数据库

2018年6月15日 19:11

1.事务的四大特性（ACID）

事务是用户定义的一个数据库操作序列，这些操作要么**全部**执行，要么**全不**执行，是一个不可分割的工作单位。例如在关系型数据库中，一个事务可以是一条sql语句，一组sql语句或整个程序。

事务的ACID特性：

（1）A原子性：

事务必须是原子工作单元；对其数据修改，要么全不执行，要么全部执行。通常，与某个事务关联的操作具有共同的目标，并且是相互依赖的。如果西永只执行这些操作的一个子集，则可能破坏事务的总体目标。原子性消除了系统处理操作子集的可能性，

（2）C一致性

事务在完成时，必须使所有的数据都保持一致状态。在相关数据库中，所有规则都必须应用于事务的修改，以保持所有数据的完整性。事务结束时，所有的内部数据结构都必须是正确的。

（3）I隔离性

指的是在并发的环境中，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其他任何并发事务所做的修改隔离。事务查看数据更新时，数据所处的状态要么是另一事务修改它之前，要么是另一事务修改它之后的状态，事务不会查看到中间状态的数据。

（4）D持久性

指的是只要事务成功结束，它对数据库所做的更新就必须永久保存下来。即使发生系统崩溃，重新启动数据库系统后，数据库还能会恢复到书屋成功结束时的状态。

事务的ACID特性是由关系数据库管理系统（RDBMS）来实现的。**数据库管理系统采用日志来保证事务的原子性、一致性和持久性。**日志记录了事务对数据库所做的更新，如果某个事务在执行过程中发生错误，就可以根据日志撤销事务对数据已做的更新，是数据库退回到执行事务前的初始状态。**数据库管理系统采用锁机制来实现可事务的隔离性。**当多个事务同时更新了数据库中相同的数据时，只允许持有锁的事务更新数据，其他事务必须等待，直到前一个事务释放了锁，其他事务才有机会更新该数据。

2.数据库隔离级别，每个级别会引发什么问题，mysql默认是哪个级别的？

（一）在标准SQL规范中，定义了4个事务隔离级别，不同隔离级别对事务的处理也不相同：

（1）未授权读取（READ UNCOMMITTTED）事务中的修改：即使没有提交，对其他事务也是可见的（如果一个事务已经开始写数据，则另一个事务不允许同时进行写操作，但允许其他事务读此行数据）；该隔离级别可以用“排他锁”来实现；

（2）授权获取（READ COMMITTED）：一个事务只能读取已经提交的事务所做的修改。换句话说，一个事务所作的修改在提交之前对其他事务是不可见的。该级别可以通过“瞬间共享读锁”和“排他写锁”实现。

（3）可重复读取（REPEATABLE READ）：保证同一个事务中多次读取同样的数据结果是一样的。通过“共享读锁”和“排他写锁”实现

（4）序列化（SERIALIXABLE）：强制事务串行执行，提供严格的事务隔离。它要求事务序列化执行，事务只能一个接着一个执行不能并发执行。仅仅通过“行级锁”是无法实现事务序列化的，必须通过其他机制保证新插入的数据不会被刚执行查询操作的事务访问到。

隔离级别	脏读	不可重复读	幻影读
未提交读	YES	YES	YES
提交读	NO	YES	YES
可重复读	NO	NO	YES
可串行化	NO	NO	NO

隔离级别越高，越能保证数据的完整性和一致性，但是对并发性能的影响也越大。对于多数应用程序，可以优先考虑把数据库系统的隔离级别设为Read Committed。它能够避免脏读取，而且具有较好的并发性能。尽管它会导致不可重复读、幻读和第二类丢失更新这些并发问题，在可能出现这类问题的个别场合，可以由应用程序采用悲观锁或乐观锁来控制。

（二）数据库是要被广大客户所共享访问的，那么在数据库操作过程中很可能出现以下几种不确定的情况：

（1）更新丢失：两个事务都同时更新一行数据，一个事务对数据的更新把另一个事务对数据的更新覆盖了。这是因为系统没有执行任何的锁操作，因此并发事务并没有被隔离开来。

（2）脏读：一个事务读取到了另一个事务未提交的数据操作结果。这是相当危险的，因为很可能所有的操作都被回滚。

（3）不可重复读：一个事务对同一行数据重复读取两次，但是却得到了不同的结果。包括以下情况：

① 虚读：事务T1读取某一数据后，事务T2对其做了修改，当事务T1再次读该数据时得到与前一次不同的值。

② 幻读（Phantom Reads）：事务在操作过程中进行两次查询，第二次查询的结果

包含了第一次查询中未出现的数据或者缺少了第一次查询中出现的数据（这里并不要求两次查询的SQL语句相同）。这是因为在两次查询过程中有另外一个事务插入数据造成的。

3.MYSQL的两种存储引擎区别（事务、锁级别等等），各自使用的场景

（1）INNODB

①具有事务（commit）、回滚（rollback）和崩溃修复能力（crash recovery capabilities）的事务安全（transaction-safe（ACID compliant））型表，

