International Journal of Computer Information Systems and Industrial Management Applications.
ISSN 2150-7988 Volume 10 (2018) pp. 219-226

Received: 30 Dec, 2017; Accepted: 7 August, 2018; Publish: 17 August, 2018

The Potential Application of Blind Write Protocol

Nanna Suryana[1], Khairul Anshar[2] and Noraswaliza Binti Abdullah[3]

Faculty of Information and Communication Technology,
Universiti Teknikal Malaysia Melaka, Melaka, Malaysia
[1,3] *{nsuryana, noraswaliza}@utem.edu.my*
[2] *p031420004@student.utem.edu.my*

***Abstract*: The current approach to handle interleaved write operation and preserve consistency in relational database system relies on locking protocol. The application system does not have other option to deal with interleaved write operation. In other hand, allowing more write operations to be interleaved will increase the throughput of database but it can result to an inconsistent database state. Since the application system has their own consistency and availability requirement then this paper proposes blind write protocol as a complement to the current concurrency control.**

Since blind write protocol will not lock any entity, then it should use read commited isolation level, auto commit, and request one read operation only to be used in consistency validation. Because, in between two read operations there could be another transaction perform blind write operation to the same entity. These two read operations which access the same entity may return different value

Keywords: Concurrency control, interleaved transaction, locking, consistency, availability, blind write**.**

I. Introduction

Current implementation of concurrency control in Database Management System [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. When any transaction is trying to lock an entity, which is already lock by other transaction, then it should wait or preempt. All the locking and waiting operations are handled in the database system level.

Stearns et al. propose another approach that utilizing a version of entity and certification process [3]. Each version of entity is unique and it is used to identified 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.

Kung et al. in [6] proposed an optimistic approach which utilizing 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 is fail while performing the integrity validation, then it must be restarted.

These 3 concurrency controls above are handling the temporary inconsistent state at the system level. Thus, application system has no option to deal with temporary inconsistent state. In other hand, each application has different consistency and availability requirement. It is developed to fulfill the business requirement which is transformed into read and write operation. Therefore, the application system has the knowledge on how to deal with the consistency.

Moreover, the main objective of the concurrency control is to increase the throughput of database by allowing more operations to be interleaved as many as possible and at the same time deliver the consistency required by the applicaiton. Hence, this paper proposes blind write protocol as a complement of current concurrency control to be applied in the database system. Our motivation is to give the application system a new option to deal with interleaved write operation.

Terry Doug explained in [16], *high availability is not sufficient for most application system, but strong consistency is not needed either*. Vogels argued in [13], *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 [17] that *the high availability increases the application complexity to handle inconsistent data*. Therefore, let the application system decide. If application system wants to preserve strong consistency, then they can use normal write otherwise use the blind write protocol.

We found several discussions about blind write. On 1981, Stearns et al. explained in [7] "*We make the assumption, called the no blind writes assumption, that a process does not issue a write request on a particular entity without first issuing a read request on that entity*." On 1994, Mendonca et al explained in [9] "*In this paper we present a new replica control protocol that logically imposes a hierarchy onto the* *set of copies and introduces the blind write as another operation. During a blind write operation, copies are modified regardless of their previous values; such situation occurs, for instance, in initializations*." On 1997, Burger et all explained in [11] "*One of the significant differences between our work and the works reviewed above is that we have simulated a write as a blind write (a read is not performed before the data item is written)*."

There are also some discussions which aims to allow more operation to be interleaved such as in [10] and [15]. They discussed about Read Committed and Snapshot Isolation. Kemme et al explained in [12] that snapshot isolation with First Committer Wins (FCW) feature can prevent dirty read, lost update, nonrepeatable read, and read skew but it still allows write skew concurrency anomalies. It means, the database management system which use the snapshot isolation still relies on the locking protocol to preserve consistency or to make interleaved transactions are serializable [12].

The latest discussion on the concurrency control is trying to make the snapshot isolation able to prevent write skew concurrency anomaly. In other word, it is trying to make the interleaved transactions become serializable [14] [15] [20]. The discussion on making the interleaved transactions in the Read Committed Isolation become serializable is started in [18]. Their approaches are to abort one of the interleaved transaction to make the Read Committed and Snapshot Isolation becomes serializable if conflict pattern called dangerous structure appears [19].

