1. 创建线程的方法：

答案：1.继承thread：定义thread类的子类，并重写该类的run（）方法，该方法的方法体就是线程需要完成的任务，run（）方法也称为线程执行体；创建thread子类的实例，也就是创建了线程对象，启动线程，即调用线程的start（）方法。 2.实现runnable接口创建线程：定义runnable接口的实现类，一样重写run（）方法，这个run（）方法和thread（）中的run（）方法一样也是线程的执行体；创建runnable实现类的实例，并用这个实例作为thread的target来创建thread对象，这个thread对象才是真正的线程对象，第三步依然是调用start（）方法来启动线程 3.使用callable和future创建线程 ：callable和runnable接口不一样的是callable接口提供一个call（）方法作为线程执行体，call（）方法要比run（）方法功能强大，call方法可以有返回值，也可以声明抛出异常。过程：创建callable接口的实现类，并实现call（）方法，然后创建该实现类的实例（可以用lambda表达式），使用FutureTask类来包装callable对象，该FutureTask对象封装了Callable对象的call（）方法的返回值；使用FutureTask对象作为Thread对象的target创建并启动线程（FutureTask实现了Runnable接口）；调用FutureTask对象的get（）方法获得子线程执行结束后的返回值 4.使用线程池，例如用Executor框架：

1. 创建（实例化）对象的方式

答案：1.通过new关键字创建对象2.调用对象的clone（）方法创建对象,无论何时我们调用一个对象的clone（）方法，JVM都会创建一个新的对象，将前面对象的内容全部拷贝进去，用clone方法创建对象并不会调用任何构造函数，要使用clone方法，我们必须先实现clonable接口并实现其定义的clone方法。3. 使用class类的newInstance方法，通过反射对对象进行初始化：Person.class.newInstance();或者Class.forName(路径.Person).newInstance(); 调用的是无参的构造函数创建对象4.序列化：通过IO流（包括反序列化），如运用反序列化手段，调用java.io.ObjectInputStream对象的readObject（）方法。5.通过工厂方法返回对象：如String str = String.valueOf(23); 单例模式 6.使用Constructor类的newInstance方法,像class的newInstance，构造器类也有一个newInstance方法创建对象，可以调用无参的和私有的构造函数，Person.class.getConstructor().newInstance();

1. 观察者设计模式：zookeeper分布式锁，解决了解耦的问题。就是能保证足够的扩展性，后续可能会随意要求增添或删除公告牌，可以看<https://zhuanlan.zhihu.com/p/20540213>
2. 讲讲高并发下的接口幂等性怎么实现？

答案：应用场景，实际系统中很多操作，是不管做多少次，都应该产生一样的效果或返回一样的结果，幂等函数，或幂等方法，是指可以使用相同参数重复执行，并能获得相同结果的函数。这些函数不会影响系统状态，也不用担心重复执行会对系统造成改变。如getUsername（）和setTrue（）函数就是一个幂等函数，更复杂的实现幂等性是利用唯一交易号（流水号）实现。