②支持外键

③InnoDB中不保存表的具体行数，也就是说，执行select count（*） from table时，InnoDB要扫描整一遍整个表来计算有多少行，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对AUTO_INCREMENT类型的字段，InnoDB中必须包含只有该字段的索引

⑤DELETE FROM table时，InnoDB不会重新建立表，而是一行一行的删除。

（2）MyISAM

①不支持事务操作

②不支持外键操作

③MyISAM保存表的具体行数，执行select count（*） from table时只要简单的读出保存好的行数即可

④对于AUTO_INCREMENT类型的字段，在MyISAM表中，可以和其他字段一起建立联合索引

⑤MyISAM存储的读锁和写锁是互斥的，读写操作是串行的。那么如果一个进程请求某个MyISAM表的读锁，同时另一个进程也请求同一表的写锁，MySQL如何处理？----------**写进程先获得锁。不仅如此，即使读请求先到锁等待队列，写请求后到，写锁也会插到读锁请求之前。**这是因为MySQL认为写请求比读请求要重要。这也正是MyISAM表不太适合于有大量更新操作和查询操作应用的原因，因为，**大量的更新操作会造成查询操作很难获得读锁，从而可能永远阻塞。**这种情况有时可能会变得非常糟糕！

3.数据库的优化（从sql语句的优化和索引两个部分来回答）（**没有回答完**）

http://www.jasongj.com/2015/03/07/Join1/

SQL优化（一） Merge Join vs. Hash Join vs. Nested Loop

Nested Loop，Hash Join，Merge Join介绍

- Nested Loop:

  对于被连接的数据子集较小的情况，Nested Loop是个较好的选择。Nested Loop就是扫描一个表（外表），每读到一条记录，就根据Join字段上的索引去另一张表（内表）里面查找，若Join字段上没有索引查询优化器一般就不会选择 Nested Loop。在Nested Loop中，内表（一般是带索引的大表）被外表（也叫“驱动表”，一般为小表——不紧相对其它表为小表，而且记录数的绝对值也较小，不要求有索引）驱动，外表返回的每一行都要在内表中检索找到与它匹配的行，因此整个查询返回的结果集不能太大（大于1 万不适合）。

- Hash Join:

  Hash Join是做大数据集连接时的常用方式，优化器使用两个表中较小（相对较小）的表利用Join Key在内存中建立散列表，然后扫描较大的表并探测散列表，找出与Hash表匹配的行。

  这种方式适用于较小的表完全可以放于内存中的情况，这样总成本就是访问两个表的成本之和。但是在表很大的情况下并不能完全放入内存，这时优化器会将它分割成若干不同的分区，不能放入内存的部分就把该分区写入磁盘的临时段，此时要求有较大的临时段从而尽量提高I/O 的性能。它能够很好的工作于没有索引的大表和并行查询的环境中，并提供最好的性能。大多数人都说它是Join的重型升降机。Hash Join只能应用于等值连接(如WHERE A.COL3 = B.COL4)，这是由Hash的特点决定的。

- Merge Join:

  通常情况下Hash Join的效果都比排序合并连接要好，然而如果两表已经被排过序，在执行排序合并连接时不需要再排序了，这时Merge Join的性能会优于Hash Join。Merge join的操作通常分三步：

  1. 对连接的每个表做table access full;
  2. 对table access full的结果进行排序。
  3. 进行merge join对排序结果进行合并。

  在全表扫描比索引范围扫描再进行表访问更可取的情况下，Merge Join会比Nested Loop性能更佳。当表特别小或特别巨大的时候，实行全表访问可能会比索引范围扫描更有效。Merge Join的性能开销几乎都在前两步。Merge Join可适于于非等值Join（>，<，>=，<=，但是不包含!=，也即<>）

