ValScope: Value-Semantics-Aware Metamorphic Testing for Detecting Logical Bugs in DBMSs





Li Lin, Liehang Chen, Rongxin Wu
School of Informatics Xiamen University

OSDI26

Database Management System

- **Important**

On January 11, 2023, a DBMS correctness bug resulted in over 9,700 flights being delayed and over 1,300 flights being canceled.[1]



- **Widely used**

There are 263,223 companies worldwide using MySQL.[2]

MySQL MariaDB TiDB OCEANBASE

1 https://mp.weixin.qq.com/s/fwn3IAIpn-F7HRLBkg_WAA
2 https://enlyft.com/tech/products/mysql

Logical Bugs

- Make a DBMS return an incorrect result set for a given SQL query.

```
CREATE TABLE t (c1 FLOAT);
INSERT INTO t VALUES (-1);

(SELECT 1 FROM t WHERE COT(0.2)=0)
UNION ALL (SELECT (BINARY c1 | 0) FROM t);
--result: {0}
```



```
SELECT (BINARY c1 | 0) FROM t;
```

```
+----------------------------+
|  (BINARY c1 | 0)           |
+----------------------------+
|  18446744073709551615 |
+----------------------------+
1 row in set (0.001 sec)
```

Challenge



- Do not result in apparent manifestations, e.g., crashes
- Go unnoticed by users

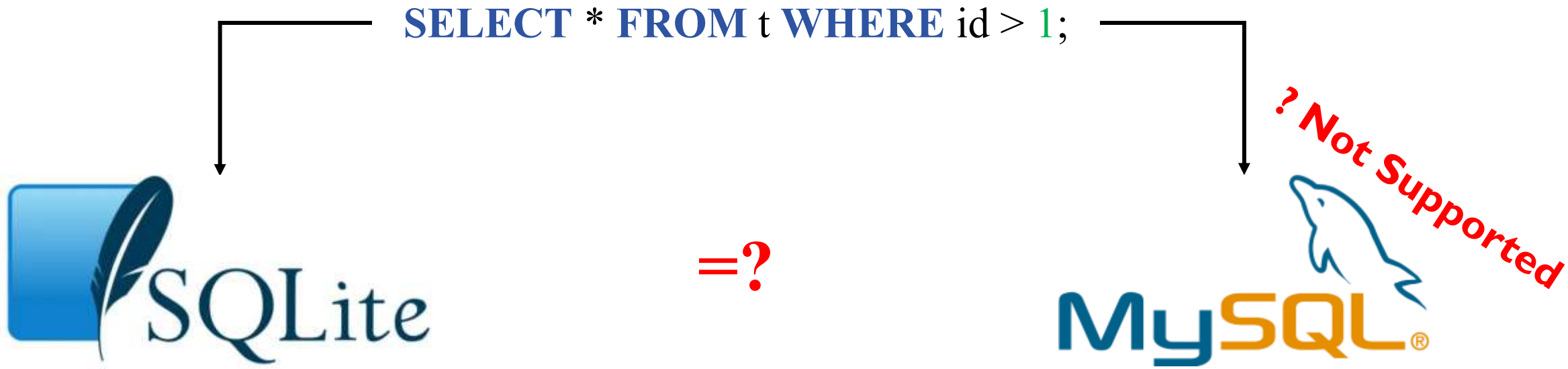
How to design the test oracle?

How to Design the Test Oracle?

- **Differential Testing**
  - Slutz RAGS (VLDB'98)
- **Oracle-guided Synthesis**
  - Rigger et al. PQS (OSDI 2020)
- **Metamorphic Testing**
  - ***Equivalence Relation***: Rigger et al. SQLancer (FSE'20, OOPSLA'20)
  - ***Set-semantic Approximation Relation***: Hao et al. Pinolo (ATC'23)

Differential Testing

Massive stochastic testing of SQL [Slutz, VLDB 1998]



Oracle-guided Synthesis

Pivoted Query Synthesis (PQS) [Rigger, OSDI 2020]

ID	NAME	AGE
1	Alice	18
2	Bob	18
3	Tom	10

pivot row →

```
SELECT * FROM t WHERE
ID > 1 AND AGE > 10;
```

Is the pivot row in the result set?



Oracle-guided Synthesis

Pivoted Query Synthesis (PQS) [Rigger, OSDI 2020]

