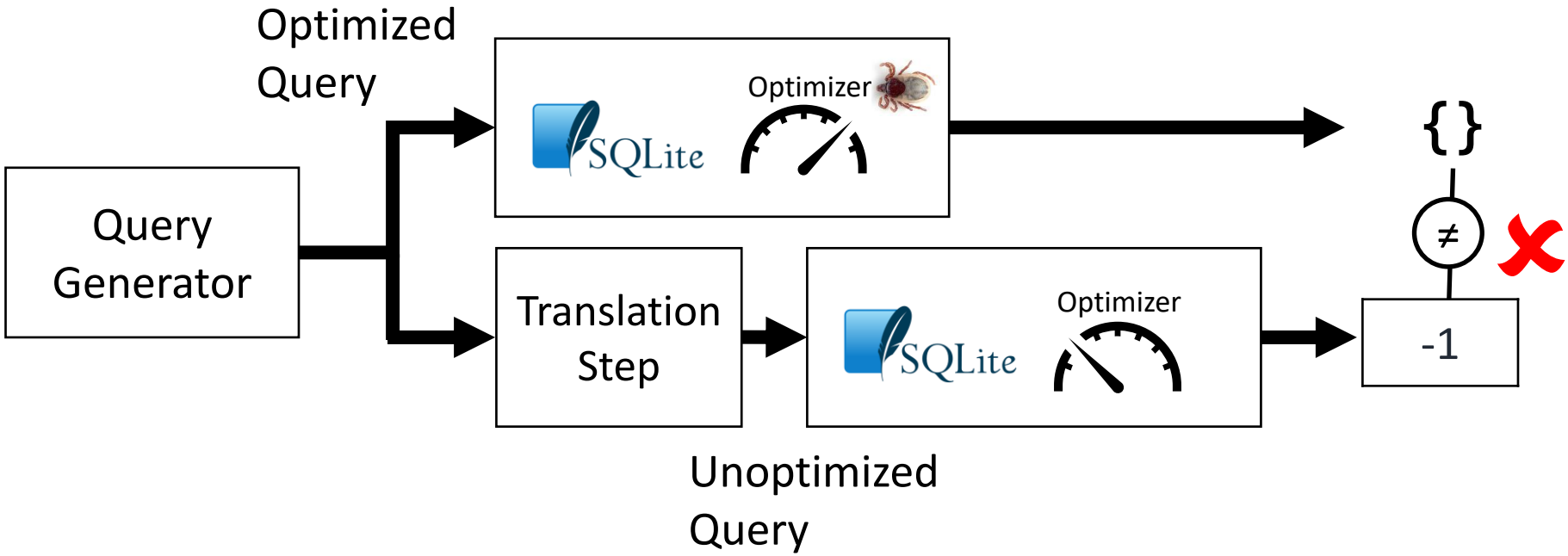


Retrospectively **adding options** to disable optimizations might be **impractical**

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 '-*'
```

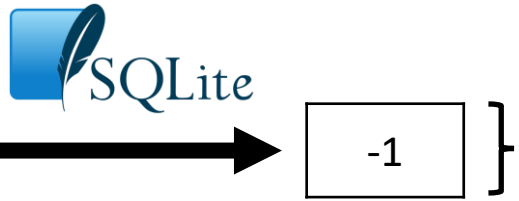
It is unobvious how we could derive an unoptimized query

Insight

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

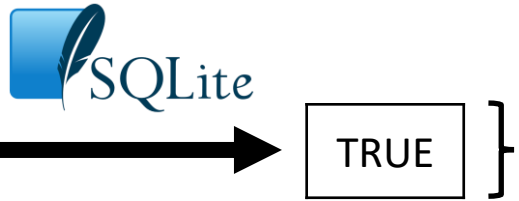
Translation Step (Correct Case)

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



p evaluates to **TRUE** for one row

```
SELECT p
FROM t0;
```



p evaluates to **TRUE** for one row

Insights

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

Translation Step

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

Optimizer

```
QUERY PLAN
`--SEARCH TABLE t0 USING COVERING INDEX
sqlite_autoindex_t0_1 (c0>? AND c0<?)
```

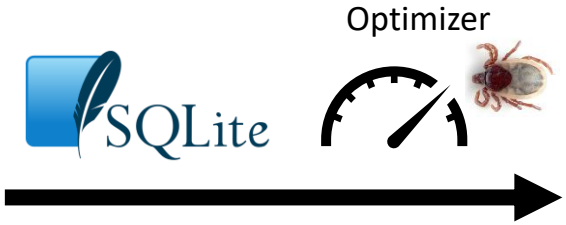
```
SELECT p
FROM t0;
```

Optimizer

```
QUERY PLAN
`--SCAN TABLE t0
```

Translation Step

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



{}

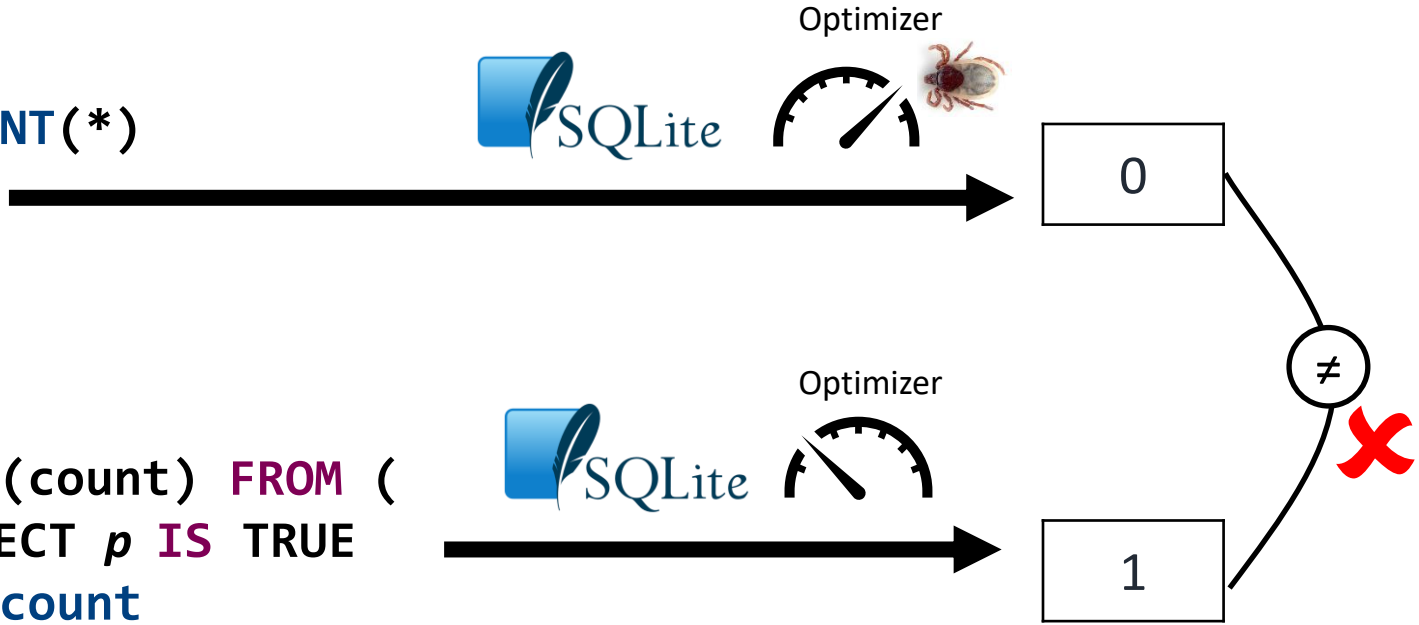
```
SELECT p
FROM t0;
```

TRUE

Result should contain one row

Counting Implementation

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




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

Evaluation: Found Bugs

			Closed	
DBMS	Fixed	Verified	Intended	Duplicate
SQLite	110	0	6	0
MariaDB	1	5	0	1
PostgreSQL	5	2	1	0
CockroachDB	28	7	0	1

We found **159 bugs**, 141 of which have been fixed!

Evaluation: Found Bugs

			Closed	
DBMS	Fixed	Verified	Intended	Duplicate
SQLite	110	0	6	0
MariaDB				1
PostgreSQL	5	2	1	0
CockroachDB	28	7	0	1

We **concentrated** on testing **SQLite**

Evaluation: Test Oracles

			Crash	
DBMS	Logic	Error	Release	Debug
SQLite	39	30	15	26
MariaDB	5	0	1	0
PostgreSQL	0	4	2	1
CockroachDB	7	24	4	0

Evaluation: Test Oracles

			Crash	
DBMS	Logic	Error	Release	Debug
SQLite	39	30	15	26
MariaDB	5	0	1	0
PostgreSQL	0	4	2	1
CockroachDB	7	24	4	0

We found **51** optimization bugs!

Thinking About Research

- How would you rate this approach?
  - Simple vs complex
  - Ad-hoc vs systematic
  - General vs. specific

How can we evaluate research?

ICSE Reviewer Guidelines

- How can we evaluate research (in software engineering)?
- Five main criteria
  - Soundness
  - Significance
  - Novelty
  - Verifiability and Transparency

https://conf.researchr.org/getImage/icse-2023/orig/ICSE+2023+Review+Process+and+Guidelines.pdf

ICSE Reviewer Guidelines

Review — *Soundness*

The extent to which the paper's contributions and/or innovations address its research questions and are supported by rigorous application of appropriate research methods

The paper should answer the questions it poses, and it should do so with rigor in its research methodology (including choosing an appropriate research methodology and procedures). This is an important difference between research papers and other kinds of knowledge sharing (e.g., experience reports), and the source of certainty researchers can offer.

https://conf.researchr.org/getImage/icse-2023/orig/ICSE+2023+Review+Process+and+Guidelines.pdf

ICSE Reviewer Guidelines

Review — *Significance*

The extent to which the paper's contributions beyond prior work in terms of implications for software engineering research and practice, and if needed, under which assumptions

In all generality, impact relates to advances in the practice of software engineering (including making software less costly, more maintainable, more reliable, more reusable, safer, more secure, more usable … – this is not an exhaustive list)

Note that it is the **authors' responsibility** to explain and interpret the significance of their contributions, why they matter, what their potential implications will be, and under which assumptions.

https://conf.researchr.org/getImage/icse-2023/orig/ICSE+2023+Review+Process+and+Guidelines.pdf

ICSE Reviewer Guidelines

Review — *Novelty*

The extent to which the contributions are sufficiently original with respect to the state-of-the-art

Grounded in adequate review of prior work in a respective topic, it is up to the authors to convince you that the discoveries advance our knowledge in some way, whether it sheds more light on prior work, or adds a significant new idea.

Secondarily, there should be **someone who might find the discovery interesting**. It does not have to be interesting to you, and you do not have to be 100% confident that an audience exists. **A possible audience** is sufficient for publication, as the PC does not necessarily perfectly reflect the broader audience of readers.

https://conf.researchr.org/getImage/icse-2023/orig/ICSE+2023+Review+Process+and+Guidelines.pdf

ICSE Reviewer Guidelines

Review — *Verifiability and Transparency*

The extent to which the paper includes sufficient information to understand how an innovation works; to understand how data was obtained, analyzed, and interpreted; and how the paper supports independent verification or replication of the paper's claimed contributions.

This aims to check whether the described research is *recoverable*. You should be able to understand most of the key details about how the authors conducted their work, how they made their invention possible, or how the research findings were inferred from the collected evidence. This is key for replication and meta-analysis of studies underpinned by the positivist or post-positivist approaches. For interpretivist works, it is also key for evaluating qualitative work. Focus your critiques on omissions of research process or innovation details that would significantly alter your judgement of the paper's validity, or the *credibility* of results for research that uses qualitative methods.

https://conf.researchr.org/getImage/icse-2023/orig/ICSE+2023+Review+Process+and+Guidelines.pdf

ICSE Reviewer Guidelines

Review — *Presentation*

The extent to which the paper's quality of writing meets the high standards of ICSE, including clear descriptions, as well as adequate use of the English language, absence of major ambiguity, clearly readable figures and tables, and adherence to the formatting instructions provided below.

Papers also need to be clear and concise, and comprehensible to diverse audiences.

We recognize that not all authors are fluent English writers. But if the language issues make the paper not comprehensible, it is not yet ready for publication.

https://conf.researchr.org/getImage/icse-2023/orig/ICSE+2023+Review+Process+and+Guidelines.pdf

Novelty

- No silver bullet to evaluating research

The Toxic Culture of Rejection in Computer Science

Post by: Edward Lee

22 Aug 2022 | 0

https://sigbed.org/2022/08/22/the-toxic-culture-of-rejection-in-computer-science/

Finding Logic Bugs in DBMSs

Ternary Logic Partitioning

OOPSLA '20

Conceptually new approach that addresses NoREC's **limitation of being limited to WHERE clauses**

Non-optimizing Reference Engine Construction

ESEC/FSE '20

Pivoted Query Synthesis

OSDI '20

Motivating Example

t0

c0
0.0

t1

c0
-0.0

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

MySQL → ?

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

Motivating Example

t0

c0
0.0

t1

c0
-0.0

false?

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

MySQL

t0.c0	t1.c0

0	**0**	0	0	0	...	0	0	0	0
-0	**1**	0	0	0	...	0	0	0	0

Different binary representation

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;
```

