加快Oracle查询速度的方法研究

张新香 （武汉中南财经政法大学信息学院 430064）

摘要：目前各企、事业单位的数据量逐渐增大，数据库环境日益复杂，提高查询速度的研究极为必要。鉴于此，本文介绍了几种加快ORACLE查询速度的有效方法。

关键词：索引 数据簇 分区 实体化视图 并行查询

大型数据库管理系统ORACLE以它优秀的数据管理功能而为大众所喜爱，但是随着数据量的增大，数据库环境复杂化的增强，如何在海量数据中实现快速查询，是困绕程序开发人员和数据库维护人员的一大难题。据统计，在数据库的操作中，有百分之七十以上是基于数据查询的。因此，探讨如何加快查询速度是很有价值的。下面从几个方面来探讨加快Oracle查询速度的方法。

1 索引

存储在关系数据库中的数据如果没有进行处理，没有特别的顺序，用户在数据量浩瀚的表中，查询满足某一条件的记录，速度很慢，效率低下，要想提高查询速度，我们首先想到的方法是建立索引，因为索引的数据都是排了序的。

举一例子：用户ZXX想要查询表Z1中满足条件的C列数据，而表Z1共有A、B、······Z 26列数据，按照通常的方法：

```
select colc
  from z1
  where 条件
```

来实现查询。ORACLE系统必须把Z1中所有的表列读入缓冲区，而缓冲区的空间大小是有限的，这样很明显会降低查询速度，如果以C列为关键字建立索引：create index c_ind on z1(colc); 索引只包含表中用来建立索引的表列信息，可以想象查询满足条件的C列数据将会快很多。

如果用户想要同时查询多列数据，如上例，要查询b,c,d列，并且每列都含有重复值时，可以建立组合索引：create index bcd_ind on z1(colb,colc,cold)。组合索引与单列索引相比，只是索引表稍大，管理与提高查询速度的原理一样。但是要注意建立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）而且其前导列一定是使用最频繁的列。

举例：

```
create index pda_ind on record(place,date,amount);
select count(*) from record where date>'19991201' and date<'19991214' and amount>2000;
```

这是一个很不合理的组合查询，因为它的前导列是place，而以上查询语句没有引用place，也就是没有利用索引。

如果建立下列查询：

```
select count(*) from record where date>'19990901' and place in('bj','sh')
```

这个SQL使用了place，且引用的所有列都包含在组合索引中，形成了索引覆盖，所以查询速度很快。

以上建立的索引都是直接针对表列建立起来的，是最常用的索引，即B*索引，排序完全依靠表列进行，很多情况下，这种索引不能满足要求，比如，要以与表列相关的条件进行查询就无法实现，此时可以建立基于函数的索引。举例：有一员工工资表GZ，表中有字段：应发工资YFGZ、应扣款项YKKX，现要查询实发工资>500的员工人数，无法对表列直接建立索引，于是建立基于函数的索引。函数由用户自定义：

```
create or replce founction get_sf(
    yfgz number,
    ykkx number)
```

```
retu number deterministic is
   begin
   return(yfgz-ykkx);
end;
create index sf_ind on gz(get_sf(yfgz,ykkx));
select count(*)
from gz
where get_sf(yfgz,ykkx)>500;
```

利用基于函数的索引提供对数据的一个快速访问通道，从而加速查询。当然函数不仅可是用户自定义的，也可以是系统自带的，但是要建立以函数为基础的索引必须满足诸多条件：实例必须使用以成本为基础的优化器，以规则为基础的优化器不能识别以函数为基础的索引；索引只能使用可重复的函数，不可包含对SYSDATE和UESR这样的函数调用；另外也不能包含对产生随机数字的DBMS-RAND封装的调用；以函数为基础的索引不可在LOB、REF、嵌套表或变化列的基础上创建；以函数为基础的索引不可使用集合函数，如SUM（）等，只能使用单行SQL函数。这些限制随着ORACLE的升级可能会得以放松。