Nested Loop，Hash JOin，Merge Join对比

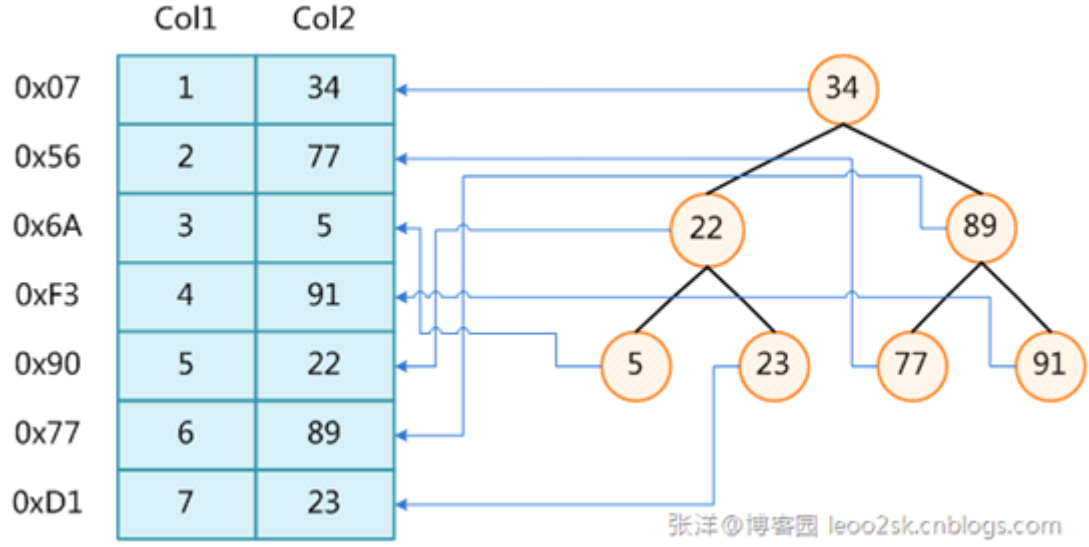
类别	Nested Loop	Hash Join	Merge Join
使用条件	任何条件	等值连接（=）	等值或非等值连接(>，<，=，>=，<=)，‘<>’除外
相关资源	CPU、磁盘I/O	内存、临时空间	内存、临时空间
特点	当有高选择性索引或进行限制性搜索时效率比较高，能够快速返回第一次的搜索结果。	当缺乏索引或者索引条件模糊时，Hash Join比Nested Loop有效。通常比Merge Join快。在数据仓库环境下，如果表的纪录数多，效率高。	当缺乏索引或者索引条件模糊时，Merge Join比Nested Loop有效。非等值连接时，Merge Join比Hash Join更有效
缺点	当索引丢失或者查询条件限制不够时，效率很低；当表的纪录数多时，效率低。	为建立哈希表，需要大量内存。第一次的结果返回较慢。	所有的表都需要排序。它为最优化的吞吐量而设计，并且在结果没有全部找到前不返回数据。

5.数据库索引，索引有B+索引和hash索引，说一下它们各自的区别

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、更新数据表中的数据。索引的实现通常使用B树及其变种B+树。

在数据之外，数据库系统还维护着满足特定查找算法的数据结构，这些数据结构以某种方式引用（指向）数据，这样就可以在这些数据结构上实现高级查找算法，这种数据结构就是索引。

为表设计索引要付出代价：（1）增加了数据库的存储空间；（2）插入和修改数据时要花费较多的时间（因为索引也要随之变动）



上图展示了一种可能的索引方式。左边是数据表，一共有两列七条记录，最左边的是数据记录的物理地址（注意逻辑上相邻的记录在磁盘上也并不是一定物理相邻的）。为了加快Col2的查找，可以维护一个右边所示的二叉查找树，每个节点分别包含索引键值和一个指向对应数据记录物理地址的指针，这样就可以运用二叉查找在O(log2n)的复杂度内获取到相应数据。

创建索引可以大大提高系统的性能。（优点）

（1）通过创建唯一性索引，可以保证数据库表每行数据的唯一性。

（2）可以大大加快数据检索的速度，这也是创建索引的最主要的原因

（3）可以加速表和表之间的连接，特别是在实现数据参考完整性方面有特殊的意义。

（4）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分组和排序的时间。

（5）通过使用索引，可以在查询过程中，使用优化隐藏器，提高系统的性能。

增加索引也有很多不利的方面（缺点）

（1）创建索引和维护索引要消耗时间，这种时间随着数据量的增加而增加

（2）索引要占物理空间，除了数据表占数据空间之外，每个索引还要占一定的物理空间，如果建立聚簇索引，那么空间就会更大。

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

索引是建立在数据库表中的某些列的上面。在创建索引的时候，应该考虑在哪些列上可以创建索引，在哪些列上不能创建索引。一般来说，**应该在这些列上创建索引**：在经常需要搜索的列上，可以加快搜索的速度；在作为主键的列上，强制该列的唯一性和组织表中数据的排列结构；在经常用在连接的列上，这些列主要是一些外键，可以加快连接的速度；在经常需要根据范围进行搜索的列上创建索引，因为索引已经排序，其指定的范围是连续的；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，加快排序查询时间；在经常使用在WHERE子句中的列上面创建索引，加快条件的判断速度。