In the locking protocol, the serializable is achieved by making one transaction wait until the required locked entity is released. The concurrency control will not abort any transaction until the deadlock or timeout occurs. In other hand, serializable snapshot isolation will abort one of the conflict transactions even it is not required by the application system requirement.

Both approches above have same objective that is preserving consistency at any cost and trade off which applied at the database management system level. Hence, application system does not have other option to deal with interleaved write operation. While, this blind write protocol, which will not lock any entity when performing write operation, is proposed to allow more write operations to be interleaved. With the blind write protocol, the application system has another option other than waiting, preempting, or abortion when dealing with interleaved write operations.

The key point here is that the application systems must have more than one option to deal with interleaved write operation. This gives a freedom to the application systems in order to deal with interleaved write operations. As a result, preserving consistency becomes application system responsibility.

To understand more on blind write protocol, we start the discussion by reviewing the concurrency anomaly in Section 2. Then, we describe about blind write protocol and its implementation in next section. The last section concludes the topic.

II. Concurrency Control and Anomaly

The discussion on the concurrency control aims to preserve the consistency by solving the concurrency control anomaly. The more transactions are being processed will increase the throughput of accesses to the database [6], but it can result an inconsistent database state [5]. Therefore, database system requires a concurrency control to handle two or more transactions that access same entity. In the absence of concurrency control, any two or more transactions will have concurrency anomalies. Bernstein et al. in [8] described about two concurrency anomalies, i.e. Lost Update Anomaly and Inconsistent Retrieval.

A. Lost Update Anomaly

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

Seq.	Initial State e_1 = 10;	
	T_1	T_2
1	begin	begin
2	$e_1 \leftarrow e_1 + 10$; ***Temporary Inconsistent State e_1= 20;***	$e_1 \leftarrow e_1 + 30$; ***Temporary Inconsistent State e_1= 40;***
3	commit;	commit**;**
4	end;	end;
	Final State can either e_1 = 20 or e_1 = 40	

Figure 1. Lost Update Anomaly

On Figure 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 to item on the left. In the absence of concurrency control, the final state can either e_1=20 or e_1=40. This result is known as lost update anomaly. Therefore, in order to preserve consistency then the DBMS requires a concurrency control to handle these 2 interleaved write operations coming from different transaction. The locking protocol will make either T_2 wait until T_1 is completed or vice versa. Thus, the final state will be consistent i.e. e_1 is equal to 50.

B. Inconsistent Retrieval Anomaly

To illustrate this anomaly, let say there are two interleaved transactions T_1 and T_2 are executed at the same time as shown on Figure 2. At the time T_2 displays/ prints the value of x then it still shows the initial value of e_1, i.e. 10, which is different with T_1.

Seq.	Initial State e_1 = 10;	
	T_1	T_2
1	begin	Begin
2	$e_1 \leftarrow e_1$+10; ***Temporary Inconsistent State e_1= 20;***	x $\leftarrow e_1$;
3	commit**;**	print x: 10
4	end;	end;
	Final State is e_1 = 20	

Figure 2. Inconsistent Retrieval Anomaly

At the end of these two transactions, the final state is still correct, i.e. e_1=20. But, if T_2 or any others transaction use the value of x, then application system may experience the lost update anomaly.

C. Write Skew Anomaly

To illustrate this anomaly, let say there are two interleaved transactions T_1 and T_2 are executed at the same time as shown on Figure 3. The initial balance of e_1 is 100 and e_2 is 50. The application has requirement or constraint that the e_1+e_2 should always be greater or equal to 0. If T_1 is withdrawing money from e_1 with amount is 100 and T_2 is withdrawing money from e_2 with amount 60, then total amount is greater then e_1+e_2. Since both transaction will pass the validation in the Seq. no 3 as shown in Figure 3, then final state e_1+e_2 will be less than 0. This condition against the requirement or constraint.

