1.索引

1.1：索引是什么

索引（Index）是帮助 MySQL 高效获取数据的数据结构，是一种排好序的数据结构

1.2：为什么要用索引（优点）

1. 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。
2. 可以大大加快 数据的检索速度（大大减少的检索的数据量），这也是创建索引的最主要的原因。
3. 帮助服务器避免排序和临时表。
4. 将随机IO变为顺序IO
5. 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。

1.3：索引这么多优点，为什么不对表中的每一个列创建一个索引呢？（缺点）

1. 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. 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。
3. 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。

1.4：索引的主要原理，常用算法

通过不断地缩小想要获取数据的范围来筛选出最终想要的结果，同时把随机的事件变成顺序的事件，也就是说，有了这种索引机制，我们可以总是用同一种查找方式来锁定数据。

1.5：索引使用场景

1. 为经常出现在关键字order by. group by. distinct后面的字段 ,建立索引。
2. 在union等集合操作的结果集字段上,建立索引。
3. 为经常用作查询选择的字段,建立索引.
4. 在经常用作表连接的属性上,建立索引。
5. 考虑使用索引覆盖。

6. 对数据很少被更新的表,如果用户经常只查询其中的几个字段，可以考虑在这几个字段上建立索引,从而将表的扫描改变为索引的扫 描。

1.6：创建索引原则：

1. 对于查询频率高的字段创建索引；
2. 对排序、分组、联合查询频率高的字段创建索引；
3. 索引的数目不宜太多
    - 因为每创建一个索引都会占用相应的物理控件，过多的索引会导致insert、update、delete语句的执行效率降低；
4. 若在实际中，需要将多个列设置索引时，可以采用多列索引
5. 选择唯一性索引

唯一性索引的值是唯一的，可以更快速的通过该索引来确定某条记录。例如，学生表中学号是具有唯一性的字段。为该字段建立唯一性索引可以很快的确定某个学生的信息。如果使用姓名的话，可能存在同名现象，从而降低查询速度。

6. 尽量使用数据量少的索引

如果索引的值很长，那么查询的速度会受到影响。

7. 尽量使用前缀来索引

如果索引字段的值很长，最好使用值的前缀来索引。例如，TEXT和BLOG类型的字段，进行全文检索会很浪费时间。如果只检索字段的前面的若干个字符，这样可以提高检索速度。

8. 删除不再使用或者很少使用的索引

1.7：创建索引的注意事项

1. 限制表上的索引数目。
2. 避免在取值朝一个方向增长的字段（例如：日期类型的字段）上，建立索引；对复合索引，避免将这种类型的字段放置在最前面。由于字段的取值总是朝一个方向增长，新记录总是存放在索引的最后一个叶页中，从而不断地引起该叶页的访问竞争、新叶页的分配、中间分支页的拆分。此外，如果所建索引是聚集索引，表中数据按照索引的排列顺序存放，所有的插入操作都集中在最后一个数据页上进行，从而引起插入“热点”。
3. 对复合索引，按照字段在查询条件中出现的频度建立索引。在复合索引中，记录首先按照第一个字段排序。对于在第一个字段上取值相同的记录，系统再按照第二个字段的取值排序，以此类推。因此只有复合索引的第一个字段出现在查询条件中，该索引才可能被使用。因此将应用频度高的字段，放置在复合索引的前面，会使系统最大可能地使用此索引，发挥索引的作用。
4. 删除不再使用，或者很少被使用的索引。表中的数据被大量更新，或者数据的使用方式被改变后，原有的一些索引可能不再被需要。数据库管理员应当定期找出这些索引，将它们删除，从而减少索引对更新操作的影响。

1.8：为什么索引能够提高查询速度

如果我们写select * from user where name = 'xxx'这样没有进行任何优化的sql语句，默认会这样做：

定位到记录所在的页：需要遍历双向链表，找到所在的页

从所在的页内中查找相应的记录：由于不是根据主键查询，只能遍历所在页的单链表了

很明显，在数据量很大的情况下这样查找会很慢！这样的时间复杂度为O（n）。

使用索引之后，其实就是通过二分查找的思想将无序的数据变成有序(相对)

