**1、数据库ACID？**

A原子性：事务是一个不可再分割的工作单位，事务中的操作要么都发生，要么都不发生。

C一致性：事务开始之前和事务结束以后，数据库的完整性约束没有被破坏。这是说数据库事务不能破坏关系数据的完整性以及业务逻辑上的一致性。

I隔离性：多个事务并发访问时，事务之间是隔离的，一个事务不应该影响其它事务运行效果。

D持久性：当一个事物提交之后，数据库状态永远的发生了改变。

**1、MySQL 中 InnoDB 支持的四种事务隔离级别名称，以及逐级之间的区别？**

SQL 标准定义的四个隔离级别为：

1、read uncommited ：读到未提交数据

2、read committed：脏读，不可重复读

3、repeatable read：可重读

4、serializable ：串行事物

Mysql默认为第三级隔离级别，sqlserver为第二个隔离级别。

前三种隔离级别为乐观锁，mysql中通过记录版本号来实现，MVCC技术，即多版本并发控制。

在MySQL中可重复读和读已提交都是通过MVCC进行实现的，却别在于可重读是事务启动的时候就生成read view整个事务结束都一直使用这个read view，而在读已提交中则是每执行一条语句就重新生成最新的read view。

第四种隔离级别为悲观锁，通过读写锁来实现

**2、mvcc如何实现读已提交和重复读？**

在 RC 隔离级别下，每个 SELECT 语句开始时，都会重新将当前系统中的所有的活跃事务拷贝到一个列表生成 ReadView。二者的区别就在于生成 ReadView 的时间点不同，可重复读是事务之后第一个 SELECT 语句开始，读已提交是事务中每条 SELECT 语句开始。

ReadView 中是当前活跃的事务 ID 列表，称之为 m\_ids，其中最小值为 up\_limit\_id，最大值为 low\_limit\_id，事务 ID 是事务开启时 InnoDB 分配的，其大小决定了事务开启的先后顺序，因此我们可以通过 ID 的大小关系来决定版本记录的可见性，具体判断流程如下：

如果被访问版本的 trx\_id 小于 m\_ids 中的最小值 up\_limit\_id，说明生成该版本的事务在 ReadView 生成前就已经提交了，所以该版本可以被当前事务访问。

如果被访问版本的 trx\_id 大于 m\_ids 列表中的最大值 low\_limit\_id，说明生成该版本的事务在生成 ReadView 后才生成，所以该版本不可以被当前事务访问。需要根据 Undo Log 链找到前一个版本，然后根据该版本的 DB\_TRX\_ID 重新判断可见性。

如果被访问版本的 trx\_id 属性值在 m\_ids 列表中最大值和最小值之间(包含)，那就需要判断一下 trx\_id 的值是不是在 m\_ids 列表中。如果在，说明创建 ReadView 时生成该版本所属事务还是活跃的，因此该版本不可以被访问，需要查找 Undo Log 链得到上一个版本，然后根据该版本的 DB\_TRX\_ID 再从头计算一次可见性；如果不在，说明创建 ReadView 时生成该版本的事务已经被提交，该版本可以被访问。

**2、什么是脏读？不可重复读？幻读？**

脏读：A事务执行过程中，B事务读取了A事务的修改。但是由于某些原因，A事务可能没有完成提交，发生RollBack了操作，则B事务所读取的数据就会是不正确的。

不可重复读：B事务读取了两次数据，在这两次的读取过程中A事务修改了数据，B事务的这两次读取出来的数据不一样。B事务这种读取的结果，即为不可重复读（Nonrepeatable Read）。修改了原数据

幻读：B事务读取了两次数据，在这两次的读取过程中A事务添加了数据，B事务的这两次读取出来的集合不一样。增加了数据

**3、 什么是乐观锁和悲观锁？**

乐观锁的特点先进行业务操作，不到万不得已不去拿锁；即“乐观”的认为拿锁多半是会成功的，因此在进行完业务操作需要实际更新数据的最后一步再去拿一下锁就好；乐观锁在数据库上的实现完全是逻辑的，不需要数据库提供特殊的支持。一般的做法是在需要锁的数据上增加一个版本号，或者时间戳。

悲观锁的特点是先获取锁，再进行业务操作，即“悲观”的认为获取锁是非常有可能失败的，因此要先确保获取锁成功再进行业务操作；通常所说的“一锁二查三更新”即指的是使用悲观锁。