Seq.	Initial State e_1 = 100; e_2 = 50; e_1+e_2=150; Constraint: e_1+e_2 >= 0.	
	T_1	T_2
1	begin	begin
2	x_withdraw ← 100;	x_withdraw ← 60;
3	if (e_1+e_2>=x_withdraw) then	İf (e_1+e_2>=x_withdraw) then
4	e_1 ← e_1 - x_withdraw;	e_2 ← e_2 - x_withdraw;
	Temporary Inconsistent State e_1= 0;	***Temporary Inconsistent State e_2= -10;***
5	commit;	commit;
6	end if;	end if;
7	end;	end;
	Final State $e_1 + e_2$ = -10, it is contradictory with the above constraint.	

Figure 3. Write Skew Anomaly

Gray et al. in [4], Berenson et al. in [10] and Kemme et al. in [12] discussed about the concurrency anomalies and different isolation level. The read uncommitted, read committed, and snapshot isolation were proposed to improve the concurrency. But the concurrency control still relies on locking to make interleaved write operations become serializable.

These concurrency anomalies and different read protocols with their weakness and limitation give us the base knowledge. It becomes an important information to establish and develop an algorithm that can preserve consistency in blind write protocol.

III. Blind Write Protocol

The blind write protocol is proposed as a complement to allow more write operation to be interleaved and transaction should not lock any entity and no transaction should be restarted. Since the blind write protocol is a complement then the application system has another option to perform write operation. If application system does not want to create their own specific approach to achieve consistency, then it can use normal write protocol to achieve the consistency. Moreover, since two write operations, i.e. normal and blind, can be used together, then the blind write protocol should be able to make them work together.

There will be three combinations if two interleaved write operations are writing to same entity and they are executed at the same time, i.e.:

1. both are using normal write protocols
2. one transaction is using normal and another one is using blind write protocols
3. both are using blind write protocol

Point no. 1 above is clear. Normal write protocol is using locking protocol to preserve consistency. Since both are using locking protocol, then one transaction should wait for or preempt from other transaction. Before we discuss point no. 2, let discuss point no. 3 first. Because, we should know whether the application system can create and develop their own approach to prevent the lost update and write skew anomaly when two blind operations executed at the same time.

A. Two Interleaved Operations are Using Blind Write Protocol

To begin with, let start with making proper definition and its principal of blind write operation. This definition is related to database system discussed in [2], [3], and [6] which refers to [1]. Interaction between client and database system is known as transaction. The content of interaction consists of one or more operations. The operation can be read or write. Write operation is an action to create new entity, modify or delete existing entity value. Read operation is an action to get entity value, it can be uncommitted or committed value as discussed in Section 2.

1) Blind Write Definition

Before we discuss more detail on how to handle 2 or more interleaved transactions that use blind write protocol, we need to give proper definition on database system. We define database system as D which consists of n number of entity.

$D = \{e_1, e_2, e_3, \ldots, e_n\}$ (1)

These entities can be either tables, rows, or columns. This paper is focusing on the Data Manipulation Language (DML) protocol, which create, modify or delete a row into, in, or from a table. The Data Definition Language (DDL) is not part of our paper scope. 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. We use ← notation as assigning a value on the right to entity on the left as discussed on Section 2.

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. (2)

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

$e_i \leftarrow$ NULL; where $1 < i < n$. (3)

Modify/ update operation is considered as assigning a value, v, to existing entity, e_i,

$e_i \leftarrow v$; where v is not NULL and $1 < i < n$. (4)

The value of v above can be defined as:

1. function of any entity, e_j. It is known as normal write operation. Therefore,

$e_{n+1} \leftarrow f(ej)$; where $1 < j < n$. (5)
$e_i \leftarrow f(e_j)$; where $1 \leq i \leq n$, and $1 \leq j \leq n$. (6)
If i=j then it means the new value depends on the initial value of entity

2. Constant or Fixed value, e.g. ‘APPROVED’, ‘536980 MALAYSIA’, ‘+6012345678’, 20, etc. It is known as blind write operation. The Constant or Fixed value should not be NULL. Therefore,

$e_{n+1} \leftarrow c$; where c is fixed value and c is not NULL. (7)
$e_i \leftarrow c$; where $1 < i < n$ and c is fixed value and c is not NULL. (8)

Since delete operation is considered as assigning NULL to the entity, then there is no different between normal and blind write protocol. The main different between them is that blind write protocol will not apply any locking to any entity. Based on Bernstein argument in [20] that the high availability increases the application complexity to handle inconsistent data. One concrete example is handling lost update and write skew anomaly.

