

# 锁

# 目录

1

锁类型

2

锁的作用和冲突规则

3

死锁

# 课堂练习：事务期间修改隔离级别

```
CREATE TABLE `t` (  
  `id` int(11) NOT NULL,  
  `c` int(11) DEFAULT NULL,  
  `d` int(11) DEFAULT NULL,  
  PRIMARY KEY (`id`),  
  KEY `c` (`c`)  
) ENGINE=InnoDB;  
  
insert into t values(0,0,0),(5,5,5),  
(10,10,10),(15,15,15),(20,20,20),(25,25,25);
```

执行语句 Q2 后，是什么现象：

- A. 事务主动提交，并切换成 RC 隔离级别
- B. 事务主动回滚，并切换成 RC 隔离级别
- C. 事务继续，整体保留 RR，直到提交
- D. 事务继续，整体保留 RC，直到提交
- E. 事务继续，Q1 保留 RR，Q3 改成 RC

```
set transaction_isolation='repeatable-read';
```

```
begin;
```

```
select * from t where id>5 and id <10 for update; (Q1)
```

```
set transaction_isolation='read-committed'; (Q2)
```

```
select * from t where id>15 and id <20; for update (Q3)
```

```
commit
```

# 课堂练习：事务期间修改隔离级别 - 验证方式

```
root@devops_db 18:35: [test]> select * from information_schema.innodb_trx\G
***** 1. row *****
      trx_id: 3322
      trx_state: RUNNING
      trx_started: 2024-10-26 18:35:49
trx_requested_lock_id: NULL
      trx_wait_started: NULL
      trx_weight: 2
      trx_mysql_thread_id: 27
      trx_query: NULL
      trx_operation_state: NULL
      trx_tables_in_use: 0
      trx_tables_locked: 1
      trx_lock_structs: 2
      trx_lock_memory_bytes: 1192
      trx_rows_locked: 2
      trx_rows_modified: 0
trx_concurrency_tickets: 0
      trx_isolation_level: REPEATABLE READ
      trx_unique_checks: 1
```

# 课堂练习：事务期间修改隔离级别 - 验证方式

```
select ENGINE_TRANSACTION_ID,THREAD_ID,OBJECT_NAME,index_name,
OBJECT_INSTANCE_BEGIN,LOCK_TYPE,LOCK_MODE,LOCK_STATUS,LOCK_DATA
from performance_schema.data_locks;
```

ENGINE_TRANSACTION_ID	THREAD_ID	OBJECT_NAME	index_name	OBJECT_INSTANCE_BEGIN	LOCK_TYPE	LOCK_MODE	LOCK_STATUS	LOCK_DATA
3322	74	t	NULL	130040603925344	TABLE	IX	GRANTED	NULL
3322	74	t	PRIMARY	130040603922288	RECORD	X,GAP	GRANTED	10
3322	74	t	PRIMARY	130040603922288	RECORD	X,GAP	GRANTED	20



# performance\_schema.data\_locks 字段意义

```
***** 3. row *****
ENGINE_TRANSACTION_ID: 3322
      THREAD_ID: 74
      OBJECT_NAME: t
      index_name: PRIMARY
OBJECT_INSTANCE_BEGIN: 130040603922288
      LOCK_TYPE: RECORD
      LOCK_MODE: X,GAP
      LOCK_STATUS: GRANTED
      LOCK_DATA: 20
```

# performance\_schema.data\_locks 字段意义

\*\*\*\*\* 1. row \*\*\*\*\*

ENGINE\_TRANSACTION\_ID: 3322

THREAD\_ID: 74

OBJECT\_NAME: t

index\_name: NULL

OBJECT\_INSTANCE\_BEGIN: 130040603925344

LOCK\_TYPE: TABLE

LOCK\_MODE: IX

LOCK\_STATUS: GRANTED

LOCK\_DATA: NULL

# 这个 IX / IS 是什么?

IX: 准备给表里的行加行的写锁

IX: DML / select for update

IS: 准备给表里的行加行的读锁

IS: select for share

```
mysql> create table t(id int primary key, c int);  
Query OK, 0 rows affected (0.09 sec)
```

```
mysql> insert into t values(1,1),(2,2);  
Query OK, 2 rows affected (0.02 sec)  
Records: 2  Duplicates: 0  Warnings: 0
```



