Basic principles of blind write protocol

Khairul Anshar, Nanna Suryana, Noraswaliza
Centre for Advanced Computing Technology, Faculty of Information and Communication Technology,
Universiti Teknikal Malaysia Melaka, Malaysia

Article Info

Article history:

Received Aug 31, 2019
Revised Nov 4, 2019
Accepted Feb 8, 2020

Keywords:

Availability
Blind write protocol
Concurrency control
Consistency
Locking protocol

ABSTRACT

The current approach to handle interleaved write operation and preserve consistency in relational database system still relies on the locking protocol. If any entity is locked by any transaction, then it becomes temporary unavailable to other transaction until the lock is released. The temporary unavailability can be more often if the number of write operation increases as happens in the application systems that utilize IoT technology or smartphone devices to collect the data. To solve this problem, this research is proposed blind write protocol which does not lock the entity while the transaction is performing a write operation. This paper presents the basic principles of blind write protocol implementation in a relational database system.

Corresponding Author:

Khairul Anshar,
Centre for Advanced Computing Technology,
Faculty of Information and Communication Technology,
Universiti Teknikal Malaysia Melaka,
Hang Tuah Jaya, 76100 Durian Tunggal, Melaka, Malaysia.
Email: p031420004@student.utem.edu.my

1. INTRODUCTION

Current implementation of concurrency control in Relational Database Management System (RDBMS) [1] handles the interleaved operations and temporary inconsistent at the database system level. Eswaran et al. described in [2], when someone is transferring money from one to another bank account, there will be a window that one bank account has been deducted but the other account not yet added because they are performed in one transaction that execute all the operations one by one. If this happens, then there should no other transaction access those 2 bank accounts to preserve the consistency. Therefore, Eswaran et al. proposed Locking Protocol (LP). This implementation is applied in the database system level.

Stearns et al. proposed another approach that utilize a version of entity and certification process [3]. Each version of entity is unique, and it is used to identify the temporary inconsistent entity. In this approach, any transaction can access any entity including the one in the temporary inconsistent state (uncertified version) with the consequence that the transaction may be restarted by the concurrency control. Once the transaction can get the terminate request granted, the they become certified version otherwise it must be restarted. This implementation is also applied in the database system level.

Kung et al. in [4] proposed an optimistic approach which is to utilize local copies to handle temporary inconsistent. In this approach, all reads, and writes will be performed in the local copies during the read phase. To make them available to other transaction globally then it requires the integrity validation before going to write phase. If the transaction fails while performing the integrity validation, then it must be restarted. This implementation is also applied in the database system level. The LP is proposed to achieve more on consistency. The last two approaches are proposed to achieve more on availability. These 3 concurrency controls above are handling the interleaved write operations at the system level. Therefore, an application does not have flexibility to determine in each write operation whether it needs to preserve consistency over availability or the opposite.

The objective of the concurrency control is to increase the throughput of database by allowing more operations to be processed and interleaved as many as possible. Moreover, each operation in application has different consistency and availability requirement. The application system is developed to fulfill the business or process requirement which is transformed into read and write operation in the software lines of code. Therefore, the application system has the knowledge on the consistency and availability requirement of each operation. Some of the write operation depend on the current or other entity value, e.g. withdraw or deposit operation to the bank account. To perform these operations, the concurrency control should use locking protocol to achieve consistency and prevent the concurrency anomaly. But there is also a situation that the write operation does not depend on the current or other entity, e.g. any write operation which uses a constant or fix value. These operations are considered as blind write operation.

Stearns et al. explained in [5], "*blind writes is defined as a process to write on a particular entity without first issuing a read request on that entity*." Mendonca et al explained in [6], "*during a blind write operation, copies are modified regardless of their previous values*." Burger et all explained in [7], "*a blind write operation does not perform a read before the data item is written*." For this type of operation, the result depends on the last operation, know as the last performed wins. Hence, it does not need to perform locking protocol to the entity in order to improve the availability.

This paper proposes blind write protocol (BWP) as a complement to the current concurrency control to be applied in the RDBMS. Our motivation is to give the application system a new option to deal with interleaved write operation. As a result, if the write operations are using constant or fix value then they can use blind write protocol which does not lock the entity. This approach can prevent unnecessary temporary unavailable situation and prevent unnecessary waiting. As a result, it increases the availability of entity. To understand more on the BWP, we start the discussion by reviewing the concurrency control in Section 2. Then, we describe about blind write protocol and its implementation in next section. We present the basic unit testing result in Section 4. The last section concludes the topic.

2. CONCURRENCY CONTROL

Serialization using locking protocol is achieved by making one or more transaction wait until the lock is released. The concurrency control will not abort any transaction until the deadlock or wait-lock-timeout occurs. Serialization using version control and local copy is achieved by restarting a transaction if it fails to get certified process or fails while performing the integrity validation. We do not find any discusion on the concurrency control that allowed data to be not consistent in order to achieve availability in RDBMS. The hierarchy chart of RDBMS concurrency control can be seen in Figure 1. Since it is a high level of hierarchy, then we do not add the granularity of lock as discussed in [8].

