Keep It Simple: Testing Databases via Differential Query Plans

Jinsheng Ba, Manuel Rigger
National University of Singapore

NUS National University of Singapore

TEST Trustworthy Engineering of Software Technologies Lab

Background: Logic Bugs

Logic bugs refer to incorrect results returned by DBMSs.

t0

c0
2

t1

c0
0

```sql
SELECT COUNT(*)
FROM t0 INNER JOIN
t1 ON t0.c0==t1.c0;
```

→ 0 ✓

→ 1 🐞

Study

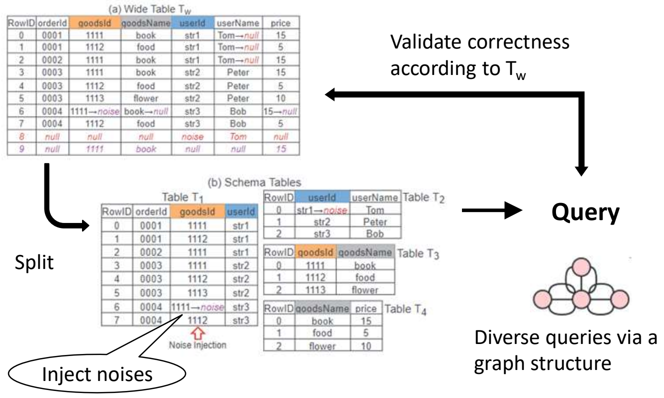
We found that most bugs found by TQS can be reproduced by comparing the execution results of the same query of different query plans.

```sql
SELECT t0.c0 FROM t0 WHERE t0.c0 IN
 (SELECT t0.c0 FROM t0 WHERE
  (t0.c0 NOT IN
   (SELECT t0.c0 FROM t0 WHERE t0.c0 )
  ) = (t0.c0)
 ); -- {0000001985} ,{0000001996}
SELECT t0.c0 FROM t0 WHERE t0.c0 IN
 (SELECT /*+ no_semijoin()*/ t0.c0 FROM t0 WHERE
  (t0.c0 NOT IN
   (SELECT t0.c0 FROM t0 WHERE t0.c0 )
  ) = (t0.c0)
 ); -- empty set ✗
```

Problem and Challenges

How to automatically find logic bugs?

A state-of-the-art work: Transformed Query Synthesis (TQS)



(a) Wide Table T_W

RowID	orderId	goodsId	goodsName	userId	userName	price
0	0001	1111	book	str1	Tom→*null*	15
1	0001	1112	food	str1	Tom→*null*	5
2	0002	1111	book	str1	Tom→*null*	15
3	0003	1111	book	str2	Peter	15
4	0003	1112	food	str2	Peter	5
5	0003	1113	flower	str2	Peter	10
6	0004	1111→*noise*	book→*null*	str3	Bob	15→*null*
7	0004	1112	food	str3	Bob	5
8	*null*	*null*	*null*	*noise*	*Tom*	*null*
9	*null*	*1111*	*book*	*null*	*null*	*15*

Validate correctness according to T_w

Split

(b) Schema Tables

Table T_1

RowID	orderId	goodsId	userId
0	0001	1111	str1
1	0001	1112	str1
2	0002	1111	str1
3	0003	1111	str2
4	0003	1112	str2
5	0003	1113	str2
6	0004	1111→*noise*	str3
7	0004	1112	str3

Table T_2

RowID	userId	userName
0	str1→*noise*	Tom
1	str2	Peter
2	str3	Bob

Table T_3

RowID	goodsId	goodsName
0	1111	book
1	1112	food
2	1113	flower

Table T_4

RowID	goodsName	price
0	book	15
1	food	5
2	flower	10

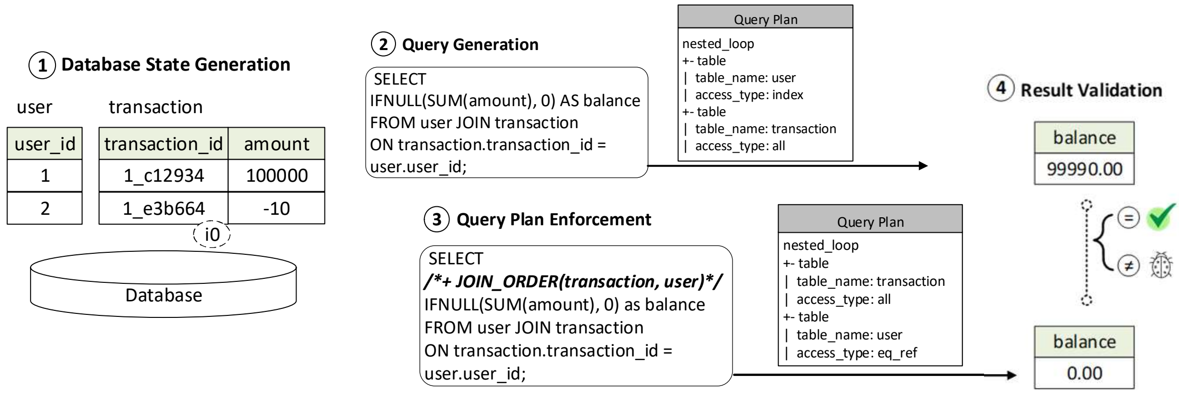
Diverse queries via a graph structure

Challenges:

1. TQS is a **sophisticated** method that requires splitting tables and retrieving results from the first table.
2. This method can **only be applied** to equijoin.

Method: Differential Query Plans (DQP)

Comparing query plan executions is a simple testing method, and we demonstrated that such a simple method can be as effective as TQS.
We propose the method **Differential Query Plans (DQP)**:



① Database State Generation

user

user_id
1
2

transaction

transaction_id	amount
1_c12934	100000
1_e3b664	-10

② Query Generation

```
SELECT
IFNULL(SUM(amount), 0) AS balance
FROM user JOIN transaction
ON transaction.transaction_id =
user.user_id;
```

Query Plan

```
nested_loop
+- table
|  table_name: user
|  access_type: index
+- table
|  table_name: transaction
|  access_type: all
```

③ Query Plan Enforcement

```
SELECT
/*+ JOIN_ORDER(transaction, user)*/
IFNULL(SUM(amount), 0) as balance
FROM user JOIN transaction
ON transaction.transaction_id =
user.user_id;
```

Query Plan

```
nested_loop
+- table
|  table_name: transaction
|  access_type: all
+- table
|  table_name: user
|  access_type: eq_ref
```

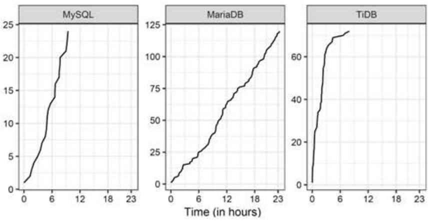
④ Result Validation

balance
99990.00

balance
0.00

Results

In MySQL, MariaDB, and TiDB, DQP can reproduce 14 of 15 bugs found by TQS. Additionally, DQP found 26 new bugs.



Number of bug-inducing test cases found by DQP in 24 hours.

Conclusion

1. DQP is a **general black-box** method, which is easy to understand.
2. DQP is **lightweight** as it was implemented in less than 100 lines of Java code.
3. DQP is **applicable** as at least 8 of 10 most popular relational DBMSs support enforcing query plans.
4. DQP is **a simple alternative** to TQS that achieves the same level of effectiveness.

Paper

Code

Session 35: Security (3)
Thursday June 13
5:15 pm – 5:30 pm
Location: Puyehue/Calbuco

Author: Jinsheng Ba

Author: Manuel Rigger