除了以上方式的索引外，还有一种常见索引，即位图索引。位图索引存储位图，每个位图是一系列的ON（1）和OFF（0）字位，位图索引包含索引中的一列或一组列中每一个不同值的位图，当相关的键值存在时，位图中的该位为1，否则为0。位图索引比传统的索引节约时间和空间，而且使得星型查询优化工作在位图索引上得到很好的完成，但是位图索引只对查询数据的决策支持和数据仓库的应用程序有用，不应该创建位图索引来支持有频繁插入和更新数据操作的应用程序。

上述诸多索引中都是把基表和索引分别存放，而索引组织表提出了一种特殊的利用索引来组织的表结构。它把表的数据只存放在相应的索引里面，数据库系统只维护单独一个B*树索引，索引组织表必须有主键来担当该结构的索引，这个索引包含编码列的值和相应行的别的列的值，而基表没有数据，表内数据的更新只需要修改相应的索引里面存储的数据，对基表没有什么操作，这样的话，一方面可以避免为进行多种查询，而建立多个索引，从而造成索引空间比基表占用的空间大的多的空间浪费。另一方面对于通过主键、准确匹配或基于范围查询的查询来说，查询速度可以大大提高。索引组织表的创建方法如下：

```
create table 表名
   (
   列名1 数据类型1,
   列名2 数据类型2,
   ……
   列名N 数据类型N,
   constraint 约束名 primary key(列名1,列名2……
列名M)
   )
   organization index
   tablespace 表空间名1
   pcthold number
   overflow tablespace 表空间名2;
```

在索引组织表中，由于所有的数据都存放其中，索引可能包含数据太多，这将导致每个索引的叶节点只包含一个行甚至一行的一个部分，造成索引优越性的丢失。解决的方法是利用pcthold参数和overflow tablespace子句，当一行的数据超过由pcthold设定的大小时，将把余下的非键列数据存于由overflow tablespace指定的空间中。

建立索引的方法很多，这里因为篇幅有限，不再叙述。但要注意在建立索引时，一定要根据实际情况，仔细分析查询要求，合理建立索引，否则，将造成索引无效。

2 数据簇

有两种数据簇：索引数据簇和散列数据簇。索引数据簇加快数据查询的功能很小，通常采用散列数据簇。基于索引的查询，最终数据依然来自数据表，而非索引表，可想而知，要想获取数据，至少需要两次读操作，（先在索引中找到关键值，然后在基表读取数据行）假如一个表相对稳定，即更新很少，近似静态表，我们可以考虑采用单表散列索引簇来优化查询，在单表散列索引簇的情况下。簇关键字会采用散列形式，散列关键字和数据行建立起“一对一”的关系，这样对应的行只需要单独一次读操作，即可读取，从而能显著提高精确匹配搜索的速度，但对范围查询效力很弱。举例说明其创建方法：

```
create cluster store_item (item_no number)
size 256 single table haskeys 10000
   pctused 80
   pctfree 5
   tablespace users
   storage (initial 1M next 1M pctincrease 0);
```

3 分区

分区即把大型数据表和索引按照某一条件分成小的区间，对满足条件的数据查找可以只在少数几个分区中进行，从而加速查询。可以建立范围分区、散列分区、混合分区等，可以从范围分区取值，也可以从散列分区、混合分区取值，但因为散列分区和混合分区都是基于散列函数确定的分区，各分区所存放数据没有明显的规律，所以对于加速查询没有意义，因此只介绍范围分区。范围分区根据列值范围，把数据分到不同的分区内，从范围分区的建立机制中可以看出其提高查询速度的原因。

举例：对员工工资表按照实发工资进行分区：

```
create table gz(
    bh  varchar2(6)  not  NULL,
    xm  varchar2(8)  not  NULL,
    … …
    sfgz  number);
storage  (initial  100M  next  100M  pctincrease  0)
    partition  by  range (sfgz)
    (partition group1
        values less then (2000) tablespace gz1,
    partition group2
        values less then (4000) tablespace gz2,
    … …
    partition group10
        values less then (20000)  tablespace gz10);
```

建立分区后进行查询：

```
selete  xm  from  gz  partition(group2) 相当于
selete  xm  from  gz
    where sfgz>2000 and sfgz<4000;
```

