1 索引概览

1.1 索引介绍

索引是一个schema对象，它在逻辑和物理上独立于所关联的对象的数据，因此，索引能够在不影响表的情况下进行删除或者创建。

对于单行数据，索引是通向它的快速访问路径，它只影响执行速度，给定一个已被索引的数据，索引直接指向包含该值的数据行的位置。

索引创建后，数据库自动的维护和使用它，数据库也会自动的反映数据的变化，比如在所有相关的索引中添加、更新和删除行，而不需要用户进行额外的操作。即使插入了新行，索引数据的检索性能几乎保持不变，但是，由于数据库对索引的更新，表上有过多索引会降低DML的性能。

1.2 索引特性

索引有下面的属性：

- 可用性（Usability）

索引分为usable或者unusable，默认为usable。unusable索引不会被DML操作维护，并且被优化器忽略。unusable索引能够改善批量加载的性能。可以使索引unusable，然后rebuild它，而不是删除一个索引，然后重新创建它。unusable索引和索引分区不会占用空间，当改变一个usable索引为unusable时，数据库将会删除它的索引段。

- 可见性（Visibility）

索引也可分为visible或者invisible，默认是visible。invisible索引会被DML操作维护，默认不会被优化器使用。将一个索引置为invisible是使索引unusable或者删除的一种替代方法。在不影响整个应用程序的情况下，删除索引之前测试索引的删除或者临时的使用索引，采用invisible索引是尤其有用的方法。

主键和唯一键可以自动的创建索引，而外键需要手动创建索引。

1.3 索引存储

Oracle数据库在索引段（index segment）中存储索引数据。索引段的表空间可以是用户的默认表空间，也可以在创建索引（create index）时指定一个表空间，为便于管理，把索引和表存放到不同的表空间。

2 Oracle实现数据访问的方法

Oracle访问数据的方法主要有以下三种，分别为：

- 全表扫描；
- 通过Rowid；
- 使用索引。

当Oracle决定使用索引时会使用Rowid来访问数据，如果没有索引或不选择使用索引时，将会使用全表扫描的访问访问数据。

2.1 全表扫描

在使用全表扫描时，Oracle会读取表中所有的行，此时通过多块读操作可以大大减少I/O次数，利用多块读可大大提高全表扫描的速度，当然，只有在全表扫描的情况下才能使用多块读，如果表较大，不建议使用全表扫描。

```
SQL> explain plan for select * from hr.employees;

Explained.

SQL> select *from table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 1445457117

--------------------------------------------------------------------------------
| Id  | Operation         | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |           |   107 |  7383 |     3   (0)| 00:00:01 |
```

```
|   1 |  TABLE ACCESS FULL| EMPLOYEES |   107 |  7383 |     3 (0)| 00:00:01 |
--------------------------------------------------------------------------------
```

8 rows selected.

8 rows selected.

2.2 通过rowid

对于表对象，在向表中插入数据时隐含创建该行的Rowid，Rowid指出了数据文件、块号、行号，通过Rowid是Oracle数据库中读取单行数据最快速的方法，该读取方法是单块读。

```
SQL> explain plan for select *from hr.employees where
rowid='AAASLGAADAAAQdcABD';

Explained.

SQL> select *from table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------
-------------------------
Plan hash value: 549062733

-------------------------------------------------------------------------------------------
| Id  | Operation                   | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |           |     1 |    69 |     1   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY USER ROWID| EMPLOYEES |     1 |    69 |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
```

8 rows selected.

2.3 使用索引

通过索引找到数据行的Rowid，然后通过Rowid直接到表中查找数据，这种方式成为索引查找或索引扫描，由于一个Rowid对应一个数据行，因此该方式也是单块读，在索引中，除了存储每个索引值，还存储相应的Rowid。

SQL> explain plan for select *from HR.EMPLOYEES t WHERE t.job_id='AD_VP';

Explained.

