DB Tech Club

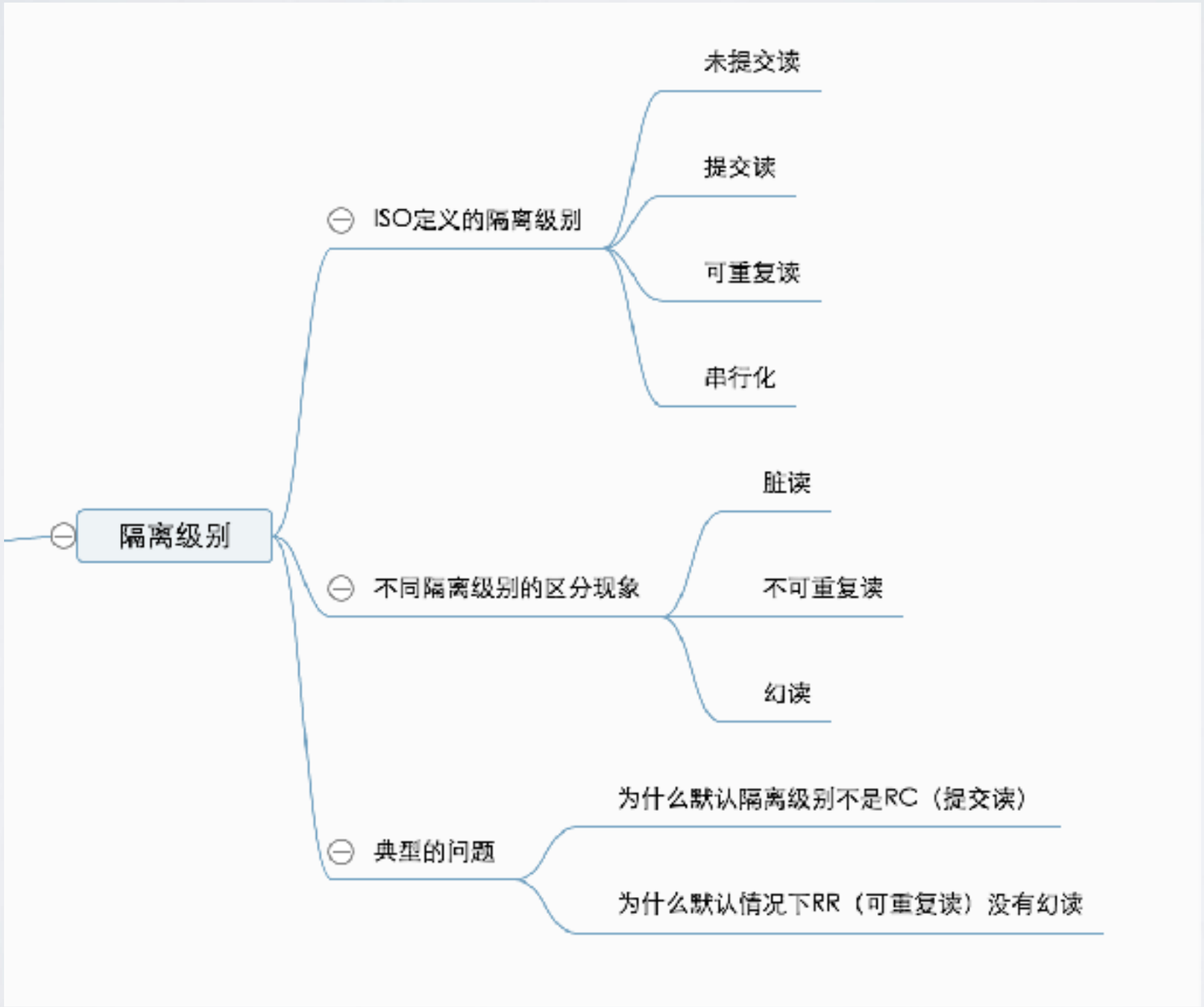
InnoDB隔离级别与锁

房晓乐

主要内容

- 隔离级别
    - ISO定义的隔离级别
    - 不同隔离级别的区分现象
    - 几个典型的问题
- 行锁介绍
    - 锁和隔离级别的关系
    - 行锁的基本属性
    - sql语句和加锁的关系
    - 行锁的冲突规则
    - InnoDB针对锁的优化

