## MySQL

### MySQL 可以分为 Server 层和存储引擎两部分。

Server 层包括：连接器、查询缓存、分析器、优化器、执行器等，涵盖了 MySQL 的大多数核心服务功能，以及所有的内置函数（如：日期、时间、数学和加密函数等），所有跨存储引擎的功能都在这一层实现，比如：存储过程、触发器、视图等等。

存储引擎层负责：数据的存储和提取。其架构是插件式的，支持 InnoDB、MyISAM 等多个存储引擎。从 MySQL5.5.5 版本开始默认的是InnoDB，但是在建表时可以通过 engine = MyISAM 来指定存储引擎。不同存储引擎的表数据存取方式不同，支持的功能也不同。

从上图中可以看出，不同的存储引擎共用一个 Server 层，也就是从连接器到执行器的部分

### 三大范式

第一范式：强调的是列的原子性，即数据库表的每一列都是不可分割的原子数据项；

第二范式：要求实体的属性完全依赖于主关键字。所谓完全依赖是指不能存在仅依赖主关键字一部分的属性；

第三范式：任何非主属性不依赖于其它非主属性。

### char和varchar

char(n) ：固定长度类型， char 优点：效率高；缺点：占用空间；适用场景：存储密码的 md5 值，primary key，固定长度的，使用 char 非常合适。

varchar(n) ：可变长度，存储的值是每个值占用的字节再加上一个用来记录其长度的字节的长度。所以，从空间上考虑 varchar 比较合适；从效率上考虑 char 比较合适，二者使用需要权衡。varchar(10) 中 10 的涵义最多存放 10 个字符，varchar(10) 和 varchar(20) 存储 hello 所占空间一样，但后者在排序时会消耗更多内存，因为 order by col 采用 fixed\_length 计算 col 长度。

### 谈谈对索引的理解

索引的出现是为了提高数据的查询效率，就像书的目录一样。但是创建索引和维护索引需要耗费时间和占用物理空间，当对表进行增、删、改、的时候索引也要动态维护，这样就降低了数据的维护速度。

建立索引的原则：

在最频繁使用的、用以缩小查询范围的字段上建立索引；

在频繁使用的、需要排序的字段上建立索引。

不适合建立索引的情况：

对于查询中很少涉及的列或者重复值比较多的列，不宜建立索引；

对于一些特殊的数据类型，不宜建立索引，比如：文本字段（text）等。

在MySQL中使用较多的索引有 Hash 索引、B+树索引等。而我们经常使用的 InnoDB 存储引擎的默认索引实现为 B+ 树索引。

### B+树的理解

B+ 树是基于 B 树和叶子节点顺序访问指针进行实现，它具有 B 树的平衡性，并且通过顺序访问指针来提高区间查询的性能。

在 B+ 树中，一个节点中的 key 从左到右非递减排列，如果某个指针的左右相邻 key 分别是 key i 和 key i+1，且不为 null，则该指针指向节点的所有 key 大于等于 key i 且小于等于 key i+1。

进行查找操作时，首先在根节点进行二分查找，找到一个 key 所在的指针，然后递归地在指针所指向的节点进行查找。直到查找到叶子节点，然后在叶子节点上进行二分查找，找出 key 所对应的 data。

插入、删除操作会破坏平衡树的平衡性，因此在插入删除操作之后，需要对树进行一个分裂、合并、旋转等操作来维护平衡性。

用 B+ 树不用 B 树考虑的是 IO 对性能的影响，B 树的每个节点都存储数据，而 B+ 树只有叶子节点才存储数据，所以查找相同数据量的情况下，B 树的高度更高，IO 更频繁。数据库索引是存储在磁盘上的，当数据量大时，就不能把整个索引全部加载到内存了，只能逐一加载每一个磁盘页（对应索引树的节点）。

