Postprint of paper in *Proceedings of the IEEE 47th Annual International Computers, Software, and Applications Conference Workshops* (*COMPSACW '23*), IEEE, Piscataway, NJ, USA (2023)

Detecting Hidden Failures of DBMS: A Comprehensive Metamorphic Relation Output Patterns Approach

Matthew Siu-Hin Tang
Department of Computer Science
The Univeristy of Hong Kong
Pokulam, Hong Kong
tshcat@connect.hku.hk

T. H. Tse*
Department of Computer Science
The University of Hong Kong
Pokulam, Hong Kong
thtse@cs.hku.hk

Zhi Quan Zhou
School of Computing and IT, University of Wollongong
Wollongong NSW 2522, Australia, and
Alibaba and Ant Group, Hangzhou, China
zhiquan@uow.edu.au

***Abstract*—The testing of large databases faces the test oracle problem, namely, that it is difficult to verify execution results against expected outcomes. Rigger and Su applied metamorphic testing through query partitioning and ternary logic partitioning techniques to alleviate the challenge. In Part (A) of our project, we conduct an in-depth investigation and have identified a gap between the two techniques. We propose a disjoint partitioning approach to address it. In Part (B), we conduct a comprehensive investigation into the metamorphic testing of DBMS by comparing disjoint partitioning with metamorphic relation output patterns (MROPs) by Segura et al. We propose an exhaustive collection of MROPs for DBMS. To the best of our knowledge, this is the first project to integrate in-depth and comprehensive approaches to tackle the diverse challenges in DBMS testing. In Part (C), we conduct an empirical case study of their applications to OceanBase, the DBMS associated with the world's fastest online transaction processing system. Although OceanBase has been extensively tested and widely used in the industry, we have detected 12 hidden failures and 8 new crashes.**

***Index Terms*—test oracle, metamorphic testing, metamorphic relation output pattern, DBMS, SQL, OceanBase**

I. Introduction

Because of the popularity of online transaction processing (OLTP) systems [1] in the financial sector, the correctness of the supporting database management system (DBMS) is crucial. However, owing to the scale of large databases, the testing of DBMS is challenging [2]. It faces the test oracle problem, which refers to the difficulty in verifying system execution results [3][4]. The metamorphic testing (MT) methodology [5][6][7] was invented in 1998 to alleviate the problem. In 2020, Rigger and Su [8] applied MT to address the issue in DBMS testing through the query partitioning (QP) and ternary logic partitioning (TLP) techniques. Empirical studies showed that they revealed 175 failures in five DBMS.

Our current project is divided into three parts. In Part (A), we conduct an in-depth investigation on [8]. We find a gap between QP and TLP, and introduce the concept of disjoint partitioning (DP) to tackle the issue. In Part (B), we conduct a comprehensive investigation of the adequacy of the metamorphic relations constructed via DP, comparing them with the metamorphic relation output patterns (MROPs) proposed by Segura et al. [9]. We propose a more comprehensive approach for classifying and constructing metamorphic relations in DBMS testing. In Part (C) of the project, we perform an empirical case study on OceanBase [10], which has been developed by the Alibaba and Ant Group and associated with the world's fastest OLTP [11]. We apply both the in-depth and comprehensive approaches to test OceanBase, and have detected 12 hidden failures and 8 new crashes.

* Corresponding author.

II. Part (A): In-depth Investigation into Metamorphic Testing of DBMS

A. Motivation Example

Consider the staff table on the right. Let us write a SELECT statement in SQL:

id	name	salary
1	Alice	5000
2	Charlie	2000
3	Bob	3000

```
SELECT * FROM staff WHERE salary < 5000;
```

Suppose the DBMS returns the result on the right. It reveals a failure because Bob also has a salary lower than 5000 and is missing from the list.

id	name	salary
2	Charlie	2000

Imagine that the staff table contains 10 000 records instead of only three. We execute the same SQL SELECT statement against the DBMS to list all the staff with a salary lower than 5000. Suppose the DBMS returns a list of 3000 staff. This time, we cannot easily tell whether the result of 3000 staff misses any legitimate record or contains any superfluous record. Owing to the large volume of data, it is difficult to detect failures.

