**MySQL性能优化：**

链接地址：<http://www.cnblogs.com/easypass/archive/2010/12/08/1900127.html>

计算机系统硬件性能从高到代依次为：

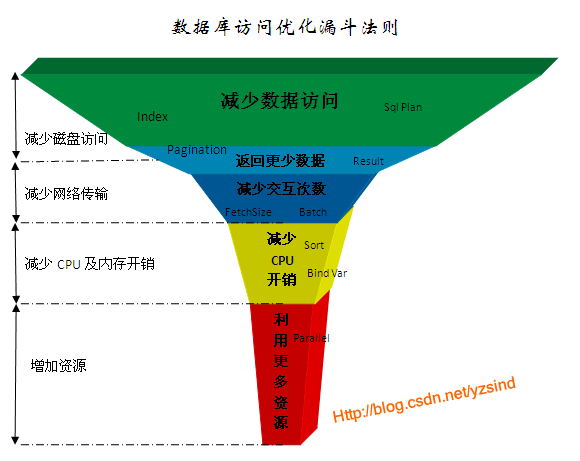
CPU——Cache(L1-L2-L3)——内存——SSD硬盘——网络——硬盘

每种硬件主要的工作内容：

CPU及内存：缓存数据访问、比较、排序、事务检测、SQL解析、函数或逻辑运算；

网络：结果数据传输、SQL请求、远程数据库访问（dblink）；

硬盘：数据访问、数据写入、日志记录、大数据量排序、大表连接。



**具体优化：**

**减少数据访问**

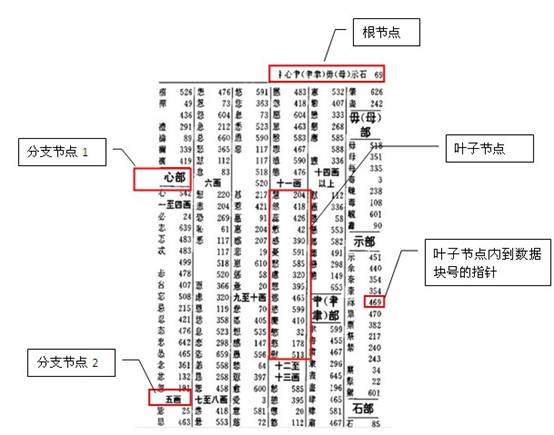
1. **创建并使用正确的索引**

索引有哪些种类？

常见的索引有B-TREE索引、位图索引、全文索引，位图索引一般用于数据仓库应用，全文索引由于使用较少。B-TREE索引包括很多扩展类型，如组合索引、反向索引、函数索引等等，以下是B-TREE索引的简单介绍：

B-TREE索引也称为平衡树索引(Balance Tree)，它是一种按字段排好序的树形目录结构，主要用于提升查询性能和唯一约束支持。B-TREE索引的内容包括根节点、分支节点、叶子节点。





叶子节点内容：索引字段内容+表记录ROWID

根节点，分支节点内容：当一个数据块中不能放下所有索引字段数据时，就会形成树形的根节点或分支节点，根节点与分支节点保存了索引树的顺序及各层级间的引用关系。

索引会大大增加表记录的DML(INSERT,UPDATE,DELETE)开销

SQL什么条件会使用索引？

当字段上建有索引时，通常以下情况会使用索引：

INDEX\_COLUMN = | > | >= | < | <= ?

INDEX\_COLUMN between ? and ?

INDEX\_COLUMN in (?,?,...,?)

