第三版：MySQL 70 道

MySQL 索引使用有哪些注意事项呢?

可以从三个维度回答这个问题：索引哪些情况会失效，索引不适合哪些场景，索引规则

索引哪些情况会失效

1、 查询条件包含 or，可能导致索引失效

2、 如何字段类型是字符串，where 时一定用引号括起来，否则索引失效

3、 like 通配符可能导致索引失效。

4、 联合索引，查询时的条件列不是联合索引中的第一个列，索引失效。

5、 在索引列上使用 MySQL 的内置函数，索引失效。

6、 对索引列运算（如，+、-、*、/），索引失效。

7、 索引字段上使用（！= 或者 < >，not in）时，可能会导致索引失效。

8、 索引字段上使用 is null， is not null，可能导致索引失效。

9、 左连接查询或者右连接查询查询关联的字段编码格式不一样，可能导致索引失效。

10、 MySQL 估计使用全表扫描要比使用索引快,则不使用索引。

索引不适合哪些场景

1、 数据量少的不适合加索引

2、 更新比较频繁的也不适合加索引

3、 区分度低的字段不适合加索引（如性别）

索引的一些潜规则

1、 覆盖索引

2、回表

3、 索引数据结构（B+树）

4、 最左前缀原则

5、 索引下推

MySQL 遇到过死锁问题吗，你是如何解决的?

我排查死锁的一般步骤是酱紫的:

1、 查看死锁日志 show engine innodb status;

2、 找出死锁 Sql

3、 分析 sql 加锁情况

4、 模拟死锁案发

5、 分析死锁日志

6、 分析死锁结果

日常工作中你是怎么优化 SQL 的?

可以从这几个维度回答这个问题:

1、 加索引

2、 避免返回不必要的数据

3、 适当分批量进行

4、 优化 sql 结构

5、 分库分表

6、 读写分离

说说分库与分表的设计

分库分表方案，分库分表中间件，分库分表可能遇到的问题

分库分表方案:

1、 水平分库：以字段为依据，按照一定策略（hash、range 等），将一个库中的数据拆分到多个库中。

2、 水平分表：以字段为依据，按照一定策略（hash、range 等），将一个表中的数据拆分到多个表中。

3、 垂直分库：以表为依据，按照业务归属不同，将不同的表拆分到不同的库中。

4、 垂直分表：以字段为依据，按照字段的活跃性，将表中字段拆到不同的表（主表和扩展表）中。

常用的分库分表中间件：

1、 sharding-jdbc（当当）

2、 Mycat

3、 TDDL（淘宝）

4、 Oceanus(58 同城数据库中间件)

5、 vitess（谷歌开发的数据库中间件）

6、 Atlas(Qihoo 360)

分库分表可能遇到的问题

1、 事务问题：需要用分布式事务啦

2、 跨节点 Join 的问题：解决这一问题可以分两次查询实现

3、 跨节点的 count,order by,group by 以及聚合函数问题：分别在各个节点上得到结果后在应用程序端进行合并。

4、 数据迁移，容量规划，扩容等问题

5、 ID 问题：数据库被切分后，不能再依赖数据库自身的主键生成机制啦，最简单可以考虑 UUID

6、 跨分片的排序分页问题（后台加大 pagesize 处理？）

InnoDB 与 MyISAM 的区别

1、 InnoDB 支持事务，MyISAM 不支持事务

2、 InnoDB 支持外键，MyISAM 不支持外键

3、 InnoDB 支持 MVCC(多版本并发控制)，MyISAM 不支持

4、 select count(*) from table 时，MyISAM 更快，因为它有一个变量保存了整个表的总行数，可以直接读取，InnoDB 就需要全表扫描。

5、 Innodb 不支持全文索引，而 MyISAM 支持全文索引（5.7 以后的 InnoDB 也支持全文索引）

6、 InnoDB 支持表、行级锁，而 MyISAM 支持表级锁。

7、 InnoDB 表必须有主键，而 MyISAM 可以没有主键

8、 Innodb 表需要更多的内存和存储，而 MyISAM 可被压缩，存储空间较小，。

9、 Innodb 按主键大小有序插入，MyISAM 记录插入顺序是，按记录插入顺序保存。

10、 InnoDB 存储引擎提供了具有提交、回滚、崩溃恢复能力的事务安全，与 MyISAM 比 InnoDB 写的效率差一些，并且会占用更多的磁盘空间以保留数据和索引

数据库索引的原理，为什么要用 B+树，为什么不用二叉树?

可以从几个维度去看这个问题，查询是否够快，效率是否稳定，存储数据多少，以及查找磁盘次数，为什么不是二叉树，为什么不是平衡二叉树，为什么不是 B 树，而偏偏是 B+树呢?

为什么不是一般二叉树?

如果二叉树特殊化为一个链表，相当于全表扫描。平衡二叉树相比于二叉查找树来说，查找效率更稳定，总体的查找速度也更快。

为什么不是平衡二叉树呢?

我们知道，在内存比在磁盘的数据，查询效率快得多。如果树这种数据结构作为索引，那我们每查找一次数据就需要从磁盘中读取一个节点，也就是我们说的一个磁盘块，但是平衡二叉树可是每个节点只存储一个键值和数据的，如果是 B 树，可以存储更多的节点数据，树的高度也会降低，因此读取磁盘的次数就降下来啦，查询效率就快啦。

那为什么不是 B 树而是 B+树呢?

1）B+树非叶子节点上是不存储数据的，仅存储键值，而 B 树节点中不仅存储键值，也会存储数据。innodb 中页的默认大小是 16KB，如果不存储数据，那么就

会存储更多的键值，相应的树的阶数（节点的子节点树）就会更大，树就会更矮更胖，如此一来我们查找数据进行磁盘的 IO 次数有会再次减少，数据查询的效率也会更快。

2）B+树索引的所有数据均存储在叶子节点，而且数据是按照顺序排列的，链表连着的。那么 B+树使得范围查找，排序查找，分组查找以及去重查找变得异常简单。

聚集索引与非聚集索引的区别

1、 一个表中只能拥有一个聚集索引，而非聚集索引一个表可以存在多个。

2、 聚集索引，索引中键值的逻辑顺序决定了表中相应行的物理顺序；非聚集索引，索引中索引的逻辑顺序与磁盘上行的物理存储顺序不同。

3、 索引是通过二叉树的数据结构来描述的，我们可以这么理解聚簇索引：索引的叶节点就是数据节点。而非聚簇索引的叶节点仍然是索引节点，只不过有一个指针指向对应的数据块。

4、 聚集索引：物理存储按照索引排序；非聚集索引：物理存储不按照索引排序；

limit 1000000 加载很慢的话，你是怎么解决的呢?

方案一：如果 id 是连续的，可以这样，返回上次查询的最大记录(偏移量)，再往下 limit

```
select id，name from employee where id>1000000 limit 10.
```

方案二：在业务允许的情况下限制页数：

建议跟业务讨论，有没有必要查这么后的分页啦。因为绝大多数用户都不会往后翻太多页。

方案三：order by + 索引（id 为索引）

```
select id，name from employee order by id  limit 1000000，10
```

方案四：利用延迟关联或者子查询优化超多分页场景。（先快速定位需要获取的 id 段，然后再关联）

```
SELECT a.* FROM employee a, (select id from employee where 条件 LIMIT 1000000,10 ) b where a.id=b.id
```

如何选择合适的分布式主键方案呢?

1、 数据库自增长序列或字段。

2、 UUID。

3、 Redis 生成 ID

4、 Twitter 的 snowflake 算法

5、 利用 zookeeper 生成唯一 ID

6、 MongoDB 的 ObjectId

事务的隔离级别有哪些？MySQL 的默认隔离级别是什么?