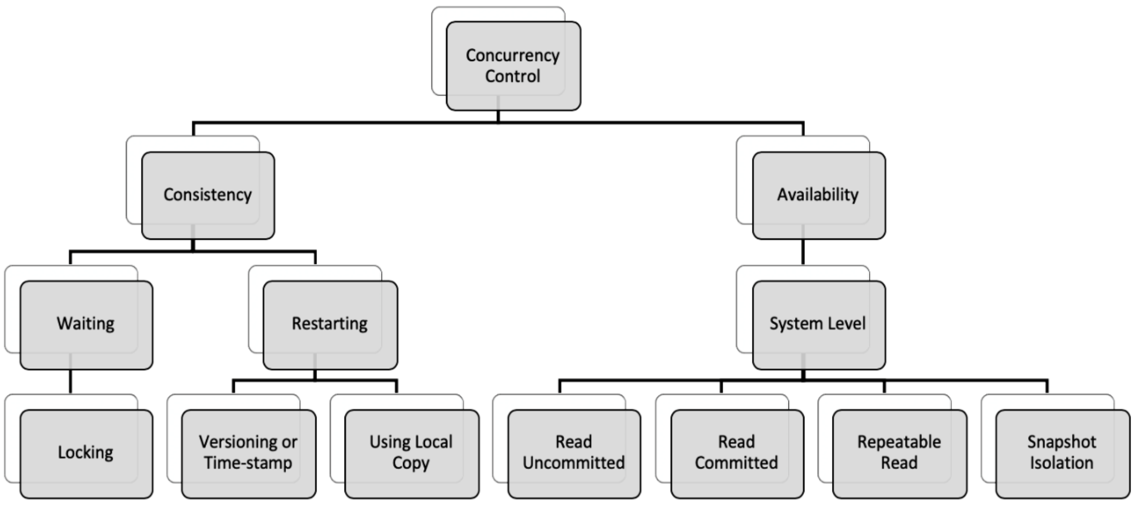


Figure 1. The K-Chart of RDBMS concurrency control

In Figure 1, there are two properties below the concurrency control. They are coming from the trade off between consistency and availability in partitioned system as discussed in CAP theory [9, 10]. Since all write operations in RDBMS are forced to achieve consistency, then there is no approach proposed for write operation to achieve the availability property and ignore the consistency. The consistency property can be achieved by forcing a transaction to wait or restarted, if conflict is raised. In the locking protocol approach, the conflict is raised when a transaction tries to access the locked entity. In the version control approach, the conflict is raised if it fails to get certified process. In the local copy approach, the conflict is raised if it fails in the integrity validation. The discussion on the consistency aims to prevent the concurrency control anomaly. Some of write operation depends on the current or other entity value, e.g. withdrawal or deposit operation to the bank account. To perform these operations, the concurrency control should prevent the lost update and write skew concurrency anomaly.

2.1. Lost update anomaly

This anomaly happens when two transactions perform write operation to the same entity at same time. To describe it, let's say there are two transactions, T_1 and T_2 are executed at the same time as shown at Listing 1. Both transactions are based on the initial state of $e_1=10$.

Seq.	Initial balance amount, e_1=10;	
	T_1	T_2
1	begin	begin
2	$e_1 \leftarrow e_1 + 10$;	$e_1 \leftarrow e_1 + 30$;
3	commit;	commit;
4	end;	end;
	End result of balance amount is either e_1=20 or e_1=40	

Listing 1. Lost update anomaly

In Listing 1, the operations are performed from the top to the bottom indicated by sequence number. We use $\leftarrow$ notation as assigning a value from the right, called new value, to the balance amount entity on the left, e_1. In the absence of concurrency control, the end result of balance amount can either $e_1=20$ or $e_1=40$. This result is known as lost update anomaly. In order to preserve consistency then the RDBMS requires a concurrency control to handle these 2 interleaved deposit operations performed by two different transactions. The LP will make either T_2 wait until T_1 is completed or T_1 wait until T_2 is completed. The end result of balance amount is consistent i.e. $e_1=50$ regardless which transaction is processed at first.

The LP works perfectly on the write operation which depend on the current or other entity value. When the entity is locked then it becomes temporary unavailable to other transaction. If other transactions want to access the entity, then they need to wait until the entity is available. The BWP is started with throwing basic question such as how if the new value is a constant or fix value as shown in Listing 2. Do we need to lock the entity? Eventhough the LP is applied, the end result still remains the same, i.e. either $e_1=20$ or $e_1=30$, known as Last Performed Wins (LPW).