同样，对于有些列不应该创建索引。一般来说，**不应该创建索引的的这些列具有下列特点**：

（1）对于那些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。

（2）对于那些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。

（3）对于那些定义为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。

（4）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。

索引的实现方式

（1）B+树　我们经常听到B+树就是这个概念，用这个树的目的和红黑树差不多，也是为了尽量保持树的平衡，当然红黑树是二叉树，**但B+树就不是二叉树了**，节点下面可以有多个子节点，数据库开发商会设置子节点数的一个最大值，这个值不会太小，所以B+树一般来说比较矮胖，而红黑树就比较瘦高了。关于B+树的插入，删除，会涉及到一些算法以保持树的平衡，这里就不详述了。ORACLE的默认索引就是这种结构的。如果经常需要同时对两个字段进行AND查询,那么使用两个单独索引不如建立一个复合索引，因为两个单独索引通常数据库只能使用其中一个，而使用复合索引因为索引本身就对应到两个字段上的，效率会有很大提高。

（2）**散列索引**　第二种索引叫做散列索引，就是通过散列函数来定位的一种索引，不过很少有单独使用散列索引的，反而是散列文件组织用的比较多。散列文件组织就是根据一个键通过散列计算把对应的记录都放到同一个槽中，这样的话相同的键值对应的记录就一定是放在同一个文件里了，也就减少了文件读取的次数，提高了效率。

散列索引呢就是根据对应键的散列码来找到最终的索引项的技术，其实和B树就差不多了，也就是一种索引之上的二级辅助索引，我理解散列索引都是二级或更高级的稀疏索引，否则桶就太多了，效率也不会很高。

（3）位图索引 位图索引是一种针对多个字段的简单查询设计一种特殊的索引，适用范围比较小，只适用于字段值固定并且值的种类很少的情况，比如性别，只能有男和女，或者级别，状态等等，并且只有在同时对多个这样的字段查询时才能体现出位图的优势。位图的基本思想就是对每一个条件都用0或者1来表示，如有5条记录，性别分别是男，女，男，男，女，那么如果使用位图索引就会建立两个位图，对应男的10110和对应女的01001,这样做有什么好处呢，就是如果同时对多个这种类型的字段进行and或or查询时，可以使用按位与和按位或来直接得到结果了。

B+树最常用，性能也不差，用于范围查询和单值查询都可以。特别是范围查询，非得用B+树这种顺序的才可以了。HASH的如果只是对单值查询的话速度会比B+树快一点，但是ORACLE好像不支持HASH索引，只支持HASH表空间。位图的使用情况很局限，只有很少的情况才能用，一定要确定真正适合使用这种索引才用（值的类型很少并且需要复合查询），否则建立一大堆位图就一点意义都没有了。

聚集索引和非聚集索引的区别？（数据库）

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

聚集索引（CLUSTER INDEX）是指索引项的顺序与表中记录的物理顺序**一致**的索引组织。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。用户可以在最经常查询的列上建立聚簇索引以提高查询效率。在一个基本表上最多只能建立一个聚簇索引。聚集索引的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：

（1）此列包含有限数目的不同值

（2）查询结果返回一个区间的值

（3）查询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：

（1）此列包含了大量数目不同的值

（2）查询结束返回少量的结果集

（3）order by 子句中使用了该列

6.B+索引数据结构和B树的区别

7.索引的分类（主键索引、唯一索引），最左前缀原则，哪些情况下索引会失效

Mysql索引，看到一个很少比如：索引就好比一本书的目录，它会让你很快找到内容，显然索引不是越多越好，加入这本书有10001页，有500是目录，当然它的效率是很低的，因为索引是要占磁盘空间的。

Mysql索引方式主要有两种结构：B+树和hash

（1）hash索引在Mysql中比较少用，他把数据形式以hash形式组织起来，因此查找某一条家记录的时候，速度非常快。但是因为hash结构，每个键对应一个值，而且是以散列形式分布的，所以它并不支持范围查找和排序等功能。

（2）B+树：这是Mysql中使用最频繁的一个索引数据结构，数据结构以平衡树的形式组织，所以更适合来处理数据，范围查找等功能。相对于hash索引，B+树在单条记录的速度虽然比不上hash索引，但是它更适合排序等操作，因此适用范围更广。

（3）Mysql常见的索引类别有：主键索引、唯一索引、普通索引、全文索引、组合索引

PRIMARY KEY（主键索引）ALTER TABLE 'table_name' ADD PRIMARY KEY ('column')

UNIQUE（唯一索引）ALTER TABLE 'table_name' ADD UNIQUE ('column')

