Keep It Simple: Testing Databases via Differential Query Plans

Jinsheng Ba, Manuel Rigger

National University of Singapore

Logic Bugs

user

user_id
1
2

transaction

transaction_id	amount
1_c12934	100000
1_e3b664	-10

i0

Logic bugs refer to incorrect results returned by DBMSs.

Checking the balance of user 1:

```
SELECT /*+ JOIN_ORDER(transaction, user)*/ IFNULL(SUM(amount), 0) as balance
FROM user JOIN transaction ON transaction.transitition_id = user.user_id
WHERE user.user_id=1; -- 99990.00 ✓ 0.00 ✗
```

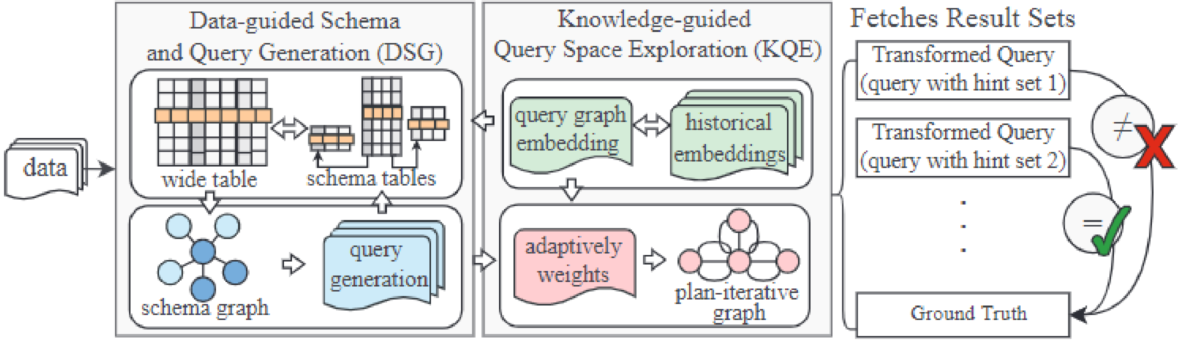
Problem: how do we automatically find logic bugs?

Challenge

- Unlike crash bugs, which terminate DBMSs, logic bugs silently compute incorrect results?
- We need a test oracle, which can tell us whether the results are correct.

Transformed Query Synthesis (TQS)

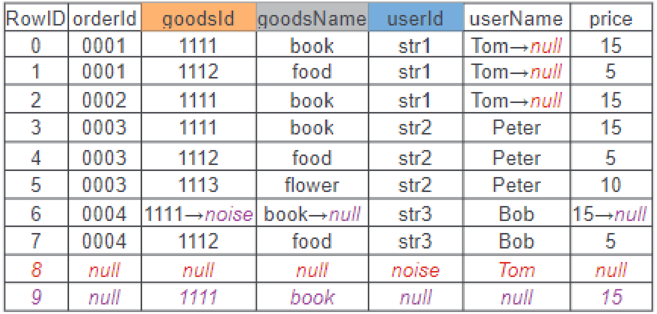
- TQS* is the state-of-the-art approach to realize a test oracle.
- However,
  - 1) TQS is sophisticated,
  - 2) TQS can only be applied to equijoins.


data
Data-guided Schema and Query Generation (DSG)
wide table
schema tables
schema graph
query generation
Knowledge-guided Query Space Exploration (KQE)
query graph embedding
historical embeddings
adaptively weights
plan-iterative graph
Fetches Result Sets
Transformed Query (query with hint set 1)
Transformed Query (query with hint set 2)
≠
=
Ground Truth

* Xiu Tang, Sai Wu, Dongxiang Zhang, Feifei Li, and Gang Chen. 2023. Detecting Logic Bugs of Join Optimizations in DBMS. Proc. ACM Manag. Data 1, 1, Article 55.