SQL> select *from table(dbms_xplan.display);

PLAN_TABLE_OUTPUT

Plan hash value: 2096651594

| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time |

| 0 | SELECT STATEMENT | | 2 | 138 | 2 (0)| 00:00:01 |

| 1 | TABLE ACCESS BY INDEX ROWID BATCHED| EMPLOYEES | 2 | 138 | 2 (0)| 00:00:01 |

|* 2 | INDEX RANGE SCAN | EMP_JOB_IX | 2 | | 1 (0)| 00:00:01 |

Predicate Information (identified by operation id):

PLAN_TABLE_OUTPUT

```
   2 - access("T"."JOB_ID"='AD_VP')

14 rows selected.
```

3 索引类型

Oracle数据库提供了如下索引：

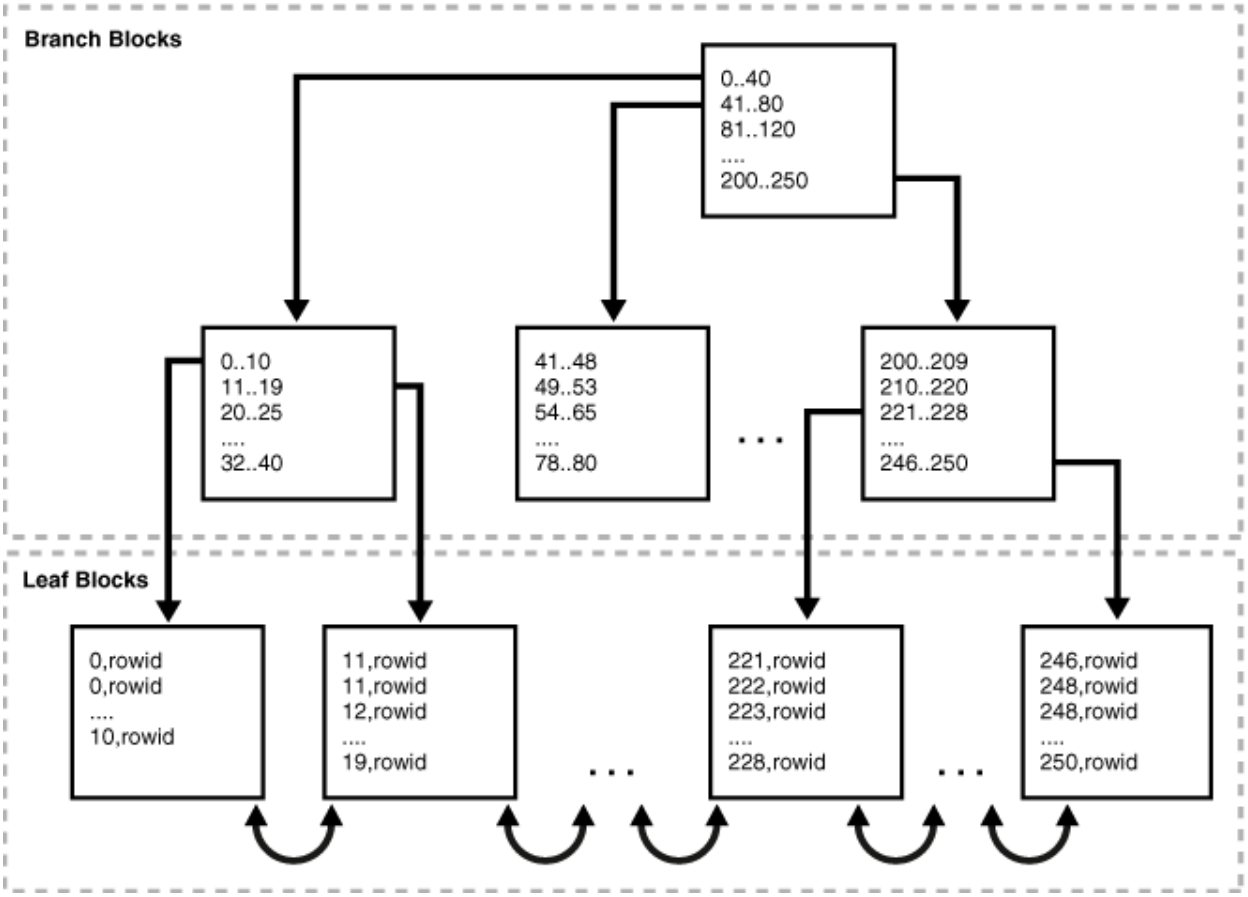
- B-tree索引（B-tree indexes）；
- 位图和位图连接索引（Bitmap and bitmap join indexes）；
- 基于函数的索引（Function-based indexes）；
- 应用域索引（Application domain indexes）；