哈希索引能以 O(1) 时间进行查找，但是失去了有序性。无法用于排序与分组、只支持精确查找，无法用于部分查找和范围查找。

InnoDB 存储引擎有一个特殊的功能叫“自适应哈希索引”，当某个索引值被使用的非常频繁时，会在 B+ 树索引之上再创建一个哈希索引，这样就让 B+Tree 索引具有哈希索引的一些优点，比如：快速的哈希查找。

### 索引的分类

数据结构：树索引 (O(log(n)))Hash 索引

从物理存储角度：聚集索引（clustered index）非聚集索引（non-clustered index）

从逻辑角度 普通索引 唯一索引 主键索引 联合索引 全文索引

### 怎么知道创建的索引有没有被使用到？或者说怎么才可以知道这条语句运行很慢的原因?

使用 Explain 命令来查看语句的执行计划，MySQL 在执行某个语句之前，会将该语句过一遍查询优化器，之后会拿到对语句的分析，也就是执行计划，其中包含了许多信息。可以通过其中和索引有关的信息来分析是否命中了索引，例如：possilbe\_key、key、key\_len 等字段，分别说明了此语句可能会使用的索引、实际使用的索引以及使用的索引长度。

### 什么情况下索引会失效？即查询不走索引？

1、索引列参与表达式计算：

SELECT 'sname' FROM 'stu' WHERE 'age' + 10 = 30;

Mysql

2、 函数运算：

SELECT 'sname' FROM 'stu' WHERE LEFT('date',4) < 1990;

Mysql

3、%词语%–模糊查询：

SELECT \* FROM 'manong' WHERE `uname` LIKE '码农%' -- 走索引

SELECT \* FROM 'manong' WHERE `uname` LIKE '%码农%' -- 不走索引

Mysql

4、 字符串与数字比较不走索引：

“`mysql

CREATE TABLE 'a' ('a' char(10));

EXPLAIN SELECT \* FROM 'a' WHERE 'a'="1" — 走索引

EXPLAIN SELECT \* FROM 'a'WHERE 'a'=1 — 不走索引，同样也是使用了函数运算

“`

5、 查询条件中有 or ，即使其中有条件带索引也不会使用。换言之，就是要求使用的所有字段，都必须建立索引：

select \* from dept where dname='xxx' or loc='xx' or deptno = 45;

Mysql

6、正则表达式不使用索引。

7、 MySQL 内部优化器会对 SQL 语句进行优化，如果优化器估计使用全表扫描要比使用索引快，则不使用索引。

### 查询性能优化的方法

减少请求的数据量

只返回必要的列：最好不要使用 SELECT \* 语句。

只返回必要的行：使用 LIMIT 语句来限制返回的数据。

缓存重复查询的数据：使用缓存可以避免在数据库中进行查询，特别在要查询的数据经常被重复查询时，缓存带来的查询性能提升将会是非常明显的。

减少服务器端扫描的行数

最有效的方式是使用索引来覆盖查询

### 请你描述下事务的特性ACID

原子性：事务是最小的执行单位，不允许分割。事务的原子性确保动作要么全部完成，要么完全不起作用；

一致性：执行事务前后，数据库从一个一致性状态转换到另一个一致性状态。

隔离性：并发访问数据库时，一个用户的事物不被其他事务所干扰，各并发事务之间数据库是独立的；

持久性：一个事务被提交之后。它对数据库中数据的改变是持久的，即使数据库 发生故障也不应该对其有任何影响。

READ\_UNCOMMITTED（未提交读）: 最低的隔离级别，允许读取尚未提交的数据变更，可能会导致脏读、幻读或不可重复读；

READ\_COMMITTED（提交读）: 允许读取并发事务已经提交的数据，可以阻止脏读，但是幻读或不可重复读仍有可能发生；

REPEATABLE\_READ（可重复读）: 对同一字段的多次读取结果都是一致的，除非数据是被本身事务自己所修改，可以阻止脏读和不可重复读，但幻读仍有可能发生；

SERIALIZABLE（串行化）: 最高的隔离级别，完全服从 ACID 的隔离级别。所有的事务依次逐个执行，这样事务之间就完全不可能产生干扰，也就是说，该级别可以防止脏读、不可重复读以及幻读。但是这将严重影响程序的性能。通常情况下也不会用到该级别。