1、 读未提交（Read Uncommitted）

2、 读已提交（Read Committed）

3、 可重复读（Repeatable Read）

4、 串行化（Serializable）

MySQL 默认的事务隔离级别是可重复读(Repeatable Read)

什么是幻读，脏读，不可重复读呢?

1、 事务 A、B 交替执行，事务 A 被事务 B 干扰到了，因为事务 A 读取到事务 B 未提交的数据,这就是**脏读**

2、 在一个事务范围内，两个相同的查询，读取同一条记录，却返回了不同的数据，这就是**不可重复读。**

3、 事务 A 查询一个范围的结果集，另一个并发事务 B 往这个范围中插入/删除了数据，并静悄悄地提交，然后事务 A 再次查询相同的范围，两次读取得到的结果集不一样了，这就是**幻读。**

在高并发情况下，如何做到安全的修改同一行数据?

要安全的修改同一行数据，就要保证一个线程在修改时其它线程无法更新这行记录。一般有悲观锁和乐观锁两种方案~

使用悲观锁

悲观锁思想就是，当前线程要进来修改数据时，别的线程都得拒之门外~

比如，可以使用 select...for update ~

```
select * from User where name= ‘jay’  for update
```

以上这条 sql 语句会锁定了 User 表中所有符合检索条件（name= ‘jay’ ）的记录。本次事务提交之前，别的线程都无法修改这些记录。

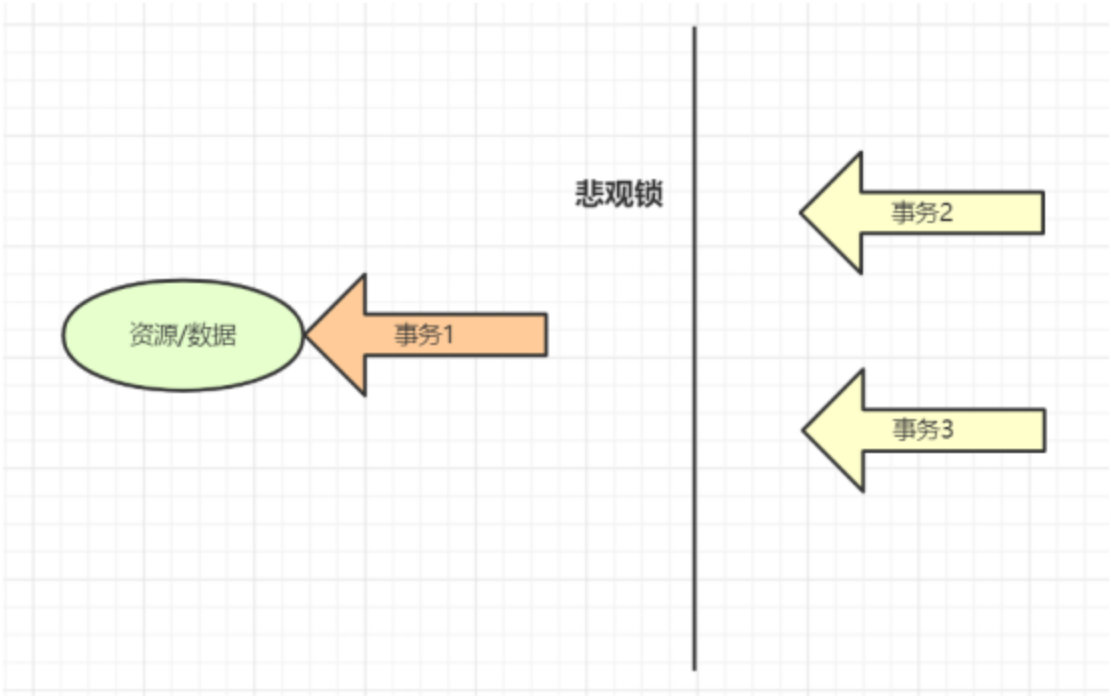
使用乐观锁

乐观锁思想就是，有线程过来，先放过去修改，如果看到别的线程没修改过，就可以修改成功，如果别的线程修改过，就修改失败或者重试。实现方式：乐观锁一般会使用版本号机制或 CAS 算法实现。

数据库的乐观锁和悲观锁。

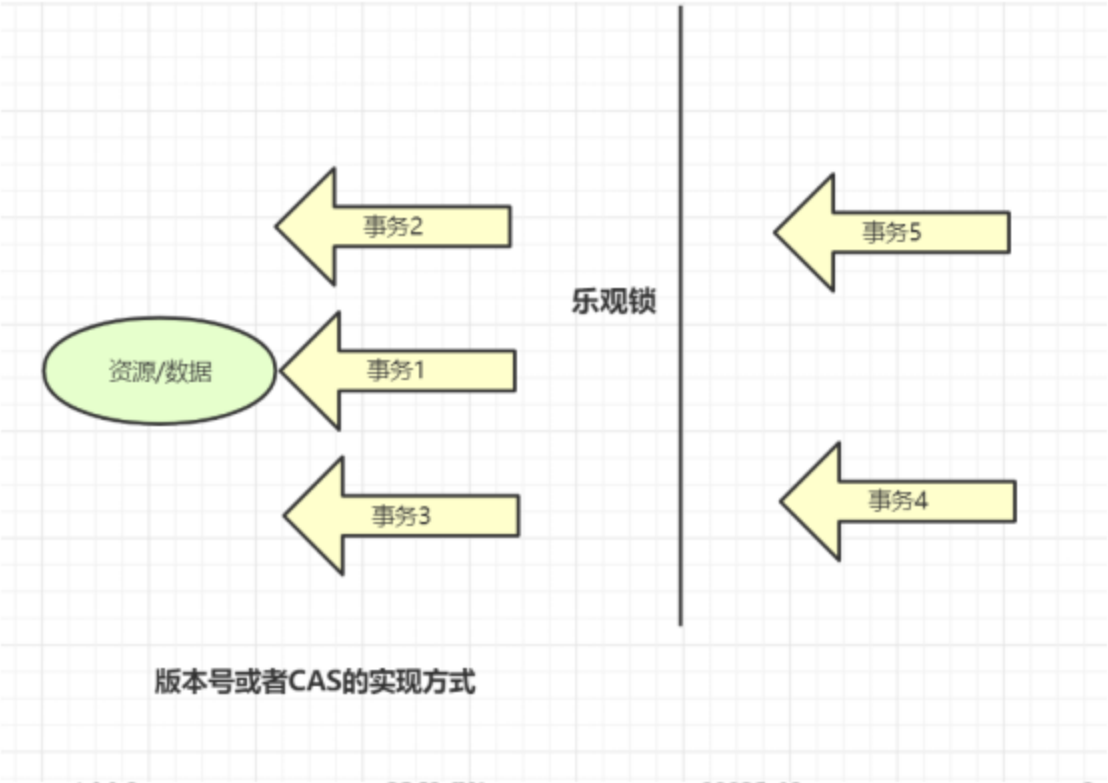
悲观锁：

悲观锁她专一且缺乏安全感了，她的心只属于当前事务，每时每刻都担心着它心爱的数据可能被别的事务修改，所以一个事务拥有（获得）悲观锁后，其他任何事务都不能对数据进行修改啦，只能等待锁被释放才可以执行。



