Bugs found in Database Management Systems

Keqiang Li

School of Data Science & Engineering, East China Normal University, Shanghai, China

Leopard has successfully discovered 21 transactional bugs from real-world production-level DBMSs, including 4 bugs in MySQL, 2 bugs in PostgreSQL, 11 bugs in TiDB, 2 bugs in OpenGauss, and 2 bugs in a series of commercial DBMSs. We are thankful to the DBMS developers for responding to our bug reports and fixing all the bugs that we found.

Unique fixed bugs

TiDB

Dirty write of SI

Data Found:

2019/12/12

Severity:

(S1)Critical

Test Case:

Transaction ID	Operation Start Timestamp	Operation End Timestamp	Operation Detail	State
Schema Creation			Create table table_7_2(a int primary key, b int, c double);	Success
Database Population			Insert into table_7_2 values(676, 5012153, 2240641.4);	Success
739	104865095693539	104865097351053	Update table_7_2 set b=-5012153, c=2240641.4 where a=676	Success
723	104865111029861	104865114503063	Update table_7_2 set b=852150 where a=676	Success
739	104865115	104865118	Commit	Success

	133146	426143		

Transaction 739 writes a record(676) of table_ 7_2. Transaction 723 also writes this record before transaction 739 was committed, resulting in dirty write.

Read inconsistency of SI

Data Found:

2019/12/14

Severity:

(S1)Critical

Test Case:

Transaction ID	**Operation Start Timestamp**	**Operation End Timestamp**	**Operation Detail**	**State**
Schema Creation			Create table table_7_2(primarykey int primary key, attribute1 double,attribute6 double);	Success
Database Population			Insert into table_7_2 values(3873, 0.213, 0.234);	Success
904	105092107947136	105092115337912	Update table_8_2 set attribute6=-0.386 where primarykey=3873	Success
904	105092144188546	105092148615512	Commit	Success
No other transaction writes the record(3873) of table_8_2 from 105092144188546 to 105092186223727				
914	105092149615124	105092170650214	Set @@global.tx_isolation='REPEATABLE-READ';	Success
907	105092111994965	105092150997653	Update table_8_2 set attribute6=0.484 where primarykey=3873	Success
907	105092182650339	105092186223727	Commit	Success
No other transaction writes the record(3873) of table_8_2 from 105092182650339 to 105092189561012				
914	105092187673511	105092189561012	Select attribute1 from table_8_2 where primarykey=3873	Success
914	105092189	105092191	Select attribute6 from	Success(attribute6

	611217	263618	table_8_2 where primarykey=3873	=-0.368)

In the table above, for the record(3873) of table_ 8_2, there are two historical versions of attribute6, the first is -0.386, the creation time is (10509214418854610509214861551 2); the second is 0.484, the creation time is (10509218265039105092186223727). The start timestamp of transaction 914 is in the interval (10509218767351050921912636 18), then the attribute6 of transaction 914 reading record 3873 should be 0.484, but TiDB returns -0.386, indicating that there is a problem with the consistency reading of TiDB.

Schema version check error

Data Found:

2020/01/01

Severity:

(S2)Serious

Transaction ID	Operation Start Timestamp	Operation End Timestamp	Operation Detail	State
712	107685099231231	107685110245123	Drop db0.table_1_2	Success
723	107685095692321	107685097353242	Select * from db1.table_5_1 where primarykey=2114	Success
No other transaction modify schema of db1				
723	107685111022412	107685114502321	Update db1.table_5_1 set attribute2=8132130 where primarykey=6123	Exception(Information schema is changed)

The first line modifies db0's schema information, and the fourth line modifies db1's data with exception: information schema is changed, which is a bug.

Timestamp acquisition mechanism error of RC

Data Found:

2020/03/01

Severity:

(S1)Critical

Transaction ID	Operation Start Timestamp	Operation End Timestamp	Operation Detail	State
232	112242421212321	112242421786874	Select * from table_2_1 where primarykey=4323	Stall(never response)

Under the RC isolation level recently developed by TiDB team, in order to optimize the performance of timestamp acquisition, asynchronous timestamp acquisition mechanism is

adopted, but there are internal problems in this mechanism, as shown in the above table.

