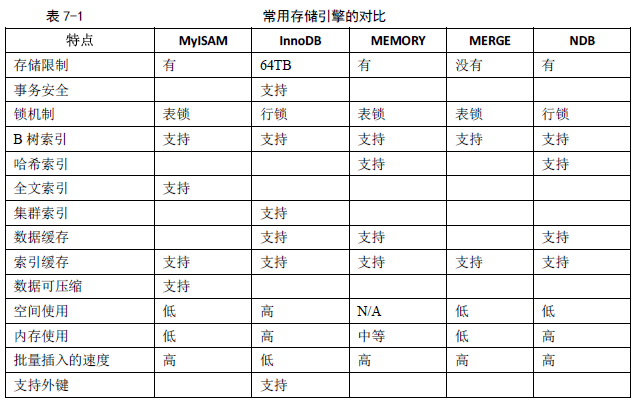
MyISAM和InnoDB的区别



MyISAM（默认存储引擎）：不支持事务、不支持外键，只支持行锁，优势访问速度快

适用环境：对事务完整性、并发性要求不高，或以批量select、insert为主的应用

InnoDB：支持事务，支持外键，支持行锁和表锁。

适用环境：对事务的完整性有比较高的要求，在并发条件下要求数据的一致性，数据操作除了插入和查询以外，还包括很多的更新、删除操作，那么InnoDB 存储引擎应该是比较合适的选择。

主要区别：

（1）索引实现方式不同：都采用B+树实现索引，MyISAM的叶节点存储的是指向数据项的指针，主索引和次索引保存指针都指向数据项。InnoDB主键索引为聚集索引，叶节点存储的是数据项。辅助索引叶节点存储的是主索引的相应记录主键的值。

（2）MyISAM不支持事务，InnoDB支持事务。

（3）锁的粒度不同：MyISAM只支持表锁，而InnoDB支持行锁和表锁。

数据库优化篇

1. SQL优化（索引、表分析和排序）

（1）优化SQL语句一般步骤：

A）通过show status命令了解各种sql的执行频率

B）定位执行效率较低的sql语句

通过慢查询日志定位那些执行效率较低的sql语句

C）通过EXPLAIN分析低效的SQL的执行计划

D）确定问题并采取相应的优化措施

（2）索引问题（索引是数据库优化中最常用也是最重要的手段之一）

可能使用索引的情况（记住：最左前缀匹配原则）

A）对于多列索引，只要查询条件中用到了最左边的列，索引一般就会使用

B）对于使用like的查询，不以%开头才会使用索引

C）如果对大的文本进行搜索，使用全文索引而不用使用like ‘%...%’

D）如果列名是索引，使用column\_name is null将使用索引

存在索引但不使用索引

A）如果mysql估计使用索引比全表扫描更慢，则不使用索引

B）用or分割开的条件

C）如果不是索引列的第一部分

D）如果like是以%开始

E）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

（3）常用优化

大批量插入数据

因为InnoDB 类型的表是按照主键的顺序保存的，所以将导入的数据按照主键的顺序排列，可以有效地提高导入数据的效率

优化insert语句

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

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

优化GROUP BY语句（排序问题引起）

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（排序与索引）

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

优化嵌套查询（表结构）

子查询可以被更有效率的连接（JOIN）替代

连接（JOIN）之所以更有效率一些，是因为MySQL 不需要在内存中创建临时表来完成这个逻辑上的需要两个步骤的查询工作

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

2.优化数据库对象

（1）优化表的数据类型

表需要使用何种数据类型，是需要根据应用来判断的。虽然应用设计的时候需要考虑字段的长度留有一定的冗余，但是不推荐让很多字段都留有大量的冗余，这样即浪费磁盘存储空间,同时在应用程序操作时也浪费物理内存。