乐观锁：

乐观锁的“乐观情绪”体现在，它认为数据的变动不会太频繁。因此，它允许多个事务同时对数据进行变动。实现方式：乐观锁一般会使用版本号机制或 CAS 算法实现。



SQL 优化的一般步骤是什么，怎么看执行计划（explain），如何理解其中各个字段的含义。

1、 show status 命令了解各种 sql 的执行频率

2、 通过慢查询日志定位那些执行效率较低的 sql 语句

3、 explain 分析低效 sql 的执行计划(这点非常重要,日常开发中用它分析 Sql，会大大降低 Sql 导致的线上事故)

select for update 有什么含义，会锁表还是锁行还是其他。

select for update 含义

select 查询语句是不会加锁的，但是 select for update 除了有查询的作用外，还会加锁呢，而且它是悲观锁哦。至于加了是行锁还是表锁，这就要看是不是用了索引/主键啦。

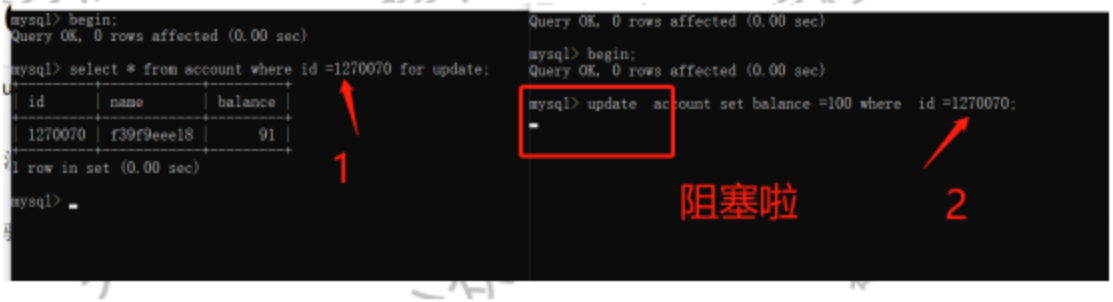
没用索引/主键的话就是表锁，否则就是是行锁。

select for update 加锁验证

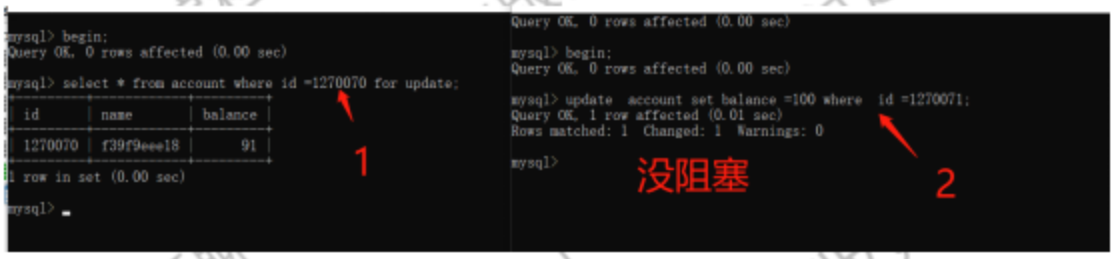
表结构：

```
//id 为主键，name 为唯一索引
CREATE TABLE `account` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) DEFAULT NULL,
  `balance` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `idx_name` (`name`) USING BTREE
) ENGINE=InnoDB AUTO_INCREMENT=1570068 DEFAULT
CHARSET=utf8
```

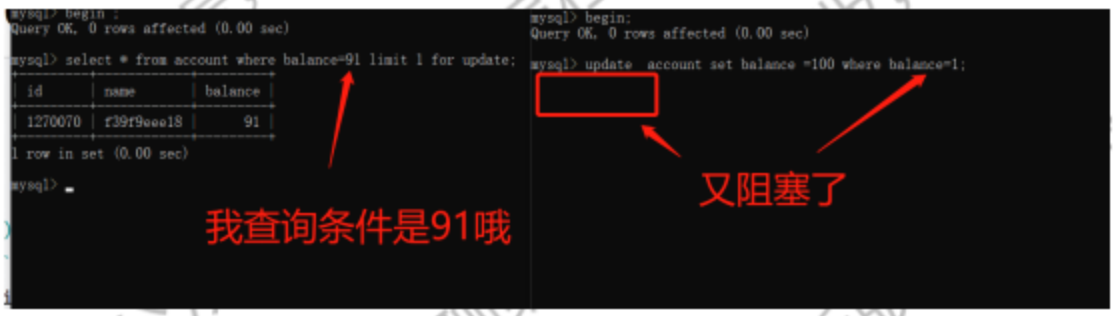
id 为主键，select for update 1270070 这条记录时，再开一个事务对该记录更新，发现更新阻塞啦，其实是加锁了。如下图：



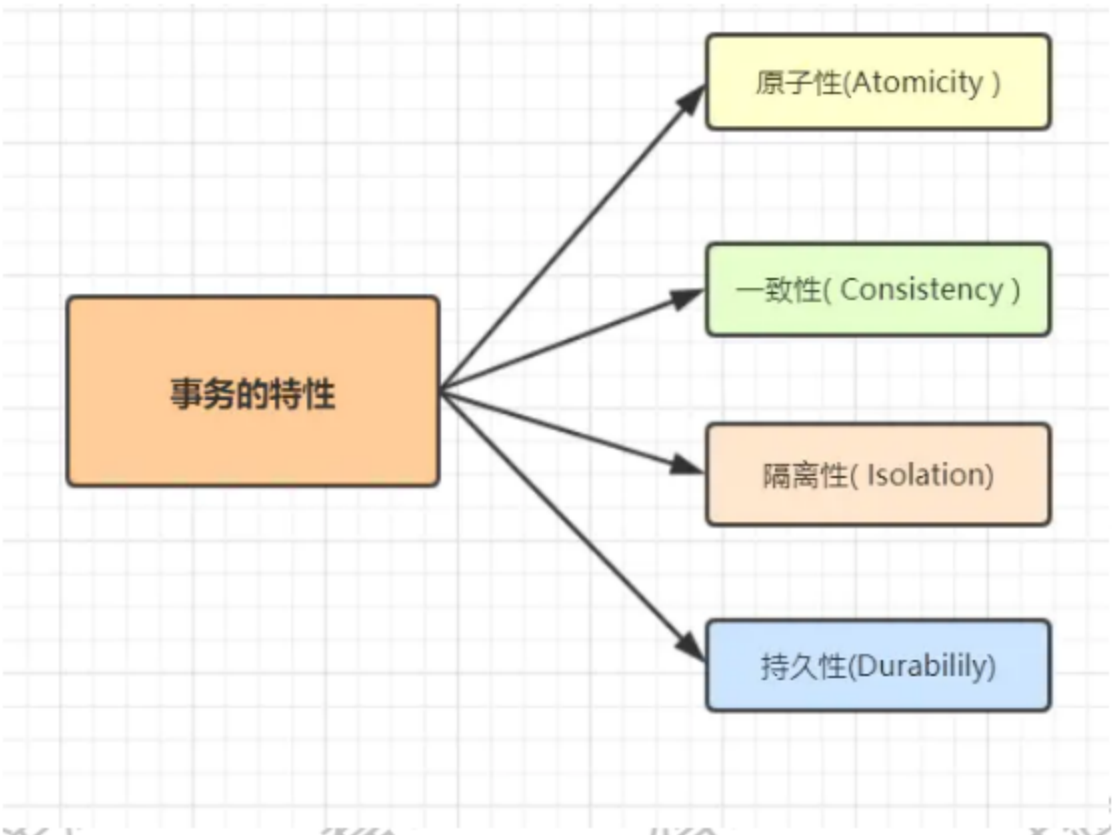
我们再开一个事务对另外一条记录 1270071 更新，发现更新成功，因此，如果查询条件用了索引/主键，**会加行锁~**