幂等性场景：

1. 查询操作：查询一次和多次，在数据不变的情况下，查询结果是一样的，select是天然的幂等操作。
2. 删除操作：删除操作也是幂等的，删除一次和多次删除都是把数据删除，（注意可能返回的结果不一样，删除的数据不存在，返回0，删除的数据多条，返回结果多个）
3. 唯一索引：防止新增脏数据，支付宝的资金账户，支付宝也有用户账户，每个用户账户只能有一个资金账户，怎么防止给用户创建多个资金账户，那么给资金账户表中的用户ID加唯一索引，所以一个用户新增一个资金账户记录成功。要点：唯一索引或唯一组合索引来防止新增数据存在脏数据（当表存在唯一索引，并发时新增报错时，再查询一次就可以了，数据应该已经存在了，返回结果即可
4. Token机制：防止页面重复提交。原理上通过session token来实现的（也可以用过redis来实现），当客户端请求页面时，服务器会生成一个随机数token，并且将token放到session中，然后将token发给客户端（一般通过构造hidden表单）。下次客户端提交请求时，token会随着表单一起提交到服务器端。服务器端第一次验证验证相同后，会将session的token值更新一下，若用户重复提交，第二次的验证判断将失败，因为用户提交的表单中的token没变，但服务器端session中token已经变了。
5. 悲观锁：获取数据的时候加锁获取，select \* from table\_xxx where id = ‘xxx’ for update;注意：id字段一定是主键或者唯一索引，不然是锁表，会死人的，悲观锁使用时一般伴随事务一起使用，数据锁定时间可能会很长，根据实际情况选用。
6. 乐观锁：乐观锁只是在更新数据那一刻锁表，其他时间不锁表，所以相对于悲观锁，效率更高。乐观锁的实现方式多种多样，可以通过版本号机制或者其他条件。1.通过版本号机制实现update table\_xxx set name = #name#, version=version + 1 where version = #version# 2.通过条件限制update table\_xxx set avai\_amount = avai\_amount - #subAmount# where avai\_amount - #subAmount# >= 0，要求：quality - #subQuality# >= 0,这个情景不适合用版本号机制，只更新是做数据安全校验，适合库存模型，扣份额和回滚份额，性能较高。
7. 分布式锁：如果是分布式系统，构建全局唯一索引比较困难，例如唯一性的字段没法确定，这时候就可以引入分布式锁，通过第三方的系统（redis或者zookeeper）,在业务系统插入数据或者更新数据，获取分布式锁，然后做操作，之后释放锁，这样其实是把多线程并发的思路，引入多个系统，也就是分布式系统中的解决思路。要点：某个长流程处理过程要求不能并发执行，可以在流程执行之前根据某个标志（用户ID+后缀等）获取分布式锁，其他流程执行时获取锁就会失败，也就是说同一时间该流程只能有一个能执行成功，执行成功后，释放分布式锁（分布式锁要第三方系统提供）。
8. Select+insert：并发不高的系统，或者一些任务JOB，为了支持幂等，支持重复执行，简单的处理方法是，先查询下一些关键数据，判断是否已经执行过，在进行业务处理，就可以了。注意：核心高并发流程不要用这种方法。
9. 状态机幂等：在设计单据相关的业务，或者是任务相关的业务，肯定会涉及到状态机（状态变更图），就是业务单据上面有状态，状态在不同情况下会发生变更，一般情况下存在有限状态机，这时候，如果状态机已经处于下一个状态，这时候来了上一个状态的变更，理论上是不能够变更的，这样的话，保证了有限状态机的幂等。注意：订单等单据类业务，存在很长的状态流转，一定要深刻理解状态机，对业务系统设计能力提高有很大帮助。
10. 对外提供接口的API如何保证幂等：如银联提供的付款接口：需要接入商户提交付款请求时附带：source来源，seq序列号，source+seq在数据库里唯一索引，防止多次付款（并发时只能处理一个请求）。重点：对外提供接口为了支持幂等调用，接口有两个字段必须传，一个是source，一个是来源方序列号seq，这两个字段在提供方系统里面做联合唯一索引，这样当第三方调用时，先在本方系统里面查询一下，是否已经处理过，返回相应处理结果，没有处理过，进行相应处理，返回结果。注意，为了幂等友好，一定要先查询一下，是否处理过该笔业务，不查询直接插入业务系统，会报错，但实际已经处理了。
11. MySQL与Redis数据不一致怎么处理？不管是先写MySQL数据库，再删除redis缓存，还是先删除缓存，再写库，都有可能出现数据不一致的情况。举例子，如果删除了缓存redis，还没有来得及写库MySQL，另一个线程就来读取，发现缓存为空，则去数据库中读取数据写入缓存，此时缓存中为脏数据。或者先写了库，在删除缓存前，写库的线程宕机了，没有删除掉缓存，则也会出现数据不一致的情况。因为读和写是并发的，没办法保证顺序，就会出现缓存和数据库的数据不一致的情况。

答案：第一种方案：采用延时双删策略，在写库前都设置redis.del(key)操作，并且设置合理的超时时间。具体步骤：1.先删除缓存，2.再写数据库 3.休眠500毫秒 4.再次删除缓存。具体需要休眠多久呢？需要评估自己的项目的读数据业务逻辑的耗时，这么做的目的，就是确保读请求结束，写请求可以删除读请求造成的缓存脏数据。当然，这种策略还要考虑redis和数据库主从同步的耗时，最后的写数据的休眠时间，则在读数据业务逻辑的耗时基础上，加几百ms即可。设置合理的缓存过期时间：从理论上来说，给缓存设置过期时间，是保证最终一致性的解决方案。所有的写操作以数据库为准，只要到达缓存过期时间，则后面的读请求自然会从数据库中读取新值然后回填缓存。

该方法的弊端，结合双删策略+缓存过期时间，这样最差的情况就是在超时时间内数据存在不一致，而且又增加了写请求的耗时。

第二种方案：异步更新缓存（基于订阅binlog的同步机制），技术整体思路：MySQL binlog增量订阅消费+消息队列+增量数据更新到redis。具体的步骤：1.读redis：热数据基本都在redis中 2.写MySQL：增删改都是操作MySQL 3.更新redis数据：MySQL的数据操作binlog 来更新到redis（MySQL数据的增删改命令都会被存在binlog,且通过二进制的形式存的）然后通过binlog来更新redis。

Redis更新：数据操作主要分为两大块，一个是全量（将全部数据一次写入到redis）另一个是增量（实时更新），这里说的增量，指的是MySQL的update、insert、delete变更数据。读取binlog后分析，利用消息队列，推送更新各台的redis缓存数据，这样一旦MySQL中产生了新的写入、更新、删除等操作，就可以把binlog相关的消息推送到redis、redis再根据binlog中的记录，对redis进行更新，其实这种机制，很类似MySQL的主从备份机制，因为MySQL的主备也是通过binlog来实现的数据一致性。这里可以结合使用canal（阿里的一款开源框架），通过该框架可以对MySQL的binlog进行订阅，而canal正是模仿了MySQL的slave数据库的备份请求，使得redis的数据更新更新达到了相同。当然，也可以使用第三方，kafka,rabbitMQ等来实现推送更新redis。

1. 主从复制？

答案： 主服务器上面的任何修改都会保存在二进制日志Binary log里面，从服务器上面启动一个I/O thread（实际上就是一个主服务器的客户端进程），连接到主服务器上面请求读取二进制日志，然后把读取到的二进制日志写到本地的一个Realy log里面。从服务器上面开启一个SQL thread定时检查Realy log，如果发现有更改立即把更改的内容在本机上面执行一遍。