**4、索引的底层实现原理和优化？**

B+树，经过优化的 B+树。

主要是在所有的叶子结点中增加了指向下一个叶子节点的指针，因此 InnoDB 建议为大部分表使用默认自增的主键作为主索引。

**5、索引为什么不用b树？**

前提知识：我们知道从磁盘文件中读取数据是比较耗时的，数据库的select操作的时间，取决于执行磁盘IO的次数，因此尽量减少磁盘IO就可以显著的提升数据的查询速度。

我们应该知道B树和B+树最重要的一个区别就是B+树只有叶节点存放数据，其余节点用来索引，而B-树是每个索引节点都会有Data域。

这就决定了B+树更适合用来存储外部数据，也就是所谓的磁盘数据。

从Mysql（Inoodb）的角度来看，B+树是用来充当索引的，一般来说索引非常大，尤其是关系性数据库这种数据量大的索引能达到亿级别，所以为了减少内存的占用，索引也会被存储在磁盘上。

那么Mysql如何衡量查询效率呢？磁盘IO次数。B-树（B类树）的特点就是每层节点数目非常多，层数很少，目的就是为了就少磁盘IO次数，当查询数据的时候，最好的情况就是很快找到目标索引，然后读取数据，使用B+树就能很好的完成这个目的，但是B-树的每个节点都有data域（指针），这无疑增大了节点大小，说白了增加了磁盘IO次数（磁盘IO一次读出的数据量大小是固定的，单个数据变大，每次读出的就少，IO次数增多，一次IO多耗时啊！），而B+树除了叶子节点其它节点并不存储数据，节点小，磁盘IO次数就少。这是优点之一。

另一个优点是什么，B+树所有的Data域在叶子节点，一般来说都会进行一个优化，就是将所有的叶子节点用指针串起来。这样遍历叶子节点就能获得全部数据，这样就能进行区间访问啦。B+树只要遍历叶子节点就可以实现整棵树的遍历。而且在数据库中基于范围的查询是非常频繁的，而B树不支持这样的操作（或者说效率太低）

**6、简单描述 MySQL 中，索引，主键，唯一索引，联合索引**

**的区别，对数据库的性能有什么影响（从读写两方面）**

索引是一种特殊的文件，它们包含着对数据表里所有记录的引用指针。

索引可以极大的提高数据的查询速度，但是会降低插入、删除、更新表的速度，因为在执行这些写操作时，还要操作索引文件。

唯一索引则是非null值唯一的索引，保证唯一性。

主键，是一种特殊的唯一索引，不能有null值，在一张表中只能定义一个主键索引，主键用于唯一标识一条记录，使用关键字 PRIMARY KEY 来创建。

索引可以覆盖多个数据列，如像 INDEX(columnA, columnB)索引，这就是联合索引。

**7、什么情况下设置了索引但无法使用**

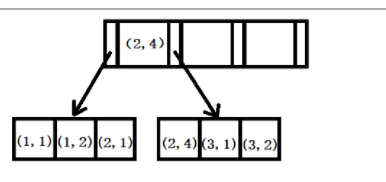
1、以“%”开头的 LIKE 语句，模糊匹配

2、OR 语句前后没有同时使用索引

3、数据类型出现隐式转化（如 varchar 不加单引号的话可能会自动转换为 int 型）

4、对于多列索引，不是使用的第一部分，则不会使用索引

**8、讲讲数据库最左适配原则？**



如图所示，最左适配原则和索引的b+树底层有关，优先按照最左侧的关键字进行排序，除非第一个关键字相同，则按照后面的关键字排序，有点像sql语句里的order by

**9、聚集索引与非聚集索引？**

**正文内容本身就是一种按照一定规则排列的目录称为“聚集索引”。**

**目录纯粹是目录，正文纯粹是正文的排序方式称为“非聚集索引”。**

**每个表只能有一个聚集索引，因为目录只能按照一种方法进行排序。如果不指定，一般情况下主键为默认聚集索引。**

**区别：**

聚集索引一个表只能有一个，而非聚集索引一个表可以存在多个

聚集索引存储记录是物理上连续存在，而非聚集索引是逻辑上的连续，物理存储并不连续

聚集索引:物理存储按照索引排序；聚集索引是一种索引组织形式，索引的键值逻辑顺序决定了表数据行的物理存储顺序。

非聚集索引:物理存储不按照索引排序；非聚集索引则就是普通索引了，仅仅只是对数据列创建相应的索引，不影响整个表的物理存储顺序。

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。