DB常见问题

TODO：幻像读的解决办法：index lock和间隙锁还要具体了解一下。

存储模型

OLTP：事务处理多，互联网常见，经常查询，适合行存储。

OLAP：将数据用来分析，大批量数据，数据仓库，适合列存储。

HTAP：混合

为什么不用OS而要自己管理磁盘

DBMS自己管理有这些好处：

1. 将脏page按照合适的顺序写入磁盘
2. 特制化的预读取顺序。

3. buffer替换策略可以定制化
4. 多线程/多进程的处理和schedule。

ACID是什么？怎么实现？C和CAP中的有什么区别？

1. 原子性(Atomicity)。事务中的操作要不都执行成功，要不都不执行。如果部分操作失败，则会回退到未执行的情况。

    1. 实现原理：实现原子性的关键，是当事务回滚时能够撤销所有已经成功执行的sql语句。InnoDB实现回滚靠的是undo log，当事务对数据库进行修改时，InnoDB会生成对应的undo log。如果事务执行失败或调用了rollback，导致事务需要回滚，便可以利用undo log中的信息将数据回滚到修改之前的样子。

2. 一致性(Consistency)。指在事务发生前后，数据库始终保持一致性状态，前后都符合数据库完整性约束。相比较于CAP中的C强调的是外部一致性，即多个设备或者存储之间的一致性。

    - 保证原子性、持久性和隔离性，如果这些特性无法保证，事务的一致性也无法保证。
    - 数据库本身提供保障，例如不允许向整形列插入字符串值、字符串长度不能超过列的限制等。
    - 应用层面进行保障，例如如果转账操作只扣除转账者的余额，而没有增加接收者的余额，无论数据库实现的多么完美，也无法保证状态的一致

3. 隔离性(Isolation)。要求每个事务的读写对象和其他事务的操作对象能相互分离，即该事务提交前对其他事务都不可见。通过锁来实现。

4. 持久性(Durability)。事务一旦提交，结果是永久性的，即使发生宕机也能回复。

    1. 将修改写入磁盘。但是磁盘IO慢，所以有buffer pool。buffer pool定期写入磁盘，或者被换出的时候写入磁盘。**但是如果还没来得及将buffer pool中的page写入磁盘就宕机了，就出问题，所以有redo log**。
    2. 当数据修改时，除了修改Buffer Pool中的数据，还会在redo log记录这次操作。当事务提交时，会调用fsync接口对redo log进行刷盘。如果MySQL宕机，重启时可以读取redo log中的数据，对数据库进行恢复。redo log采用的是WAL（Write-ahead logging，预写式日志），所有修改先写入日志，再更

新到Buffer Pool，保证了数据不会因MySQL宕机而丢失，从而满足了持久性要求。

3. redo log写入磁盘为什么更快？redo log是追加操作，顺序写入。刷脏是随机IO。刷脏是以page为单位，redo log只需写入需要的部分。

事务的隔离级别是什么？要怎么实现？为了解决什么问题？

可能造成的问题

1. 脏读取：事务A修改了某个数据，事务B读取了，事务A abort了，导致事务B读取到了脏数据。
2. 不可重复读：事务A读取了某个数据之后，进行了其他操作，然后这个数据被B修改了，事务A再读取的时候这个数据已经被修改了。
3. 幻象读：事务A读取某个数据之后，数据的行数发生了变化。与不可重复读的区别是幻象读是数据的行数被修改，不可重复读是数据被修改。

解决方式（隔离级别）

1. 读取未提交（Read Uncommited，RU）：会碰到上面的三种问题。**读不加锁**，只在需要写数据的时候加排他锁，写完就unlock。
2. 读取已提交（Read Commited，RC）：解决了脏读取的问题。**读的时候加共享锁，读完就解共享锁。写的时候加排它锁，commit之后再解排他锁**。
3. 可重复读（Read Repeatable，RR）：解决了不可重复读的问题。严格两阶段锁协议（strict 2PL），在加锁阶段只能加锁，一旦开始解锁就进入shrinking阶段，该阶段就只能解锁，不能再加锁了。其实就相当于**共享锁也在commit之后了**。