主要内容脑图（1）


隔离级别
ISO定义的隔离级别
未提交读
提交读
可重复读
串行化
不同隔离级别的区分现象
脏读
不可重复读
幻读
典型的问题
为什么默认隔离级别不是RC（提交读）
为什么默认情况下RR（可重复读）没有幻读

隔离级别

隔离级别

- ISO定义的隔离级别
- 不同隔离级别的区分现象
- 几个典型的问题

数据库隔离级别（简介）

- 隔离级别的目的
  - 定义并发事务之间互相影响的程度，两个并发事务之间能看到对方哪些数据。
- ISO定义的数据库隔离级别
  - Read uncommitted（未提交读）
  - Read committed（提交读，RC）
  - Repeatable read（可重复读，RR）
  - Serializable（全串行）

隔离级别

- ISO定义的隔离级别
- 不同隔离级别的区分现象
- 几个典型的问题

数据库隔离级别（基本现象）

- 脏读
  - 事务A修改了数据data，但并未提交;与此同时事务B读到了data的修改，称为脏读。
- 不可重复读
  - 在事务A的两次读data之间，事务B访问了数据data，并修改了data，并进行了提交。
  - 如果事务A的前后两次读由于事务B的修改，导致的不一致称为不可重复读。
  - 不可重复读的重点是修改:同样的条件的select, 你读取过的数据, 再次读取出来发现值不一样。
- 幻读（phantom）
  - 事务A读db内的某一范围数据并进行了修改，与此同时事务B在该范围新增一行insert_data。
  - 事务A可重复读，能读到原数据，但提交时发现有之前未查到的数据insert_data。
  - 幻读的重点在于新增或者删除:同样的条件的select, 读出来的记录数不一样。

ISO隔离级别对应关系

隔离级别	脏读	不可重复读	幻读
Read uncommitted	可能	可能	可能
Read committed	不可	可能	可能
Repeatable read（default）	不可	不可	可能 (innodb默认不可能)
Serializable	不可	不可	不可

locks_unsafe_for_binl

隔离级别

- ISO定义的隔离级别
- 不同隔离级别的区分现象
- 几个典型的问题

问题：

- innodb的默认隔离级别不是RC?
- 为什么innodb的RR（Repeatable Read）在默认情况不会有幻读?
  - **statement-based binary logging**

解答

- 为什么innodb的默认隔离级别不是RC?
  - 为了支持statement-based binary logging（基于提交去同步主从），即为了支持基于statement的binlog
  - 在RC的隔离级别且statement下，主从同步会导致问题。
  - 参考：http://dev.mysql.com/doc/refman/5.7/en/binary-log-setting.

If you are using `InnoDB` tables and the transaction isolation level is `READ COMMITTED` or `READ UNCOMMITTED`, only row-based logging can be used. It is ***possible*** to change the logging format to `STATEMENT`, but doing so at runtime leads very rapidly to errors because `InnoDB` can no longer perform inserts.

为什么默认隔离级别不是RC（举例）

txn1： update t2 set c2 = c2+1 where c1 in (select c1 from t1);
txn2： update t1 set c1 = c1+1 where c1 = 2;
txn2： commit;
txn1： commit;

主库的执行情况

txn2： update t1 set c1 = c1+1 where c1 = 2;
txn2： commit;
txn1： update t2 set c2 = c2+1 where c1 in (select c1 from t1);
txn1： commit;

从库的执行情况

解答

- 为什么innodb的RR在默认情况不会有幻读?
  - 默认情况下禁用了locks_unsafe_for_binlog（即lock_safe_for_binlog）
  - 扫描时使用的next-key locks锁住了插入，可以保证避免幻读
  - https://dev.mysql.com/doc/refman/5.6/en/innodb-parameters.html#sysvar_innodb_locks_unsafe_for_binlog
  - By default, the value of innodb_locks_unsafe_for_binlog is 0 (disabled), which means that gap locking is enabled: InnoDB uses next-key locks for searches and index scans. To enable the variable, set it to 1. This causes gap locking to be disabled: InnoDB uses only index-record locks for searches and index scans.