2) Achieving the Consistency using Blind Write Protocol

The example of lost update and write skew anomaly can be seen in Section 2. In that example, it is utilizing one entity only to handle and maintain the operation. The entity in that example is considered as a table. To give more explanation please see Table 1 below. It is a balance table consist of one entity, in this case the entity is a row, with 4 columns i.e. account_id, account_number, balance_amount and last_updated_date.

Table 1. Balance Table.

Account_id	***Account_number***	***Balance_amount***	***Last_updated_date***
1	1234-567-890	1000	10-Jan-1980 00:00:01

As explained above, the value of blind write operation should be a fixed value, c. Therefore, one table is not enough to preserve the consistency using blind write protocol. To achieve that, then it required at least one table to handle and maintain historical write operation as can be seen on Table 2.

Table 2. History Table.

History_id	***Account_id***	***Transaction_amount***	***Transaction_date***	***Status***
0	1	1000	10-Jan-1980 00:00:01	approved
1	1	100	20-Jan-1980 00:00:01	approved
2	1	300	20-Jan-1980 00:00:01	approved

The history table has foreign key of balance table, i.e. account_id. For deposit operation then the transaction_amount should be greater than 0. For withdraw operation, the transaction_amount should be less than 0. The transaction_date is used to record the time stamp when the operation is committed. The status is used to differentiate whether the operation is approved or rejected. The status will be set to rejected if the operation for particular account_id do not meet with specific constrain. The history_id is primary key of the history table, it is a running number generated from sequence object. Two or more entity (row) may have same transaction_date but they should have unique history_id value. Using history table, there will be no aborted transaction. All the operation from all transactions will be recorded in this table as one entity (record). The balance_amount of particular account_number in the balance table is aggregation of transaction_amount in history table which has same account_id and the status should be approved. To achive the consistency using blind write operation, then we need to discuss the possibility of interleaved write operation combinations, i.e.:

1. both operations are deposit
2. both operations are withdrawal
3. one operation is deposit and another one is withdrawal

a) Both Operations are Deposit

Let say the entity of balance and history table is eb and et respectively. It may consist of many account_id. To indicate account_id=1, we use eb1 and et[1]. The update operation value of eb is (balance_amount, last_updated_date) and for insert operation of et[1] is (history_id, account_id, transaction_amount, transaction_date, status). If two transactions, T_1 and T_2, are using blind write protocol and executed at the same time follow the same step, as can be seen on Figure 4, then the result can be same if they are executed one by one, either T_1 first or T_2.

Seq.	Initial State e_{b1} = 1000;	
	T_1	T_2
1	begin	begin
2	seq_id $\leftarrow$ history_seq.nextval;	seq_id $\leftarrow$ history_seq.nextval;
3	$e_t[1]_{n+1} \leftarrow$ (seq_id,1,100,sysdate,’approved’);	$e_t[1]_{n+1} \leftarrow$ (seq_id,1,300,sysdate,’approved’);
4	$e_{b1} \leftarrow (\{\sum_{j=1}^{n} e_t[1]_j$(transaction_amount); where status=’approved’}, sysdate);	$e_{b1} \leftarrow (\{\sum_{j=1}^{n} e_t[1]_j$(transaction_amount); where status=’approved’}, sysdate);
5	end;	end;
	Final State e_{b1} = 1400;	

Figure 4. The Aggregation

To achieve this then there are some conditions need to be applied as follows:

1. the read operation should use read committed isolation level
2. it should apply auto commit on each write operation to prevent the lost update anomaly

The first condition is clear. It was explained on the previous section. To show that condition no. 2 is required then let say there are two commit operations. The first commit is between seq. no. 3 and 4 and the second one is between seq. no 4 and 5. The sequence of operation is as follow:

T_1[seq. no. 1] → T_1[seq. no. 2] → T_1[seq. no. 3]→ T_1[commit] → T_1[seq. no. 4] → T_2[seq. no. 1] → T_2[seq. no. 2] → T_2[seq. no. 3]→ T_2[commit] → T_2[seq. no. 4] → T_2[commit] → T_1[commit]

Since the T_1[seq. no. 4] has not been committed then the T_2[seq. no. 4] and the second T_2[commit] will be overwritten by the second T_1[commit] which will eventually experience the lost update anomaly. Therefore, to prevent the lost update anomaly the blind write protocol should apply auto commit.

