SIGMOD 2025

Constant Optimization Driven Database System Testing

CHI ZHANG, Nanjing University, China
MANUEL RIGGER, National University of Singapore, Singapore

Motivation

- **Logical Bug Detection Attracts Attention**
  - **NoREC and DQE:** Assume a given row must be included;
  - **TLP:** Decompose a query into three parts (p, not p, p is null);
  - **TQS:** Provide a test oracle for join operators using database schema normalization;
  - **PQS:** Random join hints should have identical result;
  - **EET:** Adding boolean expression.
- **Limitations:**
  - None of these approaches support testing subqueries, an important feature that allows query nesting.

```
--- Transformed query , result set: empty
SELECT t2.c0 FROM t2
WHERE (t2.c1 >= t2.c0) <> (t2.c5 = (
    SELECT t2.c4 AS c_0
    FROM (t1 AS ref_0 INNER JOIN t0 AS ref_1
        ON (ref_0.c0 = ref_1.c0))
    WHERE (CASE WHEN (((ref_0.c0 LIKE 'z~%')
              AND (NOT (ref_0.c0 LIKE 'z~%')))
              AND ((ref_0.c0 LIKE 'z~%') IS NOT NULL))
            THEN t2.c3 ELSE t2.c3 END) =
        (CASE WHEN (((ref_1.c0 NOT LIKE '_%%')
            AND (NOT (ref_1.c0 NOT LIKE '_%%')))
            AND ((ref_1.c0 NOT LIKE '_%%') IS NOT NULL))
        THEN t2.c4 ELSE t2.c2 END)
    ORDER BY c_0 DESC LIMIT 1));
```

Fig.1 EET Example

Background

Predicate

- A predicate is a Boolean expression that evaluates to TRUE, FALSE, or NULL when applied to given values or rows.
- Predicates are used in various clauses of SQL, such as the WHERE clauses of SELECT, UPDATE, and DELETE, as well as the JOIN ON, HAVING, GROUP BY, and ORDER BY clauses of SELECT.

Subqueries

- Correlated subqueries are SELECT queries nested within outer queries, referencing columns from the outer queries to construct their predicate.
- Non-correlated subqueries do not reference columns in the outer query and are evaluated once in execution.

非相关子查询：
获取所有员工中工资大于平均工资的员工信息

```
SELECT * FROM employees WHERE salary >
(SELECT AVG(salary) FROM employees);
```

Listing 2. A correlated subquery example.

```
CREATE TABLE t0(ID INT, score INT, classID INT);
INSERT INTO t0 VALUES (0, 90, 1), (1, 80, 1), (2, 83, 2);
Ⓞ SELECT x.ID FROM t0 AS x WHERE x.score >
    (SELECT AVG(y.score) FROM t0 AS y WHERE x.classID = y.classID);
Ⓐ SELECT x.classID,
    (SELECT AVG(y.score) FROM t0 AS y WHERE x.classID = y.classID) FROM t0 AS x;
Ⓕ SELECT x.ID FROM t0 AS x WHERE x.score >
    (CASE WHEN x.classID = 1 THEN 85
          WHEN x.classID = 1 THEN 85
          WHEN x.classID = 2 THEN 83 END);
```

Background

- **SQL CASE expression**
  - The CASE expression is a feature of SQL to process if / then logic.
- **Metamorphic testing**
  - Formally, given an input I and $P(I) = O$, where P is the program under test, a follow-up input I' is derived, so that a known relationship between O and $P(I') = O'$ is validated.
  - The core challenge is to identify a so-called metamorphic relation that derives the follow-up input I' and relates the outputs O and O'.