B. Metamorphic Testing

In software testing, a *test oracle* is the mechanism to verify the execution result against the expected outcome [12]. The *test oracle problem* refers to the situation where such a mechanism is either missing or extremely difficult to apply [3][4]. As we have observed in the motivating example, it would be challenging to verify the result of a given query in large databases.

In 1998, T. Y. Chen invented the *metamorphic testing* (*MT*) methodology [5][6][7], which supports test case generation and alleviates the test oracle problem. He defines *metamorphic*

relations (*MRs*) as necessary properties of the target function or program in relation to **multiple** inputs and their expected outputs. To conduct MT, some program inputs are first constructed as original test cases (called *source* test cases). On the basis of a target MR, new inputs are constructed as *follow-up* test cases. Contrary to traditional testing, which verifies the correctness of each individual test result, MT verifies the relation among the source and follow-up inputs and outputs with respect to the MR.

About 500 papers on MT have been published. However, research work on the application of MT to DBMS has been very limited [8][13][14].

When applying MT to DBMS testing, a source test case may be an initial query whereas a follow-up test case may be a subsequent query constructed according to a target MR. Then, the relationship between their outputs (resultant lists returned by the DBMS) can be verified with reference to the MR. In the case of the staff table example, we may specify an MR such that "if we add one more condition to the query, the DBMS should return fewer records." For instance, we may refine our query to ask for all the staff with salary < 5000 AND name = 'Alice'. On execution, if the DBMS returns 4000 staff, which is more than the previous result, we say that there is a violation of the MR. It indicates a failure of the DBMS. In this way, failures are revealed using MT without the need for a test oracle.

C. Revisit of QP and TLP by Rigger and Su

Rigger and Su [8] proposed query partitioning (QP) and ternary logic partitioning (TLP) for metamorphic testing of DBMS. QP is a general strategic concept that describes an MR among DBMS queries. Outputs of the follow-up queries are non-overlapping sublists of the output of the source query. The MR specifies that the concatenation of the outputs of the follow-up queries must be the same as the output of the source query. A violation of the MR indicates a failure.

TLP is a special case of QP. An original query is transformed into three partitioning queries, which are three predicate variants derived from a randomly generated predicate. Let us consider the staff table again, with a source query to return all the staff. We generate a random predicate name = 'Alice'. We derive three partitioning queries with three predicate variants: name = 'Alice', NOT (name = 'Alice'), and name = 'Alice' IS NULL. Since SQL is based on ternary Boolean logic [14], we know that name = 'Alice' can be TRUE, FALSE, or NULL, falling into the resultant lists of the three partitioning queries. Hence, following the MR stated in QP, by concatenating the three resultant lists, we should obtain the result of the original query, namely, all the staff in the table. TLP has revealed 175 failures in five DBMS, including SQLite, MySQL, CockroachDB, TiDB, and DuckDB.

D. Extension of QP and TLP to Disjoint Partitioning[1]

Our thorough investigation reveals that the high-level QP is too general for practical MR construction, while the low-level TLP is too specific. We propose an innovative technique called *disjoint partitioning* (*DP*) to fill the gap. The core idea is that resultant lists returned by the DBMS can be divided into **exhaustive and mutually exclusive** partitions using, for instance, the SQL keywords LIMIT and OFFSET. For example, given a staff table with 100 rows, we can write two queries

(Q0.1) SELECT * FROM staff LIMIT 50 OFFSET 0;

(Q0.2) SELECT * FROM staff LIMIT 50 OFFSET 50;

to return two resultant sublists, each containing the first and last 50 staff in the table, respectively.

Consider the entry_exit table below, recording the id, name, and entry_exit_count of 10 000 staff in a large corporation:

id	name	entry_exit_count
1	Alice	4
...	...	...
10000	Dave	3

By examining the entry_exit_count for each staff, we can determine that those with odd values are inside the building, whereas those with even values are outside.