如果一主多从的话，这时主库既要负责写又要负责为几个从库提供二进制日志。此时可以稍做调整，将二进制日志只给某一从，这一从再开启二进制日志并将自己的二进制日志再发给其它从。或者是干脆这个从不记录只负责将二进制日志转发给其它从，这样架构起来性能可能要好得多，而且数据之间的延时应该也稍微要好一些。

1. 数据库主从一致性的几种解决方法？数据源至少都会配有两个节点，一个业务处理使用的节点，一个甚至多个从节点，这些从节点就是我们说的冷备。业务处理节点和备份节点在一定的时间间隔内进行数据同步，从而保证党一个数据源坏掉之后，数据也不会丢失，或者丢失很少（主要看同步的时间间隔）。但是为了提高资源的使用效率，所以有人就提出了，可不可以让冷备也被利用起来，替主节点分担部分压力（并发情况），所以就提出了读写分离的方案。读写分离提高了资源的利用效率的同时也引出了一个问题，就是由于延时（网络传输、操作）而引起的数据库主从不一致的问题，解决：

答案：1. 半同步复制：方案优点：利用数据库原生功能，比较简单；方法缺点：主库的写请求时延会增长，吞吐量会降低。主库在执行完客户端提交的事务后（写完binlog）不是立刻返回给客户端，而是等待至少一个从库接收并写到relaylog中（写完relaylog之后返回一个ACK给主库，主库接收到ACK后认为写操作完成了）才返回给客户端，半同步复制提高了数据的安全性，同时它也造成了一定的时间延迟，这个延迟至少是一个TCP/IP往返时间，所以半同步复制最好在低延时的网络中使用。

早期仅仅支持异步复制：MySQL默认的复制是异步的，主库在执行完客户端提交的事务后会立即将结果返回给客户端，并不关心从库是否已经接收并处理，这样就会有一个问题，主如果cash掉了，此时主上已经提交的事务可能并没有传到从库上，如果这时候，强行将从提升为主，可能导致数据不一致。

2.数据库中间件：1.所有的读写都走数据库中间件，通常情况下，写请求路由到主库，读请求路由到从库。2.记录所有路由到写库的key，在主从同步时间窗口内，如果有读请求访问中间件，此时有可能从库还是旧数据，就把这个key上的读请求路由到主库。3.在主从同步时间过完后，对应key的读请求继续路由到从库。相关的中间件有canal和otter

Canal基于数据库增量日志解析，提供增量数据订阅&消费。Otter是一个分布式数据库同步系统，尤其是在跨机房数据库同步方面，有很强的功能，他是基于数据库增量日志解析，实时将数据同步到本机房或跨机房的MySQL/Oracle数据库。该方案的优点：能保证数据绝对一致，缺点：数据库中间件的成本比较高。

3.缓存记录写key法，写流程：1.如果key要发生写操作，记录在cache里，并设置“经验主从同步时间”的cache超时时间，2.然后修改主库。读流程：1.先到缓存里查看，对应key有没有相关数据，2.有相关数据说明缓存命中，这个key刚发生过写操作，此时需要将请求路由到主库读最新的数据。3.如果缓存没有命中，说明这个key上近期没有发生过写操作，此时将请求路由到从库，继续读写分离。该方案优点：相对数据库中间件，成本较低，缺点：为了保证“一致性”，引入了一个cache组件，并且读写数据库时都多了缓存操作。

4. 主库将变更写binlog日志，然后从库连接到主库之后，从库有一个IO线程，将主库的binlog日志拷贝到自己本地relaylog中（中继日志），接着从库中有一个SQL线程会从中继日志读取binlog,然后执行binlog日志中的内容，也就是在自己本地再执行一遍SQL，这样就可以保证自己跟主库的数据是一样的。注意：从库同步主库数据的过程是串行化的，也就是说主库上并行的操作，在从库上会串行执行，加上拷贝日志，所以在高并发场景下，从库的数据一定比主库慢一下，是有延时的，即刚写入主库的数据是读不到的，要过几十毫秒，甚至几百毫秒才能读取到。另外，如果主机突然宕机了，然后恰好数据还没有同步到从库，那么有些数据可能在从库上是没有的，有些数据就丢失了。

1. 如何实现MySQL读写分离？

答案：读写分离，把读操作分散到从服务器中，并且对多个从服务器能实现负载均衡，1. MaxScale可以帮我们实现读写分离的同时负载均衡功能。MaxScale目前提供的插件功能分为5类，1.认证插件：提供了登陆认证功能，Maxscale会读取并缓存数据库中user表中的信息，当有连接进来时，先从缓存信息中进行验证，如果没有此用户，会从后端数据库中更新消息，再次进行验证。 2.协议插件：包括客户端连接协议，和连接数据库的协议。 3.路由插件：决定如何把客户端的请求转发给后端数据库服务器，读写分离和负载均衡的功能就是由这个模块实现的。 4.监控功能：对各个数据库服务器进行监控，例如发现某个数据库服务器响应很慢，那么就不向其转发请求了。 5.日志和过滤插件：提供简单的数据库防火墙功能，可以对SQL进行过滤和容错。

配置方式：配置前需要在master中为Maxscale创建两个用户，用于监控模块和路由模块。4006是read-write listener使用的端口，用于连接Maxscale，6603是maxadmin listener使用的端口，用于Maxscale管理器。

