# Bugs found in Database Management Systems

We have successfully discovered 23 transactional bugs from real-world production-level DBMSs, including 5 bugs in MySQL, 2 bugs in PostgreSQL, 12 bugs in TiDB, 2 bugs in OpenGauss, and 2 bugs in a commercial DBMS. We are thankful to the DBMS developers for responding to our bug reports and fixing the bugs that we found.

## Fixed bugs

### TiDB

#### Dirty write in SI

**Severity:**

(S1)Critical

**Test Case:**

| **Transaction ID** | **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 | Update table_7_2 set b=-5012153, c=2240641.4 where a=676 | Success |
| 723 | Update table_7_2 set b=852150 where a=676 | Success |
| 739 | Commit | Success |

Transaction 739 updates a record 676 in table_ 7_2 and holds a exclusive lock on the record 676. Before transaction 739 releases the exclusive lock on record 676, transaction 723 also successfully acquires a exclusive lock on record 676 and updates it, which results dirty write anomalies.

#### Read inconsistency in SI

**Severity:**

(S1)Critical

**Test Case:**

| Transaction ID | 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 | Update table_8_2 set attribute6=-0.386 where primarykey=3873 | Success |
| 904 | Commit | Success |
| 914 | Set @@global.tx_isolation='REPEATABLE-READ'; | Success |
| 907 | Update table_8_2 set attribute6=0.484 where primarykey=3873 | Success |
| 907 | Commit | Success |
| 914 | Select attribute6 from table_8_2 where primarykey=3873 | Success(attribute6 =-0.368) |

In the table above, for a record 3873 in table_ 8_2, there are two historical versions on attribute6, the first is -0.386 created by transaction 904;the second is 0.484 created by transaction 907. Since the select operation of transaction 914 happens after the committing of transaction 907, transaction 914 should sees the second version, i.e., 0.484, instead of the first version, i.e., -0.368. However, the select operation of transaction 914 returns the first version -0.368, which indicates there is a problem about the consistency read in TiDB.

## Schema version check error

**Severity:**

(S2)Serious

| Transaction ID | Operation Detail | State |
|---|---|---|
| 712 | Drop db0.table_1_2 | Success |
| 723 | Update db1.table_5_1 set attribute2=8132130 where primarykey=6123 | Exception(In formation schema is changed) |

The first line in transaction 712 modifies db0's schema information, and the second line in transaction 723 modifies db1's data with exception "information schema is changed". However, there is no modification on db1's schema information, which indicates a bug hidden in checking schema version.

## Timestamp acquisition mechanism error in RC

**Severity:**

(S1)Critical

| Transaction ID | Operation Detail | State |
|---|---|---|

|  |  |  |
|---|---|---|
| 232 | 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

**Severity:**

(S3)Critical

**Test Case:**

| **Operation ID** | **Operation Detail** | **State** |
|---|---|---|
| 1 | Update tablecsacas0 set attributeqwdcwq3=FILE(“./data_case/obj/12obj_file.obj”) where primarykeycqwda0 = 15363173 and primarykeycqwda1 = 940396828 and primarykeycqwda2 = 1209414904 | Success |
| 2 | Update tablecsacas0 set attributeqwdcwq3=FILE(“./data_case/obj/12obj_file.obj”) and other column where primarykeycqwda0 = 15363173 and primarykeycqwda1 = 940396828 and primarykeycqwda2 = 1209414904 | Success |
| 3 | Select attributeqwdcwq3 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.

## Two locks of the FOR UPDATE statements are not mutually exclusive

**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 |

| Operation ID | **Session1** | **Session2** | **State** |
|---|---|---|---|
| 1 | Begin |  | Success |
| 2 |  | Begin | Success |
| 3 | update t1 set b=12 where a=1; |  | Success |
| 4 |  | select * from view0 where t2_a>3 for update;<br>`+------+------+------+`<br>`\| t2_a \| t2_b \| t1_b \|`<br>`+------+------+------+`<br>`\|    5 \|    1 \|    2 \|`<br>`\|    4 \|    5 \|    6 \|`<br>`+------+------+------+` | Success |
| 5 |  | Commit;(Success) | Success |
| 6 | Commit;(Success) |  | Success |

Operation 3 holds a exclusive lock on a record 1 in table t1 until operation 6 releases the lock. Due to the nature of exclusive locks, operation 4 attempts to acquire a exclusive lock on record 1 in table t1, which should be blocked. However, TiDB grants operation 4 a exclusive lock on record 1 in table t1, which indicates a locking violation.