我们继续一路向北吧，换普通字段 balance 吧，发现又阻塞了。因此，没用索引/主键的话，**select for update 加的就是表锁**



MySQL 事务得四大特性以及实现原理



1、 原子性： 事务作为一个整体被执行，包含在其中的对数据库的操作要么全部被执行，要么都不执行。

2、 一致性： 指在事务开始之前和事务结束以后，数据不会被破坏，假如 A 账户给 B 账户转 10 块钱，不管成功与否，A 和 B 的总金额是不变的。

3、 隔离性： 多个事务并发访问时，事务之间是相互隔离的，即一个事务不影响其它事务运行效果。简言之，就是事务之间是进水不犯河水的。

4、 持久性： 表示事务完成以后，该事务对数据库所作的操作更改，将持久地保存在数据库之中。

事务 ACID 特性的实现思想

1、 原子性：是使用 undo log 来实现的，如果事务执行过程中出错或者用户执行了 rollback，系统通过 undo log 日志返回事务开始的状态。

2、 持久性：使用 redo log 来实现，只要 redo log 日志持久化了，当系统崩溃，即可通过 redo log 把数据恢复。

3、 隔离性：通过锁以及 MVCC,使事务相互隔离开。

4、 一致性：通过回滚、恢复，以及并发情况下的隔离性，从而实现一致性。

如果某个表有近千万数据，CRUD 比较慢，如何优化。

分库分表

某个表有近千万数据，可以考虑优化表结构，分表（水平分表，垂直分表），当然，你这样回答，需要准备好面试官问你的分库分表相关问题呀，如

1、 分表方案（水平分表，垂直分表，切分规则 hash 等）

2、 分库分表中间件（Mycat，sharding-jdbc 等）

3、 分库分表一些问题（事务问题？跨节点 Join 的问题）

4、 解决方案（分布式事务等）

索引优化

除了分库分表，优化表结构，当然还有所以索引优化等方案~

如何写 sql 能够有效的使用到复合索引。

复合索引，也叫组合索引，用户可以在多个列上建立索引,这种索引叫做复合索引。

当我们创建一个组合索引的时候，如(k1,k2,k3)，相当于创建了（k1）、(k1,k2)和(k1,k2,k3)三个索引，这就是最左匹配原则。

```
select * from table where k1=A AND k2=B AND k3=D
```

有关于复合索引，我们需要关注查询 Sql 条件的顺序，确保最左匹配原则有效，同时可以删除不必要的冗余索引。

MySQL 中 in 和 exists 的区别。

这个，跟一下 demo 来看更刺激吧，啊哈哈

假设表 A 表示某企业的员工表，表 B 表示部门表，查询所有部门的所有员工，很容易有以下 SQL:

```
select * from A where deptId in (select deptId from B);
```

这样写等价于：

1、 先查询部门表 B

2、 select deptId from B

3、 再由部门 deptId，查询 A 的员工

4、 select * from A where A.deptId = B.deptId

可以抽象成这样的一个循环：

```
List<> resultSet ;
 for(int i=0;i<B.length;i++) {
         for(int j=0;j<A.length;j++) {
         if(A[i].id==B[j].id) {
             resultSet.add(A[i]);
             break;
         }
     }
 }
```

显然，除了使用 in，我们也可以用 exists 实现一样的查询功能，如下：

```
select * from A where exists (select 1 from B where A.deptId = B.deptId);
```

因为 exists 查询的理解就是，先执行主查询，获得数据后，再放到子查询中做条件验证，根据验证结果（true 或者 false），来决定主查询的数据结果是否得意保留。

那么，这样写就等价于：

select * from A,先从 A 表做循环
select * from B where A.deptId = B.deptId,再从 B 表做循环.

同理，可以抽象成这样一个循环：

```
    List<> resultSet ;
     for(int i=0;i<A.length;i++) {
```

```
            for(int j=0;j<B.length;j++) {
            if(A[i].deptId==B[j].deptId) {
                resultSet.add(A[i]);
                break;
            }
        }
    }
```

数据库最费劲的就是跟程序链接释放。假设链接了两次，每次做上百万次的数据集查询，查完就走，这样就只做了两次；相反建立了上百万次链接，申请链接释放反复重复，这样系统就受不了了。即 MySQL 优化原则，就是小表驱动大表，小的数据集驱动大的数据集，从而让性能更优。

因此，我们要选择最外层循环小的，也就是，如果 **B 的数据量小于 A，适合使用 in，如果 B 的数据量大于 A，即适合选择 exists**，这就是 in 和 exists 的区别。

数据库自增主键可能遇到什么问题。

1、 使用自增主键对数据库做分库分表，可能出现诸如主键重复等的问题。解决方案的话，简单点的话可以考虑使用 UUID 哈

2、 自增主键会产生表锁，从而引发问题

3、 自增主键可能用完问题。

MVCC 熟悉吗，它的底层原理?

MVCC,多版本并发控制,它是通过读取历史版本的数据，来降低并发事务冲突，从而提高并发性能的一种机制。

MVCC 需要关注这几个知识点：

1、 事务版本号

2、 表的隐藏列

3、 undo log

4、 read view

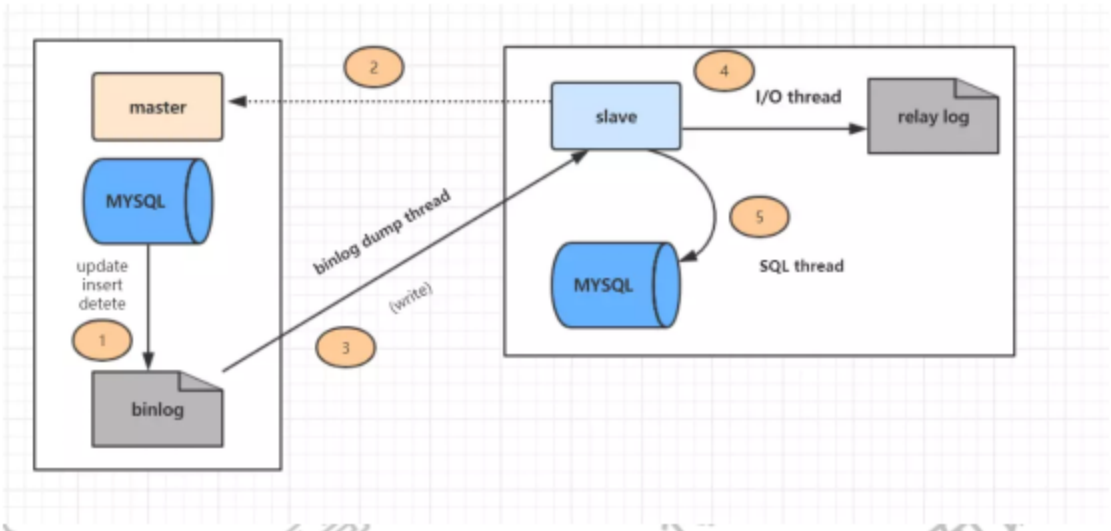
数据库中间件了解过吗，sharding jdbc，mycat?

1、 sharding-jdbc 目前是基于 jdbc 驱动，无需额外的 proxy，因此也无需关注 proxy 本身的高可用。

2、 Mycat 是基于 Proxy，它复写了 MySQL 协议，将 Mycat Server 伪装成一个 MySQL 数据库，而 Sharding-JDBC 是基于 JDBC 接口的扩展，是以 jar 包的形式提供轻量级服务的。

MYSQL 的主从延迟，你怎么解决?