Listing 3. A CASE expression example.

```
CREATE TABLE grade (score INT);
INSERT INTO grade(score) VALUES (100), (80), (60);
SELECT score, CASE WHEN score = 100 THEN "A"
                   WHEN score >= 80 AND score < 100 THEN "B"
                   ELSE "C" END FROM grade;
```

Observations

- ## Constant folding
  - It is a well-known compiler optimization that evaluates constant expressions at compile time, rather than computing them at run time.
  - For example, it evaluates the statement i = 1 + 2 + 3; to i = 6; at compile time.
- ## Constant propagation
  - Through reachability analysis, it determines constant values for variables by assessing their reachability at specific program points.
  - For example, the statement sequence a = 1; i = a + 2 + 3; can be optimized to i = 6;.
- ## Key Insights
  - Within an SQL query, by assuming a constant database state and given query, we can apply constant folding and constant propagation to a specific expression in a predicate, assuming that the query’s result remains unchanged.
  - **Any discrepancy in the results of these two queries indicates a potential bug.**

Overview

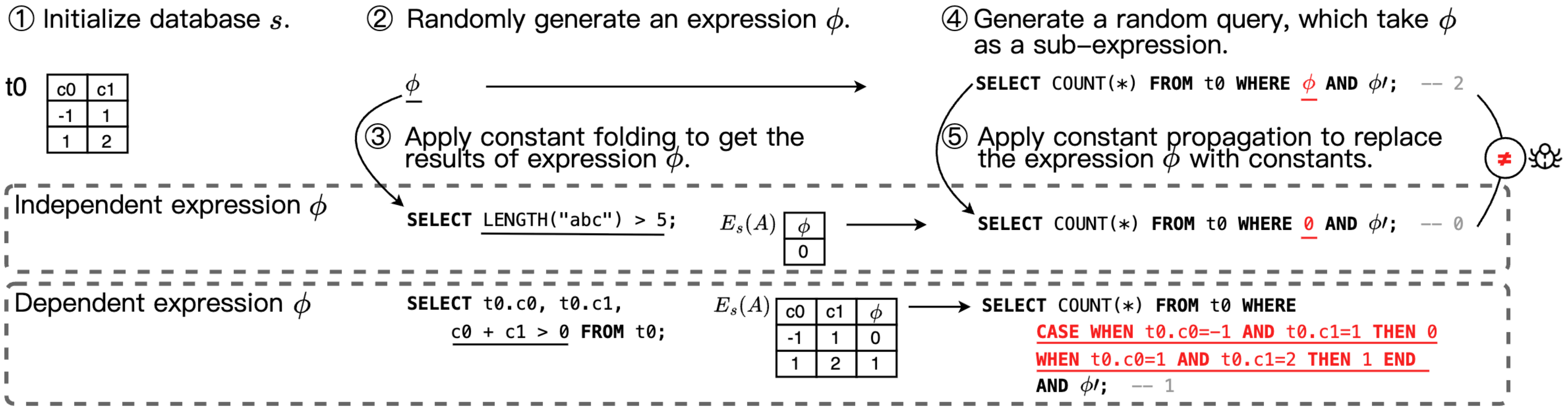

Fig. 1. Overview of approach. CODDTest generates pairs of equivalent queries by applying constant folding and propagation to the expression ϕ. The application of constant propagation and folding differs for independent and dependent expressions.

The **original** query includes the randomly generated predicate.

The **folded** query is derived by substituting the predicate in the original query with the corresponding constant-folded predicate.

Algorithm Sketch

Algorithm 1: Algorithm of CODDTest

function `TestOracleGen`(*DatabaseState* s)