## The update statement locks data that does not exist

**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);

| Operation ID | **Session1** | **Session2** | **State** |
|---|---|---|---|
| 1 | Begin | | Success |
| 2 | | Begin | Success |
| 3 | Update t set b=314 where a=1; | | Success with row count = 0 |
| 4 | | Insert into t values(1,3); | blocking |
| 5 | Commit;(success) | | Success |
| 6 | | Insert into t values(1,3);(success) | Success with row count = 1 |
| 7 | | Commit;(Success) | 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

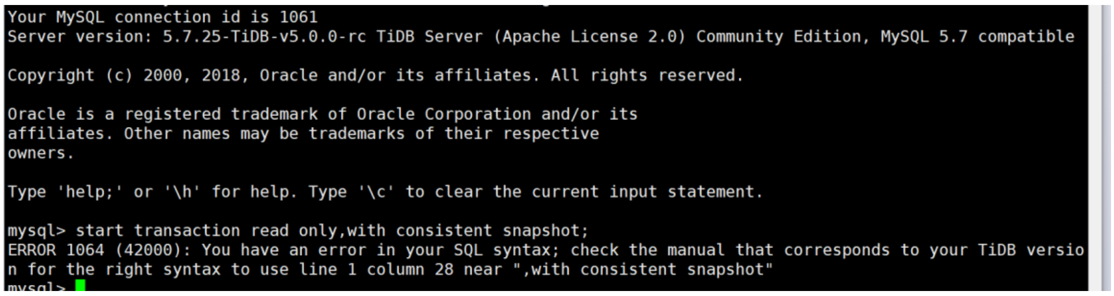
See https://github.com/pingcap/tidb/issues/24227

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

https://github.com/pingcap/tidb/issues/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:



```
Your MySQL connection id is 1061
Server version: 5.7.25-TiDB-v5.0.0-rc TiDB Server (Apache License 2.0) Community Edition, MySQL 5.7 compatible

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> start transaction read only,with consistent snapshot;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your TiDB versio
n for the right syntax to use line 1 column 28 near ",with consistent snapshot"
mysql>
```

## Query Error in information_schema.slow_query

See https://github.com/pingcap/tidb/issues/28069

## Select under repeatable read isolation level returns stale version of data

See https://github.com/pingcap/tidb/issues/36718

## MySQL

## Two parallel threads trigger error code '1032 Can't find record in 'table'

See https://bugs.mysql.com/bug.php?id=103891

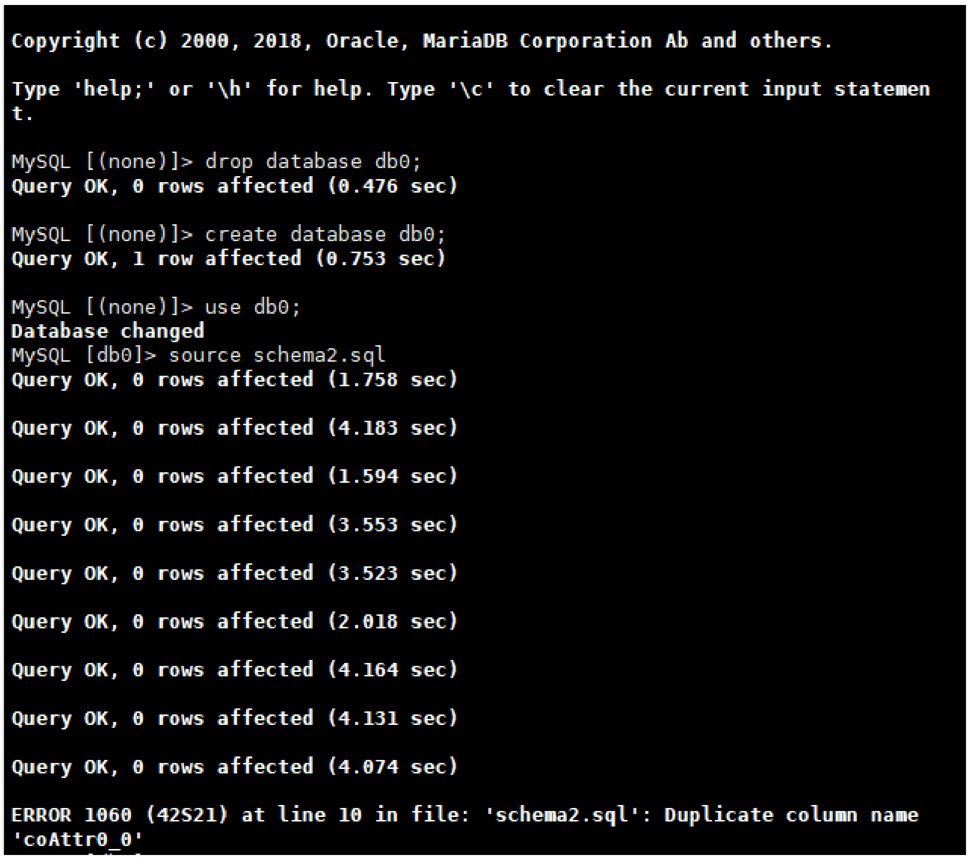
## Select under repeatable read isolation level returns stale version of data