Seq.	Initial balance amount, e_1=10;	
	T_1	T_2
1	begin	Begin
2	$e_1 \leftarrow 10$;	$e_1 \leftarrow 20$;
3	commit;	commit;
4	end;	end;
	End result of balance amount is either e_1=20 or e_1=30	

Listing 2. Blind write operation

The answer for the question above depends on the application system requirement. If the situation is same as displayed in Listing 2, then it does not have any effect. The LP can only affect to a situation as displayed in Listing 3, i.e. a transaction has more than one blind write operations. With the LP, the end result has two possibilities only, i.e. either $e_1=20$ and $e_2=200$ or $e_1=30$ and $e_2=300$. With BWP, the end result has another 2 possibilities, i.e. $e_1=20$ and $e_2=300$ or $e_1=30$ and $e_2=200$.

Seq.	Initial balance amount, e_1=10;	
	T_1	T_2
1	begin	begin
2	$e_1 \leftarrow 20;$	$e_1 \leftarrow 30;$
3	$e_2 \leftarrow 200;$	$e_2 \leftarrow 300;$
4	commit;	commit;
5	end;	end;
	End result is either e_1=30; and e_2=300; or e_1=20; and e_2=200; or e_1=20; and e_2=300; or e_1=30; and e_2=200;	

Listing 3. Blind write operation with 2 operations

As discussed above, if these four possibilities of end result are accepted by the application system, then it does not require to lock the entity. This can prevent unnecessary temporary unavailable situation and prevent unnecessary waiting. As a result, it preserves the availability of entity. Terry Doug explained in [11], "*high availability is not sufficient for most application system, but strong consistency is not needed either*." Vogels argued in [12], "*there is a range of applications that can handle slightly stale data, and they are served well under this model*. " In other hand, Bernstein argued in [13] that "*the high availability increase the application complexity to handle inconsistent data*." Therefore, the transaction should have another option in addition to preserve the consistency. If a transaction wants to preserve consistency, then LP can be used otherwise use BWP.

2.2. Write skew anomaly

This anomaly happens if two transactions perform withdrawal operation to the same account number at same time. To describe it, let say two transactions, T_1 and T_2, are executed at the same time as shown at Listing 4. Both transactions are based on the initial balance amount, e_1*=100*. The withdrawal operation has a condition to be fulfilled, i.e. the end result should be greater or equal to 0. Therefore, each transaction should apply a validation operation as shown in seq#3 in Listing 4.

Seq.	Initial balance amount, e_1=100;	
	T_1	T_2
1	begin	begin
2	$t_{amount} \leftarrow 60;$	$t_{amount} \leftarrow 80;$
3	if (e_1 - t_{amount}) >=0	if (e_1 - t_{amount}) >=0
4	$e_1 \leftarrow e_1$ - tAmount ;	
5	commit;	
6	end if;	$e_1 \leftarrow e_1 - t_{amount};$
7	end;	commit;
8		end if;
9		end;
	End result of balance amount can be e_1=-40	

Listing 4. Write skew anomaly

Since there is no LP applied, the result of T_1 on seq#3 is 40, which is greater than 0, and the result of T_2 is 20, which is greater than 0. Both transactions meet the specified condition for withdrawal operation, so it can continue to the next operation. When T_1 is able to complete seq#4 and 5, then value of e_1 on seq. no. 6 is *100–60=40*. After T_2 perform seq#6 and 7, then it gives a negative balance amount, i.e. *-40*.

If the LP is applied by using “lock for update”, then the operation on seq#3 of T_2 will wait until T_1 completes seq# 5 in Listing 4. It can be seen in Listing 5. Since T_1 is able to lock the e_1 on the seq#3 then T_2 should wait until T_1 performs commit to release e_1 or preempt. Once the lock on e_1 is released, then T_2 can perform lock for update on e1 as shown on seq#7 in Listing 5. Now, the value of eforupdate on the seq# 6 is 40 instead of 100. When the T_2 perform seq#8 then it does not meet with the specified condition. As a result, the T_2 will not performs seq#9 and 10. Therefore, the end result of balance amount will remain the same, i.e. e_1*=40*.

BWP does not lock any entity when write operation is executed to allow more transaction to be interleaved. This forces the application system has to apply its own approach to preserve the consistency. By utilizing the transaction history as discussed on [14-16], BWP can preserve the consistency for both the withdrawal and deposit operation shown on Table 1.