MySQL

t0.c0	t1.c0
0	-0

Motivating Example

t0

c0
0.0

t1

c0
-0.0

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

MySQL

t0.c0	t1.c0
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;
```

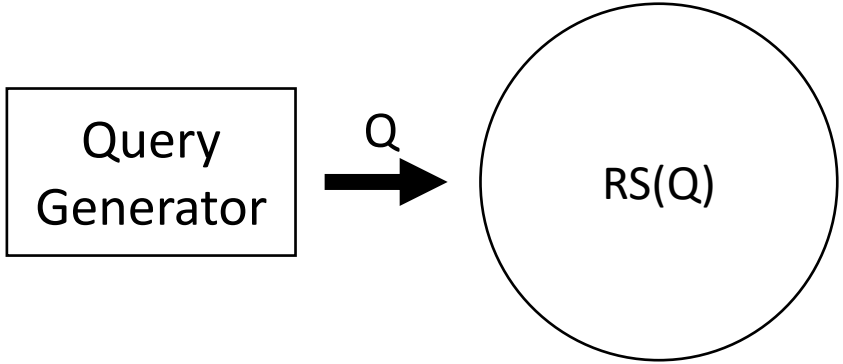
t0.c0	t1.c0

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

Query Partitioning

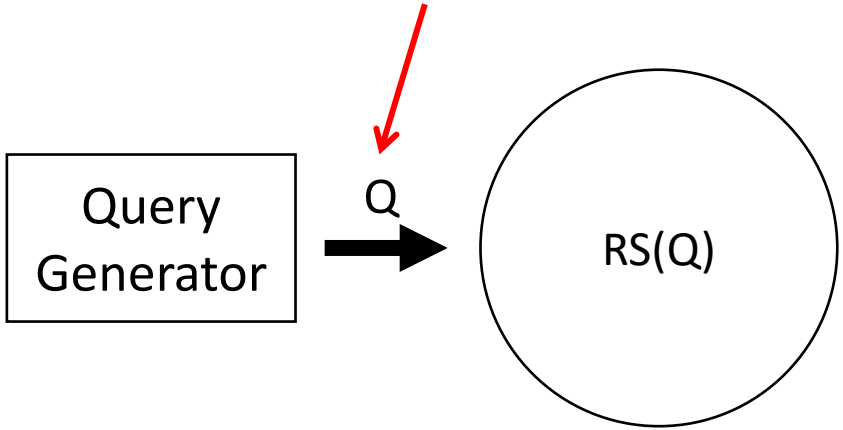
Ternary Logic Partitioning (TLP) is based on a conceptual framework called **Query Partitioning**

Query Partitioning

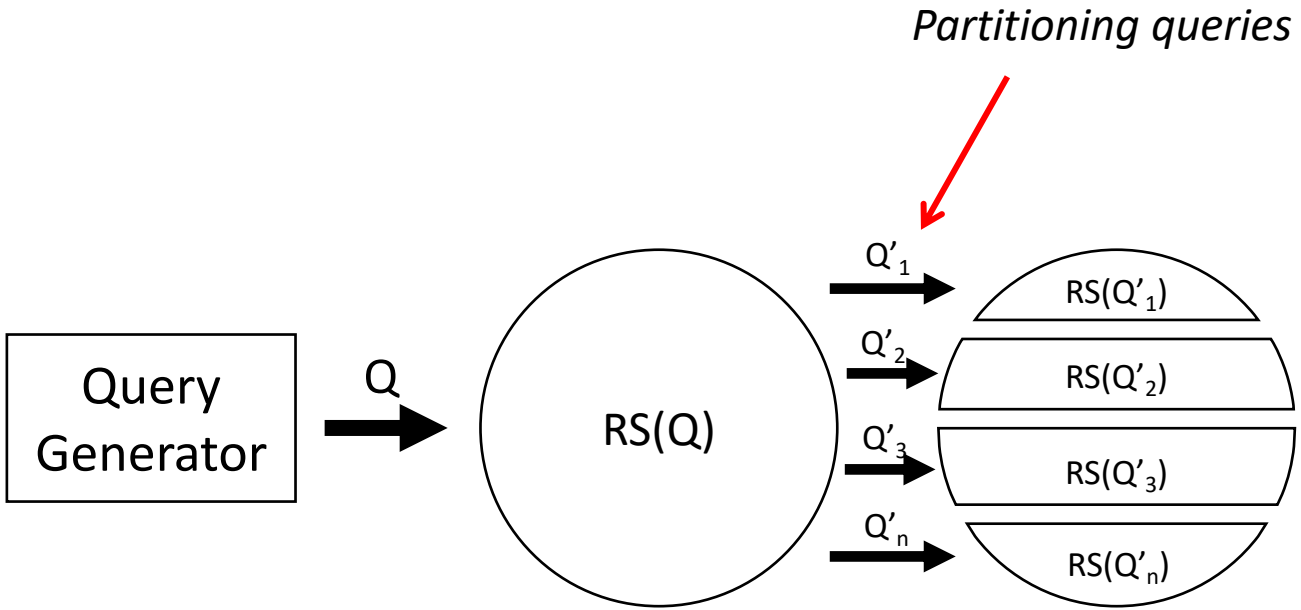


Query Partitioning

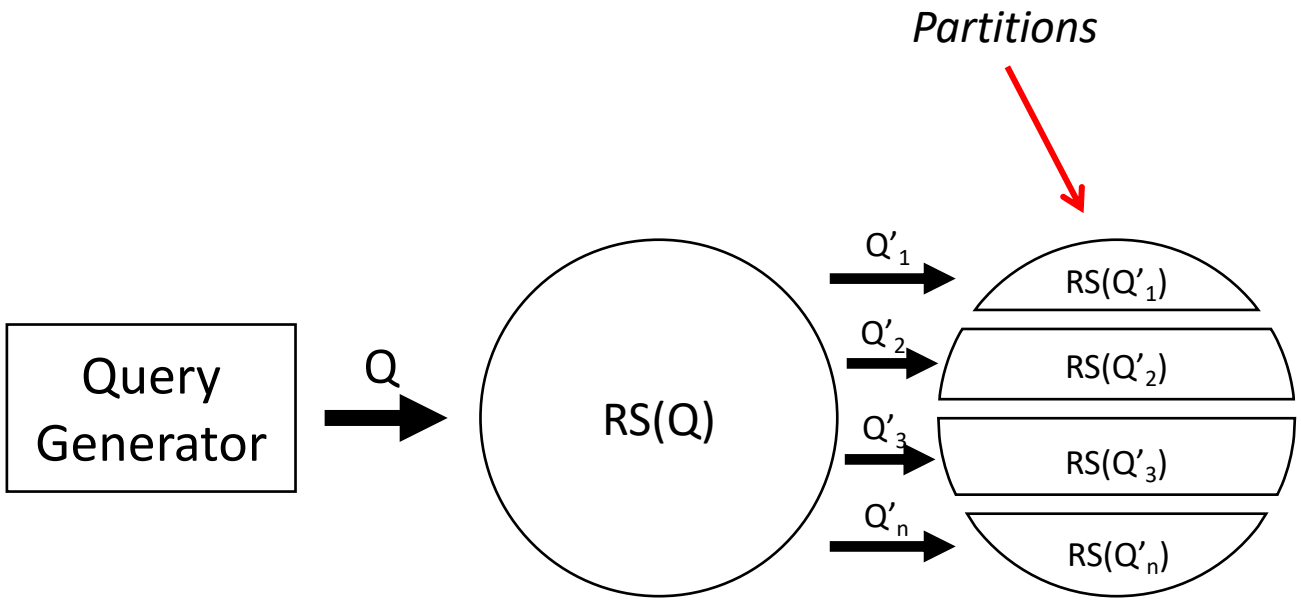
Q denotes the (randomly-generated) *original query*



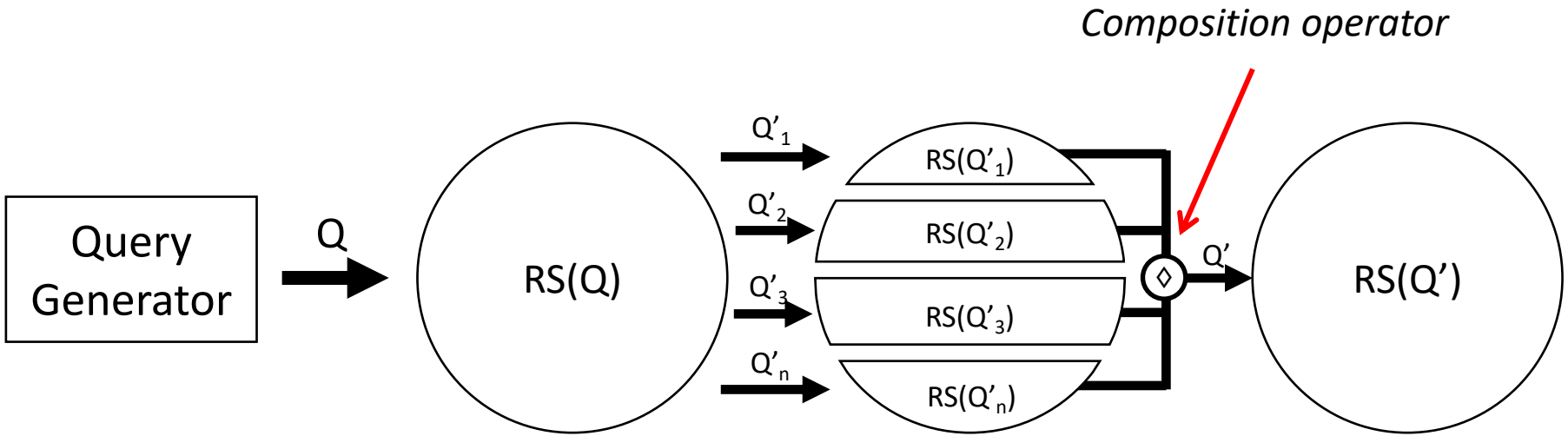
Query Partitioning



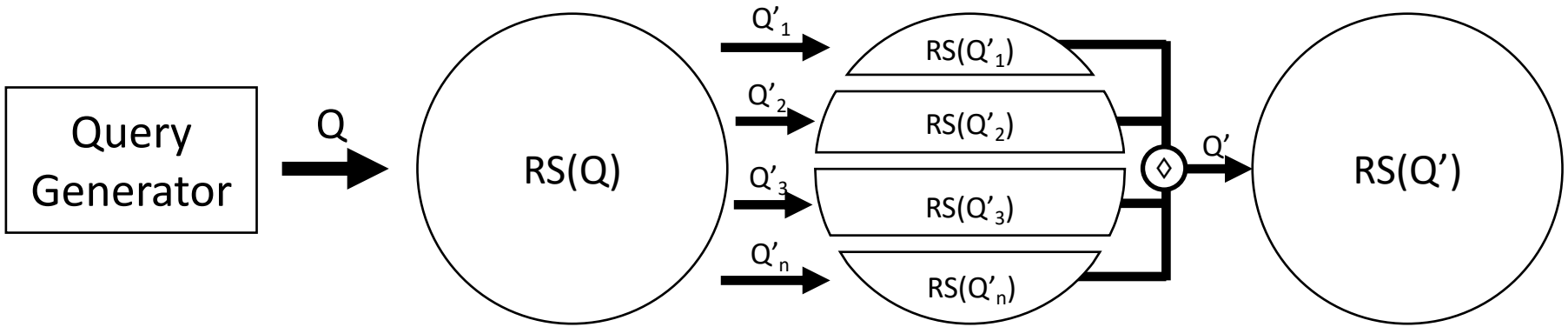
Query Partitioning



Query Partitioning

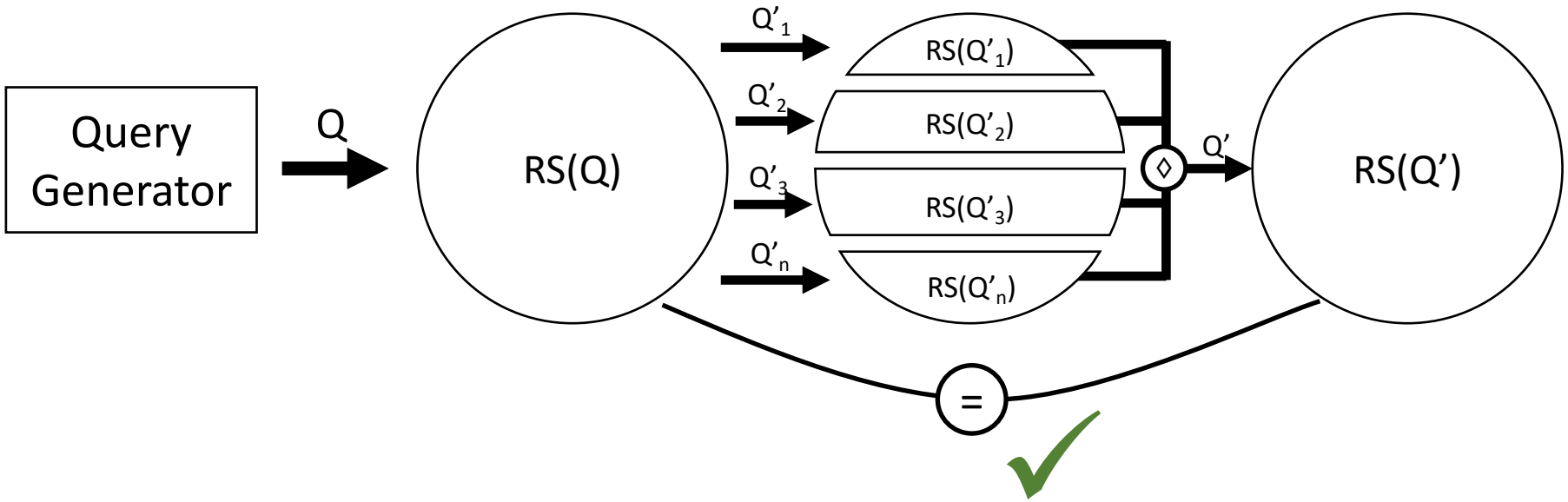


Query Partitioning

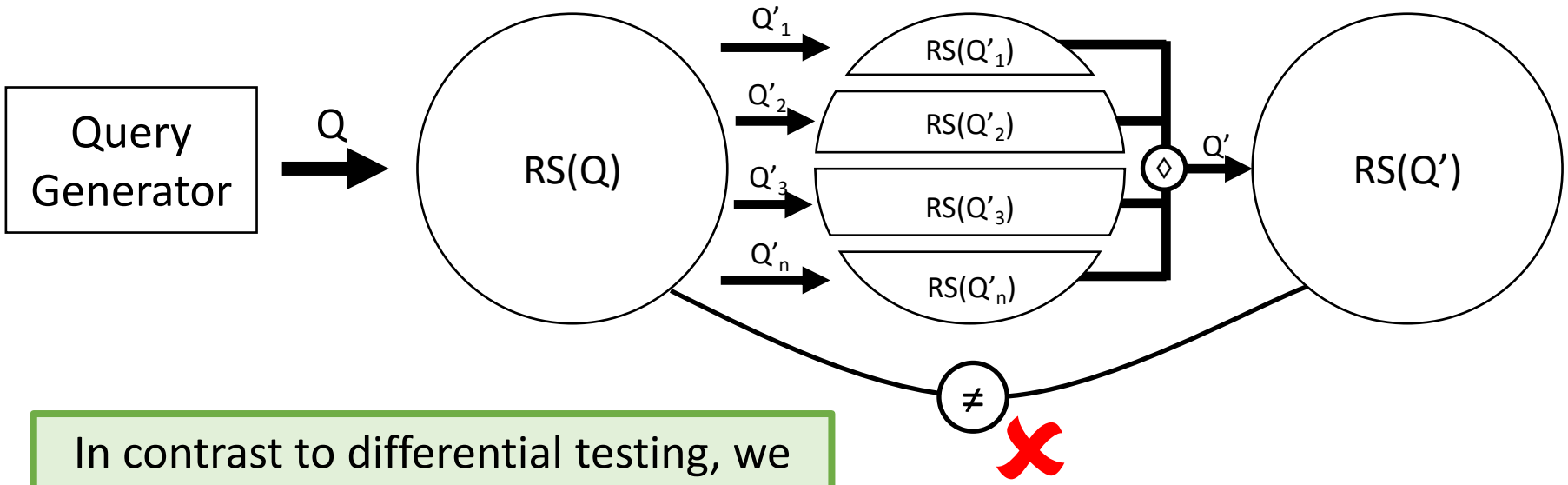


Combine the results so that RS(Q)=RS(Q')

Query Partitioning



Query Partitioning



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

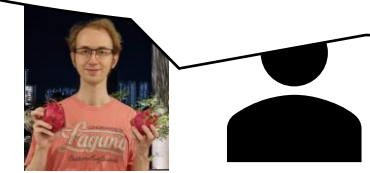
How to Realize This Idea?

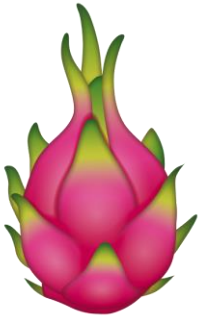
Key challenge: find a **valid partitioning strategy** that **stresses the DBMS**

Scenario: Dragon Fruits

White vs. Red Dragon Fruits