2. 使用MySQL的com.mysql.jdbc.ReplicationDriver驱动

3. 构造不同的数据库连接池，在service层通过判断方法的读写类型，读方法使用从数据库连接池，写方法使用主数据库连接池。

4. 使用阿里的mycat或者360的Atlas来实现分库分表，读写分离和负载均衡等。

5. 使用Spring AOP解决数据源切换的问题，那么就可以和mybatis，Druid整合到一起了，我们在整合spring和mybatis时，我们只需写dao接口和对应的SQL语句，Spring创建DAO实例，他通过我们注入的数据源，帮我们完成从中获取数据库连接、使用连接执行SQL语句的过程以及最后归还连接给数据源的过程。如果我们能在调用dao接口时根据接口方法动态地选择数据源，就可以做到读写分离了。实现动态切换数据源主要依赖spring-aop和spring aspects,其中RoutingDataSourceImpl是实现动态切换功能的核心类。

1. 从服务器实现负载均衡？

答案：有三种，轮询，加权轮询，负载分配。1.轮询算法：每一次来自网络的请求轮流分配给内部的每台服务器，从1到N然后重新开始，此种算法适用于服务器组中的所有服务器都有相同的软硬件配置并且平均服务请求相对均衡的情况。2.加权轮询法：不同的后台服务器机器的配置和当前系统的负载均衡并不相同，因此他们的抗压能力也不一样，给配置高，负载低的机器分配更高的权重，使其能处理更多的请求，而配置低，负载高的机器分配较低的权重，降低其系统负载，加权轮询很好的处理了这一问题，并将请求按照顺序且根据权重分配给后端。3.负载分配：依据当前的空闲状态（但是测试每个节点的内存使用率，CPU利用率，再做比较选出最闲的那个，效率太低）。

1. MySQL集群高可用？

答案：在服务器架构时，为了保证服务器不宕机在线状态，需要为每台单点服务器提供冗余机，对于写服务器来说，需要提供一台同样的写-冗余服务器，当写服务器健康时（写-冗余通过心跳检测），写-冗余作为一个从机的角色复制写服务器的内容与其做一个同步，当写服务器宕机时，写-冗余服务器便顶上来作为写服务器继续提供服务，对外界来说这个处理过程是透明的，即外界仅通过一个IP访问。

1. 分库分表之后id主键如何处理？

答案：因为分成多个表之后每个表都是从1开始累加，这样是不对的，我们需要创建一个全局唯一的id来支持。生成全局id有以下几种方式：需满足主键的全局唯一性、单调递增性、也要满足业务相关性。

1. UUID：不适合作为主键，因为太长了，并且无序不可读，查询效率低，比较适合用于生成唯一的名字的标识比如文件的名字。主键ID自增的好处：可以通过id来排序，提升数据的写入性能，主键其实就是索引，B+树是顺序存储的，如果有序，直接插入到最后就好了，如果无序，则每次插入之前还得查找他应该所在的位置，无疑会增加数据的移动等相关的开销。UUID还不具备业务相关性，而我们的唯一主键是和业务相关的。2.数据库自增id:两台数据库分别设置不同步长，生成不重复id的策略来实现高可用。这种方式生成的id有序，但是需要独立部署数据库实例，有些浪费资源，成本高，还会有性能瓶颈。 3.利用redis INCR命令生成id，redis 的incr命令具备了“incr and get”的院子操作，且redis是单线程架构，incr不会出现重复id，性能比较好，灵活方便，不依赖于数据库，但是引入了新的组件造成系统更加复杂，可用性降低，编码更加复杂，增加了系统成本。 4.Twitter的snowflake算法：生成64位的id，1bit是不用的，二进制里第一bit为1的话代表负数，但是我们生成的id都是正数，所以第一bit统一都是0，41bit作为毫秒数，表示时间戳，换算成年是69年的时间，10bit记录工作机器id，代表的是这个服务最多可以部署在2^10（1024）台机器上，但是其中前5个代表机房id，后5个代表机器id，即32个机房，32台机器。接下来的12bit，记录同一个毫秒内产生的不同id. 5.美团的Leaf分布式id生成系统：能保证全局唯一性，趋势递增，单调递增，信息安全；原来的数据库的方案：每次获取id都得读写一次数据库，造成数据库压力大，改为利用proxy server批量获取，每次获取一个segment（step决定大小）号段的值。用完之后去数据库获取新的号段，可以大大减轻数据库的压力。各个业务不同的发号需求用biz\_tag字段来区分，每个biz\_tag的id获取相互隔离，互不影响，如果以后有性能需求需要对数据库进行扩容，不需要对上述描述的复杂的扩容操作，只需要对biz\_tag分库分表就行。Id号码是趋势递增的64位数字，缺点是id号码不够随机，能够泄露发号数量的信息，不太安全。TP999数据波动大，当号段使用完之后还是会hang在更新数据库的IO上，tg999数据会出现偶尔的尖刺，针对这个问题，双buffer的方式，leaf服务器内部有两个号段缓存区segment，当前号段已下发10%时，如果下一个号段未更新，则另启一个更新线程去更新下一个号段，当前号段全部下发完成后，如果下个号段准备好了则切换到下一个号段为当前segment接着下发，循环往复。每个biz\_tag都有消息速度监控，通常推荐segment长度设置为服务器高峰期发号QPS的600倍（10分钟），这样即使DB宕机，leaf仍然持续发号10-20分钟不受影响，每次请求都会判断下个号段的状态，从而更新此号段，所以偶尔的网络科技抖动不会影响下个号段的更新。

