索引

> MySQL索引原理及慢查询优化

BTREE索引和HASH索引。MyISAM 和 InnoDB 存储引擎的表默认创建的都是 BTREE 索引。

mysql索引的种类

普通索引

这是最基本的索引，它没有任何限制。

唯一索引

与普通索引类似，不同的是：索引列的值唯一，但可以为空值。

主键索引

它是一种特殊的唯一索引，不允许有空值，一个表只能有一个主键。

组合索引

为了进一步榨取mysql的效率，就要考虑建立组合索引。

mysql建立索引的原则

1. 最左前缀匹配原则，非常重要的原则，mysql会一直向右匹配直到遇到范围查询(>、<、between、like)就停止匹配，比如a = 1 and b = 2 and c > 3 and d = 4 如果建立(a,b,c,d)顺序的索引，d是用不到索引的，如果建立(a,b,d,c)的索引则都可以用到，a,b,d的顺序可以任意调整。
2. =和in可以乱序，比如a = 1 and b = 2 and c = 3 建立(a,b,c)索引可以任意顺序，mysql的查询优化器会帮你优化成索引可以识别的形式
3. 尽量选择区分度高的列作为索引,区分度的公式是count(distinct col)/count(*)，表示字段不重复的比例，比例越大我们扫描的记录数越少，唯一键的区分度是1，而一些状态、性别字段可能在大数据面

前区分度就是0，那可能有人会问，这个比例有什么经验值吗？使用场景不同，这个值也很难确定，一般需要join的字段我们都要求是0.1以上，即平均1条扫描10条记录

4. 索引列不能参与计算，保持列“干净”，比如from_unixtime(create_time) = ’2014-05-29’就不能使用到索引，原因很简单，b+树中存的都是数据表中的字段值，但进行检索时，需要把所有元素都应用函数才能比较，显然成本太大。所以语句应该写成create_time = unix_timestamp(’2014-05-29’);
5. 尽量的扩展索引，不要新建索引。比如表中已经有a的索引，现在要加(a,b)的索引，那么只需要修改原来的索引即可

使用索引

1. 对于创建的多列索引,只要查询的条件中用到了最左边的列,索引一般就会被使用, 举例说明如下。
2. 对于使用 like 的查询,后面如果是常量并且只有%号不在第一个字符,索引才可能会 被使用，另外,如果如果 like 后面跟的是一个列的名字,那么索引也不会被使用。
3. 如果对大的文本进行搜索,使用全文索引而不用使用 like ‘%...%’。
4. 如果列名是索引,使用column_name is null将使用索引。如下例中查询name为null 的记录就用到了索引。

存在索引但不使用索引

1. 用 or 分割开的条件,如果 or 前的条件中的列有索引,而后面的列中没有索引, 那么涉及到的索引都不会被用到
2. 如果不是索引列的第一部分
3. 如果 like 是以%开始
4. 如果列类型是字符串,那么一定记得在 where 条件中把字符常量值用引号引 起来,否则的话即便这个列上有索引,MySQL 也不会用到的

mysql事务

MySQL 支持对 MyISAM 和 MEMORY 存储引擎的表进行表级锁定,对 BDB 存储引擎的表进行 页级锁定,对 InnoDB 存储引擎的表进行行级锁定。默认情况下,表锁和行锁都是自动获得 的,不需要额外的命令。但是在有的情

况下,用户需要明确地进行锁表或者进行事务的控制, 以便确保整个事务的完整性,这样就需要使用事务控制和锁定语句来完成。

脏读：事务 A 读取了事务 B 未提交的数据，并在这个基础上又做了其他操作。
不可重复读：事务 A 读取了事务 B 已提交的更改数据。
幻读：事务 A 读取了事务 B 已提交的新增数据。

SQL安全问题

SQL优化

优化load语句

优化insert语句

1. 如果同时从同一客户插入很多行,尽量使用多个值表的 INSERT 语句,这种方式将大大缩减客户端与数据库之间的连接、关闭等消耗,使得效率比分开执行的单个 INSERT 语 句快(在一些情况中几倍)。下面是一次插入多值的一个例子:
   insert into test values(1,2),(1,3),(1,4)...
2. 如果从不同客户插入很多行,能通过使用 INSERT DELAYED 语句得到更高的速度。 DELAYED 的含义是让 INSERT 语句马上执行,其实数据都被放在内存的队列中,并没有真正写入磁盘,这比每条语句分别插入要快的多;LOW_PRIORITY 刚好相反,在所有其他用户对表的读写完后才进行插入;
3. 将索引文件和数据文件分在不同的磁盘上存放(利用建表中的选项);
4. 如果进行批量插入,可以增加 bulk_insert_buffer_size 变量值的方法来提高速度,但是这只能对 MyISAM 表使用;
5. 当从一个文本文件装载一个表时,使用 LOAD DATA INFILE。这通常比使用很多 INSERT 语句快 20倍。