Since it is using auto commit and balance_ammount of eb1 is calculated by summing up all transaction_amount of et[1], then it always gives the latest result, regardless T_1 is executed first or T_2.

b) *Both Operations are Withdrawal*

From pevious section, we find that the blind write protocol can handle lost update anomaly without lock any entity. The example above is involving deposit operation only. But how if both operations are withdrawal and it must be in accordance with certain rules as follow:

1. the balance_amount should not be minus
2. the operation should not be rejected if the balance_amount is greater or equal than absolute(transaction_amount). The withdrawal amount is always less than 0

To discuss this, let say the current balance amount eb1=1000 as shown in Table 1 above. We set two interleaved transactions, T_1 and T_2, and execute at the same time. These transactions are performing withdrawal operation respectively with different scenarios as follows:

1. -100 and -300. Since 1000-100-300>0 then both should not be rejected
2. -900 and -500. Since 1000-900-500<0 and 1000-900>0 and 1000-500>0 then one of them should be rejected and the other one should be approved
3. -1100 and -900. Since 1000-1100-900<0 and 1000-1100<0 and 1000-900>0 then T1 should be rejected and T2 should be approved
4. -1100 and -1200. Since 1000-1100-900<0 and 1000-1100<0 and 1000-1200<0 then both transactions should be rejected

To handle all the scenarios above, we introduce 2 functions. The first function is simple function used to get account_id for specific account number from the balance table. The second function has 2 input arguments, i.e. account id and transaction amount. It has one output either true or false. This discussion is focusing more on the second function, we do not explain the first function in detail.

Let name the second function as transact. We modify the steps in Figure 4 above to implement both functions as shown in Figure 5 below. The transact function is shown in Figure 6.

Seq.	Initial State e_{b1} = 1000; account_no = '1234-567-890';	
	T_1	T_2
1	begin	begin
2	v_acct_id ← ***getAccountId***(account_no)	v_acct_id ← ***getAccountId***(account_no)
3	if (***transact***(v_acct_id, -100)) then	if (***transact***(v_acct_id, -300)) then
4	e_{b1} ← ({$\sum_{j=1}^{n} e_t[1]_j$(transaction_amount); where status='approved'}, sysdate);	e_{b1} ← ({$\sum_{j=1}^{n} e_t[1]_j$(transaction_amount); where status='approved'}, sysdate);
5	end if;	end if;
6	end;	end;
	Final State e_{b1} = 600;	

Figure 5. The Aggregation with Transac Function

The explanation of transact function is as follows:

- Seq. no. 1 defines function name, its input argument and output
- Seq. no. 2 begins the function
- Seq. no. 3 gets history id from sequence object and put into seq_history_id. It is running number
- Seq. no. 4 sets default value of v_status to 'approved'. If both operations are Deposit, there is no any validation required since it will not make the balance amount become negative. Hence, the status should always be approved
- Seq. no. 5 assigns v_status value to 'not approved' if a_transaction_amount is minus (withdrawal operation)
- Seq. no. 6 inserts new record to History table with history_id value is seq_history_id. The operations from seq. no. 3 until 6 can be executed as one statement by utilizing output in insert statement and decode clause. So, it can be treated as one operation. The example of DML statement for these operations is:

  insert into history values (history_seq.nextval, a_account_id, a_transaction_amount, sysdate, decode((a_transaction_amount/abs(a_transaction_amount)), 1, 'approve', 'not approve')) returning history_id into seq_history_id;

The *returning history_id into seq_history_id* is used for seq. no. 3.

decode((a_transaction_amount/abs(a_transaction_amount)), 1, 'approve', 'not approve') is used for seq. no. 4 and 5.