嘻嘻，先复习一下主从复制原理吧，如图：



主从复制分了五个步骤进行：

1、 步骤一：主库的更新事件(update、insert、delete)被写到 binlog

2、 步骤二：从库发起连接，连接到主库。

3、 步骤三：此时主库创建一个 binlog dump thread，把 binlog 的内容发送到从库。

4、 步骤四：从库启动之后，创建一个 I/O 线程，读取主库传过来的 binlog 内容并写入到 relay log

5、 步骤五：还会创建一个 SQL 线程，从 relay log 里面读取内容，从 Exec_Master_Log_Pos 位置开始执行读取到的更新事件，将更新内容写入到 slave 的 db

主从同步延迟的原因

一个服务器开放N个链接给客户端来连接的，这样有会有大并发的更新操作，但是从服务器的里面读取 binlog 的线程仅有一个，当某个 SQL 在从服务器上执行的时间稍长 或者由于某个 SQL 要进行锁表就会导致，主服务器的 SQL 大量积压，未被同步到从服务器里。这就导致了主从不一致， 也就是主从延迟。

主从同步延迟的解决办法

1、 主服务器要负责更新操作，对安全性的要求比从服务器要高，所以有些设置参数可以修改，比如 sync_binlog=1，innodb_flush_log_at_trx_commit = 1 之类的设置等。

2、 选择更好的硬件设备作为 slave。

3、 把一台从服务器当度作为备份使用， 而不提供查询， 那边他的负载下来了，执行 relay log 里面的 SQL 效率自然就高了。

4、 增加从服务器喽，这个目的还是分散读的压力，从而降低服务器负载。

说一下大表查询的优化方案

1、 优化 shema、sql 语句+索引；

2、 可以考虑加缓存，Memcached, Redis，或者 JVM 本地缓存；

3、 主从复制，读写分离；

4、 分库分表；

什么是数据库连接池?为什么需要数据库连接池呢?

连接池基本原理：

数据库连接池原理：在内部对象池中，维护一定数量的数据库连接，并对外暴露数据库连接的获取和返回方法。

应用程序和数据库建立连接的过程：

1、 通过 TCP 协议的三次握手和数据库服务器建立连接

2、 发送数据库用户账号密码，等待数据库验证用户身份

3、 完成身份验证后，系统可以提交 SQL 语句到数据库执行

4、 把连接关闭，TCP 四次挥手告别。

数据库连接池好处：

1、 资源重用（连接复用）

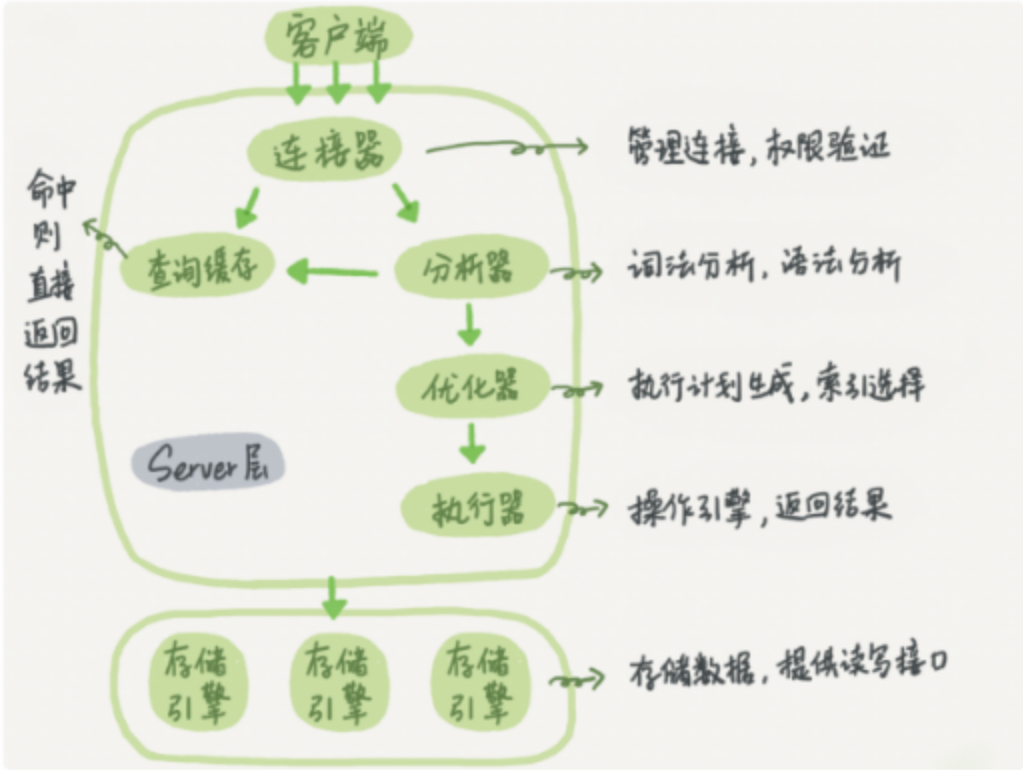
2、 更快的系统响应速度

3、 新的资源分配手段

4、 统一的连接管理，避免数据库连接泄漏

一条 SQL 语句在 MySQL 中如何执行的?

先看一下 MySQL 的逻辑架构图吧~



查询语句：

1、 先检查该语句是否有权限

2、 如果没有权限，直接返回错误信息

3、 如果有权限，在 MySQL8.0 版本以前，会先查询缓存。

4、 如果没有缓存，分析器进行词法分析，提取 sql 语句 select 等的关键元素。然后判断 sql 语句是否有语法错误，比如关键词是否正确等等。

5、 优化器进行确定执行方案

6、 进行权限校验，如果没有权限就直接返回错误信息，如果有权限就会调用数据库引擎接口，返回执行结果。

InnoDB 引擎中的索引策略，了解过吗？

1、 覆盖索引

2、 最左前缀原则

3、 索引下推

索引下推优化是 MySQL 5.6 引入的，可以在索引遍历过程中，对索引中包含的字段先做判断，直接过滤掉不满足条件的记录，减少回表次数。

数据库存储日期格式时，如何考虑时区转换问题？

datetime 类型适合用来记录数据的原始的创建时间，修改记录中其他字段的值，datetime 字段的值不 1、会改变，除非手动修改它。

2、 timestamp 类型适合用来记录数据的最后修改时间，只要修改了记录中其他字段的值，timestamp 字段的值都会被自动更新。

一条 sql 执行过长的时间，你如何优化，从哪些方面入手？

1、 查看是否涉及多表和子查询，优化 Sql 结构，如去除冗余字段，是否可拆表等

2、 优化索引结构，看是否可以适当添加索引

3、 数量大的表，可以考虑进行分离/分表（如交易流水表）

4、 数据库主从分离，读写分离

5、 explain 分析 sql 语句，查看执行计划，优化 sql

6、 查看 MySQL 执行日志，分析是否有其他方面的问题

MYSQL 数据库服务器性能分析的方法命令有哪些?

Show status，一些值得监控的变量值：

1、 Bytes_received 和 Bytes_sent 和服务器之间来往的流量。

2、 Com_*服务器正在执行的命令。

3、 Created_*在查询执行期限间创建的临时表和文件。

4、 Handler_*存储引擎操作。

5、 Select_*不同类型的联接执行计划。

6、 Sort_*几种排序信息。

Show profiles 是 MySql 用来分析当前会话 SQL 语句执行的资源消耗情况

Blob 和 text 有什么区别?

