浅析MySQL事务隔离级别与锁

周晓 - EC运维组

2016.08.31

http://seanlook.com

Agenda

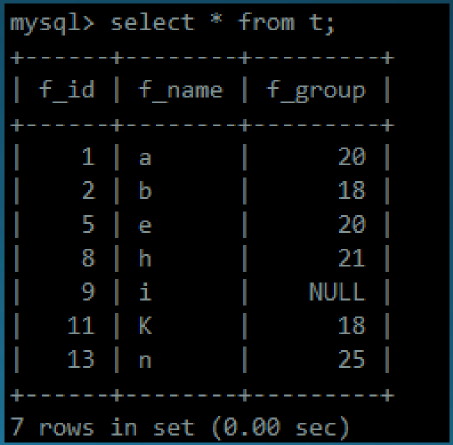
- MySQL锁类型
- 事务的实现
- 四大隔离级别
- 事务加锁处理
- 几例实际锁问题分析

Thinking

- select * from t where f_id=9

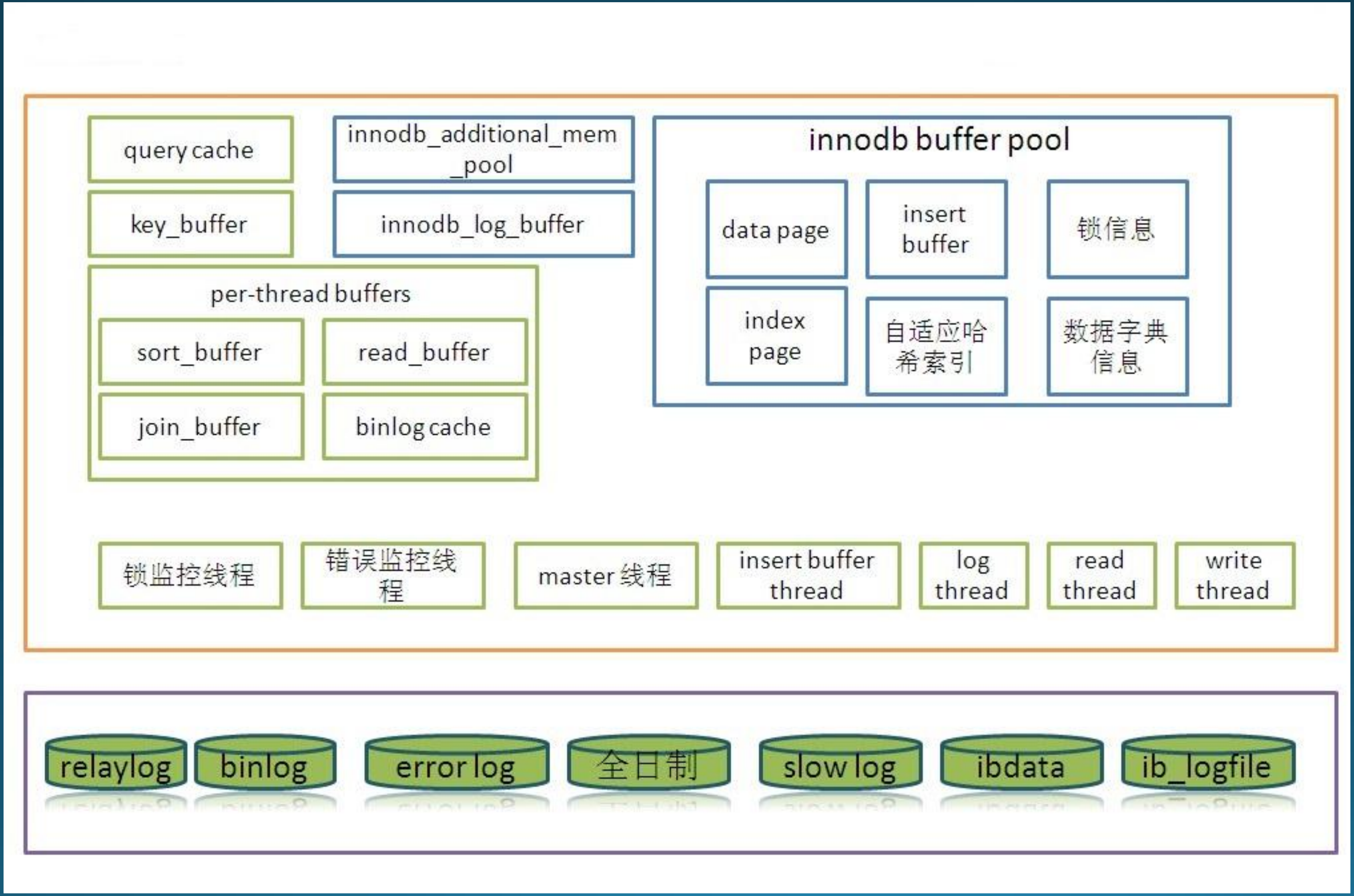
会锁定多少行记录?

```
CREATE TABLE `t` (
  `f_id` int(11) AUTO_INCREMENT NOT NULL,
  `f_name` varchar(30) NOT NULL DEFAULT  ‘’,
  `f_group` tinyint(3) unsigned,
  PRIMARY KEY (`f_id`),
  UNIQUE KEY `idx_name` (`f_name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```



