

# MySQL 加锁分析

范佚伦

数据来自MySQL 5.7 InnoDB

# Agenda

- 为什么要加锁
- 怎样加锁
- 隔离级别
- 不同语句的加锁情况
- 死锁示例
- 避免死锁

# 为什么要加锁

- 保障事务的ACID，不同事务之间互不干扰
- 锁可以做并发控制，保证事务的一致性
- 事务的一致性需要解决的问题
  - 避免脏读
  - 避免不可重复读
  - 避免幻读

# 怎样加锁

- 传统加锁思想：读加共享锁，写加互斥锁
- 读读并行



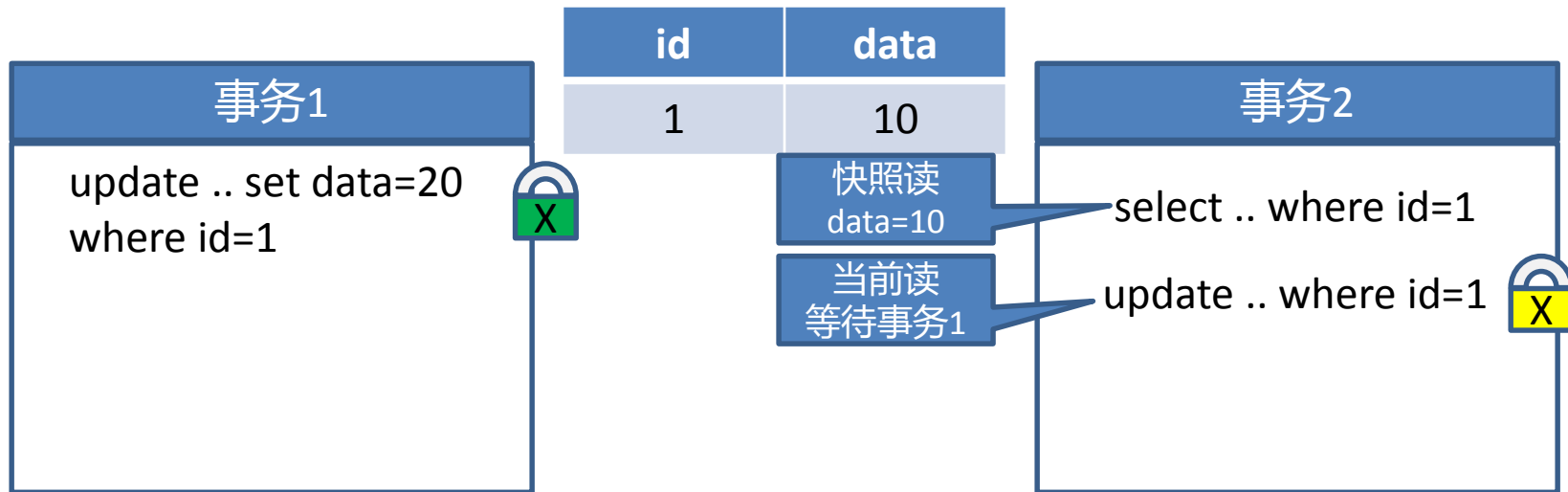
# 怎样加锁

- 进一步增强并发能力的办法：多版本（MVCC）
- 【读】在MySQL中的分成两种语义
  - 快照读(consistent read)：普通select
  - 当前读：update, delete, select...for update
- 读写并行