为什么RR没有幻读（举例）

txn1： update t2 set c2 = c2+1 where c1 in (select c1 from t1);
txn2： insert into t1(c1, c2, c3) values(1,2,3);
txn2： commit;
txn1： commit;

主库的执行情况

txn2： insert into t1 (c1, c2, c3) values(1,2,3);
txn2： commit;
txn1： update t2 set c2 = c2+1 where c1 in (select c1 from t1);
txn1： commit;

从库的执行情况

上例解析补充

- txn2提交的事务会因为txn1提交的事务上的next-key lock而锁住提交，因此不会发生幻读。

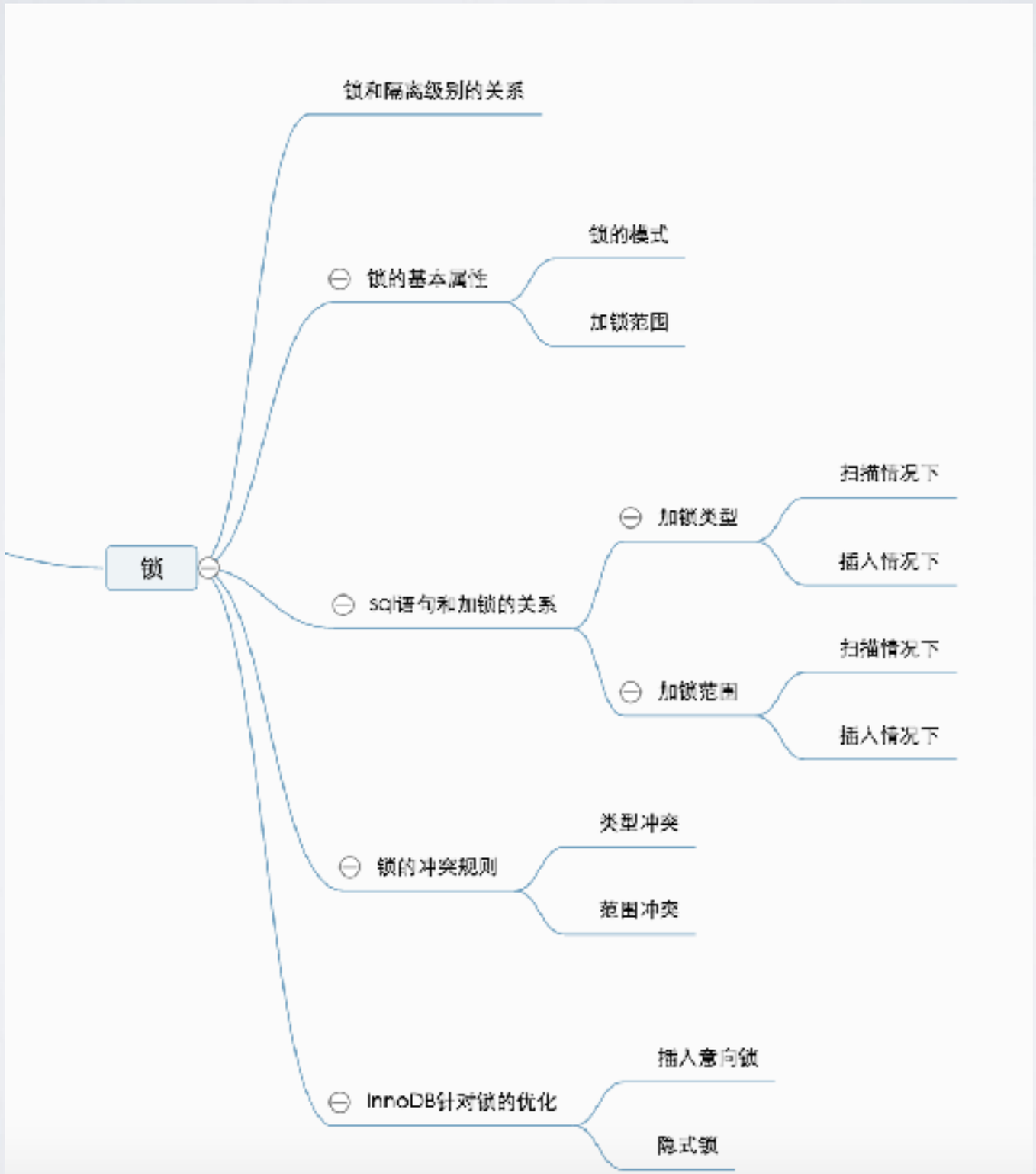
- 从库的执行顺序是不会发生的。

锁介绍

锁介绍

- 锁和隔离级别的关系
- 行锁的基本属性
- sql语句和加锁的关系
- 行锁的冲突规则
- InnoDB针对锁的优化

主要内容脑图（2）


锁
锁和隔离级别的关系
锁的基本属性
锁的模式
加锁范围
sql语句和加锁的关系
加锁类型
扫描情况下
插入情况下
加锁范围
扫描情况下
插入情况下
锁的冲突规则
类型冲突
范围冲突
InnoDB针对锁的优化
插入意向锁
隐式锁

锁介绍

- 锁和隔离级别的关系
- 行锁的基本属性
- sql语句和加锁的关系
- 行锁的冲突规则
- InnoDB针对锁的优化

锁和隔离级别的关系

- 隔离级别的实现主要通过MVCC和锁
  - 可重复读：事务一致性读(trx consistent read)和记录锁(rec lock)
  - 防止幻读：**范围锁**

锁介绍

- 锁和隔离级别的关系
- 行锁的基本属性
- sql语句和加锁的关系
- 行锁的冲突规则
- InnoDB针对锁的优化

行锁的基本属性（1）

- 锁的模式
  - LOCK_S（共享锁）
  - LOCK_X（排它锁）