```
mysql> select * from t;
+------+--------+---------+
| f_id | f_name | f_group |
+------+--------+---------+
|    1 | a      |      20 |
|    2 | b      |      18 |
|    5 | e      |      20 |
|    8 | h      |      21 |
|    9 | i      |    NULL |
|   11 | K      |      18 |
|   13 | n      |      25 |
+------+--------+---------+
7 rows in set (0.00 sec)
```

InnoDB体系结构


query cache
innodb_additional_mem_pool
innodb buffer pool
key_buffer
innodb_log_buffer
data page
insert buffer
锁信息
per-thread buffers
sort_buffer
read_buffer
join_buffer
binlog cache
index page
自适应哈希索引
数据字典信息
锁监控线程
错误监控线程
master 线程
insert buffer thread
log thread
read thread
write thread
relaylog
binlog
error log
全日制
slow log
ibdata
ib_logfile

1. MySQL锁类型

- 共享锁与排它锁
- 粒度(granular)
  - 行锁、页锁、表锁
- 意向锁(Intension Lock)

检测表锁和行锁之间的冲突
为的是准备上更细粒度的锁

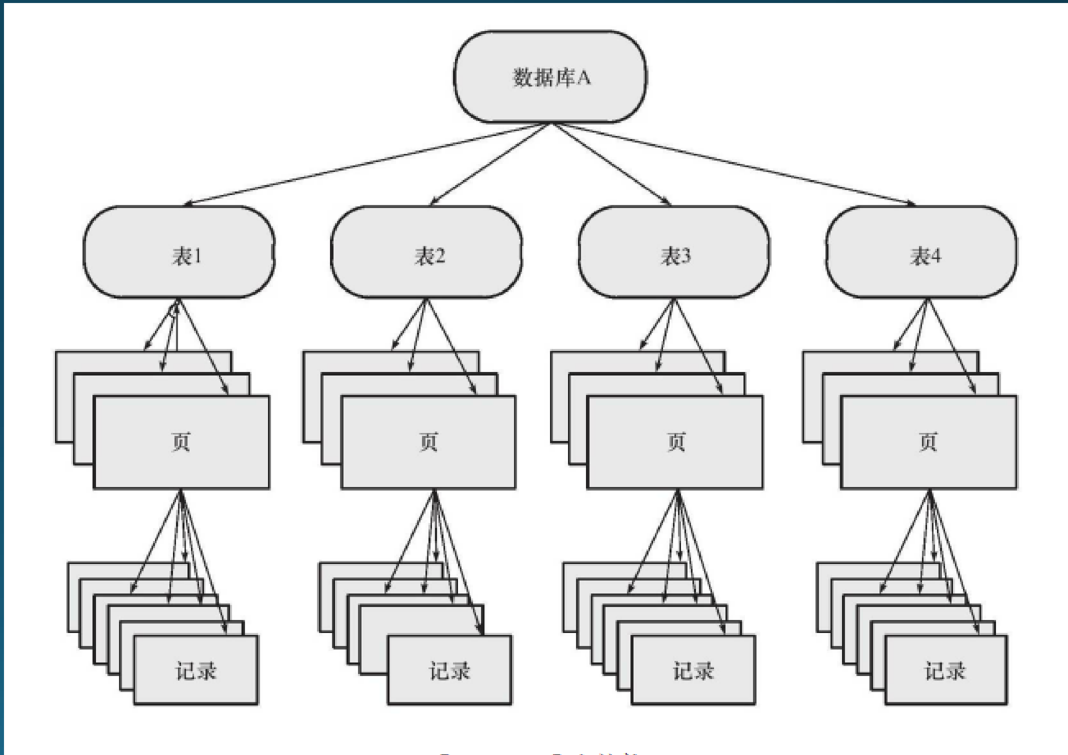


图 6-3 层次结构

1. MySQL锁类型

- 共享锁与排它锁
- 粒度(granular)
  - 行锁、页锁、表锁
- 意向锁(Intension Lock)

session A	session B
BEGIN WORK select count(*)	BEGIN WORK update a record

InnoDB Lock Compatity	S	X	IS	IX
S	兼容	不兼容	兼容	不兼容
X	不兼容	不兼容	不兼容	不兼容
IS	兼容	不兼容	兼容	兼容
IX	不兼容	不兼容	兼容	兼容

1. MySQL锁类型

- 乐观锁与悲观锁
- 隐式锁与显式锁

两段封锁法

