sql性能优化简易方程

一.引起性能原因

1.cpu消耗

2.内存使用

3.磁盘，网络，I/O设备等操作

4.低效sql

二.sql调优

1.应用程序级调优

- a.sql语句优化
- b.管理变化调优

2.示例级调优

- a.内存调优
- b.数据结构
- c.实例配置

3.操作系统交互

- a.I/O
- b.swqp
- c.Parameters

三.优化方法

- 1.优化业务数据
- 2.优化数据设计
- 3.优化流程设计
- 4.优化sql语句
- 5.优化物理结构
- 6.优化内存分配
- 7.优化I/O
- 8.优化内存竞争
- 9.优化操作系统

四.sql优化过程

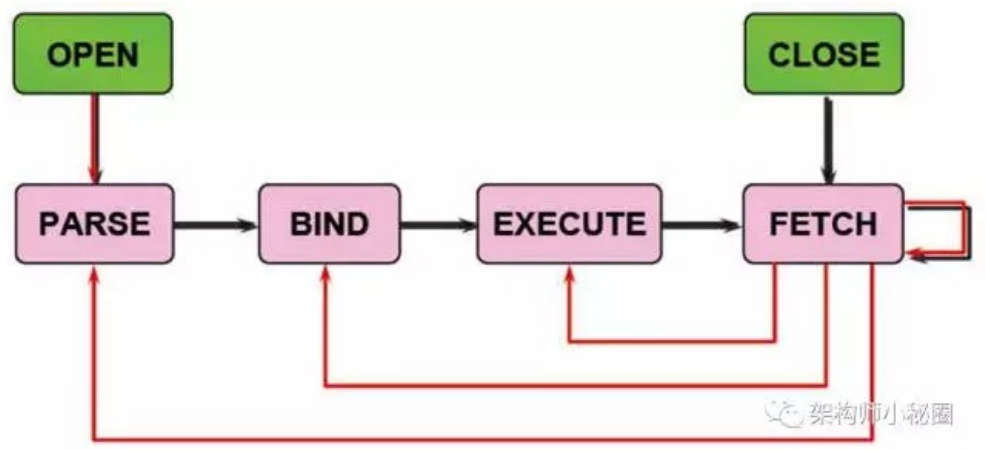
- 1.定位有问题的语句
- 2.检查执行计划
- 3.检查执行计划中优化器的统计信息
- 4.分析相关表的记录数、索引情况
- 5.改写sql、使用HINT、调整索引、表分析

五.什么是好的sql语句

- 1.简单、模块化
- 2.易读、易维护
- 3.节省资源
  - a.内存
  - b.cpu
  - c.扫描数据块少
  - d.少排序
- 4.不造成死锁

六.sql处理过程

四个阶段：



解析（PARSE）

1.检查语法

2.检查语义和相关权限

3.在共享池中查找sql语句

4.合并视图定义和子查询

5.确定执行计划

绑定（BIND）

1.在语句中查找绑定变量

2.赋值（重新赋值）

执行（EXECUTE）

1.应用执行计划

2.执行必要的I/O和排序操作

提取（FETCH）

1.重查询结果中返回记录

2.必要时进行排序

3.使用ARRAY FETCH机制

SQL的语法顺序：

select 【distinct】 ....from ....【xxx join】【on】....where....group by ....having....【union】....order by.....

SQL的执行顺序：

from ....【xxx join】【on】....where....group by ....avg()、sum()....having....select 【distinct】....order by.....

from 子句--执行顺序为从后往前、从右到左

表名(最后面的那个表名为驱动表，执行顺序为从后往前, 所以数据量较少的表尽量放后）

where子句--执行顺序为自下而上、从右到左

将可以过滤掉大量数据的条件写在where的子句的末尾性能最优

group by 和order by 子句执行顺序都为从左到右

select子句--少用*号，尽量取字段名称。 使用列名意味着将减少消耗时间。

优化实践

1.避免使用笛卡儿积

2.避免使用*

使用*相对比较低效，sql解析过程中还需将*依次转成列名，而转换需要通过查询数据字典完成。

3.使用where替换having

where子句在进行分组操作之前执行，

havaing子句在进行分组操作之后执行，

having在检索出所有记录后对结果集进行过滤，这个过程处理需要排序，总计等操作。

可以使用where限制记录数目，减小开销。

4.采用exists、not exists和in not in互换

in适用于外表大，内表小

exists适用于外表小，内表大

5.使用exists替代distinct

多表联查，使用distinct浪费资源，造成性能低下。

distinct需要对数据重新进行排序，而排序是最浪费资源的操作。

使用exists替代，在子查询中，一旦条件满足后立马返回结果，使查询变得更为迅速。

低效写法：

select distinct dept_no,dept_name from dept d,emp e where d.dept_no=e.dept_no

高效写法：

select dept_no,dept_name from dept d where exists (select 1 from emp e where e.dept_no=d.dept_no)

6.避免使用隐式数据转换

隐式数据类型转换不能适用索引，导致全表扫描。

例如：varchar类型，隐式转成int类型

select cid ，cname from person where cid = 123

select cid ，cname from person where cid = '123'

有些数据中会对类型进行严格限定，例如：postgresql不支持隐式转换。

7.使用索引避免排序操作

执行频度高，包含排序sql，建议使用索引排序。

排序操作相对比较昂贵，往往会消耗大量的系统资源，导致性能低下。

索引为有序结果，在order by后字段建立索引，将大大提高效率。

8.使用前端匹配模糊查询

采用前端匹配模糊查询，可进行索引扫描

但采用前后端模糊查询，将无法使用索引，导致全表扫描

9.避免在使用频率低的字段上建立索引。

在使用频率较低的地方建立索引，不但不会降低逻辑I/O,反而会增加大量逻辑I/O降低性能

10.避免对字段进行操作

例如：

select money from record where amount/30<1000

对任何列直接操作可能导致全表扫描，包括数据库函数，计算表达式等。

尽量不要对列进行操作

select money from record where amount<1000*30

11.尽量避免in，or

含有“in”，“or"的子句在where子句中会使得索引失效，可考虑才开子句。

用union替换

12.尽量去掉<>

全表扫描

可以使用or替代

13.避免使用is null 或者not

无法使用索引

且在不同数据库下对null的解释不一样，容易造成错乱

14.批量提交sql

锁表