We can use the SQL bitwise and (&) operator with a constant operand 1 to determine whether an entry_exit_count is odd or even. An odd number & 1 returns an integer 1, and an even number & 1 returns an integer 0.

Hence, we can write the following statement in OceanBase DBMS to find all the staff (whose entry_exit_count is an odd number) *inside* the building:

```
(Q1.1) SELECT * FROM entry_exit
       WHERE entry_exit_count & 1;
```

OceanBase returns a list with 3000 staff. However, we do not know whether the list of 3000 records is correct, because we do not have a test oracle.

In DP, we propose follow-up queries such that they partition list of rows in the original table into disjoint portions and apply the same WHERE condition. We write the two follow-up queries as follows:

```
(Q1.2) SELECT * FROM (SELECT * FROM entry_exit
       LIMIT 5000 OFFSET 0) AS offset1
       WHERE entry_exit_count & 1;

(Q1.3) SELECT * FROM (SELECT * FROM entry_exit
       LIMIT 5000 OFFSET 5000) AS offset2
       WHERE entry_exit_count & 1;
```

Query (Q1.2) retrieves the staff with an odd entry_exit_count in rows 1 to 5000, whereas query (Q1.3) retrieves those in rows 5001 to 10 000. OceanBase returns 1500 and 1501 staff, respectively.

The MR states that simple **concatenation** of the outputs of follow-up queries (Q1.2) and (Q1.3) should produce the same result as the original query (Q1.1). The MR is therefore violated because there are more rows in the concatenated output from (Q1.2) and (Q1.3). Thus, we have revealed a hidden failure.

III. Part (B): Comprehensive Investigation into Metamorphic Testing of DBMS

Although our previous extension [16] of Rigger and Su's work to DP reveals failures, it only tests for one kind of MR. In order to identify more MRs in DBMS, we draw our inspiration from existing MR frameworks from the literature.

[1] This part of the project was published in [16].

A. Revisit of MROPs

Metamorphic relation output patterns (*MROPs*) are a framework proposed by Segura et al. with the aim to capture the shape of typical MRs [9]. They identified six MROPs:

1) *Equivalence:* Relations where the source and follow-up outputs include the same items, not necessarily in the same order.
2) *Equality:* Relations where the source and follow-up outputs contain the same items in the same order.
3) *Subset:* Relations where follow-up outputs are subsets of the source output.
4) *Disjoint:* Relations where the source and follow-up outputs have no elements in common.
5) *Complete:* Relations where the union of the follow-up outputs should contain the same items as the source output.
6) *Difference:* Relations where the source and follow-up outputs should differ in a specific set of items.

B. Extension of MROPs to Comprehensive Metamorphic Testing of DBMS

Let us review whether the MROP framework in [9] is sufficiently comprehensive for DBMS testing. We have identified four output attributes of DBMS: ordered, partitioned, complete, and disjoint. These output attributes are *not* mutually exclusive, and are achieved via DBMS queries as follows:

1) *Ordered results* are obtained by explicitly specifying an ORDER BY clause with respect to specific column(s). Unordered results are achieved by *not* specifying any ORDER BY clause.
2) *Partitioned results* are obtained by specifying a constraint such that only specific rows of the table are returned. A non-partitioned result is obtained by *not* specifying any constraint that restricts the records returned.
3) *Complete results* are obtained by specifying a query or multiple queries such that their results exhaustively cover all possible rows of the database table. Non-complete results do not exhaustively cover the database table.
4) *Disjoint results* are obtained by specifying multiple queries such that their results do not overlap with one another. Note that disjoint results are not necessarily complete, and complete results are not necessarily disjoint.

We have studied the correlation between the original MROP framework and the four output attributes, as summarized in Table 1. We find that the former is not sufficiently comprehensive to cover all possible output attributes of DBMS.