Update BLOB data error

Data Found:

2020/05/02

Severity:

(S3)Critical

Test Case:

Transaction ID	Operation Start Timestamp	Operation End Timestamp	Operation Detail	State
1	2020-08-05 15:52:27.477	2020-08-05 15:52:27.484	Update tablecsacas0 set attributeqwdcwq3=FILE(“./data_case/obj/12obj_file.obj”) where primarykeycqwda0 = 15363173 and primarykeycqwda1 = 940396828 and primarykeycqwda2 = 1209414904	Success
1	2020-08-05 15:52:27.495	2020-08-05 15:52:27.501	Update tablecsacas0 set attributeqwdcwq3=FILE(“./data_case/obj/12obj_file.obj”) and other column where primarykeycqwda0 = 15363173 and primarykeycqwda1 = 940396828 and primarykeycqwda2 = 1209414904	Success
1	2020-08-05 15:52:27.508	2020-08-05 15:52:27.512	Select attributeqwdcwq3 and other column from tablecsacas0 where primarykeycqwda0 = 15363173 and primarykeycqwda1 = 940396828 and primarykeycqwda2 = 1209414904 for update	Success and Return attributeqwdcwq3 = NULL (ERROE)

For BLOB data type, when the new value and the old value written by the update operation are for the same binary file, the value actually written is null and success is returned.

The long lock of the FOR UPDATE statement and the long lock of the UPDATE statement are not mutually exclusive

Data Found:

2020/05/25

Severity:

(S1)Critical

Test Case:

```
drop database if exists db1;
create database db1;
use db1;
create table t1(a int primary key, b int);
create table t2(a int primary key, b int, constraint fk1 foreign key(b) references t1(a));
create view view0(t2_a,t2_b,t1_b) as select t2.a,t2.b,t1.b from t2,t1 where t2.b=t1.a;

insert into t1 values(1,2);
insert into t1 values(2,3);
insert into t1 values(3,4);
insert into t1 values(4,5);
insert into t1 values(5,6);

insert into t2 values(1,2);
insert into t2 values(2,3);
insert into t2 values(3,4);
insert into t2 values(4,5);
insert into t2 values(5,1);
```

So the status of view0 is

```
t2_a,t2_b,t1_b
1,2,3
2,3,4
3,4,5
4,5,6
5,1,2
```

	Session1	**Session2**
1	Begin	
2		Begin
3	update t1 set b=12 where a=1;	
4		select * from view0 where t2_a>3 for update; +------+------+------+ | t2_a | t2_b | t1_b |

		+------+------+------+ | 5 | 1 | 2 | | 4 | 5 | 6 | +------+------+------+
5		Commit;(Success)
6	Commit;(Success)	

At third line TiDB locks the records of table t1 a = 1 until the sixth line releases the lock. Due to the nature of exclusive locks, the fourth line's attempt to acquire a lock on the record with a = 1 in table t1 is blocked, but TiDB grants the session2 lock.

The update statement locks data that does not exist

Data Found:

2020/06/12

Severity:

(S1)Non-Critical

Test Case:

Drop database if exists db;

Create database db;

Use db;

Create table t(a int primary key, b int);

	Session1	**Session2**
1	Begin	
2		Begin
3	Update t set b=314 where a=1;(empty)	
4		Insert into t values(1,3);(blocking)
5	Commit;(success)	Insert into t values(1,3);(blocking)
6		Insert into t values(1,3);(success)
7		Commit;(Success)

The write operation of TiDB reads the latest submitted data, only locks the data that meets the conditions, but does not avoid the phantom (although the read operation can avoid the phantom through MVCC), then the write operation of the third line above will not lock the data, but in fact, TIDB locks it, blocking the insertion operation of another transaction.

JDBC ResultSetMetaData.getColumnName for view query returns the attribute name defined in the table instead of the one defined in the view #24227

DBx

Create View Error

When creating a correct view defined in scenario schema2.sql, an error that cannot be imported may occur, and the error message "duplicate column name" will be reported, as shown in the following. However, after careful inspection, there are no duplicate column names in the statement, and the same DDL statement can run normally on MySQL 5.7.

```
Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statemen
t.

MySQL [(none)]> drop database db0;
Query OK, 0 rows affected (0.476 sec)

MySQL [(none)]> create database db0;
Query OK, 1 row affected (0.753 sec)

MySQL [(none)]> use db0;
Database changed
MySQL [db0]> source schema2.sql
Query OK, 0 rows affected (1.758 sec)

Query OK, 0 rows affected (4.183 sec)

Query OK, 0 rows affected (1.594 sec)

Query OK, 0 rows affected (3.553 sec)

Query OK, 0 rows affected (3.523 sec)

Query OK, 0 rows affected (2.018 sec)

Query OK, 0 rows affected (4.164 sec)

Query OK, 0 rows affected (4.131 sec)

Query OK, 0 rows affected (4.074 sec)

ERROR 1060 (42S21) at line 10 in file: 'schema2.sql': Duplicate column name
'coAttr0_0'
```