- 加锁范围
  - RECORD LOCK(记录锁)
  - RANGE LOCK(区间锁)，也是加在记录上，由锁类型区分
    - GAP LOCK
    - NEXT LOCK，**包含RECORD LOCK**
    - INSERT INTENTION LOCK(插入意向锁)

行锁的基本属性（2）

- gap lock和next-key lock基本概念：
  - gap lock为左右开区间
  - next-key lock为record lock（在索引记录上）和gap lock（索引记录之前）的组合（左开右闭）
- 举例说明：
  - 一个索引的行有1，2，3，4
  - 可能gap lock有(1,2)，(2,3)等
  - 可能的next-key lock包括(-无穷,1]，(1,2]，(2,3]，(3,4]，(4,+无穷)等

锁介绍

- 锁和隔离级别的关系
- 行锁的基本属性
- sql语句和加锁的关系
- 行锁的冲突规则
- InnoDB针对锁的优化

SQL语句和加锁的关系

- 正确的理解读操作
- 扫描和插入时的加锁类型
- 扫描和插入时的加锁范围

SQL语句和加锁的关系

- 正确的理解读操作
- 扫描和插入时的加锁类型
- 扫描和插入时的加锁范围

正确理解扫描(读)操作

- **读操作分snapshot read和current read**
- **snapshot read（快照读）**
  - 最普通的select操作（不包括for update等）
  - 举例： select from table
- **current read(也叫lock read)**
  - 删除、更新等必然涉及读，预先读进行检查。
  - 举例：
    - select from table lock in share mode;
    - select from table for update;
    - delete/update table;
    - delete/update from in(select from table)等。

