数据库原理实验 实验7-8

李宁

2020/10/31

1. 实验七 - 事务常用命令

功能	例子
查看当前活动事务	select * from information_schema.innodb_trx;

设置/查看是否自动提交	set @@autocommit=0; set global @@autocommit=0;
	set autocommit = 0; set global autocommit = 0;
	show global/session variables like 'autocommit';
	(省略global/session 修饰符时，默认为session范围)

设置/查看全局隔离级别	set global transaction isolation level repeatable read; show global variables like 'transaction_isolation';
设置/查看当前连接(session)隔离级别	set transaction isolation level repeatable read; set session transaction isolation level repeatable read;
	show variables like 'transaction_isolation'; show session variables like 'transaction_isolation'; select @@transaction_isolation;

1. 实验七 - 查看锁

- show engine innodb status
- 三张表：innodb_locks, innodb_trx, innodb_lock_waits.

```
select * from information_schema.innodb_trx
select * from performance_schema.data_locks
select * from sys.innodb_lock_waits;
```

innodb_locks表

字段名	说明
lock_id	锁的 ID
lock_trx_id	事务 ID
lock_mode	锁的模式
lock_type	锁的类型，表锁还是行锁
lock_table	要加锁的表
lock_index	锁住的索引
lock_space	锁对象的 space id
lock_page	事务锁定页的数量。若是表锁，则该值为 NULL
lock_rec	事务锁定行的数量，若是表锁，则该值为 NULL
lock_data	事务锁定记录的主键值，若是表锁，则该值为 NULL

1. 实验七 - 查看锁

■查看锁的方式:

innodb_trx表（事务信息，没有表，可知trx_id）

字段名	说明
trx_id	InnoDB 存储引擎内部唯一的事务 ID
trx_state	当前事务的状态
trx_started	事务的开始时间
trx_requested_lock_id	等待事务的锁 ID。如 trx_state 的状态为 LOCK WAIT，那么该值代表当前的事务等待之前事务占用锁资源的 ID。若 trx_state 不是 LOCK WAIT，则该值为 NULL
trx_wait_started	事务等待开始的时间
trx_weight	事务的权重，反映了一个事务修改和锁住的行数。在 InnoDB 存储引擎中，当发生死锁需要回滚时，InnoDB 存储引擎会选择该值最小的进行回滚
trx_mysql_thread_id	MySQL 中的线程 ID，SHOW PROCESSLIST 显示的结果
trx_query	事务运行的 SQL 语句

innodb_lock_waits表

字段	说明	字段	说明
requesting_trx_id	申请锁资源的事务 ID	blocking_trx_id	阻塞的事务 ID
requesting_lock_id	申请的锁的 ID	blocking_trx_id	阻塞的锁的 ID

命令提示符 - mysql -u root -p123456

```
Database changed
mysql> show engine innodb status\G
*************************** 1. row ***************************
  Type: InnoDB
  Name:
Status:
=====================================
2020-11-08 10:57:46 0x1f54 INNODB MONITOR OUTPUT
=====================================
Per second averages calculated from the last 38 seconds
-----------------
BACKGROUND THREAD
-----------------
srv_master_thread loops: 2 srv_active, 0 srv_shutdown, 277 srv_idle
srv_master_thread log flush and writes: 0
----------
SEMAPHORES
----------
OS WAIT ARRAY INFO: reservation count 0
OS WAIT ARRAY INFO: signal count 0
RW-shared spins 0, rounds 0, OS waits 0
RW-excl spins 0, rounds 0, OS waits 0
RW-sx spins 0, rounds 0, OS waits 0
Spin rounds per wait: 0.00 RW-shared, 0.00 RW-excl, 0.00 RW-sx
------------
TRANSACTIONS

Trx id counter 294156
Purge done for trx's n:o < 294150 undo n:o < 0 state: running but idle
History list length 0
LIST OF TRANSACTIONS FOR EACH SESSION:
---TRANSACTION 283793654245984, not started
0 lock struct(s), heap size 1136, 0 row lock(s)
---TRANSACTION 283793654242656, not started
0 lock struct(s), heap size 1136, 0 row lock(s)
---TRANSACTION 283793654243488, not started
0 lock struct(s), heap size 1136, 0 row lock(s)
---TRANSACTION 283793654241824, not started
0 lock struct(s), heap size 1136, 0 row lock(s)
---TRANSACTION 283793654240992, not started
0 lock struct(s), heap size 1136, 0 row lock(s)
---TRANSACTION 294155, ACTIVE 10 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 1136, 1 row lock(s)
MySQL thread id 17, OS thread handle 23184, query id 234 localhost ::1 root updating
update t set name='c2' where id=1
------- TRX HAS BEEN WAITING 10 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 354 page no 4 n bits 72 index GEN_CLUST_INDEX of table `trans`.`t` trx id 294155 lock_mode
X waiting
Record lock, heap no 2 PHYSICAL RECORD: n_fields 5; compact format; info bits 0
 0: len 6; hex 000000001113; asc        ;;
 1: len 6; hex 000000047d0a; asc     } ;;
 2: len 7; hex 02000001d11fc8; asc        ;;
 3: len 4; hex 80000001; asc     ;;
 4: len 10; hex 63312020202020202020; asc c1        ;;

------------------
---TRANSACTION 294154, ACTIVE 15 sec
2 lock struct(s), heap size 1136, 3 row lock(s), undo log entries 1
MySQL thread id 16, OS thread handle 8664, query id 233 localhost ::1 root
--------
FILE I/O
--------
```

```
mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> update t set name='c1' where id=1;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

T1