# 这个 IX / IS 是什么?

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

mysql> select ENGINE_TRANSACTION_ID,THREAD_ID,OBJECT_NAME,index_name, OBJECT_INSTANCE_BEGIN,LOCK_TYPE,LOCK_MODE,LOCK_STATUS,LOCK_DATA from performance_schema.data_locks;
Empty set (0.01 sec)

mysql> select * from t limit 1;
+-----+-----+
| id | c   |
+-----+-----+
| 1 | 1 |
+-----+-----+
1 row in set (0.00 sec)

mysql> select ENGINE_TRANSACTION_ID,THREAD_ID,OBJECT_NAME,index_name, OBJECT_INSTANCE_BEGIN,LOCK_TYPE,LOCK_MODE,LOCK_STATUS,LOCK_DATA from performance_schema.data_locks;
Empty set (0.00 sec)
```

# 这个 IX / IS 是什么?

```
mysql> select * from t limit 1 lock in share mode;
```

```
+-----+
| id | c   |
+-----+
|  1 |  1 |
+-----+
```

```
1 row in set (0.00 sec)
```

```
mysql> select ENGINE_TRANSACTION_ID,THREAD_ID,OBJECT_NAME,index_name, OBJECT_INSTANCE_BEGIN,LOCK_TYPE,LOCK_MODE,LOCK_STATUS,LOCK_DATA from performance_schema.data_locks;
```

ENGINE_TRANSACTION_ID	THREAD_ID	OBJECT_NAME	index_name	OBJECT_INSTANCE_BEGIN	LOCK_TYPE	LOCK_MODE	LOCK_STATUS	LOCK_DATA
421416061812728	48	t	NULL	139940973304848	TABLE	IS	GRANTED	NULL
421416061812728	48	t	PRIMARY	139940973301792	RECORD	S	GRANTED	1

```
2 rows in set (0.002 sec)
```

# 课堂练习：表锁

`lock table t read, t2 write;`

**session1** 执行了这个语句后,以下说法正确的是:

- A. session1 可以对表 t 执行读操作;
- B. session1 可以对表 t 执行写操作;
- C. session1 可以对表 t2 执行读操作;
- D. session1 可以对表 t2 执行写操作;
- E. session1 可以对表 t3 执行读操作;
- F. session2 可以对表 t 执行读操作;
- G. session2 可以对表 t2 执行读操作;

# IX 和表锁

	X	IX	S	IS
X	Conflict	Conflict	Conflict	Conflict
IX	Conflict	Compatible	Conflict	Compatible
S	Conflict	Conflict	Compatible	Compatible
IS	Conflict	Compatible	Compatible	Compatible

	X	S
X	✖	✖
S	✖	☑

	IX	IS
IX	☑	☑
IS	☑	☑

	X lock write	S lock read
IX (for update)	✖	✖
IS (for share)	✖	☑



# 课堂练习

session1	session2
begin; select * from t where id>15 and id <20 for update;	
	begin; select * from t where id>15 and id <20 for update; //现象?

ENGINE_TRANSACTION_ID	THREAD_ID	OBJECT_NAME	index_name	OBJECT_INSTANCE_BEGIN	LOCK_TYPE	LOCK_MODE	LOCK_STATUS	LOCK_DATA
3327	75	t	NULL	130040603925344	TABLE	IX	GRANTED	NULL
3327	75	t	PRIMARY	130040603922288	RECORD	X,GAP	GRANTED	20
3328	53	t	NULL	130040603937872	TABLE	IX	GRANTED	NULL
3328	53	t	PRIMARY	130040603934896	RECORD	X,GAP	GRANTED	20

# Gap lock 干嘛用的

session1	session2
begin; select * from t where id>15 and id <20 for update;	
	begin; select * from t where id>15 and id <20 for update; //现象?

ENGINE_TRANSACTION_ID	THREAD_ID	OBJECT_NAME	index_name	OBJECT_INSTANCE_BEGIN	LOCK_TYPE	LOCK_MODE	LOCK_STATUS	LOCK_DATA
3327	75	t	NULL	130040603925344	TABLE	IX	GRANTED	NULL
3327	75	t	PRIMARY	130040603922288	RECORD	X,GAP	GRANTED	20
3328	53	t	NULL	130040603937872	TABLE	IX	GRANTED	NULL
3328	53	t	PRIMARY	130040603934896	RECORD	X,GAP	GRANTED	20

# Gap lock 干嘛用的

session1	session2
<pre>begin; select * from t where id&gt;15 and id &lt;20 for update;</pre>	
	<pre>begin; select * from t where id&gt;15 and id &lt;20 for update; insert into t values(17,17,17); //现象?</pre>

# Gap lock 干嘛用的