See https://bugs.mysql.com/bug.php?id=108015

## 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'
```

schema2.sql contains following statements:

```
create table table0
(
        pkId         integer,
        pkAttr0     integer,
        pkAttr1     integer,
        pkAttr2     integer,
        pkAttr3     integer,
        pkAttr4     integer,
        coAttr0_0 integer,
        coAttr0_1 decimal(10, 0),
        coAttr0_2 varchar(100),
        primary key (pkAttr0, pkAttr1, pkAttr2, pkAttr3, pkAttr4)
);
alter table table0 add index index_pk(pkAttr0, pkAttr1, pkAttr2, pkAttr3, pkAttr4);
create table table1
```

```
(
        pkId            integer,
        pkAttr0      integer,
        pkAttr1      integer,
        coAttr0_0 varchar(100),
        coAttr0_1 integer,
        coAttr0_2 varchar(100),
        primary key (pkAttr0, pkAttr1)
);
alter table table1 add index index_pk(pkAttr0, pkAttr1);
alter table table1 add index index_commAttr0(coAttr0_0, coAttr0_1, coAttr0_2);
create table table2
(
        pkId            integer,
        pkAttr0      integer,
        pkAttr1      integer,
        pkAttr2      integer,
        pkAttr3      integer,
        pkAttr4      integer,
        pkAttr5      integer,
        pkAttr6      integer,
        coAttr0_0 decimal(10, 0),
        coAttr0_1 varchar(100),
        coAttr0_2 varchar(100),
        fkAttr0_0 integer,
        fkAttr0_1 integer,
        fkAttr0_2 integer,
        fkAttr0_3 integer,
        fkAttr0_4 integer,
        fkAttr1_0 integer,
        fkAttr1_1 integer,
        primary key (pkAttr0, pkAttr1, pkAttr2, pkAttr3, pkAttr4, pkAttr5, pkAttr6),
        foreign key (fkAttr0_0, fkAttr0_1, fkAttr0_2, fkAttr0_3, fkAttr0_4) references table0
(pkAttr0, pkAttr1, pkAttr2, pkAttr3, pkAttr4),
        foreign key (fkAttr1_0, fkAttr1_1) references table1 (pkAttr0, pkAttr1)
);
alter table table2 add index index_pk(pkAttr0, pkAttr1, pkAttr2, pkAttr3, pkAttr4, pkAttr5,
pkAttr6);
alter table table2 add index index_fk0(fkAttr0_0, fkAttr0_1, fkAttr0_2, fkAttr0_3, fkAttr0_4);
alter table table2 add index index_fk1(fkAttr1_0, fkAttr1_1);
create view view2 (pkAttr0, pkAttr1, pkAttr2, pkAttr3, pkAttr4, pkAttr5, pkAttr6, fkAttr0_0,
fkAttr0_1, fkAttr0_2, fkAttr0_3, fkAttr0_4, fkAttr1_0, fkAttr1_1, coAttr0_0, coAttr0_1, coAttr0_2,
coAttr1_0, coAttr1_1, coAttr1_2)
as
```