1、 Blob 用于存储二进制数据，而 Text 用于存储大字符串。

2、 Blob 值被视为二进制字符串（字节字符串）,它们没有字符集，并且排序和比较基于列值中的字节的数值。

3、 text 值被视为非二进制字符串（字符字符串）。它们有一个字符集，并根据字符集的排序规则对值进行排序和比较。

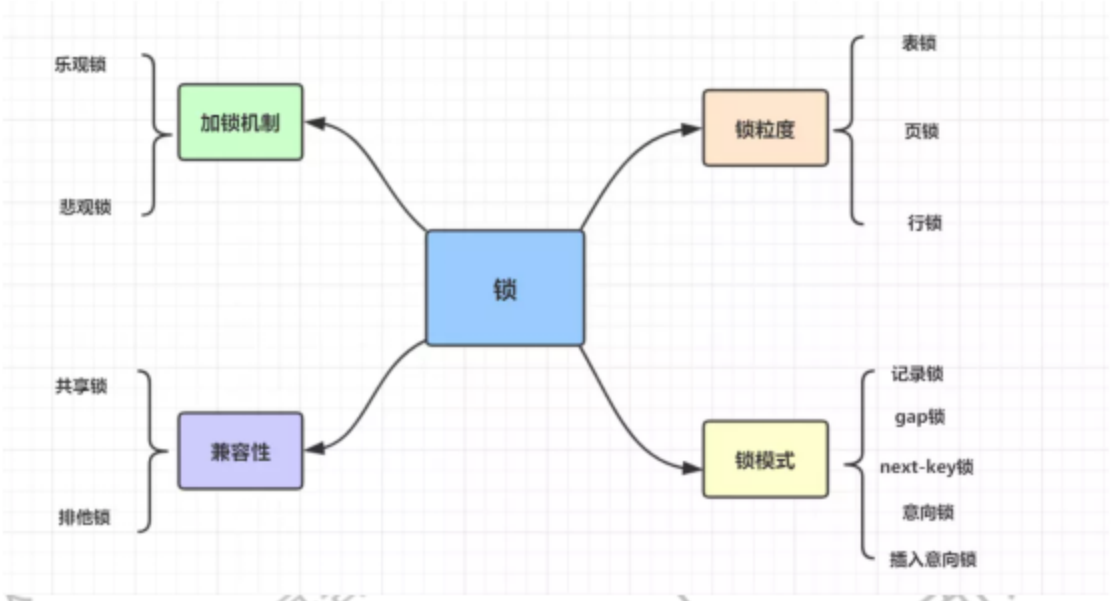
MySQL 里记录货币用什么字段类型比较好?

1、 货币在数据库中 MySQL 常用 Decimal 和 Numric 类型表示，这两种类型被 MySQL 实现为同样的类型。他们被用于保存与金钱有关的数据。

2、 salary DECIMAL(9,2)，9(precision)代表将被用于存储值的总的小数位数，而 2(scale)代表将被用于存储小数点后的位数。存储在 salary 列中的值的范围是从-9999999.99 到 9999999.99。

3、 DECIMAL 和 NUMERIC 值作为字符串存储，而不是作为二进制浮点数，以便保存那些值的小数精度。

MySQL 中有哪几种锁，列举一下?



如果按锁粒度划分，有以下 3 种：

1、 表锁： 开销小，加锁快；锁定力度大，发生锁冲突概率高，并发度最低;不会出现死锁。

2、 行锁： 开销大，加锁慢；会出现死锁；锁定粒度小，发生锁冲突的概率低，并发度高。

3、 页锁： 开销和加锁速度介于表锁和行锁之间；会出现死锁；锁定粒度介于表锁和行锁之间，并发度一般

Hash 索引和 B+树区别是什么？你在设计索引是怎么抉择的?

1、 B+树可以进行范围查询，Hash 索引不能。

2、 B+树支持联合索引的最左侧原则，Hash 索引不支持。

3、 B+树支持 order by 排序，Hash 索引不支持。

4、 Hash 索引在等值查询上比 B+树效率更高。

5、 B+树使用 like 进行模糊查询的时候，like 后面（比如%开头）的话可以起到优化的作用，Hash 索引根本无法进行模糊查询。

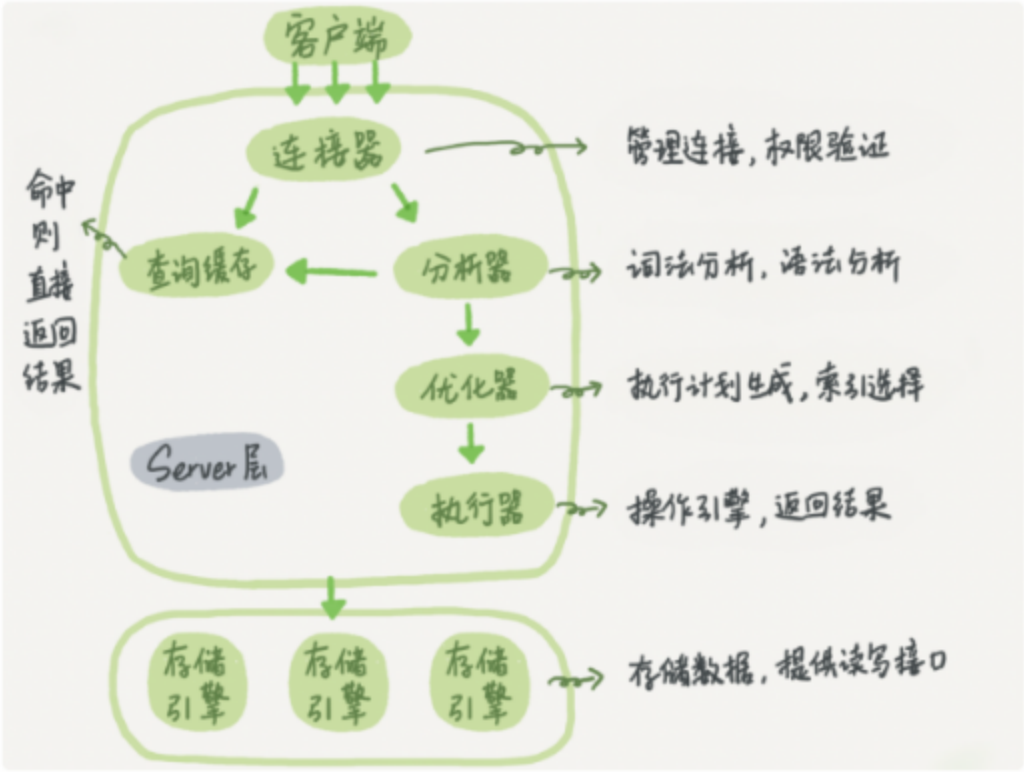
MySQL 的内连接、左连接、右连接有什么区别?

1、 Inner join 内连接，在两张表进行连接查询时，只保留两张表中完全匹配的结果集

2、 left join 在两张表进行连接查询时，会返回左表所有的行，即使在右表中没有匹配的记录。

3、 right join 在两张表进行连接查询时，会返回右表所有的行，即使在左表中没有匹配的记录。

说说 MySQL 的基础架构图



MySQL 逻辑架构图主要分三层：

1、 第一层负责连接处理，授权认证，安全等等

2、 第二层负责编译并优化 SQL

3、 第三层是存储引擎。

什么是内连接、外连接、交叉连接、笛卡尔积呢?

1、 内连接（inner join）：取得两张表中满足存在连接匹配关系的记录。

2、 外连接（outer join）：取得两张表中满足存在连接匹配关系的记录，以及某张表（或两张表）中不满足匹配关系的记录。

3、 交叉连接（cross join）：显示两张表所有记录一一对应，没有匹配关系进行筛选，也被称为：笛卡尔积。

说一下数据库的三大范式

1、 第一范式：数据表中的每一列（每个字段）都不可以再拆分。

2、 第二范式：在第一范式的基础上，分主键列完全依赖于主键，而不能是依赖于主键的一部分。

3、 第三范式：在满足第二范式的基础上，表中的非主键只依赖于主键，而不依赖于其他非主键。

MySQL 有关权限的表有哪几个呢?