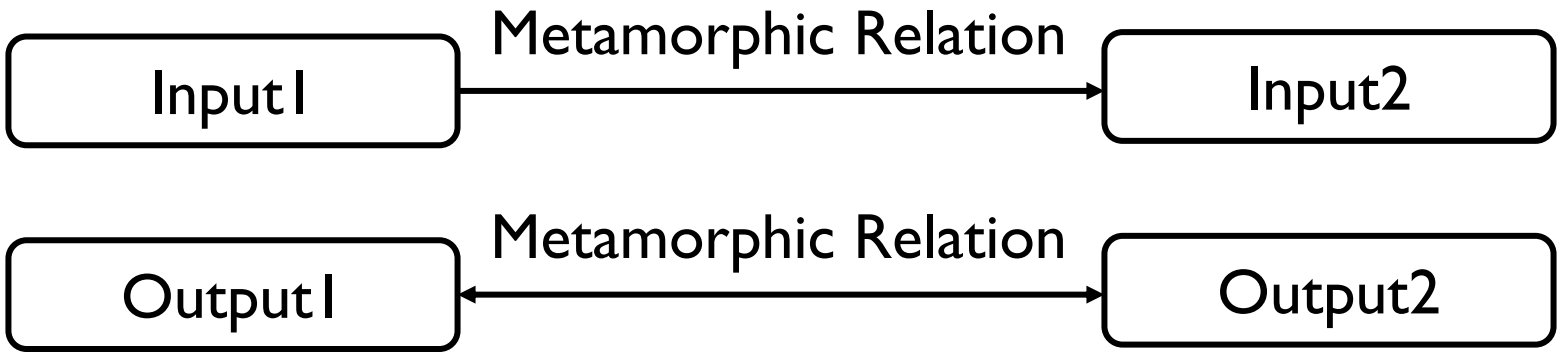
```
(SELECT 1 FROM t WHERE COT(0.2)=0)
UNION ALL (SELECT (BINARY c1 | 0) FROM t);
```



PQS cannot perform operator processing on the original row data

Miss the bug!

Metamorphic Testing



Metamorphic Relation

- ***Equivalence Relation***： Rigger et al. SQLancer (ESEC/FSE'20, OOPSLA'20)
- ***Set-semantic Approximation Relation***: Hao et al. Pinolo (ATC'23)

Equivalence-based Metamorphic Testing

Non-optimizing Reference Engine Construction(NoREC) [Rigger, FSE 2020]

SELECT * FROM t **WHERE** ID=2; ⟶ **SELECT** ID=2 **FROM** t;

ID	NAME	AGE
2	Bob	18

=?

ID=2
0
1
0

Equivalence-based Metamorphic Testing

Non-optimizing Reference Engine Construction(NoREC) [Rigger, FSE 2020]

```
CREATE TABLE t (c1 DOUBLE);
INSERT INTO t VALUES (-79.1819);

SELECT c1-DATE_SUB('2008-05-25', INTERVAL 1 DAY) AS f1 FROM t
HAVING f1 != 0;
--result: {-1928.8181}
```



NoREC share the same buggy operators

```
SELECT (c1-DATE_SUB('2008-05-25', INTERVAL 1 DAY)) != 0 FROM t;
--result: {1}
```