### 解释下什么叫脏读、不可重复读和幻读？

脏读：没提交就能读取

表示一个事务能够读取另一个事务中还未提交的数据。比如：某个事务尝试插入记录 A，此时该事务还未提交，然后另一个事务尝试读取到了记录 A。

不可重复读 ：两次读取之间有修改

是指在一个事务内，多次读同一数据。在这个事务还没有结束时，另外一个事务也访问该同一数据。那么，在第一个事务中的两 次读数据之间，由于第二个事务的修改，那么第一个事务两次读到的的数据可能是不一样的。这样就发生了在一个事务内两次读到的数据是不一样的，因此称为是不可重复读。例如，一个编辑人员两次读取同一文档，但在两次读取之间，作者重写了该文档。当编辑人员第二次读取文档时，文档已更改。原始读取不可重复。如果只有在作者全部完成编写后编辑人员才可以读取文档，则可以避免该问题不可重复读的重点是修改 :同样的条件 , 你读取过的数据 , 再次读取出来发现值不一样了

幻读：

指同一个事务内多次查询返回的结果集不一样。比如同一个事务 A 第一次查询时候有 n 条记录，但是第二次同等条件下查询却有 n+1 条记录，这就好像产生了幻觉。发生幻读的原因也是另外一个事务新增或者删除或者修改了第一个事务结果集里面的数据，同一个记录的数据内容被修改了，所有数据行的记录就变多或者变少了。幻读的重点在于新增或者删除：同样的条件 , 第 1 次和第 2 次读出来的记录数不一样

1. 数据库的隔离级别

首先用通俗的语言介绍以下事务（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）的特性（ACID）：

原子性（Atomicity）：一个事务中的操作，要么全部成功，要么全部失败，如果失败，就回滚到事务开始前的状态。

一致性（Consistency）：事务必须使数据库从一个一致性状态变换到另一个一致性状态，也就是说一个事务执行之前和执行之后都必须处于一致性状态。那转账举栗子，A账户和B账户之间相互转账，无论如何操作，A、B账户的总金额都必须是不变的。

隔离性（Isolation）：当多个用户并发的访问数据库时，如果操作同一张表，数据库则为每一个用户都开启一个事务，且事务之间互不干扰，也就是说事务之间的并发是隔离的。再举个栗子，现有两个并发的事务T1和T2，T1要么在T2开始前执行，要么在T2结束后执行，如果T1先执行，那T2就在T1结束后在执行。关于数据的隔离性级别，将在后文讲到。

持久性（Durability）：如果事务一旦被提交，数据库中数据的改变就是永久性的，即使断电或者宕机的情况下，也不会丢失提交的事务操作。

事务的隔离性：多个用户的并发事务访问同一个数据库时，一个用户的事务不应该被其他用户的事务干扰，多个并发事务之间要相互隔离。

如果不考虑隔离性，会发生什么事呢？

1.脏读：一个事务在处理数据的过程中，读取到另一个未提交事务的数据。

2.不可重复读：对于数据库中的某个数据，一个事务范围内的多次查询却返回了不 同的结果，这是由于在查询过程中，数据被另外一个事务修改并提交了。

不可重复读和脏读的区别：脏读读取到的是一个未提交的数据，不可重复读读取到的是前一个事务提交的数据。

而不可重复读在一些情况也并不影响数据的正确性，比如需要多次查询的数据也是要以最后一次查询到的数据为主。

3.幻读：事务非独立执行时发生的一种现象。例如事务T1对一个表中所有的行的某个数据项做了从“1”修改为“2”的操作，这时事务T2又对这个表中插入了一行数据项，而这个数据项的数值还是为“1”并且提交给数据库。而操作事务T1的用户如果再查看刚刚修改的数据，会发现还有一行没有修改，其实这行是从事务T2中添加的，就好像产生幻觉一样，这就是发生了幻读。

