**请你说一说数据库索引**

**参考回答：**

索引是对数据库表中一列或多列的值进行排序的一种结构，使用索引可快速访问数据库表中的特定信息。如果想按特定职员的姓来查找他或她，则与在表中搜索所有的行相比，索引有助于更快地获取信息。

索引的一个主要目的就是加快检索表中数据的方法，亦即能协助信息搜索者尽快的找到符合限制条件的记录ID的辅助数据结构。

**● 请你说一说数据库事务**

**参考回答：**

数据库事务(Database Transaction) ，是指作为单个逻辑工作单元执行的一系列操作，要么完全地执行，要么完全地不执行。 事务处理可以确保除非事务性单元内的所有操作都成功完成，否则不会永久更新面向数据的资源。通过将一组相关操作组合为一个要么全部成功要么全部失败的单元，可以简化错误恢复并使应用程序更加可靠。一个逻辑工作单元要成为事务，必须满足所谓的ACID（原子性、一致性、隔离性和持久性）属性。事务是数据库运行中的逻辑工作单位，由DBMS中的事务管理子系统负责事务的处理。

**● 请你说一说数据库事务隔离**

**参考回答：**

同一时间，只允许一个事务请求同一数据，不同的事务之间彼此没有任何干扰。比如A正在从一张银行卡中取钱，在A取钱的过程结束前，B不能向这张卡转账。

**● 请你说一说inner join和left join**

**参考回答：**

left join(左联接) 返回包括左表中的所有记录和右表中联结字段相等的记录 right join(右联接) 返回包括右表中的所有记录和左表中联结字段相等的记录  
inner join(等值连接) 只返回两个表中联结字段相等的行

**● 请你聊一聊数据库事物的一致性**

**参考回答：**

事务（Transaction）是由一系列对系统中数据进行访问与更新的操作所组成的一个程序执行逻辑单元。事务是DBMS中最基础的单位，事务不可分割。

事务具有4个基本特征，分别是：原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）、持久性（Duration），简称ACID。

1）原子性（Atomicity）

原子性是指事务包含的所有操作要么全部成功，要么全部失败回滚，[删删删]因此事务的操作如果成功就必须要完全应用到数据库，如果操作失败则不能对数据库有任何影响。

2）一致性（Consistency）

一致性是指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。

拿转账来说，假设用户A和用户B两者的钱加起来一共是5000，那么不管A和B之间如何转账，转几次账，事务结束后两个用户的钱相加起来应该还得是5000，这就是事务的一致性。

3）隔离性（Isolation）

隔离性是当多个用户并发访问数据库时，比如操作同一张表时，数据库为每一个用户开启的事务，不能被其他事务的操作所干扰，多个并发事务之间要相互隔离。

即要达到这么一种效果：对于任意两个并发的事务T1和T2，在事务T1看来，T2要么在T1开始之前就已经结束，要么在T1结束之后才开始，这样每个事务都感觉不到有其他事务在并发地执行。

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。这指的是在并发环境中，当不同的事务同时操纵相同的数据时，每个事务都有各自的完整数据空间。由并发事务所做的修改必须与任何其他并发事务所做的修改隔离。

不同的隔离级别：

Read Uncommitted（读取未提交[添加中文释义]内容）：最低的隔离级别，什么都不需要做，一个事务可以读到另一个事务未提交的结果。所有的并发事务问题都会发生。

Read Committed（读取提交内容）：只有在事务提交后，其更新结果才会被其他事务看见。可以解决脏读问题。

Repeated Read（可重复读）：在一个事务中，对于同一份数据的读取结果总是相同的，无论是否有其他事务对这份数据进行操作，以及这个事务是否提交。可以解决脏读、不可重复读。