Miss the bug!

Set-semantic Approximation Metamorphic Testing

Approximate Query Synthesis(Pinolo) [Hao, ATC 2023]

Set-based Over Approximation

SQL1 $\xrightarrow{\subseteq}$ **SQL2**

Set-based Under Approximation

SQL1 $\xrightarrow{\supseteq}$ **SQL2**

```
SELECT c1-DATE_SUB('2008-05-25', INTERVAL 1 DAY) AS f1 FROM t
HAVING f1 != 0;
--RESULT1: {-1928.8181}
```

Over Approximation

$\text{RESULT1} \subseteq \text{RESULT2}$?

```
SELECT c1-DATE_SUB('2008-05-25', INTERVAL 1 DAY) AS f1 FROM t
HAVING TRUE;
--RESULT2: {-20080444.8181}
```

Set-semantic Approximation Metamorphic Testing

- Do not support all features of DBMSs:
  - ➢aggregate functions
  - ➢window functions
  - ➢left/right join
- These features break the inclusion relation:

c1
2
1
0

```
SELECT SUM(c1) FROM t WHERE c1 > 1;
--result: {2}
```

Set → Value

Value increase?

```
SELECT SUM(c1) FROM t WHERE c1 >= 1;
-result: {-20.0001}
```

Miss the bug!

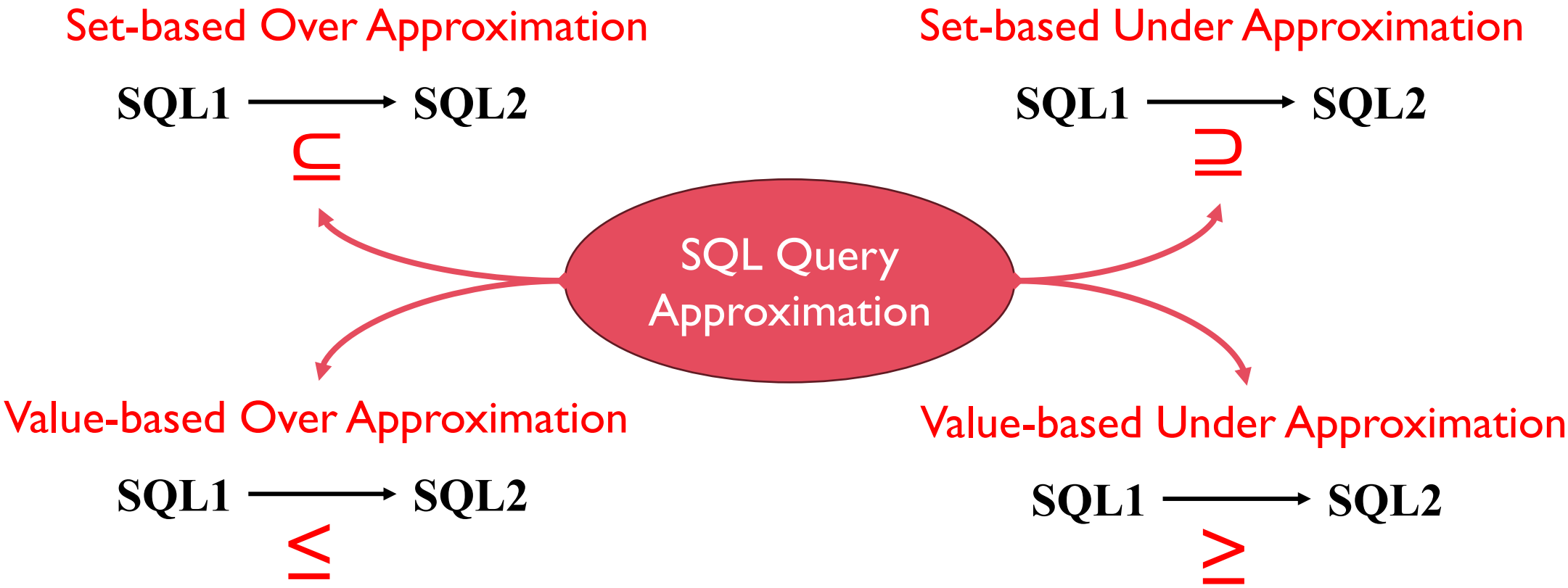
Limitation of Existing Logical Bug Detection Techniques