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

Result

Split

(b) Schema Tables

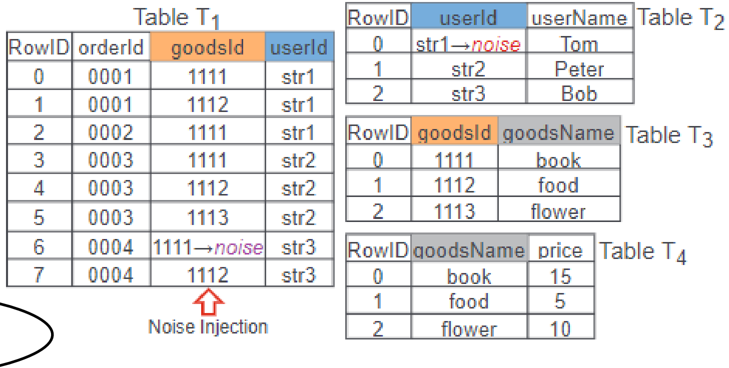


Table T_1

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

Noise Injection

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

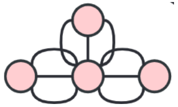
Table T_4

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

Inject noises

Query

T1 Join T2 Join T3



Diverse queries via a graph structure

TQS Study

- To understand how TQS finds bugs, we studied their bug reports.

```
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
```

The bugs found by TQS can be found by comparing query plan executions.

***https://bugs.mysql.com/bug.php?id=106713**

TQS Study

- We found 21 bug reports from TQS public bug list*.
- 20 confirmed bugs.
- 15 unique bugs.
- 10 join-related unique bugs.
- 14 unique bugs can be found by comparing query plan executions.

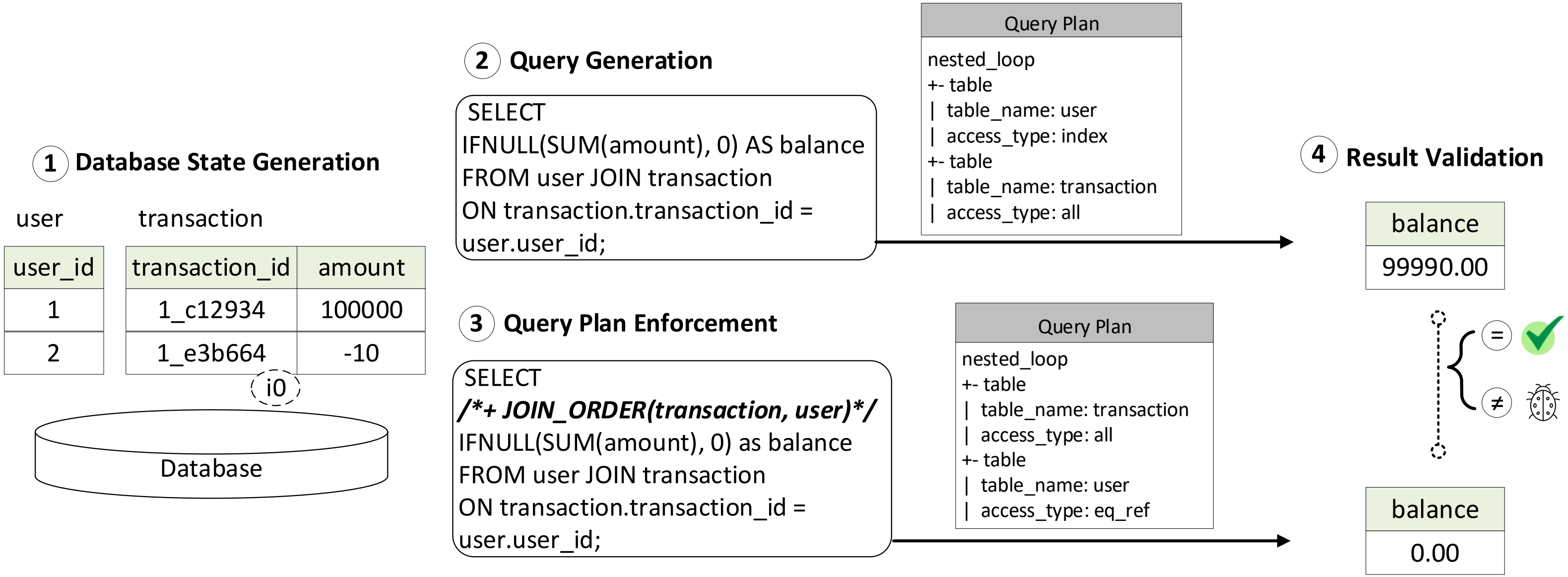
DBMS	Bug	Unique	Join	Query Plan
MySQL	106713	✓		✓
MySQL	106715	✓	✓	✓
MySQL	106716	✓	✓	✓
MySQL	106717	✓		✓
MySQL	106718	✓		✓
MySQL	106611			✓
MySQL	106710	✓		✓
MySQL	99273	✓		
MySQL	109211	✓	✓	✓
MySQL	109212	✓	✓	✓
MariaDB	28214	✓	✓	✓
MariaDB	28215	✓	✓	✓
MariaDB	28216	✓	✓	✓
MariaDB	28217	✓	✓	✓
MariaDB	29695	✓	✓	✓
TiDB	33039		✓	✓
TiDB	33041		✓	✓
TiDB	33042	✓	✓	✓
TiDB	33045		✓	✓
TiDB	33046		✓	✓

