**数据库索引：**

**1.请简述常用的索引有哪些种类?**

1. 普通索引: 即针对数据库表创建索引
2. 唯一索引: 与普通索引类似，不同的就是：MySQL 数据库索引列的值必须唯一，但**允许有空值**
3. 主键索引: 它是一种特殊的唯一索引，**不允许有空值**。一般是在建表的时候同时创建主键索引
4. 组合索引(联合索引): 为了进一步榨取 MySQL 的效率，就要考虑建立组合索引。即将数据库表中的**多个字段联合**起来作为一个组合索引。

**2. MySQL数据库中索引的工作机制是什么？**

数据库索引，是数据库管理系统中一个**排序的数据结构**，以协助快速查询、更新数据库表中数据。索引的实现通常使用 B 树及其变种 B+树

**3. 唯一和主键索引的区别，唯一与主与聚簇、非聚簇的关系？**

唯一索引 与 主键索引:

1. 唯一索引是在表上一个或者多个字段组合建立的索引，这个（或这几个）字段的值组合起来在表中不可以重复。一张表可以建立任意多个唯一索引，但一般只建立一个。
2. 主键是一种特殊的唯一索引，区别在于，唯一索引列允许null值，而主键列不允许为null值。一张表最多建立一个主键，也可以不建立主键。

聚簇索引、非聚簇索引、主键

**聚簇索引**和**非聚簇索引**的区别的：

（MySQL的InnoDB索引数据结构是B+树，主键索引叶子节点的值存储的就是MySQL的数据行，普通索引的叶子节点的值存储的是主键值，这是了解聚簇索引和非聚簇索引的前提）

1. **聚簇索引**：**找到了索引就找到了需要的数据**，那么这个索引就是聚簇索引，所以主键就是聚簇索引，修改聚簇索引其实就是修改主键。
2. **非聚簇索引**：**索引的存储和数据的存储是分离的**，也就是说找到了索引但没找到数据，需要根据索引上的值(主键)再次回表查询,非聚簇索引也叫做**辅助索引**。

* **聚簇**索引的**叶子节点**就是**数据节点**，而**非聚簇**索引的**叶子节点**仍然是**索引节点**，只不过**有指向对应数据块的指针**。
* 聚簇索引的顺序，就是数据在硬盘上的**物理顺序**。一般情况下主键就是默认的聚簇索引。
* **一张表只允许存在一个聚簇索引**，因为真实数据的物理顺序只能有一种。如果一张表上还没有聚簇索引，为它**新创建聚簇索引时，就需要对已有数据重新进行排序**，所以**对表进行修改速度较慢是聚簇索引的缺点**，对于经常更新的列不宜建立聚簇索引。
* **聚簇索引性能最好**，因为一旦具有第一个索引值的记录被找到，具有连续索引值的记录也一定物理地紧跟其后。一张表只能有一个聚簇索引，所以非常珍贵，必须慎重设置，一般要根据这个表最常用的SQL查询方式选择某个（或多个）字段作为聚簇索引（或复合聚簇索引）。
* **聚簇索引默认是主键**，如果表中没有定义主键，InnoDB[1]会选择一个唯一的非空索引代替（“唯一的非空索引”是指列不能出现null值的唯一索引，跟主键性质一样）。如果没有这样的索引，InnoDB会隐式地定义一个主键来作为聚簇索引。