SQL语句和加锁的关系

- 正确的理解读操作
- 扫描和插入时的加锁类型
- 扫描和插入时的加锁范围

扫描加锁类型

	Read Committed	Repeatable Read (unsafe_for_binlog)	Repeatable Read (safe_for_binlog)	Serializable
SELECT ... FROM	consistent read (stmt read view)	consistent read (trx read view)	consistent read (trx read view)	autocommit=ture: consistent read autocommit=false: range lock(LOCK_S)
SELECT ... FROM ... LOCK IN SHARE MODE	rec lock(LOCK_S)	rec lock(LOCK_S)	range lock(LOCK_S)	range lock (LOCK_S)
SELECT ... FROM ... FOR UPDATE	rec lock(LOCK_X)	rec lock(LOCK_X)	range lock(LOCK_X)	range lock(LOCK_X)
UPDATE ... WHERE … **DELETE** ... WHERE ...	rec lock(LOCK_X)	rec lock(LOCK_X)	range lock (LOCK_X)	range lock(LOCK_X)
DELETE …WHERE …(**SELECT**)	rec lock(LOCK_S)	rec lock(LOCK_S)	range lock(LOCK_S)	range lock(LOCK_S)
UPDATE … WHERE …(**SELECT**) INSERT INTO **SELECT** CREATE TABLE AS **SELECT** REPLACE INTO **SELECT**	consistent read (stmt read view)	consistent read (trx read view)	range lock(LOCK_S)	range lock(LOCK_S)

插入加锁类型

	Read Committed	Repeatable Read (unsafe_for_binlog)	Repeatable Read (safe_for_binlog)	Serializable
INSERT/UPDATE	duplicate rec check(LOCK_S) range lock check(LOCK_X)			
INSERT ... ON DUPLICATE KEY UPDATE REPLACE	duplicate rec check(LOCK_X) range lock check(LOCK_X)			

加锁的类型和范围与隔离级别无关

SQL语句和加锁的关系

- 正确的理解读操作
- 扫描和插入时的加锁类型
- 扫描和插入时的加锁范围

扫描加锁范围影响因素

- 影响因素
  - 索引类型：Clustered Index，Unique Index，普通Index
  - 扫描类型：Unique扫描和非Unique扫描
  - 范围条件：L，LE，EQ，GE，G
  - 扫描顺序：前向、后向

扫描加锁范围

- Unique扫描

(unique index || primary key)

&& search field number == unique key number

&& (clustered index || search field not contain null)

Example： unique index(c1, c2)：

c1 = 2 and c2 =2 ===>unique 扫描

c1=2 ===>不是unique扫描

c1=2 and c2 is null===>不是unique扫描

扫描加锁范围规则(1)

- 默认情形：所有满足条件的rec，都加next key lock，**第一个不满足条件的rec，也加next key lock**
- 举例
  - 对于记录1，2，3，5，7
  - 扫描数据2，3
  - 则加的锁为(1,2]，(2,3]，(3,5]

扫描加锁范围规则(2)

- 下述规则，可以理解为在默认条件下的优化，提高了next-key lock下的并发性能
- 等值条件扫描
  - **规则1**：unique扫描，且满足条件的rec不是deleted，则该满足条件的rec加not gap锁
  - **规则2**：非unique扫描的第一个不满足条件行加gap lock
- **规则3**：带条件的后向扫描，则第一个满足条件的下一个rec加gap lock
- **规则4**：primary key的前向GE条件且第一个满足条件的rec等于search_tuple，则该rec加not gap lock
- **规则5**：Supremum record永远加gap lock，Infimum record永远不加锁