MySQL 服务器通过权限表来控制用户对数据库的访问，权限表存放在 MySQL 数据库里，由 MySQL_install_db 脚本初始化。这些权限表分别 user，db，table_priv，columns_priv 和 host。

1、 user 权限表：记录允许连接到服务器的用户帐号信息，里面的权限是全局级的。

2、 db 权限表：记录各个帐号在各个数据库上的操作权限。

3、 table_priv 权限表：记录数据表级的操作权限。

4、 columns_priv 权限表：记录数据列级的操作权限。

5、 host 权限表：配合 db 权限表对给定主机上数据库级操作权限作更细致的控制。这个权限表不受 GRANT 和 REVOKE 语句的影响。

MySQL 的 binlog 有几种录入格式？分别有什么区别？

有三种格式哈，statement，row 和 mixed

1、 statement，每一条会修改数据的 sql 都会记录在 binlog 中。不需要记录每一行的变化，减少了 binlog 日志量，节约了 IO，提高性能。由于 sql 的执行是有上下文的，因此在保存的时候需要保存相关的信息，同时还有一些使用了函数之类的语句无法被记录复制。

2、 row，不记录 sql 语句上下文相关信息，仅保存哪条记录被修改。记录单元为每一行的改动，基本是可以全部记下来但是由于很多操作，会导致大量行的改动(比如 alter table)，因此这种模式的文件保存的信息太多，日志量太大。

3、 mixed，一种折中的方案，普通操作使用 statement 记录，当无法使用 statement 的时候使用 row。

InnoDB 引擎的 4 大特性，了解过吗

1、 插入缓冲（insert buffer)

2、 二次写(double write)

3、 自适应哈希索引(ahi)

4、 预读(read ahead)

索引有哪些优缺点?

优点：

1、 唯一索引可以保证数据库表中每一行的数据的唯一性

2、 索引可以加快数据查询速度，减少查询时间

缺点：

1、 创建索引和维护索引要耗费时间

2、 索引需要占物理空间，除了数据表占用数据空间之外，每一个索引还要占用一定的物理空间

3、 以表中的数据进行增、删、改的时候，索引也要动态的维护。

索引有哪几种类型?

1、 主键索引：数据列不允许重复，不允许为 NULL，一个表只能有一个主键。

2、 唯一索引：数据列不允许重复，允许为 NULL 值，一个表允许多个列创建唯一索引。

3、 普通索引：基本的索引类型，没有唯一性的限制，允许为 NULL 值。

4、 全文索引：是目前搜索引擎使用的一种关键技术，对文本的内容进行分词、搜索。

5、 覆盖索引：查询列要被所建的索引覆盖，不必读取数据行

6、 组合索引：多列值组成一个索引，用于组合搜索，效率大于索引合并

创建索引有什么原则呢?

1、 最左前缀匹配原则

2、 频繁作为查询条件的字段才去创建索引

3、 频繁更新的字段不适合创建索引

4、 索引列不能参与计算，不能有函数操作

5、 优先考虑扩展索引，而不是新建索引，避免不必要的索引

6、 在 order by 或者 group by 子句中，创建索引需要注意顺序

7、 区分度低的数据列不适合做索引列(如性别)

8、 定义有外键的数据列一定要建立索引。

9、 对于定义为 text、image 数据类型的列不要建立索引。

10、 删除不再使用或者很少使用的索引

创建索引的三种方式

在执行 CREATE TABLE 时创建索引

```
CREATE TABLE `employee` (
  `id` int(11) NOT NULL,
  `name` varchar(255) DEFAULT NULL,
  `age` int(11) DEFAULT NULL,
  `date` datetime DEFAULT NULL,
  `sex` int(1) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `idx_name` (`name`) USING BTREE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

使用 ALTER TABLE 命令添加索引

```
ALTER TABLE table_name ADD INDEX index_name (column);
```

使用 CREATE INDEX 命令创建