```
mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> update t set name='c2' where id=1;
```

T2

```
mysql> select * from information_schema.innodb_trx\G
*************************** 1. row ***************************
                    trx_id: 294155
                 trx_state: LOCK WAIT
               trx_started: 2020-11-08 10:57:36
     trx_requested_lock_id: 2318677534496:354:4:2:2318640615912
          trx_wait_started: 2020-11-08 10:57:36
                trx_weight: 2
       trx_mysql_thread_id: 17
                 trx_query: update t set name='c2' where id=1
       trx_operation_state: starting index read
         trx_tables_in_use: 1
         trx_tables_locked: 1
          trx_lock_structs: 2
     trx_lock_memory_bytes: 1136
           trx_rows_locked: 1
         trx_rows_modified: 0
   trx_concurrency_tickets: 0
       trx_isolation_level: REPEATABLE READ
         trx_unique_checks: 1
    trx_foreign_key_checks: 1
trx_last_foreign_key_error: NULL
 trx_adaptive_hash_latched: 0
 trx_adaptive_hash_timeout: 0
          trx_is_read_only: 0
trx_autocommit_non_locking: 0
       trx_schedule_weight: 1
*************************** 2. row ***************************
                    trx_id: 294154
                 trx_state: RUNNING
               trx_started: 2020-11-08 10:57:31
     trx_requested_lock_id: NULL
          trx_wait_started: NULL
                trx_weight: 3
       trx_mysql_thread_id: 16
                 trx_query: NULL
       trx_operation_state: NULL
         trx_tables_in_use: 0
         trx_tables_locked: 1
          trx_lock_structs: 2
     trx_lock_memory_bytes: 1136
           trx_rows_locked: 3
         trx_rows_modified: 1
   trx_concurrency_tickets: 0
       trx_isolation_level: REPEATABLE READ
         trx_unique_checks: 1
    trx_foreign_key_checks: 1
trx_last_foreign_key_error: NULL
 trx_adaptive_hash_latched: 0
 trx_adaptive_hash_timeout: 0
          trx_is_read_only: 0
trx_autocommit_non_locking: 0
       trx_schedule_weight: NULL
2 rows in set (0.00 sec)
```

T2: 正在进行锁等待

T1：update结束，还未提交

```
mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> update t set name='c1' where id=1;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

T1

```
mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> update t set name='c2' where id=1;
```