***https://github.com/xiutangzju/tqs/blob/d5f8f5/index.md**

Goal

Comparing query plan executions is a simple testing method, and we aim to answer whether such a simple method can be as effective as TQS.

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

Query Plan Enforcement

- Query hints and system variables are two major ways.
- Query hints: a comment-like clause in a query and can affect the behaviors of the query optimizer.

```
SELECT /*+ JOIN_ORDER(t1, t2) */ * FROM t1 INNER JOIN t2 ON
t1.c0 = t2.c0; -- enforce the join order
SELECT /*+ HASH_JOIN(t1) */ * FROM t1 INNER JOIN t2 ON t1.c0 =
t2.c0; -- enforce using hash join algorithm for joining t1
SELECT /*+ INDEX(i1) */ * FROM t1 INNER JOIN t2 ON t1.c0 =
t2.c0; -- enforce using index i1 when accessing data
```

***https://dev.mysql.com/doc/refman/8.0/en/optimizer-hints.html**

Query Plan Enforcement

- Query hints and system variables are two major ways.
- System variables: specific variables that affect the executions of subsequent queries.

```
SET optimizer_switch='block_nested_loop=on'; --enable hash join
SET optimizer_switch='index_condition_pushdown=ff'; --disable
condition pushdown optimization for indexes

SELECT * FROM t1 INNER JOIN t2 ON t1.c0 = t2.c0;
…
```

***https://dev.mysql.com/doc/refman/8.0/en/switchable-optimizations.html**

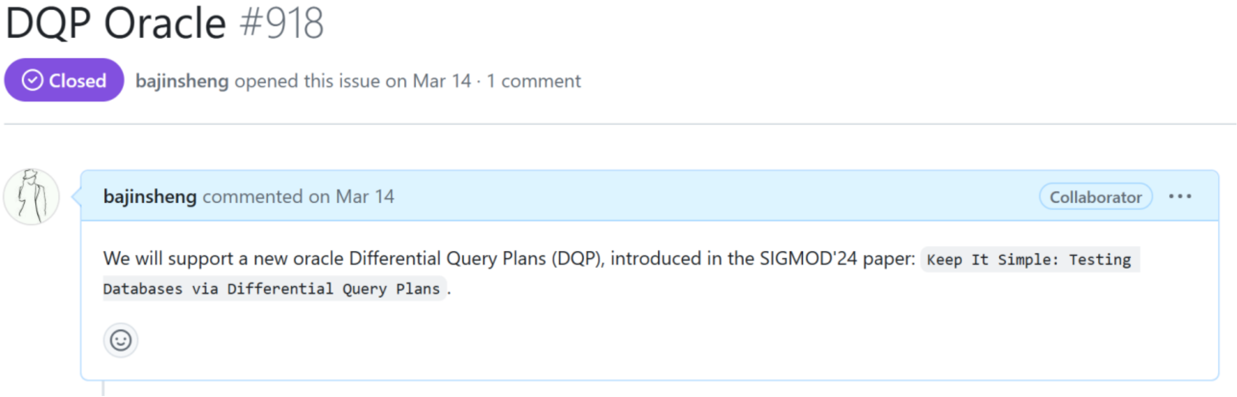
Query Plan Enforcement

- Query hints and system variables are two major ways.

DBMS	Query Hints	System Variables
MySQL	32	26
MariaDB		37
TiDB	22	

Implementation

- We implemented DQP in SQLancer, a DBMS testing framework that randomly generates databases and queries complying with the SQL grammar.
- Code is open-sourced.



