

18.MySQL存储引擎-(事务,MVCC)✓

1.0 事务介绍

事务：Transaction （交易）。伴随着交易类的业务出现的概念（工作模式）

交易？

物换物，等价交换。

货币换物，等价交换。

虚拟货币换物（虚拟物品），等价交换。

现实生活中怎么保证交易“和谐”，法律、道德等规则约束。

数据库中为了保证线上交易的“和谐”，加入了“事务”工作机制。

1.1 事务的ACID

A: 原子性

不可再分性：一个事务生命周期中的DML语句，要么全成功要么全失败，不可以出现中间状态。

C：一致性

事务发生前，中，后，数据都最终保持一致。

CR + DWB

I：隔离性

事务操作数据行的时候，不会受到其他时候的影响。

读隔离：隔离级别、MVCC

写隔离：锁

D: 持久性

一旦事务提交，永久生效（落盘）。

1.2 事务的生命周期

1.2.1 标准(显示)的事务控制语句

Bash | Copy

```
1 1. 标准(显示)的事务控制语句
2 开启事务
3 begin;
4      DML语句
5 提交事务
6 commit;
7 回滚事务
8 rollback;
9 注意: 事务生命周期中, 只能使用DML语句 (select、update、delete、insert)
10
```

1.2.2事务自动提交机制

Bash | Copy

```
1 MySQL的自动提交机制 (auto_commit)
2 参数:
3 mysql> select @@autocommit;
4 +-----+
5 | @@autocommit |
6 +-----+
7 | 1 |
8 +-----+
9 作用: 在没有显示的使用begin语句的时候, 执行DML, 会在DML前自动添加begin, 并在DML执行后自动添加 commit。
10 建议: 频繁事务业务场景中, 关闭autocommit。或者每次事务执行时都是显示的begin和commit;
11 关闭方法:
12 # 临时:
13 mysql> set global autocommit=0; 退出会话, 重新连接配置生效。
14 # 永久:
15 [root@db01 ~]# vim /etc/my.cnf
16 autocommit=0 重启生效。
```

1.2.3 隐式提交和回滚

Bash | Copy

```
1 当事务执行过程中出现DDL,DCL会自动提交, 这种情况称为隐式提交
2 导致提交的非事务语句:
3 DDL语句: (ALTER、CREATE 和 DROP)
4 DCL语句: (GRANT、REVOKE 和 SET PASSWORD)
5 锁定语句: (LOCK TABLES 和 UNLOCK TABLES)
6 隐式回滚的情况如下
7 1. 会话窗口被关闭。
8 2. 数据库关闭。
9 3. 出现事务冲突 (死锁)。
```

1.2.4事务生命周期举例

Bash | Copy

```
1  生命周期举例说明
2  1.提交事务练习
3  mysql> begin;
4  mysql> delete from world.city where id=1;
5  mysql> commit;
6  mysql> mysql> select * from city limit 5;
7  +-----+-----+-----+-----+
8  | ID | Name          | CountryCode | District      | Population |
9  +-----+-----+-----+-----+
10 | 2 | Qandahar      | AFG         | Qandahar      | 237500    |
11 | 3 | Herat         | AFG         | Herat         | 186800    |
12 | 4 | Mazar-e-Sharif | AFG         | Balkh         | 127800    |
13 | 5 | Amsterdam     | NLD         | Noord-Holland | 731200    |
14 | 6 | Rotterdam     | NLD         | Zuid-Holland  | 593321    |
15 +-----+-----+-----+-----+
16 2.回滚事务练习
17 mysql> begin;
18 mysql> delete from world.city where id=2;
19 mysql> rollback;
20 mysql> mysql> select * from city limit 5;
21 mysql> select * from city limit 5;
22 +-----+-----+-----+-----+
23 | ID | Name          | CountryCode | District      | Population |
24 +-----+-----+-----+-----+
25 | 2 | Qandahar      | AFG         | Qandahar      | 237500    |
26 | 3 | Herat         | AFG         | Herat         | 186800    |
27 | 4 | Mazar-e-Sharif | AFG         | Balkh         | 127800    |
28 | 5 | Amsterdam     | NLD         | Noord-Holland | 731200    |
29 | 6 | Rotterdam     | NLD         | Zuid-Holland  | 593321    |
30 +-----+-----+-----+-----+
31
```

