MYSQL索引与算法

Innodb存储引擎支持B+树，全文，哈希索引(自适应哈希)。

B+树索引的构造类似于二叉树，根据健值快速找到数据。(B+树的B不是代表二叉binary,而是代表平衡balance)因为B+树是从最早的平衡二叉树演变来的，但是B+树不是一个二叉树。B+树索引并不能找到一个约定健值的具体行，它找到的只是被查找数据行所在的页。然后数据库通过把页读入到内存，再内存中进行查找，最后得到所要的数据。

由于B树的树深度，为了起到平衡所以出来了个平衡二叉树。目的是用左旋或右旋来达到平衡。因此对一棵平衡树的维护是有开销的，不过平衡二叉树多用于内存结构对象中，所以维护开锁相对较小。

B+树是为磁盘或其它直接存取辅助设备设计的一种平衡查找树。在B+树中，所有记录节点都是按健值的大小顺序存放在同一层的叶子上，由各叶子节点指针进行连接。一般高度在2到4层，也就是说查找某一健值的行记录时最多只要2到4次IO，当前机械硬盘每秒最少可以100次IO，也就是意味着0.02～0.04秒每次。

B+树索引可分为聚集索引(clustered index)和辅助索引(secondary index)

聚集索引

Innodb存储引擎表是索引组织表，表中数据按照主健顺序存放，而聚集索引就是按每张表的主健构造一棵B+树，同时叶子节点中放的为整张表的行记录数据，也将聚集索引的叶子节点称为数据页。每个数据页都通过一个双向链表进行连接。

show index from tb_name可以看见里面的各项。如：Cardinality(存储引擎层进统计的)值很送健，优化器会根据这个值来判断是否使用这个索引。这个值不是时时更新所以不太正确，只是一个大约的值。可以用ANALYZE TABLE进行整理。(非高峰时操作)，如果表中1/16的数据发生过变化或stat_modified_counter(innodb内部计数器，更新update时增加)>2000 000 000时会更新，innodb的叶子节点(left page)默认为8个**innodb_stats_sample_page**可以控制，是根据随机这left page来进行预估的，所以每次show index from …的值是不一定一样的。

innodb_stats_method：用来判断如何对待索引中出现的NULL值记录。默认为nulls_equal表示null值视为相等的记录。有效值还有nulls_unequal,nulls_ignored。分别表示将null值视为不同的记录和忽略NULL值记录。如某页中索引记录为NULL NULL 1 2 2 3 3 3在参数innodb_stats_method为nulls_equal时该页的Caridinality为4，如果为nulls_unequal则为5，如果是nulls_ignored则Cardinality为3

Multi-Range Read优化

mysql5.6开始支持，目的是为了减少磁盘的随机访问，MRR优化适合于range ref eq_ref，有EXTR里机看到using MRR

MRR的好处有：

1 使数据访问变得较为顺序，查询辅助索引时，先根据得到查询结果，按照主健进行排序，并按主健排序的顺序进行书签查找。

2 减少缓冲池中页被替换的次数

3 批量处理对健值的查询操作

对于InnoDB和MyISAM存储引擎的范围查询和JOIN查询操作，MRR的工作方式如下：
1 把查询得到的辅助索引健值存放于一个缓存中，这时缓存中的数据是根据辅助索引健值排序的。
2 把缓存中的健值根据RowID进行排序
3 根据RowID的排序顺序来访问实际的数据文件
如果innodb myisam的缓冲泡不够大，就不能存放下一张表中的所有数据，此时频繁的离散读操作还会导致缓存中的页被替换出缓冲池，然后又不断地被读入缓冲泡。若是按主健顺序进行访问，则可以将此重复行为降为最低。
optimizer_switch:中的mrr为on时表示MRR启用。
read_rnd_buffer_size:
SELECT * FROM t WHERE key_part1 >=1000 AND key_part1 < 2000 AND key_part2 = 1000;(key_part1,key_part2)是联合索引
以上SQL如没开启MRR时查询类型为Range，优化器会先将key_part1大于1000并小于2000的取出，即使key_part2不等于1000的，取出后再根据key_part2的条件进行过滤。这会导致无用数据被取出。如启动MRR时优化器会将查询条件分为(1000,100),(1001,1000),(1002,1000).....(1999,1000)
开启MRR：查看是否开启mrr=on 开启：set @@optimizer_switch='mrr=on,mrr_cost_based=off'

Index Condition Pushdown(ICP)优化

ICP优化支持range ref eq_ref ref_or_null类型的查询 extra里会看到Using index condition
也是5.6以后支持。当进行索引查询时，首先根据索引来查找记录，然后再根据WHERE条件来过滤记录，在支持ICP后，MYSQL数据库会在取出索引的同时，判断是否可以进行WHERE条件的过滤，也就是将WHERE的部分过滤操作放在了存储引擎层。在某些查询下，可减少上层SQL层对记录的索取(fetch)，从而提高数据库的性能。

锁(分为lock与latch)

latch被称为闩(shuan)锁(轻量级锁)，如果时间太长性能会非常差，latch又分为互斥量(mutex)和读写锁(rwlock)。目的是用来保证并发线程操作临界资源的正确性，并通常没有死锁检测机制。show engine innodb status可查看。
lock的对象是事务，用来锁定的是数据库中的对象，如表、页、行。并一般lock的对象仅在事务commit或rollback后进行释放(不同事务隔离级别释放时间可能不同)。此外lock是有死锁机制的。