Unique confirmed bugs

TiDB

Query Error in information_schema.slow_query#28069

MySQL

Update BLOB data error

Data Found:

2020/05/02

Severity:

(S3)Critical

Test Case:

Transaction ID	Operation Start Timestamp	Operation End Timestamp	Operation Detail	State
1	2020-08-05 15:52:27.477	2020-08-05 15:52:27.484	Update tablecsacas0 set attributeqwdcwq3=FILE("./data_case/obj/12obj_file.obj") where primarykeycqwda0 = 15363173 and primarykeycqwda1 = 940396828 and primarykeycqwda2 = 1209414904	Success
1	2020-08-05 15:52:27.495	2020-08-05 15:52:27.501	Update tablecsacas0 set attributeqwdcwq3=FILE("./data_case/obj/12obj_file.obj") and other column where primarykeycqwda0 = 15363173 and primarykeycqwda1 = 940396828 and primarykeycqwda2 =	Success

			1209414904	
1	2020-08-05 15:52:27.508	2020-08-05 15:52:27.512	Select attributeqwdcwq3 and other column from tablecsacas0 where primarykeycqwda0 = 15363173 and primarykeycqwda1 = 940396828 and primarykeycqwda2 = 1209414904 for update	Success and Return attributeqwdcwq3 = NULL (ERROE)

For BLOB data type, when the new value and the old value written by the update operation are for the same binary file, the value actually written is null and success is returned.

Predicate Lock ERROR

Create Table t(a int, b int, c int, primary key(a,b));
Insert into t values(1,2,3);
Insert into t values(2,4,5);

Session1	Session2
set session transaction isolation level serializable;	
	set session transaction isolation level serializable;
Start Transaction;	
	Start Transaction;
Select a from t;	
	Update t set b=123 where a=1 and b=2;

The update statement of session2 can be executed successfully before the transaction of session1 ends

RU transaction read the result of a failed write operation

Create Table t(a int primary key, b int);
Insert into t values(1,2);
Insert into t values(2,4);

Session1	Session2	Session3
	Begin;	
	set session transaction isolation level read uncommitted;	
		Begin;
		set session transaction

		isolation level read uncommitted;
Begin;		
set session transaction isolation level read uncommitted;		
	Delete from t where a=1;	
		Update t set b=321 where a=2;
	Update t set b=1421 where a=2;	
Select * from t where a=1; Query Result: (1,1231)**ERROR**		Insert into t value(1,1231) Deadlock

Select * from t where a = 1 of session1 and insert into t value (11231) of session3 need to be done in parallel to cause problems.

PostgreSQL

Write skew of SSI

Data Found:

2020/07/25

Severity:

(S1)Critical

Test Case:

Transaction ID	Operation Start Timestamp	Operation End Timestamp	Operation Detail	State
206	255567481387300	255567482259400	Select attribute1 from table_7_1 where primarykey= 832	Success
204	255567479507200	255567480060500	Select attribute1 from table_7_4 where primarykey= 1460	Success
206	255567484738700	255567485188500	Update table_7_4 set attribute where primarykey=1460	Success
204	255567484625200	255567485012500	Update table_7_1 set attribute1 = -635092 where primarykey= 832	Success

204	255567485386900	255567485913000	Commit	Success
206	255567486411400	255567486923500	Commit	Success

Transaction 206 reads the record(832) of table_ 7_ 1， then transaction 204 writes a new record to cover it, so transactions 206 to 204 have a RW dependency. Similarly, transaction 204 reads the record(1460) of table_ 7_ 4, then transaction 206 writes a new record to cover it, so transactions 204 to 206 have a RW dependency. Finally, transactions 204 to 206 generate a circular dependency, that is, write skew. However, the test environment is the SSI isolation level of PostgreSQL, which is a bug.

OpenGauss

Violating First-Updater-Wins

T1	T2	操作的开始时间	操作的结束时间	Operation ID
	Begin;			0
	set session transaction isolation level repeatable read;			1
	update "table0" set "coAttr31_0" = 1048.0 where ("pkAttr0" = 280) and ("pkAttr1" = 241) and ("pkAttr2" = ‘vc204’) and ("pkAttr3" u003d ‘vc361’) and ("pkAttr4" u003d 363);--row count=1	6415323038516010	6415323055101045	2
Begin;				3

set session transaction isolation level repeatable read;				4
select "pkAttr0", "pkAttr1", "pkAttr2", "pkAttr3", "pkAttr4", "pkAttr5", "pkAttr6", "pkAttr7", "fkAttr0_0", "fkAttr0_1", "fkAttr0_2", "fkAttr0_3", "fkAttr0_4" from "view0" where ("fkAttr0_0" = 94) and ("fkAttr0_1" = 239) or ("fkAttr0_2" < 'vc119') and ("fkAttr0_3" > 'vc81u') and ("fkAttr0_4" = 278) ;		6415323048509244	6415323056824999	5
	COMMIT	6415323105021414	6415323238908520	6
delete from "table0" where ("pkAttr0" u003d 280) and ("pkAttr1" = 241) and ("pkAttr2" = 'vc204') and ("pkAttr3" u003d 'vc361') and ("pkAttr4" u003d 363); --row count=1		6415324085437525	6415324087742609	7

