MySQL高级

性能下降：1.查询语句写的烂 2.索引失效 3.关联查询太多join 4.服务器调优及参数设置（缓存，线程数等）

索引：

缺点：降低了更新表的速度，如对表进行insert，update，delete，因为更新表时，mysql不仅要保存数据，还要保存索引文件每次跟新添加索引列的字段

索引分类：

单值索引 唯一（主键）索引（索引唯一，非空值） 复合索引

基本语法：

create（unique）index indexName on table（columnname（length））；drop index （indexName）on table;

show index from table\_name

添加索引：alter TABLE table1\_name add （unique）index\_name（column\_list）

索引结构：

适合建索引

1. 主键自动建立唯一索引
2. 频繁作为查询条件字段创建索引

3.查询中与其他表关联的字段，外键关系建立索引

4.频繁更新字段不适合建立索引

5.where条件用不到的字段不建立索引

6.查询中排序的字段，建立索引大大提高排序速度

7.查询中统计或者分组字段

不适合建索引

1. 表记录少
2. 经常增删改的表
3. 某个数据包含了很多重复内容，为他建立索引没有太大的实际效果

性能优化：explain+sql语句

MySql常见瓶颈：

1. CPU：cpu在饱和的时候一般发生在数据装入内存或者从磁盘读取数据时
2. IO:磁盘I/O 瓶颈发生在装入数据远大于内存容量的时候
3. 服务器硬件性能瓶颈：top,free,iostat。。。查看系统性能状态

单表索引优化：

1. Where后面字段建索引
2. 范围以后索引失效，解决方案：绕开范围查询建索引

两表索引优化：

1. 左连接加右表
2. 右连接建左表

三表索引优化：

1. 小的结果集驱动大的结果集
2. 优先优化嵌套里面的内层循环
3. 保证join语句中被驱动表join条件字段已经被索引

**索引失效原因**

1. 全值匹配我最爱（最佳左前缀法则）
2. 最佳左前缀法则（查询从索引的最左前列开始并且不跳过索引中的列）简言之：带头大哥不能死
3. 不在索引列上做任何操作（计算，函数，（自动or手动）类型转换），会导致索引失效而转向全表扫描
4. 存储引擎不能使用索引中范围右边的列
5. 尽量使用覆盖索引（只访问索引的查询（索引列和查询列一致）），减少select \*
6. 使用不等于（！=或者<>）时候无法使用索引导致全表扫描
7. Is null,is not null也无法使用索引
8. Like以通配符（‘%abc....’）mysql索引失效会变成全表扫描的操作，所以 like写右边，%aa%用覆盖索引（完全一致）解决问题
9. 字符串不加单引号索引失效
10. 少用or，用它来连接时索引会失效

**实际SQL优化**

①观察，至少跑一天，看生产的慢sql情况

②开启慢查询日志，设置阙值，并抓取

③Explain+慢sql分析

④Show profile

⑤DBA进行sql数据库服务器参数调优

**排序优化**

1. 小表驱动大表

**In和exists**

Select \* from A where id in（select id from B（返回数据））

当B表数据集必须小于A表的数据集时，用in优于exists

Select \* from A where exists （select 1 from B where B.id=A.id（返回true或false））

当A表的数据集小于B表的数据集时，用exists优于in

1. Order by 优化

两种情况会使用index方式排序：

①order by 语句使用索引最左前列（order by a desc,b desc,c desc）

②使用where字句与order by字句条件组合满足索引最左前列

③如果where使用索引的最左前缀定义为常量，则order by能使用索引 where a=const order by b,c

1. Group by 优化

①group by实质是先排序后分组，遵循最左前缀

②无法使用索引时，增大参数设置

③where高于having，能用where限定的不用having限定

**存储过程**

Delimiter $$

Create procedure insert\_emp（in start int(10),in max\_num int(10)）

Begin

Declare i int default 0；

Set autocommit=0;

Repeat

Set i=i+1;

Insert into emp(............);

Until i=max\_num

End repeat;

Commit;

End $$;

调用存储过程

**Call** insert\_dept(100,10) 100开始插入10条

**锁机制**

操作类型：读锁，写锁，

粒度：行锁，表锁，页锁（了解）

手动添加表锁：lock table 表名 read（write）,表名2 read(write)，其他

**MyIsam--表锁**

**读锁（共享）**

Session1

lock table mylock read

Update mylock set name=’a2’ where id=1;（error）

Select \* from book（error）

Session2

Select \* from mylock

Select \* from book

Update mylock set name=’a3’ where id=1;（阻塞）

写锁（独占）

Session1

lock table mylock write

Select \* from mylock

Update mylock set name=’a4’ where id=1;

Select \* from book；（error）

Session2

Select \* from book;

Select \* from mylock;（阻塞）

**总结：读锁会阻塞写，不会阻塞读，写锁会把读和写都阻塞**

Show open tables（查看哪些表被锁）

**Innodb--行锁（5.5之后）**

**锁定某一行**

Begin;

Select \* from table where a=8 **for update**;

Commit;

Show status like ‘innodb\_row\_lock%’;

行锁优化建议

①尽可能让所有数据通过所有完成，避免无所有行锁升级为表锁

②合理设计索引，尽量缩小锁的范围

③尽可能较少检索条件，避免间隙锁

④尽量控制事物大小，减少锁定资源量和时间长度

⑤尽可能低级别事物隔离