```
SELECT * FROM t where f_id=1
SELECT * FROM t where f_id=1 FOR update
SELECT * FROM t where f_id>10 LOCK IN share mode
insert into table values (…)
update table set ? where ?
delete from table where ?
```

2. 事务的实现

- 事务的四个属性
  - Atomicity
    - 事务被认为不可分的一个工作单元，要么全部正常执行，要么全部不执行
  - Consistency
    - 事务操作对数据库总是从一种一致性的状态转换成另外一种一致性状态
  - Isolation
    - 某个事务的结果只有在完成之后才对其他事务可见
  - Durability
    - 事务在未提交前数据一般情况下可以回滚恢复数据，一旦提交(commit)数据的改变则变成永久(当然用update肯定还能修改)

2. 事务的实现

- REDO
  - Write Ahead Log (buffer)
  - Checkpoint (LSN)
  - Rotate write (顺序IO)
- UNDO
  - Rollback
  - Consistency Unlock Read
- 2 Phase Commit
  - innodb_flush_log_at_trx_commit
  - sync_binlog

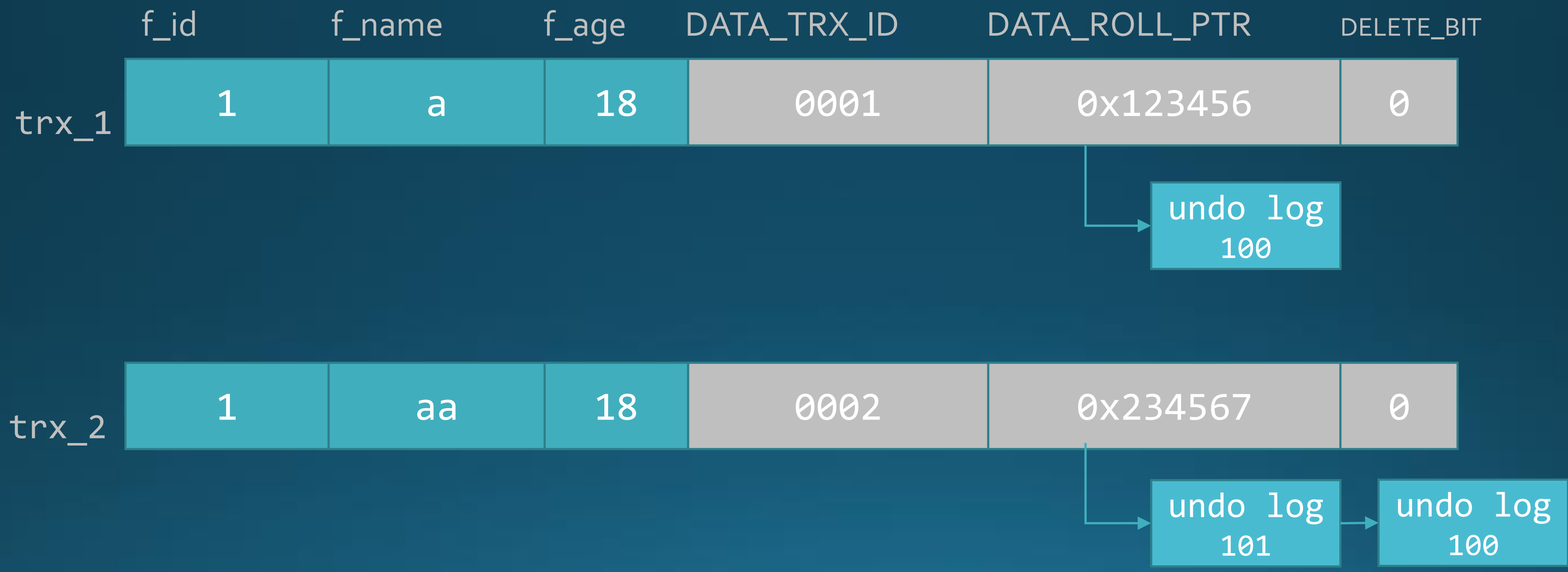
UNDO	-> 原子性
REDO	-> 永久性
Lock+Undo	-> 隔离性
Constraint	-> 一致性

2. 事务的实现

- 当前读与快照读
- 多版本并发控制技术(MVCC)
  - SELECT时，读取创建版本号<=当前事务版本号，删除版本号为空或>当前事务版本号。
  - INSERT时，保存当前事务版本号为行的创建版本号
  - DELETE时，保存当前事务版本号为行的删除版本号
  - UPDATE时，插入一条新纪录，保存当前事务版本号为行创建版本号，同时保存当前事务版本号到原来删除的行

2. 事务的实现

- 多版本并发控制技术(MVCC)