（2）分表和分库

垂直拆分：即把主码和一些列放到一个表，然后把主码和另外的列放到另一个表中。

如果一个表中某些列常用，而另外一些列不常用，则可以采用垂直拆分，另外垂直拆分可以使得数据行变小，一个数据页就能存放更多的数据，在查询时就会减少I/O 次数。

其缺点是需要管理冗余列，查询所有数据需要联合（JOIN）操作

水平拆分：即根据一列或多列数据的值把数据行放到两个独立的表中

表很大，降低索引层数，降低了查询是需要读数据和索引的页数

表中的数据本来就有独立性

需要把数据存在多个介质上

（3）逆范式化

增加冗余列、增加派生列、重新组表、分割表

因为规范化越高，那么产生的关系就越多，关系过多的直接结果就是导致表之间的连接操作越频繁，而表之间的连接操作是性能较低的操作，直接影响到查询的速度，所以对于查询较多的应用就需要根据实际情况运用逆规范化对数据进行设计，通过逆规范化来提高查询的性能

（4）使用中间表提高统计查询速度

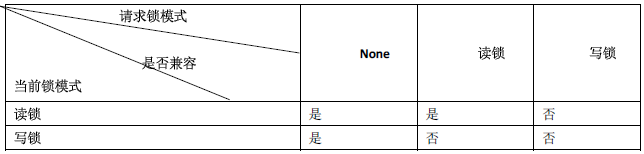
中间表复制源表部分数据，并且与源表相“隔离”，在中间表上做统计查询不会对在线应用产生负面影响

 中间表上可以灵活的添加索引或增加临时用的新字段,从而达到提高统计查询效率和辅助统计查询作

3.锁问题

（1）类型：表级锁、行级锁和页面锁

（2）表级锁的锁模式：表共享锁、表独占锁



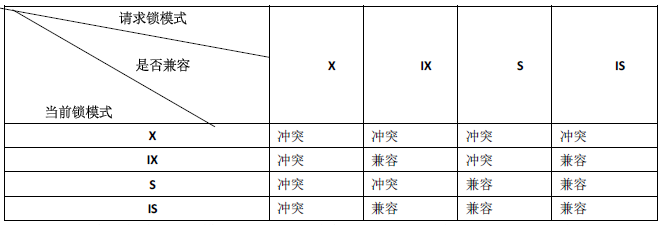
（3）MyISAM的锁问题（不支持事务）

死锁问题：表级锁，不会产生死锁，一次获取全部（待操作的）表锁

锁调度：读写锁互斥，写进程优先获得锁（即使读请求先到锁等待队列）

（4）InnoDB锁问题（支持事务）

a)行锁：共享锁（S）、排他锁（X）；表锁：意向共享锁（IS）、意向排他锁（IX）



note：

1.如果没有意向锁，加表锁，则需遍历每行判断是否存在行锁。反之，只需

判兼容性。

2. IX，IS是表级锁，不会和行级的X，S锁发生冲突。只会和表级的X，S发生

冲突。

（5）InnoDB行锁实现方式

InnoDB行锁是通过给索引项加锁来实现的

a)只有通过索引条件检索数据，InnoDB才使用行级锁，否则，InnoDB使用表锁！

b)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。

c) 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。

（6）间隙锁

**定义**：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）

**目的**：防止幻读

**note**：InnoDB 除了通过范围条件加锁时使用间隙锁外，如果使用相等条件请求

给一个不存在的记录加锁，InnoDB 也会使用间隙锁

事务

1. ACID属性：原子性、一致性、**隔离性**、持久性
2. 并发事务（事务隔离级别导致）带来的问题：更新丢失、脏读、不可重复读、幻读

**不可重复读**：一个事务在读取某些数据后的某个时间，再次读取以前读过的数据，却发现其读出的数据已经发生了改变、或某些记录已经被删除了！这种现象就叫做“不可重复读”。

**幻读：**一个事务按相同的查询条件重新读取以前检索过的数据，却发现其他事务插入了满足其查询条件的新数据，这种现象就称为“幻读”。

1. 事务隔离级别