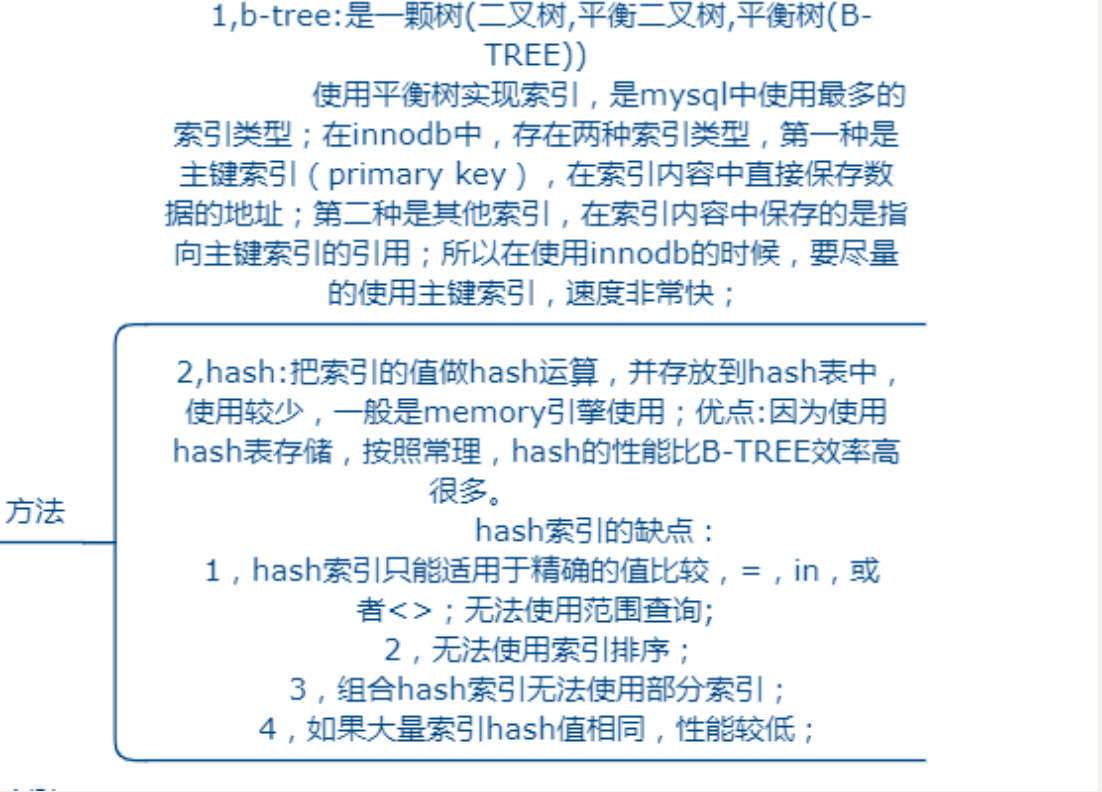
聚簇索引 与 唯一索引：

严格来说，**聚簇索引不一定是唯一索引**，聚簇索引的索引值并不要求是唯一的，唯一聚簇索引才是！在一个有聚簇索引的列上是可以插入两个或多个相同值的，这些相同值在硬盘上的物理排序与聚簇索引的排序相同，仅此而已。

（首先指出一个误区，**主键并不一定是聚集索引**，只是在SQL SERVER中，未明确指出的情况下，默认将主键定义为聚集，而ORACLE中则默认是非聚集，因为SQL SERVER中的ROWID未开放使用。）

1. **InnoDB存储引擎有哪些常见的索引？**

B+树索引、全文索引和哈希索引。



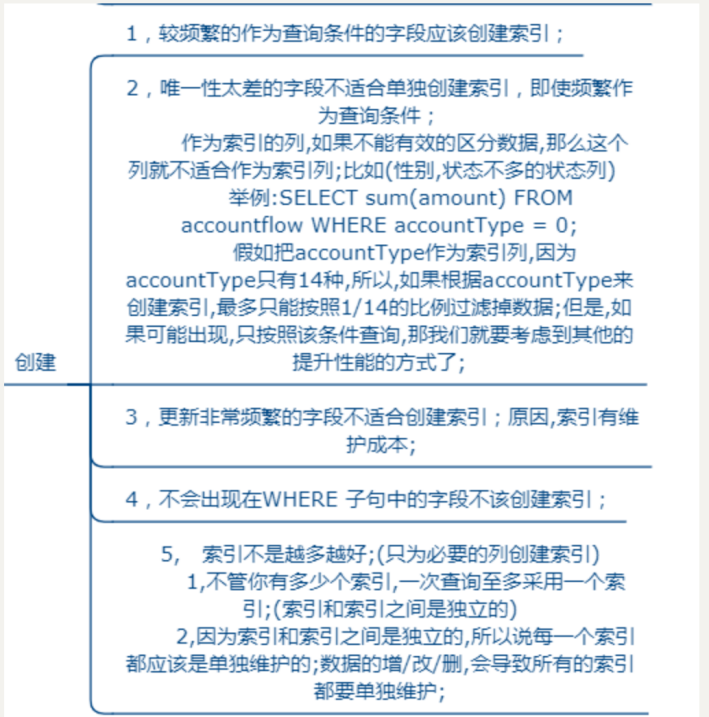
1. **B+树与B树的区别？**
2. B+树内节点不存储数据，所有 data 存储在叶节点导致查询时间复杂度固定为 log n。而B-树查询时间复杂度不固定，与 key 在树中的位置有关，最好为O(1)。
3. B+树叶节点两两相连可大大增加区间访问性，可使用在范围查询等，而B-树每个节点 key 和 data 在一起，则无法区间查找。
4. B+树更适合外部存储。由于内节点无 data 域，每个节点能索引的范围更大更精确。
5. **为什么MySQL数据库使用B+树不使用B树？**