INDEX（普通索引）ALTER TABLE 'table_name' ADD INDEX index_name ('column')

FULLIEXT（全文索引）ALTER TABLE 'table_name' ADD FULLTEXT ('column')

组合索引：ALTER TABLE 'table_name' ADD INDEX

index_name ('column1'，'column2','column3')

（4）Mysql各种索引的类别

①普通索引：最基本的索引，没有任何限制

②唯一索引：与“普通索引”类似，不同的就是：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

④全文索引：仅可用于MyISAM表，针对较大的数据，生成全文索引很耗时耗空间

⑤组合索引：为了更多的提高mysql效率可建立组合索引，遵循“最左前缀”原则

8.聚集索引和非聚集索引的区别（答案见第5题）

9.有哪些锁，SELECT时怎么加排他锁？？？？？？

数据库和操作系统一样，是一个多用户使用的共享资源。当多个用户并发地存取数据时，在数据库中就会产生多个事务同时存取同一数据的情况。若对并发操作不加控制就可能会读取和存储不正确的数据，破坏数据库的一致性。加锁是实现数据库并发控制的一个非常重要的技术。在实际应用中经常会遇到的与锁相关的异常情况，当两个事务需要一组有冲突的锁，而不能将事务继续下去的话，就会出现死锁，严重影响应用的正常执行。

在数据库中有两种基本的锁类型：**排它锁**（Exclusive Locks，即**X锁**）和**共享锁**（Share Locks，即**S锁**）。当数据对象被加上排它锁时，其他的事务不能对它读取和

修改。加了共享锁的数据对象可以被其他事务读取，但不能修改。数据库利用这两种基本的锁类型来对数据库的事务进行并发控制。

（1）悲观锁

悲观锁是指假设并发更新冲突会发生，所以不管冲突是否真的发生，都会使用锁机制。

悲观锁会完成以下功能：锁住读取的记录，防止其它事务读取和更新这些记录。其它事务会一直阻塞，直到这个事务结束.

悲观锁是在使用了数据库的事务隔离功能的基础上，独享占用的资源，以此保证读取数据一致性，避免修改丢失。

悲观锁可以使用Repeatable Read事务，它完全满足悲观锁的要求。

（2）乐观锁

乐观锁不会锁住任何东西，也就是说，它不依赖数据库的事务机制，乐观锁完全是应用于系统层面的东西。

如果使用乐观锁，那么数据库必须加版本字段，否则只能比较所有字段，但因为浮点类型不能比较，所以实际上没有版本字段是不可行的。

（3）排他锁

可以防止并发事务对资源进行访问。排他锁不与其他任何锁兼容。使用排他锁是，其他任何事务都无法修改数据；仅使用NOLOCK提示或未提交隔离级别时才会进行读取操作。

10.关系型数据库和非关系型数据库的区别

关系型数据库，是指采用了关系模型来组织数据的数据库。 简单来说，关系模型指的就是二维表格模型，而一个关系型数据库就是由二维表及其之间的联系所组成的一个数据组织。

关系模型中常用的概念：

关系：可以理解为一张二维表，每个关系都具有一个关系名，就是通常说的表名

元组：可以理解为二维表中的一行，在数据库中经常被称为记录

属性：可以理解为二维表中的一列，在数据库中经常被称为字段

域：属性的取值范围，也就是数据库中某一列的取值限制

关键字：一组可以唯一标识元组的属性，数据库中常称为主键，由一个或多个列组成

关系模式：指对关系的描述。其格式为：关系名(属性1，属性2， ... ... ，属性N)，在数据库中成为表结构

关系型数据库的优点：

容易理解：二维表结构是非常贴近逻辑世界的一个概念，关系模型相对网状、层次等其他模型来说更容易理解

使用方便：通用的SQL语言使得操作关系型数据库非常方便

易于维护：丰富的完整性(实体完整性、参照完整性和用户定义的完整性)大大减低

了数据冗余和数据不一致的概率

关系型数据库瓶颈

1).高并发读写需求

网站的用户并发性非常高，往往达到每秒上万次读写请求，对于传统关系型数据库来说，硬盘I/O是一个很大的瓶颈

2).海量数据的高效率读写

网站每天产生的数据量是巨大的，对于关系型数据库来说，在一张包含海量数据的表中查询，效率是非常低的

3).高扩展性和可用性

在基于web的结构当中，数据库是最难进行横向扩展的，当一个应用系统的用户量和访问量与日俱增的时候，数据库却没有办法像web server和app server那样简单的通过添加更多的硬件和服务节点来扩展性能和负载能力。对于很多需要提供24小时不间断服务的网站来说，对数据库系统进行升级和扩展是非常痛苦的事情，往往需要停机维护和数据迁移。

