## 数据库索引

数据库索引是一个可选择性创建的数据库对象，主要用于增加查询性能的。数据库索引的功能就像是书的背面中的索引一样的。一个书的索引和一个主题链接起来并且链接一个页码。当你想要查找书中的信息的时候，一般看下目录索引会快很多。找到自己喜欢的主题，然后去具体的页码上面去。用这些信息，你可以直接定位到书中的具体位置。否则，如果你想要看的主题就在书中最开始的几页，那么你可以很快的找到内容。这种情况，索引的功效是减弱了的。但是当你查询的主题离书中很远的时候，这个时候索引的功效就体现出来了。

和书的索引类似，数据库的索引也储存感兴趣的主题的行值，并且给他们具体的ROWID。这个ROWID包含着储存着主题的值在磁盘中表行的物理路径。通过ROWID，数据库可以很有效的获取表数据，通过一个很小的去瓷盘的读取。通过这种方式，索引就像是一个对表数据的快捷键。如果没有可用的索引，那么数据库必须一行一行的读取表，已确定行上面是否含有你想要的信息。

提醒：为了提高性能表现，数据库强制使用索引在主键约束以及唯一性约束中。另外，oracle能够更好的处理表锁的问题当主键呗外键列替换的时候。

当一个基于数据库的引用程序完全避免使用索引，这个可以确定性能的局限性。索引可以创造出十分强大的扩展性甚至是对于大数据集合。所以，如果索引如此对数据库查询性能如此重要，为什么不在所有的表的每个列上面都建立索引呢？答案很简单，因为索引不是免费的。它们对磁盘空间以及系统资源的消耗是巨大的。因为当每个列的值改变以后，每个相应的索引都必须更新。通过这种方式。索引对存储，IO以及CPU以及内存资源的消耗。如果没有很好的使用索引的话，会导致瓷盘空间的浪费以及系统资源的浪费。反而会导致数据库性能的下降。

因为这些原因，当你设计和构建oracle数据库应用的时候，对于索引创建的专业意见必须给定。作为应用架构，你必须索引的无力特性。什么类型的索引是可以使用的，并且选择哪些表以及列结合起来创建索引。一个正确的索引创建方式是增强你的数据库性能的至关重要的条件。

这个章节介绍你了解oracle索引的概念。我们从一个小例子开始。然后我们开始解释oracle中可以使用的索引类型并且提供指导，并对哪些列创建索引给予指导意见。如果你对索引是初学者的话，从这个地方学起是一个很好的开端。

### 使用索引提神性能

一个索引到底是如何提神查询效率的呢？为了理解索引的工作原理，看下面的这个小例子。例如你创建一个表来存储消费者信息，

create table cust(cust\_id number,last\_name varchar2(30),first\_name varchar2(30));

随着你的业务的增加，很短的时间内，上百万的消费者呗创建出来了。你运行对这个表格的每日报告，发现下面语句的执行效率越来越低：

select cust\_id,last\_name,first\_name from cust where last\_name = ‘STARK’;

当表中没有什么数据的时候，这句执行的十分块。现在，几百万条数据以及增长。这个查询的时间越来越长。这到底是什么回事儿？

当一个SQL select语句执行的时候，oracle查询优化器快速计算出每个步骤的执行计划描述它会如何获取在查询中指定的列的值。通过计算计划，优化器决定哪个表格以及索引会被用于获取数据。

当没有任何索引存在的时候，表本身是唯一的查询路径。在这种情况下，oracle没有任何选择余地只能检测每一行的数据。（这个被称之为全表扫描），用于查看是否有一个名叫STARK的人。随着更多的数据被插入到这个表格，这个查询的查询时间会持续更久。这条查询的消耗（作为测量CPU，内存，以及I/O资源消耗）随着表的体积的增大而变化。要想让这条语句运行更快速的唯一方式就是换一个更好的硬件。或者使用索引来增强性能。

你可能已经在本章节中了解到了，使用索引在对应的列上面可以增强性能。例如这个地方在last\_name中使用创建一个索引。Create index cust\_idx1 on cust(last\_name);