1.3 事务并发产生的问题读

事务A和事务B操纵的是同一个资源，事务A有若干个子事务，事务B也有若干个子事务，事务A和事务B在高并发的情况下，会出现各种各样的问题

1.3.1 脏读

所谓脏读，就是指**事务A读到了事务B还没有提交的数据**，比如银行取钱，事务A开启事务，此时切换到事务B，事务B开启事务-->取走100元，此时切换回事务A，事务A读取的肯定是数据库里面的原始数据，因为事务B取走了100块钱，并没有提交，数据库里面的账务余额肯定还是原始余额，这就是脏读。

1.3.2 不可重复读

所谓不可重复读，就是指在**一个事务里面读取了两次某个数据，读出来的数据不一致**。还是以银行取钱为例，事务A开启事务-->查出银行卡余额为1000元，此时切换到事务B事务B开启事务-->事务B取走100元-->提交，数据库里面余额变为900元，此时切换回事务A，事务A再查一次查出账户余额为900元，这样对事务A而言，在同一

一个事务内两次读取账户余额数据不一致，这就是不可重复读。

1.3.3 幻读

所谓幻读，就是指在一个事务里面的操作中发现了未被操作的数据。比如学生信息，事务A开启事务-->修改所有学生当天签到状况为false，此时切换到事务B，事务B开启事务-->事务B插入了一条学生数据，此时切换回事务A，事务A提交的时候发现了一条自己没有修改过的数据，这就是幻读，就好像发生了幻觉一样。幻读出现的前提是并发的事务中有事务发生了插入、删除操作。

各种问题读演示

①演示脏读，需要在RU级别下

0.演示前设置参数，更改隔离级别

```
1 vim /etc/my.cnf
2 transaction_isolation='read-uncommitted' 设置隔离级别为RU
3 autocommit=0 关闭事务自动提交机制
4 保存退出 重启数据库 /etc/init.d/mysqld restart
```

1.首先打开两个端口，同时开启事务（begin），输入相同dml语句，结果相同

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

mysql> select * from city where id=100;
+-----+-----+-----+-----+-----+
| ID | Name | CountryCode | District | Population |
+-----+-----+-----+-----+-----+
| 100 | Paraná | ARG | Entre Rios | 207041 |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql>
```

2.在左会话中将id=100的人口数改为10，不提交。此时右会话也可查看到左会话未提交的更改信息。

```
mysql> update city set population=10 where id=100;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> select * from city where id=100;
+-----+-----+-----+-----+-----+
| ID | Name | CountryCode | District | Population |
+-----+-----+-----+-----+-----+
| 100 | Paraná | ARG | Entre Rios | 10 |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql>
```

更改未提交

脏读

②演示不可重复读，在RC级别下

0.演示前设置参数，更改隔离级别

Bash | Copy

```

1 vim /etc/my.cnf
2 transaction_isolation='read-committed' 设置隔离级别为RC
3
4 autocommit=0 关闭事务自动提交机制
   保存退出 重启数据库 /etc/init.d/mysql restart

```

1.首先打开两个端口，同时开启事务（begin），输入相同dml语句，结果相同

```

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

mysql> select * from city where id=100;
+-----+-----+-----+-----+-----+
| ID | Name | CountryCode | District | Population |
+-----+-----+-----+-----+-----+
| 100 | Paraná | ARG | Entre Rios | 207041 |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql>

```

```

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

mysql> select * from city where id=100;
+-----+-----+-----+-----+-----+
| ID | Name | CountryCode | District | Population |
+-----+-----+-----+-----+-----+
| 100 | Paraná | ARG | Entre Rios | 207041 |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql>

```

2.左表将id=100的人口数改为10，提交。右边窗口查看人口更改为了10

```

mysql> update city set population=10 where id=100;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> select * from city where id=100;
+-----+-----+-----+-----+-----+
| ID | Name | CountryCode | District | Population |
+-----+-----+-----+-----+-----+
| 100 | Paraná | ARG | Entre Rios | 10 |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.01 sec)

mysql>

```

已提交

```

mysql> select * from city where id=100;
+-----+-----+-----+-----+-----+
| ID | Name | CountryCode | District | Population |
+-----+-----+-----+-----+-----+
| 100 | Paraná | ARG | Entre Rios | 207041 |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> select * from city where id=100;
+-----+-----+-----+-----+-----+
| ID | Name | CountryCode | District | Population |
+-----+-----+-----+-----+-----+
| 100 | Paraná | ARG | Entre Rios | 10 |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql>

```

数据不一致

③演示幻读，在RC级别下

0.模拟一张表