幻读和不可重复读都是读取了另一条已经提交的事务（这点就脏读不同），所不同的是不可重复读查询的都是同一个数据项，而幻读针对的是一批数据整体（比如数据的个数）。

不可重复读和幻读是初学者不易分清的概念，我也是看了详细的解读才明白的，总的来说，解决不可重复读的方法是锁行，解决幻读的方式是锁表。

四种隔离级别解决了上述问题

1.读未提交（Read uncommitted）：这种事务隔离级别下，select语句不加锁。此时，可能读取到不一致的数据，即“读脏 ”。这是并发最高，一致性最差的隔离级别。

2.读已提交（Read committed）：可避免脏读的发生。这是一致性最好的，但并发性最差的隔离级别。在互联网大数据量，高并发量的场景下，几乎 不会使用 上述两种隔离级别。

3.可重复读（Repeatable read）：MySql默认隔离级别。可避免脏读 、不可重复读 的发生。

4.串行化（Serializable ）：可避免脏读、不可重复读、幻读的发生。

以上四种隔离级别最高的是 Serializable 级别，最低的是 Read uncommitted 级别，当然级别越高，执行效率就越低。像 Serializable 这样的级别，就是以锁表的方式(类似于Java多线程中的锁)使得其他的线程只能在锁外等待，所以平时选用何种隔离级别应该根据实际情况。在MySQL数据库中默认的隔离级别为Repeatable read (可重复读) 。

在MySQL数据库中，支持上面四种隔离级别，默认的为Repeatable read (可重复读) ；而在 Oracle数据库 中，只支持Serializable (串行化) 级别和 Read committed (读已提交) 这两种级别，其中默认的为 Read committed（读已提交） 级别。

2. 存储引擎

不同的存储引擎提供不同的存储机制、索引技巧、锁定水平等功能，使用不同的存储引擎，还可以获得特定的功能。现在许多不同的数据库管理系统都支持多种不同的数据引擎。

因为在关系数据库中数据的存储是以表的形式存储的，所以存储引擎也可以称为表类型(Table Type，即存储和操作此表的类型)。

1.Mysql中的myisam与innodb的区别

1. InooDB支持事务，而MyISAM不支持；

2. InnoDB支持行级锁，而MyISAM支持表级锁；

3. InnoDB支持MVCC，而MyISAM不支持；

4. InnoDB支持外键，而MyISAM不支持；

5. //InnoDB不支持全文索引，而MyISAM支持；

6. InnoDB不能通过直接拷贝表文件的方法拷贝表到另外一台机器， myisam 支持；

7. InnoDB表支持多种行格式，myisam 不支持；

8. InnoDB是索引组织表，myisam 是堆表；

2.myisam与innodb select count(\*)哪个更快

myisam更快，因为myisam内部维护了一个计数器，可以直接调取。

3.行级锁和表级锁

相对其他数据库而言，MySQL的锁机制比较简单，其最显著的特点是不同的存储引擎支持不同的锁机制。比如，MyISAM和MEMORY存储引擎采用的是表级锁（table-level locking）；InnoDB存储引擎既支持行级锁（ row-level locking），也支持表级锁，但默认情况下是采用行级锁。

表级锁： 开销小，加锁快；不会出现死锁(因为MyISAM会一次性获得SQL所需的全部锁)；锁定粒度大，发生锁冲突的概率最高,并发度最低。

行级锁： 开销大，加锁慢；会出现死锁；锁定粒度最小，发生锁冲突的概率最低,并发度也最高。

考虑上述特点，表级锁使用与并发性不高，以查询为主，少量更新的应用，比如小型的web应用；而行级锁适用于高并发环境下，对事务完整性要求较高的系统，如在线事务处理系统。

3. 索引

1. 什么是索引

索引(Index)是帮助MySQL高效获取数据的数据结构。我们可以简单理解为：快速查找排好序的一种数据结构。（索引其实是一种数据结构，能够帮助我们快速的检索数据库中的数据）