Serialization（可串行化）：事务串行化执行，隔离级别最高，牺牲了系统的并发性。可以解决并发事务的所有问题。

4）持久性（Durability）

持久性是指一个事务一旦被提交了，那么对数据库中的数据的改变就是永久性的，即便是在数据库系统遇到故障的情况下也不会丢失提交事务的操作。

例如我们在使用JDBC操作数据库时，在提交事务方法后，提示用户事务操作完成，当我们程序执行完成直到看到提示后，就可以认定事务以及正确提交，即使这时候数据库出现了问题，也必须要将我们的事务完全执行完成，否则就会造成我们看到提示事务处理完毕，但是数据库因为故障而没有执行事务的重大错误。

**● 请你说说索引是什么，多加索引一定会好吗**

**参考回答：**

1、索引

数据库索引是为了增加查询速度而对表字段附加的一种标识，是对数据库表中一列或多列的值进行排序的一种结构。

DB在执行一条Sql语句的时候，默认的方式是根据搜索条件进行全表扫描，遇到匹配条件的就加入搜索结果集合。如果我们对某一字段增加索引，查询时就会先去索引列表中一次定位到特定值的行数，大大减少遍历匹配的行数，所以能明显增加查询的速度。

优点：

通过创建唯一性索引，可以保证数据库表中每一行数据的唯一性。

可以大大加快数据的检索速度，这也是创建索引的最主要的原因。

可以加速表和表之间的连接，特别是在实现数据的参考完整性方面特别有意义。

在使用分组和排序子句进行数据检索时，同样可以显著减少查询中分组和排序的时间。

通过使用索引，可以在查询的过程中，使用优化隐藏器，提高系统的性能。

缺点：

创建索引和维护索引要耗费时间，这种时间随着数据量的增加而增加。

索引需要占物理空间，除了数据表占数据空间之外，每一个索引还要占一定的物理空间，如果要建立聚簇索引，那么需要的空间就会更大。

当对表中的数据进行增加、删除和修改的时候，索引也要动态的维护，这样就降低了数据的维护速度。

2、添加索引原则

在查询中很少使用或者参考的列不应该创建索引。这是因为，既然这些列很少使用到，因此有索引或者无索引，并不能提高查询速度。相反，由于增加了索引，反而降低了系统的维护速度和增大了空间需求。

只有很少数据值的列也不应该增加索引。这是因为，由于这些列的取值很少，例如人事表的性别列，在查询的结果中，结果集的数据行占了表中数据行的很大比例，即需要在表中搜索的数据行的比例很大。增加索引，并不能明显加快检索速度。

定义为text、image和bit数据类型的列不应该增加索引。这是因为，这些列的数据量要么相当大，要么取值很少。

当修改性能远远大于检索性能时，不应该创建索引。这是因为，修改性能和检索性能是互相矛盾的。当增加索引时，会提高检索性能，但是会降低修改性能。当减少索引时，会提高修改性能，降低检索性能。因此，当修改性能远远大于检索性能时，不应该创建索引。

**● 请你说一下数据库事务以及四个特性**

**参考回答：**

事务（Transaction）是由一系列对系统中数据进行访问与更新的操作所组成的一个程序执行逻辑单元。事务是DBMS中最基础的单位，事务不可分割。

事务具有4个基本特征，分别是：原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）、持久性（Duration），简称ACID。

1. 原子性（Atomicity）

原子性是指事务包含的所有操作要么全部成功，要么全部失败回滚，[删删删]因此事务的操作如果成功就必须要完全应用到数据库，如果操作失败则不能对数据库有任何影响。

2. 一致性（Consistency）

一致性是指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。

拿转账来说，假设用户A和用户B两者的钱加起来一共是5000，那么不管A和B之间如何转账，转几次账，事务结束后两个用户的钱相加起来应该还得是5000，这就是事务的一致性。

3. 隔离性（Isolation）

隔离性是当多个用户并发访问数据库时，比如操作同一张表时，数据库为每一个用户开启的事务，不能被其他事务的操作所干扰，多个并发事务之间要相互隔离。

即要达到这么一种效果：对于任意两个并发的事务T1和T2，在事务T1看来，T2要么在T1开始之前就已经结束，要么在T1结束之后才开始，这样每个事务都感觉不到有其他事务在并发地执行。