这条语句创建了一个B-tree类型的索引。这个是Oracle的默认索引类型。创建索引后，进行搜索的时候的速度斗升，生活又变的很好。

为了理解索引所如何提神性能的。还记得吗？一个索引存储两种信息，表行的值，以及它对应的ROWID，ROWID唯一指向表格的唯一行（对于堆类型表）在一个数据库中，并且包含他的物理路径（文件，阻塞，以及行位置对于阻塞）。（一旦索引被创建。子序列查询执行，查询优化器会考虑索引是否会减少对资源的消耗，然后返回具体的信息）

警告：ROWID唯一性标示一条行对于当前堆内存组织中的表格是这样的，但是对于表集群的时候。有可能有行在不同的表格中，却位于同一个块中，并且油相同的ROWID

在上面的例子中，假设有几百万条数据在CUST表中，但是只有一条记录在表格中，使用last\_name 为STARK。这个时候的查询优化器能够在索引的作用下通过ROWID在几次读写下面查询到具体的位置。这种情况下，不论表格有多少纪录查询速度依然是一样的快。

相反，考虑考虑如果LAST\_NAME的值不是唯一的。如果有成百上千条记录在CUST表中都有值LEE。如果查询优化器使用索引，他会从成百上千条数据中读取几百万次获取ROWIDs，然后读取表格成百上千次。这个时候不使用索引，直接进行全表查询反而更快，因为这个原因。有时候优化器计算如果使用索引不能提神效率的话，就会忽略它。

提醒：唯一性纬度越强的列值，B-tree索引的效率越高。在数据库行话中，一个唯一性的列值相比于全部行来说是非常高效的。比如，对几个唯一数据的查询，相比较于整张表格的查询。

有一个很有趣的场景我这边需要指出来。假设我们上面查询的不是所有的CUST表格的所有的值，而是LAST\_NAME这一个列的值。在这种场景下面，由于索引包含着我们select需要的值。因此oracle深知不用去查询表，就可以返回结果。这种索引叫做是覆盖索引。这些索引效率特别高。因为只有索引需要进行读取。

在我们进行下面的介绍前，先介绍下下面的一些概念在前面介绍的，

* 索引是一种选择性的定义在表格中的可用于一列或者多列的对象
* 索引是需要消耗资源的
* 一个B-tree 索引是oracle的默认索引类型
* 一个相对唯一的列相比较于全部的列来说，B-tree索引效率更高
* 合理的创建索引，可以提高效率
* 在有些场景下面，查询优化器会选择不使用索引，换句话说，优化器会计算全表扫描的消耗然后去和使用index的进行对比，如果小的话，就会进行全表扫描
* 一些场景下，oracle甚至可以不用对表进行读取。知识读取索引就可以获取数据。

对于上述基本的索引该了的理解对于本章节后面的介绍是很重要的。现在我们开始将我们的注意力放到索引的类型。

### 决定哪种类型的索引我们使用

oracle提供了大量的索引类型以及各种特性。正确使用这些索引可以有很好的效果以及扩展。相反，如果你没有正确的使用这些特性，可能会降低性能。表格1-1列出了oracle的索引的各种种类。初步看到的时候，这是一个很长的列表，可能对一些初学者来说有点过于复杂。决定使用那种索引类型来进行使用。不像是我们感觉到的那么复杂。对于大多数应用来说。我们只需要单纯的使用B-tree就可以了。

提醒：下面的一些索引有的只是针对基本的B-tree索引进行区分的。一个reverse-key 索引，例如，就是基本上对B-tree在扩展IO当使用序列生成和插入相似值的时候。