- Differential Testing: dialect problem
- Oracle-guided Synthesis: unable to generate complex SQL statements
- **Equivalence-based Metamorphic Testing: insufficient to detect deeply-hidden bugs**
- **Set-semantic Approximation Metamorphic Testing : blind to value-semantic bugs**

Our Approach: ValScope

- ValScope reasons about how values should change, not just which tuples appear.



An Example of Value-based Under Approximation

c2	c1
A	10
A	20
B	5
B	7

Q1: **SELECT** c2, **MAX**(c1) **FROM** t **GROUP BY** c2;

Value decrease?

Q2: **SELECT** c2, **MIN**(c1) **FROM** t **GROUP BY** c2;

Result Q1

c2	MAX(c1)
A	20
B	7

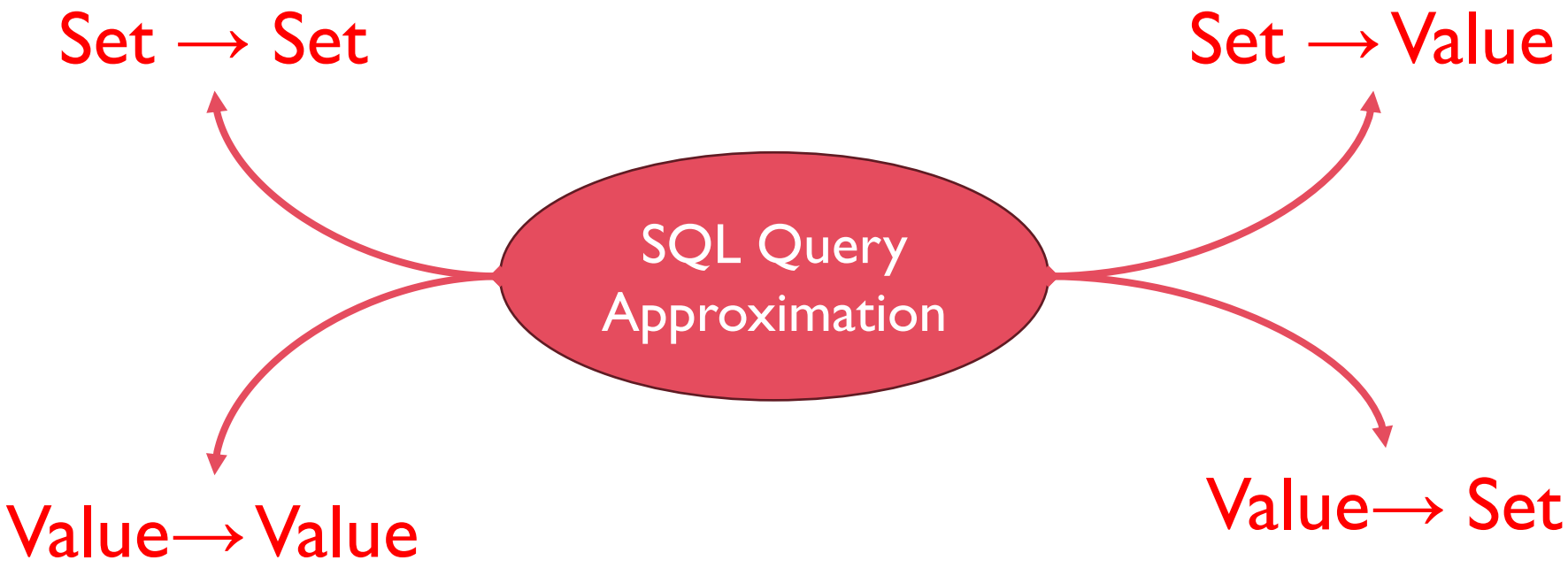
≥

Result Q2

c2	MIN(c1)
A	10
B	5

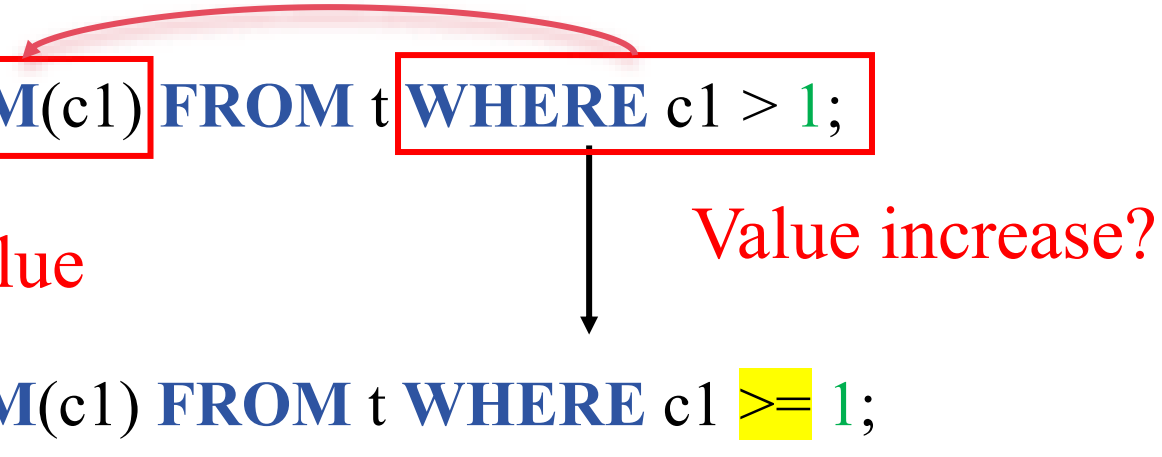
Approximation Propagation

- Local mutation at AST node
- Bottom-up propagation through SQL AST



An Example of Set → Value Approximation

c1
2
1
0

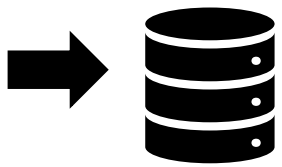


ValScope: Overview

① Populate database tables randomly

t1:

c0	c1
1	2
2	-1

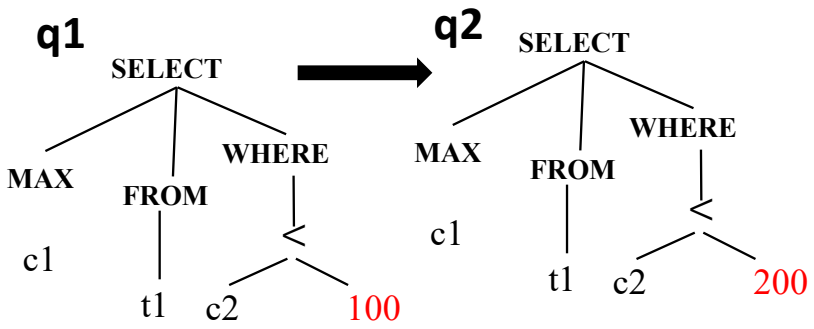


② Generate original queries

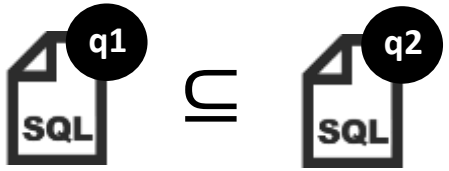
SQL1: SELECT MAX(c1)
FROM t1 **WHERE** c2<100; --r1

