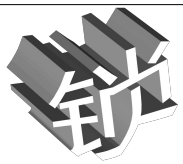
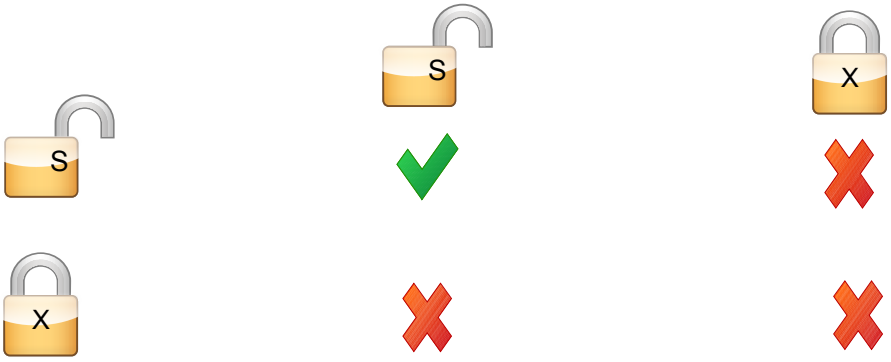


	LOCK 锁	LATCH 闩
	管理对共享资源的并发访问	
对象	事务	线程
保护	数据库内容	内存数据结构
持续时间	整个事务过程	临界资源
模式	行锁、表锁、意向锁	读写锁、互斥量
死锁	通过 wait-for graph 、 time out 等机制进行死锁检测与处理	无死锁检测与处理机制。仅通过应用程序加锁的顺序 lock leveling 保证无死锁
存在于	Lock Manager 的哈希表	每个数据结构的对象中



Latch 轻量级闩锁，锁定时间必须非常短。若持续时间长则性能会差。目的是为了保证并发线程操作临界资源的正确性，并且没有死锁检测机制。
Lock 我们一般说的锁，使用对象为事务，锁定对象为数据库中的对象，例如表、页、行。一般 lock 的对象仅在 commit 或 rollback 后进行释放。有死锁机制。

共享锁 S ：允许事务读
排他锁 X ：允许事务写（ update 或 delete ）

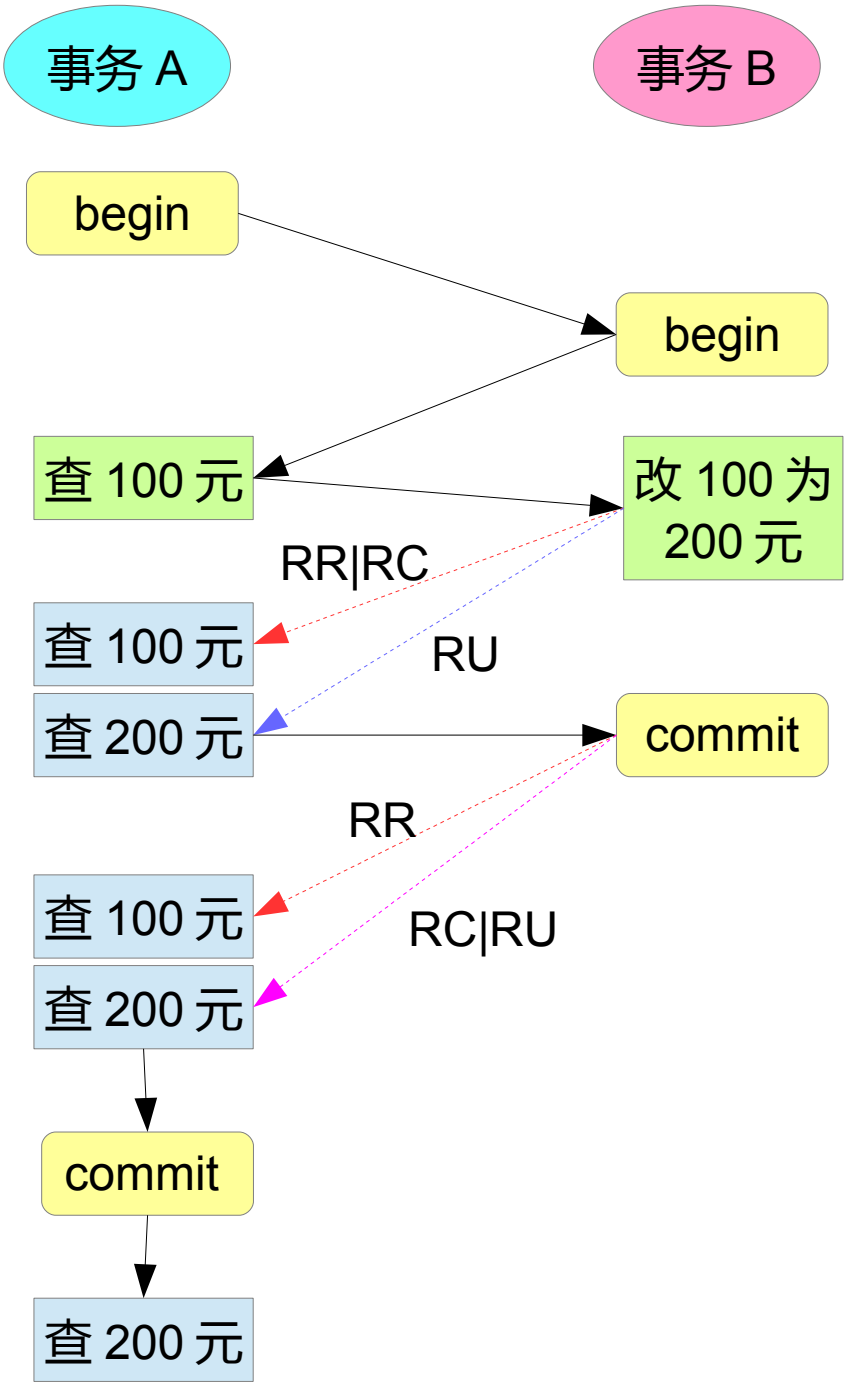
查看 innodb 存储引擎中的 latch ：

```
> show  engine innodb mutex;
```