索引具体采用哪种数据结构：

Mysql索引主要有两种结构：B+Tree索引和Hash索引。我们平常所说的索引，如果没有特别指明，一般都是指B树结构组织的索引(B+Tree索引)。

为什么加索引能优化慢查询？

因为索引其实就是一种优化查询的数据结构，比如Mysql中的索引是用B+树实现的，而B+树就是一种数据结构，可以优化查询速度，可以利用索引快速查找数据，所以能优化查询。

2. MySQL主要的索引类型

普通索引：是最基本的索引，它没有任何限制；

唯一索引：索引列的值必须唯一，但允许有空值。如果是组合索引，则列值的组合必须唯一；

主键索引：是一种特殊的唯一索引，一个表只能有一个主键，不允许有空值；

组合索引：指多个字段上创建的索引，只有在查询条件中使用了创建索引时的第一个字段，索引才会被使用。使用组合索引时遵循最左前缀集合；多列值组成一个索引，专门用于组合搜索，其效率大于索引合并

全文索引：主要用来查找文本中的关键字，而不是直接与索引中的值相比较，mysql中MyISAM支持全文索引而InnoDB不支持；

3. 数据库索引有哪些，优缺点？

hash索引和B+树索引

B+树索引：数据有序，能够进行范围查询

哈希表适合等值查询（因为哈希索引底层是哈希表，哈希表是一种key-value存储数据的结构）

哈希索引没办法利用索引完成排序 ,因此不能进行范围查询

哈希索引不支持多列联合索引的最左匹配规则

如果有大量重复键值的情况下，哈希索引的效率会很低，因为存在哈希碰撞问题

4. 为什么不用二叉查找树作为数据库索引？

二叉查找树，查找到指定数据，效率其实很高logn。查询速度和比较次数都是较小的。

实际上，查询索引操作最耗资源的不在内存中，而是磁盘IO。数据库索引文件有可能很大，关系型数据存储了上亿条数据，索引文件大则上G，不可能全部放入内存中，而是需要的时候换入内存，方式是磁盘页。一般来说树的一个节点就是一个磁盘页。如果使用二叉查找树，那么每个节点存储一个元素，查找到指定元素，需要进行大量的磁盘IO，效率很低。而B树解决了这个问题，通过单一节点包含多个data，大大降低了树的高度，大大减少了磁盘IO次数

8. 索引的缺点

1. 索引要占用磁盘空间。通常情况下，这个问题不是很突出。但是，如果你创建每一种可能列组合的索引，索引文件体积的增长速度将远远超过数据文件。如果你有一个很大的表，索引文件的大小可能达到操作系统允许的最大文件限制。

2.对于需要写入数据的操作，比如DELETE、UPDATE以及INSERT操作，索引会降低它们的速度。这是因为MySQL不仅要把改动数据写入数据文件，而且它还要把这些改动写入索引文件。

9.哪种情况需要创建索引

　　1.主键自动建立唯一索引

　　2.频繁作为查询条件的字段应该创建索引

　　3.查询中与其他表关联的字段,外键关系建立索引

　　5.Where条件里用得到的字段适合创建索引

　　6.单键/组合索引的选择问题,在高并发下倾向创建组合索引

　　7.查询中排序的字段,排序字段若通过索引去访问将大大提高排序速度

　　8.查询中统计或者分组字段

10哪种情况不需要建索引

　　1.表记录太少(一般生产环境下,三百万条记录性能就可能开始下降,官方说的是五百万到八百万)

　　2.经常增删改的表

　　3.某个数据列的值包含许多重复的内容

12一个B+树的节点中到底存储多少个元素合适呢？

　　B+树中一个节点为一页或页的倍数最为合适。因为如果一个节点的大小小于1页，那么读取这个节点的时候其实也会读出1页，会造成资源的浪费；如果一个节点的大小大于1页，比如1.2页，那么读取这个节点的时候会读出2页，也会造成资源的浪费；所以为了不造成资源的浪费，最后把一个节点的大小控制在1页、2页、3页、4页等倍数页大小最为合适！