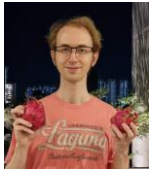
Red dragon fruits are tastier than white ones, but it is impossible to tell them apart

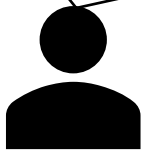




White vs. Red Dragon Fruits

Easy Lah! Just browse the web





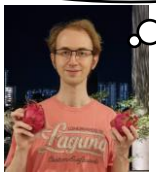
In Summary: Comparison Table

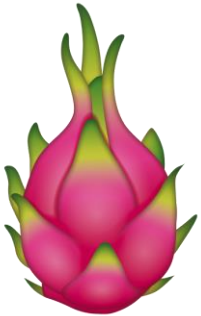
FEATURE	RED	WHITE
Scales	Narrower, more	Wider, fewer
Skin	Deep dark red	Bright pinkish
Flower margins	Red-purple	Green-yellow
Branches	Wavy, spiky	Milder, less spiky
Anti-oxidants	More	Less
Sugar content	Usually more	Less

https://zenyrgarden.com/what-is-the-difference-between-red-dragon-fruit-and-white-dragon-fruit/

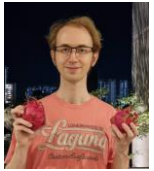
White vs. Red Dragon Fruits

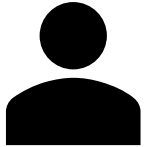
How can I test whether you can really tell the difference?

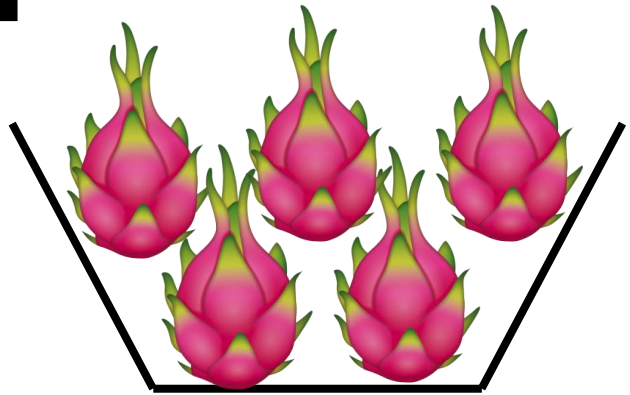




Tangerines vs. Clementines

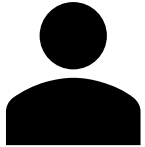


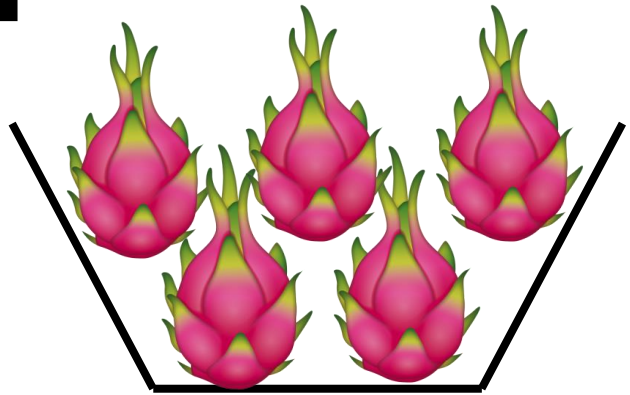




Tangerines vs. Clementines

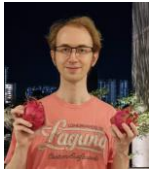
Bring me all red ones!

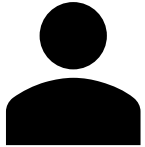


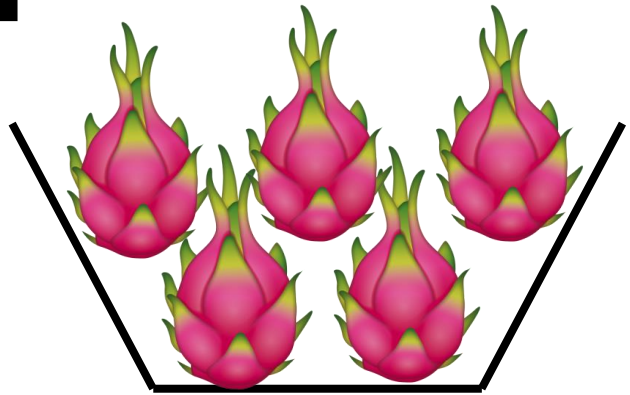


2 fruits

Tangerines vs. Clementines

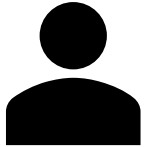