Seq.	Initial balance amount, e_1=100;	
	T_1	T_2
1	begin	begin
2	$t_{amount} \leftarrow 60;$	$t_{amount} \leftarrow 80;$
3	$e_{forUpdate} \leftarrow$ lock(e_1)	
4	if ($e_{forUpdate}$ - t_{amount}) >=0	
5	$e_1 \leftarrow e_1$ - tAmount ;	
6	commit;	$e_{forUpdate} \leftarrow$ lock(e_1)
7	end if;	if (e_1 - t_{amount}) >=0
8	end;	$e_1 \leftarrow e_1 - t_{amount};$
9		commit;
10		end if;
11		end;
12		
	End result of balance amount is e_1=40	

Listing 5. Lock for update

Table 1. Transaction history table

History#	Previous Balance	Trasaction Amount	Balance Amount	Trasaction Status
1	0	1000	1000	Approved
2	1000	-100	900	Approved
3	900	-200	700	Approved
4	700	-800	700	Rejected
5		200		In Process
6		-500		In Process
7		-300		In Process
8		-200		In Process

The first transaction in Table 1 is deposit with amount 1000. The transaction amount, which is less than 0, it indicates that the type of transaction is withdrawal. The transaction status become rejected if the account does not have enough balance to perform withdrawal operation and the balance amount remains the same with the previous balance. In the LP, the serialization is achieved by locking the entity. In BWP, it is achieved by generating the database sequence value for each transaction.

2.3. Isolation level

The discussion in [14] focuses on allowing more operations to be interleaved because of strict 2-phase-locking protocol, all locking is released after the transaction perform commit or rollback. Streans et al in [5] argue that since the new value of T_1 has not been committed yet, then it is natural choice to make the T_2 wait until T_1 is committed. However, Streans explained further, since the database system also has the initial value before any other transaction commit their changes, then giving the initial value to T_2 can improve the concurrency. The uncommitted read, committed read and snapshot isolation level are proposed to improve the concurrency and known as isolation level.Uncommitted read allows the read operation to get uncommitted entity value. But RDBMS may become inconsistent if the uncommitted transaction is rollback. Committed read allows the read operation to get committed entity value only. But two read committed operations to the same entity in one transaction may return different value. It is due to in between two operations there could be a write operation by other transaction. It is known as non-repeatable read anomaly.

Read lock, and snapshot isolation guarantee the two read operations in one transaction get same entity value. Read lock to locked entity should wait until the lock is released by other transaction or preempt. Snapshot isolation is proposed to avoid read lock [17]. Snapshot isolation reads the entity value from the log, but it still relies on the locking protocol to prevent write skew [18].The latest discussion on the concurrency control is trying to make the snapshot isolation is able to prevent write skew concurrency anomaly and makes the interleaved transactions become serializable [19-21]. The discussion on making the interleaved transactions in the read committed isolation become serializable is started in [22]. Their approaches are similar with [3] and [4], i.e. the system has to abort or restart one of the interleaved transactions if conflict pattern called dangerous structure appears [23, 24]. Another discussion tries to improve the throughput by staged allocation and deallocation of locks in bulk [25].

3. BLIND WRITE PROTOCOL

In this section, we are describing the definition and the basic principles of blind write protocol. The interaction between client end point with RDBMS is known as transaction. This interaction consists of one or more operations. The operation can be read or write. Write operation is a process to create new entity, modify or delete any existing entity. Read operation is a process to get entity value.

3.1. Definition

Let's define database system as *D* which consists of *n* number of entity.

$D=\{e_1, e_2, e_3, ..., e_n\}$

These entities can be either tables, rows, or columns. The BWP is focusing on the Data Manipulation Language (DML), which create, modify or delete a row into, in, or from a table. The Data Definition Language (DDL) is not part of BWP discussion. We also consider that modifying a current value of one column as modifying a row. Therefore, the write operation is action to assign a value to the entity. Create operation is considered as assigning any value, v, to new entity, e_{n+1},

$e_{n+1} \leftarrow v$; where v is not *NULL*.

Delete operation is considered as assigning *NULL* to existing entity, e_i,

$e_i \leftarrow NULL$; where $1 \leq i \leq n$.

Update operation is considered as assigning a value, v, to existing entity, e_i,

$e_i \leftarrow v$; where v is not *NULL* and $1 \leq i \leq n$.

The value of v above can be defined as:
- Function of any entity, known as normal write operation.
- Constant or fixed value, e.g. '*APPROVED*', '*536980 MALAYSIA*', '*+6012345678*', *20*, etc. It is known as blind write operation.

3.2. Data manipulation language

We propose a set of DML statements to distinguish the blind write protocol with the normal write protocol, as a follow:
- Create Operation:
  BLIND INSERT INTO *table_name* (*list_of_columns*) **VALUES** (*list_of_values*)
- Delete Operation:
  BLIND UPDATE *table_name* **SET** *column_name=value* [, *column_name=value*] [**WHERE** *condition*]
- Update Operation:
  BLIND DELETE FROM *table_name* [**WHERE** *condition*]

As per blind write protocol definition, the value on the proposed statements above should be a fixed value. It should not use a function from other or its own entity. But in the implementation, it may have no validation, either the value is fixed or a function of any entity. Since the application systems know what the detail requirement is, then it becomes their responsibility either they want to use normal or blind write operation. Since the implementation allows to use such function then the blind write protocol may experience any concurrency anomaly. Therefore, the blind write protocol is not intended to replace the existing concurrency control that can prevent concurrency anomaly. The blind write protocol is proposed to give more option to the application to allow more operation to be interleaved with the consequence that preserving consistency becomes application system responsibility.