We also find that, using sets and subsets to categorize output patterns in DBMS, we may overlook hidden failures because repeated elements are not considered. Consider the output list ⟨Alice, Charlie, Bob⟩ from a source query. Suppose we restrict the follow-up query to male staff, and obtain ⟨Charlie, Charlie, Bob⟩. This is obviously erroneous for DBMS queries, yet the two outputs satisfy the subset relation {Charlie, Charlie, Bob} ⊆ {Alice, Charlie, Bob}. While repeated elements are insignificant for YouTube video searches in Segura et al.'s work, it makes a huge difference in output lists from DBMS queries.

TABLE 1. CORRELATION BETWEEN SEGURA ET AL. FRAMEWORK AND OUTPUT ATTRIBUTES OF DBMS

	Partitioned	Complete	Ordered	Disjoint
Equivalence	-	-	N	-
Equality	-	-	Y	-
Subset	Y	N	N	-
Disjoint	Y	-	-	Y
Complete	Y	Y	-	-
Difference	-	-	-	-

TABLE 2. COMPREHENSIVE LIST OF MROPS WITH REFERENCE TO OUTPUT ATTRIBUTES OF DBMS

	Partitioned	Complete	Ordered	Disjoint
List equality without partitioning	N	-	Y	-
Bag equality without partitioning	N	-	N	-
Sublist equality	Y	N	Y	-
Subbag equality	Y	N	N	-
List equality of complete disjoint partitioning	Y	Y	Y	Y
Bag equality of complete disjoint partitioning	Y	Y	N	Y
List equality of complete non-disjoint partitioning	Y	Y	Y	N
Bag equality of complete non-disjoint partitioning	Y	Y	N	N

These concerns motivate us for further investigation. We propose a more comprehensive and precise list of eight MROPs for DBMS testing. Their correlation with DBMS output attributes is shown in Table 2. In particular, we find that QP and TLP [8] as well as our proposed DP cover only two MROPs, namely list equality and bag equality of complete disjoint partitioning (as explained in Subsection II.D). Details of the other six MROPs will be explained in the following subsections.

We will use the staff table on the right for illustrating our proposed MROPs throughout the remainder of Section III.

id	name	salary
1	Alice	5000
2	Charlie	2000
3	Bob	3000
4	Charlie	4000

C. List equality without partitioning

For the MROP covering list equality without partitioning, we may construct source queries and follow-up queries such that their output lists are exactly identical, and no partitioning is involved. Consider, for instance the source query

```
SELECT name FROM staff ORDER BY name;
```

It produces ⟨Alice, Bob, Charlie, Charlie⟩ because the default name sequence is ASC for ascending. To verify list equality without partitioning, the follow-up query must contain all the original ORDER BY clauses as per the source query. In addition, we may put in other ORDER BY parameters after them *provided that* the parameters are not part of the SELECT output. In our example, we may have

```
SELECT name FROM staff ORDER BY name, salary;
```

It will generate the same result ⟨Alice, Bob, Charlie, Charlie⟩.

The output sequence in list equality is important. In other words, all the elements in the source and follow-up outputs must match in terms of both values and positions.

D. Bag equality without Partitioning

In addition to the MROP for list equality, we may construct source and follow-up queries such that they satisfy a bag equality relation. Consider, for instance, a source query

(Q2.1) SELECT name FROM staff ORDER BY **name**;

which produces the list ⟨Alice, Bob, Charlie, Charlie⟩. We may then construct a follow-up query by substituting ORDER BY name with ORDER BY salary:

(Q2.2) SELECT name FROM staff ORDER BY **salary**;

which returns ⟨Charlie, Bob, Charlie, Alice⟩. Obviously, the two lists are not expected to be identical, but can be considered as consistent if the values agree while the positions are not relevant. This is formally described as *bag equality* in data structures, such that [Alice, Bob, Charlie, Charlie] = [Charlie, Bob, Charlie, Alice]. Note that we may also add any ORDER BY predicate(s) to either the source or the follow-up query, and the MR will still be preserved.

E. Sublist equality

Consider the following source query for the staff table:

(Q3.1) SELECT name FROM staff ORDER BY name;

We obtain ⟨Alice, Bob, Charlie, Charlie⟩.