DQP Oracle #918

bajinsheng opened this issue on Mar 14 · 1 comment

bajinsheng commented on Mar 14

We will support a new oracle Differential Query Plans (DQP), introduced in the SIGMOD'24 paper: `Keep It Simple: Testing Databases via Differential Query Plans`.

https://github.com/sqlancer/sqlancer/issues/918

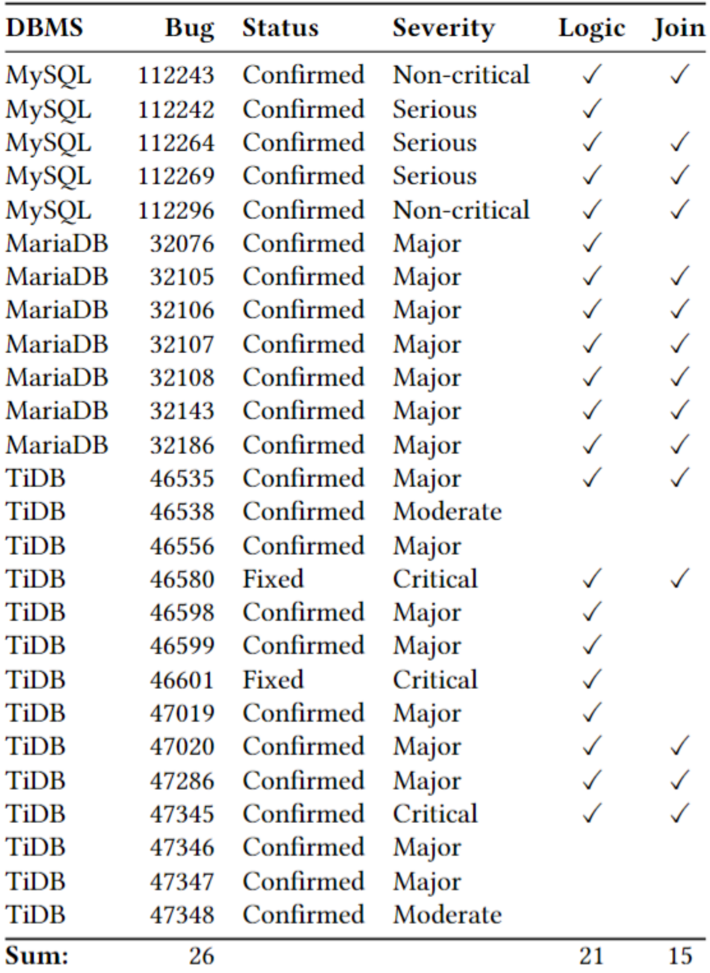
Evaluation

- 14 of 15 unique bugs found by TQS can be reproduced by our method DQP.

DBMS	Bug	Unique	Join	Query Plan
MySQL	106713	√		√
MySQL	106715	√	√	√
MySQL	106716	√	√	√
MySQL	106717	√		√
MySQL	106718	√		√
MySQL	106611			√
MySQL	106710	√		√
MySQL	99273	√		
MySQL	109211	√	√	√
MySQL	109212	√	√	√
MariaDB	28214	√	√	√
MariaDB	28215	√	√	√
MariaDB	28216	√	√	√
MariaDB	28217	√	√	√
MariaDB	29695	√	√	√
TiDB	33039		√	√
TiDB	33041		√	√
TiDB	33042	√	√	√
TiDB	33045		√	√
TiDB	33046		√	√

Evaluation

- DQP additionally found 26 previously unknown and unique bugs.
- 21 logic bugs, and 15 logic bugs are related to JOIN.



DBMS	Bug	Status	Severity	Logic	Join
MySQL	112243	Confirmed	Non-critical	✓	✓
MySQL	112242	Confirmed	Serious	✓	
MySQL	112264	Confirmed	Serious	✓	✓
MySQL	112269	Confirmed	Serious	✓	✓
MySQL	112296	Confirmed	Non-critical	✓	✓
MariaDB	32076	Confirmed	Major	✓	
MariaDB	32105	Confirmed	Major	✓	✓
MariaDB	32106	Confirmed	Major	✓	✓
MariaDB	32107	Confirmed	Major	✓	✓
MariaDB	32108	Confirmed	Major	✓	✓
MariaDB	32143	Confirmed	Major	✓	✓
MariaDB	32186	Confirmed	Major	✓	✓
TiDB	46535	Confirmed	Major	✓	✓
TiDB	46538	Confirmed	Moderate		
TiDB	46556	Confirmed	Major		
TiDB	46580	Fixed	Critical	✓	✓
TiDB	46598	Confirmed	Major	✓	
TiDB	46599	Confirmed	Major	✓	
TiDB	46601	Fixed	Critical	✓	
TiDB	47019	Confirmed	Major	✓	
TiDB	47020	Confirmed	Major	✓	✓
TiDB	47286	Confirmed	Major	✓	✓
TiDB	47345	Confirmed	Critical	✓	✓
TiDB	47346	Confirmed	Major		
TiDB	47347	Confirmed	Major		
TiDB	47348	Confirmed	Moderate		
Sum:	26			21	15