Seq.	Transact Function
1	Function ***transact*** (**a_account_id** number, **a_transaction_amount** number) return boolean
2	begin
3	seq_history_id ← history_seq.nextval;
4	v_status ← 'approved';
5	if (**a_transaction_amount** <0) v_status ← 'not approved';
6	$et[1]_{n+1}$ ← (seq_history_id, **a_account_id**, **a_transaction_amount**, sysdate, v_status);
7	v_sysdate ← get transaction_date from history table where history_id = seq_history_id;
8	if (**a_transaction_amount** >=0) then return true;
9	else
10	v_array ← get {$\sum_{j=1}^{n} e_t[1]_j$(transaction_amount); where status='approved'} union [$e_t[1]_k$; where 1 $\leq$ k $\leq$ n and $e_t[1]_k$(status) ='not approved' and sysdate <= v_sysdate] order by transaction_date and history_id;
11	v_balance ← v_array[0](amount);
12	v_success ← false;
13	for (i=1; i<length(v_array); i++) then
14	v_amount ← v_array[i] (amount);
15	v_history_id ← v_array[i] (history_id);
16	if (v_history_id = seq_history_id) then
17	if (v_balance + v_amount >=0) then
18	v_balance ← v_balance + v_amount;
19	v_success ← true;
20	$et[1]_{seq\_history\_id}$ (status) ← 'approved';
21	else
22	$et[1]_{seq\_history\_id}$ (status) ← 'rejected';
23	end if;
24	else
25	if (v_balance + v_amount >=0) then
26	v_balance ← v_balance + v_amount;
27	end if;
28	end if;
29	end for;
30	return v_success;
31	end if;
32	end;

Figure 6. Transact Function

- Seq. no. 7 gets transaction_date from the history table where history_id is equal to seq_history_id. The example of DML statement for this operation is:

  select transaction_date into v_ sysdate from history where history_id=seq_history_id;

- Seq. no. 8 determines whether the operation is deposit or withdrawal. If a_transaction_amount > 0 then end the function and return true. Otherwise, then it continues to Seq no. 9. It means for deposit operaion, it does not need any further validation.

- Seq. no. 9 is else condition
- Seq. no. 10 gets collection of history records for specific account_id. The example of DML statement for this operation is:

select min (transaction_date), -1 history_id, sum(transaction_amount) transaction_amount from history where account_id= a_account_id and status='approved'
union
select transaction_date, history_id, transaction_amount from history where account_id= a_account_id and status='not approved' and transaction_date <=v_sysdate order by transaction_date, history_id;

This DML statement is utilizing 'union' that will be executed as one operation. If it does not use 'union' in the statement above, then the DML will become two statements (operations) as follows:
DML statement 1:
select min (transaction_date), -1 history_id, sum(transaction_amount) transaction_amount from history where account_id= a_account_id and status='approved';

DML statement 2:
select transaction_date, history_id, transaction_amount from history where account_id= a_account_id and status='not approved' and history_id<=seq_history_id;

Moreover, if there is blind write operation, which update the status from 'not approved' to either 'approved' or 'rejected', between these two DML statements then it will affect sum(transaction_amount) in DML statement 1 and the collection of records for DML statement 2. This will end with lost update anomaly. Therefore, the third condition requierd by blind write protocol is:

Since blind write protocol will not lock any entity, then the transaction should request one read operation to be used in validation to prevent write skew anomaly.

Table 3. History Table with Withdrawal Operation.

History_id	***Account_id***	***Transaction_amount***	***Transaction_date***	***Status***
0	1	1000	10-Jan-1980 00:00:01	approved
1	1	-900	20-Jan-1980 00:00:01	not approved
2	1	-500	20-Jan-1980 00:00:01	not approved

To prove this, let execute T_1 and T_2 at the same time. T_1 is a transaction with history_id=1 and T_2 is a transaction with history_id=2 as shown on Table 3. T_1 has executed DML statement 1 and it returns 1000. Then T_2 is executing seq. no. 11 and 20 (it updates and auto commits T_2 status become 'approved' see Figure 3). If T_1 continue to execute DML statement 2 then it will return one record only since T_2 status has become 'approved'. Thus, T_1 status will be updated and auto committed to 'approved' also because the validation 1000-900>0. Now, update Balance table as shown in Figure 5 seq. no. 4. It shows that the balance amount will become 1000 -900 -500 = -400 since T_1 and T_2 was updated as 'approved'.