3.3. Basic principles

The objective of blind write protocol is to allow more transactions to be processed then it should make the entity to be highly available. There are some basic principles that must be applied into RDBMS to deliver high availability. The basic principles are as follows:
- The transaction should not lock the entity. It is required in order to achieve the main objective, i.e. to prevent unnecessary temporary unavailable situation and prevent unnecessary waiting. As a result, it increases the availability of entity.

– The transaction should apply autonomous auto-commit for every blind write operation execution. It is required to prevent the dirty read anomaly due to the blind write operation does not lock entity. So that, the effect of blind write operation can be accessed by other transaction.
– If the blind write operation need to access the locked entity, then the blind write operation still has to wait until it is released. It is required in order to preserve the consistency.

The autonomous auto-commit should persistently store the effect of the blind write operation only and it should not store any previous normal write operation effect that have not been committed yet. Example, there is a transaction with three operations with the first operation is normal write, the second is blind write operation and the third operation is rollback. The rollback should apply to the first operation only which is normal write operation. The result of second operation, which is blind write operation, should be committed persistently into the disk even the following operation is rollback. The algorithm for invoking autonomous auto-commit for blind write operation is shown in Figure 2.

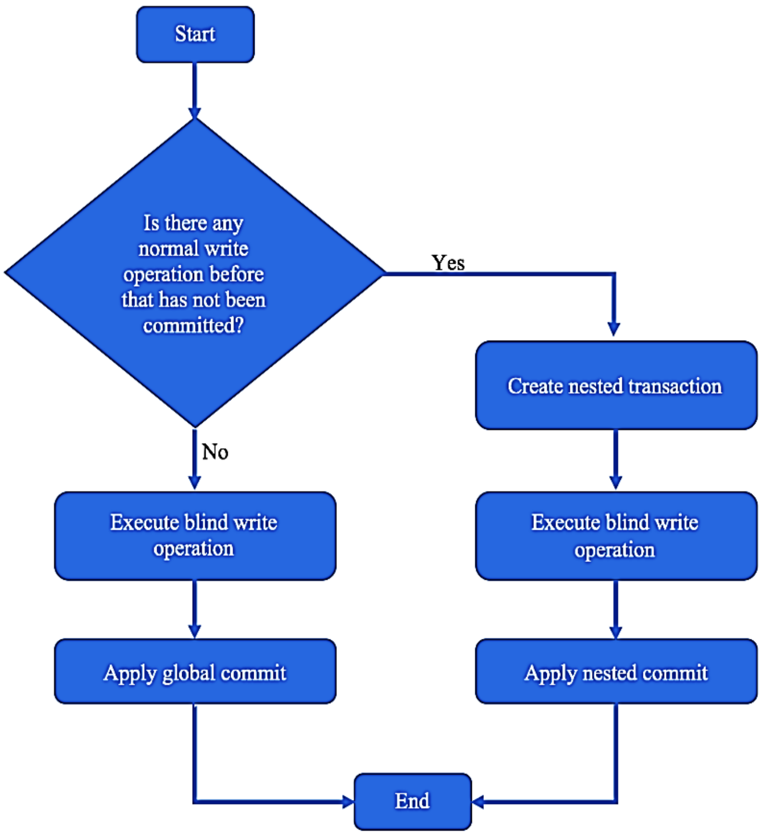

Start
Is there any normal write operation before that has not been committed?
Yes
No
Create nested transaction
Execute blind write operation
Execute blind write operation
Apply global commit
Apply nested commit
End

Figure 2. Autonomous auto-commit algorithm

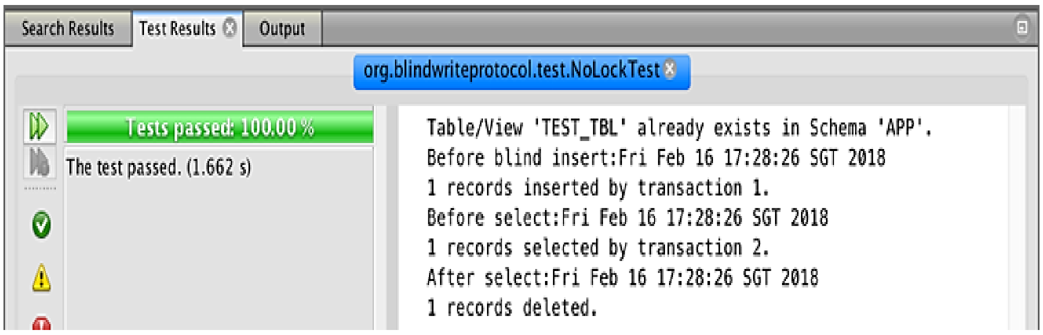
4. RESEARCH METHOD AND UNIT TESTING

In this research we use a prototype system as proof of concept (POC) system to perform necessary unit test. It is performed to validate the blind write protocol basic principles. We have successfully implemented the blind write protocol into the Apache Derby Database as POC system as discussed in [16]. We use 3 unit-test plans to observe and verify the blind write protocol basic principles implementation. We present the testing result in this Section.