Violating Read-Consistency

Create table table2 (primarykey in primary key, `coAttr25_0 int`);
Insert into table2 values(6,0);
Insert into table2 values(7,0);

T1	T2	操作的开始时间	操作的结束时间	Operation ID

Begin;				0
set session transaction isolation level repeatable read;				1
update "table2" set "coAttr25_0" = 78354, where "primaryKey" = 7;		10027984097097733	10027984142856932	2
	Begin;			3
	set session transaction isolation level repeatable read;			4
	"update "table2" set " coAttr25_0" = 14 where "primaryKey" = 6;	10027984197774891	10027984212816248	5
	Commit	10027984340067174	10027984341596462	6
select "primaryKey","fkAttr0_0", " coAttr25_0" from "table2";--result set "primaryKey": "6", " coAttr25_0": "14"		10027984594834117	10027984601161926	7

DBx

Read inconsistency

T1	T2	操作的开始时间	操作的结束时间	Operation ID
set session transaction isolation level repeatable read;		启动会话时设置		0
	set session transaction isolation level repeatable read;		启动会话时设置	1
START TRANSACTION READ		2021-03-23	2021-03-23	2

ONLY,WITH CONSISTENT SNAPSHOT;		16:04:15.428	16:04:15.429	
	START TRANSACTION;	2021-03-23 16:04:16.350	2021-03-23 16:04:16.350	3
	update table0 set coAttr17 = 19635, coAttr18 = 1244, coAttr19 = 92947 where (pkAttr0 = 'vc239') and (pkAttr1 = 'vc234') and (pkAttr2 = 'vc233'); **return rowCount=1;**	2021-03-23 16:04:17.836	2021-03-23 16:04:17.872	4
	COMMIT	2021-03-23 16:04:17.885	2021-03-23 16:04:18.099	5
select pkAttr0, pkAttr1, pkAttr2, coAttr17, coAttr18, coAttr19 from table0 order by pkAttr0 ; **return query result including:** {"pkAttr2":"vc233","pkAttr0":"vc239","pkAttr1":"vc234"} {"coAttr18":"1244"} {"coAttr19":"92947"} {"coAttr17":"19635"}		2021-03-23 16:04:18.296	2021-03-23 16:04:18.386	6

At the DBx RR isolation level, after T1 starts the transaction, that is, after obtaining the consistency snapshot, another parallel transaction T2 generates a write operation, and T1 can see the write result generated by the parallel transaction T2.

Closed/Duplicate bug reports

MySQL

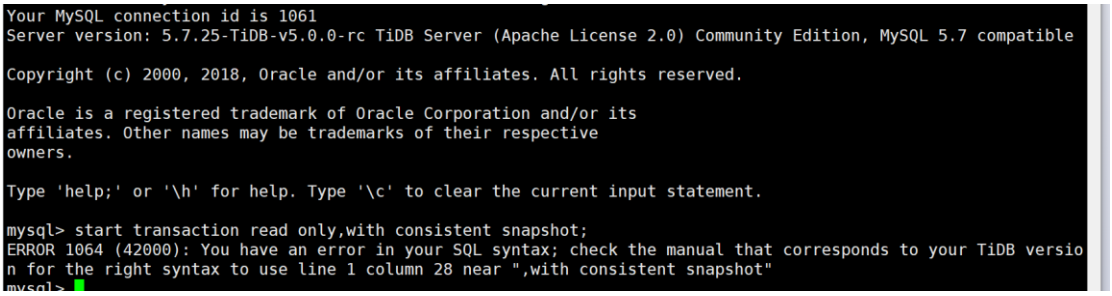
MySQL Bug #103891

TiDB

Query in transaction may return rows with same unique index column value #24195

Bug in Start Transaction

As for the start transaction statement, the PingCAP official document shows that it supports the keywords with concern snapshot and read only. However, in the process of TiDB, we found that tidb cannot support these two keywords at the same time, as show in the following figure:



PostgreSQL

PostgreSQL BUG #17017