因为查询只限于某一区间进行，就比基于基表的查询快很多。并且在不同的分区中，可以根据查询条件选择相应的查询规则，增加了查询的灵活性。

从分区查询可以看出，分区中的数据是无序的，于是我们想到利用分区和索引相结合的方法来加快查询。即建立索引分区。索引分区包括两种类型：本地索引分区和全局索引分区。本地索引分区根据同基表同一分区关键列进行分区，其分区数与基本表的分区数一样，并随着基本表分区的改变而作响应的改变。举例如下：

```
create  index gz_ind on gz(sfgz) local
    store  in  (sfgz1,sfgz2.....sfgz10)
    storage (initial 100M next 100M pctincrease  0);
```

本地索引分区不支持实施非分区式列的唯一性，全局索引分区对非分区式列实施唯一性。全局索引可以选择地进行分区，但是不应该采用基于基表的分区方式，只通过范围分区方法进行分区，且最高分区必须有一个MAXVALUE的分区边界，以保证表中所有行都在索引中得以表现。

```
create unique index gz_ind2  on  gz(bh) global
    partition by range(bh)
    partition bh1  values less then (1000) tablespace bh1,
    partition bh2  values less then (2000) tablespace bh2,
    ……
    partition bh5  values less then (maxvalue) tablespace
    bh5;
```

4 实体化视图

实体化视图用于汇总、预计算和分布数据的模式对象。它将算好的查询结果和累计存储在数据仓库中。当用户发出查询要求，优化器先到实体化视图中去搜索，如果找到满足条件的视图，就让查询直接指向实体化视图而不是基表或视图，提高了数据仓库的查询速度，并且对于复杂的汇总和预计算，不需要直接访问数据库，从而减轻了数据库负担，优化了查询。

因为实体化视图是用来存放汇总、预计算的结果，所以创建视图时可能用到聚合、连接或两者均用到，因此把实体化视图分成三种类型：实体化连接视图（以执行内部或外部等同连接的查询为基础，但不含聚合）、单独的表聚合视图（包含聚合，但不包含连接）、即包含聚合又包含连接的视图。举例说明他们创建的方法：

```
create  materialized view sales_details  /*创建实体化连接视图 */
    build immediate
    refresh fast on commit
as
select sales.rowid,product.rowid,sales.units,sales.dollars,product.department
    from sales,product
    where sales.prod_id=product.prod_id;
create materialized view sales_summary  /*创建单独的表聚合视图 */
    build immediate
    refresh fast on demand
as
select  sales.store_id,sales.prod_id,count(*),sum(sales.
```

```
cost),sum(sales.units)
      from sail
      group by sales.store_id,sales.prod_id;
   create materialized view prod_season /* 创建包含连接和聚合的视图 */
      refresh fast on demand
   as
   select sales.season,product.department,conut(*),sum(units),avg(units)
      from sales,product,time
   where sales.prod_id=product.prod_id and sales.time_id=product.time_id
   group by time.season,product.department
```

5 并行查询

随着计算机硬件的迅速发展，已经出现了多CPU的计算机，于是我们利用并行查询来提高查询速度，对于同一个查询任务，启用多个并行查询进程同时执行查询工作，这样会使得查询速度成倍的增加，当然，要使查询以最快的速度完成，各个进程之间还需要用动态平衡装载系统来调节。

6 结束语

提高查询方法还有很多，比如采用并行数据库来优化查询，将经常同时查询和频繁查询的对象放在各自的磁盘上来加速查询等。但要注意，对于任何一种查询，都要根据具体情况选用合适的优化措施，否则将达不到预期的效果，甚至起到反作用。

以上只是探讨了加快查询的一些常用有效措施，要深入研究还会涉及到数据库层的资源配置、网络层的流量控制以及操作系统层的总体设计。感兴趣的读者可以进一步研究。■

参考文献

1 [美] William G·age Jr Nathan Hughes: 《即学即用Oracle8》，电子工业出版社， 1999年版。

2 [美] Steve Bobbrowsk: 《Oracle8i for Window NT 实用指南》，机械工业出版社， 2000年版。

3 [美] Joline Morrison, Mike Morrison《Oracle数据库指南》，机械工业出版社， 1999年版。

4 [美] Douglas·Scherer William·Gaynor《Oracle8i 数据库开发技术与技巧》，机械工业出版社，2000版。

5 [美] Scott·Urman《Oracle8 PL/SQL程序设计》，机械工业出版社。2000年版。