4.1. No-locking unit test and result

To verify the basic principle number one, we create two connections. The first connection is used to perform blind insert and the second one is used to perform select statement. We also disable the apache auto-commit, to make sure that there is no commit operation performed by any connection. The result of this testing can be seen on Figure 3.

Since the select operation is using different transaction with the blind insert operation, then the select operation should wait if the locking protocol is applied until the first connection performs commit or rollback operation. But in Figure 3, we can see that the select operation is able to get the result of the blind insert operation which is using different connection with the select operation. It shows us that the blind write protocol did not lock any entity while performing blind insert operation.


Search Results
Test Results
Output
org.blindwriteprotocol.test.NoLockTest
Tests passed: 100.00 %
The test passed. (1.662 s)

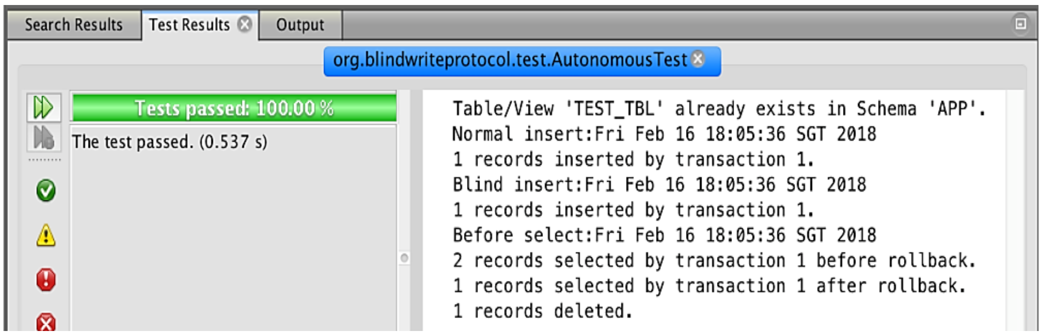
```
Table/View 'TEST_TBL' already exists in Schema 'APP'.
Before blind insert:Fri Feb 16 17:28:26 SGT 2018
1 records inserted by transaction 1.
Before select:Fri Feb 16 17:28:26 SGT 2018
1 records selected by transaction 2.
After select:Fri Feb 16 17:28:26 SGT 2018
1 records deleted.
```

Figure 3. No-locking unit testing result

4.2. Autonomous auto-commit unit test and result

In the previous testing, we did not lock any entity while performing blind insert operation. It means, other transaction can access the effect of blind write operation immediately. To prevent the dirty read anomaly, as discussed in Section 3, blind write protocol should apply autonomous auto-commit for every blind write operation execution. Moreover, this commit should apply into a blind write operation only. Therefore, we should use only one connection to test the Autonomous Auto-Commit unit testing.

In this testing, we perform normal insert before blind insert operation. After both insert operations are performed, then we perform select statement operation to check the number of record before we perform rollback and it returns two records as expected as shown in Figure 4. Once the rollback is performed, then we perform a select statement operation again and it returns one record only as expected as shown in Figure 4. It means the blind write operation was successfully committed autonomously and the rollback is performed to the normal write operation effect only.


Search Results
Test Results
Output
org.blindwriteprotocol.test.AutonomousTest
Tests passed: 100.00 %
The test passed. (0.537 s)

```
Table/View 'TEST_TBL' already exists in Schema 'APP'.
Normal insert:Fri Feb 16 18:05:36 SGT 2018
1 records inserted by transaction 1.
Blind insert:Fri Feb 16 18:05:36 SGT 2018
1 records inserted by transaction 1.
Before select:Fri Feb 16 18:05:36 SGT 2018
2 records selected by transaction 1 before rollback.
1 records selected by transaction 1 after rollback.
1 records deleted.
```