INDEX\_COLUMN like ?||'%'（后导模糊查询）

T1. INDEX\_COLUMN=T2. COLUMN1（两个表通过索引字段关联）

SQL什么条件不会使用索引？

|  |  |
| --- | --- |
| 查询条件 | 不能使用索引原因 |
| INDEX\_COLUMN <> ?  INDEX\_COLUMN not in (?,?,...,?) | 不等于操作不能使用索引 |
| function(INDEX\_COLUMN) = ?  INDEX\_COLUMN + 1 = ?  INDEX\_COLUMN || 'a' = ? | 经过普通运算或函数运算后的索引字段不能使用索引 |
| INDEX\_COLUMN like '%'||?  INDEX\_COLUMN like '%'||?||'%' | 含前导模糊查询的Like语法不能使用索引 |
| INDEX\_COLUMN is null | B-TREE索引里不保存字段为NULL值记录，因此IS NULL不能使用索引 |
| NUMBER\_INDEX\_COLUMN='12345'  CHAR\_INDEX\_COLUMN=12345 | Oracle在做数值比较时需要将两边的数据转换成同一种数据类型，如果两边数据类型不同时会对字段值隐式转换，相当于加了一层函数处理，所以不能使用索引。 |
| a.INDEX\_COLUMN=a.COLUMN\_1 | 给索引查询的值应是已知数据，不能是未知字段值。 |
| 注：  经过函数运算字段的字段要使用可以使用函数索引，这种需求建议与DBA沟通。  有时候我们会使用多个字段的组合索引，如果查询条件中第一个字段不能使用索引，那整个查询也不能使用索引  如：我们company表建了一个id+name的组合索引，以下SQL是不能使用索引的  Select \* from company where name=?  Oracle9i后引入了一种index skip scan的索引方式来解决类似的问题，但是通过index skip scan提高性能的条件比较特殊，使用不好反而性能会更差。 | |

我们一般在什么字段上建索引？

这是一个非常复杂的话题，需要对业务及数据充分分析后再能得出结果。主键及外键通常都要有索引，其它需要建索引的字段应满足以下条件：

1、字段出现在查询条件中，并且查询条件可以使用索引；

2、语句执行频率高，一天会有几千次以上；

3、通过字段条件可筛选的记录集很小，那数据筛选比例是多少才适合？

这个没有固定值，需要根据表数据量来评估，以下是经验公式，可用于快速评估：

小表(记录数小于10000行的表)：筛选比例<10%；

大表：(筛选返回记录数)<(表总记录数\*单条记录长度)/10000/16

      单条记录长度≈字段平均内容长度之和+字段数\*2

以下是一些字段是否需要建B-TREE索引的经验分类：

|  |  |  |
| --- | --- | --- |
|  | 字段类型 | 常见字段名 |
| 需要建索引的字段 | 主键 | ID,PK |
| 外键 | PRODUCT\_ID,COMPANY\_ID,MEMBER\_ID,ORDER\_ID,TRADE\_ID,PAY\_ID |
| 有对像或身份标识意义字段 | HASH\_CODE,USERNAME,IDCARD\_NO,EMAIL,TEL\_NO,IM\_NO |
| 索引慎用字段,需要进行数据分布及使用场景详细评估 | 日期 | GMT\_CREATE,GMT\_MODIFIED |
| 年月 | YEAR,MONTH |
| 状态标志 | PRODUCT\_STATUS,ORDER\_STATUS,IS\_DELETE,VIP\_FLAG |
| 类型 | ORDER\_TYPE,IMAGE\_TYPE,GENDER,CURRENCY\_TYPE |
| 区域 | COUNTRY,PROVINCE,CITY |
| 操作人员 | CREATOR,AUDITOR |
| 数值 | LEVEL,AMOUNT,SCORE |
| 长字符 | ADDRESS,COMPANY\_NAME,SUMMARY,SUBJECT |
| 不适合建索引的字段 | 描述备注 | DESCRIPTION,REMARK,MEMO,DETAIL |
| 大字段 | FILE\_CONTENT,EMAIL\_CONTENT |

只通过索引访问数据

有些时候，我们只是访问表中的几个字段，并且字段内容较少，我们可以为这几个字段单独建立一个组合索引，这样就可以直接只通过访问索引就能得到数据，一般索引占用的磁盘空间比表小很多，所以这种方式可以大大减少磁盘IO开销。

这种只通过索引访问数据的方法一般只用于核心应用，也就是那种对核心表访问量最高且查询字段数据量很少的查询。

**返回更少的数据**

**1.数据分页处理**

一般数据分页方式有：

1.客户端(应用程序或浏览器)分页:

将数据从应用服务器全部下载到本地应用程序或浏览器，在应用程序或浏览器内部通过本地代码进行分页处理

优点：编码简单，减少客户端与应用服务器网络交互次数

缺点：首次交互时间长，占用客户端内存

适应场景：客户端与应用服务器网络延时较大，但要求后续操作流畅，如手机GPRS，超远程访问（跨国）等等。

2.应用服务器分页:

将数据从数据库服务器全部下载到应用服务器，在应用服务器内部再进行数据筛选。以下是一个应用服务器端Java程序分页的示例：

List list=executeQuery(“select \* from employee order by id”);

Int count= list.size();

List subList= list.subList(10, 20);

优点：编码简单，只需要一次SQL交互，总数据与分页数据差不多时性能较好。

缺点：总数据量较多时性能较差。

适应场景：数据库系统不支持分页处理，数据量较小并且可控。

1. 数据库SQL分页(一般为这种):

采用数据库SQL分页需要两次SQL完成 ,一个SQL计算总数量 ,一个SQL返回分页后的数据

优点：性能好

缺点：编码复杂，各种数据库语法不同，需要两次SQL交互。

**2.只返回需要的字段**

通过去除不必要的返回字段可以提高性能，例：

调整前：select \* from product where company\_id=?;

调整后：select id,name from product where company\_id=?;

优点：

1、减少数据在网络上传输开销

2、减少服务器数据处理开销

3、减少客户端内存占用

4、字段变更时提前发现问题，减少程序BUG

5、如果访问的所有字段刚好在一个索引里面，则可以使用纯索引访问提高性能。

缺点：增加编码工作量

由于会增加一些编码工作量，所以一般需求通过开发规范来要求程序员这么做，否则等项目上线后再整改工作量更大。

如果你的查询表中有大字段或内容较多的字段，如备注信息、文件内容等等，那在查询表时一定要注意这方面的问题，否则可能会带来严重的性能问题。如果表经常要查询并且请求大内容字段的概率很低，我们可以采用分表处理，将一个大表分拆成两个一对一的关系表，将不常用的大内容字段放在一张单独的表中。如一张存储上传文件的表：

T\_FILE（ID,FILE\_NAME,FILE\_SIZE,FILE\_TYPE,FILE\_CONTENT）

我们可以分拆成两张一对一的关系表：

T\_FILE（ID,FILE\_NAME,FILE\_SIZE,FILE\_TYPE）

T\_FILECONTENT（ID, FILE\_CONTENT）

         通过这种分拆，可以大大提少T\_FILE表的单条记录及总大小，这样在查询T\_FILE时性能会更好，当需要查询FILE\_CONTENT字段内容时再访问T\_FILECONTENT表。

**减少交互次数**

1. **batch DML**

数据库访问框架一般都提供了批量提交的接口，jdbc支持batch的提交处理方法，当你一次性要往一个表中插入1000万条数据时，如果采用普通的executeUpdate处理，那么和服务器交互次数为1000万次。如果采用批量提交模式，1000条提交一次，那么和服务器交互次数为1万次，交互次数大大减少。采用batch操作一般不会减少很多数据库服务器的物理IO，但是会大大减少客户端与服务端的交互次数，从而减少了多次发起的网络延时开销，同时也会降低数据库的CPU开销。

1. **In List**

很多时候我们需要按一些ID查询数据库记录，我们可以采用一个ID一个请求发给数据库，如下所示：

for :var in ids[] do begin

  select \* from mytable where id=:var;

end;

我们也可以做一个小的优化， 如下所示，用ID INLIST的这种方式写SQL：

select \* from mytable where id in(:id1,id2,...,idn);

通过这样处理可以大大减少SQL请求的数量，从而提高性能。那如果有10000个ID，那是不是全部放在一条SQL里处理呢？答案肯定是否定的。首先大部份数据库都会有SQL长度和IN里个数的限制，如ORACLE的IN里就不允许超过1000个值。