```
▼ Bash Copy
1  创建模拟表
2  mysql> create table cry(id int not null primary key auto_increment, name varchar(20) not null ,sal int not null);
3  插入数据
4  insert into cry values(1,'a',1000),(2,'b',2000),(3,'c',3000),(4,'d',4000);
5  查看模拟表
6  mysql> select * from cry;
7  +----+-----+-----+
8  | id | name | sal |
9  +----+-----+-----+
10 | 1  | a   | 1000 |
11 | 2  | b   | 2000 |
12 | 3  | c   | 3000 |
13 | 4  | d   | 4000 |
14 +----+-----+-----+
```

1.首先打开两个端口，同时开启事务（begin），输入相同dml语句，结果相同

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

mysql> select * from cry;
+----+-----+-----+
| id | name | sal |
+----+-----+-----+
| 1  | a   | 1000 |
| 2  | b   | 2000 |
| 3  | c   | 3000 |
| 4  | d   | 4000 |
+----+-----+-----+
4 rows in set (0.00 sec)

mysql>

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

mysql> select * from cry;
+----+-----+-----+
| id | name | sal |
+----+-----+-----+
| 1  | a   | 1000 |
| 2  | b   | 2000 |
| 3  | c   | 3000 |
| 4  | d   | 4000 |
+----+-----+-----+
4 rows in set (0.00 sec)

mysql>
```

2.操作看图

```
mysql> select * from cry;
+----+-----+-----+
| id | name | sal |
+----+-----+-----+
| 1 | a | 1000 |
| 2 | b | 2000 |
| 3 | c | 3000 |
| 4 | d | 4000 |
+----+-----+-----+
4 rows in set (0.00 sec)
```

② 这两行有行锁不能被修改, 但是其他行可以被修改

```
mysql> update cry set sal=3000 where sal<3000;
Query OK, 2 rows affected (0.00 sec)
Rows matched: 2 Changed: 2 Warnings: 0
```

① 目的将工资小于3000的改为3000

④ 这边更改工资完成也提交

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

```
mysql> select * from cry;
+----+-----+-----+
| id | name | sal |
+----+-----+-----+
| 1 | a | 3000 |
| 2 | b | 3000 |
| 3 | c | 3000 |
| 4 | d | 4000 |
| 5 | e | 1000 |
+----+-----+-----+
5 rows in set (0.00 sec)
```

⑤ 目的修改工资小于3000的改为3000
所以不应该有工资小于3000的。
所以产生小于3000的行就是幻读情况

```
mysql> select * from cry;
+----+-----+-----+
| id | name | sal |
+----+-----+-----+
| 1 | a | 1000 |
| 2 | b | 2000 |
| 3 | c | 3000 |
| 4 | d | 4000 |
+----+-----+-----+
4 rows in set (0.00 sec)
```

```
mysql> insert into cry values(5,'e',1000);
Query OK, 1 row affected (0.00 sec)
```

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

```
mysql>
```

③ 插入一行数据, 不会影响被修改的两行, 所以插入成功, 提交

④演示幻读, 在RR级别下

0.演示前设置参数, 更改隔离级别

```
1 vim /etc/my.cnf
2 transaction_isolation='REPEATABLE-READ' 设置隔离级别为RR
3 autocommit=0 关闭事务自动提交机制
4 保存退出 重启数据库 /etc/init.d/mysqld restart
```

1.模拟一张表

```
▼ Bash Copy
1  创建模拟表
2  mysql> create table cry(id int not null primary key auto_increment, name varchar(20) not null ,sal int not null);
3  插入数据
4  insert into cry values(1,'a',1000),(2,'b',2000),(3,'c',3000),(4,'d',4000);
5  查看模拟表
6  mysql> select * from cry;
7  +----+-----+-----+
8  | id | name | sal |
9  +----+-----+-----+
10 | 1  | a   | 1000 |
11 | 2  | b   | 2000 |
12 | 3  | c   | 3000 |
13 | 4  | d   | 4000 |
14 +----+-----+-----+
```

2.首先打开两个端口，同时开启事务（begin），输入相同dml语句，结果相同

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

mysql> select * from cry;
+----+-----+-----+
| id | name | sal |
+----+-----+-----+
| 1  | a   | 1000 |
| 2  | b   | 2000 |
| 3  | c   | 3000 |
| 4  | d   | 4000 |
+----+-----+-----+
4 rows in set (0.00 sec)