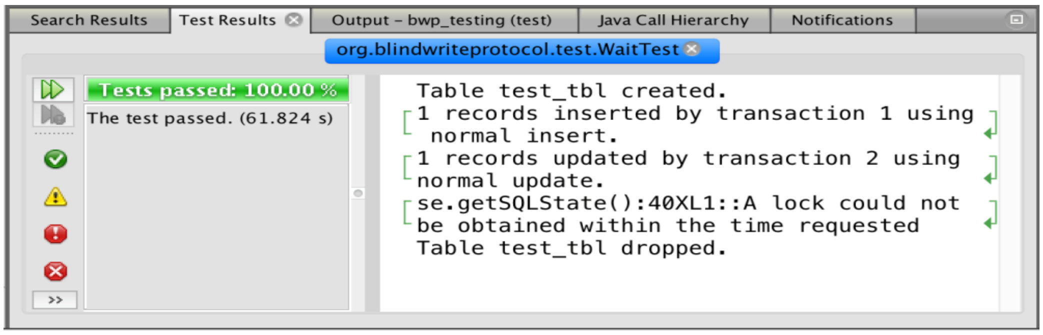
Figure 4. Autonomous auto-commit unit testing result

4.3. Blind write operation on the locked entity unit test and result

The blind write protocol is proposed as a complement to the current concurrency control. There will be a situation that the blind write operation needs to access the entity that has been locked by other transaction. Even though, the blind write protocol does not lock entity, but it still should wait if the blind write operation wants to access the locked entity. Therefore, we perform this testing to verify this situation.

In this testing, we use three connections. The first connection is used to insert a record using locking operation and then perform commit operation. The second connection is used to update the same record using locking operation without performing commit operation. The third connection is used to update the same record using blind update operation. The result of this testing can be seen on Figure 5. On the Figure 5, we can see that the blind write operation is not able to access the locked entity. As a result, the RDBMS throws an error operation after the blind write operation is waiting for more than the lock wait timeout.

Even though on the basic principle number one, we state that the blind write operation should not lock entity. The lock that we described here is an exclusive lock or lock for update. The blind write protocol still need to perform read or sharing lock on the entity in order to check whether the entity is free or in exclusive locked or lock for update.



The test passed. (61.824 s)

```
Table test_tbl created.
1 records inserted by transaction 1 using
 normal insert.
1 records updated by transaction 2 using
normal update.
se.getSQLState():40XL1::A lock could not
be obtained within the time requested
Table test_tbl dropped.
```

Figure 5. Blind write operation on the locked entity unit testing result

5. CONCLUSION

In this paper, we proposed the BWP with its basic principles and DML statement. The basic principles as follows: the transaction should not lock the entity, the transaction should apply autonomous auto-commit for every blind write operation execution, the transaction should wait or preempt if it wants to perform write operation on the locked entity by other transaction.

The lock that we referred here is an exclusive lock or lock for update. The BWP should not apply an exclusive lock or lock for update to the entity in order to prevent unnecessary temporary unavailable situation and prevent unnecessary waiting. As a result, it increases the availability of entity. If the blind write operation needs to access the locked entity by other transaction, then the blind write operation still has to wait until it is released. Therefore, the blind write protocol still requires applying read or sharing lock which will not make the entity becomes temporary unavailable. The autonomous auto-commit that we described here that it should only store the effect of the blind write operation persistently. Hence, any previous normal write operation effect still can be rolled-back.

REFERENCES

[1] E. F. Codd, "A Relational Model of Data for Large Shared Data Banks," *Commun. ACM*, pp. 377-387, 1970.
[2] K. P. Eswaran, et al., "The Notions of Consistency and Predicate Lock in a Database System," ACM Comput. Surv., vol. 19, no. 11. pp. 624-633, 1976.
[3] R. E. Stearns, et al., "Concurrency control for database systems," *7th Symp. Foundations of Computer Science*, pp. 19-32, 1976.
[4] H. T. Kung and J. T. Androbinson, "An Optimistic Methods for Concurrency Control," *ACM Transactions on Database Systems*, vol. 6, no. 2, pp. 213-226, 1981.
[5] R. E. Stearns and D. J. Rosenkrantz, "Distributed Database Concurrency Controls using Before-values," *Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data*, pp. 74-83, 1981.
[6] N. das Chagas Mendonca and R. de Oliveira Anido, "Using Extended Hierarchical Quorum Consensus to Control Replicated Data: from Traditional Voting to Logical Structures," *Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences*, vol. 27. pp. 303–312, 1994.
[7] A. Burger, V. Kumar, and M. L. Hines., "Performance of Multiversion and Distributed Two-Phase Locking Concurrency Control Mechanisms in Distributed Databases," *Inf. Sci.*, vol. 96, no. 1-2, pp. 129-152, 1997.
[8] J. Gray, et al., "Granularity of Locks and Degrees of Consistency in a Shared Data Base," *IFIP Working Conference on Modelling in Data Base Management Systems*, pp. 365–394, 1976.
[9] S. Gilbert and N. Lynch, "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services," *ACM SIGACT News*, vol. 33, no. 2, pp. 51-59, 2002.
[10] S. Gilbert and N. Lynch, "Perspectives on the CAP Theorem," *Computer*, vol. 45, no. 2, pp. 30-36, 2012.
[11] D. Terry, "Replicated Data Consistency Explained Through Baseball," *Commun. ACM 56*, vol. 12, pp. 82-89, 2013.
[12] W. Vogels, "Eventually Consistent," *Commun. ACM 52*, vol. 52. no. 1, pp. 40-44, 2009.
[13] P. A. Bernstein and S. Das, "Rethinking Eventual Consistency," *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pp. 923-928, 2013.
[14] K. Anshar, N. Suryana, and N. B. Abdullah, "Blind Write Protocol," *International Conference on Intelligent Systems Design and Applications,"* pp. 868-879, 2017.