	f_id	f_name	f_age	DATA_TRX_ID	DATA_ROLL_PTR	DELETE_BIT
trx_1	1	a	18	0001	0x123456	0
trx_2	1	aa	18	0002	0x234567	0

3.四大隔离级别

- READ-UNCOMMITTED
- READ-COMMITTED (default for RDS)
- REPEATABLE-READ (default for Official)
- SERIALIZABLE

隔离级别	脏读 (Dirty Read)	不可重复读 (Non-Repeatable Read)	幻读 (Phantom Read)
读未提交 (Read Uncommitted)	YES	YES	YES
已提交读 (Read Committed)	NO	YES	YES
可重复读 (Repeatable Read)	NO	NO	YES ?
可串行化 (Serializable)	NO	NO	NO

3.四大隔离级别

分别在RC和RR级别下执行

	session A	session B
1	START TRANSACTION	START TRANSACTION
2	select * from t where f_id>0 and f_id<5;	
3		update t set f_name='aa' where f_id=1;
4		insert into t values(3,'c');
5	select * from t where f_id>0 and f_id<5;	
6		COMMIT
7	select * from t where f_id>0 and f_id<5;	
8	COMMIT	
9	select * from t where f_id>0 and f_id<5;	

3.四大隔离级别

分别在RC和RR级别下执行

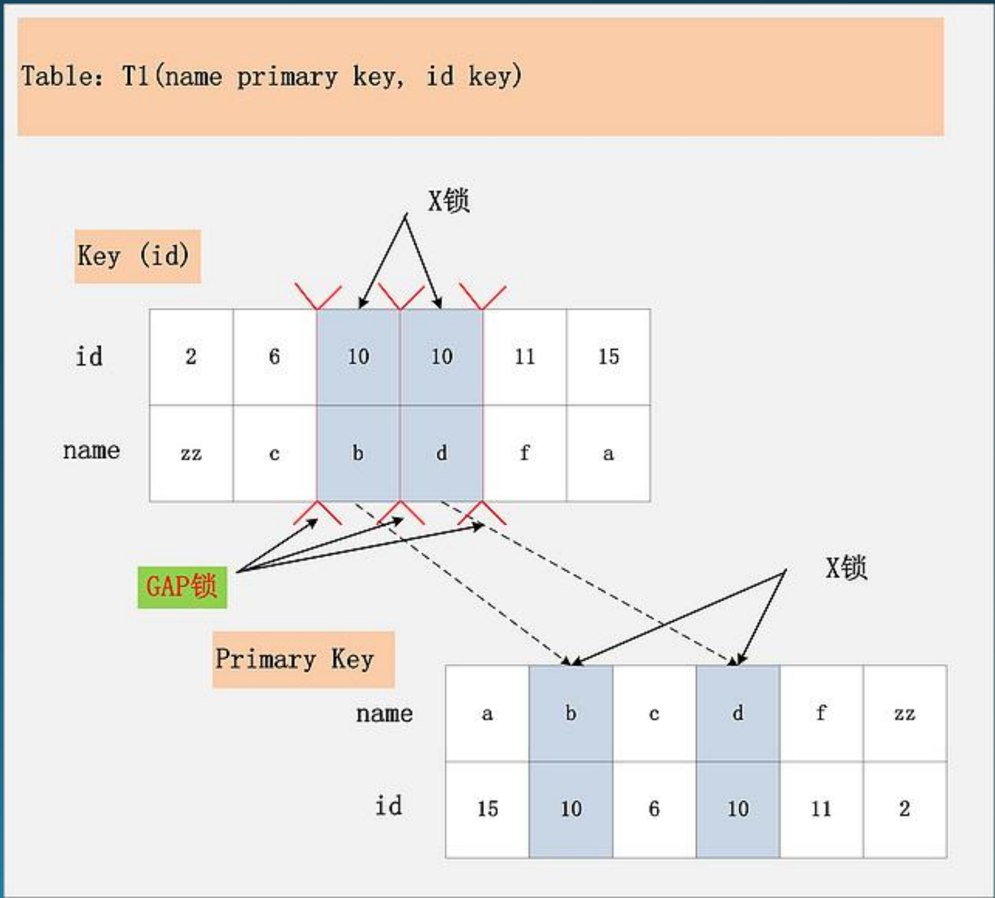
	session A	session B
1	START TRANSACTION	START TRANSACTION
2	select * from t where f_group>10 and f_gender=1 for update;	
3		update t set f_gender=0 where f_id=99;

