索引：

不生效的情况

回表：

（1）先通过普通索引定位到主键值id=5；

（2）在通过聚集索引定位到行记录；

这就是所谓的回表查询，先定位主键值，再定位行记录，它的性能较扫一遍索引树更低

二级索引的叶子节点中存储的不是“行指针”，而是主键值，并以此作为指向行的“指针”。这样的策略减少了当出现行移动或者数据页分裂时二级索引的维护工作。使用主键当做指针会让二级索引占更多空间，但好处是InnoDB在移动行时无需更新二级索引中的这个指针。

ACID

A:原子性：回滚日志，当事务对数据库修改时，INNODB会生成对应的undo log

C:一致性: 事务开始前，结束后 数据库完整性没有被破坏(undo log+redo log)

I:隔离性：一个事务写对另一个事务写 锁机制保证隔离性，一个事务读对另一个事务读 MVCC保证隔离性

D:持久性：buffer poll+redo log (预写式日志)

Undo log

为了满足事务的原子性，在操作任何数据之前，首先将数据备份到一个地方（这个存储数据备份的地方称为Undo Log），然后进行数据的修改。如果出现了错误或者用户执行了 ROLLBACK语句，系统可以利用Undo Log中的备份将数据恢复到事务开始之前的状态。  
undo log实现多版本并发控制（MVCC）来辅助保证事务的隔离性。

事务提交后并不能马上删除undo log，这是因为可能还有其他事务需要通过undo log 来得到行记录之前的版本。故事务提交时将undo log 放入一个链表中，是否可以删除undo log 根据操作不同分以下2种情况：

Insert undo log： insert操作的记录，只对事务本身可见，对其他事务不可见(这是事务隔离性的要求),故该undo log可以在事务提交后直接删除。不需要进行 purge操作。

update undo log：记录的是对 delete和 update操作产生的 undo log。该undo log可能需要提供MVCC机制，因此不能在事务提交时就进行删除。提交时放入undo log链表,等待 purge线程进行最后的删除。

Redo log

重做日志 redo log 分为两部分：一部分是内存中的重做日志缓冲（redo log buffer），是易丢失的；二部分是重做日志文件（redo log file），是持久的。InnoDB通过Force Log at Commit机制来实现持久性，当commit时，必须先将事务的所有日志写到重做日志文件进行持久化，待commit操作完成才算完成。

事务隔离级别

锁

MVCC全称是： Multiversion concurrency control，多版本并发控制，提供并发访问数据库时，对事务内读取的到的内存做处理，用来避免写操作堵塞读操作的并发问题。

分库分表

索引的本质