- // Randomly generate an expression ϕ, which will undergo constant folding and constant propagation. We extract the set of the referenced columns $\{c_i\}$ in ϕ, which come from outer context, along with the tables set $\{t_i\}$ to which these columns c_i are associated.
- $\phi, \{c_i\}, \{t_i\} \leftarrow$ `GenExpr`(s)
- // Constant folding of ϕ under s, this step differs based on ϕ is dependent or independent expressions. Specifically, ϕ is considered an independent expression when $\{c_i\}$ is empty; otherwise, ϕ is classified as a dependent expression.
- **if** `Size`$(\{c_i\})$ == 0 **then**
  - // Construct the auxiliary query for independent expression
  - $A \leftarrow$ "SELECT " + ϕ
  - $E_s(A) \leftarrow$ `ExecQuery`(A, s)
  - // Transform the constant result of ϕ as a constant expression R_ϕ
  - $R_\phi \leftarrow E_s(A)$
- **else**
  - // Dependent expression has different result for each row of $\{c_i\}$
  - $A \leftarrow$ "SELECT " + $(\{c_i\}, \phi)$ + "FROM " + $\{t_i\}$
  - $E_s(A) \leftarrow$ `ExecQuery`(A, s)
  - // Map the results of ϕ to each row of $\{c_i\}$ as an expression R_ϕ
  - $R_\phi \leftarrow$ `Map`$(E_s(A))$
- // Generate the original query based on the current database state, using ϕ as a sub-expression in predicate
- $O \leftarrow$ `QueryGenerate`$(s, \phi, \{t_i\})$
- $E_s(O) \leftarrow$ `ExecQuery`(O, s)
- // Generate the folded query by replacing ϕ with R_ϕ
- $F \leftarrow$ `ReplaceExpr`(ϕ, R_ϕ, O)
- $E_s(F) \leftarrow$ `ExecQuery`(F, s)
- // A bug is identified if there is a discrepancy between the results of the original query and the folded query
- **if** $E_s(O) != E_s(F)$ **then**
  - `ReportBug`(O, F, A)

Folding Independent Expressions

- **Case1: The expression ϕ has no column references**
  - It is a constant expression that always yields a constant value.
  - For example, the independent expression LENGTH ("abc") > 5 shown is used in a SELECT statement to derive its results.
- **Case2: The expression ϕ is a non-correlated subquery**
  - It computes a constant result assuming a fixed database state.
  - The subquery of query returns the same result regardless of the outer query's result.

```
CREATE TABLE t0 (c0);
INSERT INTO t0(c0) VALUES (1);
CREATE INDEX i0 ON t0(c0 > 0);
CREATE VIEW v0(c0) AS SELECT AVG(t0.c0) FROM t0 GROUP BY 1 > t0.c0;
Ⓞ SELECT COUNT(*) FROM t0 INDEXED BY i0 WHERE (SELECT COUNT(*) FROM v0 WHERE v0.c0
   BETWEEN 0 AND 0); -- 1
Ⓐ SELECT COUNT(*) FROM v0 WHERE v0.c0 BETWEEN 0 AND 0; -- 0
Ⓕ SELECT COUNT(*) FROM t0 INDEXED BY i0 WHERE 0; -- 0 ✓
```

Folding Dependent Expressions

- **Constant folding:**
  - It first obtains the results of the expression on each row (i.e., step 3 for dependent expression).
  - It then represents them using a mapping (i.e., step 5 for dependent expression).
- **Supporting Join:**
  - The auxiliary queries must use the same JOIN clauses as the original query, except in cases where ϕ serves as the predicate within the JOIN clause.
  - Although it generates non-empty tables, an empty result can still occur, for example, when using an INNER JOIN with a false predicate. In such scenarios, it discards the test.

Listing 4. **JOIN** can affect the values of ϕ for a given row.

```
CREATE TABLE t0 (c0 INT);
CREATE TABLE t1 (c0 INT);
INSERT INTO t0 VALUES (0);
INSERT INTO t1 VALUES (1);
Ⓞ SELECT * FROM t0 LEFT JOIN t1 ON t0.c0 = t1.c0 WHERE t1.c0 IS NULL; -- 0|NULL
Ⓐ SELECT t1.c0, t1.c0 IS NULL FROM t0 LEFT JOIN t1 ON t0.c0 = t1.c0; --NULL|1
Ⓕ SELECT * FROM t0 LEFT JOIN t1 ON t0.c0 = t1.c0 WHERE
    CASE WHEN t1.c0 is NULL THEN 1 END; --0|NULL
```

Construction of Original Query

- **Predicate construction**
  - It randomly generate predicates that contain or correspond to ϕ based on SQLancer.
  - Subqueries can evaluate to three different result types: (1) a scalar value, which is a single value; (2) a row value, which is an ordered list of two or more scalar values; (3) multiple row values.
- **Query construction**
  - It supports placing these predicates not only in the WHERE, JOIN, HAVING, GROUP BY, and ORDER BY clauses of SELECT, but also in other statements that require predicates, such as CREATE INDEX, CREATE VIEW, UPDATE, INSERT, and DELETE.