1.9：创建索引的三种方式

1.10：最左前缀原则

以最左边的为起点任何连续的索引都能匹配上。

（1）如果第一个字段是范围查询需要单独建一个索引；

（2）在创建多列索引时，要根据业务需求，where子句中使用最频繁的一列放在最左边；

当创建(a,b,c)复合索引时，想要索引生效的话，只能使用 a和ab、ac和abc三种组合！

1.11：最左匹配原则

最左优先，以最左边的为起点任何连续的索引都能匹配上。同时遇到范围查询(>、<、between、like)就会停止匹配。

2.索引的分类

2-1：索引的分类

1. 主键索引
2. 二级索引
    - 唯一索引
    - 普通索引
    - 前缀索引
    - 全文索引

2-2：各种索引定义

1. 主键索引：数据表的主键列使用的就是主键索引。

在mysql的InnoDB的表中，当没有显示的指定表的主键时，InnoDB会自动先检查表中是否有唯一索引的字段，如果有，则选择该字段为默认的主键，否则InnoDB将会自动创建一个6Byte的自增主键。

2. 二级索引(辅助索引)

因为二级索引的叶子节点存储的数据是主键。通过二级索引，可以定位主键的位置。

3. 唯一索引(Unique Key)：目的是为了该属性列的数据的唯一性，而不是为了查询效率。

4. 普通索引(Index)： 普通索引的唯一作用就是为了快速查询数据，一张表允许创建多个普通索引，并允许数据重复和NULL。

5. 前缀索引(Prefix)： 前缀索引只适用于字符串类型的数据。前缀索引是对文本的前几个字符创建索引，相比普通索引建立的数据更 小， 因为只取前几个字符。

6. 全文索引(Full Text)： 全文索引主要是为了检索大文本数据中的关键字的信息

7. 覆盖索引：如果一个索引包含（或者说覆盖）所有需要查询的字段的值

3.索引的结构

3-1：mysql索引的结构

1. B树索引与B+树索引

2. 聚簇索引与非聚簇索引

3. Hash索引

4. 全文索引

5. 空间索引

3-2：B+树比B树的优势

1. B+树空间利用率更高，可减少1/O次数

一般来说，索引本身也很大，不可能全部存储在内存中，因此索引往往以索引文件的形式存储的磁盘上。这样的话，索引查找过程中就要产生磁盘1O消耗。而因为B+树的内部节点只是作为索引使用，而不像B-树那样每个节点都需要存储硬盘指针。也就是说: B+树中每 个非叶子节点没有指向某个关健字具体信息的指针，所以好个节点可以存放更多的关键字数量，减少了I/0操作。

2. 增删文件(节点)时，效率更高 因为B+树的叶子节点包含所有关键字，并以有序的链表结构存储，这样可很好提高增删效率，基于范围查询更好。

3. B+树的查询效率更加稳定 因为B+树的每次查询过程中，都需要遍历从根节点到叶子节点的某条路径。所有关键字的查询路径长度相同，导致每一次查询的效率相 当。

3-3：B+树与红黑树比较

1. 更少的查找次数

复杂度和树高h相关，红黑树的树高h很明显比B+Tee大非常多，查找的次数也就更多。

2. 利用磁盘预读特性

为了减少磁盘IO操作，磁盘往往不是严格按需读取，而是每次都会预读。预读过程中，磁盘进行顺序读取，顺序读取不需要进行磁盘寻 道，并且只需要很短的旋转时间，速度会非常快。

3-4：B+树与hash索引比较

1. 如果是等值查询，那么哈希索引明显有绝对优势，因为只需要经过一次算法即可找到相应的键值。当然了，这个前提是，键值都是唯 一的。 如果键值不是唯的，就需要先找到该键所在位置，然后再根据链表往后扫描，直到找到相应的数据:

2. 如果是范围查询检索，原先是有序的键值，经过哈希算法后，有可能变成不连续的了，就没办法再利用索引完成范围查询检索:

3-5：聚簇索引与非聚簇索引概念

聚集索引即索引结构和数据一起存放的索引。主键索引属于聚集索引。

非聚集索引即索引结构和数据分开存放的索引。

3-6：聚簇索引的优缺点

一、优点

由于数据都是紧密相连，数据库不用从多个数据块中提取数据，所以节省了大量的io操作。

二、缺点

1. 对于mysql数据库目前只有innodb数据引擎支持聚簇索引，而Myisam并不支持聚簇索引。
2. 由于数据物理存储排序方式只能有一种，所以每个Mysql的表只能有一个聚簇索引。一般情况下就是该表的主键。
3. 为了充分利用聚簇索引的聚簇的特性，所以innodb表的主键列尽量选用有序的顺序id，而不建议用无序的id，比如uuid这种。