Leaf-segment方案可以生成趋势递增的ID，同时ID是可计算的，不适用于订单ID生成场景，竞争对手在两天中午12点下单就能推算出来公司一天的订单量。这个是不能忍受的，所以有了leaf-snowflake方案，完全沿用snowflake方案的bit位设计，即是“1+41+10+12”的方式组装ID号，对于workerID的分配，当服务集群数量小的情况下，完全可以手动配置，但leaf服务规模较大，动手配置成本太高，所以使用zookeeper持久顺序节点的特性自动对snowflake节点配置workerID，leaf-snowflake是按照下面几个步骤启动的：1. 启动leaf-snowflake服务，连接zookeeper，在leaf\_forever父节点下检查自己是否已经注册过（是否有该顺序子节点）。2. 如果有注册过直接取回自己的workerID（zk顺序节点生成的int类型ID号），启动服务。3. 如果没有注册过，就在该父节点下面创建一个持久顺序节点，创建成功后取回顺序号当做自己的workerID号，启动服务。弱依赖zookeeper，除了每次去zk拿数据以外，也会在本机文件系统上缓存一个workerID文件，当zookeeper出现问题，恰好机器出现需要重启时，能保证服务正常启动，这样做到了对三方组件的弱依赖，一定程度上提高了SLA。但是这种方案依赖时间，如果机器的时钟发生了回拨，那么就会有可能生成重复的ID号，需要解决时钟回退问题。

1. 数据库为什么要优化？

答案：一个应用吞吐量瓶颈往往出现在数据库的处理速度上，随着应用程序的使用，数据库数据逐渐增多，数据库压力逐渐增大，关系型数据库的数据是存放在磁盘上的，读写速度较慢（与内存中的数据相比）。

1. 索引优化？

答案：1.保持独立列，不要用函数和运算，这样会导致索引失效而进行全表扫描，2.避免隐式转换，当查询条件左右两侧类型不匹配的时候发生隐式转换，隐式转换带来的影响就是可能导致索引失效进而进行全表扫描，3. 索引不会包含有null的值，设计多列复合索引时一定要注意，所有列必须不能为null，因为含有null的列是不会被加入索引的，直接无效 4.避免使用or连接，同列or可以转换为in，innodb对or的查询条件会运行多次单独的查询然后合并结果，时间复杂度为o(n)，而对于in操作，innodb使用的是基于树的二分查找，时间复杂度是o(logn),显然log更快。 5.适当利用覆盖索引加速，如果要查询的数据都在复合索引列中，那么就可以使用覆盖索引，直接查询到结果返回，省去“回表”，但要注意别把索引搞得太多或者太长，维护索引会降低性能。 6.正确认识最左前缀匹配索引，一定要注意从最左开始匹配才能利用上索引，一旦使用前缀匹配，那么将不能使用覆盖索引，也不能利用索引来排序或者group 7.复合索引顺序很重要，复合索引第一列不需要再次建立单列索引，order by和group by后面接的列与复合索引顺序一致时可以利用复合索引避免排序和临时表 8.范围查询会导致右侧条件索引失效，查询中的某个列有范围查询，则其右边所有列都无法使用索引优化查找。

1. 使用索引的注意事项？

答案：1.在经常需要搜索的列上，可以加快搜索的速度。 2.在经常使用where子句中的列上面创建索引，加快条件的判断速度， 3.在经常需要排序的列上创建索引，因为索引已经排序，这样查询可以利用索引的排序顺序，加快排序查询时间， 4.对于中到大型表索引都是非常有效的，但是特大型表的话维护开销会很大，不适合建索引 5.在经常用在连接的列上，这些列主要是以下外键，可以加速连接的速度， 6.避免where子句中对字段加函数，这会造成索引无法命中。 7.在使用innodb时使用与业务无关的自增主键作为主键，即使用逻辑主键，而不要使用业务主键。 8.将打算加索引的列设置为not null，否则将导致引擎放弃使用索引而进行全表扫描。 9.删除长期未使用的索引，不用的索引的存在会造成不必要的性能损耗，可以使用sys库的chema\_unused\_indexes视图来查询哪些索引从未被使用 10.在使用limit offset查询缓存时，可以借助索引来提高性能。

1. 压力测试工具mysqlslap？ 1.可以自动生成SQL测试，2.并发测试 3.多轮测试 4.存储引擎测试
2. 请解释一下什么是Nginx？

答案：Nginx是一款免费的、自由的、开源的、高性能HTTP服务器和反向代理服务器。Nginx以其高性能、稳定性、丰富的功能、简单的配置和低资源消耗而闻名。Nginx可以像Tomcat似的托管网站，进行HTTP服务处理，也可以作为反向代理服务器，负载均衡器和HTTP缓存。Nginx解决了服务器的C10K（就是在一秒之内连接客户端的数目为10K即1万）问题。他的设计不像传统那样使用线程处理请求，而是更加高级的机制—事件驱动机制，是一种异步事件驱动结构（因为采用了最新的epoll事件处理模型（消息队列））。