[15] K. Anshar, N. Suryana, and N. B. Abdullah, "Blind Write Protocol Implementation in Apache Derby Database," *Journal of Information Assurance and Security,* vol. 13, no. 1, pp. 48-55, 2018.
[16] K. Anshar, N. Suryana, and N. B. Abdullah, "The Potential Application of Blind Write Protocol," *International Journal of Computer Information Systems and Industrial Management Applications,* vol. 10, pp. 219-226, 2018.
[17] H. Berenson, et al., "Critique of ANSI SQL Isolation Levels," *ACM Record,* vol. 24, no. 2. pp. 1-10, 1995.
[18] B. Kemme and G. Alonso, "A New Approach to Developing and Implementing Eager Database Replication Protocols," *ACM Trans. Database Syst,* vol. 25, no. 3, pp. 333–379, 2000.
[19] M. Alomari, A. Fekete, and U. Rohm, "A Robust Technique to Ensure Serializable Executions with Snapshot Isolation DBMS," *IEEE 25th International Conference on Data Engineering,* pp. 341-352, 2009.
[20] M. J. Cahill, U. Rohm, and A. Fekete, "Serializable Isolation for Snapshot Databases," *ACM Transactions on Database System,* vol. 34, no. 4, pp. 1-42, 2009.
[21] X. Zhou, et al., "Posterior Snapshot Isolation," *IEEE 33rd International Conference on Data Engineering,* pp. 797-808, 2017.
[22] M. Alomari and A. Fekete, "Serializable Use of Read Committed Isolation Level," *IEEE/ACS 12th International Conference of Computer Systems and Applications*, pp. 1-8, 2015.
[23] F. Zendaoui and W. K. Hidouci, "Performance Evaluation of Serializable Snapshot Isolation in PostgreSQL," *12th International Symposium on Programming and Systems*, pp. 1-11, 2015.
[24] Y. Yuan, et al., "BCC: Reducing False Abort in Optimistic Concurrency Control with Low Cost for in-Memory Databases," *Proc. VLDB Endow.*, vol. 9, no. 6, pp. 504-515, 2016.
[25] J. Hyungsoo, et al., "A scalable Lock Manager for Multicores," *ACM Transaction on Database System,* vol. 39, no. 4, pp. 1-29, 2014.

BIOGRAPHIES OF AUTHORS

Khairul Anshar was born in Garut, West Java, Indonesia on 20 January 1980. He obtained his degree in Physics on from Bandung Institute of Technology, Indonesia. He obtained his Master of Science (MSc) in Information and Communication Technology by research from Faculty of Information and Communication Technology (FTMK), Universiti Teknikal Malaysia Melaka, Malaysia on 2013.sg.linkedin.com/in/khairulanshar.

Prof. Gs. Ts. Dr. Nanna Suryana Herman currently works as a full Professor in Advanced Databases at the Faculty of Information and Communication Technology (FTMK) UTeM. At the same time, he holds the position being the Project Leader for UTeM-Indonesia PhD Program (UIPP). He obtained his degree in Soil and Water Engineering, UNPAD Bandung Indonesia. He obtained his Master of Science (MSc) in Computer Assisted Regional Planning at the International Institute for Geoinformatics and Earth Observation (ITC), Enschede, The Netherlands. In year 1996, he obtained his Doctorate Degree from the Department of Remote Sensing and GIS, Research University of Wageningen, the Netherlnads. He currently supervises Master and Doctorate students who are undertaking research in system interoperability, mobile computing, handing and managing large (spatial) data, 3D imaging and image processing and image analysis as well as indoor navigation. He published numbers of International Journals, book chapters. He is actively involved in Editorial Board of International Journals, member of ASEA UNINET, EURAS, member of AACHA, Royal Academic Singapore, MBOT and IGRSM.

Dr. Noraswaliza Abdullahhird is a senior lecturer in the Department of Software Engineering, teaching programming and database subjects. She is also a member of the faculty's Computational Intelligence Technology Research Group. Her research interests include data mining, recommender system, and database technology. She received her honors degree in Management Information System from Universiti Sains Malaysia, her master's degree in Management Information System from Universiti Putra Malaysia and her PhD in Recommender System area from Queensland University of Technology, Australia. Her PhD work includes exploring user generated contents from the Internet to extract knowledge for recommendation by applying data mining techniques and developing a novel hybrid recommender technique for recommending infrequently purchased products.