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。

这指的是在并发环境中，当不同的事务同时操纵相同的数据时，每个事务都有各自的完整数据空间。由并发事务所做的修改必须与任何其他并发事务所做的修改隔离。

不同的隔离级别：

Read Uncommitted（读取未提交[添加中文释义]内容）：最低的隔离级别，什么都不需要做，一个事务可以读到另一个事务未提交的结果。所有的并发事务问题都会发生。

Read Committed（读取提交内容）：只有在事务提交后，其更新结果才会被其他事务看见。可以解决脏读问题。

Repeated Read（可重复读）：在一个事务中，对于同一份数据的读取结果总是相同的，无论是否有其他事务对这份数据进行操作，以及这个事务是否提交。可以解决脏读、不可重复读。

Serialization（可串行化）：事务串行化执行，隔离级别最高，牺牲了系统的并发性。可以解决并发事务的所有问题。

4. 持久性（Durability）

持久性是指一个事务一旦被提交了，那么对数据库中的数据的改变就是永久性的，即便是在数据库系统遇到故障的情况下也不会丢失提交事务的操作。

例如我们在使用JDBC操作数据库时，在提交事务方法后，提示用户事务操作完成，当我们程序执行完成直到看到提示后，就可以认定事务以及正确提交，即使这时候数据库出现了问题，也必须要将我们的事务完全执行完成，否则就会造成我们看到提示事务处理完毕，但是数据库因为故障而没有执行事务的重大错误。

**● 请你说一说数据库的三大范式**

**参考回答：**

第一范式：当关系模式R的所有属性都不能再分解为更基本的数据单位时，称R是满足第一范式，即属性不可分

第二范式：如果关系模式R满足第一范式，并且R得所有非主属性都完全依赖于R的每一个候选关键属性，称R满足第二范式

第三范式：设R是一个满足第一范式条件的关系模式，X是R的任意属性集，如果X非传递依赖于R的任意一个候选关键字，称R满足第三范式，即非主属性不传递依赖于键码

**● 请你介绍一下数据库的ACID特性**

**参考回答：**

1)原子性：事务被视为不可分割的最小单元，事物的所有操作要不成功，要不失败回滚，而回滚可以通过日志来实现，日志记录着事务所执行的修改操作，在回滚时反向执行这些修改操作。

2)一致性：数据库在事务执行前后都保持一致性状态，在一致性状态下，所有事务对一个数据的读取结果都是相同的。

3)隔离性：一个事务所做的修改在最终提交以前，对其他事务是可不见的。

4)持久性：一旦事务提交，则其所做的修改将会永远保存到数据库中。

**请你说一说mysql的四种隔离状态**

**参考回答：**

Mysql主要包含四种隔离状态：

|  |  |  |  |
| --- | --- | --- | --- |
| 事务隔离级别 | 脏读 | 不可重复读 | 幻读 |
| 读未提交（read-uncommitted） | 是 | 是 | 是 |
| 不可重复读（read-committed） | 否 | 是 | 是 |
| 可重复读（repeatable-read） | 否 | 否 | 是 |
| 串行化（serializable） | 否 | 否 | 否 |

**● 请你介绍一下mysql的MVCC机制**

**参考回答：**

MVCC是一种多版本并发控制机制，是MySQL的InnoDB存储引擎实现隔离级别的一种具体方式，用于实现提交读和可重复读这两种隔离级别。MVCC是通过保存数据在某个时间点的快照来实现该机制，其在每行记录后面保存两个隐藏的列，分别保存这个行的创建版本号和删除版本号，然后Innodb的MVCC使用到的快照存储在Undo日志中，该日志通过回滚指针把一个数据行所有快照连接起来。