查看 innodb 存储引擎中的 lock:

```
> show  engine innodb status;
> select * from information_schema.innodb_trx;
> select * from information_schema.innodb_locks;
> select * from information_schema.innodb_lock_warits;
```

ISOLATION 隔离级别	READ UNCOMMITTED	READ COMMITTED	REPEATABLE READ	SERIALIZABLE



InnoDB 默认是可重复读的（ REPEATABLE READ ）， MVCC 多版本并发控制，实现一致性地非锁定读操作。

1. 命令行用— transaction-isolation 选项
2. 配置文件，为所有连接设置默认隔离级别。

```
[mysqld]
transaction-isolation = {READ-UNCOMMITTED | READ-COMMITTED
                         | REPEATABLE-READ | SERIALIZABLE}
```

3. 用户可以用 SET TRANSACTION 语句改变单个会话或者所有新连接的隔离级别。

```
SET [SESSION | GLOBAL] TRANSACTION ISOLATION LEVEL
                       {READ UNCOMMITTED | READ COMMITTED
                        | REPEATABLE READ | SERIALIZABLE}
```

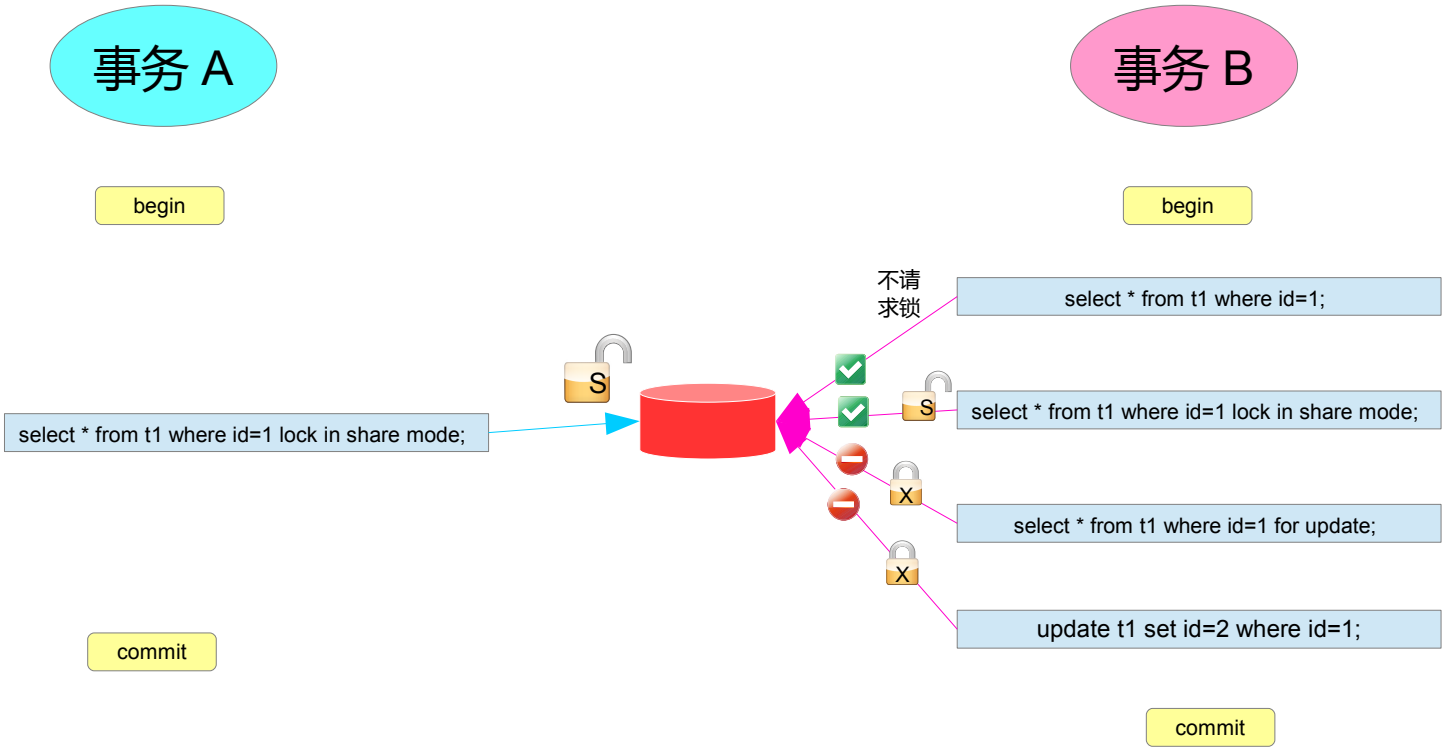
4. 查询全局和会话事务隔离级别：

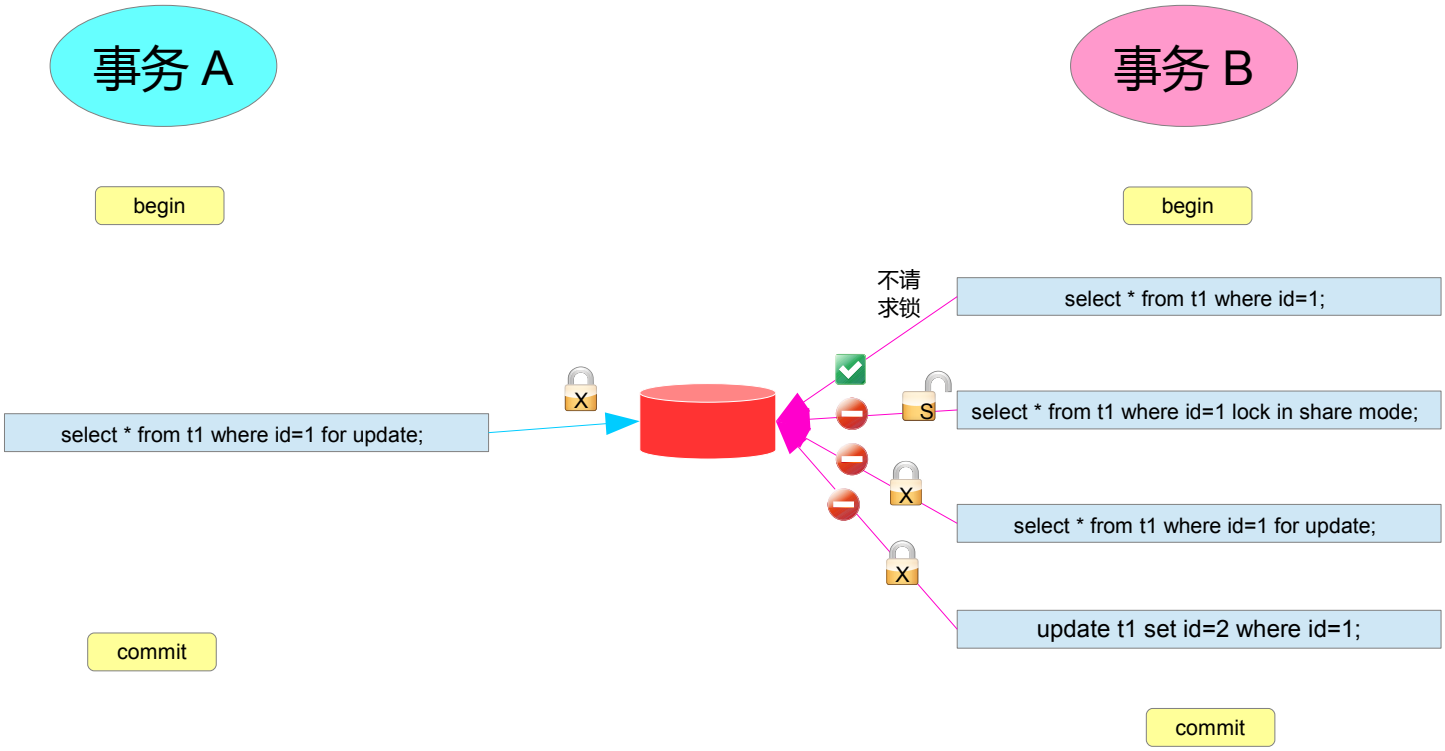
```
SELECT @@global.tx_isolation;
SELECT @@tx_isolation;
```