11.哪些数据结构能提高查询速度？MySQL为何选择使用B+树？

哈希表、完全平衡二叉搜索树、B树、B+树等等；

哈希表的特点就是可以快速的精确查询，但是不支持范围查询

完全平衡二叉搜索树是有序的，简单的说就是 "左边的小于右边的"，假如我们现在来查找 '周瑜' ，需要查找2次（第一次操作，第二次周瑜），比哈希表要多一次。而且由于完全平衡二叉搜索树是有序的，所以支持范围查找。

B树表示的要比完全平衡二叉搜索树要 "矮",原因在于B树中的一个节点可以存储多个元素！

我们可以发现同样的元素，B+树的表示要比B树要 "胖"，原因在于B+树中的非叶子节点会冗余一份在叶子节点中，并且叶子节点之间用指针相连！

13.MySQL中B+树的一个节点大小为多大？

　　一页，这里说的 "页" 是MySQL自定义的单位（其实和操作系统类似），MySQL的Innodb引擎中一页的默认大小是16K（如果操作系统中一页大小是4K，那么MySQL中1页 = 操作系统中的4页），这样存取数据的时候都是一页一页的获取索引文件中节点数据的！

14一系列问题

1. 系统中某个sql查询比较慢，怎么办？

看条件字段是否加索引，若没加可以加索引

2. mysql存储索引用什么数据结构？

B+树

3. B+树查询时间大约是多少？

和树的高度有关，log(n)

4. 如果用hash存储索引，查询时间？

O(1)

5. hash比B+快，为啥不用hash

如果只选一条 hash快，但实际数据库经常选多条。时候由于B+树索引有序，并且又有链表相连，它的查询效率比hash就快很多了。且数据库中的索引一般是在磁盘上，数据量大的情况可能无法一次装入内存，B+树的设计可以允许数据分批加载，同时树的高度较低，提高查找效率。

6. 树你知道哪些？

7. 前中序遍历，二叉树，二叉搜索树，平衡二叉树，更高级一点红黑树，B tree，B+树

8. 二叉搜索树极端退化成链表，所以用AVL树，红黑树就是平衡树的一种，复杂的定义和规则最后都是为了保证树的平衡性。

9. 为什么要保证树的平衡性

树的查找性能取决于树的高度，让树尽可能平衡，就是为了降低树的高度。

10 .B tree

是一种多路搜索树，每个节点可以拥有多于两个孩子节点，M路的B树最多能拥有M个孩子节点

11.为什么设计成多路

为了进一步降低树的高度

12. 可以设计成无限多路吗

退化成有序数组了

13. B树一般用在文件系统的索引

14. 为什么文件系统的索引用B树而不用红黑树或有序数组

文件系统和数据库的索引存在硬盘上，并且如果数据量大的话，不一定能一次性加载到内存中。

15. 如果一棵树无法一次性加载进内存，怎么查找？

每次加载树的一个节点，然后一步步向下找

如果在内存中，红黑树比B树效率更高，但涉及到硬盘操作，B树更优。

16. B+树在B树基础上进行改进，数据都在叶子结点，同时叶子结点之间还加了指针形成链表。

17. B+树为什么这么设计

Select不一定只选一条，很多时候会选多条。这时B树做局部的中序遍历，可能要跨层访问。而B+树由于所有数据都在叶子结点，不用跨层，同时由于有链表结构，只需要找到首尾，通过链表就能把所有数据取出来了。

16为什么B+树比B树更适合做数据库索引？

B+树的磁盘读写代价更低。由于B+树内部的结点并没有指向关键字的指针，所以其内部结点相对B树更小，如果把同一内部结点的关键字存放在同一块磁盘中，那么盘块所能容纳的关键字数量也就较多，一次性读入内存需要查找的关键字也越多，相对IO读写次数也就越低。

B+树的查询效率更加稳定。由于非叶子结点并不是指向文件内容的结点，而只是叶子结点中关键字的索引，所以任何关键字的查找必须走一条从根结点到叶子结点的路，所有关键字查询的路径长度相同，导致每一个数据的查询效率一样。