1. Nginx的特性：跨平台，配置简单，非阻塞、高并发连接：数据复制时，磁盘IO的第一阶段是非阻塞的，得益于Nginx采用了最新的epoll事件处理模型（消息队列，提供了一个队列，排队解决），Nginx代理和后端web服务器间无需长连接，Nginx接收用户请求是异步的，即先将用户请求全部接收下来，再一次性发送给后端web服务器，极大减轻了后端web服务器的压力， 发送响应报文时，是边接收来自后端web服务器的数据，边发送给客户端， 网络依赖性低，只要能ping就可以实现负载均衡， 支持内置服务器检测，Nginx能够根据应用服务器处理页面返回的状态码、超时信息等检测服务器是否出现故障，并及时返回错误的请求重新提交到其他节点上， 节省带宽、稳定性高、内存消耗小、成本低。
2. Nginx和Apache之间的不同点？

答案：1.Nginx是基于事件的web服务器，Apache是一个基于流程的服务器。 2.Nginx所有请求都由一个线程处理，Apache单个线程处理单个请求 3.Nginx避免子进程的概念，Apache基于子进程 4.Nginx类似于速度，Apache类似于功率， 5.Nginx在内存消耗和连接方面比较好，Apache在内存消耗和连接上没有提高 6.Nginx在负载均衡方面表现很好，Apache当流量达到进程的极限时，Apache将拒绝新的连接， 7.Nginx只具有核心功能，Apache提供了比Nginx更多的功能 8.Nginx的性能和可伸缩性不依赖于硬件，Apache依赖于CPU和内存等硬件组件

1. Nginx如何处理HTTP请求？

答案：Nginx是一个高性能的web服务器，能够同时处理大量的并发请求，结合使用多进程和异步非阻塞机制， 多进程：服务器每当收到一个客户端时，就有服务器主进程生成一个子进程出来和客户端建立连接进行交互，直到断开连接，该子进程就结束了。使用进程的好处就是各个进程间相互独立，不需要加锁，减少了使用锁对性能造成影响，同时降低编程的复杂度，降低开发成本，其次，采用独立的进程，可以让进程互相之间不会影响，如果一个进程发生异常退出时，其他进程正常工作，master进程很快启动新的worker进程，确保服务不会中断，从而降低风险，缺点是操作系统生成一个进程需要内存复制等操作，在资源和时间上产生一定的开销，当有大量请求时，会导致系统性能降低。 异步非阻塞机制：每个工作进程使用异步非阻塞方式，可以处理多个客户端请求，当某个工作进程接收到客户端的请求以后，调用IO进行处理，如果不能立即得到结果，就去处理其他请求（即为非阻塞），而客户端在此期间也无需等待响应，可以去处理其他事情（即为异步）。当IO返回时，就会通知此工作进程，该进程得到通知，暂时挂起当前处理的事务去响应客户端的请求。

1. Nginx反向代理服务器的优点是什么？

答案：反向代理服务器可以隐藏源服务器的存在和特征，他充当互联网云和web服务器之间的中间层，这对于安全方面来说很好，特别是当您使用web托管服务时。

1. Nginx服务器的最佳用途？

答案：最佳用法是在网络上部署动态HTTP内容，使用SCGI、WSGI应用程序服务器、用于脚本的FastCGI处理程序，他还可以作为负载均衡器。

1. 使用Nginx做负载均衡？

答案：将多台服务器组成一个集群，当用户访问到一个转发服务器，再由转发服务器将访问分发到压力更小的服务器。Upstream定义负载节点池，使用Nginx做负载均衡的两大模块，location模块进行URI匹配，proxy模块发送请求给upstream定义的节点池。Nginx实现负载均衡的策略有五种：1.轮询：每个请求按时间顺序逐一分配到不同的后端服务器，如果后端某个服务器宕机，能自动剔除故障系统。 2.权重weight：weight的值越大分配到的访问概率越高，主要用于后端每台服务器性能不均衡的情况下，其次是在主从的情况下设置不同的权值，达到合理有效的利用主机资源。 3.ip\_hash(ip绑定)：每个请求按访问IP的哈希结果被分配，使来自同一个IP的访客固定访问一台后端服务器，并且可以有效解决动态网页存在的session共享问题 4.fair（第三方插件）必须安装upstream\_fair模块，对比weight、ip\_hash更加智能的负载均衡算法，fair算法可以根据页面大小和加载时间长短智能地进行负载均衡，响应时间短的优先分配。哪个服务器的响应速度快，就将请求分配到那个服务器上。 5.url\_hash(第三方插件)：必须安装Nginx的hash软件包，按访问URL的hash结果来分配请求，使每个URL定向到同一个后端服务器，可以进一步提高后端缓存服务器的效率。

1. Nginx服务器上的master和worker是什么？

答案：主程序master process 启动后，通过一个for循环来接收和处理外部信号，主进程通过fork（）函数产生worker子进程，每一个子进程执行一个for循环来实现Nginx服务器对事件的接收和处理。一般推荐worker进程数和CPU内核数一致，这样一来不存在大量的子进程生成和管理任务，避免了进程之间竞争CPU资源和进程切换的开销。而且Nginx更好的利用多核特性，提供了CPU亲缘性的绑定选项，我们可以将某一个进程绑定在某一个核上，这样不会因为进程的切换带来cache的失效。对于每个请求有且只有一个工作进程对其进行处理，每个worker都是从master进程fork过来的，在master里面，先建立好需要listen的socket（listenfd）之后，然后再fork出多个worker进程。所有worker进程的listenfd会在新连接到来时变得可读，为保证只有一个进程处理该连接，所有worker进程在注册listenfd读时间前抢占accept\_mutex，抢占互斥锁的那个进程注册listenfd读事件，在读事件里调用accept接受该连接。当一个worker在accept这个连接之后，就开始读取请求，解析请求，处理请求，产生数据，返回给客户端，最后才断开连接。 Master和worker的交互依赖于socket实现的管道来实现。