T2

1. 实验七 查看锁


```
mysql> select * from sys.innodb_lock_waits\G
*************************** 1. row ***************************
                wait_started: 2020-11-08 10:57:36
                    wait_age: 00:00:11
               wait_age_secs: 11
                locked_table: `trans`.`t`
         locked_table_schema: trans
           locked_table_name: t
      locked_table_partition: NULL
   locked_table_subpartition: NULL
                locked_index: GEN_CLUST_INDEX
                 locked_type: RECORD
              waiting_trx_id: 294155
         waiting_trx_started: 2020-11-08 10:57:36
             waiting_trx_age: 00:00:11
     waiting_trx_rows_locked: 1
   waiting_trx_rows_modified: 0
                 waiting_pid: 17
               waiting_query: update t set name='c2' where id=1
             waiting_lock_id: 2318677534496:354:4:2:2318640615912
           waiting_lock_mode: X
             blocking_trx_id: 294154
                blocking_pid: 16
              blocking_query: NULL
            blocking_lock_id: 2318677533664:354:4:2:2318640610936
          blocking_lock_mode: X
        blocking_trx_started: 2020-11-08 10:57:31
            blocking_trx_age: 00:00:16
    blocking_trx_rows_locked: 3
  blocking_trx_rows_modified: 1
     sql_kill_blocking_query: KILL QUERY 16
sql_kill_blocking_connection: KILL 16
1 row in set (0.01 sec)
```

1. 实验七 - 表锁


```
# 获取表锁
LOCK TABLES
    tbl_name [[AS] alias] lock_type
    [, tbl_name [[AS] alias] lock_type] ...

lock_type:
    READ [LOCAL]
  | [LOW_PRIORITY] WRITE

# 释放表锁
UNLOCK TABLES
```

```
LOCK TABLE t1 read, t2 read;
select count(t1.id1) as 'sum' from t1;
select count(t2.id1) as 'sum' from t2;
UNLOCK TABLES;
```

1. 实验七 - 行锁

可以通过以下语句显式加共享锁或排他锁：

- **共享锁（S）：**

  SELECT * FROM table_name WHERE ... LOCK IN SHARE MODE。

其他 session 仍然可以查询记录，也可对该记录加share mode锁。但是当前事务需要对该记录进行更新操作，则很有可能造成死锁。

- **排他锁（X）：**

  SELECT * FROM table_name WHERE ... FOR UPDATE

其他 session 可以查询该记录，但是不能对该记录加共享锁或排他锁，而是等待获得锁。

1. 实验七 - 查看锁

■InnoDB的行锁

- **Record Lock：单个行记录上的锁。**
  Record Lock总是锁住索引记录，若没有索引表会使用隐式的主键来进行锁定。

- **Gap Lock：间隙锁，锁定一个范围，不包含记录本身。**
  在索引记录间隙上的锁，或者是第一条索引记录之前、最后一条索引记录之后上的间隙锁。

- **Next-Key Lock：Gap Lock+Record Lock，锁定一个范围，并且锁定记录本身。前开后闭区间，如(5,10】。**

1. 实验七 - 隔离级别

丢失更新：隔离级别（repeatable read）

表: card , 初值:

id	value
a	100

T1

START

select value into @v1 from card where id='a'; ①

@V1
100

T2

START

select value into @v2 from card where id='a'; ②

@V2
100

update card set value = @v2 + 200 where id='a'; ③

id	value
a	300

COMMIT ④

update card set value = @v1 + 100 where id='a'; ⑤

Update之后

id	value
a	200

ROLLBACK ⑥

select value from card where id='a'; ⑦

id	value
a	300

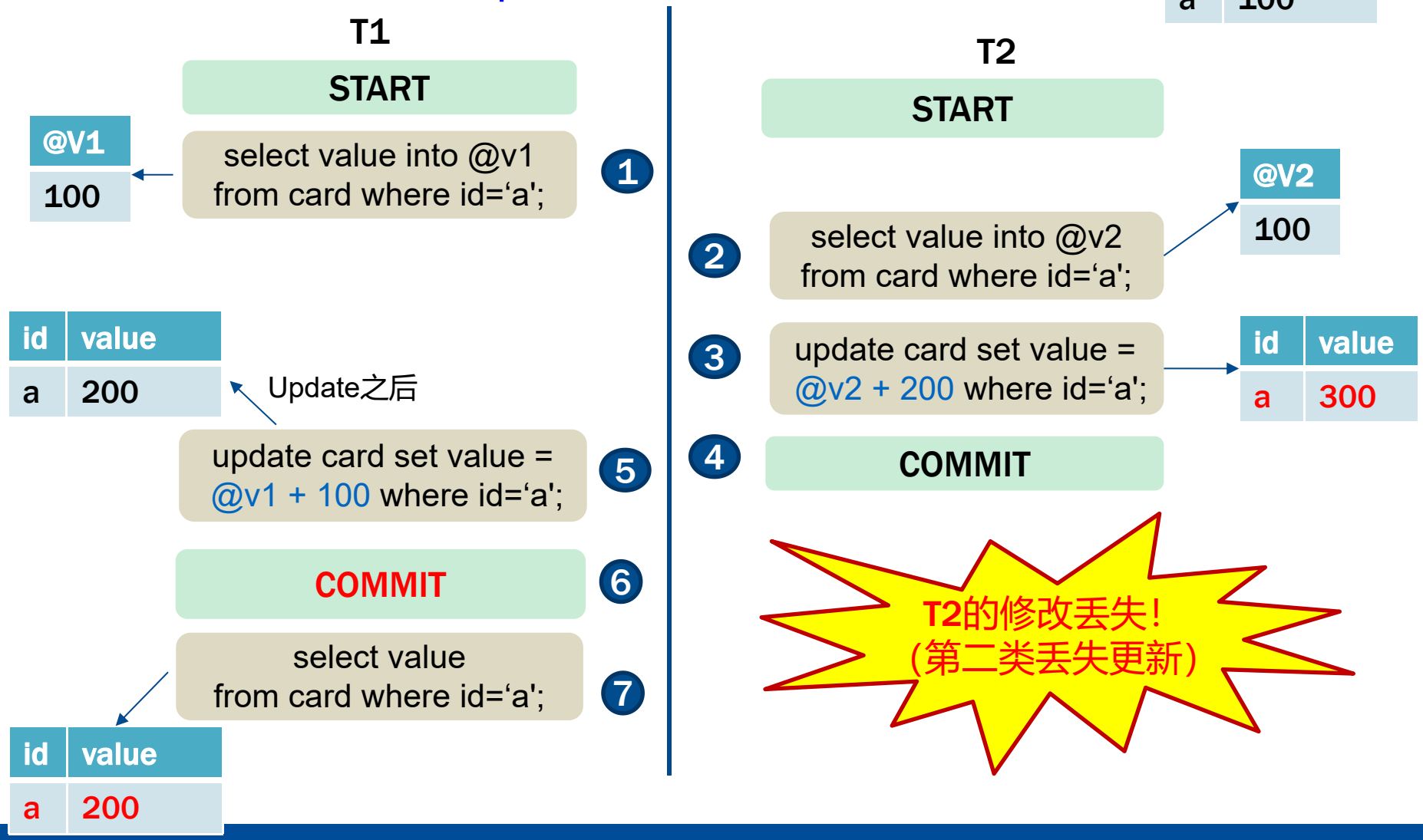
T2的修改不会丢失
（第一类丢失更新）

1. 实验七 - 隔离级别

丢失更新：隔离级别（repeatable read）

表: card , 初值:

