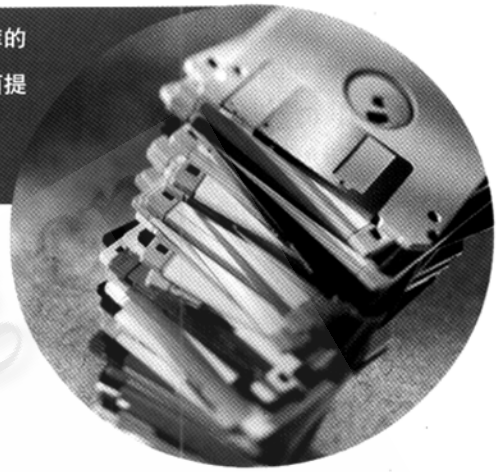
大型数据库使用经验

于 华（山东财政学院计算机信息工程系 250014）

摘要：大型数据库的使用，优化性能是关键。根据笔者的经验，大型数据库的性能提高主要取决于逻辑数据库设计，索引设计和查询设计。本文从这几个方面提供了有益的建议。

关键词：数据库 逻辑设计 索引 SQL



众所周知，大型数据库无论在运行速度、数据安全性、数据处理能力方面都有着桌面数据库所不可比拟的优越性，因此随着计算机应用系统的扩大，大型数据库已成为大中型企业管理应用的首选数据库平台。但是，大型数据库的设计、使用、维护也相对比较复杂，如果设计使用不当，不但不能充分发挥其优越性能，反而会造成其效率的低下，甚至造成整个应用系统的失败。根据笔者的经验，大型数据库的性能提高主要取决于逻辑数据库设计，索引设计和查询设计。笔者在多年设计和使用数据库的基础上，归纳出以下几点经验，供大家参考。

1 合理进行数据库的逻辑设计和物理设计

数据库的设计通常按两步法进行，即首先进行逻辑设计，而后进行物理设计。在数据库逻辑设计过程中，按照关系数据库的规范化要求，为了保证数据库的一致性和完整性，设计人员往往会设计过多的表间关联，以尽可能地降低数据冗余。表间关联是一种强制性措施，建立后，对父表和子表的插入、更新、删除操作均要占用系统的开销。另外，在进行数据查询时也增加了表间连接查询的操作（尤其是大数据表），从而使数据库的性能大为降低。因此物理设计需折中考虑，根据业务规则，确定对关联表的数据量大小、数据项的访问频度，对比较频繁的数据表关联查询应适当提高数据冗余设计。以下方法经实践验证往往能提高性能。

(1) 如果规范化设计产生了许多4路或更多路合并关系，就可以考虑在数据库表中加入重复列

(2) 常用的计算字段（如总计、最大值等）可以考虑存储到数据库表中

比如一个工资管理系统中有工资表，其字段为：职工编号、基础工资、职务工资、补贴...等等，而应发工资（基础工资+职务工资+补贴+...）是用户经常要在查询和报表中用到的，在表的记录量很大时，有必要把应发工资作为一个独立的字段加入到表中。这里可以采用触发器以保持数据的一致性。

(3) 把频繁被访问的数据同较少被访问的数据分开存储。常用的做法是：

① 把一个表分割成2个表（把所有的列分成2组）。这种做法要求在每个表中复制首要关键字。这样产生的设计有利于并行处理，并将产生列数较少的表。

② 把一个表分割成2个表（把所有的行分成2组）。这种方法适用于那些将包含大量数据的表。在应用中常要保留历史记录，但是历史记录很少用到。因此可以把频繁被访问的数据同较少被访问的历史数据分开。而且如果数据行是作为子集被逻辑工作组（部门、销售分区、科目类别、地理区域等）访问的，那么这种方法也很有好处。

2 合理进行索引设计

2.1 适当使用簇式索引

大型数据库有两种索引即簇式索引和非簇式索引。在非簇式索引下，数据在物理上随机存放在数据页上，如果执行范围查找，必须执行一次表扫描才能找到这一范围内的全部行。在簇式索引下，数据在物理上会按照簇索引键的顺序存储，因而在范围查找时。可以先找到这个范围的起点，且只在这个范围内扫描数据页，避免了大范围扫描，可以显著提高查询速度。例如下面的这条SQL语句：

```
Select sale_date,sum(money) From sale Group By sale_date
```

当在sale_date上建有簇式索引时，其查询速度远高于在其上建有非簇式索引的查询速度。缺省情况下建立的索引是非簇式索引，但有时它并不是最佳的；合理的索引

设计要建立在对各种查询的分析和预测上。一般来说，有大量重复值、且经常有范围查询（between,<,>,>=,<=）和order by、group by发生的列，可考虑建立簇式索引。

2.2 选择合适的填充因子（Fill Factor）

索引的建立能提高按索引查询的速度，但同时也会降低插入、更新、删除操作的性能，尤其是当填充因子较大时。这是因为在数据修改、插入或删除时需要同时更新索引，而这会增加额外的开销。所以对索引较多的表进行频繁的插入、更新、删除操作，建表和索引时应设置较小的填充因子，以便在各数据页中留下较多的自由空间，减少页分割及重新组织的工作。