优化group by 语句

默认情况下,MySQL 对所有 GROUP BY col1,col2....的字段进行排序。如果查询包括 GROUP BY 但用户想要避免排序结果的消耗,则可以指定 ORDER BY NULL 禁止排序。

优化order by 语句

在某些情况中，MySQL 可以使用一个索引来满足 ORDER BY 子句，而不需要额外的排序。WHERE 条件和 ORDER BY 使用相同的索引，并且 ORDER BY 的顺序和索引顺序相同，并且 ORDER BY 的字段都是升序或者都是降序。

例如，下列 SQL 可以使用索引。

```
SELECT * FROM t1 ORDER BY key_part1,key_part2,... ;
SELECT * FROM t1 WHERE key_part1=1 ORDER BY key_part1 DESC, key_part2 DESC;
SELECT * FROM t1 ORDER BY key_part1 DESC, key_part2 DESC;
```

但是在以下几种情况下则不使用索引：

```
SELECT * FROM t1 ORDER BY key_part1 DESC, key_part2 ASC;
--order by 的字段混合 ASC 和 DESC
SELECT * FROM t1 WHERE key2=constant ORDER BY key1;
--用于查询行的关键字与 ORDER BY 中所使用的不相同
SELECT * FROM t1 ORDER BY key1, key2;
--对不同的关键字使用 ORDER BY：
```

Paste_Image.png

优化嵌套查询

子查询可以被更有效率的连接(JOIN)替代。

优化or条件

对于含有 OR 的查询子句,如果要利用索引,则 OR 之间的每个条件列都必须用到索引; 如果没有索引,则应该考虑增加索引。

mysql锁

1. 表级锁:开销小,加锁快;不会出现死锁;锁定粒度大,发生锁冲突的概率最高,并发度最低。
2. 行级锁:开销大,加锁慢;会出现死锁;锁定粒度最小,发生锁冲突的概率最低,并发 度也最高。
3. 页面锁:开销和加锁时间界于表锁和行锁之间;会出现死锁;锁定粒度界于表锁和行锁 之间,并发度一般。

MyISAM锁是表级锁。InnoDB是行级锁。
MyISAM 在执行查询语句(SELECT)前,会自动给涉及的所有表加读锁,在执

行更新操作 (UPDATE、DELETE、INSERT 等)前,会自动给涉及的表加写锁,这个过程并不需要用户干 预,因此,用户一般不需要直接用 LOCK TABLE 命令给 MyISAM 表显式加锁

表锁兼容性:

表 20-1　　MySQL 中的表锁兼容性

请求锁模式 / 是否兼容 / 当前锁模式	None	读锁	写锁
读锁	是	是	否
写锁	是	否	否

Paste_Image.png

意向锁是 InnoDB 自动加的,不需用户干预。对于 UPDATE、DELETE 和 INSERT 语句,InnoDB 会自动给涉及数据集加排他锁(X);对于普通 SELECT 语句,InnoDB 不会加任何锁;事务可 以通过以下语句显示给记录集加共享锁或排他锁。

1. 共享锁(S):SELECT * FROM table_name WHERE ... LOCK IN SHARE MODE。
2. 排他锁(X):SELECT * FROM table_name WHERE ... FOR UPDATE。

请求锁模式 / 是否兼容 / 当前锁模式	X	IX	S	IS
X	冲突	冲突	冲突	冲突
IX	冲突	兼容	冲突	兼容
S	冲突	冲突	兼容	兼容
IS	冲突	兼容	兼容	兼容

InnoDB 行锁模式兼容性列表

如果一个事务请求的锁模式与当前的锁兼容,InnoDB 就将请求的锁授予该事务;反之, 如果两者不兼容,该事务就要等待锁释放。

InnoDB行锁实现特点：只有通过索引条件检索数据，InnoDB才使用行级锁，否则，InnoDB将使用表锁。

1. 在不通过索引条件查询的时候,InnoDB 确实使用的是表锁,而不是行锁。
2. 由于 MySQL 的行锁是针对索引加的锁,不是针对记录加的锁,所以虽然是访问不同行 的记录,但是如果是使用相同的索引键,是会出现锁冲突的。
3. 当表有多个索引的时候,不同的事务可以使用不同的索引锁定不同的行,另外,不论 是使用主键索引、唯一索引或普通索引,InnoDB 都会使用行锁来对数据加锁。
4. 即便在条件中使用了索引字段,但是否使用索引来检索数据是由 MySQL 通过判断不同 执行计划的代价来决定的,如果 MySQL 认为全表扫描效率更高,比如对一些很小的表,它 就不会使用索引,这种情况下 InnoDB 将使用表锁,而不是行锁。因此,在分析锁冲突时, 别忘了检查 SQL 的执行计划,以确认是否真正使用了索引。