锁定读
非锁定读

一致性锁定读

一致性非锁定读





InnoDB 默认是可重复读的（ REPEATABLE READ ）， MVCC 多版本并发控制，实现一致性地非锁定读操作。

InnoDB 存储引擎的 select 操作使用一致性非锁定读；也就是说， select 操作不会去请求共享锁 S ；
如何显示地使用一致性锁定读呢？

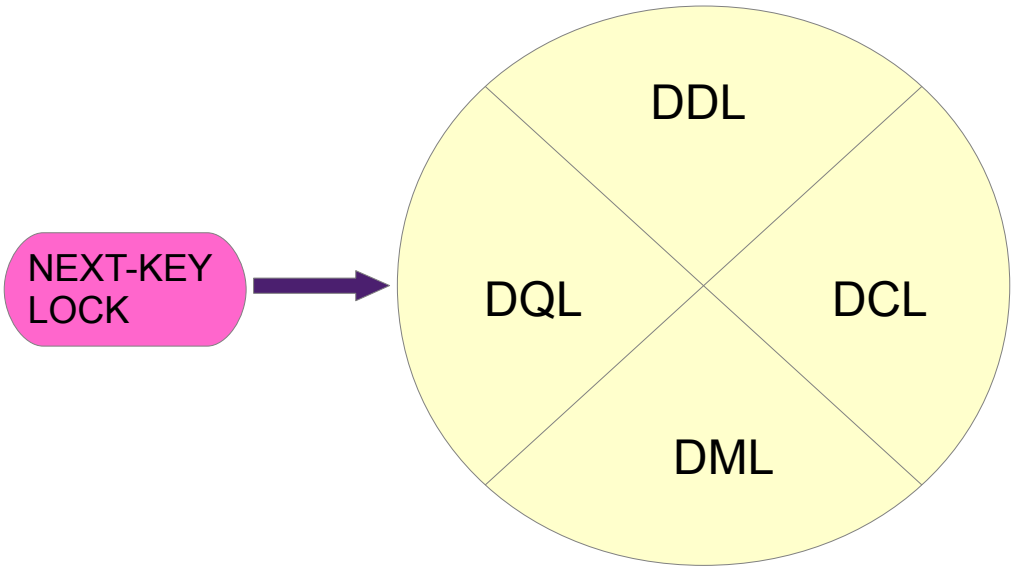
第一种方法，显式地加共享锁 S ： select * from t1 where id=1 lock on share mode;
第二种方法，显式地加排他锁 X ： select * from t1 where id=1 for update;