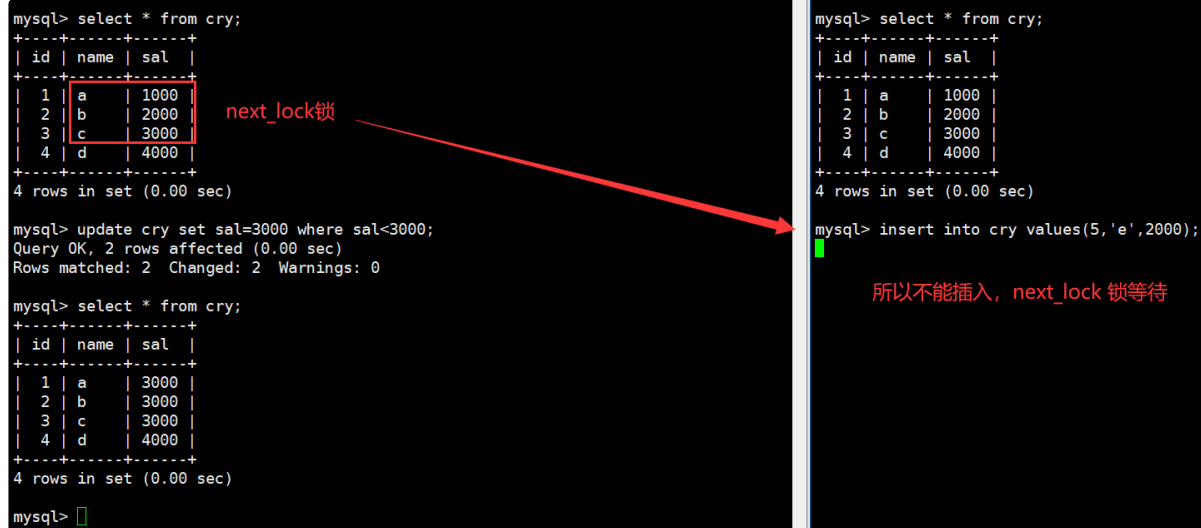
mysql>

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

mysql> select * from cry;
+----+-----+-----+
| id | name | sal |
+----+-----+-----+
| 1  | a   | 1000 |
| 2  | b   | 2000 |
| 3  | c   | 3000 |
| 4  | d   | 4000 |
+----+-----+-----+
4 rows in set (0.00 sec)

mysql>
```


3. 左会话更改数据, 会加上next_lock锁, 所以右会话锁等待, 无法插入



```
mysql> select * from cry;
+----+-----+-----+
| id | name | sal |
+----+-----+-----+
| 1  | a    | 1000 |
| 2  | b    | 2000 |
| 3  | c    | 3000 |
| 4  | d    | 4000 |
+----+-----+-----+
4 rows in set (0.00 sec)

mysql> update cry set sal=3000 where sal<3000;
Query OK, 2 rows affected (0.00 sec)
Rows matched: 2  Changed: 2  Warnings: 0

mysql> select * from cry;
+----+-----+-----+
| id | name | sal |
+----+-----+-----+
| 1  | a    | 3000 |
| 2  | b    | 3000 |
| 3  | c    | 3000 |
| 4  | d    | 4000 |
+----+-----+-----+
4 rows in set (0.00 sec)

mysql>
```

next_lock锁

```
mysql> select * from cry;
+----+-----+-----+
| id | name | sal |
+----+-----+-----+
| 1  | a    | 1000 |
| 2  | b    | 2000 |
| 3  | c    | 3000 |
| 4  | d    | 4000 |
+----+-----+-----+
4 rows in set (0.00 sec)

mysql> insert into cry values(5,'e',2000);

```

所以不能插入, next_lock 锁等待

1.4 事务的隔离级别(解决各种问题读)

1.READ_UNCOMMITTED---RU

读未提交, 即能够读取到没有被提交的数据, 所以很明显这个级别的隔离机制无法解决脏读、不可重复读、幻读中的任何一种, 因此很少使用

2.READ_COMMITTED---RC

读已提交, 即能够读到那些已经提交的数据, 自然能够防止脏读, 但是无法限制不可重复读和幻读

3.REPEATABLE_READ---RR

防止了 脏读和不可重复读, 使用next_lock锁可以防止幻读

4.SERIALIZABLE---SE

串行化, 最高的事务隔离级别, 不管多少事务, 挨个运行完一个事务的所有子事务之后才可以执行另外一个事务里面的所有子事务, 这样就解决了脏读、不可重复读和幻读的问题了

隔离级	脏读可能性	不可重复读可能性	幻读可能性	加锁读
READ UNCOMMITTED	是	是	是	否
READ COMMITTED	否	是	是	否
REPEATABLE READ	否	否	是	否
SERIALIZABLE	否	否	否	是

1.5事务隔离级别配置方法

Bash | Copy