锁的类型

innodb引擎实现了以下两种标准的行级锁：
共享锁(S Lock)：允许事务读一行数据
排他锁(X Lock)：允许事务删除或更新一行数据
如一个事务T1已经获得行r的S共享锁，那另外事务T2可以立即获得行r的共享锁，因为读取没有改变行r的数据，这称为锁兼容(Lock Compatible)。但若有其他事务T3想获得r的X排他锁，则必须等事务T1、T2释放行r上的共享S锁，这种情况叫做锁不兼容。特别注意的是S和X锁都是行锁，美容是指对同一记录(row)锁的美容情情况。
innodb支持多粒度锁定(granular),还支持一种额外的锁方式，称为意向锁(Intention

Lock)
一致性非锁定读：innodb默认的，是通过MVCC来控制的，如果读取的行正在执行DELETE或UPDATE时，这时读取操作不会因此去等待上锁的释放，innodb会去读取行的一个快照数据。之称为非锁定是因为不用要等待访问的行上X锁的释放。快照是指该行的之前版本数据，该实现是通过undo段来完成，而undo用来在事务中回滚数据。因此快照数据是没有额外开销的。但在不同隔离状态下是不一样的。在READ COMMITTED和REPERTABLE READ(存储引擎的默认事务隔离级别)下，对于快照数据的定义却不同，如下：
SELECT @@TX_ISOLATION 查看当前默认的级别
READ COMMITTED级别下：对于快照数据，非一致性读总是读取被锁定行的最新一份快照数据。
REPEAABLE READ级别下：对于快照数据非一致性读总是读取事务开始时的行数据版本。
一致性锁定读：显示地对数据库读取操作进行加锁保证数据逻辑的一致性。对于SELECT语句支持两种锁定读操作：
SELECT … FRO UPDATE：对行记录加一个X锁，其它事务不能对已锁定的行加上任何锁。
SELECT … LOCK IN SHARE MODE：对读取的行记录加一个S锁，其它事务可以向被锁定的行加S锁，但如果加X锁则会被阻塞。
对于一致性非锁定读，即使读取的行被执行了SELECT FOR UPDATE也是可以进行读取的，和之前的一样。但是LOCK INSHAR MODE必须要在一个事务中。
innodb锁的3种算法
Record Lock:单个行记录上的锁(总是锁索引记录，如果没有innodb会使用隐式的主健来进行锁定)
Gap Lock:间隙锁，锁定一个范围，但不包含记录本身
Next-Key Lock:Gap Lock+Record Lock，锁定一个范围，并锁定记录本身。
当查询含有唯一属性时，innodb会对Next-Key Lock进行降级为Record Lock，即锁住索引本身，而不是范围。如下
如：一个表t如下，有一个a列为主健有三行记录 1 2 5在查询a=5 FRO UPDATE时可以进行insert 其它的数。
列名：a(主健) b(二级索)

1	1
3	1
5	3
7	6
10	8

因为在A会话中先对a=5进行X锁定，由于a是主健并唯一，所以锁定的仅是5这个值，而不是范围。所以next-key lock算法降级为record lock。但如果是辅助索引时就不一样了，如下：
在A会话中SELECT * FROM t WHERE b=3 FRO UPDATE很明显是对b进行查询，所以使用传统的next-key locking技术加锁，对于上面介绍如果对a列也就是对聚集索引所用的是record lock，而对于辅助索引其加上的则是next-key lock，锁定的范围

是(1,3),特别要注意的是innodb还会对辅助索引的下一个健值加上gap lock，即还有一个辅助索引范围为(3,6)的锁，所以在新会话B中运行下面SQL语句都会被阻塞：
SELECT * FROM t WHERE a=5 LOCK IN SHARE MODE(因为已经对聚集索引中列a=5加上了X锁)
INSERT 4,2(主健插入4没问题，但是插入的辅助索引值2在锁定范围(1,3)中)
INSERT 6,5(插入主健6没有被锁，5也不在1-3的范围中，但插入的值5在另一个锁的范围中(3,6))
以下是可以的，插入8,6、2,0、6,7都是没问题的
可以看到Gap Lock作用是为了防止多个事务将记录插入到同一范围内。
脏读：是指在不同的事务下，当前事务可以读到另外事务未提交的数据。简单说就是可以读到脏数据undo里的未commit的。但是和脏页并不是一回事。
不可重复读：是指一个事务内多次读取同一数据集合。在这个事务还没结束时，另外一个事务也访问该同一数据集合，并做了DML操作。因此，在第一个事务中的两次读数据之间，由于第二个事务的修改，那么第一个事务两次读到的数据可能是不一样的，就会在一个事务内两次读到的数据不一样，称为不可重复读。
丢失更新：一个事务的更新操作被另一个事务的更新操作覆盖，导致数据不一致。如：取钱，有100在一个窗口取90，因为网络或其它原因没提交，然后另一个窗口又取20，最后提交全成功。这样可以用froupdate来解决。