Back to ② and iterate

③ Generate mutated queries

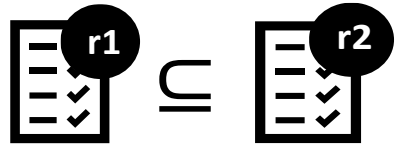


④ Approximation Propagation Analysis

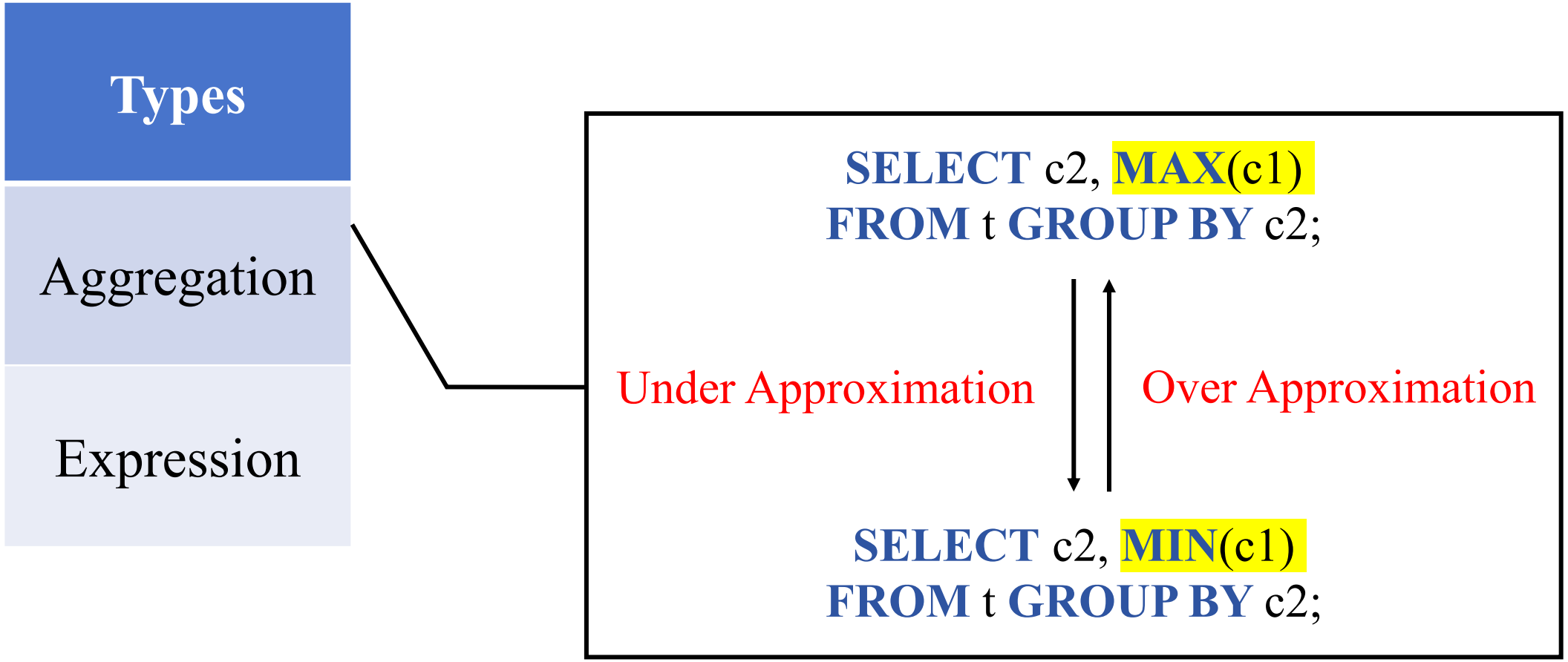


Value-level under-approximation

⑤ Results Checking



ValScope: Value-Semantic Mutators

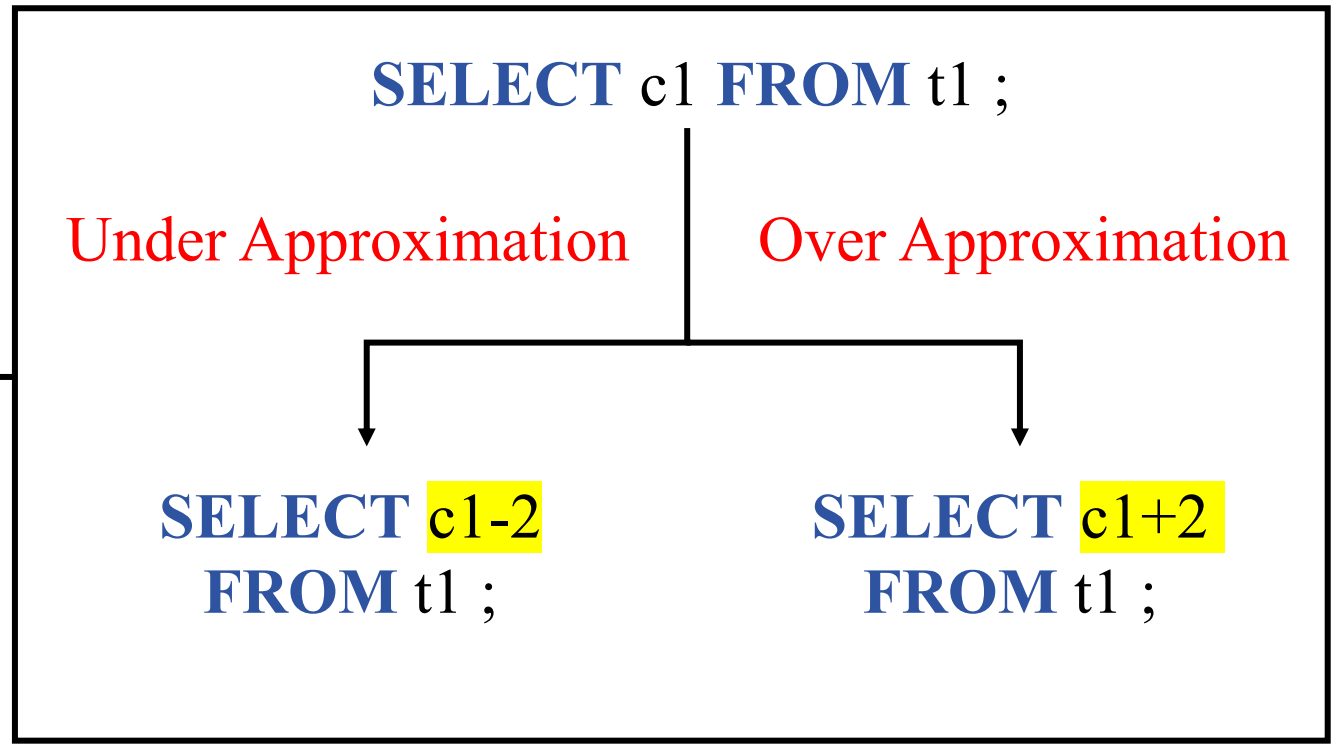


ValScope: Value-Semantic Mutators

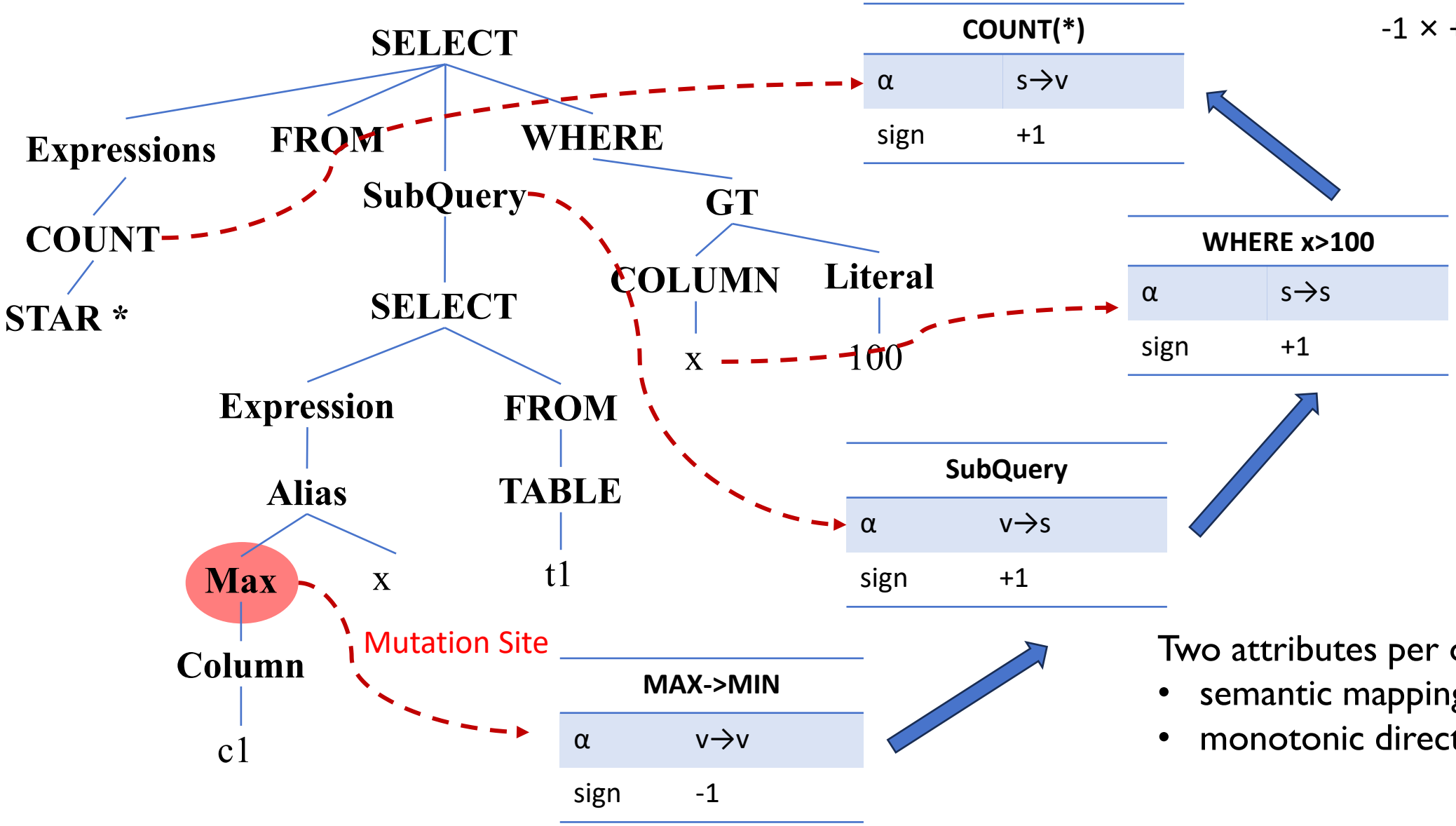
Types

Aggregation

Expression



ValScope: Approximation Propagation Analysis



Two attributes per operator:

- semantic mapping α (set ↔ value)
- monotonic direction sign (+ / –)

Evaluation

- **RQ1:** Can ValScope find real logical bugs in widely used and extensively tested DBMSs?

- **RQ2:** Can ValScope find logical bugs that are missed by existing approaches?

Experimental Setup

Tested DBMS:

DBMS	GitHub Stars	DB-Engine
MySQL	11.9K	2
MariaDB	6.7K	13
OceanBase	9.8K	99
Percona	1.2K	122
PolarDB	1.6K	105
TiDB	39.4K	85