另外当前数据库一般都是采用基于成本的优化规则，当IN数量达到一定值时有可能改变SQL执行计划，从索引访问变成全表访问，这将使性能急剧变化。随着SQL中IN的里面的值个数增加，SQL的执行计划会更复杂，占用的内存将会变大，这将会增加服务器CPU及内存成本。

综合考虑，一般IN里面的值个数超过20个以后性能基本没什么太大变化，也特别说明不要超过100，超过后可能会引起执行计划的不稳定性及增加数据库CPU及内存成本，这个需要专业DBA评估。

1. **设置Fetch Size**

当我们采用select从数据库查询数据时，数据默认并不是一条一条返回给客户端的，也不是一次全部返回客户端的，而是根据客户端fetch\_size参数处理，每次只返回fetch\_size条记录，当客户端游标遍历到尾部时再从服务端取数据，直到最后全部传送完成。所以如果我们要从服务端一次取大量数据时，可以加大fetch\_size，这样可以减少结果数据传输的交互次数及服务器数据准备时间，提高性能。

 fetchsize并不会存在一个最优的固定值，因为整体性能与记录集大小及硬件平台有关。根据测试结果建议当一次性要取大量数据时这个值设置为100左右，不要小于40。注意，fetchsize不能设置太大，如果一次取出的数据大于JVM的内存会导致内存溢出。

1. **优化业务逻辑**

要通过优化业务逻辑来提高性能是比较困难的，这需要程序员对所访问的数据及业务流程非常清楚。

举一个案例：

某移动公司推出优惠套参，活动对像为VIP会员并且2010年1，2，3月平均话费20元以上的客户。

那我们的检测逻辑为：

select avg(money) as avg\_money from bill where phone\_no='13988888888' and date between '201001' and '201003';

select vip\_flag from member where phone\_no='13988888888';

if avg\_money>20 and vip\_flag=true then

begin

  执行套参();

end;

如果我们修改业务逻辑为:

select avg(money) as  avg\_money from bill where phone\_no='13988888888' and date between '201001' and '201003';

if avg\_money>20 then

begin

  select vip\_flag from member where phone\_no='13988888888';

  if vip\_flag=true then

  begin

    执行套参();

  end;

end;

通过这样可以减少一些判断vip\_flag的开销，平均话费20元以下的用户就不需要再检测是否VIP了。

如果程序员分析业务，VIP会员比例为1%，平均话费20元以上的用户比例为90%，那我们改成如下：

select vip\_flag from member where phone\_no='13988888888';

if vip\_flag=true then

begin

  select avg(money) as avg\_money from bill where phone\_no='13988888888' and date between '201001' and '201003';

  if avg\_money>20 then

  begin

    执行套参();

  end;

end;

这样就只有1%的VIP会员才会做检测平均话费，最终大大减少了SQL的交互次数。

以上只是一个简单的示例，实际的业务总是比这复杂得多，所以一般只是高级程序员更容易做出优化的逻辑，但是我们需要有这样一种成本优化的意识。

1. **使用ResultSet游标处理记录**

现在大部分Java框架都是通过jdbc从数据库取出数据，然后装载到一个list里再处理，list里可能是业务Object，也可能是hashmap。

   很多持久层框架为了尽量让程序员使用方便，封装了jdbc通过statement执行数据返回到resultset的细节，导致程序员会想采用分页的方式处理问题。实际上如果我们采用jdbc原始的resultset游标处理记录，在resultset循环读取的过程中处理记录，这样就可以一次从数据库取出所有记录。显著提高性能。

这里需要注意的是，采用resultset游标处理记录时，应该将游标的打开方式设置为FORWARD\_READONLY模式(ResultSet.TYPE\_FORWARD\_ONLY,ResultSet.CONCUR\_READ\_ONLY)，否则会把结果缓存在JVM里，造成JVM Out of memory问题。

iBatis等持久层框架考虑到会有这种需求，所以也有相应的解决方案，在iBatis里我们不能采用queryForList的方法，而应用该采用queryWithRowHandler加回调事件的方式处理，如下所示：

MyRowHandler myrh=new MyRowHandler();

sqlmap.queryWithRowHandler("getAllEmployee", myrh);

class MyRowHandler implements RowHandler {

    public void handleRow(Object o) {

       //todo something

    }

}

iBatis的queryWithRowHandler很好的封装了resultset遍历的事件处理，效果及性能与resultset遍历一样，也不会产生JVM内存溢出。

**减少数据库服务器CPU运算**

1. **使用绑定变量**

绑定变量是指SQL中对变化的值采用变量参数的形式提交，而不是在SQL中直接拼写对应的值。

非绑定变量写法：Select \* from employee where id=1234567

绑定变量写法：

Select \* from employee where id=?