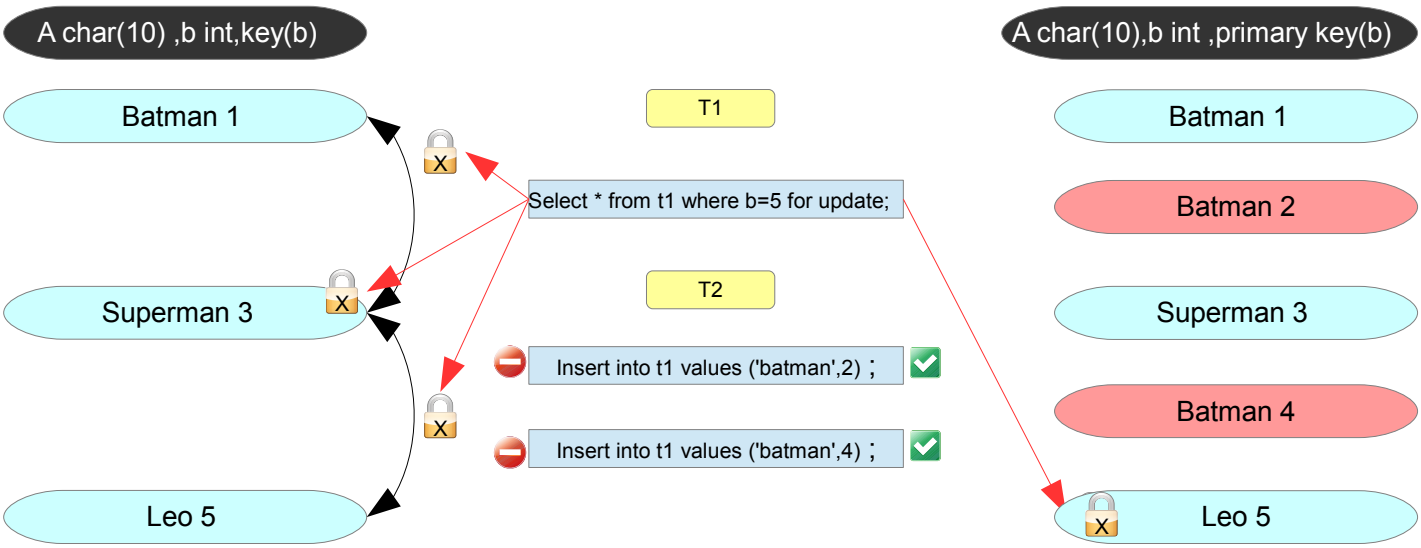
LOCK ALGORITHM
锁的算法

READ LOCK
单个行锁

GAP LOCK
间隙锁

NEXT-KEY LOCK
GAP+RECORD





InnoDB 存储引擎的锁的算法有三种：

Record lock ：单个行记录上的锁

Gap lock ：间隙锁，锁定一个范围，不包括记录本身

Next-key lock ： record+gap 锁定一个范围，包含记录本身

Lock 的精度（ type ）分为 行锁、表锁、意向锁

Lock 的模式（ mode ）分为

锁的类型 ——【读锁和写锁】或者【共享锁和排他锁】即 【 X or S 】

锁的范围 ——【 record lock 、 gap lock 、 Next-key lock 】

1. innodb 对于行的查询使用 next-key lock
2. Next-locking keying 为了解决 Phantom Problem 幻读问题
3. 当查询的索引含有唯一属性时，将 next-key lock 降级为 record key
4. Gap 锁设计的目的是为了阻止多个事务将记录插入到同一范围内，而这会导致幻读问题的产生
5. 有两种方式显式关闭 gap 锁：（除了外键约束和唯一性检查外，其余情况仅使用 record lock ）
   A. 将事务隔离级别设置为 RC
   B. 将参数 innodb_locks_unsafe_for_binlog 设置为 1