1. 请解释代理设计中的正向代理和反向代理？

答案：代理服务器一般指局域网内部的机器通过代理服务器发送请求到互联网上的服务器，代理服务器一般作用在客户端，我们的客户端在进行翻墙的时候，我们使用的是正向代理，通过正向代理，在我们的客户端运行一个软件，将我们的HTTP请求转发到其他不同的服务器，实现请求的分发。反向代理服务器作用在服务器端，他在接收客户端的请求之后，然后将请求分发给具体的服务器进行处理，然后再将服务器的相应结果返回给客户端，Nginx就是一个反向代理服务器软件。客户端必须设置正向代理服务器，当然前提是要知道正向代理服务器的IP地址，还有代理程序的端口，而反向代理就不需要，客户端向反向代理的命名空间（name-space）中的内容发送普通请求。反向代理将判断像何处（原始服务器）转交请求，并将获得的内容返回给客户端。

HR面面经？

1. 为什么想来顺丰？

答案：顺丰是国内最好最大的物流公司，有着海量的货物的吞吐量，是一个非常好的技术平台，在这里能够得到技术上的锻炼，实现我的技术追求。同时可以给公司带来一点技术上的奉献，然后通过几年的努力，希望自己能够成为独当一面的人，独自负责一项业务或者带领一个小团队

1. 你做的最有成就感的项目是什么

答案：互联网+

1. 你再这段经历中，你觉得对你最大的挑战是什么呢

答案：时间短，任务重，要在短时间内快速的完成整个系统的设计，调研技术，然后跟团队商量确定出最终方案，然后自己动手实现网站所有功能。

1. 听起来你很了解顺丰，你是从哪些途径了解顺丰的呢

答案：平时自己也经常寄快递，用顺丰，体验非常好，就格外关注了一下顺丰公司招聘，然后刚好看到了牛客sp专场宣传的，我就赶紧投了

1. 你有收到其他公司的offer吗，或者投递其他的公司吗

答案：京东，刚过了一面，华为，今天早上刚投递

1. 你对于薪资的期望和要求是多少

答案：18K

1. 你的父母是做什么的，对你的工作职位和地点有要求吗

答案：农民，对我的工作的话，他们也不懂，不过最重要的是他们非常支持我

1. 在你的过往中有受到什么挫折么

答案：挫折的话我就说一下，我考研阶段我爸生病了，然后我是蒙古族的么，我爸妈不太会说汉语，就需要我亲力亲为，带我爸去长春做手术，那时候都11月份了，我复习时间也比较紧张，我就只能白天陪爸爸，晚上熬夜学习，还好，最后考上了自己想上的学校。

1. 如果加入顺丰想往哪方面发展（考察对顺丰的了解）

答案：如果可以的话，我就想去做智慧地图的那个部门或者岗位上去，智慧物流提供一站式地图服务解决方案，在客户下单之后、智能调度、中转分拣、规划运输、末端配送等环节提供货车和骑行场景的高效路径规划服务，提高效率。我觉得如果我有幸能参加这个部门，那么我的技术水平可以得到大幅度提高，通过我的努力，几年之后自己可以成为独当一面的人，带领一个小团队，给公司提供一点技术上的奉献。

1. 介绍一个项目
2. 遇到的感觉最有压力的事情

答案：我这个人对压力平时不敏感，抗压能力比较强，如果真说有压力的话就是男朋友非常优秀，有字节，腾讯的offer。

1. 身边同学老师怎么评价你

答案：乐观开朗，积极抗压能力强，面对项目比较多时间比较紧张的时候也能很好的推进各项目的进度。

1. 为什么想做软件开发这个方向呢

答案：自己做出来东西有成就感

1. 考虑过，毕业之后通过什么维度去评判一份更适合自己的工作呢

答案：1.是不是真正喜欢，能不能从这个工作里获得成就感，如果能真正喜欢，并且全身心投入到公祖中感觉到快了并且获得成就感，那么能力进步升职是一件水到去程的事情，2.看工作的长远的发展，自己倾向于经过不断的积累，给公司带来一定的产出，能够在业界有一定的影响力，

1. 你觉得自己是一个怎样性格的人

抗压能力强，喜欢做技术，表现出对技术的热爱，钻研精神

1. 在深圳科创企业多，但生活节奏快，你会担心工作之后压力会大吗

答案：我这个人适应能力强，乐观开朗，之前项目比较紧张的时候也能很好的调节情绪

1. 学生经历比较多，介绍一下有什么收获

答案：

1. 学过的课程，师兄师姐的去向

答案：华为，如果继续问，你咋不去，我就说在面试中，还没有结果，如果发了，会平衡工作和进去之后的工作情况，顺丰是一家我非常喜欢的公司。工资差距不是很大的话优先考虑顺丰，物流的信息化是一套很庞杂的系统，收件，分拣，派件，仓储，中转等，每一步都是一个很大的系统。我觉得在这里我能得到很大的提升，