**● 请问SQL优化方法有哪些**

**参考回答：**

通过建立索引对查询进行优化

对查询进行优化，应尽量避免全表扫描

**● 请你说一下MySQL引擎和区别**

**参考回答：**

1、MySQL引擎

MySQL中的数据用各种不同的技术存储在文件（或者内存）中。这些技术中的每一种技术都使用不同的存储机制、索引技巧、锁定水平并且最终提供广泛的不同的功能和能力。通过选择不同的技术，你能够获得额外的速度或者功能，从而改善你的应用的整体功能。

数据库引擎是用于存储、处理和保护数据的核心服务。利用数据库引擎可控制访问权限并快速处理事务，从而满足企业内大多数需要处理大量数据的应用程序的要求。使用数据库引擎创建用于联机事务处理或联机分析处理数据的关系数据库。这包括创建用于存储数据的表和用于查看、管理和保护数据安全的数据库对象（如索引、视图和存储过程）。

MySQL存储引擎主要有： MyIsam、InnoDB、Memory、Blackhole、CSV、Performance\_Schema、Archive、Federated、Mrg\_Myisam。

但是最常用的是InnoDB和Mylsam。

2、InnoDB

InnoDB是一个事务型的存储引擎，有行级锁定和外键约束。

Innodb引擎提供了对数据库ACID事务的支持，并且实现了SQL标准的四种隔离级别，关于数据库事务与其隔离级别的内容请见数据库事务与其隔离级别这类型的文章。该引擎还提供了行级锁和外键约束，它的设计目标是处理大容量数据库系统，它本身其实就是基于MySQL后台的完整数据库系统，MySQL运行时Innodb会在内存中建立缓冲池，用于缓冲数据和索引。但是该引擎不支持FULLTEXT类型的索引，而且它没有保存表的行数，当SELECT COUNT(\*) FROM TABLE时需要扫描全表。当需要使用数据库事务时，该引擎当然是首选。由于锁的粒度更小，写操作不会锁定全表，所以在并发较高时，使用Innodb引擎会提升效率。但是使用行级锁也不是绝对的，如果在执行一个SQL语句时MySQL不能确定要扫描的范围，InnoDB表同样会锁全表。

适用场景：

经常更新的表，适合处理多重并发的更新请求。

支持事务。

可以从灾难中恢复（通过bin-log日志等）。

外键约束。只有他支持外键。

支持自动增加列属性auto\_increment。

索引结构：

InnoDB也是B+Treee索引结构。Innodb的索引文件本身就是数据文件，即B+Tree的数据域存储的就是实际的数据，这种索引就是聚集索引。这个索引的key就是数据表的主键，因此InnoDB表数据文件本身就是主索引。

InnoDB的辅助索引数据域存储的也是相应记录主键的值而不是地址，所以当以辅助索引查找时，会先根据辅助索引找到主键，再根据主键索引找到实际的数据。所以Innodb不建议使用过长的主键，否则会使辅助索引变得过大。建议使用自增的字段作为主键，这样B+Tree的每一个结点都会被顺序的填满，而不会频繁的分裂调整，会有效的提升插入数据的效率。

3、Mylsam

MyIASM是MySQL默认的引擎，但是它没有提供对数据库事务的支持，也不支持行级锁和外键，因此当INSERT或UPDATE数据时即写操作需要锁定整个表，效率便会低一些。MyIsam 存储引擎独立于操作系统，也就是可以在windows上使用，也可以比较简单的将数据转移到linux操作系统上去。

适用场景：

不支持事务的设计，但是并不代表着有事务操作的项目不能用MyIsam存储引擎，可以在service层进行根据自己的业务需求进行相应的控制。

不支持外键的表设计。

查询速度很快，如果数据库insert和update的操作比较多的话比较适用。

整天对表进行加锁的场景。

MyISAM极度强调快速读取操作。

MyIASM中存储了表的行数，于是SELECT COUNT(\*) FROM TABLE时只需要直接读取已经保存好的值而不需要进行全表扫描。如果表的读操作远远多于写操作且不需要数据库事务的支持，那么MyIASM也是很好的选择。