```
***** 5. row *****
ENGINE_TRANSACTION_ID: 3328
      THREAD_ID: 53
      OBJECT_NAME: t
      index_name: PRIMARY
OBJECT_INSTANCE_BEGIN: 130040603935248
      LOCK_TYPE: RECORD
      LOCK_MODE: X,GAP,INSERT_INTENTION
      LOCK_STATUS: WAITING
      LOCK_DATA: 20
5 rows in set (0.00 sec)
```



# Gap lock 干嘛用的

session1	session2
<pre>begin; select * from t where id&gt;15 and id &lt;20 for update;</pre>	
	<pre>begin; select * from t where id&gt;15 and id &lt;20 for update; insert into t values(17,17,17); //现象?</pre>
<pre>insert into t values(17,17,17); //现象?</pre>	

# LOCK\_MODE 的可选值

lock mode
X / S , [GAP/REC_NOT_GAP]

```
begin;  
select * from t where id>=15 and id <=20  
for update;
```



ENGINE_TRANSACTION_ID	THREAD_ID	OBJECT_NAME	index_name	OBJECT_INSTANCE_BEGIN	LOCK_TYPE	LOCK_MODE	LOCK_STATUS	LOCK_DATA
3329	75	t	NULL	130040603925344	TABLE	IX	GRANTED	NULL
3329	75	t	PRIMARY	130040603922288	RECORD	X, REC_NOT_GAP	GRANTED	15
3329	75	t	PRIMARY	130040603922640	RECORD	X	GRANTED	20

# Gap lock 干嘛用的

```
begin;  
select * from t for update;
```

ENGINE_TRANSACTION_ID	THREAD_ID	OBJECT_NAME	index_name	OBJECT_INSTANCE_BEGIN	LOCK_TYPE	LOCK_MODE	LOCK_STATUS	LOCK_DATA
3331	75	t	NULL	130040603925344	TABLE	IX	GRANTED	NULL
3331	75	t	PRIMARY	130040603922288	RECORD	X	GRANTED	supremum pseudo-record
3331	75	t	PRIMARY	130040603922288	RECORD	X	GRANTED	0
3331	75	t	PRIMARY	130040603922288	RECORD	X	GRANTED	5
3331	75	t	PRIMARY	130040603922288	RECORD	X	GRANTED	10
3331	75	t	PRIMARY	130040603922288	RECORD	X	GRANTED	15
3331	75	t	PRIMARY	130040603922288	RECORD	X	GRANTED	20
3331	75	t	PRIMARY	130040603922288	RECORD	X	GRANTED	25

(25, supremum]

(20,25]

# Thread\_id 字段

```
begin;  
select * from t where id>15 and id <20 for  
update;
```

ENGINE_TRANSACTION_ID	THREAD_ID	OBJECT_NAME	index_name	OBJECT_INSTANCE_BEGIN	LOCK_TYPE	LOCK_MODE	LOCK_STATUS	LOCK_DATA
3348	75	t	NULL	130040603925344	TABLE	IX	GRANTED	NULL
3348	75	t	PRIMARY	130040603922288	RECORD	X,GAP	GRANTED	20



# Thread\_id 字段