- Seq. no. 11 assigns sum(transaction_amount) value of approved status to v_balance
- Seq. no. 12 sets default value of v_success to false
- Seq. no. 13 until 29 validates the transaction amount with balance amount
- Seq. no. 32 returns the validation result. If the history status is updated and committed to 'rejected' then it returns false, otherwise it returns true.

c) Combination of deposit and wirdrawal operation

There are no significant obstacles with the deposit operation in this combination. Likewise, with the withdrawal operation. The main obstacle with this combination is about the timing.
As explained above that the seq. no. 10 operation is fetching collection of history record which ordered by transaction_date and history_id. If two transactions have same transaction_date then it will be ordered by history_id which is unique for each history record.

B. Two Interleaved Operations are Using Normal and Blind Write Protocol

Until this section, we have already shown that blind write protocol can preserve the consistency in different approach with the normal write protocol. For two or more transactions that use different protocol, then they are two options. First option is the blind write protocol should wait until the locked entity is released. The second option is the blind write protocol should not wait other transaction to release the lock on the entity. These options provide more choice to the application to determine which one suits with the business requirements. This wait and no wait option should be applied in the DML statement along with blind write option.
The wait option will work for blind write protocol to wait until the locked entity is released. Since the blind write protocol will not lock any entity then the normal write protocol can start to perform any operation including lock any entity at any time. Once the entity is locked then any write operation that wants to access the locked entity, including blind write with wait option, should wait or preempt.

C. DML Statement of Blind Write Operation

We propopse a set of DML statements that can be used to determine whether the blind write protocol should wait or not as well as to distinguish the blind write protocol with the normal write protocol. These DML statements for blind write protocol as follows:

1. DML statement for blind write protocol with wait option.
   a. Insert Statement:
   BLIND INSERT INTO *table_name (list_of_columns)*

***VALUES** (list_of_values) **WITH WAIT**;*

b. Update Statement:
***BLIND UPDATE** table_name*
***SET** column_name = value [, column_name = value]*
*[**WHERE** condition] **WITH WAIT**;*

c. Delete Statement:
***BLIND DELETE** table_name*
*[**WHERE** condition] **WITH WAIT**;*

This wait option will only work for blind write protocol to wait until the locked entity is released. Since the blind write protocol will not lock any entity then the normal write protocol can start to perform any operation including lock any entity at any time. Once the entity is locked then any write operation that want to access the locked entity, including blind write, should wait or preempt if blind write is using wait option.

2. DML statement for blind write protocol without wait option.
   a. Insert Statement:
   ***BLIND INSERT INTO** table_name (list_of_columns)*
   ***VALUES** (list_of_values) **WITHOUT WAIT**;*
   b. Update Statement:
   ***BLIND UPDATE** table_name*
   ***SET** column_name = value [, column_name = value]*
   *[**WHERE** condition] **WITHOUT WAIT**;*
   c. Delete Statement:
   ***BLIND DELETE** table_name*
   *[**WHERE** condition] **WITHOUT WAIT**;*

IV. Summary

This paper proposes blind write protocol as a complement of current concurrency control to give more option to the application on dealing with the interleaved write operation. The blind protocol provides more option besides wait or preempt. The blind write protocol also can be used together with normal write operation with wait or no wait option.

Since, the blind write operation does not use locking protocol, then the database system will experience a lost update and write skew anomaly. Therefore, the blind write protocol should apply their own approach to prevent these anomalies. To achieve this, there are some conditions need to be applied in the transaction as follows:

1. the read operation should use read committed isolation level
2. it should apply auto commit on each write operation to prevent the lost update anomaly
3. the transaction should request one read operation to be used in validation to prevent write skew anomaly.

References

[1] E. F. Codd "A Relational Model of Data for Large Shared Data Banks". Commun. ACM 13, pp. 377-387, 1970.

[2] K. P. Eswaran, J. N. Gray, R. A. Lorie, and I. L. Traiger "The Notions of Consistency and Predicate Lock in a Database System". In: ACM Comput. Surv, vol 19, pp. 624-633, 1976.