缺点：就是不能在表损坏后主动恢复数据。

索引结构：

MyISAM索引结构：MyISAM索引用的B+ tree来储存数据，MyISAM索引的指针指向的是键值的地址，地址存储的是数据。B+Tree的数据域存储的内容为实际数据的地址，也就是说它的索引和实际的数据是分开的，只不过是用索引指向了实际的数据，这种索引就是所谓的非聚集索引。

3、InnoDB和Mylsam的区别：

1）事务：MyISAM类型不支持事务处理等高级处理，而InnoDB类型支持，提供事务支持已经外部键等高级数据库功能。

2）性能：MyISAM类型的表强调的是性能，其执行数度比InnoDB类型更快。

3）行数保存：InnoDB 中不保存表的具体行数，也就是说，执行select count() fromtable时，InnoDB要扫描一遍整个表来计算有多少行，但是MyISAM只要简单的读出保存好的行数即可。注意的是，当count()语句包含where条件时，两种表的操作是一样的。

4）索引存储：对于AUTO\_INCREMENT类型的字段，InnoDB中必须包含只有该字段的索引，但是在MyISAM表中，可以和其他字段一起建立联合索引。MyISAM支持全文索引（FULLTEXT）、压缩索引，InnoDB不支持。

MyISAM的索引和数据是分开的，并且索引是有压缩的，内存使用率就对应提高了不少。能加载更多索引，而Innodb是索引和数据是紧密捆绑的，没有使用压缩从而会造成Innodb比MyISAM体积庞大不小。

InnoDB存储引擎被完全与MySQL服务器整合，InnoDB存储引擎为在主内存中缓存数据和索引而维持它自己的缓冲池。InnoDB存储它的表＆索引在一个表空间中，表空间可以包含数个文件（或原始磁盘分区）。这与MyISAM表不同，比如在MyISAM表中每个表被存在分离的文件中。InnoDB 表可以是任何尺寸，即使在文件尺寸被限制为2GB的操作系统上。

5）服务器数据备份：InnoDB必须导出SQL来备份，LOAD TABLE FROM MASTER操作对InnoDB是不起作用的，解决方法是首先把InnoDB表改成MyISAM表，导入数据后再改成InnoDB表，但是对于使用的额外的InnoDB特性(例如外键)的表不适用。

MyISAM应对错误编码导致的数据恢复速度快。MyISAM的数据是以文件的形式存储，所以在跨平台的数据转移中会很方便。在备份和恢复时可单独针对某个表进行操作。

InnoDB是拷贝数据文件、备份 binlog，或者用 mysqldump，在数据量达到几十G的时候就相对痛苦了。

6）锁的支持：MyISAM只支持表锁。InnoDB支持表锁、行锁 行锁大幅度提高了多用户并发操作的新能。但是InnoDB的行锁，只是在WHERE的主键是有效的，非主键的WHERE都会锁全表的。

**请你回答一下mongodb和redis的区别**

**参考回答：**

内存管理机制上：Redis 数据全部存在内存，定期写入磁盘，当内存不够时，可以选择指定的 LRU 算法删除数据。MongoDB 数据存在内存，由 linux系统 mmap 实现，当内存不够时，只将热点数据放入内存，其他数据存在磁盘。