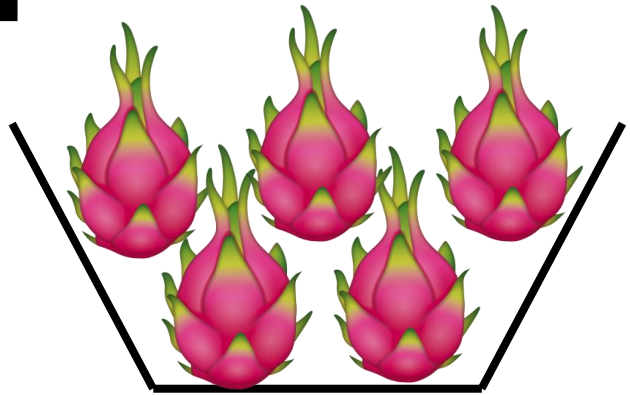


Tangerines vs. Clementines

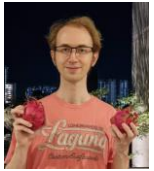
Bring me all **but** the red ones!

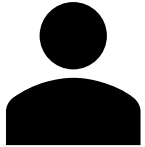


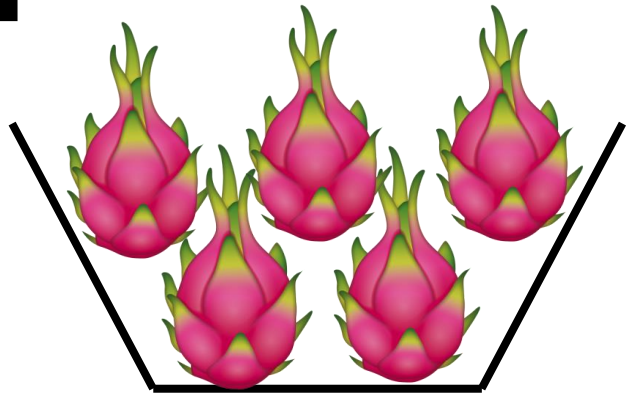
4 fruits



Tangerines vs. Clementines







2 fruits

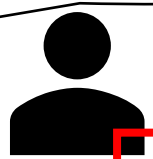
4 fruits

---

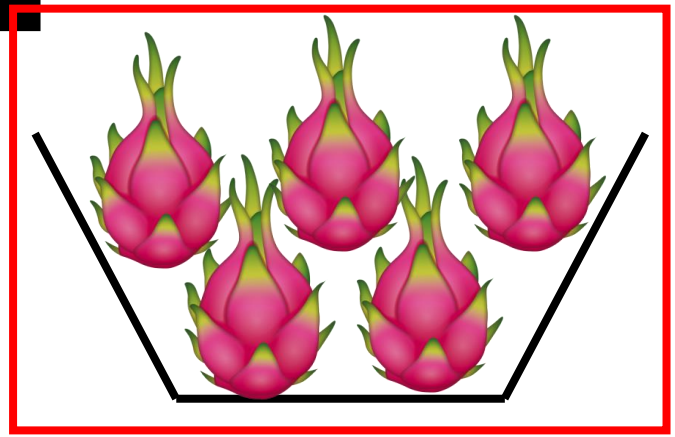
6 fruits

Tangerines vs. Clementines

You likely classified a dragon fruit as **both** a red and white!



5 fruits



2 fruits
4 fruits
6 fruits

Insight

Insight: Every object in a (mathematical) universe is either **a red dragon fruit** or **not**

Ternary Logic

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

- φ
- NOT φ
- φ IS NULL

Ternary Logic

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

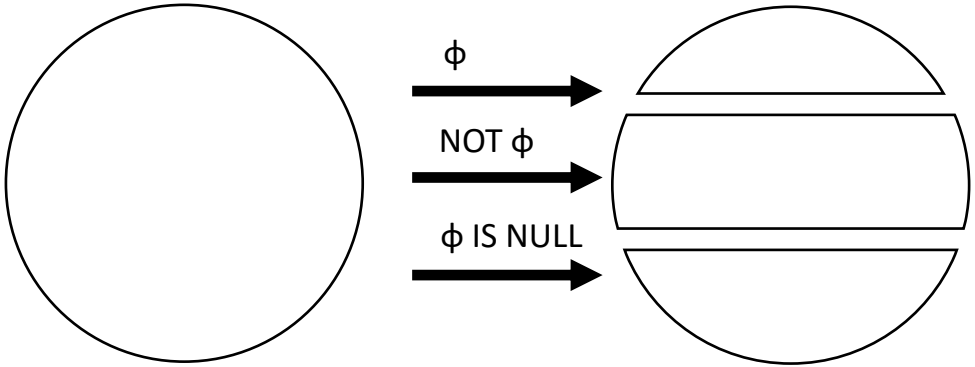
- φ
- NOT φ
- φ IS NULL

ternary predicate variants

Ternary Logic

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

- φ
- NOT φ
- φ IS NULL



Motivating Example

t0

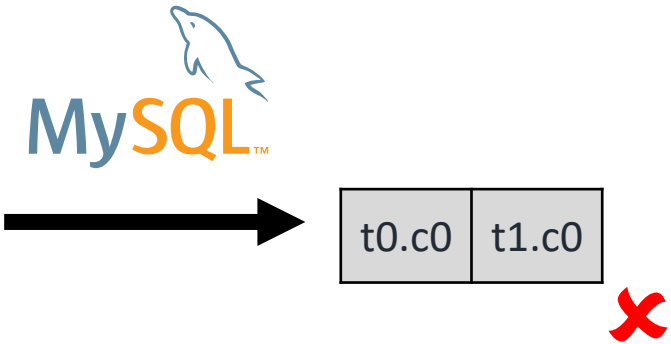
c0
0.0

t1

c0
-0.0

How did this insight allow us to detect this bug?

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

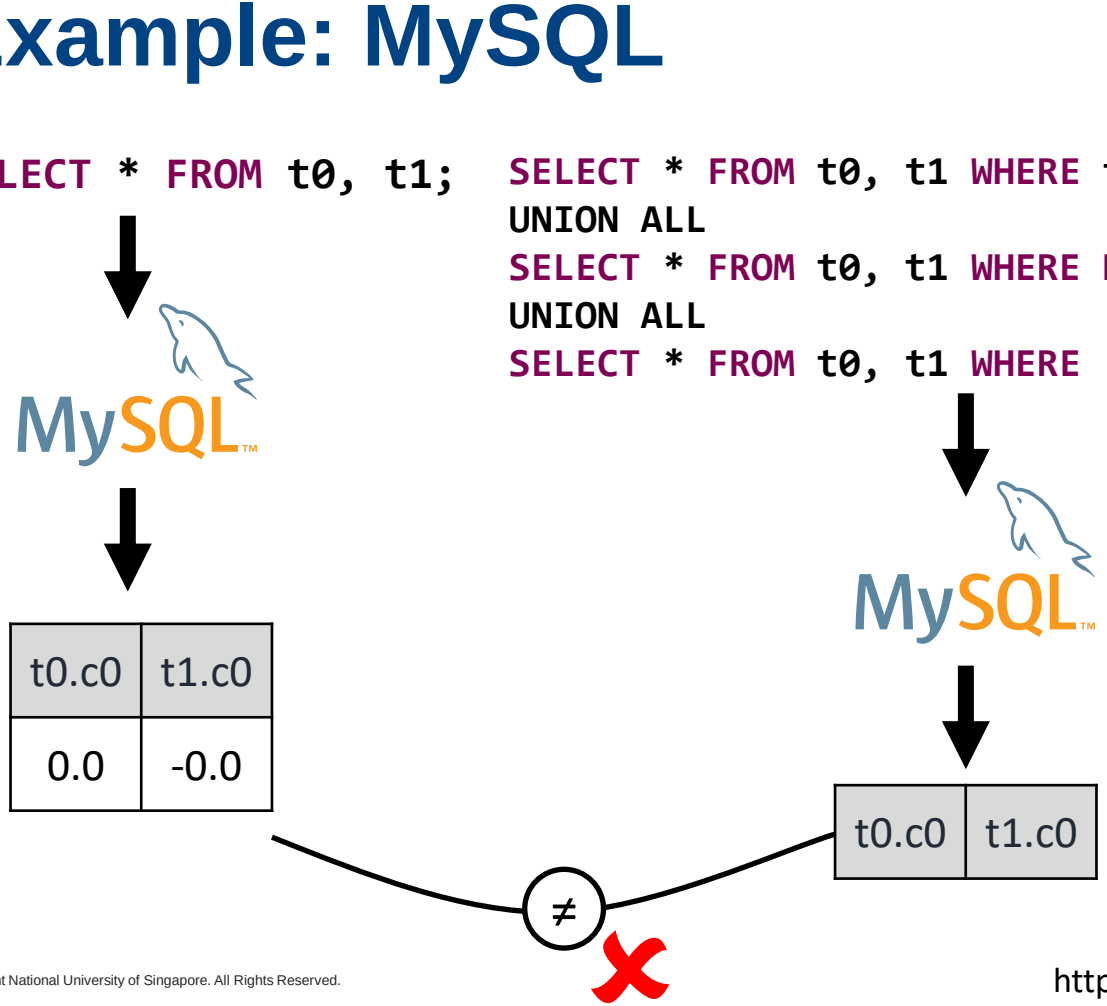


https://bugs.mysql.com/bug.php?id=99122

Example: MySQL