Other Evaluation Results in the Paper

- Bug-finding efficiency: DQP found 216 bug-inducing test cases in 24 hours in MySQL, MariaDB, and TiDB.
- Bug-finding Effectiveness: NoREC and TLP cannot find 17 of 21 logic bugs found by DQP.
- Query plan coverage, Join coverage, code coverage.
- …

Discussion

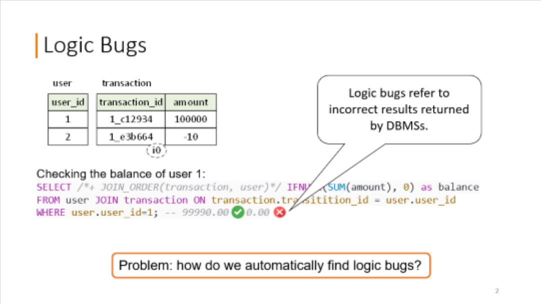
- DQP is a general black-box method, which is easy to understand.
- DQP is lightweight as it was implemented in less than 100 lines of Java code.
- DQP is applicable as at least 8 of 10 most popular relational DBMSs* support query hints and system variables.

***https://db-engines.com/en/ranking/relational+dbms**

Conclusion

Paper

Code



Logic Bugs

user
user_id
1
2

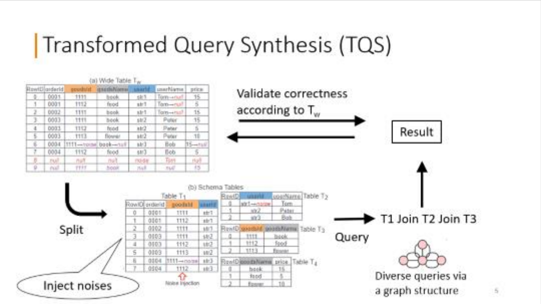
transaction	
transaction_id	amount
1_c12934	100000
1_e3b664	-10

Logic bugs refer to incorrect results returned by DBMSs.

Checking the balance of user 1:

```
SELECT /*+ JOIN_ORDER(transaction, user)*/ IFNULL(SUM(amount), 0) as balance
FROM user JOIN transaction ON transaction.transitition_id = user.user_id
WHERE user.user_id=1; -- 99990.00 ✓ 0.00 ✗
```

Problem: how do we automatically find logic bugs?



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

Result

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

Noise Injection

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

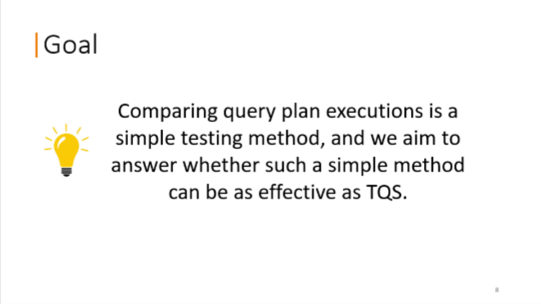
RowID	goodsName	price
0	book	15
1	food	5
2	flower	10

Inject noises

Query

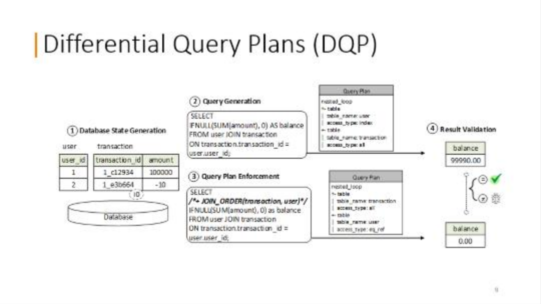
T1 Join T2 Join T3

Diverse queries via a graph structure



Goal

Comparing query plan executions is a simple testing method, and we aim to answer whether such a simple method can be as effective as TQS.



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

Simple testing methods, such as DQP, are practical, and might outperform more conceptually appealing ones such as TQS.