支持的数据结构上：Redis 支持的数据结构丰富，包括hash、set、list等。

MongoDB 数据结构比较单一，但是支持丰富的数据表达，索引，最类似关系型数据库，支持的查询语言非常丰富

**● 请你说一下mysql引擎以及其区别**

**参考回答：**

在Mysql数据库中，常用的引擎为Innodb和MyIASM,其中Innodb是一个事务型的存储引擎，有行级锁定和外键约束，提供了对数据库ACID事物的支持，实现了SQL标准的四种隔离级别，即读未提交，不可重复读，可重复读以及串行,其涉及目标就是处理大数据容量的数据库系统。而MyIASM引擎是Mysql默认的引擎，不提供数据库事务的支持，也不支持行级锁和外键，因此当写操作时需要锁定整个表，效率较低。不过其保存了表的行数，当金星select count(\*)form table时，可直接读取已经保存的值，不需要进行全表扫描。因此当表的读操作远多于写操作，并且不需要事务支持时，可以优先选择MyIASM

**● 请你来说一说Redis的定时机制怎么实现的**

**参考回答：**

Redis服务器是一个事件驱动程序，服务器需要处理以下两类事件：文件事件（服务器对套接字操作的抽象）和时间事件（服务器对定时操作的抽象）。Redis的定时机制就是借助时间事件实现的。

一个时间事件主要由以下三个属性组成：id：时间事件标识号；when：记录时间事件的到达时间；timeProc：时间事件处理器，当时间事件到达时，服务器就会调用相应的处理器来处理时间。一个时间事件根据时间事件处理器的返回值来判断是定时事件还是周期性事件

一个时间事件主要由以下三个属性组成：id：时间事件标识号；when：记录时间事件的到达时间；timeProc：时间事件处理器，当时间事件到达时，服务器就会调用相应的处理器来处理时间。一个时间事件根据时间事件处理器的返回值来判断是定时事件还是周期性事件。

**● 请你来说一说Redis是单线程的，但是为什么这么高效呢?**

**参考回答：**

虽然Redis文件事件处理器以单线程方式运行，但是通过使用I/O多路复用程序来监听多个套接字，文件事件处理器既实现了高性能的网络通信模型，又可以很好地与Redis服务器中其他同样以单线程运行的模块进行对接，这保持了Redis内部单线程设计的简单性。

**● 请问Redis的数据类型有哪些，底层怎么实现?**

**参考回答：**

1）字符串：整数值、embstr编码的简单动态字符串、简单动态字符串（SDS)

2）列表：压缩列表、双端链表

3）哈希：压缩列表、字典

4）集合：整数集合、字典

5）有序集合：压缩列表、跳跃表和字典

**● 请问Redis的rehash怎么做的，为什么要渐进rehash，渐进rehash又是怎么实现的?**

**参考回答：**

因为redis是单线程，当K很多时，如果一次性将键值对全部rehash，庞大的计算量会影响服务器性能，甚至可能会导致服务器在一段时间内停止服务。不可能一步完成整个rehash操作，所以redis是分多次、渐进式的rehash。渐进性哈希分为两种：

1）操作redis时，额外做一步rehash

对redis做读取、插入、删除等操作时，会把位于table[dict->rehashidx]位置的链表移动到新的dictht中，然后把rehashidx做加一操作，移动到后面一个槽位。

2）后台定时任务调用rehash

后台定时任务rehash调用链，同时可以通过server.hz控制rehash调用频率

**● 请你来说一下Redis和memcached的区别**

**参考回答：**

1）数据类型 ：redis数据类型丰富，支持set liset等类型；memcache支持简单数据类型，需要客户端自己处理复杂对象

2）持久性：redis支持数据落地持久化存储；memcache不支持数据持久存储。)

3）分布式存储：redis支持master-slave复制模式；memcache可以使用一致性hash做分布式。

4）value大小不同：memcache是一个内存缓存，key的长度小于250字符，单个item存储要小于1M，不适合虚拟机使用

5）数据一致性不同：redis使用的是单线程模型，保证了数据按顺序提交；memcache需要使用cas保证数据一致性。CAS（Check and Set）是一个确保并发一致性的机制，属于“乐观锁”范畴；原理很简单：拿版本号，操作，对比版本号，如果一致就操作，不一致就放弃任何操作

6）cpu利用：redis单线程模型只能使用一个cpu，可以开启多个redis进程