最后对意向锁做简单的说明：
innoDB为什么要设置意向锁?
<small>答：如果没意向锁，需要表锁时，要一行行检查某个表是否发生了行锁，进而判断能否表锁成功，如果有了意向锁，不用一个个去检查，直接从表的层次就可判断。</small>

间隙锁（Next-Key锁）
当我们用范围条件而不是相等条件检索数据,并请求共享或排他锁时,InnoDB 会给符合条件的已有数据记录的索引项加锁;对于键值在条件范围内但并不存在的记录,叫做“间隙 (GAP)”,InnoDB 也会对这个“间隙”加锁,这种锁机制就是所谓的间隙锁(Next-Key 锁)。 举例来说,假如emp表中只有101条记录,其empid的值分别是 1,2,...,100,101,下面的SQL:
Select * from emp where empid > 100 for update;
是一个范围条件的检索,InnoDB 不仅会对符合条件的 empid 值为 101 的记录加锁,也会对 empid 大于 101(这些记录并不存在)的“间隙”加锁。

InnoDB 使用间隙锁的目的,一方面是为了防止幻读,以满足相关隔离级别的要求,对于上 面的例子,要是不使用间隙锁,如果其他事务插入了 empid 大于 100 的任何记录,那么本 事务如果再次执行上述语句,就会发生幻读;另外一方面,是为了满足其恢复和复制的需要。

关于死锁

总结

对于 MyISAM 的表锁,主要讨论了以下几点:

1. 共享读锁(S)之间是兼容的,但共享读锁(S)与排他写锁(X)之间,以及排他 写锁(X)之间是互斥的,也就是说读和写是串行的。
2. 在一定条件下,MyISAM 允许查询和插入并发执行,我们可以利用这一点来解决 应用中对同一表查询和插入的锁争用问题。
3. MyISAM 默认的锁调度机制是写优先,这并不一定适合所有应用,用户可以通过 设置 LOW_PRIORITY_UPDATES 参数,或在 INSERT、UPDATE、DELETE 语句中指定 LOW_PRIORITY 选项来调节读写锁的争用。
4. 由于表锁的锁定粒度大,读写之间又是串行的,因此,如果更新操作较多,MyISAM 表可能会出现严重的锁等待,可以考虑采用 InnoDB 表来减少锁冲突。

对于 InnoDB 表,主要讨论了以下几项内容:

1. InnoDB 的行锁是基于索引实现的,如果不通过索引访问数据,InnoDB 会使用表锁。
2. 介绍了 InnoDB 间隙锁(Next-key)机制,以及 InnoDB 使用间隙锁的原因。
3. 在不同的隔离级别下,InnoDB 的锁机制和一致性读策略不同。MySQL 的恢复和复制对 InnoDB 锁机制和一致性读策略也有较大影响。
4. 锁冲突甚至死锁很难完全避免。
   在了解 InnoDB 锁特性后,用户可以通过设计和 SQL 调整等措施减少锁冲突和死锁,包括:
   a： 尽量使用较低的隔离级别;
   b: 精心设计索引,并尽量使用索引访问数据,使加锁更精确,从而减少锁冲突的机会;
   c: 选择合理的事务大小,小事务发生锁冲突的几率也更小;
   d: 给记录集显示加锁时,最好一次性请求足够级别的锁。比如要修改数据的话,最好 直接申请排他锁,而不是先申请共享锁,修改时再请求排他锁,这样容易产生死锁;
   e: 不同的程序访问一组表时,应尽量约定以相同的顺序访问各表,对一

个表而言,尽 可能以固定的顺序存取表中的行。这样可以大大减少死锁的机会;
f: 尽量用相等条件访问数据,这样可以避免间隙锁对并发插入的影响;
g: 不要申请超过实际需要的锁级别;除非必须,查询时不要显示加锁;
h: 对于一些特定的事务,可以使用表锁来提高处理速度或减少死锁的可能。