```
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

Scope

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

Testing WHERE Clauses

Q	Q'_{ptern}	$\Diamond(Q'_p, Q'_{\neg p}, Q'_{p\ \text{IS NULL}})$
SELECT <columns> **FROM** <tables> [<joins>]	**SELECT** <columns> **FROM** <tables> [<joins>] **WHERE** p_{tern}	$Q'_p \uplus Q'_{\neg p} \uplus Q'_{p\ \text{IS NULL}}$

Testing WHERE Clauses

Q	Q'_{ptern}	$\Diamond(Q'_p, Q'_{\neg p}, Q'_{p\ \text{IS NULL}})$
SELECT <columns> **FROM** <tables> [<joins>]	**SELECT** <columns> **FROM** <tables> [<joins>] **WHERE** p_{tern}	$Q'_p \uplus Q'_{\neg p} \uplus Q'_{p\ \text{IS NULL}}$

Testing WHERE Clauses

Q	Q'_{ptern}	$\Diamond(Q'_p, Q'_{\neg p}, Q'_{p \text{ IS NULL}})$
SELECT <columns> **FROM** <tables> [<joins>]	**SELECT** <columns> **FROM** <tables> [<joins>] **WHERE** p_{tern}	$Q'_p \uplus Q'_{\neg p} \uplus Q'_{p \text{ IS NULL}}$

Testing WHERE Clauses

Q	Q'_{ptern}	$\Diamond(Q'_p, Q'_{\neg p}, Q'_{p\ \text{IS NULL}})$
SELECT <columns> **FROM** <tables> [<joins>]	**SELECT** <columns> **FROM** <tables> [<joins>] **WHERE** p_{tern}	$Q'_p \uplus Q'_{\neg p} \uplus Q'_{p\ \text{IS NULL}}$

Testing WHERE Clauses

Q	Q'_{ptern}	$\Diamond(Q'_p, Q'_{\neg p}, Q'_{p\ \text{IS NULL}})$
SELECT <columns> **FROM** <tables> [<joins>]	**SELECT** <columns> **FROM** <tables> [<joins>] **WHERE** p_{tern}	$Q'_p \uplus Q'_{\neg p} \uplus Q'_{p\ \text{IS NULL}}$

The multiset addition can be implemented using **UNION ALL**

Scope

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

Testing Self-decomposable Aggregate Functions

Q	Q'_{ptern}	$\Diamond(Q'_p, Q'_{\neg p}, Q'_{p \text{ IS NULL}})$
SELECT MAX(<e>) **FROM <tables>** **[<joins>]**	**SELECT MAX(<e>)** **FROM <tables>** **[<joins>]** **WHERE p_{tern};**	$MAX(Q'_p \uplus Q'_{\neg p} \uplus Q'_{p \text{ IS NULL}})$

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

Testing Self-decomposable Aggregate Functions

Q	Q'_{ptern}	$\Diamond(Q'_p, Q'_{\neg p}, Q'_{p \text{ IS NULL}})$
SELECT MAX(<e>) **FROM <tables>** **[<joins>]**	**SELECT MAX(<e>)** **FROM <tables>** **[<joins>]** **WHERE p_{tern};**	$MAX(Q'_p \uplus Q'_{\neg p} \uplus Q'_{p \text{ IS NULL}})$

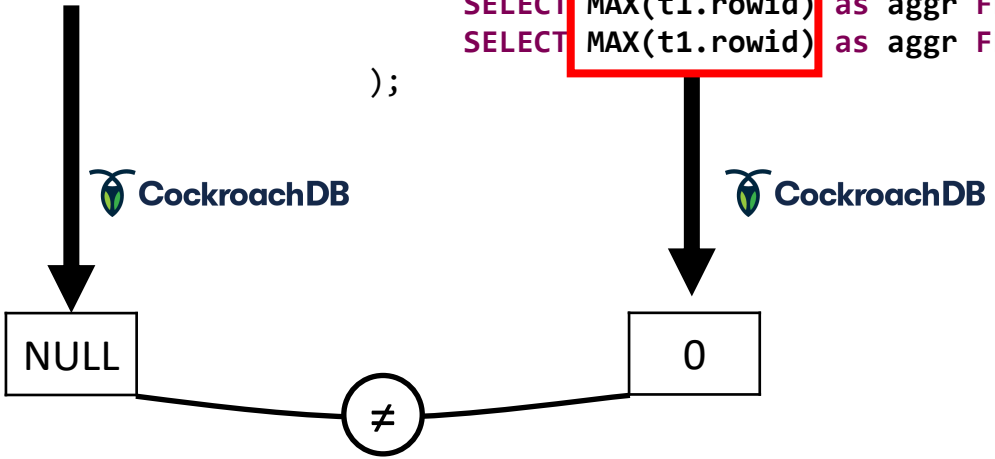
We use **MAX** in the **composition operator** to compute the overall maximum value

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
);
```

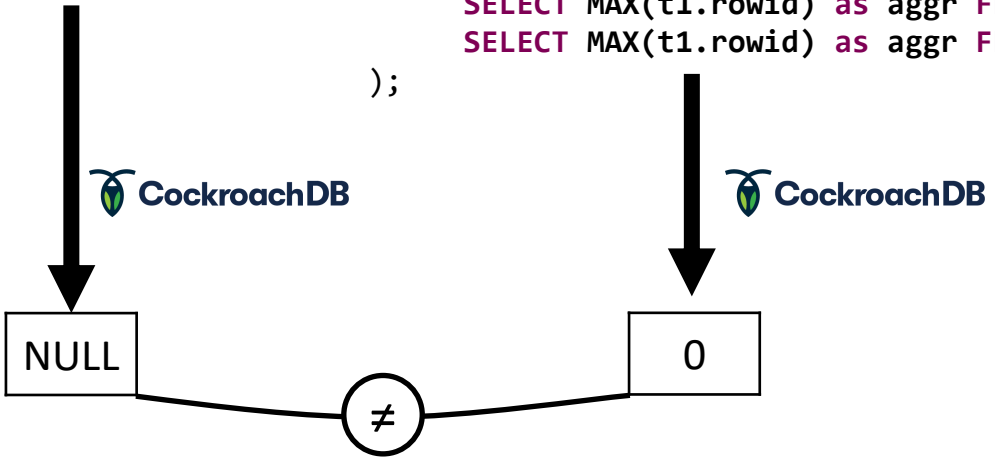


Bug Example: CockroachDB

```
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}	$\Diamond(Q'_p, Q'_{\neg p}, Q'_{p \text{ IS NULL}})$

`SELECT AVG(<e>)`
`FROM <tables>`
`[<joins>];`
	`SELECT SUM(<e>) as s,`
	`COUNT(<e>) as c`
	`FROM <tables>`
	`[<joins>];`
		$\frac{SUM(s(Q'_p \uplus Q'_{\neg p} \uplus Q'_{p \text{ IS NULL}}))}{SUM(c(Q'_p \uplus Q'_{\neg p} \uplus Q'_{p \text{ IS NULL}}))}$

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

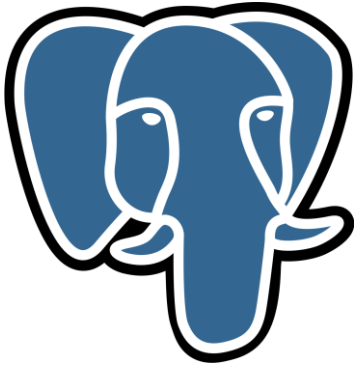
Scope

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

Similar insights can be used to test **other SQL features**

Evaluation

PostgreSQL



We **evaluated the effectiveness of** our approach in a three-month period on **seven** widely-used DBMSs

Evaluation: Found Bugs

			Closed	
DBMS	Fixed	Verified	Intended	Duplicate
SQLite	4	0	0	0
MySQL	1	6	3	0
H2	16	2	0	1
CockroachDB	23	8	0	0
TiDB	26	35	0	1
DuckDB	72	0	0	2

We found **193 unique, previously unknown bugs**, 142 of which have been fixed!