```
session1
begin;
select * from t where id>15 and id <20 for
update;
insert into t values(17,17,17);
```

执行 insert 后, 加了以下哪些锁:

- A. (15,17]
- B. (15,17)
- C. (17, 20)
- D. (17,20]
- E. (15,20)

ENGINE_TRANSACTION_ID	THREAD_ID	OBJECT_NAME	index_name	OBJECT_INSTANCE_BEGIN	LOCK_TYPE	LOCK_MODE	LOCK_STATUS	LOCK_DATA
3338	75	t	NULL	130040603925344	TABLE	IX	GRANTED	NULL
3338	75	t	PRIMARY	130040603922288	RECORD	X	GRANTED	17
3338	75	t	PRIMARY	130040603922640	RECORD	X,GAP	GRANTED	20

# Thread\_id 字段

```
session1
begin;
select * from t where id>15 and id <20 for
update;
insert into t values(17,17,17);
```

执行insert后,加了以下哪些锁?

(15,17)  
(17,  
20)  
id=17

```
session2
insert into t
values(17,17,17);
```

ENGINE_TRANSACTION_ID	THREAD_ID	OBJECT_NAME	index_name	OBJECT_INSTANCE_BEGIN	LOCK_TYPE	LOCK_MODE	LOCK_STATUS	LOCK_DATA
3357	53	t	NULL	130040603937872	TABLE	IX	GRANTED	NULL
3357	53	t	PRIMARY	130040603934896	RECORD	S,REC_NOT_GAP	WAITING	17
3356	75	t	NULL	130040603925344	TABLE	IX	GRANTED	NULL
3356	75	t	PRIMARY	130040603922288	RECORD	X,GAP	GRANTED	20
3356	75	t	PRIMARY	130040603922288	RECORD	X,GAP	GRANTED	17
3356	53	t	PRIMARY	130040603922640	RECORD	X,REC_NOT_GAP	GRANTED	17

# 死锁和死锁检测

怎么发现死锁

**A==>B**

**B==>A**



# 死锁要回滚了选谁回滚？

为了解决这个问题，数据库系统实现了各种形式的死锁检测和超时。更复杂系统，如 InnoDB 存储引擎，将注意到循环依赖关系并立即返回错误。这可能是件好事——否则，死锁将以非常缓慢的查询表现出来。其他系统将在查询超过锁定等待超时后放弃，这并不总是好事。InnoDB 目前处理死锁的方式是回滚具有最少独占行锁（最容易回滚的近似度量）的事务。

```
create table t1(id int primary key, c int, d int, index(c));
```

```
insert into t1 values(1,1,1),(2,2,2),(3,3,3).....(10,10,10);
```

```
create table t2(id int primary key, c int, d int);
```

```
insert into t2 values(1,1,1),(2,2,2),(3,3,3).....(10,10,10);
```



以下操作序列，死锁后，回滚 session1 还是 2？

session1	session2
begin; select * from t1 where id=1 for update;	
	begin; select * from t1 where id=2 for update; select * from t1 where id=1 for update; //blocked
select * from t1 where id=2 for update;	

# 相关知识点：死锁检测是异步发生的，先锁再检测

session1	session2
begin; select * from t1 where id=1 for update;	
	begin; select * from t1 where id=2 for update; select * from t1 where id=1 for update; //blocked
select * from t1 where id=2 for update;	

回滚 session 1

# 以下操作序列，死锁后，回滚 session1 还是 2?

session1	session2
<pre>begin; select * from t1 where id=1 for update; select * from t1 where id=3 lock in share mode ;</pre>	
	<pre>begin; select * from t1 where id=2 for update; select * from t1 where id=4 for update; select * from t1 where id=1 for update; //blocked</pre>
<pre>select * from t1 where id=2 for update;</pre>	

相关知识点：同一个page上，相同类型的锁，只算一个

相关知识点：意向锁也算1个

# 以下操作序列，死锁后，回滚 session1 还是 2?

session1	session2
<pre>begin; select * from t1 where id=1 for update; select * from t1 where id=3 lock in share mode ;</pre>	
	<pre>begin; select * from t1 where id=2 for update; select * from t1 where id in (4,5,6,7,8) for update; select * from t1 where id=1 for update; //blocked      lock数3</pre>
<pre>select * from t1 where id=2 for update; lock数4</pre>	回滚session2

# 相关知识点：事务权重的算法