3-7：非聚簇索引的优缺点

一、优点

更新代价比聚集索引要小，非聚集索引的叶子节点是不存放数据的

二、缺点

非聚集索引也依赖于有序的数据 可能会二次查询(回表) :这应该是非聚集索引最大的缺点了。 当查到索引对应的指针或主键后，可能还需要根据指针或主键再到数据文件或表中查询。

4.数据库优化

4-1：为什么要优化

1. 避免网站页面出现访问错误

   由于慢查询造成页面无法加载 由于阻塞造成数据无法提交 增加数据库的稳定性

2. 很多数据库问题都是由于低效的查询引起的

4-2：索引优化

书本

4-3：查询优化

书本

4-4：当MySQL单表记录数过大时，数据库的CRUD性能会明显下降，如何解决

1. 限定数据的范围 务必禁止不带任何限制数据范围条件的查询语句。比如：我们当用户在查询订单历史的时候，我们可以控制在一个月的范围内；
2. 读/写分离 经典的数据库拆分方案，主库负责写，从库负责读；
3. 垂直分区 根据数据库里面数据表的相关性进行拆分。
4. 水平分区

保持数据表结构不变，通过某种策略存储数据分片。这样每一片数据分散到不同的表或者库中，达到了分布式的目的。 水平拆分可以支撑非常大的数据量。

4-5：垂直分表

垂直拆分的优点： 可以使得列数据变小，减少I/O次数。此外，垂直分区可以简化表的结构，易于维护。

垂直拆分的缺点： 主键会出现冗余，需要管理冗余列，并会引起Join操作，可以通过在应用层进行Join来解决。此外，垂直分区会让 事务变得更加复杂

4-6：水平分表

支持非常大的数据量存储，应用端改造也少，但 分片事务难以解决 ，跨节点Join性能较差，逻辑复杂

一般是水平分库，数据库分片的两种常见方案：

1. 客户端代理： 分片逻辑在应用端，封装在jar包中，通过修改或者封装JDBC层来实现。
2. 中间件代理： 在应用和数据中间加了一个代理层。分片逻辑统一维护在中间件服务中。

4-7：分库分表，id如何处理

方式1——UUID：不适合作为主键，因为太长了，并且无序不可读，查询效率低。比较适合用于生成唯一的名字的标示 比如文件的名字。

方式2——数据库自增 id : 两台数据库分别设置不同步长，生成不重复ID的策略来实现高可用。这种方式生成的 id 有序，但是需要独 立部署数据库实例，成本高，还会有性能瓶颈。

方式3——利用 redis 生成 id : 性能比较好，灵活方便，不依赖于数据库。但是，引入了新的组件造成系统更加复杂，可用性降低，编 码更加复杂，增加了系统成本。

5. 事务

5-1：什么是事务

事务是逻辑上的一组操作，要么都执行，要么都不执行。

5-2：数据库事务特性

1. 原子性（Atomicity）： 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；
2. 一致性（Consistency）： 执行事务前后，数据保持一致，多个事务对同一个数据读取的结果是相同的；
3. 隔离性（Isolation）： 并发访问数据库时，一个用户的事务不被其他事务所干扰，各并发事务之间数据库是独立的；
4. 持久性（Durability）： 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。

3、什么是脏读？幻读？不可重复读？ 4、什么是事务的隔离级别？MySQL的默认隔离级别是什么？

5-3：四大隔离级别

1. 读取未提交： 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。
2. 读取已提交： 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。
3. 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。
4. 可串行化： 最高的隔离级别， 该级别可以防止脏读、不可重复读以及幻读。

5-4：并发事务带来什么问题

1. 脏读:
2. 不可重复读:
3. 幻读:

书本

6.数据库基础知识

6-1：为什么要使用数据库

6-2: 什么是SQL?

6-3：什么是MySQL?