```
RC: 0 rows affected
RR: blocked
```

4. 事务加锁处理

- 行锁模式
  - Record Lock
  - Gap Lock
  - Next-key Lock

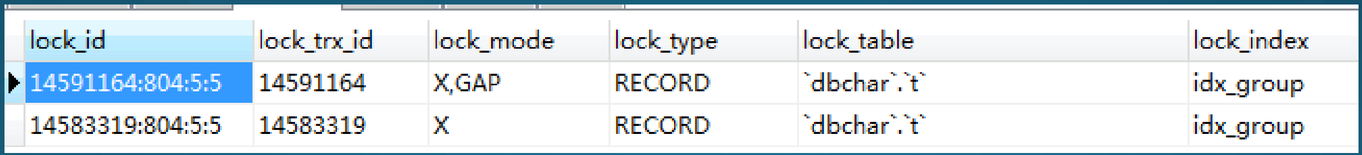
```
delete from t1 where id = 10;
```



4. 事务加锁处理

- update t set f_group=0 where f_group < 20
- select * from t where f_name=zz for update;

RR	session A	session B
1	START TRANSACTION	START TRANSACTION
2	update t set f_group=0 where f_group < 20	
3		insert into t values(111,'10K',18);
4	commit	



lock_id	lock_trx_id	lock_mode	lock_type	lock_table	lock_index
14591164:804:5:5	14591164	X,GAP	RECORD	\`dbchar\`.\`t\`	idx_group
14583319:804:5:5	14583319	X	RECORD	\`dbchar\`.\`t\`	idx_group

5.实际锁问题分析

- select * from t where f_group >20 and f_gender=1
- 外键与锁的关系

- replace into seq死锁检测

5.实际锁问题分析

- 死锁检测
  - replace into t(f_name) values('cc')

	session A	session B	session C
1	START TRANSACTION	START TRANSACTION	START TRANSACTION
2	INSERT INTO t (f_name) VALUES ('cc');		
3		INSERT INTO t (f_name) VALUES ('cc');	
4			INSERT INTO t (f_name) VALUES ('cc');
5	rollback		

```
ERROR 1213 (40001): Deadlock found when trying
to get lock; try restarting transaction
```

References

- http://dev.mysql.com/doc/refman/5.6/en/innodb-locking.html
- http://tech.meituan.com/innodb-lock.html
- http://mysql.taobao.org/monthly/2015/12/01/?spm=5176.100239.blogcont4270.9.cwMW19
- http://hedengcheng.com/?p=771

ec
用连接改变生意

Q&A