Evaluation: Found Bugs

	Query Partitioning Oracle						
DBMS	WHERE	Aggregate	GROUP BY	HAVING	DISTINCT	Error	Crash
SQLite	0	3	0	0	1	0	0
CockroachDB	3	3	0	1	0	22	2
TiDB	29	0	1	0	0	27	4
MySQL	7	0	0	0	0	0	0
DuckDB	21	4	1	2	1	13	19
H2	2	0	0	0	0	16	0

Evaluation: Found Bugs

	Query Partitioning Oracle						
DBMS	WHERE	Aggregate	GROUP BY	HAVING	DISTINCT	Error	Crash
SQLite	0	3	0	0	1	0	0
CockroachDB	3	3	0	1	0	22	2
TiDB	29	0	1	0	0	27	4
MySQL	7	0	0	0	0	0	0
DuckDB	21	4	1	2	1	13	19
H2	2	0	0	0	0	16	0

The WHERE oracle is the **simplest**, but **most effective** oracle

Evaluation: Found Bugs

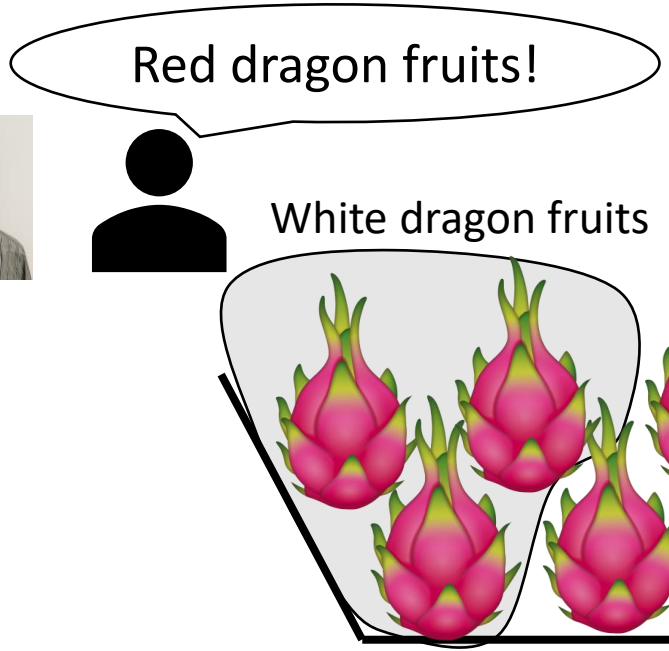
	Query Partitioning Oracle						
DBMS	WHERE	Aggregate	GROUP BY	HAVING	DISTINCT	Error	Crash
SQLite	0	3	0	0	1	0	0
CockroachDB	3	3	0	1	0	22	2
TiDB	29	0	1	0	0	27	4
MySQL	7	0	0	0	0	0	0
DuckDB	21	4	1	2	1	13	19
H2	2	0	0	0	0	16	0

The other oracles found **interesting**, but **fewer** bugs

Evaluation: Found Bugs

	Query Partitioning Oracle						
DBMS	WHERE	Aggregate	GROUP BY	HAVING	DISTINCT	Error	Crash
SQLite	0	3	0	0	1	0	0
CockroachDB	3	3	0	1	0	22	2
TiDB	29	0	1	0	0	27	4
MySQL	7	0	0	0	0	0	0
DuckDB	21	4	1	2	1	13	19
H2	2	0	0	0	0	16	0

Limitations



TLP does not provide a ground truth and **fails to detect consistently incorrect cases**

Recap: Metamorphic Testing

- Why is TLP a metamorphic testing approach?

Recap: Metamorphic Testing

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

MySQL →

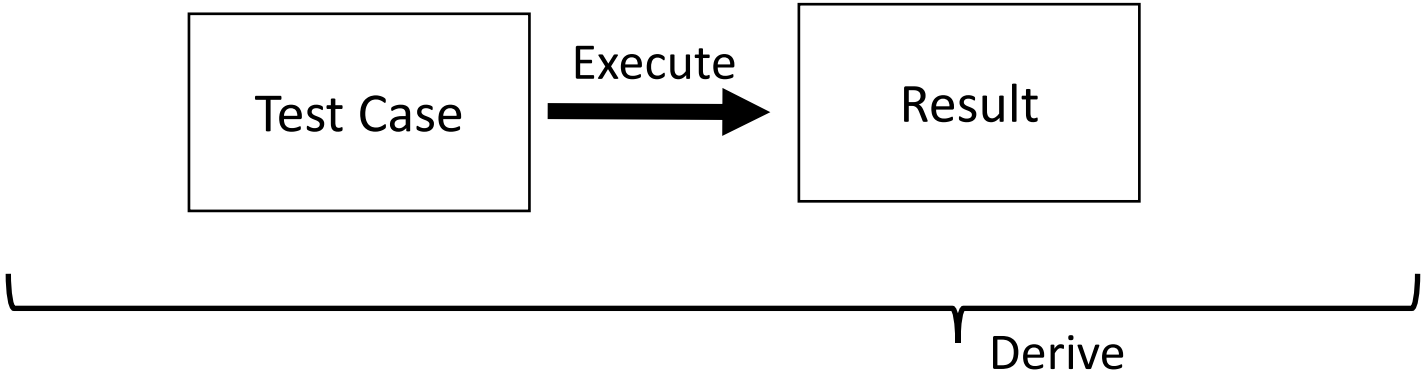
t0.c0	t1.c0
0.0	-0.0

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;
```

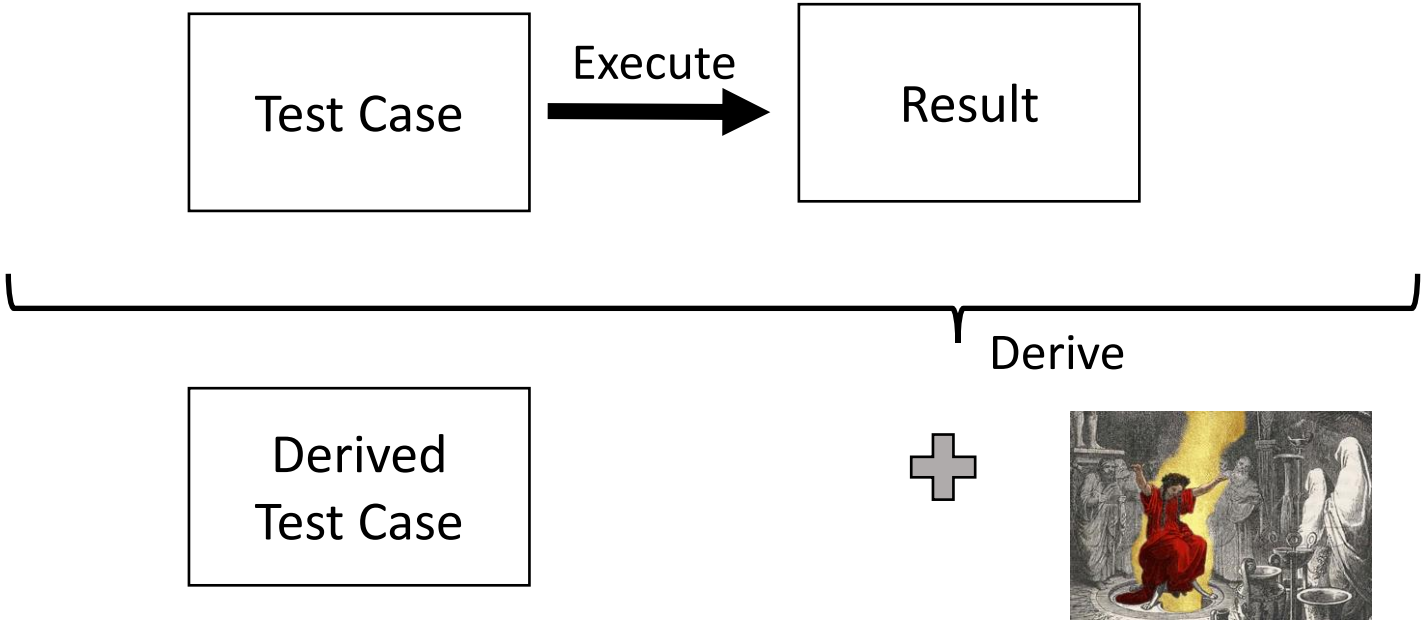
+

Recap: 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;
```

Recap: Metamorphic Testing



Finding Logic Bugs in DBMSs

Ternary Logic Partitioning

OOPSLA ‘20

Non-optimizing Reference Engine Construction

ESEC/FSE ‘20

Pivoted Query Synthesis

OSDI ‘20

Thinking About Research

- What are the limitations of the technique?
- Can we design other partitioning strategies?
- Can we apply this technique to other domains?

Comparison

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

Both TLP are metamorphic testing approaches

Finding Logic Bugs in DBMSs

Ternary Logic Partitioning

OOPSLA '20

Generate a query for which it is **ensured** that a randomly-selected row, the **pivot row**, **is fetched**

Non-optimizing Reference Engine Construction

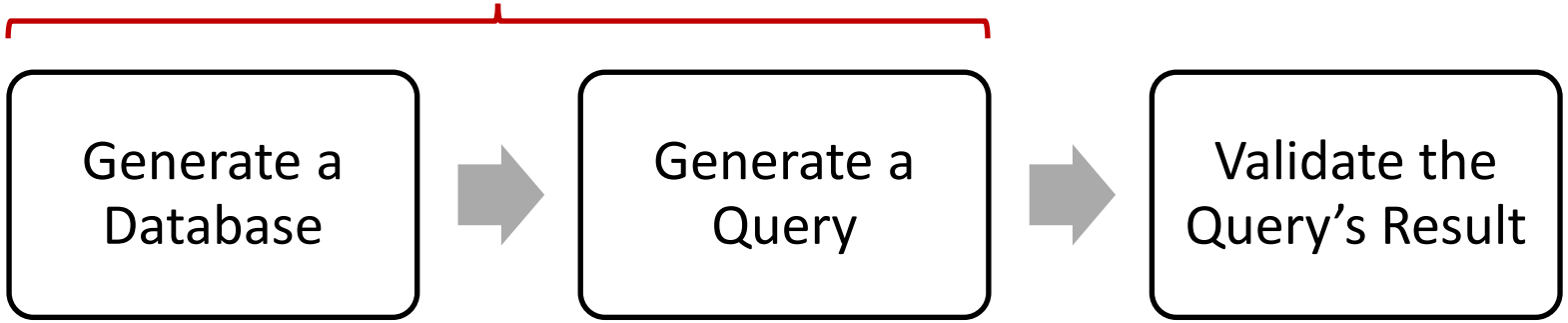
ESEC/FSE '20

Pivoted Query Synthesis

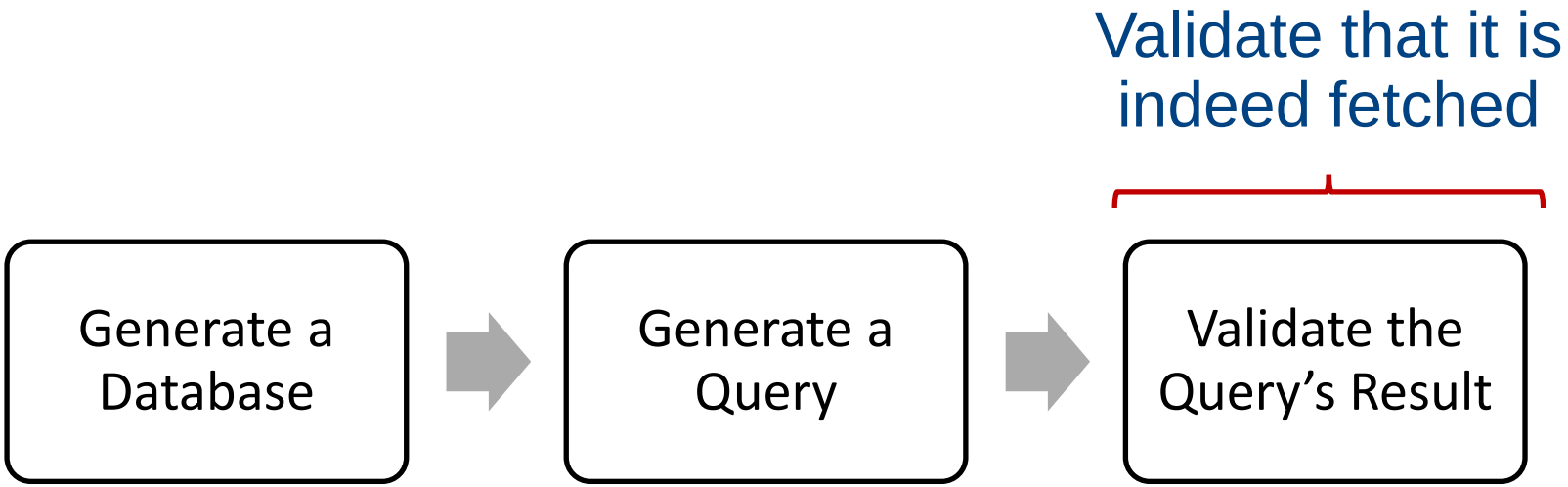
OSDI '20

Automatic Testing Core Challenges

Generate a query that at least fetches on row



Automatic Testing Core Challenges



Comparison

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

PQS uses expression evaluators to provide a ground truth

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;
```

0

TRUE

SQLite

0

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;
```