Listing 5. The subquery, when used as the fetch keyword in a **SELECT**, must return only one column and one row.

```
CREATE TABLE t0(c0 INT);
CREATE TABLE t1(c0 INT);
INSERT INTO t0 VALUES (1);
INSERT INTO t0 VALUES (2), (3);
SELECT t0.c0, (SELECT t1.c0 FROM t1 WHERE t1.c0 > t0.c0) FROM t0;
    -- Error: Subquery returns more than 1 row
```

Experiment

- **Target DBMS:** SQLite, MySQL, CockroachDB, DuckDB, and TiDB.
- **Baselines:** NoREC, TLP, DQE, and EET.
- **Environment:** A server with a 64-Core AMD EPYC 7763 Processor at 2.45GHz and 512GB of memory running Ubuntu 22.04.

Bug Number

Table 1. CODDTest found 45 unique bugs in five mature DBMSs.

DBMS	Bug type				Bug status	
	Logic bug	Internal error	Crash	Hang	Fixed	Verified
SQLite	6	1	0	0	7	0
MySQL	1	1	0	0	0	2
CockroachDB	7	4	0	2	11	2
DuckDB	5	2	2	3	12	0
TiDB	5	6	0	0	3	8
Total	24	14	2	5	33	12

Table 2. The number of detectable bugs by test oracles.

Oracles	NoREC	TLP	DQE	Only by CODDTest
Num	11	12	4	11

Efficiency

Table 3. The number of tests conducted by each approach.

Oracle	# of tests	# of successful queries	# of unsuccessful queries	QPT	# of unique query plans	branch coverage
NoREC	2,086,646k	4,207,286k	149,036k	2.05	172,808	63.18%
TLP	976,216k	2,180,736k	398,919k	2.23	137,743	63.63%
DQE	441,350k	7,502,402k	21,997k	17.00	486	46.71%
CODDTest	497,092k	1,655,518k	53,102k	3.33	2,577,603	63.06%
CODDTest & Expression	1,423,068k	4,411,510k	326,849k	3.10	7,399	63.23%
CODDTest & Subquery	423,310k	1,488,817k	47,141k	3.51	2,755,619	62.19%