Buffer Pool可能碰到的问题和优化方式

缓冲池(buffer pool)，这次彻底懂了！！！ - 掘金

磁盘管理是怎样的：参考disk_manager部分的课程笔记。

不同的索引

Join的实现

AVL，红黑树，B树，B+树，跳表，LSM树

红黑树、B(+)树、跳表、AVL等数据结构，应用场景及分析，以及一些英文缩写 - blcblc - 博客园

https://github.com/wardseptember/notes/blob/master/docs/B%E6%A0%91%E5%92%8CB%2B%E6%A0%91%E8%AF%A6%E8%A7%A3.md

https://www.zhihu.com/question/20202931

AVL

平衡二叉搜索树，左子树和右子树高度接近（差距不超过1），很严格。**写操作损耗很大，只适合读多写少的情况。**

红黑树

AVL的改进版，左子树和右子树不能超过彼此高度的两倍。没有AVL树严格，所以写操作损耗没有那么大，**性能比较均衡**，比较适合内存环境下。但是内存环境下又有了跳表。

B树，B+树

B树体系就是m阶的多叉树，一个节点最多有m-1个key，最少有ceil(m/2)-1个key，这就是它的平衡。

B树在每个节点都会存储value，key和value放在一起，B+树只在叶子节点存储value，并且叶子节点之间彼此链接起来。

读写时间复杂度都为logn

B+树相比于B树优势：

1. 搜索时间更平衡稳定。
2. 更便于范围查询，模糊查询。叶子节点穿起来了，找到开始节点然后往后遍历就好了。

3. 索引节点可以放更多的索引(m)，导致树的高度更小，查找时间（I/o次数）更小，cache命中率也会更高。

B+树相比于红黑树优势：

1. 支持范围查询，模糊查询等。
2. 在磁盘存储型数据库中，I/O次数很重要。B+树一个节点一次I/o，只需树的层树的I/o就可以找到，而红黑树层数高，需要多次I/O。

跳表

跳表就是在链表上加索引，相当于链表的二分查找。

时间复杂度为3*logn，因为索引高度为logn。

空间复杂度为O(n)，即n/2, n/4...., 4, 2，等比数列求和可以得到和为n-2，也可以让索引变得更稀疏，但是总体空间复杂度大概就是这个量级。

跳表与B+树的比较：

1. **跳表适合内存型数据库，B+树适合磁盘型数据库**。因为跳表的高度比B+树高，如果在磁盘里I/O次数多。
2. 如果都在内存中，在增加或者删除时，跳表不涉及页分裂等复杂的操作，跳表只需要用随机数增加索引就行。**跳表写入数据效率高**。

跳表与红黑树比较：

他两都适合内存中使用，但是有区别。

1. 跳表更适合范围查询。
2. 跳表实现更简单。
3. 跳表在多线程时表现更好。因为红黑树有平衡操作，涉及许多节点，所以锁竞争会更激烈。

LSM树

两个部分，内存部分和外存部分。**增删改（写操作）都只需要内存部分，写操作都是追加的，写操作性能极快。但是读操作是短板**，没找到就去下一层找，可能需要遍历，多次I/O。所以优化读操作是重点。还有重点的compaction，合并操作，防止数据冗余过多。这也比较耗时，也需要优化。

B+树是in-place update, LSM树是out-place update。前者原地更新，后者追加更新。**前者适合磁盘，后者适合分布式存储和固态硬盘SSD**。

读操作优化：

1. 使用SSD。SSD相比于磁盘，没有寻道时间，读性能大大提升。并且SSD是闪存，闪存不能覆盖写，需要擦除才能写入，而LSM树的追加写适合这种模式，减少了擦除，增加了SSD寿命。

Undo log, Redo log, MVCC