Preparestatement.setInt(1,1234567)

一些不使用绑定变量的场景：

a、数据仓库应用

b、数据分布不均匀的特殊逻辑，如产品表，记录有1亿，有一产品状态字段，上面建有索引，有审核中，审核通过，审核未通过3种状态，其中审核通过9500万，审核中1万，审核不通过499万。

要做这样一个查询：

select count(\*) from product where status=?

采用绑定变量的话，那么只会有一个执行计划，如果走索引访问，那么对于审核中查询很快，对审核通过和审核不通过会很慢；如果不走索引，那么对于审核中与审核通过和审核不通过时间基本一样；

对于这种情况应该不使用绑定变量，而直接采用字符拼接的方式生成SQL，这样可以为每个SQL生成不同的执行计划，如下所示。

select count(\*) from product where status='approved'; //不使用索引

select count(\*) from product where status='tbd'; //不使用索引

select count(\*) from product where status='auditing';//使用索引

1. **合理使用排序**

Oracle的排序算法一直在优化，但是总体时间复杂度约等于nLog(n)。普通OLTP系统排序操作一般都是在内存里进行的，对于数据库来说是一种CPU的消耗。大记录集排序不仅增加了CPU开销，而且可能会由于内存不足发生硬盘排序的现象。

以下列出了可能会发生排序操作的SQL语法：

Order by

Group by

Distinct

Exists子查询

Not Exists子查询

In子查询

Not In子查询

Union（并集），Union All也是一种并集操作，但是不会发生排序，如果你确认两个数据集不需要执行去除重复数据操作，那请使用Union All 代替Union。

Minus（差集）

Intersect（交集）

Create Index

Merge Join，这是一种两个表连接的内部算法，执行时会把两个表先排序好再连接，应用于两个大表连接的操作。如果你的两个表连接的条件都是等值运算，那可以采用Hash Join来提高性能，因为Hash Join使用Hash 运算来代替排序的操作。具体原理及设置参考SQL执行计划优化专题。

1. **减少比较操作**

我们SQL的业务逻辑经常会包含一些比较操作，如a=b，a<b之类的操作，对于这些比较操作数据库都体现得很好，但是如果有以下操作，我们需要保持警惕：

Like模糊查询，如下所示：

a like ‘%abc%’

Like模糊查询对于数据库来说不是很擅长，特别是你需要模糊检查的记录有上万条以上时，性能比较糟糕，这种情况一般可以采用专用Search或者采用全文索引方案来提高性能。

不能使用索引定位的大量In List，如下所示：

a in (:1,:2,:3,…,:n)   ----n>20

如果这里的a字段不能通过索引比较，那数据库会将字段与in里面的每个值都进行比较运算，如果记录数有上万以上，会明显感觉到SQL的CPU开销加大，这个情况有两种解决方式：

a、  将in列表里面的数据放入一张中间小表，采用两个表Hash Join关联的方式处理；

b、  采用str2varList方法将字段串列表转换一个临时表处理，关于str2varList方法可以在网上直接查询，这里不详细介绍。

以上两种解决方案都需要与中间表Hash Join的方式才能提高性能，如果采用了Nested Loop的连接方式性能会更差。

1. **大量复杂运算在客户端处理**

什么是复杂运算，一般我认为是一秒钟CPU只能做10万次以内的运算。如含小数的对数及指数运算、三角函数、3DES及BASE64数据加密算法等等。

如果有大量这类函数运算，尽量放在客户端处理，一般CPU每秒中也只能处理1万-10万次这样的函数运算，放在数据库内不利于高并发处理。