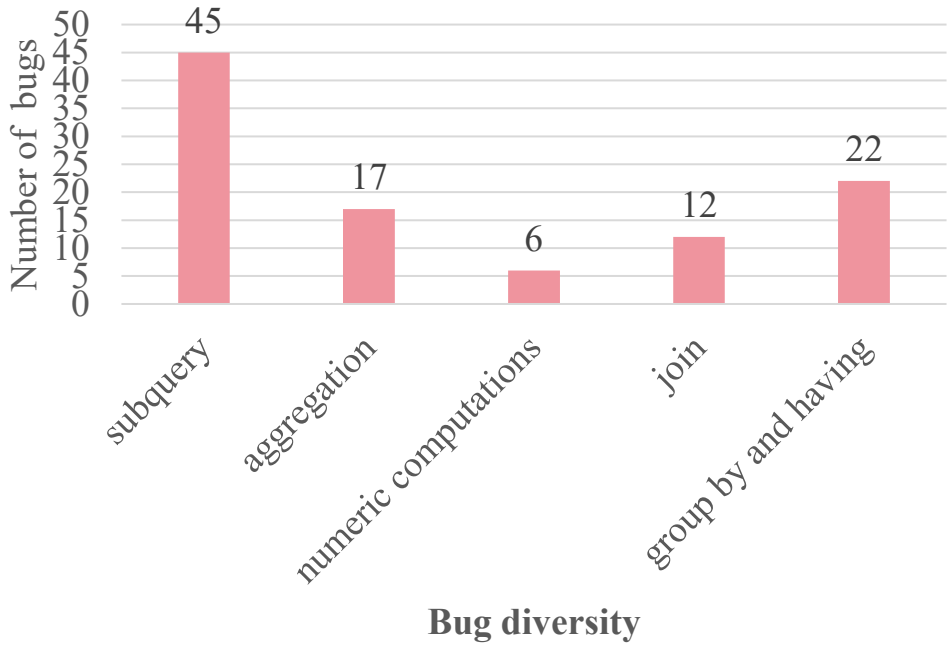




RQ1: Bug Detection

- ValScope is effective in detecting logical bugs:
  - ➢67 unique logical bugs, 57 have been confirmed by developers
  - ➢1 serious bug in MySQL, 10 major bugs in MariaDB

DBMS	Reported	Confimed	Waiting
MySQL	23	23	0
MariaDB	10	10	0
OceanBase	7	6	1
Percona	10	10	0
PolarDB	12	7	5
TiDB	5	1	4
Total	67	57	10



RQ2: Comparative Study

- ValScope is effective in finding long-standing bugs overlooked by earlier techniques

DBMS	Found	<2020	2020-2023	Longest latency
MySQL	23	18	1	20 years
MariaDB	10	9	1	13 years
OceanBase	6	3	3	5 years
Percona	10	10	0	12 years
PolarDB	7	7	0	7 years
TiDB	1	1	0	3 years
Total	57	48	5	20 years

VapScope is a DBMS testing framework that detects logical bugs using a unified set-semantic and value-semantic approximation model.

Database Management System

- **Important**

On January 11, 2023, a DBMS correctness bug resulted in over 9,700 flights being delayed and over 1,300 flights being canceled.[1]



- **Widely used**

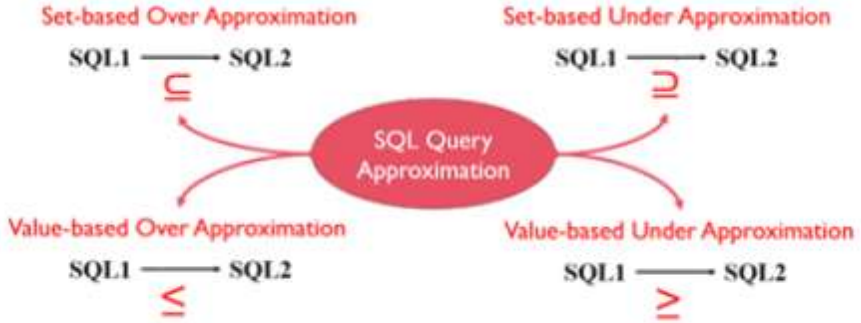
There are 263,223 companies worldwide using MySQL.[2]

MySQL MariaDB TiDB OCEANBASE

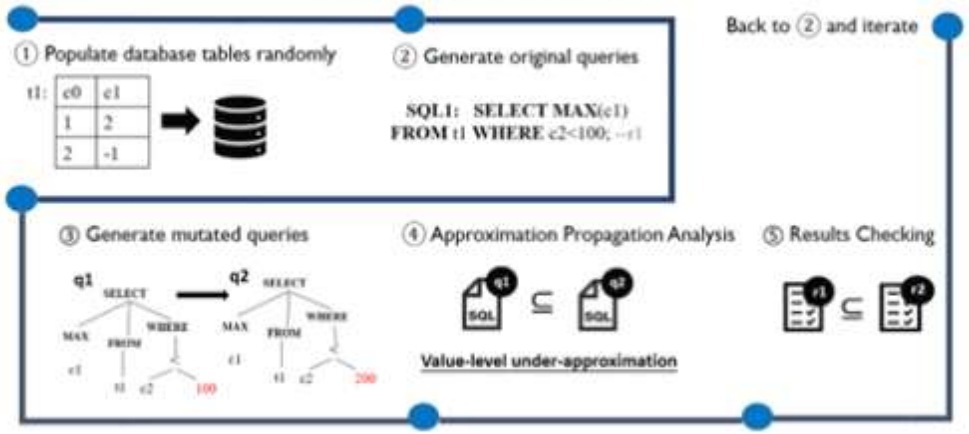
1 https://mp.weixin.qq.com/s/fwn3IAIpn-F7HRLBkg_WAA
2 https://enlyft.com/tech/products/mysql

Our Approach: ValScope

- ValScope reasons about how values should change, not just which tuples appear.