```
/** Calculates the "weight" of a transaction. The weight of one transaction
    is estimated as the number of altered rows + the number of locked rows.
    @param t transaction
    @return transaction weight */
static inline uint64_t TRX_WEIGHT(const trx_t *t) {
    return t->undo_no + UT_LIST_GET_LEN(t->lock.trx_locks);
}
```



# 以下操作序列，死锁后，回滚 session1 还是 2?

session1	session2
<pre>begin; select * from t1 where id=1 for update; update t1 set d=d+1 where id =3; update t1 set d=d+1 where id=5;</pre>	
	<pre>begin; select * from t1 where id=2 for update; update t1 set d=d+1 where id in(4,6); select * from t1 where id=1 for update; //blocked lock数3, undo_no:2</pre>
<pre>select * from t1 where id=2 for update; lock数3, undo_no:2 回滚session 1</pre>	

# 以下操作序列，死锁后，回滚 session1 还是 2?

session1	session2
<pre>begin; select * from t1 where id=1 for update; select * from t1 where c=3 limit 1 for update;</pre>	
<pre>(-8,1], (1,2], (2,3], (3,4)</pre>	<pre>begin; select * from t1 where id=2 for update; select * from t 1 where id = 4 lock in share mode; select * from t1 where id=1 for update; //blocked</pre>
<pre>select * from t1 where id=2 for update; lock数5</pre>	<pre>lock数4 回滚session 2</pre>

# 以下操作序列，死锁后，回滚 session1 还是 2?

session1	session2
<pre>begin; select * from t1 where id=1 for update; select * from t1 where c=3 for update;</pre>	
<pre>(-8,1], (1,2], (2,3], (3,4)</pre>	<pre>begin; select * from t1 where id=2 for update; select * from t 1 where id = 4 lock in share mode; select * from t1 where id=1 for update; //blocked</pre>
<pre>select * from t1 where id=2 for update; lock数5</pre>	<pre>lock数4 回滚session 2</pre>

# 语句顺序和锁策略

方案1	方案2
<pre>begin; insert 购买记录 update 账户余额 if 余额不足 rollback = update 库存 where goodlid = k and left_num&gt;=1 if fail rollback; commit</pre>	<pre>begin; select left_num for update where id= if left_num&gt;x update .. set left_num=left_num-200  where id=ID and left_num&gt;200 if affectedr_rows()==0; rollabck;  update 库存 if fail rollback; update 余额 if 余额不足 rollback insert 购买记录 commit</pre>

# 语句顺序和锁策略

减少锁时间     `commit_on_success`

对于MySQL 5.7和MySQL 8.0版本：

- 两个事务Hint为COMMIT\_ON\_SUCCESS和ROLLBACK\_ON\_FAIL：
  - COMMIT\_ON\_SUCCESS：当前语句执行成功就提交事务上下文。
  - ROLLBACK\_ON\_FAIL：当前语句执行失败就回滚事务上下文。

- 条件Hint为TARGET\_AFFECT\_ROW(NUMBER)：  
如果当前语句影响行数是指定的就成功，否则语句失败。  
语法：

```
/*+ TARGET_AFFECT_ROW(NUMBER) */
```



# 语句顺序和锁策略

减少锁时间     `commit_on_success + target_affect_row(1)`

```
mysql> UPDATE /*+ commit_on_success rollback_on_fail target_affect_row(1) */ t  
      -> SET col1 = col1 + 1  
      -> WHERE id = 1;
```

# Q&A

THANKS