MySQL

- 执行流程
  - 1、查询缓存
  - 2、解析器生成解析树
  - 3、预处理再次生成解析树
  - 4、查询优化器
  - 5、查询执行计划
  - 6、查询执行引擎
  - 7、查询数据返回结果
- 表结构对性能的影响
  - 1、冗余数据的处理（可以提高系统的整体查询性能<三范式>） 3
  - 2、大表拆小表
    - 1、一般不会设计属性过多的表
    - 2、一般不会超过500到1000万数据的表
    - 3、有大数据的列单独拆为小表
  - 3、根据需求展示更加合理的表结构
  - 4、常用属性分离为小表
- 索引
  - 1、类型
    - 1,Normal:普通的索引;允许一个索引值后面关联多个行值;
    - 2,UNIQUE:唯一索引;允许一个索引值后面只能有一个行值;之前对列添加唯一约束其实就是为这列添加了一个unique索引;当我们为一个表添加一个主键的时候,其实就是为这个表主键列(设置了非空约束),并为主键列添加了一个唯一索引;
    - 3,Fulltext:全文检索,mysql的全文检索只能用myisam引擎,并且性能较低,不建议使用;
  - 2、方法
    - 1,b-tree:是一颗树(二叉树,平衡二叉树,平衡树(B-TREE))
      使用平衡树实现索引，是mysql中使用最多的索引类型；在innodb中，存在两种索引类型，第一种是主键索引（primary key），在索引内容中直接保存数据的地址；第二种是其他索引，在索引内容中保存的是指向主键索引的引用；所以在使用innodb的时候，要尽量的使用主键索引，速度非常快；

- 2,hash:把索引的值做hash运算，并存放到hash表中，使用较少，一般是memory引擎使用；优点:因为使用hash表存储，按照常理，hash的性能比B-TREE效率高很多。
  hash索引的缺点：
  1，hash索引只能适用于精确的值比较，=，in，或者<>；无法使用范围查询;
  2，无法使用索引排序；
  3，组合hash索引无法使用部分索引；
  4，如果大量索引hash值相同，性能较低；

- 3、创建
  - 1，较频繁的作为查询条件的字段应该创建索引；
  - 2，唯一性太差的字段不适合单独创建索引，即使频繁作为查询条件；
    作为索引的列,如果不能有效的区分数据,那么这个列就不适合作为索引列;比如(性别,状态不多的状态列)
    举例:SELECT sum(amount) FROM accountflow WHERE accountType = 0;
    假如把accountType作为索引列,因为accountType只有14种,所以,如果根据accountType来创建索引,最多只能按照1/14的比例过滤掉数据;但是,如果可能出现,只按照该条件查询,那我们就要考虑到其他的提升性能的方式了;
  - 3，更新非常频繁的字段不适合创建索引；原因,索引有维护成本;
  - 4，不会出现在WHERE 子句中的字段不该创建索引；
  - 5，索引不是越多越好;(只为必要的列创建索引)
    1,不管你有多少个索引,一次查询至多采用一个索引;(索引和索引之间是独立的)
    2,因为索引和索引之间是独立的,所以说每一个索引都应该是单独维护的;数据的增/改/删,会导致所有的索引都要单独维护;

- 4、执行计划与执行明细

- 1，Explain:可以让我们查看MYSQL执行一条SQL所选择的执行计划；

1，ID：执行查询的序列号；

2，select_type：使用的查询类型

1，DEPENDENT SUBQUERY：子查询中内层的第一个SELECT，依赖于外部查询的结果集；

2，DEPENDENT UNION：子查询中的UNION，且为UNION 中从第二个SELECT 开始的后面所有SELECT，同样依赖于外部查询的结果集；

3，PRIMARY：子查询中的最外层查询，注意并不是主键查询；

4，SIMPLE：除子查询或者UNION 之外的其他查询；

5，SUBQUERY：子查询内层查询的第一个SELECT，结果不依赖于外部查询结果集；

6，UNCACHEABLE SUBQUERY：结果集无法缓存的子查询；

7，UNION：UNION 语句中第二个SELECT 开始的后面所有SELECT，第一个SELECT 为PRIMARY

8，UNION RESULT：UNION 中的合并结果；

3，table：这次查询访问的数据表；

4，type：对表所使用的访问方式：

1，all：全表扫描

2，const：读常量，且最多只会有一条记录匹配，由于是常量，所以实际上只需要读一次；

3，eq_ref：最多只会有一条匹配结果，一般是通过主键或者唯一键索引来访问；

4，fulltext：全文检索，针对full text索引列；

5，index：全索引扫描；

6，index_merge：查询中同时使用两个（或更多）索引，然后对索引结果进行merge之后再读取表数据；

7，index_subquery：子查询中的返回结果字段组合是一个索引（或索引组合），但不是一个主键或者唯一索引；

8，rang：索引范围扫描；

9，ref：Join 语句中被驱动表索引引用查询；

10，ref_or_null：与ref 的唯一区别就是在使用索引引用查询之外再增加一个空值的查询；

11，system：系统表，表中只有一行数据；

12，unique_subquery：子查询中的返回结果字段组合是主键或者唯一约束；

5，possible_keys：可选的索引；如果没有使用索引，为null；

6，key：最终选择的索引；

7，key_len：被选择的索引长度；

8，ref：过滤的方式，比如const（常量），column（join），func（某个函数）；

9，rows：查询优化器通过收集到的统计信息估算出的查询条数；

10，Extra：查询中每一步实现的额外细节信息

1，Distinct：查找distinct 值，所以当mysql 找到了第一条匹配的结果后，将停止该值的查询而转为后面其他值的查询；

2，Full scan on NULL key：子查询中的一种优化方式，主要在遇到无法通过索引访问null值的使用使用；

3，Impossible WHERE noticed after reading const tables：MySQL Query Optimizer 通过收集到的统计信息判断出不可能存在结果；