We may propose a follow-up query by adding a WHERE predicate

(Q3.2) SELECT name FROM staff **WHERE salary < 5000** ORDER BY name;

Here, we put in an additional condition that the DBMS only returns the staff with salary less than 5000. Note that the ORDER BY clause is identical to that of the source query, thus preserving the order of the output list.

The expected result would be ⟨Bob, Charlie, Charlie⟩. Each element in the follow-up output can be found in the source output. Moreover, the positions of these elements agree with those of the corresponding elements in the source output.

F. Subbag equality

This MROP is similar to sublist equality, except that the order of elements in the outputs are immaterial. Consider the source query

(Q4.1) SELECT name FROM staff ORDER BY name;

which results in ⟨Alice, Bob, Charlie, Charlie⟩. Consider a follow-up query with an additional WHERE predicate and a different ORDER BY clause:

(Q4.2) SELECT name FROM staff **WHERE salary < 5000** ORDER BY salary;

which results in ⟨Charlie, Bob, Charlie⟩. All elements in the follow-up output are included in the source output, but *not* in the same order. We say that the follow-up and the source outputs satisfy *subbag equality*, but not sublist equality. That is, [Charlie, Bob, Charlie] ⊆ [Alice, Bob, Charlie, Charlie], but ⟨Charlie, Bob, Charlie⟩ ⊈ ⟨Alice, Bob, Charlie, Charlie⟩.

G. List equality of complete non-disjoint partitioning

Consider the source query

(Q5.1) SELECT name FROM staff ORDER BY salary;

whose resultant output will be ⟨Charlie, Bob, Charlie, Alice⟩.

Let us partition the original table into two subtables part1 and part2 using LIMIT and OFFSET clauses and the same ORDER BY clause as (Q5.1):

```
CREATE TABLE part1 AS
SELECT id, name, salary FROM staff ORDER BY salary
LIMIT 3 OFFSET 0;

CREATE TABLE part2 AS
SELECT id, name, salary FROM staff ORDER BY salary
LIMIT 3 OFFSET 1;
```

Each subtable consists of consecutive elements in staff, sorted in name sequence as follows:

part1:

id	name	salary
2	Charlie	2000
3	Bob	3000
4	Charlie	4000

part2:

id	name	salary
3	Bob	3000
4	Charlie	4000
1	Alice	5000

We then construct a follow-up query to concatenate the two subtables using UNION ALL, and select the name from DISTINCT elements:

```
(Q5.2) SELECT name FROM (SELECT DISTINCT *
       FROM (SELECT * FROM part1
       UNION ALL
       SELECT * FROM part2) AS union_all)
       AS distinct_union_all;
```

The resultant output will be ⟨Charlie, Bob, Charlie, Alice⟩, which is identical to the source output from (Q5.1) in terms of both values and positions. Any violation would indicate a failure in the DBMS under test.

H. Bag equality of complete non-disjoint partitioning

This MROP is similar to list equality of complete non-disjoint partitioning, except that the order of elements in the outputs are immaterial.

Consider again the source query

(Q6.1) SELECT name FROM staff ORDER BY **salary**;

whose resultant output is ⟨Charlie, Bob, Charlie, Alice⟩.

This time, let us partition the original table into two subtables part1 and part2 using LIMIT and OFFSET clauses, but having an ORDER BY clause different from (Q6.1):

```
CREATE TABLE part1 AS
SELECT id, name, salary FROM staff ORDER BY name
LIMIT 3 OFFSET 0;

CREATE TABLE part2 AS
SELECT id, name, salary FROM staff ORDER BY name
LIMIT 3 OFFSET 1;
```