24-hour Test Over SQLite

Expression Complexity

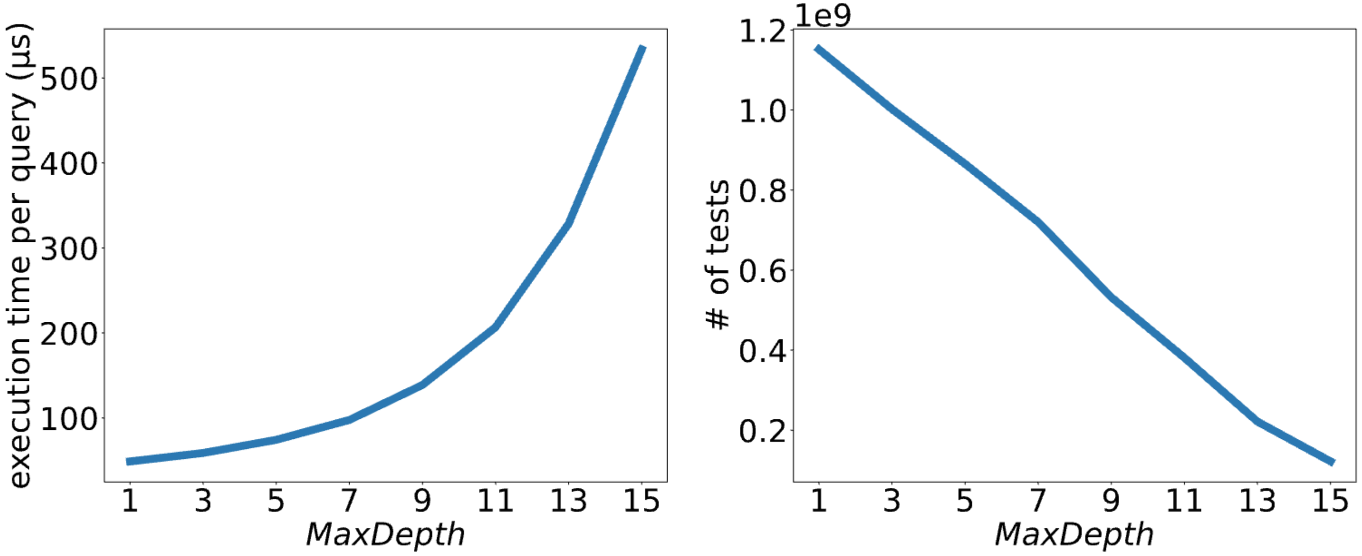

Fig. 2. The impact of expression complexity on query execution time and test throughput.

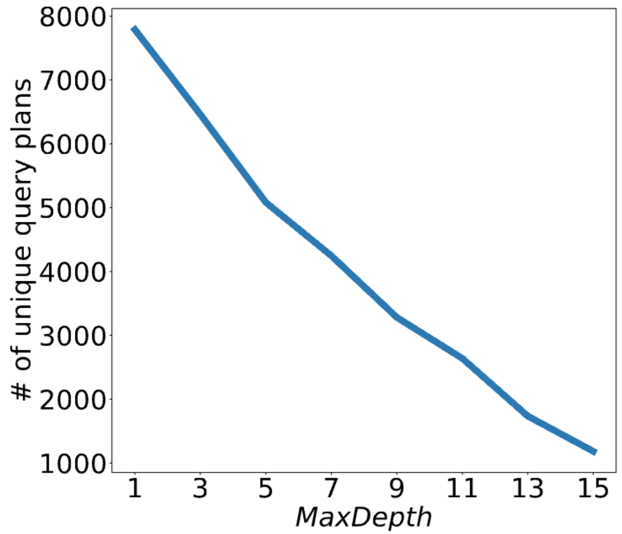

Fig. 3. The impact of expression complexity on unique query plans.

Case Study

Listing 7. A bug found in CockroachDB caused by a false predicate being always evaluated to true.

```
CREATE TABLE t1 (v VARBIT);
INSERT INTO t1 VALUES (B'11');
Ⓞ WITH t2 AS (SELECT NULL AS b) SELECT t1.v FROM t1, t2 WHERE t1.v NOT BETWEEN
    t1.v AND (CASE WHEN NULL THEN t2.b ELSE t1.v END); -- empty result ✓
Ⓐ SELECT NULL AS b; -- NULL
CREATE TABLE t2 (b BIT);
INSERT INTO t2 VALUES (NULL);
Ⓕ SELECT t1.v FROM t1, t2 WHERE t1.v NOT BETWEEN t1.v AND
    (CASE WHEN NULL THEN t2.b ELSE t1.v END); -- '11'
```

Listing 8. A bug found in SQLite related to **JOIN**.

```
CREATE TABLE vt0(c2);
CREATE TABLE t1(c0 TEXT);
INSERT INTO t1(c0) VALUES (1);
INSERT INTO vt0(c2) VALUES (-1);
CREATE VIEW v0(c0) AS SELECT 0 FROM t1;
Ⓞ SELECT vt0.c2 AS c1 FROM t1 CROSS JOIN v0 ON (
    EXISTS (SELECT v0.c0 FROM v0 WHERE false)) FULL OUTER JOIN vt0 ON 1; -- -1 ✓
Ⓐ SELECT v0.c0 FROM v0 WHERE false;-- empty result
Ⓕ SELECT vt0.c2 AS c1 FROM t1 CROSS JOIN v0 ON (0) FULL OUTER JOIN vt0 ON 1;
    -- empty result
```

Thank You Guys!