1. 为什么学习java

答案：java简单有效，移植性高，因为有封装这种么，可以用的最多的后端语言，非常受欢迎。然后我个人的话是非常喜欢java这个技术点，然后看java代码我觉得非常舒服，打从心里底里喜欢的那种。

1. 为什么没去实习
2. 如何学习掌握知识技术的，什么时候确定的方向

答案：看视频，看博客，看书，研一下

1. 学习生活压力比较大的事情如何处理，如何调节心情

答案：可能神经大条，抗压能力强，自己一般很少感觉到压力，然后一般就跑步，游泳锻炼身体

1. 项目中出现分歧怎么解决，如何沟通

答案：大家平时项目过程中出现分歧是正常的，主要分歧可能就出现在因为大家对技术的认知或者对项目的理解不一样，但是大家的目的是一致的，都是为了尽快完成这个项目，推进项目进展，大家可以坐在一起商讨一下，谁的想法好，或者更适合，实在不行，那就挨个试一下，看结果说话

1. 家庭对自己的影响

答案：家庭和睦，父母在学习生活上给了自己很大的支持，非常感谢

1. 评价自己的优缺点，导师如何评价你

导师：责任心强，做项目认真负责，耐心，踏实肯干

1. 介绍一下参加的数学建模竞赛

答案：当时我们做的题目是设计高速公路上收费站的面积并保证公路通畅且保证收费时间，我们用的模型是元胞自动机。

1. 工作规划

答案：我跟我男朋友想长期在深圳发展，然后我呢也想在顺丰长期发展，我觉得顺丰是国内最好最大的物流公司，有着海量的货物的吞吐量，是一个非常好的技术平台，在这里能够技得到锻炼，实现我的技术追求。同时给公司带来一点技术上的奉献，然后通过几年的努力，希望自己能够成为独当一面的人，独自负责一项业务或者带领一个小团队

1. 为什么读研

答案：现在本科生毕业生越来越多，就业竞争越来越强，本科毕业觉得自己还不是很强，还想进一步充实一下自己，然后本科是吉大，虽然吉大整体水平很强，但是在通信方面还是没有西电强，想再进一步

1. 怎么体现出你的优点

答案：项目管理能力强，项目紧张并且并行的时候时间分配的好。学习能力强，适应的快

1. 项目中的技术难点

答案：

1. 怎么评价自己
2. 兴趣爱好
3. 学过高并发吗？高并发容易出现什么问题？怎么解决？
4. 为什么专业是通信走java开发

答案：虽然我学的是通信，但是我非常喜欢做这种有界面性的东西，就是觉得自己做出来一个东西也挺厉害的，可以算是成就感推动着继续做java开发。

1. 怎么看待顺丰从传统的物流行业衍生至顺丰科技

答案：传统的物流的收件，分拣，派件，仓储，中转等，每一步都很大的人力，物力，随着物流量的海量增加，纯靠人工是不可能完成的，因此构建更加现代化，智能化，数字化的平台是一件顺理成章的事情。其次，作为一家公司而言，在主要的业务的基础上，开拓一下其他的业务是非常重要的，况且在当今互联网时代，如果仅仅以物流业务继续发展的话，就不合时宜。

1. 怎么看待BAT和顺丰科技

答案：BAT和顺丰科技没有直接的竞争关系，大家其实是通过技术上的互相授权来合作的，阿里的菜鸟跟顺丰是竞争关系的，但是顺丰相对于菜鸟来说有这更好的体验，更看好顺丰的长远发展。

1. 优缺点

答案：优点：现在在学校老师交给我的任务，我一般都能如期完成，具体的话，我会先看这个任务时间急不，如果非常着急的话，我会问一些有相关经历的同学或者朋友，看看他们能不能给我一些建议，让我能快速上路，如期完成项目；如果不着急的话，我会自己查阅相关文献，然后自己动手实现网上的一些相同的例子，如果这种方法可行，那么我就根据自己任务的需求做出相应的修改；缺点：自己在完成项目的时候，有的时候会陷在某些模块的细节里面，导致在这部分模块上花费太多时间，然后需要加班把这一部分工作给补上，最后虽然整体的项目虽然能够完成，但是时间的分配上还不够合理

1. 意向的城市？为什么意向这个城市

答案：深圳这个城市非常有活力，机会多，年轻人就应该要奋斗，挑战自己，同时深圳也是我国改革科技的前沿，非常看好深圳未来的发展，所以就想在这个城市闯出一片属于我的天地。

1. 工作内容意向？为什么？
2. 如何学习后台开发的

答案：看大牛视频，博客，看书

1. 跟实习的人比我的优点？

答案：虽然我没有实习，但是我学习速度快，这么多项目也证明了我的学习能力，就算没有去实习，但是我相信我进了公司之后我能很快的适应，了解公司的技术栈，肯定不比实习过的人差

1. 体现个人解决问题的能力的例子？

答案：互联网+项目，前后端搭建，一个人基本独立完成项目，按期交付

反问:

1. 顺丰的岗位具体是如何安排的，岗位是不变的，部门进去后分配
2. 顺丰这边招聘应届生的标准是什么？希望招进去什么样的人？或者说我应该往哪方面努力
3. 顺丰这边的业务，除了我们接触到的C端，能讲讲B端的业务吗