当存储同数量级的数据的时候，B+树的高度比B树的高度小，这样的话进程IO操作的次数就少，效果就高。因为B+树的所有非叶子节点只存索引，数据存在叶子节点，一般3层的树高度，即可存千万级别的数据，而B数不行。(具体的计算可以到网上去看看

1. **Hash 索引和 B+ 树索引区别是什么？设计索引如何抉择？**

* B+ 树可以进行范围查询，Hash 索引不能。
* B+ 树支持联合索引的最左侧原则，Hash 索引不支持。
* B+ 树支持 order by 排序，Hash 索引不支持。
* Hash 索引在等值查询上比 B+ 树效率更高。
* 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。

1. **如何选择在哪些列上建索引？**
2. 一张表一般都要去建主键，所以主键索引几乎是每张表必备的，这个就不多说了。
3. 选择性高的列，也就是重复度低的列。比如女子学校学生表中的性别列，所有数据的值都是女，这样的列就不适合建索引。比如学生表中的身份证号列，选择性就很高，就适合建索引。
4. 经常用于查询的列（出现在where条件中的列）。不过如果不符合上一条的条件，即便是出现在where条件中也不适合建索引，甚至就不应该出现在where条件中。
5. 多表关联查询时作为关联条件的列。比如学生表中有班级ID的列用于和班级表关联查询时作为关联条件，这个列就适合建索引。
6. \*\*值会频繁变化的列不适合建索引。\*\*因为在数据发生变化时是需要针对索引做一些处理的，所以如果不是有非常必要的原因，不要值会频繁变化的列上建索引，会影响数据更新的性能。反过来也就是说索引要建在值比较固定不变的列上。
7. \*\*一张表上不要建太多的索引。\*\*和上一条的原因类似，如果一张表上的索引太多，会严重影响数据增删改的性能。也会耗费很大的磁盘空间。



1. **什么是最左匹配原则？**

最左前缀匹配原则和联合索引的索引存储结构和检索方式是有关系的。

在组合索引树中，最底层的叶子节点按照第一列a列从左到右递增排列，但是b列和c列是无序的，b列只有在a列值相等的情况下小范围内递增有序，而c列只能在a，b两列相等的情况下小范围内递增有序。

就像上面的查询，B+树会先比较a列来确定下一步应该搜索的方向，往左还是往右。如果a列相同再比较b列。但是如果查询条件没有a列，B+树就不知道第一步应该从哪个节点查起。

可以说创建的idx\_abc(a,b,c)索引，相当于创建了(a)、（a,b）（a,b,c）三个索引。、

组合索引的最左前缀匹配原则：使用组合索引查询时，mysql会一直向右匹配直至遇到范围查询(>、<、between、like)就停止匹配。

**10.覆盖索引**

覆盖索引并不是说是索引结构，覆盖索引是一种很常用的优化手段。因为在使用辅助索引的时候，我们只可以拿到主键值，相当于获取数据还需要再根据主键查询主键索引再获取到数据。但是试想下这么一种情况，在上面abc\_innodb表中的组合索引查询时，如果我只需要abc字段的，那是不是意味着我们查询到组合索引的叶子节点就可以直接返回了，而不需要回表。这种情况就是覆盖索引。

举例子：

假设我们只需要查询商品的名称、价格信息，我们有什么方式来避免回表呢？我们可以建立一个组合索引，即商品编码、名称、价格作为一个组合索引。如果索引中存在这些数据，查询将不会再次检索主键索引，从而避免回表。

从辅助索引中查询得到记录，而不需要通过聚族索引查询获得，MySQL 中将其称为覆盖索引。使用覆盖索引的好处很明显，我们不需要查询出包含整行记录的所有信息，因此可以减少大量的 I/O 操作。

通常在 InnoDB 中，除了查询部分字段可以使用覆盖索引来优化查询性能之外，统计数量也会用到。例如， SELECT COUNT(\*) 时，如果不存在辅助索引，此时会

通过查询聚族索引来统计行数，如果此时正好存在一个辅助索引，则会通过查询辅助索引来统计行数，减少 I/O 操作。