id	value
a	100



1. 实验七 - 隔离级别

丢失更新：隔离级别（read uncommitted）

表: card , 初值：

id	value
a	100

T1

START

id	value
a	100

select value
from card where id='a'; 1

T2

START

select value
from card where id='a'; 2

id	value
a	100

update card set value = value+200 where id='a'; 3

id	value
a	300

COMMIT 4

id	value
a	400

Update前重读数据value

update card set value = value+100 where id='a'; 5

COMMIT 6

select value
from card where id='a'; 7

id	value
a	400

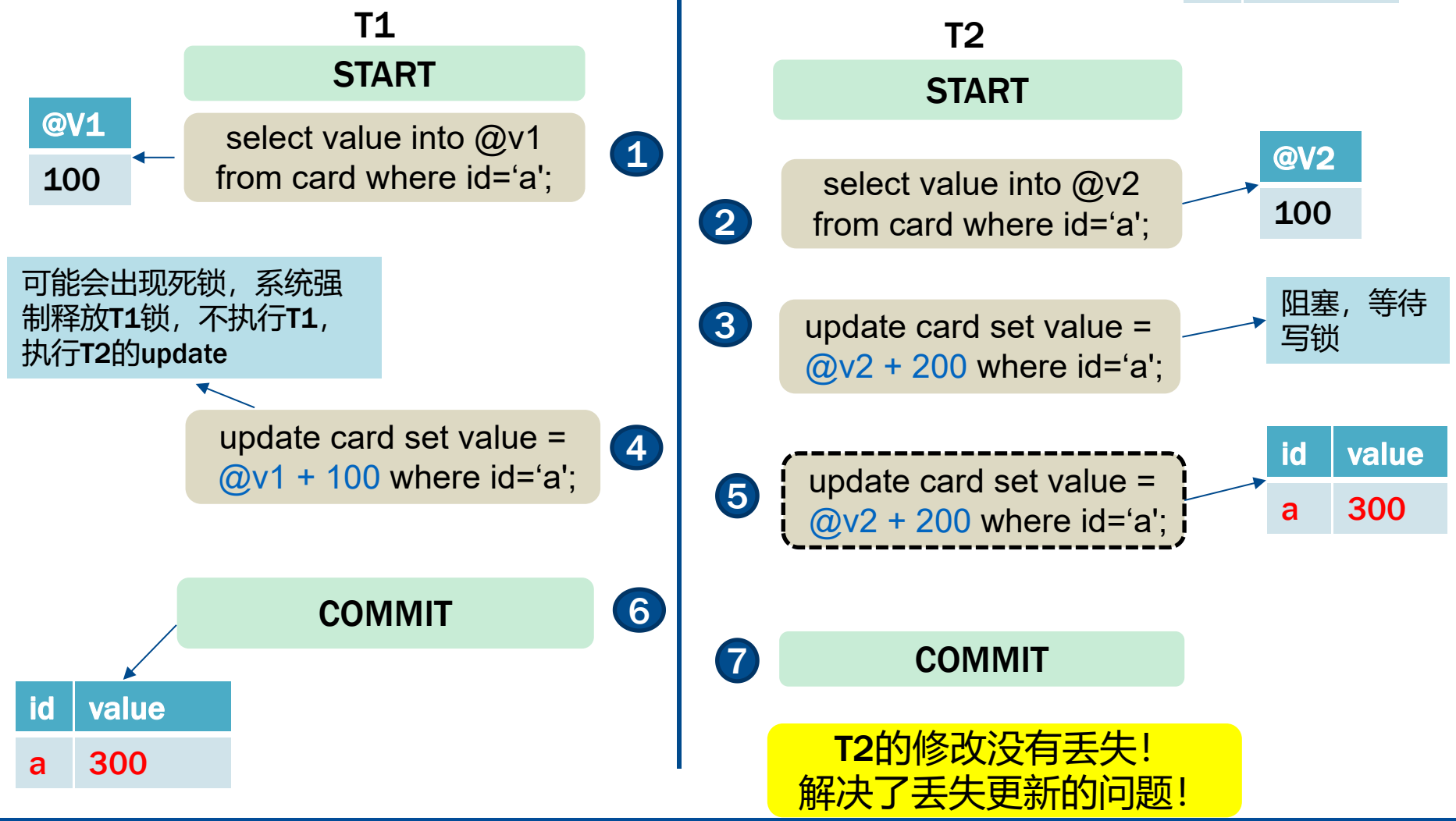
这种情况不会发生丢失更新！
（注意与前一页区别）

1. 实验七 - 隔离级别

丢失更新：隔离级别（serializable）

表: card , 初值:

id	value
a	100



1. 实验七 - 隔离级别

读脏数据：隔离级别（read uncommitted）

表: card , 初值：

id	value
a	100

T1

START

id	value
a	100

select value
from card where id='a'; ①

T2

START

select value
from card where id='a'; ②

id	value
a	100

update card set value = value+200 where id='a'; ③

id	value
a	300

id	value
a	300

读到了未提交的数据

select value
from card where id='a'; ④

COMMIT ⑥

⑤ COMMIT

T1内部，读到了其他事物还未提交的数据！

1. 实验七 - 隔离级别

不可重复读: 隔离级别（read committed）

表: card , 初值:

id	value
a	100

T1

START

id	value
a	100

select value
from card where id='a'; ①

T2

START

select value
from card where id='a'; ②

id	value
a	100

update card set value = value+200 where id='a'; ③

id	value
a	300

id	value
a	100

读到了初始的数据

select value
from card where id='a'; ④

COMMIT ⑤

id	value
a	300

select value
from card where id='a'; ⑥

读到了已提交的数据

COMMIT ⑦

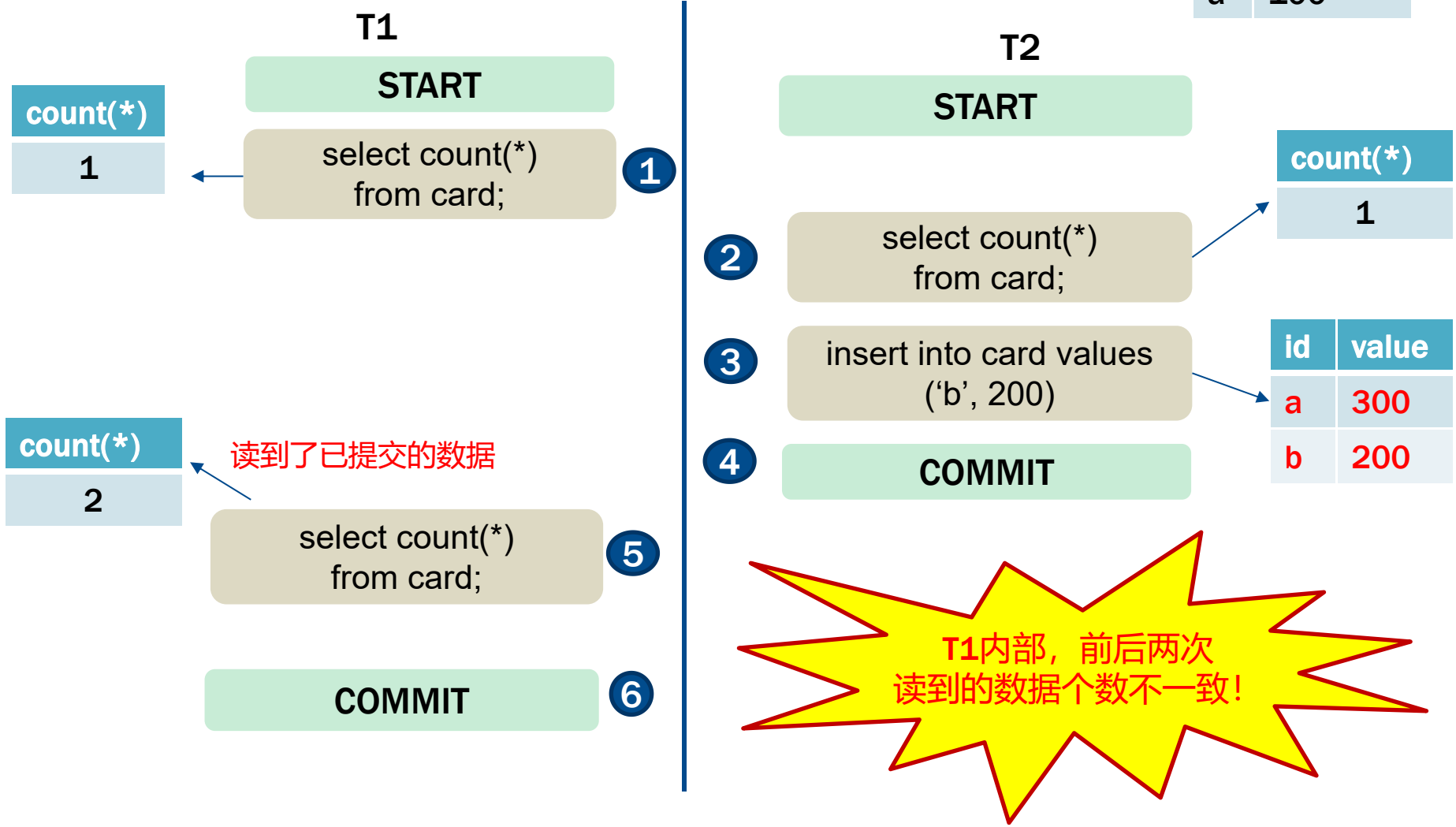
T1内部，前后两次
读到的数据值不一致！

1. 实验七 - 隔离级别

幻读（插入）：隔离级别（read committed）

表: card , 初值:

id	value
a	100



1. 实验七 - 死锁检测

MySQL默认设置下：会自动进行死锁检测，检测到死锁后，等待一个超时时间，然后强制结束一个死锁，使得另一个事务可以继续。

Deadlock found when trying to get lock; try restarting transaction

1. 实验七 - 事务日志

Binlog文件：用于进行事务恢复的日志

```
-- 显示binlog的基本信息
show variables like 'log_bin'; --确认binlog是否开启
show master logs;  -- 显示所有的binlog文件
show master status;  -- 显示最新的binlog的最后位置
show binlog events;          -- 显示所有的binlog

-- 显示指定binlog
show binlog events in 'THINKPAD_LINING-bin.000019’  ;

-- 显示指定binlog中某个位置之后的log
show binlog events in 'THINKPAD_LINING-bin.000019' from 40080; show
binlog events in 'THINKPAD_LINING-bin.000019' from 40080 limit 10;

--刷新binlog，生成一个新的binlog文件
flush logs;
```

1. 实验七 - 事务日志

Binlog文件：用于进行事务恢复的日志

- Row:不记录sql语句上下文相关信息，仅保存哪条记录被修改。
- Statement：每一条会修改数据的sql都会记录在binlog中。
- Mixedlevel: 是以上两种的混合使用，一般的语句修改使用statement格式保存binlog，如一些函数; statement无法完成主从复制的操作，则采用row格式保存binlog.MySQL会根据执行的每一条具体的sql语句来区分对待记录的日志形式。

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Log_name	Pos	Event_type	Server_id	End_log_pos	Info
THINKPAD_LINING-bin.000019	42065	Table_map	1	42130	table_id: 129 (trans.icbc_card)
THINKPAD_LINING-bin.000019	42130	Update_rows	1	42202	table_id: 129 flags: STMT_END_F
THINKPAD_LINING-bin.000019	42202	Xid	1	42233	COMMIT /* xid=1851 */
THINKPAD_LINING-bin.000019	42233	Anonymous_Gtid	1	42312	SET @@SESSION.GTID_NEXT= 'ANONYMOUS'
THINKPAD_LINING-bin.000019	42312	Query	1	42397	BEGIN
THINKPAD_LINING-bin.000019	42397	Table_map	1	42462	table_id: 129 (trans.icbc_card)
THINKPAD_LINING-bin.000019	42462	Update_rows	1	42534	table_id: 129 flags: STMT_END_F
THINKPAD_LINING-bin.000019	42534	Xid	1	42565	COMMIT /* xid=1865 */
THINKPAD_LINING-bin.000019	42565	Anonymous_Gtid	1	42644	SET @@SESSION.GTID_NEXT= 'ANONYMOUS'
THINKPAD_LINING-bin.000019	42644	Query	1	42729	BEGIN
THINKPAD_LINING-bin.000019	42729	Table_map	1	42794	table_id: 129 (trans.icbc_card)
THINKPAD_LINING-bin.000019	42794	Update_rows	1	42866	table_id: 129 flags: STMT_END_F
THINKPAD_LINING-bin.000019	42866	Xid	1	42897	COMMIT /* xid=1874 */
THINKPAD_LINING-bin.000019	42897	Anonymous_Gtid	1	42974	SET @@SESSION.GTID_NEXT= 'ANONYMOUS'
THINKPAD_LINING-bin.000019	42974	Query	1	43087	use `trans`; create table t (id int) /* xid=1920 */

1. 实验七 - 事务日志

西北工业大学 NORTHWESTERN POLYTECHNICAL UNIVERSITY

Binlog文件：用于binlog进行事务恢复

创建新的binlog日志文件

flush logs;
show master status;
假设：最新：mysql-bin. 000022

↓

执行常规SQL数据操作（含创建，增删改操作）

↓

从日志中找回待恢复之前的SQL语句，导出为test000022.sql

1. mysqlbinlog.exe mysql-bin.000022 > test_000022.txt
2. 在txt日志中查找待恢复（如DROP TABLE）日志的位置(该语句的at 2413)
3. 导出binlog日志中'DROP TABLE'之前的SQL语句

```
mysqlbinlog mysql-bin.000022 -d db1 --skip-gtids --stop-position=2413 > test000022.sql
```