Each subtable will consist of consecutive elements in staff, sorted in name sequence as follows:

part1:

id	name	salary
1	Alice	5000
3	Bob	3000
2	Charlie	2000

part2:

id	name	salary
3	Bob	3000
2	Charlie	2000
4	Charlie	4000

We then construct a follow-up query to concatenate the two subtables using UNION ALL, and select the name from DISTINCT elements:

```
(Q6.2) SELECT name FROM (SELECT DISTINCT *
       FROM (SELECT * FROM part1
       UNION ALL
       SELECT * FROM part2) AS union_all)
       AS distinct_union_all;
```

The resultant output will be ⟨Alice, Bob, Charlie, Charlie⟩, which is consistent with the source output from (Q6.1) in terms of values but not positions. This is *bag equality* in standard data structures, such that [Charlie, Bob, Charlie, Alice] = [Alice, Bob, Charlie, Charlie]. Any violation would indicate a failure in the DBMS under test.

IV. Part (C): Hidden Failures and New Crashes Detected in Empirical Case Study

By extending Rigger's open-source application SQLancer, we have implemented all our proposed MROPs for testing OceanBase Community Edition version 3.1.0. Specifically, we adapted SQLancer to be compatible with OceanBase, and developed separate modules for running metamorphic test cases for the proposed MROPs. Using randomly generated variables, our tool automatically constructs source and follow-up queries to verify target metamorphic relations in each MROP. These randomly generated variables include the number of partitions, the number of records in each partition, and the columns used in query phrases. Our tool also logs the results from source and follow-up queries. More importantly, it automatically verifies their consistencies with reference to the specified MRs, such as the number of records and the ordering of the records. Despite potential combinatorial challenges, failures were revealed shortly after executing the tool for about 100 iterations in our actual experimentation.

TABLE 3. SUMMARY OF FAILURES DETECTED IN OCEANBASE COMMUNITY EDITION VERSION 3.1.0.

MROP	Failures
List equality without partitioning	-
Bag equality without partitioning	-
Sublist equality	**(1) Incorrect ordering when using WHERE column IS TRUE** (2) Incorrect ordering when using LIKE
Subbag equality	(3) Incorrect row retrieved when using ORDER BY and LIMIT (4) Incorrect zero value when using COALESCE() and IFNULL()
List equality of complete disjoint partitioning	(5) Missing row when using BIT_COUNT() (6) Missing row when using bitwise "or" (|) operator
Bag equality of complete disjoint partitioning	(7) Missing row when using LEAST() and bitwise "and" (&) operator (8) Missing row when using NOT and NOT IN(NULL)
List equality of complete non-disjoint partitioning	(9) Missing row when using NOT and not equal (<>) operator (10) Missing row when using bitwise "and" (&) and INT
Bag equality of complete non-disjoint partitioning	(11) Missing row when using bitwise XOR (^) operator (12) Missing row when using 0 IN(NULL)

TABLE 4. SUMMARY OF CRASHES DETECTED IN OCEANBASE COMMUNITY EDITION VERSION 3.1.0.

MROP	Crashes
List equality without partitioning	(1) Error in "type conversion in expression evaluation" when using DELETE FROM
Bag equality without partitioning	-
Sublist equality	(2) "Result value was out of range" when using CAST(varchar AS SIGNED)
Subbag equality	(3) "Invalid argument" when using IS TRUE
List equality of complete disjoint partitioning	(4) Internal error when using !GREATEST(), IN(), ORDER BY, and LIMIT (5) Internal error when using (NULL =), IN(), ORDER BY, and LIMIT
Bag equality of complete disjoint partitioning	(6) Timeout when using UNION ALL for multiple SELECT statements (7) Internal error when using (NULL =), IN(), LEAST(), ORDER BY, and LIMIT
List equality of complete non-disjoint partitioning	(8) Internal error when using EXISTS(), ORDER BY, and LIMIT
Bag equality of complete non-disjoint partitioning	-

We have successfully revealed 12 hidden failures and 8 new crashes. Table 3 summarizes the respective failures under various MROPs. Table 4 summarizes the respective crashes. Among them, two have been fixed in a subsequent release version 3.1.1 by the OceanBase QA team of the Ant Group. Others are being investigated or scheduled to be fixed in future releases. Our test data, including the tables and queries, are available at https://github.com/tangsiuhin/matobas4CX.