```
CREATE INDEX index_name ON table_name (column);
```

百万级别或以上的数据，你是如何删除的?

1、 我们想要删除百万数据的时候可以先删除索引

2、 然后批量删除其中无用数据

3、 删除完成后重新创建索引。

什么是最左前缀原则？什么是最左匹配原则?

1、 最左前缀原则，就是最左优先，在创建多列索引时，要根据业务需求，where子句中使用最频繁的一列放在最左边。

2、 当我们创建一个组合索引的时候，如(k1,k2,k3)，相当于创建了（k1）、(k1,k2)和(k1,k2,k3)三个索引，这就是最左匹配原则。。

B 树和 B+树的区别，数据库为什么使用 B+树而不是 B 树?

1、 在 B 树中，键和值即存放在内部节点又存放在叶子节点；在 B+树中，内部节点只存键，叶子节点则同时存放键和值。

2、 B+树的叶子节点有一条链相连，而 B 树的叶子节点各自独立的。

1、 B+树索引的所有数据均存储在叶子节点，而且数据是按照顺序排列的，链表连着的。那么 B+树使得范围查找，排序查找，分组查找以及去重查找变得异常简单。.

2、 B+树非叶子节点上是不存储数据的，仅存储键值，而 B 树节点中不仅存储键值，也会存储数据。innodb 中页的默认大小是 16KB，如果不存储数据，那么就会存储更多的键值，相应的树的阶数（节点的子节点树）就会更大，树就会更矮更胖，如此一来我们查找数据进行磁盘的 IO 次数有会再次减少，数据查询的效率也会更快.

覆盖索引、回表等这些，了解过吗?

1、 覆盖索引： 查询列要被所建的索引覆盖，不必从数据表中读取，换句话说查询列要被所使用的索引覆盖。

2、 回表：二级索引无法直接查询所有列的数据，所以通过二级索引查询到聚簇索引后，再查询到想要的数据，这种通过二级索引查询出来的过程，就叫做回表。

网上这篇文章讲得很清晰：

MySQL 覆盖索引与回表

B+树在满足聚簇索引和覆盖索引的时候不需要回表查询数据?

- 在 B+树的索引中，叶子节点可能存储了当前的 key 值，也可能存储了当前的 key 值以及整行的数据，这就是聚簇索引和非聚簇索引。 在 InnoDB 中，只有主键索引是聚簇索引，如果没有主键，则挑选一个唯一键建立聚簇索引。如果没有唯一键，则隐式的生成一个键来建立聚簇索引。
- 当查询使用聚簇索引时，在对应的叶子节点，可以获取到整行数据，因此不用再次进行回表查询。

何时使用聚簇索引与非聚簇索引

动作描述	使用聚集索引	使用非聚集索引
列经常被分组排序	应	应
返回某范围内的数据	应	不应
一个或极少不同值	不应	不应
小数目的不同值	应	不应
大数目的不同值	不应	应
频繁更新的列	不应	应
外键列	应	应
主键列	应	应
频繁修改索引列	不应	应

非聚簇索引一定会回表查询吗?

不一定，如果查询语句的字段全部命中了索引，那么就不必再进行回表查询（哈哈，覆盖索引就是这么回事）。

举个简单的例子，假设我们在学生表的上建立了索引，那么当进行 select age from student where age < 20 的查询时，在索引的叶子节点上，已经包含了 age 信息，不会再次进行回表查询。

组合索引是什么？为什么需要注意组合索引中的顺序?

组合索引，用户可以在多个列上建立索引,这种索引叫做组合索引。

因为 InnoDB 引擎中的索引策略的最左原则，所以需要注意组合索引中的顺序。

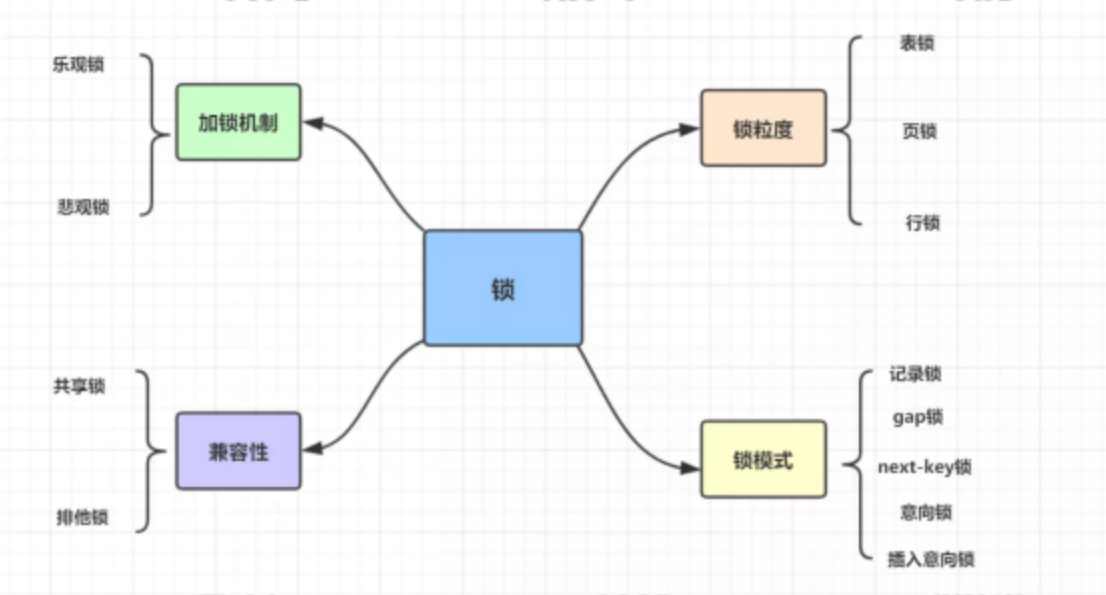
什么是数据库事务?

数据库事务（简称：事务），是数据库管理系统执行过程中的一个逻辑单位，由一个有限的数据库操作序列构成，这些操作要么全部执行,要么全部不执行，是一个不可分割的工作单位。

隔离级别与锁的关系

回答这个问题，可以先阐述四种隔离级别，再阐述它们的实现原理。隔离级别就是依赖锁和 MVCC 实现的。

按照锁的粒度分，数据库锁有哪些呢？锁机制与 InnoDB 锁算法



1、 按锁粒度分有：表锁，页锁，行锁

2、 按锁机制分有：乐观锁，悲观锁

从锁的类别角度讲，MySQL 都有哪些锁呢?

从锁的类别上来讲，有共享锁和排他锁

1、 共享锁：又叫做读锁。当用户要进行数据的读取时，对数据加上共享锁。共享锁可以同时加上多个。

2、 排他锁：又叫做写锁。当用户要进行数据的写入时，对数据加上排他锁。排他锁只可以加一个，他和其他的排他锁，共享锁都相斥。

锁兼容性如下：

兼容性	S	X
S	兼容	不兼容
X	不兼容	不兼容

MySQL 中 InnoDB 引擎的行锁是怎么实现的?

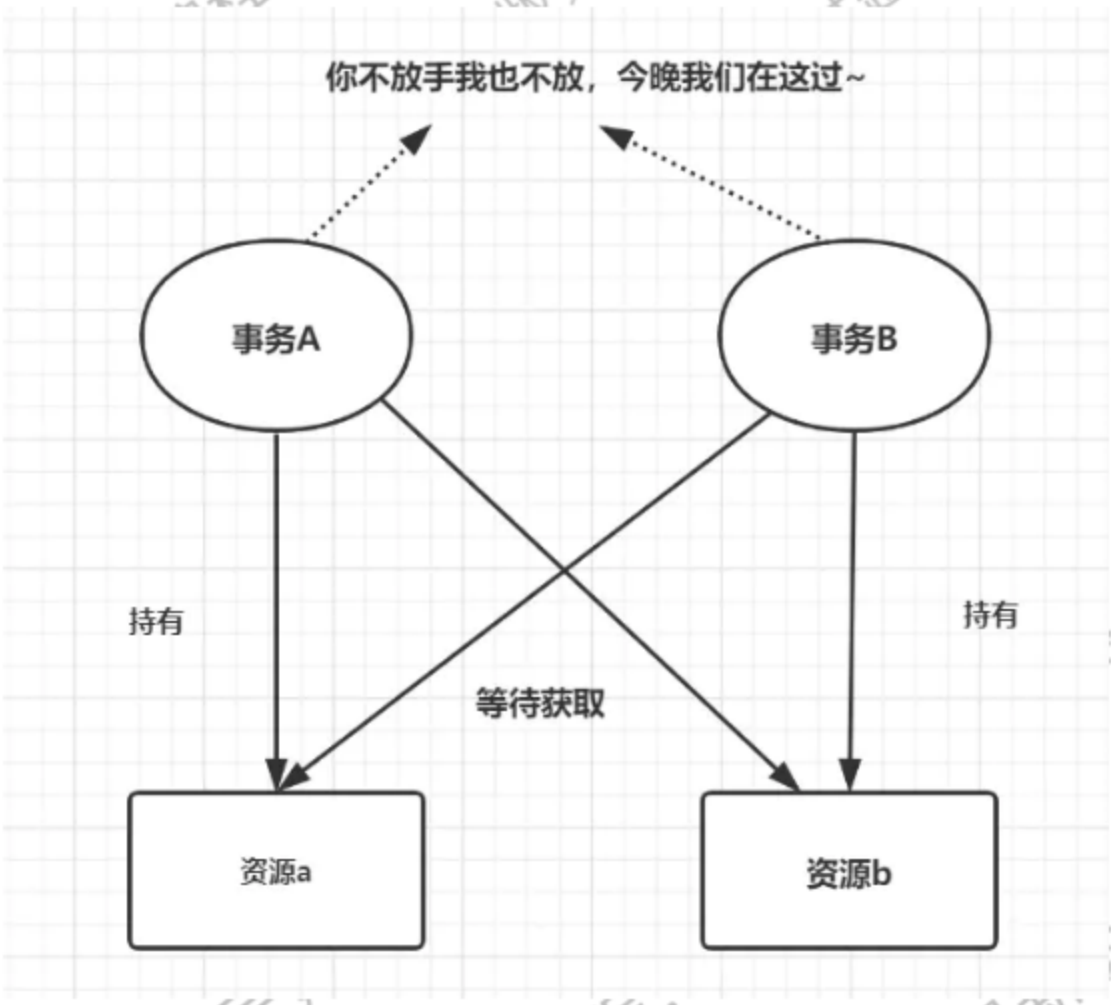
基于索引来完成行锁的。

```
select * from t where id = 666 for update;
```

for update 可以根据条件来完成行锁锁定，并且 id 是有索引键的列，如果 id 不是索引键那么 InnoDB 将实行表锁。

什么是死锁？怎么解决?

死锁是指两个或多个事务在同一资源上相互占用，并请求锁定对方的资源，从而导致恶性循环的现象。看图形象一点，如下：



死锁有四个必要条件：互斥条件，请求和保持条件，环路等待条件，不剥夺条件。

解决死锁思路，一般就是切断环路，尽量避免并发形成环路。

1、 如果不同程序会并发存取多个表，尽量约定以相同的顺序访问表，可以大大降低死锁机会。

2、 在同一个事务中，尽可能做到一次锁定所需要的所有资源，减少死锁产生概率；

3、 对于非常容易产生死锁的业务部分，可以尝试使用升级锁定颗粒度，通过表级锁定来减少死锁产生的概率；

4、 如果业务处理不好可以用分布式事务锁或者使用乐观锁

5、 死锁与索引密不可分，解决索引问题，需要合理优化你的索引，