SQLite

0

1

FALSE

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;
```

2

TRUE

SQLite

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

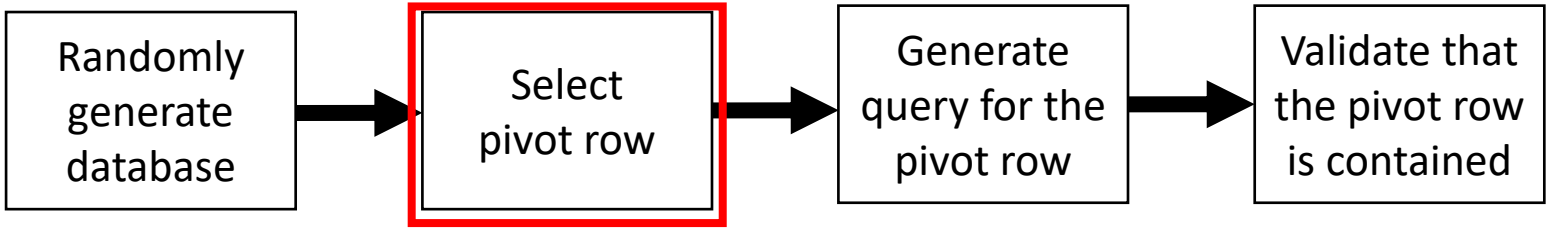
TRUE

0
2

NULL was not contained in the result set!

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

Approach

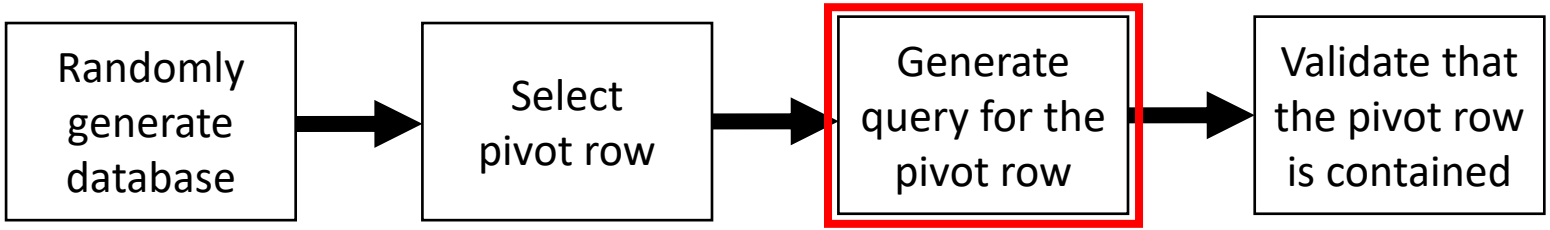


c0
0
1
2
NULL

Pivot row

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

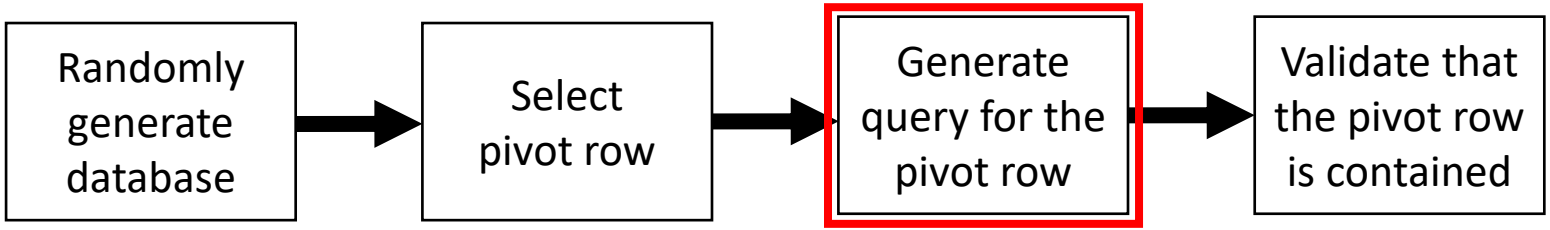
Approach

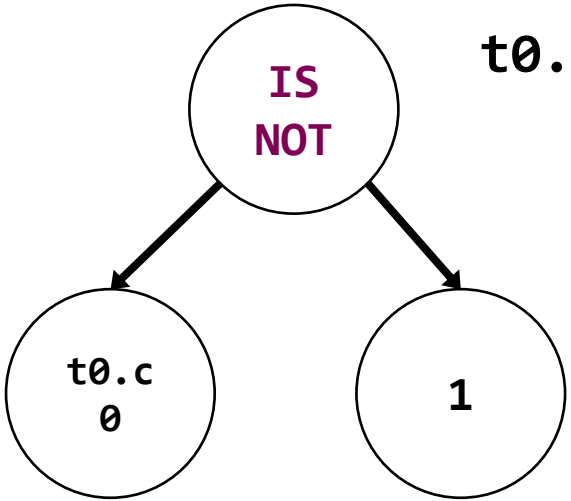



```
SELECT c0 FROM t0
WHERE [                    ]
```

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

Random Expression Generation

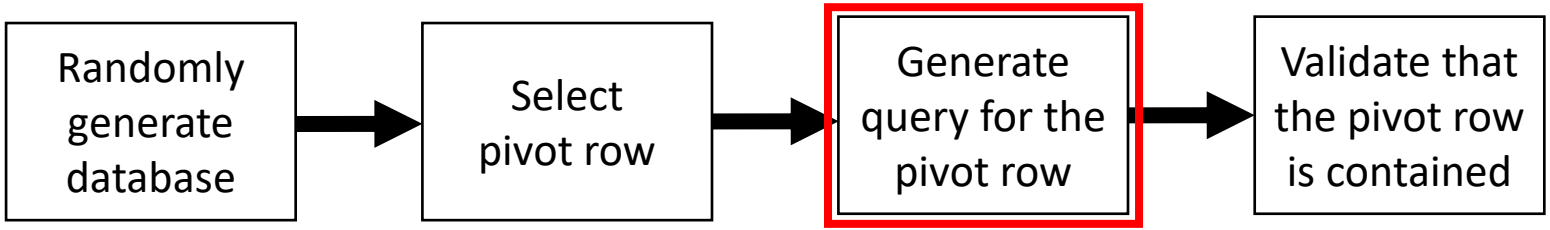