A. Sample Failure

Consider the following transaction table, which records the transactions for an online shop.

id	date	time	amount
1	2023-01-01	15:20:00	100
2	2023-01-01	16:30:00	300
3	2023-01-02	11:00:00	null
4	2023-01-02	12:10:00	600
...	...	...	...
520	2023-02-11	09:40:00	400
521	2023-02-11	10:50:00	500
...	...	...	...

We would like to list all the transactions on each date, sorted in descending order of amount, using the following source query:

```
(Q7.1) SELECT id, amount FROM transaction
       ORDER BY date ASC, amount DESC;
```

OceanBase returned a resultant list of 10 000 entries.

Suppose there are incomplete transactions with missing amount values denoted by null. In the resultant list, the incomplete transactions are also included. They are sorted to the end of all the transactions for any particular date, as shown on the right:

id	amount
...	...
4	600
3	null
...	...

Based on the concept of sublist equality, we propose the follow-up query below, with the WHERE keyword followed by amount IS TRUE, to capture all the completed transactions, that is, those not having a null amount. We continue to use the same ORDER BY clause as in the source query.

```
(Q7.2) SELECT id, amount FROM transaction
       WHERE amount IS TRUE
       ORDER BY date ASC, amount DESC;
```

OceanBase returned a resultant list containing 9900 entries.

We need to verify whether the resultant list from (Q7.2) is a sublist of that from (Q7.1). In our empirical case study, this was conducted in three steps: (a) We checked that the number of entries in the follow-up output was less than or equal to that in the source output. (b) We checked that all the transactions in the follow-up output were also in the source output. (c) We removed those transactions in the source output that did not appear in follow-up output, and checked that the modified source output was exactly the same as the follow-up output.

We found from step (c) that the modified source output was not identical to the follow-up output. Although both lists contain the same transactions, the ordering was different. Transaction 521 appeared before 520 in the source output, but after 520 in the follow-up output. Hence, a failure was detected.

B. Crashes

In addition to the failures, we have also identified eight crashes during the execution of either source test cases or follow-up test cases. OceanBase terminated itself with messages such as internal error, timeout error, or other unexpected errors.

V. Conclusion

To the best of our knowledge, this is the first project that integrates an in-depth approach and a comprehensive approach to construct metamorphic relations for DBMS testing. We have conducted our project in three parts.

Part (A) focuses on an in-depth investigation into existing MT techniques in DBMS testing. We have identified a gap between query partitioning and ternary logic partitioning in [8]. We tackle the issue using a new concept of disjoint partitioning.

Part (B) focuses on the comprehensiveness of existing MR output patterns for DBMS testing. We have reviewed the MROP framework in [9] against the diverse challenges in DBMS testing. We propose a comprehensive MROP framework for DBMS. In particular, we find that disjoint partitioning in Part (A) only covers two of the eight output patterns. Thorough investigations have been conducted for the remaining patterns.

Part (C) applies the orthogonal (in-depth and comprehensive) approaches to OceanBase. Even though OceanBase has been tested extensively and applied widely in the DBMS community, we have revealed 12 failures and 8 crashes. We find that both orthogonal approaches are necessary for constructing useful MRs in DBMS testing. When compared with existing MT techniques, our framework enables a more systematic and exhaustive exploration of potential output patterns, thereby increasing the likelihood of uncovering hidden failures in DBMS.

More recently, Segura et al. [17][18] proposed metamorphic relation input patterns (MRIPs) for testing query-based systems. As future work, we would also like to study MRIPs for the metamorphic testing of DBMS.

Acknowledgments

This project was supported in part by an internship of the first author at Alibaba and Ant Group, China. We would also like to thank their QA team for confirming our failure reports.

References

[1] *What is OLTP?*, Oracle Corporation, 2022. [Online]. Available: https://www.oracle.com/database/what-is-oltp/

[2] A. Alsharif, "Automated software testing of relational database schemas," Ph.D. dissertation, Dept. Comput. Sci., Univ. Sheffield, Sheffield, U.K., 2020.