B+树在查询时，只需要扫一遍叶子结点就可以了，因为分支结点都为索引，没有存文件内容。而B树分支结点中还存在数据，我们需要找到具体的数据，需要进行一次中序遍历按序扫描，所以B+树更适合在区间查询，因此，B+树更适合用于数据库索引。

18. 聚簇索引和非聚簇索引

1.首先，我们要认识到聚簇索引和非聚簇索引的划分依据是什么 ？

答案就是 InnoDB 会使用聚簇索索引来保存数据，而非聚簇索引的目的仅仅是加快查询速度

在第一点认知基础上，我们就可以知道

聚簇索引是唯一的，一个 InnoDB 表只有一个聚簇索引，而且一定会有一个聚簇索引，如果不存在，Innodb 存储引擎会自动添加一个

非聚簇所以可以有多个，而且只能由用户自己添加，InnoDB 默认并不会创建任何非聚簇索引。

非聚簇索引中一定包含了聚簇索引的列值，但反过来却不存在。

因此，使用非聚簇索引查询数据一定会用到聚簇索引，但反过来却不存在。

19. 聚集索引和非聚集索引

聚集索引和非聚集索引的根本区别是表记录的排列顺序和与索引的排列顺序是否一致。

聚集索引表记录的排列顺序与索引的排列顺序一致

• 优点是查询速度快，因为一旦具有第一个索引值的纪录被找到，具有连续索引值的记录也一定物理的紧跟其后。

• 缺点是对表进行修改速度较慢，这是为了保持表中的记录的物理顺序与索引的顺序一致，而把记录插入到数据页的相应位置，必须在数据页中进行数据重排， 降低了执行速度。建议使用聚集索引的场合为：

a. 此列包含有限数目的不同值；

b. 查询的结果返回一个区间的值；

c. 查询的结果返回某值相同的大量结果集。

非聚集索引指定了表中记录的逻辑顺序，但记录的物理顺序和索引的顺序不一致，聚集索引和非聚集索引都采用了B+树的结构，但非聚集索引的叶子层并不与实际的数据页相重叠，而采用叶子层包含一个指向表中的记录在数据页中的指针的方式。

非聚集索引比聚集索引层次多，添加记录不会引起数据顺序的重组。

建议使用非聚集索引的场合为：

a. 此列包含了大量数目不同的值；

b. 查询的结束返回的是少量的结果集；

c. order by 子句中使用了该列。

20. 什么情况索引会失效：

1. 条件中有or

　　注意：要想使用or，又想让索引生效，只能将or条件中的每个列都加上索引

　　2.对于多列索引，不是使用的第一部分，则不会使用索引

　　3.like查询是以%开头

　　4.如果列类型是字符串，那一定要在条件中将数据使用引号引用起来,否则不使用索引

　　5.如果mysql估计使用全表扫描要比使用索引快,则不使用索引

此外，查看索引的使用情况

show status like ‘Handler\_read%’;

大家可以注意：

handler\_read\_key:这个值越高越好，越高表示使用索引查询到的次数

handler\_read\_rnd\_next:这个值越高，说明查询低效

21 与红黑树的比较

4.说一说你能想到的sql语句优化，至少五种

1）避免select \*，将需要查找的字段列出来；

2）使用连接（join）来代替子查询；

3）拆分大的delete或insert语句；

4）使用limit对查询结果的记录进行限定；

5）用 exists 代替 in 是一个好的选择；

6）用Where子句替换HAVING 子句 因为HAVING 只会在检索出所有记录之后才对结果集进行过滤；

7）不要在 where 子句中的“=”左边进行函数、算术运算或其他表达式运算，否则系统将可能无法正确使用索引尽量避免在where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描；

8）尽量避免在 where 子句中使用 or 来连接条件，否则将导致引擎放弃使用索引而进行全表扫描；

9）尽量避免在 where 子句中使用!=或<>操作符，否则将引擎放弃使用索引而进行全表扫描；

## 计算机基础