扫描加锁范围举例

扫描条件	扫描方向（特殊规则）	非Unique扫描 row key:{1,2,2,3}	Unique扫描	
			不存在const值 row key:{1,3}	存在const值 row key:{1,2,3}
key = 2	move_up(1,2)	ori(2),ori(2), gap(3)	gap(3)	rec(2)(not deleted) **ori(2)(deleted)**
	move_down(1,2,3)	**ori(1),**ori(2),ori(2), gap(3)		
key > 2	move_up()	ori(3), gap(+Infinite)		
	move_down(3)	**ori(2)**,ori(3), gap(+Infinite)	**ori(1)**,ori(3), gap(+Infinite)	**ori(2)**,ori(3), gap(+Infinite)
key >= 2	move_up(4)	ori(2),ori(2),ori(3), gap(+Infinite)	ori(3), gap(+Infinite)	主键:rec(2),ori(3), gap(+Infinite) 非主键:ori(2),ori(3), gap(+Infinite)
	move_down(3)	**ori(1),** ori(2), ori(2), ori(3), gap(+Infinite)	**ori(1),** ori(3), gap(+Infinite)	**ori(1),** ori(2), ori(3), gap(+Infinite)
key < 2	move_up()	ori(1), **ori(2)**	ori(1), **ori(3)**	ori(1), **ori(2)**
	move_down(3)	ori(1), gap(2)	ori(1), gap(3)	ori(1), gap(2)
key <= 2	move_up()	ori(1), ori(2),**ori(3)**	ori(1),**ori(3)**	ori(1), ori(2),**ori(3)**
	move_down(3)	ori(1), ori(2),gap(3)	ori(1),gap(3)	ori(1), ori(2),gap(3)

INSERT加锁范围

	(Unique Index) 重复值检查		加锁
	主键索引	二级索引	
INSERT(Insert)	duplicate rec： not gap lock not allow_duplicate:LOCK_S allow_duplicate:LOCK_X	duplicate rec： next key lock next rec： next key lock not allow_duplicate:LOCK_S allow_duplicate:LOCK_X	next key check and waiting: LOCK_X | LOCK_GAP | LOCK_INSERT_INTENTION
INSERT(By Update)			deleted unique rec： LOCK_X | LOCK_REC_NOT_GAP

锁介绍

- 锁和隔离级别的关系
- 行锁的基本属性
- sql语句和加锁的关系
- 行锁的冲突规则
- InnoDB针对锁的优化

锁模式冲突规则

- 锁模式冲突

```
 /**          S          X
/* S  */   { FALSE,  TRUE},
 /* X  */ { TRUE,   TRUE},
```

锁范围冲突规则

持有锁 / 欲加锁	LOCK_REC_NOT_GAP （REC锁）	LOCK_GAP （GAP 锁）	LOCK_ORDINARY （Next key锁）	LOCK_INSERT_INTENTION （插入意向锁）
LOCK_REC_NOT_GAP	TRUE	FALSE	TRUE	FALSE
LOCK_GAP	FALSE	FALSE	FALSE	FALSE
LOCK_ORDINARY	TRUE	FALSE	TRUE	FALSE
LOCK_INSERT_INTENTION	FALSE	TRUE	TRUE	FALSE

需要注意的是插入意向锁和其他锁均不冲突

举例

- 支付交易数据插入失败

- 退款连接数打满

锁介绍

- 锁和隔离级别的关系
- 行锁的基本属性
- sql语句和加锁的关系
- 行锁的冲突规则
- InnoDB针对锁的优化

- 插入时如果检查INSERT INTENTION LOCK不冲突时不加锁

- 隐式锁优化：插入、更新、删除行时，如果检查不冲突时也不加锁，必要时在下次扫描或者更新加锁时根据最后更新事务号将隐式锁转化为显式锁

隔离级别选择（I）

- 选择RC成本
  - 业务是否允许存在不可重复读和幻读?
    - binlog_format=row
- RC收益
  - 没有范围锁，并发能力强
  - RC 支持半一致性读，RR不支持

隔离级别选择（2）

- 一致性区别：
  - RC隔离级别时，事务中的每一条select语句会读取到他自己执行时已经提交了的记录。
  - 而RR隔离级别时，事务中所有select语句的一致性读的ReadView是第一条select语句的运行时间的快照。
- 业内
  - 大部分公司还是默认的RR
  - 淘宝、网易等采用RC