[3] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE Trans. Softw. Eng.*, vol. 41, no. 5, pp. 507–525, 2015.

[4] K. Patel and R. M. Hierons, "A mapping study on testing non-testable systems," *Softw. Quality J.*, vol. 26, no. 4, pp. 1373–1413, 2018.

[5] T. Y. Chen, F.-C. Kuo, H. Liu, P.-L. Poon, D. Towey, T. H. Tse, and Z. Q. Zhou, "Metamorphic testing: A review of challenges and opportunities," *ACM Comput. Surv.*, vol. 51, no. 1, pp. 4:1–4:27, 2019.

[6] T. Y. Chen and T. H. Tse, "New visions on metamorphic testing after a quarter of a century of inception," in *Ideas, Visions and Reflections Track, Proc. ACM Joint Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.* (*ESEC /FSE '21*), New York, NY, USA: ACM, 2021, pp. 1487–1490.

[7] S. Segura, G. Fraser, A. B. Sanchez, and A. Ruiz-Cortes, "A survey on metamorphic testing," *IEEE Trans. Softw. Eng.*, vol. 42, no. 9, pp. 805–824, 2016.

[8] M. Rigger and Z. Su, "Finding bugs in database systems via query partitioning," *Proc. ACM Program. Lang.*, vol. 4, issue OOPSLA, pp. 211:1–211:30, 2020.

[9] S. Segura, J. A. Parejo, J. Troya, and A. Ruiz-Cortes, "Metamorphic testing of RESTful web APIs," *IEEE Trans. Softw. Eng.*, vol. 44, no. 11, pp. 1083–1099, 2018.

[10] *OceanBase*, 2021. [Online]. Available: https://www.oceanbase.com/en

[11] *TPC-C*, Wikipedia. Accessed: January 3, 2023. [Online]. Available: https://en.wikipedia.org/wiki/TPC-C

[12] W. E. Howden, "Theoretical and empirical studies of program testing," *IEEE Trans. Softw. Eng.*, vol. 4, no. 4, pp. 293–298, 1978.

[13] M. Lindvall, D. Ganesan, R. Ardal, and R. E. Wiegand, "Metamorphic model-based testing applied on NASA DAT: An experience report," in *Proc. 2015 IEEE/ACM 37th Int. Conf. Softw. Eng.* (*ICSE '15*), Piscataway, NJ, USA: IEEE, 2015, pp. 129–138.

[14] M. Rigger and Z. Su, "Detecting optimization bugs in database engines via non-optimizing reference engine construction," in *Proc. 28th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.* (*ESEC /FSE '20*), New York, NY, USA: ACM, 2020, pp. 1140–1152.

[15] *Three-valued Logic*, Wikipedia. Accessed: January 3, 2023. [Online]. Available: https://en.wikipedia.org/wiki/Three-valued_logic

[16] M. S.-H. Tang, T. H. Tse, and Z. Q. Zhou, "A disjoint-partitioning approach to enhancing metamorphic testing of DBMS," in *Proc. 2022 IEEE Int. Symp. Softw. Rel. Eng. Workshops* (*ISSREW '22*), Piscataway, NJ, USA: IEEE, 2022, pp. 130–131.

[17] S. Segura, A. Duran, J. Troya, and A. Ruiz-Cortes, "Metamorphic relation patterns for query-based systems," in *Proc. 2019 IEEE/ACM 4th Int. Workshop Metamorphic Testing* (*MET '19*), Piscataway, NJ, USA: IEEE, 2019, pp. 24–31.

[18] S. Segura, J. C. Alonso, A. Martin-Lopez, A. Duran, J. Troya, and A. Ruiz-Cortes, "Automated generation of metamorphic relations for query-based systems," in *Proc. 2022 IEEE/ACM 7th Int. Workshop Metamorphic Testing* (*MET '22*), Piscataway, NJ, USA: IEEE, 2022, pp. 48–55.