```
select table2.pkAttr0,
            table2.pkAttr1,
            table2.pkAttr2,
            table2.pkAttr3,
            table2.pkAttr4,
            table2.pkAttr5,
            table2.pkAttr6,
            table2.fkAttr0_0,
            table2.fkAttr0_1,
            table2.fkAttr0_2,
            table2.fkAttr0_3,
            table2.fkAttr0_4,
            table2.fkAttr1_0,
            table2.fkAttr1_1,
            table2.coAttr0_0,
            table2.coAttr0_1,
            table2.coAttr0_2,
            table0.coAttr0_0,
            table0.coAttr0_1,
            table0.coAttr0_2
from table2,
         table0
where table2.fkAttr0_0 = table0.pkAttr0
   and table2.fkAttr0_1 = table0.pkAttr1
   and table2.fkAttr0_2 = table0.pkAttr2
   and table2.fkAttr0_3 = table0.pkAttr3
   and table2.fkAttr0_4 = table0.pkAttr4;
```

# Unconfirmed bugs

## MySQL

### 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);
```

| Transaction ID | Session1 | Session2 | State |
|---|---|---|---|
| | set session transaction isolation level serializable; | | Success |

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

The select operation in transaction 1 acquires a table-level shared lock on table t, while the update operation in transaction 2 acquires a record-level exclusive lock on a record (1,2) in table t. MySQL successfully grants these two locks. However, these two locks are incompatible, which indicates a locking violation.

See https://bugs.mysql.com/bug.php?id=105988

## Read uncommitted transaction reads the result of a failed write operation

The repeat execution flow is shown as the following:

```
/* configuration */ Set global innodb_deadlock_detect=off;
/* init */ Create Table t(a int primary key, b int);
/* init */ Insert into t values(1,2);
/* init */ Insert into t values(2,4);

/* txn1 */ Begin;
/* txn1 */ Set session transaction isolation level read uncommitted;
/* txn2 */ Begin;
/* txn2 */ Set session transaction isolation level read uncommitted;
/* txn3 */ Begin;
/* txn3 */ Set session transaction isolation level read uncommitted;
/* txn2 */ Delete from t where a=1;
/* txn3 */ Update t set b=321 where a=2;
/* txn2 */ Update t set b=1421 where a=2;
/* txn3 */ Insert into t value(1,1231);
/* txn1 */ Select * from t where a=1;
/* init */
```

Transaction 2 writes new versions on records 1 and 2 successively, while Transaction 3 writes new versions on records 2 and 1 successively. So there is a deadlock situation between transaction 2 and 3. Before the deadlock between transaction 2 and 3 timeouts, another read uncommitted transaction 1 launch a query to read the record 1 that has been modified by transaction 2 and 3 successively. Since the second write operation of transaction 3 are failed due to deadlock, we should not see its write results. Therefore, as expected, the query result of transaction 1 should be the write result of transaction 2. However, the query result of transaction 1 is the write result of transaction 2, which is weird. We think there may be a subtle bug hidden in the current version of MySQL.

See

## Update BLOB data error

**Severity:**

(S3)Critical

**Test Case:**

| **Operation ID** | **Operation Detail** | **State** |
|---|---|---|
| 1 | Update tablecsacas0 set attributeqwdcwq3=FILE("./data_case/obj/12obj_file.obj") where primarykeycqwda0 = 15363173 and primarykeycqwda1 = 940396828 and primarykeycqwda2 = 1209414904 | Success |
| 2 | Update tablecsacas0 set attributeqwdcwq3=FILE("./data_case/obj/12obj_file.obj") and other column where primarykeycqwda0 = 15363173 and primarykeycqwda1 = 940396828 and primarykeycqwda2 = 1209414904 | Success |
| 3 | Select attributeqwdcwq3 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.