非关系型数据库提出另一种理念，例如，以键值对存储，且结构不固定，每一个元组可以有不一样的字段，每个元组可以根据需要增加一些自己的键值对，这样就不会局限于固定的结构，可以减少一些时间和空间的开销。使用这种方式，用户可以根据需要去添加自己需要的字段，这样，为了获取用户的不同信息，不需要像关系型数据库中，要对多表进行关联查询。仅需要根据id取出相应的value就可以完成查询。但非关系型数据库由于很少的约束，他也不能够提供像SQL所提供的where这种对于字段属性值情况的查询。并且难以体现设计的完整性。他只适合存储一些较为简单的数据，对于需要进行较复杂查询的数据，SQL数据库显的更为合适。

依据结构化方法以及应用场合的不同，主要分为以下几类：

1).面向高性能并发读写的key-value数据库：key-value数据库的主要特点就是具有极高的并发读写性能，Redis,Tokyo Cabinet,Flare就是这类的代表

2).面向海量数据访问的面向文档数据库：这类数据库的特点是，可以在海量的数据中快速的查询数据，典型代表为MongoDB以及CouchDB

3).面向可扩展性的分布式数据库：这类数据库想解决的问题就是传统数据库存在可扩展性上的缺陷，这类数据库可以适应数据量的增加以及数据结构的变化

关系型数据库 V.S. 非关系型数据库

关系型数据库的最大特点就是事务的一致性：传统的关系型数据库读写操作都是事务的，具有ACID的特点，这个特性使得关系型数据库可以用于几乎所有对一致性有要求的系统中，如典型的银行系统。

但是，在网页应用中，尤其是SNS应用中，一致性却不是显得那么重要，用户A看到的内容和用户B看到同一用户C内容更新不一致是可以容忍的，或者说，两个人看到同一好友的数据更新的时间差那么几秒是可以容忍的，因此，关系型数据库的最大特

点在这里已经无用武之地，起码不是那么重要了。

相反地，关系型数据库为了维护一致性所付出的巨大代价就是其读写性能比较差，而像微博、facebook这类SNS的应用，对并发读写能力要求极高，关系型数据库已经无法应付(在读方面，传统上为了克服关系型数据库缺陷，提高性能，都是增加一级memcache来静态化网页，而在SNS中，变化太快，memchache已经无能为力了)，因此，必须用新的一种数据结构存储来代替关系数据库。

关系数据库的另一个特点就是其具有固定的表结构，因此，其扩展性极差，而在SNS中，系统的升级，功能的增加，往往意味着数据结构巨大变动，这一点关系型数据库也难以应付，需要新的结构化数据存储。

于是，非关系型数据库应运而生，由于不可能用一种数据结构化存储应付所有的新的需求，因此，非关系型数据库严格上不是一种数据库，应该是一种数据结构化存储方法的集合。

必须强调的是，数据的持久存储，尤其是海量数据的持久存储，还是需要一种关系**数据库**这员老将。

11.了解nosql（非关系型数据库）

12.数据库三种范式，根据场景设计数据表（通过手绘ER图）

（1）第一范式（原子性）：第一范式是最基本的范式，数据库表的字段都是单一属性不可拆分；如果每列（或者每个属性）都是不可再分的最小数据单元（也成为最小的原子单元），则满足第一范式。例如：顾客表（姓名、编号、地址.....）其中地址哈可以细分为国家、省、市、区

（2）第二范式（确保表中的每列都和主键相关）：如果一个关系满足第一范式，并且除了主键以外的其他列都依赖与该主键，则满足第二范式。

例如：订单表（订单编号、产品编号、订购日期、价格、.....）“订单编号”和主键列没有直接关系，即“订单编号”列不依赖于主键列，应该删除该列。

（3）第三范式（确保每列都和主键列直接相关，而不是间接相关）

如果一个关系满足第二范式，并且除了主键意外的其他列都不依赖与主键列，则满足第三范式。

根据

概括起来就是三句话：

（1）第一范式：字段不可分

（2）第二范式：有主键，非主键字段依赖主键

（3）第三范式：非主键字段不能相互依赖

解释：

（1）第一范式：原子性 字段不可再分，否则不是关系型数据库

（2）第二范式：唯一性 一个表只说明有一个事物

（3）第三范式：每列都与主键有直接关系，不存在传递依赖

例子：

（2）

不符合第二范式的例子:

表:学号, 姓名, 年龄, 课程名称, 成绩, 学分;

这个表明显说明了两个事务:学生信息, 课程信息;

**********存在问题: **********

数据冗余，每条记录都含有相同信息；
删除异常：删除所有学生成绩，就把课程信息全删除了；
插入异常：学生未选课，无法记录进数据库；
更新异常：调整课程学分，所有行都调整。