[3] Stearns, R. E., Lewis, P. M., Ii, And Rosenkrantz, D. J. "Concurrency Control for Database Systems". *In Proc. 7th Symp. Foundations of Computer Science*, pp. 19-32, 1976.

[4] J. Gray, R. A. Lorie, G. R. Putzolu, and I. L. Traiger "Granularity of Locks and Degrees of Consistency in a Shared Database". In IFIP Working Conference on Modelling in Database Management Systems, pp. 365–394, 1976.

[5] Philip A. Bernstein, David W. Shipman, and Wing S. Wong "Formal Aspects of Serializability in Database Concurrency Control". In IEEE, vol SE-5, No. 3, 1979.

[6] Kung, H. T., Androbinson, J.T. "An Optimistic Methods for Concurrency Control". In ACM Trans. Database Syst. 6, 2, pp. 213-226, 1981.

[7] Stearns, R. E., Rosenkrantz D. J. "Distributed Database Concurrency Controls Using Before-values". *In Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data*, 1981.

[8] Philip A. Bernstein, Nathan Goodman "Concurrency Control in Distributed Database Systems". In ACM Computing Surveys (CSUR), v.13 n.2, pp. 185-221, 1981.

[9] N. das Chagas Mendonca and R. de Oliveira Anido "Using Extended Hierarchical Quorum Consensus to Control Eeplicated Eata: From Traditional Voting to Logical Structures". *In Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences*, Wailea, HI, USA, pp. 303-312, 1994.

[10] Hal Berenson, Phil Bernstein, Jim Gray, Jim Melton, Elizabeth O'Neil, and Patrick O'Neil "A critique of ANSI SQL isolation levels". *In Proceedings of the 1995 ACM SIGMOD International Conference on Management of data (SIGMOD '95)*, Michael Carey and Donovan Schneider (Eds.). ACM, New York, NY, USA, pp. 1-10, 1995.

[11] Albert Burger, Vijay Kumar, and Mary Lou Hines "Performance of Multiversion and Distributed Two-phase Locking Concurrency Control Mechanisms in Distributed Databases". In Inf. Sci. 96, 1-2, pp. 129-152, 1997.

[12] Bettina Kemme and Gustavo Alonso "A New Approach to Developing and Implementing Eager Database Replication Protocols". In ACM Trans. Database Syst. 25, pp. 333-379, 2000.

[13] Werner Vogels: Eventually consistent. Commun. ACM 52, pp. 40-44, 2009.

[14] M. Alomari, A. Fekete and U. Röhm "A Robust Technique to Ensure Serializable Executions with Snapshot Isolation DBMS". *In IEEE 25th International Conference on Data Engineering*, Shanghai, 2009, pp. 341-352, 2009.

[15] J. Cahill, U. ROHM and A. D. Fekete "Serializable Isolation for Snapshot Databases". In ACM Transactions on Database System, vol. 34, no. 4, 2009.

[16] Doug Terry "Replicated Data Consistency Explained Through Baseball". Commun. ACM 56, 12, pp. 82-89, 2013.

[17] Philip A. Bernstein and Sudipto Das. "Rethinking Eventual Consistency". *In Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD '13)*. ACM, New York, NY, USA, pp. 923-928, 2013.

[18] M. Alomari and A. Fekete "Serializable use of Read Committed Isolation Level". *In 2015 IEEE/ACS 12th International Conference of Computer Systems and Applications (AICCSA)*, Marrakech, 2015, pp. 1-8, 2015.
[19] F. Zendaoui and W. K. Hidouci "Performance Evaluation of Serializable Snapshot Isolation in PostgreSQL". In 2015 12th International Symposium on Programming and Systems (ISPS), Algiers, pp. 1-11, 2015.
[20] X. Zhou, Z. Yu and K. L. Tan "Posterior Snapshot Isolation". *In 2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, San Diego, CA, pp. 797-808, 2017.

Author Biographies

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.

Prof. 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 Manager of COE for Center of Advanced Computing Technology (C-ACT), Center of Research and Innovation Management (CRIM), UTeM. 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.

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.

Dr. Noraswaliza Abdullah 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 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.