# PostgreSQL

## Write skew of SSI

**Data Found:**

2020/07/25

**Severity:**

(S1)Critical

**Test Case:**

| **Transaction ID** | **Operation Detail** | **State** |
|---|---|---|
| 206 | Select attribute1 from table_7_1 where primarykey= 832 | Success |
| 204 | Select attribute1 from table_7_4 where primarykey= 1460 | Success |
| 206 | Update table_7_4 set attribute where primarykey=1460 | Success |

|  |  |  |
|---|---|---|
| 204 | Update table_7_1 set attribute1 = -635092 where primarykey= 832 | Success |
| 204 | Commit | Success |
| 206 | Commit | Success |

Transaction 206 reads a record 832 in 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 in 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 anomalies that should be avoided in Snapshot Isolation Level of PostgreSQL.

## Two different versions of the same row of records are returned in one query

See https://www.postgresql.org/message-id/17017-c37dbbadb77cfde9%40postgresql.org

## OpenGauss

### Violating First-Updater-Wins

| Transaction ID | Session1 | Session2 | State |
|---|---|---|---|
| 2 |  | Begin; | Success |
| 2 |  | set session transaction isolation level repeatable read; | Success |
| 2 |  | `update "table0" set "coAttr31_0" = 1048.0 where ( "pkAttr0" = 280 ) and ( "pkAttr1" = 241 ) and ( "pkAttr2" = ‘vc204’ ) and ( "pkAttr3" = ‘vc361’ ) and ( "pkAttr4" = 363 );--row count=1` | Success |
| 1 | Begin; |  | Success |
| 1 | set session transaction isolation level repeatable read; |  | Success |
| 1 | `select "pkAttr0", "pkAttr1",` |  | Success |

| | | | |
|---|---|---|---|
| | "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 ) ; | | |
| 2 | | COMMIT | Success |
| 1 | delete from "table0" where ( "pkAttr0" = 280 ) and ( "pkAttr1" = 241 ) and ( "pkAttr2" = 'vc204' ) and ( "pkAttr3" = 'vc361' ) and ( "pkAttr4" = 363 ); --row count=1 | | Success |

Transaction 1 starts before transaction 2 commit, and both transaction 1 and 2 write a new version on a record (280, 241,'vc204' , 'vc361' ,363 ). Therefore, transaction 1 and 2 are a pair of concurrent transaction, which should be avoided by first updater wins mechanism in OpenGauss.

## 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);

| Transaction ID | Session1 | Session2 | State |
|---|---|---|---|
| 1 | Begin; | | Success |
| 1 | set session transaction isolation level repeatable read; | | Success |
| 1 | `update "table2" set "coAttr25_0" = 78354, where "primaryKey" = 7;` | | Success |
| 2 | | Begin; | Success |
| 2 | | set session transaction isolation level repeatable read; | Success |
| 2 | | `"update "table2" set " coAttr25_0" = 14 where "primaryKey" = 6;` | Success |
| 2 | | `Commit` | Success |
| 1 | `select "primaryKey","fkAttr0_0", " coAttr25_0" from "table2";--result set "primaryKey": "6", " coAttr25_0": "14"` | | Success |

Transaction 1 launch a update operation while fetches a consistent snapshot. According to the rule of repeatable read isolation level, any operation in transaction 1 should sees a same snapshot., so transaction 1 should not see the write result created by transaction 2. However, transaction 1 sees the write result created by transaction 2, which indicates a consistency read violation.

## DBx

## Read inconsistency

| Transaction ID | Session1 | Session2 | State |
|---|---|---|---|

| | | | |
|---|---|---|---|
| 1 | set session transaction isolation level repeatable read; | | Success |
| 2 | | set session transaction isolation level repeatable read; | Success |
| 1 | START TRANSACTION READ ONLY,WITH CONSISTENT SNAPSHOT; | | Success |
| 2 | | START TRANSACTION; | Success |
| 2 | | update table0 set coAttr17 = 19635, coAttr18 = 1244, coAttr19 = 92947 where ( pkAttr0 = 'vc239' ) and ( pkAttr1 = 'vc234' ) and ( pkAttr2 = 'vc233' ); **return rowCount=1;** | Success |
| 2 | | COMMIT | Success |
| 1 | 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"} | | Success |

At the DBx repeatable read isolation level, after transaction 1 starts the transaction, that is, after obtaining the consistency snapshot, another parallel transaction 2 generates a write operation. Transaction 1 should not see the write result created by transaction 2. However, transaction 1 sees the write result created by transaction 2, which violates the rules of repeatable read isolation level.