2.3 建立组合索引

经常同时存取多列，且每列都含有重复值时可考虑建立组合索引，组合索引要尽量使关键查询形成索引覆盖（即引用的所有列都包含在组合索引中），其前导列一定是使用最频繁的列。例如下面的查询语句：

```
Select count(*) from sale
  where sale_date>'20000101' and sale_date<'20000201'
and money>3000
Select sale_date,sum(money) from sale group by sale_date
```

当在place,sale_date,money上建立组合索引时，其运行速度都比较慢。因为它的前导列是place，而这两条SQL语句都没有引用place，因此也没有利用上索引；

当在sale_date,place,money上建立组合索引时，其运行速度明显快起来。因为它将sale_date作为前导列，使每个SQL语句都可以利用索引。而且在第一条SQL语句中形成了索引覆盖，使其性能达到最优。

3 使用高效率的数据库SQL语句

应用程序的执行最终归结为数据库中的SQL语句的执行，因此SQL语句的执行效率最终决定了数据库的性能。但是，人们在使用SQL时往往会陷入一个误区，即太关注于所得的结果是否正确，而忽略了不同的实现方法之间可能存在的性能差异，这种差异在大型的或是复杂的数据库环境中（如联机事务处理OLTP或决策支持系统DSS）中表现得尤为明显。因此在使用过程中应尽可能地对它们进行优化，以提高其运行速度。

3.1 尽量使用可优化的Where子句

例：下列SQL条件语句中的列都建有恰当的索引，但执行速度却非常慢：

```
Select * from employee where sunstring(empno,1,2)='01'
Select * from employee where convert(char(10),date,112),
='19931201'
```

原因是where子句中对列的任何操作结果都是在SQL运行时逐列计算得到的，因此它不得不进行表扫描遍历每行，而没有使用该列上面的索引；如果这些结果在查询编译时就能得到，那么就可以被SQL优化器优化，使用索引，避免表扫描。因此可将SQL语句重写如下：

```
select * from employee where empno like '01%'
select * from employee where date='19931201'
```

3.2 合理使用外连接

外部联接按“*”（对MS SQL Sever而言）在“=”的左边或右边分左联接和右联接。若带“*”运算符的表中的一个行不直接匹配于不带“*”运算符的表中的任何行，则前者的行与后者中的一个空行相匹配并被返回。利用外部联接可以替代效率十分低下的not in运算，大大提高运行速度，例如，下面这条命令执行起来很慢。

```
select  a.empno from emp a where a.empno not in
 (select empno from emp1 where job='SALE');
```

倘若利用外部联接，改写命令如下：

```
select a.empno from emp a ,emp1 b
where a.empno*=b.empno
  and b.empno is null
  and b.job='SALE';
```

可以发现，运行速度明显提高。

3.3 巧用SQL Sever的CASE功能

在SQL Server的SELECT语句中可以使用CASE功能（ORACLE中DECODE函数用法与其类似），它可根据表达式的内容返回不同的值，巧用它可以使原来需多条SQL语句完成的统计在一条语句中完成。如：有学生数据表student，要统计各班学生总数、男女生总数及90分以上学生总数，通常的方法是对每个指标分别进行统计，如统计男生总数可用以下的SQL语句：

```
select class,count(*) as boys from student where sex='男
' group by class order by class
```

这样以来需4条语句方可完成所有数据的查询，再通过4个游标将数据取出放入统计表中。改用CASE则只需一条SQL语句：

```
Select class,count(name) as total ,sum(case sex when '男
' then 1 else 0 end) as boys,sum(case sex when '女' then 1 else
0 end) as goils ,sum((case sign(score-90) when 1 then 1 else 0
end)+(case score when 90 then 1 else 0 end)) as excellents
  From student
  Group by class
```

```
Order by class。
```

3.4 善于使用存储过程

存储过程是存储在数据库中的一段程序。它是在建立时就已经编译和优化的程序，并将执行计划存储在数据库中。因此它的执行速度要比独立执行同样的SQL语句快。

4 慎用游标

游标提供了对特定集合中数据逐行扫描的手段，一般使用游标逐行遍历数据，根据取出的数据的不同条件进行不同的操作。如：

```
Declare author_cursor  cursor for select auth_no from
author
 Open author_cursor  打开游标
 Fetch next from author_cursor into @mauth_no  取数
 While (fetch_status=0)
 Begin
  If @mauth_no='  条件 1
   操作 1
  if @mauth_no=' 条件 2
   操作 2
  ...
  fetch next from author_cursor into @mauth_no
 End
 ...
 ...
```

从以上游标的操作过程可以看出，对多表或大表定义的游标(大的数据集合)循环很容易使程序进入一个漫长的等待甚至死机。因此，应尽可能避免使用游标。有些场合，如果非得使用游标，可考虑将符合条件的数据行转入临时表中，再对临时表定义游标进行操作，可使性能得到明显提高。如：

```
Select * from author into #temp where 1=2
Insert into #temp select * from author where 条件
Declare author_cursor  cursor for select auth_no from
#temp /* 对临时表定义游标
... ■
```


参考文献

1 中创工作室编著．《Transact-SQL 语言精解》，2000

2 http