**********修正:**********

学生：Student(学号, 姓名, 年龄)；

课程：Course(课程名称, 学分)；

选课关系：SelectCourse(学号, 课程名称, 成绩)。

满足第2范式只消除了插入异常。

（3）

&&&&♥♥♥♥♥♥♥♥不符合第三范式的例子: ♥♥♥♥♥♥♥♥&&&&&&&&&&

学号, 姓名, 年龄, 所在学院, 学院联系电话，关键字为单一关键字"学号";

存在依赖传递: (学号) → (所在学院) → (学院地点, 学院电话)

存在问题:

数据冗余:有重复值；

更新异常：有重复的冗余信息，修改时需要同时修改多条记录，否则会出现数据不一致的情况

删除异常

修正：
学生：(学号, 姓名, 年龄, 所在学院)；
学院：(学院, 地点, 电话)。

13.数据库的主从复制

14.使用explain优化sql和索引

15.long_query怎么解决

16.内连接、外连接、交叉连接、笛卡尔积

Mysql表的几种连接方式：

TableA

id	name
1	t1
2	t2
4	t4

TableB

id	age
1	18
2	20
3	19

表的连接方式有三种（1）外连接（2）内连接（3）交叉连接

（1）外连接：包括3种①左外连接②右外连接③全外连接，对应的sql关键字是LEFT/RIGHT/FULL OUTER JOIN,通常我们都省略OUTER关键字，写成LEFT/RIGHT/FULL JOIN。在左、右连接中都会以一种表为基表，基表中的所有行、列都会显示，外表如果和条件不匹配则所有外表列值都为NULL。圈外连接则所有表的行、列都会显示，条件不匹配的值皆为NULL。

①左外连接的例子

SELECT * FROM tableA left join tableB on tableA.id=table.id

id	name	id	age
1	t1	1	18
2	t2	2	20
4	t4	NULL	NULL

表A中所有的行列都显示了，第三行的条件不匹配的所有表B的值都为NULL

②右外连接的例子

SELECT * FROM tableA right join tableB on tableA.id=table.id

id	name	id	age
1	t1	1	18
2	t2	2	20
NULL	NULL	3	19

表B中的所有行列都显示了，表A的第三行条件不匹配的行的值都为NULL

③全外连接的例子

SELECT * FROM tableA full join tableB on tableA.id=table.id

id	name	id	age
1	t1	1	18
2	t2	2	20
NULL	NULL	3	19
4	t4	NULL	NULL

表A和表B的所有行列都显示了，条件不匹配的行的值为NULL

（2）内连接

内连接是用比较运算符比较要连接的列的值的连接，不匹配的行不会被显示。关键字是JOIN或INNER JOIN。例子：SELECT * from tableA JOIN tableB on tableA.id=tableB.id

id	name	id	age
1	t1	1	18
2	t2	2	20

注释：只返回条件匹配的行

以上写法等效于

SELECT * FROM tableA，tableB where tableA.id=tableB.id

SELECT * FROM tableA cross join tableB where tableA.id=tableB.id（**cross join后只能用where不能用on**）

（3）交叉连接

概念：没有where条件的交叉连接将产生连接表所涉及的笛卡尔积，即tableA*tableB行数的结果集（tableA3行*tableB3行=9行）

```
select * from TableA cross join TableB
```

id	name	id	age
1	t1	1	18
2	t2	1	18
4	t4	1	18
1	t1	2	20
2	t2	2	20
4	t4	2	20
1	t1	3	19
2	t2	3	19
4	t4	3	19

17.MVCC机制

（1）MVCC是一种多版本并发控制机制

（2）MVCC是为了解决什么问题

①大多数MYSQL事务型存储引擎，如InooDB,Falcon以及PBXT都不使用一种简单的行锁机制事实上，他们都和MVCC（多版本并发控制）一起使用

②大家应该都知道，锁机制可以控制并发操作，但是其系统开销较大，而MVCC可以在大多数情况下代替行级锁，使用MVCC可以在大多数情况下替代行级锁，使用MVCC，能降低其系统开销。

（3）MVCC实现

MVCC是通过保存数据在某个时间点的快照来实现，不同存储引擎的MVCC实现是不同的，典型的有乐观并发控制和悲观并发控制。

（4）MVCC具体实现分析（以InnoDB的MVCC实现来分析MVCC是怎样进行并发控制的）

InnoDB的MVCC，是通过在每行记录后面保存两个隐藏的列来实现的，这两个列，分别保存了这个行的创建时间，另一个保存的是行的删除时间。这里存储的并不是具体的时间值，而是系统的版本号（可以理解为事务的ID），每开始一个新的事务，系统版本号就会自动递增，事务开始的时刻的系统的版本号作为事务的ID，下面看一下REPEATABLE READ隔离级别下，MVCC是如何操作的