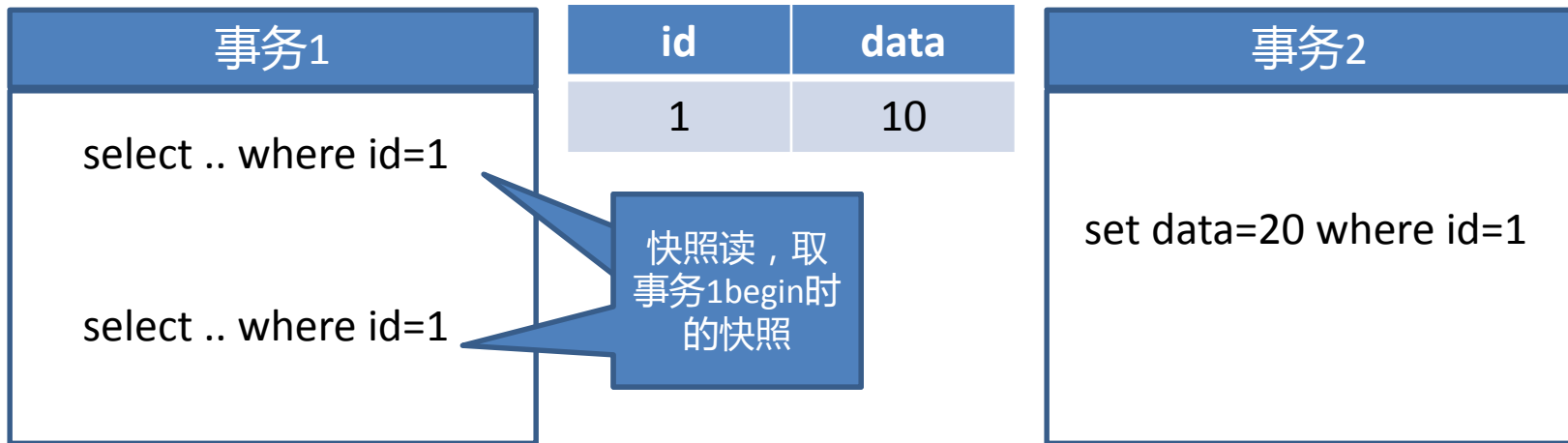
# 怎样加锁

- 避免脏读
  - 一个事务commit之前，其写入的数据不能被其他事务读到



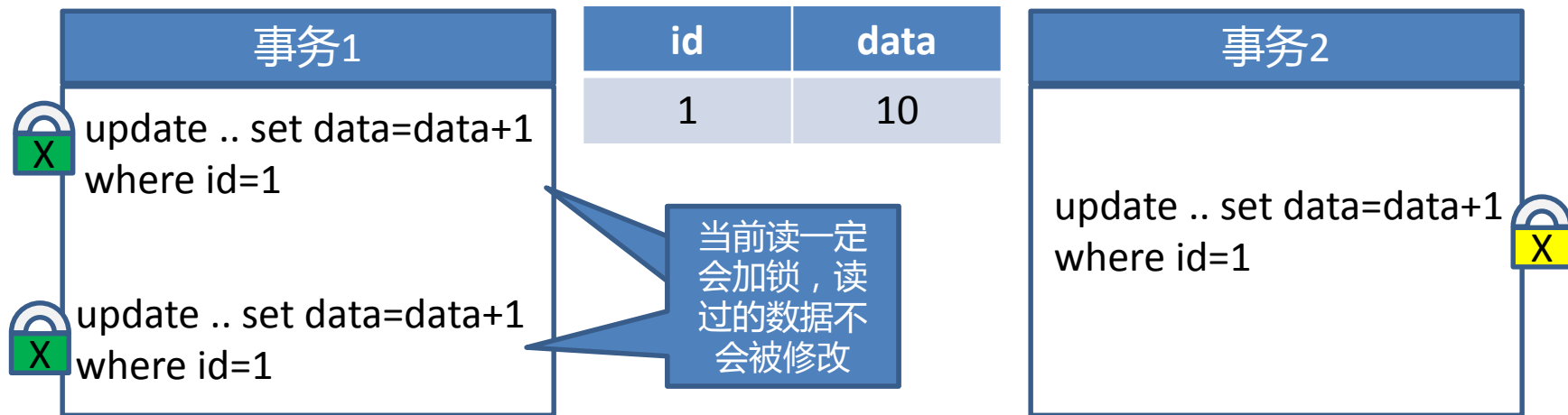
# 怎样加锁

- 避免不可重复读
  - 同样的条件，读取过的数据，再次读取出来的值不能发生变化。



# 怎样加锁

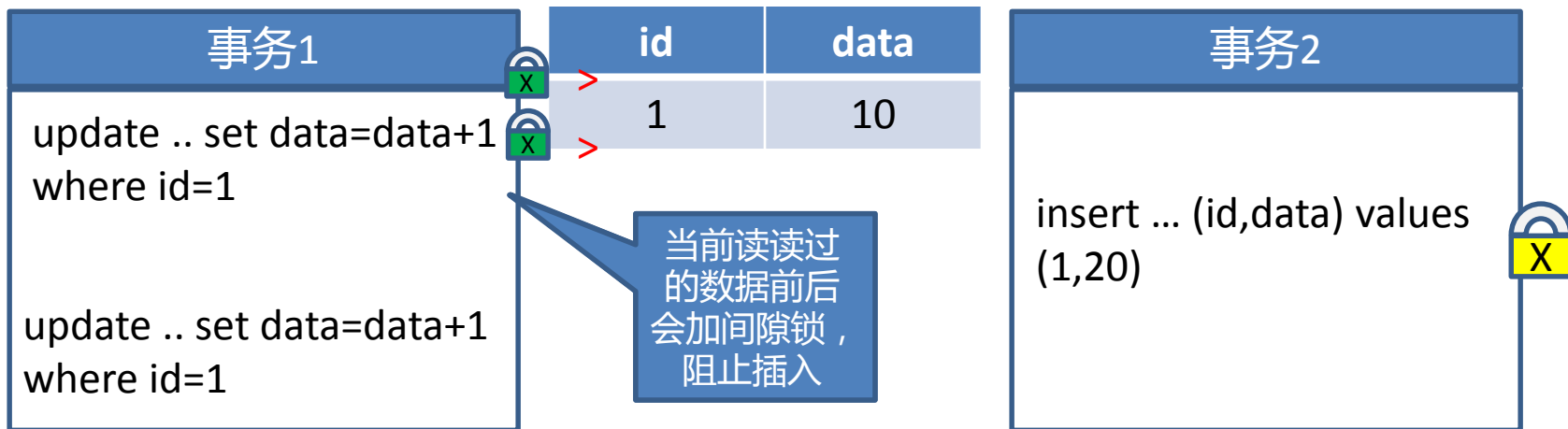
- 避免不可重复读
  - 同样的条件，读取过的数据，再次读取出来的值不能发生变化。





# 怎样加锁

- 避免幻读
  - 同样的条件，读取过的数据，再次读取出来的数目不能发生变化。



# 隔离级别

- SQL92标准的隔离级别

	脏读	不可重复读	幻读
Read Uncommitted	允许	允许	允许
Read Committed	不允许	允许	允许
Repeatable Read	不允许	不允许	允许
Serializable	不允许	不允许	不允许

# 隔离级别

- MySQL的隔离级别

	脏读	不可重复读	幻读	
Read Uncommitted	允许	允许	允许	
Read Committed	不允许	允许	允许	公司使用的级别
Repeatable Read	不允许	不允许	不允许	MySQL默认级别
Serializable	不允许	不允许	不允许	


```
SET @a = (SELECT num FROM book WHERE id =1)
SET @b = function(@a)  //某个业务逻辑操作
UPDATE book SET num=@b WHERE id =1
```