|  |  |
| --- | --- |
| 索引类型 | 用法 |
| B-tree | 默认索引类型，均衡树索引，对于高基数列有效，除非你有绝对的原因否则使用这种类型的索引 |
| Index organized table | 当大多数的列都是作为主键的时候，你取出索引，就想取出一个表一样。存储数据的类型结构就像是B-tree一样 |
| Unique | B-tree索引的一种形势，用于强化列值的唯一性。经常使用在主键以及唯一性约束的列上面。但是也能够被单独创建出来。 |
| Reverse-key | B-tree的一种形势。用于在频繁序列化插入平衡I/O |
| Key-compressed | 对于联合索引，通常主要的列都会重复，压缩的叶子节点阻塞入口，这种特性应用于B-tree 或者是IOT索引 |
| Descending | B-tree的一种形势，用于索引对应的列按照下降的顺序排序，默认是生序的。你也可以镇定降序对一个相反的键值。如果是一个bitmap的话忽略降序 |
| Bitmap | 在列数比较少的时候在数据仓库中的表现十分抢眼以及sql语句使用多次And 或者Or操作符在where从句中的时候。BitMap不适用于在线事物处理，行频繁的更新。你不能单独创建一个唯一的BitMap |
| Bitmap join | 在数据仓库中常常使用，查询利用星级实例结构连接实际以及多维度的表格 |
| Function-based | 对于列有sql函数对他们有应用，这个可以使用B-tree或者bitmap索引 |
| Indexed virtual column | 一个索引定义在一个虚拟的列（表）用于列有sql函数应用在他们身上。对于使用基于函数的索引可以选择使用。 |
| Virtual | 允许你创建索引不必须使用物理分区或者继承NOSEGMENT从句的CREATE INDEX 。对于调试SQL在没有资源消耗的情况下。任何的索引类型都能被创建成虚拟的。 |
| Invisible | 索引对于查询优化器不可见。然而，索引的数据结构被维护当表的数据改变的时候。有效测试一个索引在使他具体的可见以前。 |
| Local partitioned | 本地索引机遇不同分区在不同的表中。这个可以使用B-tree或者是bitmap索引 |
| Domain | 用于一个程序或者是子弹 |
| B-tree cluster | 对于集群式的表 |
| Hash cluster | 用于Hash集群 |

B-tree索引和其他类型的索引在下面会简单的进行介绍。在以后的章节中会具体介绍。

### B-tree Indexes

我需要指出B-tree类型索引是第二章的绝对主角。我们会在哪一章介绍他们这样你就不会和其他类型的索引弄混了。正如之前说过的，默认的索引类型在oracle中是B-tree索引。这种索引类型非常有效对于高基数的列数量。对于大部分的程序来说这种索引就足够了。并且他也是默认的索引类型。

下面的几种索引类型是B-tree的字类型

* Index-organized table（IOT）
  + 创建格式：create table……organization index 语句，例如，
  + create table test( id number, name varchar2(128), constraint pk\_test(id,name)) organization index;
  + 使用指南：所有的列都为主键的时候。第四章会对实现细节进行详细的描述
* Unique
  + 当创建一个B-tree索引你能够定义一个唯一的索引，这就像是唯一性约束一样的。大插入数据到表中的时候，唯一性索引会保证数据的唯一性。因此，唯一性索引通常情况下被用于和主键和唯一性约束在一起具体见第二章
  + 创建方式：create unique index cust\_uidx1 on cust(last\_name,first\_name);
  + 提醒：看第二章又完整的讨论对于创建唯一性索引和数据库在primary key 和unique key 约束自动创建的索引的对比
* Reverse key
  + 当有大量的序列化插入的时候，这些索引能够表现更好，在如下的场景中，当你需要一种方式来均衡的发布索引数据，否则有类似的数据集合在一起。使用这个索引，可以避免IO几种在某个物理路径。具体的讨论在第五章。
  + create index cust\_ridx1 on cust(name) revers
  + 你不能指定reverse对bitmap索引以及IOT索引，这种类型也不能指定降序。
* Key compressed
  + 用于降低储存以及IO要求对于联合索引，因为主导的列值经常是重复的，
  + create index cust\_cidx\_1 on cust(last\_name,first\_name) compress 2
* Descending
  + 默认情况下，oracle存储B-tree索引使用的是升序，例如，如果你又一个索引在一个number类型的列上，最小的值回出现在索引的第一个（最左子叶节点）最大的数据会存在在最右的子叶节点
  + create index cust\_idx1 on cust(cust\_id desc);
  + Descending index对于某些需要对某些列排序的查询十分的有效。注意使用到降序索引的时候，查找出来的值就已经是经过排序了的。