↓

在mysql中执行以上SQL文件

```
source C:\ProgramData\MySQL\MySQL Server 8.0\Data\test000022.sql
```

1. 实验七 - 事务日志

西北工业大学 NORTHWESTERN POLYTECHNICAL UNIVERSITY

Binlog文件：(输出为txt文件)

```
/*!80014 SET @@session.immediate_server_version=80021*//*!*/;
SET @@SESSION.GTID_NEXT= 'ANONYMOUS'/*!*/;
# at 2413
#201103 21:21:37 server id 1  end_log_pos 2538 CRC32 0x599dd0ba     Query   thread_id=35    exec_time=1519  error_code=0    Xid = 2255
SET TIMESTAMP=1604409697/*!*/;
DROP TABLE `t1` /* generated by server */
/*!*/;
# at 2538
#201103 21:21:37 server id 1  end_log_pos 2617 CRC32 0xe5d082ed     Anonymous_GTID  last_committed=10   sequence_number=11  rbr_only=yes    or
/*!50718 SET TRANSACTION ISOLATION LEVEL READ COMMITTED*//*!*/;
# original_commit_timestamp=1604411225757273 (2020-11-03 21:47:05.757273 中国标准时间)
# immediate_commit_timestamp=1604411225757273 (2020-11-03 21:47:05.757273 中国标准时间)
```

Window的日志默认路径：
C:\ProgramData\MySQL\MySQL Server 8.0\Data

西北工业大学 NORTHWESTERN POLYTECHNICAL UNIVERSITY

要求：实验指南

优秀作业范例：

- 01实验报告
- 02源代码
- 03数据库备份文件
- 04可执行程序
- 05演示视频
- readme.txt

基于华为云数据库的课程竞赛

目的:

为了激发学生对国产、先进数据库技术的学习兴趣与探索精神，在2020年度《数据库原理》课程中，采用课程竞赛形式，通过华为云数据库合作教改项目资助设置奖励，展开基于华为云数据库的大作业设计与开发，奖励并展示学生的优秀成果。

参赛题目： 数据库应用系统设计与开发

- 具体要求：参考实验指南，参赛作品后台需使用华为云数据库
- 参考选题：
  - 会议室预约系统
  - 工作成果管理系统
  - 图书馆管理系统
  - 社团管理系统
  - 健身锻炼管理系统
  - 自拟题目

基于华为云数据库的课程竞赛

西北工业大学 NORTHWESTERN POLYTECHNICAL UNIVERSITY

- 报名方式：腾讯共享文档中填写报名信息（默认参与）
  【腾讯文档】数据库原理课程竞赛名单
  https://docs.qq.com/sheet/DY0lGSU1NRUNTR1dN

- 作品提交截止日期：
  2020/12/13 (周日晚23:00)

- 比赛方式：

1. 初赛：每位助教推荐自己负责学生报名作品1/5
   (12/14-12/18日之间，每位助教自己组织本组作品检查)

2. 复赛：每个复赛作品演示讲解（线上，5分钟）
   助教和主讲教师共同打分（暂定：12/19 or 12/20)

- 评分规则

项目讲解(100分)[占比50%]					
软件系统功能设计（20分）	数据库相关设计（20分）	功能实现（20分）	程序演示运行正确（20分）	讲解声音、逻辑条理，管理清晰	现场答辩总分
功能设计全面(增删改查)，界面美观，创新点多	数据库设计ER图、索引、范式、完整性约束等	实现完整，代码质量较好有触发器，存储过程等服务端代码	所实现的各功能运行正常，异常系也基本可以处理	声音清晰可辨，逻辑条理顺畅，表达能力好	

实验报告(100分）[50%]						
需求分析（5分）	概念设计（10分）	逻辑设计（10分）	物理设计（5分）	程序源代码（60分）	其他（10分）	笔试总分
数据流图与数据字典	E-R图设计合理，体现需求分析	关系模式，包括函数依赖，码等优化、表设计	索引设计合理	代码规范、类/函数设计合理、变量使用正确、数据库端代码正确、	文档完整性，环境配置，演示视频等	
5	10	10	5	60	10	100

- 获奖名单公布日期：
  - 特等奖：3名， 一等奖：5名，二等奖：10名
  - 2020/12/20 (暂定)，实验课的课程群中公布