```
1 0.查看系统默认隔离级别
2
3 mysql> select @@transaction_isolation;
4
5 +-----+
6 | @@transaction_isolation |
7 +-----+
8 | REPEATABLE-READ        |
9 +-----+
10
11 1.在线修改参数，退出会话生效
12
13 set global transaction_isolation='read-committed';
14
15 2.在配置文件中修好，重启数据库生效
16 vim /etc/my.cnf
17 [mysqld]
18 transaction_isolation='read-committed'
```

1.6 事务工作逻辑如何保证ACID

A 原子性

不可再分性：一个事务生命周期中的DML语句，要么全成功要么全失败，不可以出现中间状态。

CR回滚保证 A的特性 CI有间接保证

C一致性

事务发生前，中，后，数据都最终保持一致。

CR + DWB

I：隔离性

事务操作数据行的时候，不会受到其他时候的影响。

读隔离：

1.隔离级别

2.MVCC：多版本并发控制. 使用了UNDO快照

RC ：每次做新的查询,都会获得一次全新的readview.

RR ：在开启时候后,第一次查询数据时,就会生成一致性的readview.一直持续到事务结束.一致性快照读

写隔离：lock(锁)

1.record lock 记录锁

RR级别下

2.gap lock 间隙锁

3.next lock 下键锁=gap lock+record lock

（重点）如何查看分析锁等待

1.查看默认锁等待实际

Bash | Copy

```

1  mysql> show variables like '%wait%'; 默认锁等待50秒
2
3  +-----+-----+
4  | Variable_name | Value |
5  +-----+-----+
6  | innodb_lock_wait_timeout | 50 |

```

2. 有关于锁的表在 sys 系统库下 innodb_lock_waits

Bash | Copy

```

1  mysql> select * from innodb_lock_waits\G;
2  ***** 1. row *****
3      wait_started: 2021-04-17 21:17:05
4      wait_age: 00:00:48
5      wait_age_secs: 48
6      locked_table: `world`.`cry`          查看被锁的表
7      locked_table_schema: world
8      locked_table_name: cry
9      locked_table_partition: NULL
10     locked_table_subpartition: NULL
11     locked_index: PRIMARY                innodb加锁是基于索引加锁
12     locked_type: RECORD                  加锁的类型: 行锁
13     waiting_trx_id: 12832
14     waiting_trx_started: 2021-04-17 21:16:30
15     waiting_trx_age: 00:01:23
16     waiting_trx_rows_locked: 1           加锁限定的行数个数
17     waiting_trx_rows_modified: 0
18     waiting_pid: 12                      谁在等待锁操作, 谁被锁住了
19     waiting_query: update cry set name='jj' where id=1  锁等待的要执行的语句
20     waiting_lock_id: 139751497162104:39:4:2:139751390782624
21     waiting_lock_mode: X,REC_NOT_GAP
22     blocking_trx_id: 12831
23     blocking_pid: 8                      阻塞的人是谁
24     blocking_query: NULL
25     blocking_lock_id: 139751497162952:39:4:2:139751390788784
26     blocking_lock_mode: X,REC_NOT_GAP
27     blocking_trx_started: 2021-04-17 21:15:10
28     blocking_trx_age: 00:02:43
29     blocking_trx_rows_locked: 1
30     blocking_trx_rows_modified: 1
31     sql_kill_blocking_query: KILL QUERY 8          处理锁等待方法 (谨慎)
32     sql_kill_blocking_connection: KILL 8          处理锁等待方法 (谨慎)
33  1 row in set (0.00 sec)

```

3. 分析锁等待 (连接线程----> sql线程)

Bash | Copy