6-4：数据库三大范式是什么

6-5：mysql有关权限的表都有哪几个

6-6：MySQL的binlog有有几种录入格式？分别有什么区别?

mysql引擎

MyISAM和InnoDB区别

MyISAM是MySQL的默认数据库引擎。虽然性能极佳，而且提供了大量的特性，包括全文索引、压缩、空间函数等，但MyISAM不支持事务和行级锁，而且最大的缺陷就是崩溃后无法安全恢复。不过， 5.5版本之后，MySQL引入了InnoDB（事务性数据库引擎）， MySQL 5.5版本后默认的存储引擎为InnoDB。 大多数时候我们使用的都是 InnoDB 存储引擎，但是在某些情况下使用 MyISAM 也是合适的比如读密集 的情况下。

比较9条

MyISAM和InnoDB存储引擎使用的锁：

MyISAM采用表级锁(table-level locking)。 InnoDB支持行级锁(row-level locking)和表级锁,默认为行级锁

表级锁和行级锁对比：

表级锁： MySQL中锁定 粒度最大 的一种锁，对当前操作的整张表加锁，实现简单，资源消耗也比较少，加锁快，不会出现死锁。其锁定粒度最大，触发锁冲突的概率最高，并发度最低，MyISAM和 InnoDB引擎都支持表级锁。 行级锁： MySQL中锁定 粒度最小 的一种锁，只针对当前操作的行进行加锁。 行级锁能大大减少数据库操作的冲突。其加锁粒度最小，并发度高，但加锁的开销也最大，加锁慢，会出现死锁。

InnoDB存储引擎的锁的算法：

Record lock：单个行记录上的锁 Gap lock：间隙锁，锁定一个范围，不包括记录本身 Next-key lock：record+gap 锁定一个范围，包含记录本身 渴死但是呢

数据库分片的两种常见方案：

1. 客户端代理： 分片逻辑在应用端，封装在jar包中，通过修改或者封装JDBC层来实现。
2. 中间件代理： 在应用和数据中间加了一个代理层。分片逻辑统一维护在中间件服务中。

池化

什么是池化设计思想。

这种设计会初始预设资源，解决的问题就是抵消每次获取资源的消耗，池化设计还包括如下这些特征：池子的初始值、池子的活跃值、池子的最大值等，这些特征可以直接映射到java线程池和数据库连接池的成员属性中。这篇文章对池化设计思想介绍的还不错，直接复制过来，避免重复造轮子了。

什么是数据库连接池?为什么需要数据库连接池?

数据库连接本质就是一个 socket 的连接。数据库服务端还要维护一些缓存和用户权限信息之类的 所以占用了一些内存。我们可以把数据库连接池是看做是维护的数据库连接的缓存，以便将来需要对数据库的请求时可以重用这些连接。为每个用户打开和维护数据库连接，尤其是对动态数据库驱动的网站应用程序的请求，既昂贵又浪费资源。在连接池中，创建连接后，将其放置在池中，并再次使用它，因此不必建立新的连接。如果使用了所有连接，则会建立一个新连接并将其添加到池中。 连接池还减少了用户必须等待建立与数据库的连接的时间。

分库分表

慢查询

什么是慢查询

它用来记录在MySQL中响应时间超过阀值的语句日志记录

应用

一条SQL语句执行得很慢的原因有哪些

要分两种情形：

1. 大多数情况是正常的，只是偶尔会出现很慢的情况。
    - 数据库在刷新脏页，例如 redo log 写满了需要同步到磁盘。
    - 执行的时候，遇到锁，如表锁、行锁。
2. 在数据量不变的情况下，这条SQL语句一直以来都执行的很慢。
    - 没有用上索引
    - 数据库选错了索引

为什么数据库会选错了索引

系统在执行的时候，会进行预测，是走 c 索引扫描的行数少，还是直接扫描全表扫描的行数少呢?

扫描全表的话，那么扫描的次数就是这个表的总行数了，假设为 n；而如果走索引 c 的话，我们通过索引 c 找到主键之后，还得再通过主键索引来找我们整行的数据，需要走两次索引，而且，我们也不知道符合 这个条件的数据有多少行，万一真的是n条，那就惨了，所以系统是有可能走全表扫描而不走索引的

系统如何进行预判主要依赖于索引的区分度来判断的，一个索引上不同的值越多，意味着出现相同数值的索引越少，意味着索引的区分度越高。

这个区分度也叫做基数，系统当然是不会遍历全部来获得一个索引的基数的，代价太大了，索引系统是通过遍历部分数据，也就是通过采样的方式，来预测索引的基数的

那么出现失误的地方就是采样，比如采样的那一部分数据刚好基数很小，然后就误以为索引的基数很小。然后，系统就不走索引了，直接走全部扫描了。

主要是由于统计的失误，导致系统没有走索引，而是走了全表扫描。