```
t0.c0 IS NOT 1;
```

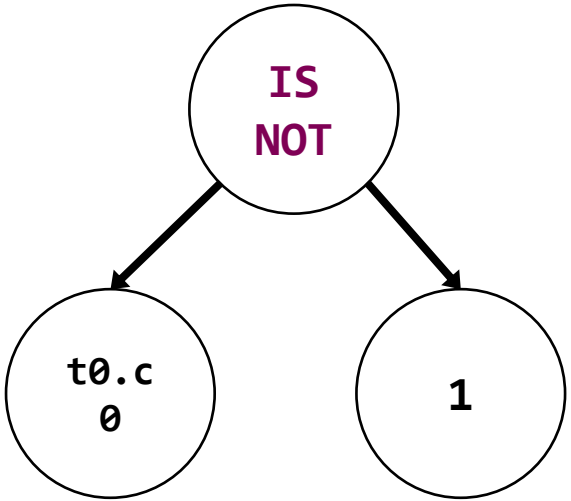
We implemented an **expression evaluator** for each node

Random Expression Generation



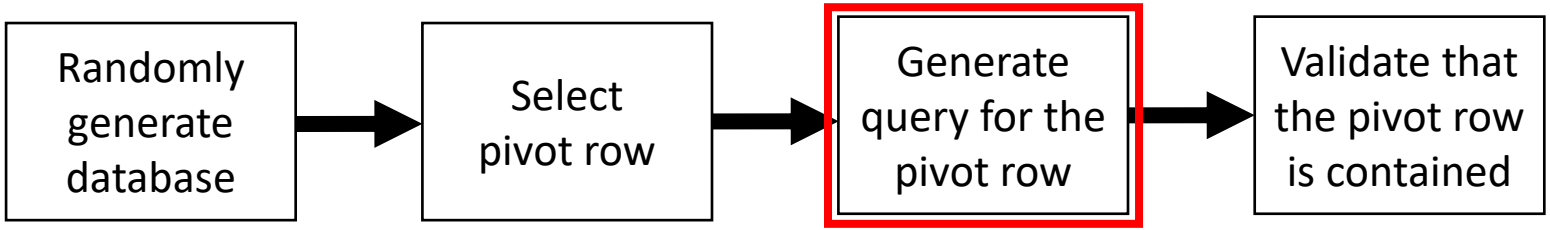
t0

c0
0
1
2
NULL

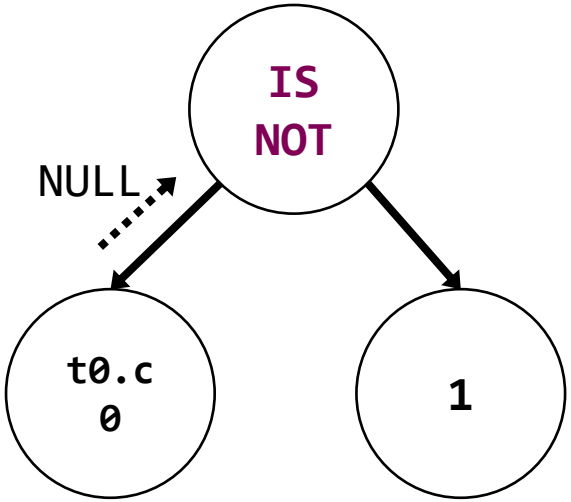


Evaluate the tree based on the **pivot row**

Random Expression Generation



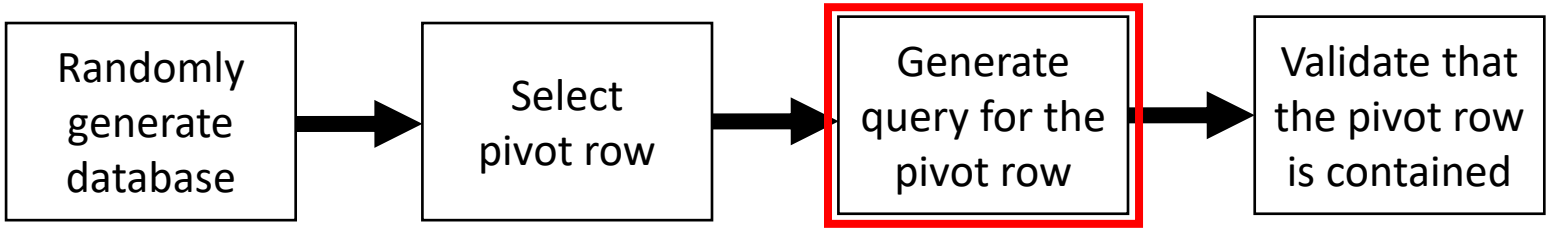
Column references return the **values** from the **pivot row**

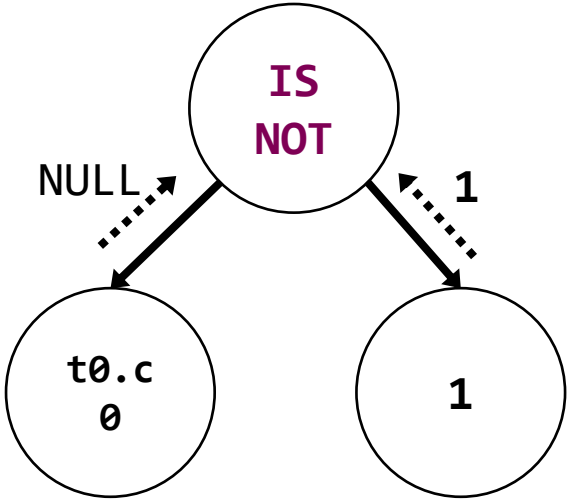


t0

c0
0
1
2
NULL

Random Expression Generation



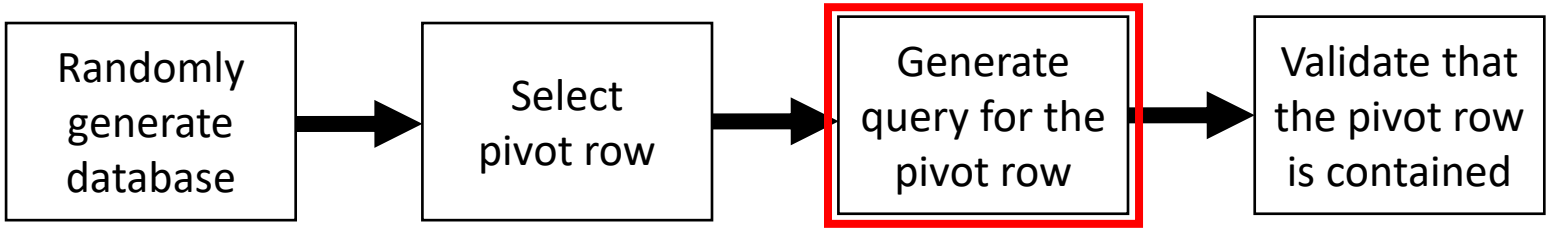


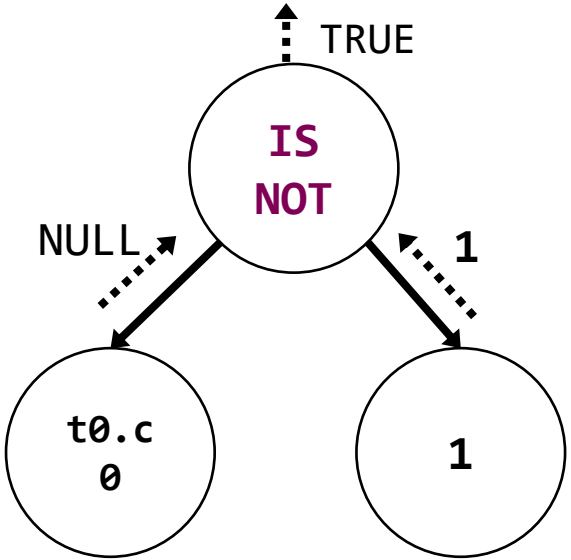
Constant nodes return their assigned **literal values**

t0

c0
0
1
2
NULL

Random Expression Generation



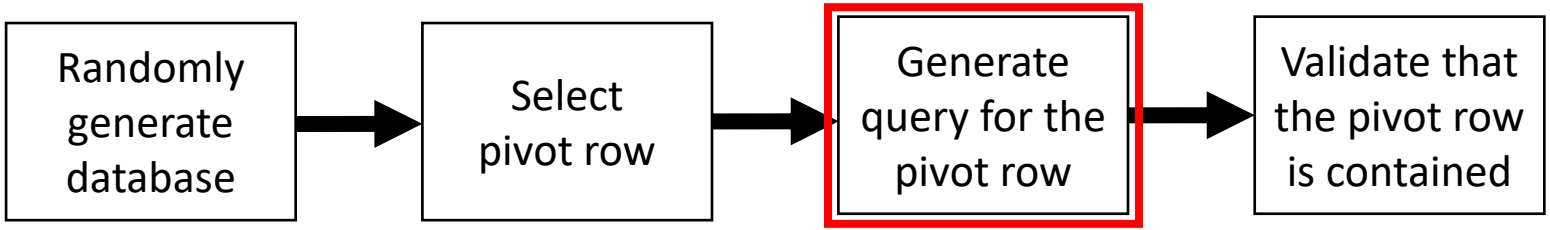


Compound nodes compute their result **based on their children**

t0

c0
0
1
2
NULL

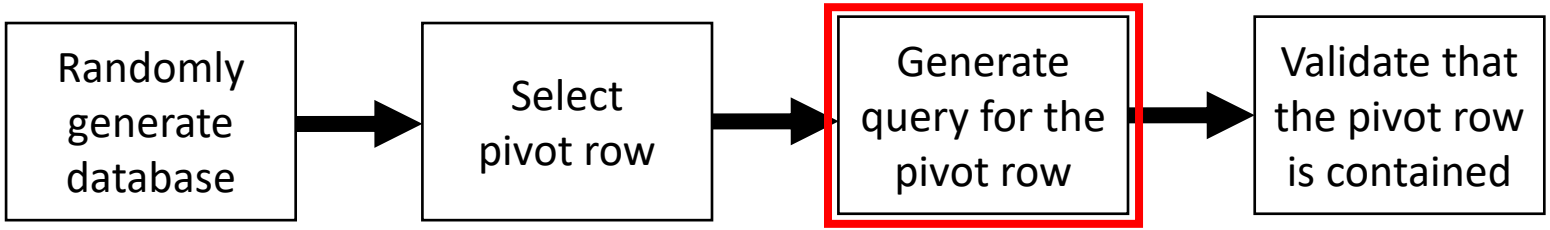
Query Synthesis




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

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

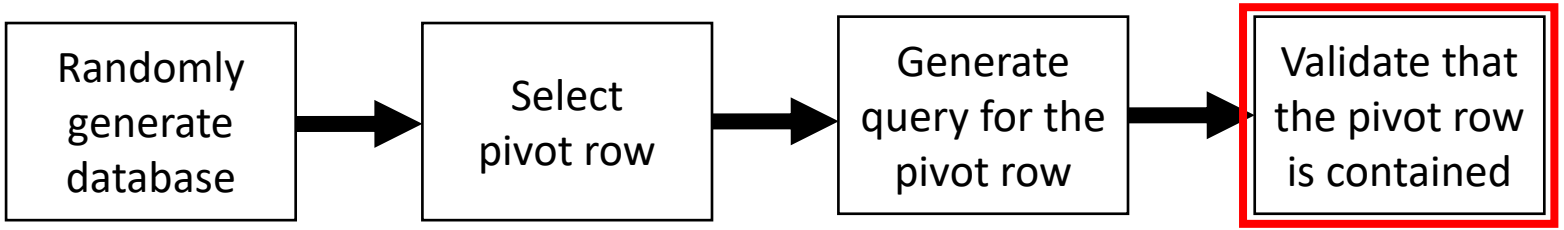
Random Expression Rectification



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

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

Approach




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

Rely on the DBMS to check whether the row is contained

Bugs Overview

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

96 unique, previously unknown bugs

Summary

SQLancer

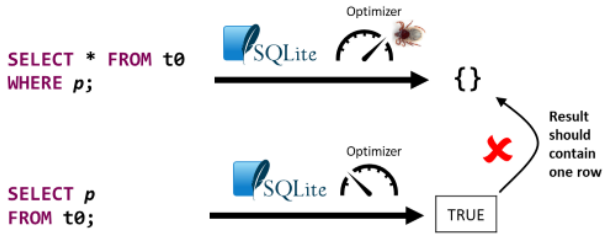
968 stars 148 forks

SQLANCER

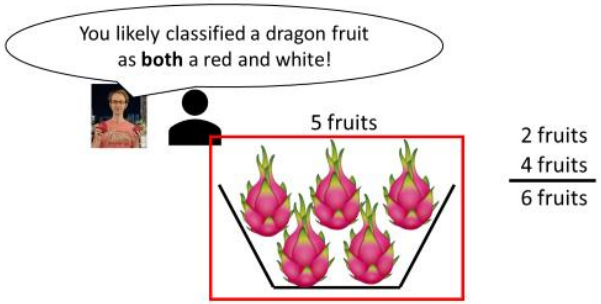
https://github.com/sqlancer

@sqlancer_dbms

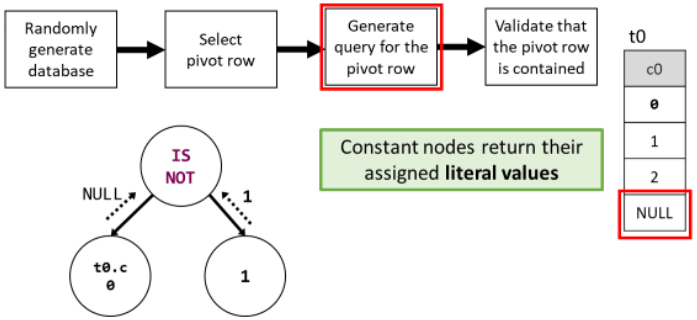
Translation Step



Tangerines vs. Clementines



Random Expression Generation



Reminders

- Next Lecture
  - Data-Oriented Differential Testing of Object-Relational Mapping Systems
  - APOLLO: automatic detection and diagnosis of performance regressions in database systems
- Upcoming deadlines
  - Team composition: September 15
    - Email me with your teammate (if you do the project in a pair)
  - Project proposal: September 16