```

1  连接线程----> sql线程
2  1.首先sql> select * from innodb_lock_waits\G;
3  找出被阻塞和阻塞者的连接线程 (show processlist) 是谁?
4  ***** 1. row *****
5      wait_started: 2021-04-17 21:17:05
6      wait_age: 00:00:48
7      wait_age_secs: 48
8      locked_table: `world`.`cry`          查看被锁的表
9      locked_table_schema: world
10     locked_table_name: cry
11     locked_table_partition: NULL
12     locked_table_subpartition: NULL
13     locked_index: PRIMARY                innodb加锁是基于索引加锁
14     locked_type: RECORD                  加锁的类型: 行锁
15     waiting_trx_id: 12832
16     waiting_trx_started: 2021-04-17 21:16:30
17     waiting_trx_age: 00:01:23
18     waiting_trx_rows_locked: 1            加锁限定的行数个数
19     waiting_trx_rows_modified: 0
20     waiting_pid: 12                      谁在等待锁操作, 谁被锁住了
21     waiting_query: update cry set name='jj' where id=1  锁等待的要执行的语句
22     waiting_lock_id: 139751497162104:39:4:2:139751390782624
23     waiting_lock_mode: X,REC_NOT_GAP
24     blocking_trx_id: 12831
25     blocking_pid: 8                      阻塞的人是谁
26     blocking_query: NULL
27     blocking_lock_id: 139751497162952:39:4:2:139751390788784
28     blocking_lock_mode: X,REC_NOT_GAP
29     blocking_trx_started: 2021-04-17 21:15:10
30     blocking_trx_age: 00:02:43
31     blocking_trx_rows_locked: 1
32     blocking_trx_rows_modified: 1
33     sql_kill_blocking_query: KILL QUERY 8          处理锁等待方法 (谨慎)
34     sql_kill_blocking_connection: KILL 8           处理锁等待方法 (谨慎)
35
36 提取信息被阻塞的是12 (连接线程id号), 发起阻塞的是8 (连接线程id号)
37
38 2.通过performance_schema下的threads找到连接线程和sql线程的对应关系
39 select * from performance.threads\G;
40
41     THREAD_ID: 52
42     NAME: thread/sql/one_connection
43     TYPE: FOREGROUND
44     PROCESSLIST_ID: 12
45     PROCESSLIST_USER: root
46     PROCESSLIST_HOST: localhost
47     PROCESSLIST_DB: world
48     PROCESSLIST_COMMAND: Sleep
49     PROCESSLIST_TIME: 585
50     PROCESSLIST_STATE: NULL

```

```
51  PROCESSLIST_INFO: update cry set name='jj' where id=1      显示会话做的最后一条sql语句
52  PARENT_THREAD_ID: NULL
53  ROLE: NULL
54  INSTRUMENTED: YES
55  HISTORY: YES
56  CONNECTION_TYPE: Socket
57  THREAD_OS_ID: 3863
58  RESOURCE_GROUP: USR_default
59
60  提取信息：找到THREAD_ID: 52 (sql线程id号)
61
62  3.通过查询 performance_schema .events_statements_history 中对应的线程id号，得到执行sql的历史记录进行分析
63  select * from performance_schema.events_statements_history where thread_id=52\G;
64
```

D: 持久性

一旦事务提交，永久生效（落盘）。主要保证ACID中的D特性。A C 也有间接保证

2.MVCC 多版本并发控制

MVCC：多版本并发控制

功能：通过UNDO生成多版本的“快照”。非锁定读取。

乐观锁：乐观。

悲观锁：悲观。

每个事务操作都要经历两个阶段：

1. MVCC采用乐观锁机制，实现非锁定读取。
2. 在RC级别下，事务中可以立即读取到其他事务commit过的readview
3. 在RR级别下，事务中从第一次查询开始，生成一个一致性readview，直到事务结束。

创建ReadView

- 获取kernel_mutex
- 遍历trx_sys的trx_list链表，获取所有活跃事务，创建ReadView
- Read Committed
- 语句开始，创建ReadView
- Repeatable Read
- 事务开始，创建ReadView

Read Committed

```
Begin;  
  
Create ReadView1;  
  
Statement 1;  
  
Drop ReadView1;  
Create ReadView2;  
  
Statement 2;  
...  
Drop ReadView2;  
Commit;
```

Repeatable Read

```
Begin;  
  
Create ReadView1;  
Statement 1;  
Statement 2;  
...  
  
Drop ReadView1;  
Commit;
```

链接

(%E4%BA%8B%E5%8A%A1%2CMVCC)%E2%88%9A%20%7C%201.0%20%E4%BA%8B%E5%8A%A1%E4%BB%8B%E7%BB%8D%E4%BA%8B%E5%8A%A1%EF%BC%9ATransaction%20%EF%BC%88%E4%E