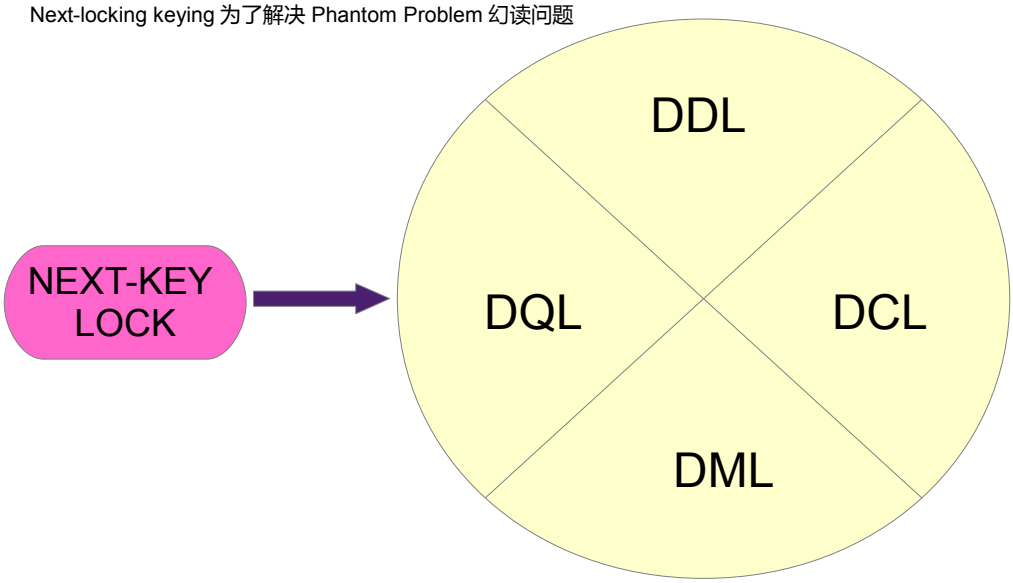
LOCK ALGORITHM
锁的算法

READ LOCK
单个行锁

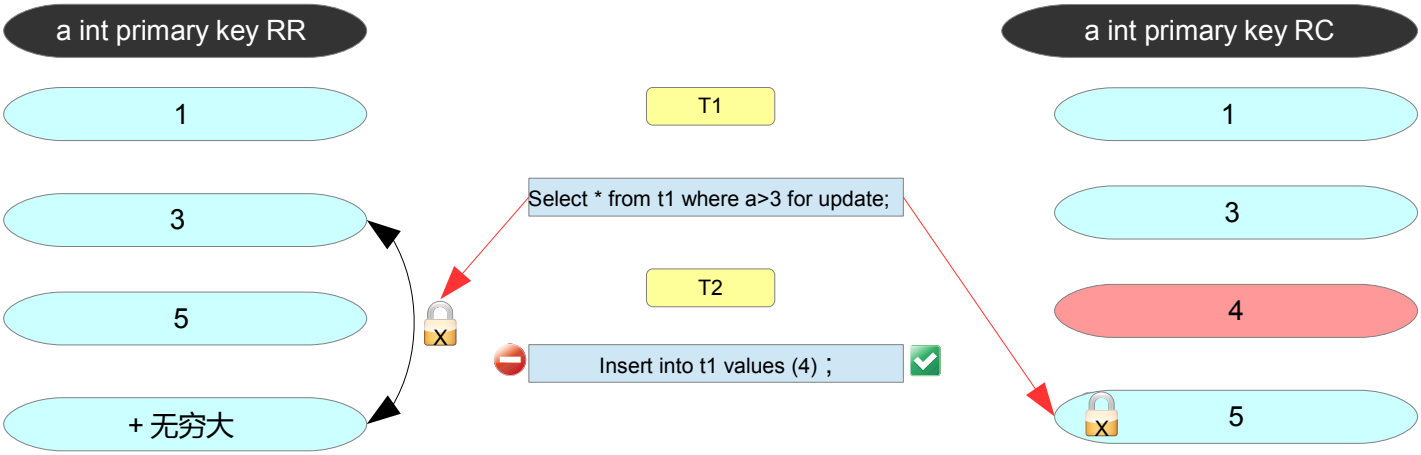
GAP LOCK
间隙锁

NEXT-KEY LOCK
GAP+RECORD

Next-locking keying 为了解决 Phantom Problem 幻读问题



Phantom Problem 幻读问题



InnoDB 存储引擎的锁的算法有三种：

Record lock ：单个行记录上的锁
Gap lock ：间隙锁，锁定一个范围，不包括记录本身
Next-key lock ： record+gap 锁定一个范围，包含记录本身

Lock 的精度（ type ）分为 行锁、表锁、意向锁
Lock 的模式（ mode ）分为

锁的类型 ——【读锁和写锁】或者【共享锁和排他锁】即 【 X or S 】
锁的范围 ——【 record lock 、 gap lock 、 Next-key lock 】

1. innodb 对于行的查询使用 next-key lock
2. Next-locking keying 为了解决 Phantom Problem 幻读问题
3. 当查询的索引含有唯一属性时，将 next-key lock 降级为 record key
4. Gap 锁设计的目的是为了阻止多个事务将记录插入到同一范围内，而这会导致幻读问题的产生
5. 有两种方式显式关闭 gap 锁：（除了外键约束和唯一性检查外，其余情况仅使用 record lock ）
   A. 将事务隔离级别设置为 RC
   B. 将参数 innodb_locks_unsafe_for_binlog 设置为 1