### BitMap Index

Bitmap 索引常用于数据仓库环境。这些索引推荐使用在相对值不是变化很大的列。Bitmap索引在sql中大量使用And 和Or的连接操作的时有用，不要使用经常要进行DML操作的列进行bitmap索引。因为bitmap的实现内部有大量的锁操作对于DML所以对于高事物操作的系统中最好不要使用。

创建bitmap索引使用关键字bitmap。如下：

create table f\_sales(

sales\_amt number

,d\_date\_id number

,d\_product\_id number

,d\_customer\_id number);

create bitmap index fs\_bmp\_idx on f\_sales(d\_date\_id);

Bitmap join 这种索引将连接两张表的结果储存起来。方便以后使用。当我们进行外连接查询的时候，使用这种索引还是挺方便的。其他特性和bitmap是一样的。

create table d\_customers

(d\_customer\_id number primary key

,cust\_name varchar2(30));

create bitmap index f\_sales\_bmj\_idx1

on f\_sales(d\_customers.cust\_name)

from f\_sales, d\_customers

where f\_sales.d\_customer\_id = d\_customers.d\_customer\_id;

Function-Based Indexes

简单点就是以前是在列上创建索引，现在是在有函数的列上创建索引。create index cust\_fidx1 on cust(upper(last\_name));

这些类型是必要的因为oracle不会让一个普通的B-tree去索引一个带有函数的索引的。其实也好理解，这个索引对应的位置是这个列的位置，但是值是对应的函数处理后的值。

提醒：function-based indexes 可能是B-tree , unique ,or bitmap

还有一些其他的这边就不在赘述，后面如果章节中有，会有详细的介绍。

Determining which Columns to Index

现在我们将注意力放到哪些列应该建立索引。对于初学者，我们推荐创建索引的一些原则。

* 为每一个表定义一个主键，这样做，oracle会为主键列自动创建一个索引
* 为值唯一的列创建唯一性约束，注意唯一性约束和主键是不一样的。每一个唯一性约束的创建oracle也会自动创建索引。
* 手动给外键创建索引。这样做的原因是为了避免一些锁的问题。（具体在第二章中会有详细的介绍）

上面的每个子项后面的章节中都有详细的介绍

Indexes on Foreign Key Columns

Oracle 不会自动创建外键索引，我们建议手动创建一个B-tree类型的索引，其中一个原因是由于外键列通常会出现在where从句中因此创建索引，可以提神性能。

第二个原因是由于外键本身存在所得问题，当我们插入或者是删除一条数据从子表的时候，这个时候父表也是被锁住的。这样其他线程对浮标的操作就会收到阻塞。在OLTP数据库中，这就会是一个问题，当有大并发数据操作这个表的时候。在数据仓库中，这不是什么很大的问题因为数据会更自动话的更新，也通常不会被删除掉。

例子：

create table address

(address\_id number primary key

,cust\_id number references cust(cust\_id)

,address varchar2(1000)

);

create index address\_fk1 on address(cust\_id);

其他的一些实用场景

当你选择索引的时候，记住下面规则：基于你要查询的表的列，你可以创建多个索引对应于同一种表并且一个索引可以包含多列。你应该首先想的问题是对这个表格哪些字段是你需要经常用到的。满足喜爱下面条件的可以考虑创建：

* where从句中经常用到的，如果经常用到多个，则可以创建多列联合索引。
* 创建覆盖索引在select从句中的列。
* 考虑创建索引在order By , Group By , Union , DISTINCT 从句的列。

Oracle允许对多列同时创建索引，多列索引称为连接索引有时候也叫做组合索引，这种索引特别的高效当在where查询中查询多列的时候。多列索引相比较于单列索引效率高很多。

在SELECT从句中以及where从句中的语句可以的话可以创建覆盖索引。这个时候，可以不用访问表获取数据。

也可以考虑在Order by , group by , distinct, union 的地方创建索引。

虽然在一个表格中可以创建多个索引，但是并不是越多越好。创建太多的索引，会导致DML的操作异常的慢。因为每一个对这个表的操作，都会去维护这些索引，因此会性能的消耗也会异常的大。（第七章会对细节进行详细的描述）

Index Guidelines