4，No tables：Query 语句中使用FROM DUAL 或者不包含任何FROM 子句；

5，Not exists：在某些左连接中MySQL Query Optimizer 所通过改变原有Query 的组成而使用的优化方法，可以部分减少数据访问次数；

6，Select tables optimized away：当我们使用某些聚合函数来访问存在索引的某个字段的时候，MySQL Query Optimizer 会通过索引而直接一次定位到所需的数据行完成整个查询。当然，前提是在Query 中不能有GROUP BY 操作。如使用MIN()或者MAX（）的时候；

7，Using filesort：当我们的Query 中包含ORDER BY 操作，而且无法利用索引完成排序操作的时候，MySQL Query Optimizer 不得不选择相应的排序算法来实现。

8，Using index：所需要的数据只需要在Index 即可全部获得而不需要再到表中取数据；

9，Using index for group-by：数据访问和Using index 一样，所需数据只需要读取索引即可，而当Query 中使用了GROUP BY 或者DISTINCT 子句的时候，如果分组字段也在索引中，Extra 中的信息就会是Using index for group-by；

10，Using temporary：当MySQL 在某些操作中必须使用临时表的时候，在Extra 信息中就会出现Using temporary 。主要常见于GROUP BY 和ORDER BY 等操作中。

11，Using where：如果我们不是读取表的所有数据，或者不是仅仅通过索引就可以获取所有需要的数据，则会出现Using where 信息；

12，Using where with pushed condition：这是一个仅仅在NDBCluster 存储引擎中才会出现的信息，而且还需要通过打开Condition Pushdown 优化功能才可能会被使用。控制参数为engine_condition_pushdown 。

- 2，Profiling:可以用来准确定位一条SQL的性能瓶颈；

1，开启profiling：set profiling=1;
2，执行QUERY，在profiling过程中所有的query都可以记录下来；
3，查看记录的query：show profiles；
4，选择要查看的profile：show profile cpu, block io for query 6；
status是执行SQL的详细过程；
Duration：执行的具体时间；
CPU_user：用户CPU时间；
CPU_system：系统CPU时间；
Block_ops_in：IO输入次数；
Block_ops_out：IO输出次数；
profiling只对本次会话有效；

- JOIN的原理 2
- sql优化原则
  - 1、选择需要优化的SQL
  - 2、Explain和Profile入手 3
  - 3、永远用小结果集驱动大的结果集
  - 4、在索引中完成排序
  - 5、使用最小Columns
  - 6、使用最有效的过滤条件
  - 7、避免复杂的JOIN和子查询
- 锁
  - 1,、lock 4
  - 2、latch<轻量级锁，锁的时间非常短，用来操作临界资源>
  - 3、一致性的非锁定读
  - 4、一致性锁定读
  - 5、死锁
- 事务
  - 1、特性

- 1、原子性<数据库事务不可分割的单位，要么都做，要么都不做>
- 3、隔离性<事务是相互不可见的>
- 4、持久性<事务一旦提交，即使宕机也是能恢复的>
- 2、一致性<事务的操作不会改变数据库的状态，比方说唯一约束>

- 2、分类
  - 1、扁平事务<使用最频繁的事务，要么都成功提交，要么都失败回滚>
  - 2、带有扁平点的扁平事务<允许事务回滚到同一个事务中比较早的一个状态>
  - 3、链事务<回滚到最近的一个保存点，在所有的事务都提交之后才会释放锁，并且下一个事务的开始需要上一个事务来进行通知>
  - 4、嵌套事务<树结构，只有当父级事务提交之后子级事务才会提交，任意一个父级事务的回滚都会导致下面的子级事务回滚>
  - 5、分布式事务<操作两个不同的数据库，使其实现数据的同步，例如将中国银行的钱转到工商银行，这个不同银行的不同数据库，为分布式事务>
- 3、隔离级别
  - 1、read uncommittted<脏读：事务A读取了事务B更新的数据，然后B回滚操作，那么A读取到的数据是脏数据>
  - 2、read committed<不可重复读：事务 A 多次读取同一数据，事务 B 在事务A多次读取的过程中，对数据作了更新并提交，导致事务A多次读取同一数据时，结果 不一致。>
  - 3、repeatable read<幻读：系统管理员A将数据库中所有学生的成绩从具体分数改为ABCDE等级，但是系统管理员B就在这个时候插入了一条具体分数的记录，当系统管理员A改结束后发现还有一条记录没有改过来，就好像发生了幻觉一样，这就叫幻读。>
  - 4、serializable<锁表，不会出现意外情况>