18.根据具体场景，说明版本控制机制

19.死锁怎么解决

20.varchar和char的使用场景

21.mysql并发情况怎么解决（通过事务、隔离级别、锁）

查询

2018年6月18日 20:24

（一）使用子查询

1.子查询就是嵌套在其它查询内的查询，例如

```
SELECT cust_id
FROM orders
WHERE order_num IN (SELECT order_num
                    FROM orderitems
                    WHERE prod_id = 'TNT2');
```

在SELECT语句中，子查询总是从内向外处理

2.使用子查询的另一方法是创建计算字段

（二）联结表：联结是利用SQL的SELECT能执行的最重要的操作，很好地理解联结及其语法是学习SQL的一个极其重要的组成部分。

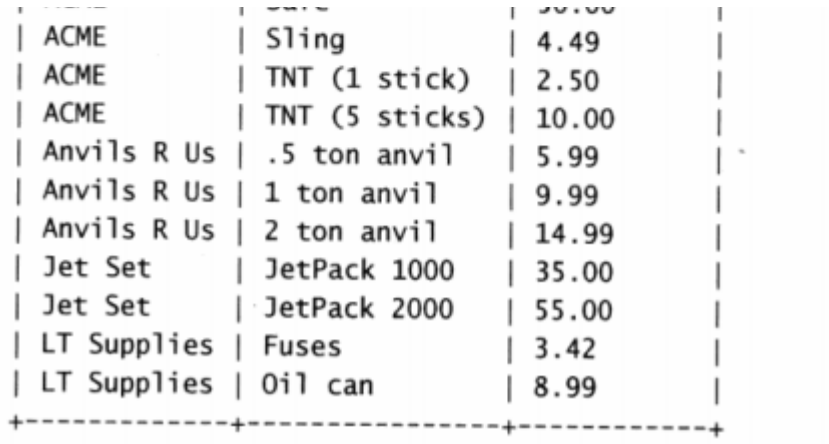
1.外键：外键是某个表的一列，它包含另一个表的另一个主键值，定义了两个表之间的关系。

例子：假设一个包含产品目录的数据库表，其中每种类别的物品占一行。对于每种物品要存储的信息包括产品描述和价格，以及生产该产品的供应商信息。可以建立两张表，一个存储供应商信息（vendors），另一个存储产品信息（products）。vendors中的主键是供应商ID，产品信息表只存储产品信息，vendors表的主键又叫做products的外键，它将vendors表与products表相关联，利用供应商ID能从ventors表中找出供应商的详细信息。

```
SELECT vend_name, prod_name, prod_price
FROM vendors, products
WHERE vendors.vend_id = products.vend_id
ORDER BY vend_name, prod_name;
+-------------+----------------+------------+
| vend_name   | prod_name      | prod_price |
+-------------+----------------+------------+
| ACME        | Bird seed      | 10.00      |
| ACME        | Carrots        | 2.50       |
```



```
| ACME        | Detonator      | 13.00      |
| ACME        | Safe           | 50.00      |
| ACME        | Sling          | 4.49       |
| ACME        | TNT (1 stick)  | 2.50       |
```



ACME	Sling	4.49
ACME	TNT (1 stick)	2.50
ACME	TNT (5 sticks)	10.00
Anvils R Us	.5 ton anvil	5.99
Anvils R Us	1 ton anvil	9.99
Anvils R Us	2 ton anvil	14.99
Jet Set	JetPack 1000	35.00
Jet Set	JetPack 2000	55.00
LT Supplies	Fuses	3.42
LT Supplies	Oil can	8.99

2.请记住，在一条SELECT语句中联结几个表时，相应的关系是在运行中构造的。在数据库表的定义中不存在将MySQL连接的东西，在连接两个表的时候，实际上做的是将第一个表 中的每一行与第二个表中的每一行配对，WHERE作为过滤条件，它只包含匹配给定的条件（联结条件）的行。没有WHERE语句，第一个表的每一行将与第二个表的每一行配对，不管逻辑上是否可以配在一起

3.笛卡尔积：没有连接条件的表关系返回的结果为笛卡尔积。检索出的行的数目将是第一个表的行数乘第二个表的行数。（所以不要忘了WHERE条件，不然MySQL将返回比想要的数据多的多的数据，同理，应该保证WHERE子句的正确性）

（三）使用高级连接

1.使用表别名

好处（1）缩短SQL语句

（2）允许在单条SELECT语句中多次使用相同的表