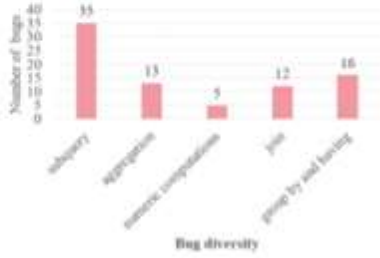
ValScope: Overview



RQ1: Bug Detection

- ValScope is effective in detecting logical bugs:
  - ➢62 unique logical bugs, 44 have been confirmed by developers
  - ➢1 serious bug in MySQL, 8 major bugs in MariaDB

DBMS	Reported	Confirmed	Waiting
MySQL	19	16	3
MariaDB	9	8	1
OceanBase	7	6	1
Percona	10	9	1
PolarDB	12	4	8
TiDB	5	1	4
Total	62	44	18



Thanks!
Q & A

Open source: https://github.com/linli1724647576/ValScope

Paper: https://matafeiyanll.github.io/paper/osdi26.pdf

Email: linli1210@stu.xmu.edu.cn

Paper



Motivating Example

Compare to Set-semantic Approximation Metamorphic Testing

```
CREATE TABLE t1 (c1 INT , c4 INT);
INSERT INTO t1 (c1,  c4) VALUES (3584,  36);
CREATE UNIQUE INDEX idx_t1_pk_c4 ON t1 (c1, c4);

-- Original Query
SELECT t1.c1 AS c1, SUM(DISTINCT t1.c4*2) AS col_1, DENSE_RANK() OVER (ORDER BY 1 = 1)
AS col_1_2 FROM t1 AS t1 WHERE (NOT REGEXP_LIKE(t1.c1, '[0-9]{3}-[0-9]{2}-[0-9]{4}'))
GROUP BY t1.c1;
--result:   3584 |     72 |         1

-- Mutated Query
SELECT t1.c1 AS c1, SUM(DISTINCT t1.c4) AS col_1, DENSE_RANK() OVER (ORDER BY 1 = 1) AS
col_1_2 FROM t1 AS t1 WHERE (NOT REGEXP_LIKE(t1.c1, '[0-9]{3}-[0-9]{2}-[0-9]{4}')) GROUP BY
t1.c1;
--result:   3584 |     36 |         1          --(expected)
--result:   3584 |     NULL |    1          --(actual)
```



Implementation

- **Database population**
  - ➢Boundary and extreme values included
  - ➢Explicit control of value sign (±)
- **Query generation and parsing**
  - ➢Type-consistent expressions and functions
  - ➢SQLGlot
- **Bug Report Post-processing**
  - ➢Bug simplification using SQLess



SQLess: Dialect-Agnostic SQL Query Simplification

Li Lin
Xiamen Key Laboratory of Intelligent Storage and Computing, School of Informatics, Xiamen University
Xiamen, Fujian, China
lindi1216@stu.xmu.edu.cn

Zongyin Hao
Xiamen Key Laboratory of Intelligent Storage and Computing, School of Informatics, Xiamen University
Xiamen, Fujian, China
haozongyin@stu.xmu.edu.cn

Chengpeng Wang
The Hong Kong University of Science and Technology
Hong Kong, China
cwangch@cse.ust.hk

Zhuangda Wang
Xiamen Key Laboratory of Intelligent Storage and Computing, School of Informatics, Xiamen University
Xiamen, Fujian, China
wangrongxin@xmu.edu.cn

Rongxin Wu
Xiamen Key Laboratory of Intelligent Storage and Computing, School of Informatics, Xiamen University
Xiamen, Fujian, China
wangzhuangda@stu.xmu.edu.cn

Gang Fan
Ant Group
Shenzhen, Guangdong, China
fangang@antgroup.com

Abstract

Database Management Systems (DBMSs) are fundamental to numerous enterprise applications. Due to the significance of DBMSs, various testing techniques have been proposed to detect DBMS bugs. However, to trigger deep bugs, most of the existing techniques focus on generating lengthy and complex queries which burdens developers with the difficult of debugging. Therefore, SQL query simplification, which aims to reduce lengthy SQL queries without compromising their ability to detect bugs, is highly demanded.

To bridge this gap, we introduce SQLess, an innovative approach that employs a dialect-agnostic method for efficient and semantically correct SQL query simplification tailored for various DBMSs. Unlike previous works that have to depend on DBMS-specific grammar, SQLess utilizes an adaptive parser, which leverages error recovery and grammar extension to support DBMS dialects. Moreover, SQLess performs a semantics-sensitive SQL query trimming, which leverages alias and dependency analysis to simplify SQL queries with preserving bug-triggering capability.

We evaluate SQLess using two datasets from the state-of-the-art database bug detection studies, encompassing six widely-used DBMSs and over 32,000 complex SQL queries. The results demonstrate SQLess's superior performance: it achieves an average simplification rate of 72.45% in the *PINOLO Dataset*, which significantly outperforms the state-of-the-art approaches by 84.91%.

CCS Concepts

• **Software and its engineering** → *Software maintenance tools*.

Keywords

SQL Query Simplification, Bug Detection, Program Trimming

ACM Reference Format:

1 Introduction

Database Management Systems (DBMSs) are widely utilized for data storage in various software applications [4]. With hundreds of DBMSs developed, each featuring distinct characteristics and functionalities [16], the complexity and large-scale nature of these systems often lead to the presence of bugs.

ISSTA 2024