Repeatable Read 不安全—— select..for update

Serializable 不安全

# 不同语句的加锁情况 ( RC )

- 查询命中聚簇索引
  - 在所有命中的聚簇索引上加锁
- 例
  - `update my_table set name='a' where id=1;`



id	1	5	8
name	aaa	bbb	bbb

# 不同语句的加锁情况 ( RC )

- 查询命中二级索引
  - 在所有命中的二级索引上锁
  - 在所有命中的行的聚簇索引上加锁
- 例
  - `delete from my_table where name='bbb';`

name	aaa	bbb	bbb
id	1	5	8

name列索引

id	1	5	8
name	aaa	bbb	bbb

聚簇索引

# 不同语句的加锁情况 ( RC )

- 对索引键值有修改
  - 修改索引键值，实际上是删掉原值，重新插入一条索引记录
  - 因此原值和新值都会被加X锁
- 例
  - `update my_table set name='ccc' where id=1 ;`

name	aaa	bbb	bbb	ccc
id	1	5	8	1

name索引

id	1	5	8
name	aaa	bbb	bbb

聚簇索引

# 不同语句的加锁情况 ( RC )

- 查询没有命中索引
  - InnoDB先会锁住聚簇索引的所有记录
  - 然后MySQL Server会过滤，把不符合条件的锁当即释放掉
  - 例： `delete from my_table where name='aaa';`

id	1	5	8
name	aaa	bbb	bbb

聚簇索引

# 不同语句的加锁情况 ( RC )

- 查询没有命中索引
  - 特别地，对于update语句，如果要加锁的记录上已经有锁，innodb不会立即锁等待，而是执行semi-consistent read：返回该数据上一次提交的快照版本，供MySQL Server层判断是否需要加锁

Transaction 1	Transaction 2
update ... where id=5;	
	update ... where name='aaa';



id	1	5	8
name	aaa	bbb	bbb






# 不同语句的加锁情况 ( RC )

- 插入数据
  - 唯一索引冲突检查：目标键值加S锁
  - 插入的记录加X锁
  - 例：insert into my\_table (id,name) values(9,'ccc');












id	1	5	8	9
name	aaa	bbb	bbb	ccc



聚簇索引

# 死锁举例 ( RC )

Transaction 1	Transaction 2
insert into my_table (id,name) values(9,'ccc');	
	insert into my_table (id,name) values(10,'ccc');
delete from my_table where name='aaa';	
	delete from my_table where name='aaa';

					 
					
id	1	5	8	9	10
name	aaa	bbb	bbb	ccc	ccc


聚簇索引|(name列无索引|)

# 避免死锁

- 业务允许的情况下，隔离级别设置成RC
- 写操作一定要有命中索引，避免全表扫描
- 表上有太多索引（>5），也会有死锁的风险
- 不同事务之间，表操作和记录操作的顺序要一致

# 不同语句的加锁情况 ( RR )

- 查询命中聚簇索引 ( 精确匹配 )
  - 在所有命中的聚簇索引上加锁
  - 与RC一样
- 例
  - `update my_table set name='a' where id=1;`



id	1	5	8
name	aaa	bbb	bbb

# 不同语句的加锁情况 ( RR )

- 查询命中二级索引
  - 在所有命中的二级索引上锁
  - 在所有命中的行的聚簇索引上加锁
  - 如果不是唯一索引，还会在这个二级索引前后加gap lock
- 例
  - `delete from my_table where name='bbb';`

name	aaa	bbb	bbb	
id	1	5	8	

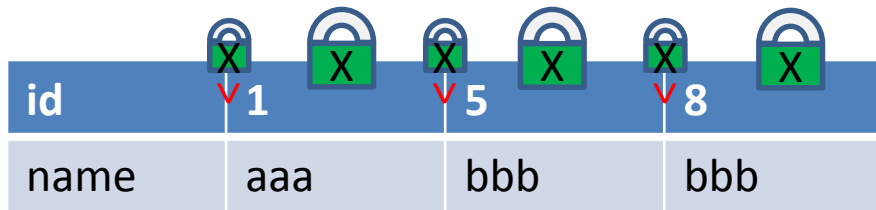
name列索引

id	1	5	8
name	aaa	bbb	bbb

聚簇索引

# 不同语句的加锁情况 ( RR )

- 查询没有命中索引
  - 对全表的所有聚簇索引加锁
  - 对全表聚簇索引的所有间隙加gap lock
- 例如
  - `update my_table set ... where name='aaa';`



聚簇索引

全表写阻塞

**感谢您的观看**