3.1 B-tree索引（B-tree indexes）

3.1.1 B-tree索引原理

B-tree，简称为平衡树，是最常见的数据库索引类型。B-tree是被分成范围的值的一个有序列表，通过将一行或行范围和键值相关联，B-tree提供了优秀的检索性能，包括精确匹配和范围搜索。

下面展示B-tree索引的内部结构，如图：



分支块和叶子块（Branch Blocks and Leaf Blocks）

B-tree索引有两种块类型，分别是用于搜索的分支块和存储值的叶子块。B-tree索引的上层分支块包含指向低层索引块的索引数据。在上图中，根分支块有一个条目0-40，它指向下一分支层的最左侧块，该分支块包含如0-10和11-19的条目，每一个条目指向包含落到范围的键值的叶子块。

由于所有的叶子块自动保持在相同的深度，B-tree索引是平衡的。因此，从索引中的任何地方检索任何记录花费的时间几乎相同。索引的高度是从根块到叶子块所需要的块数，分支层的高度是索引的高度减1，在上图中，索引的高度是3，分支层的高度是2。
分支块存放在两个键之间做分支决策所需的最小键的前缀，这种技术使数据库能够在每一个分支块上尽可能多的匹配数据，分支块包含一个指针指向包含键的子块，键和指针的数量受限于块的大小。
叶子块包含每个索引的数据值和用来定位实际行的相对应的rowid，每个条目按照（key，rowid）排序，在叶子块中，键和rowid链接到它的左和右兄弟条目，叶子块自身也是双向链接的，比如上图中的最左侧叶子块（0-11）链接到第二个叶子块（11-19）。
带字符数据的列中的索引基于数据库字符集的字符的二进制值。

3.1.2 B-tree索引扫描方式

1）索引扫描（Index Scans）

在索引扫描中，数据库通过使用SQL语句指定的索引列值遍历索引来检索一行数据。如果数据库扫描索引来查找一个值，那么它将在n次I/O中找到该值，其中n是B-tree的高度，这就是Oracle数据库索引的基本原理。

如果SQL语句只访问被索引的列，那么数据库直接从索引中读这些值，而不是从表读取。如果语句访问除索引的列之外的列，那么数据库就使用rowid来查找表中的行。通常，数据库通过交替读取索引块和数据块来检索表数据。

2）索引全扫描（Index Full Scan）

在全索引扫描中，数据库按顺序读取整个索引。如果SQL语句中的谓词（where语句）引用索引中的列，在某些情况下不指定谓词，那么则可以使用全索引扫描，全索引扫描消除了排序，因为数据已经按索引建排序了。

3）快速全索引扫描（Fast Full Index Scan）

快速全索引扫描是一个全索引扫描，数据库访问索引本身的数据，而不用访问表，数据库读取索引块并没有特别的顺序。

扫描索引中的所有的数据块，与index full scan很类似，但是一个显著的区别就是它不对查询出的数据进行排序，即数据不是以排序顺序被返回。在这种存取方法中，可以使用多块读功能，也可以使用并行读入，以便获得最大吞吐量与缩短执行时间。

当满足下面的条件时，快速全索引扫描可以替代全表扫描：

- 索引必须包含查询所需的所有列；
- 查询结果集中不能出现NULL值；

4）索引范围扫描（Index Range Scan）

范围扫描是一个有序扫描，满足下面的特性：

- 索引的一个或多个前导列在查询条件中指定；

- 0、1或更多的值可以作为索引建；

5）索引唯一扫描（Index Unique Scan）

和索引范围扫描相比，索引唯一扫描必须具有与索引键相关的0个或1个rowid。当谓词引用一个唯一索引键时，数据库执行唯一扫描，一旦查到到第一条记录，索引唯一扫描就停止，因为不会再有第二条记录满足条件。

6）索引跳跃扫描（Index Skip Scan）

索引跳跃扫描使用复合索引的逻辑子索引。如果复合索引的前导列有更少的不同值，并且该索引的非前导列有许多不同的值，那么使用索引跳跃扫描将会有意想不到的好处。

7）反向键索引（Reverse Key Indexes）

反向键索引是一种B-tree索引，它在保持列顺序的同时，物理上反转每个索引键的字节。例如，如果索引键是20，如果这两个字节用标准索引以十六进制存储为C1,15，那么反向键索引存储为15,C1。

反向键解决了B-tree索引右侧叶子块的争用的问题，该问题在RAC数据库多个实例多次修改相同的数据块时尤为突出。

8）升序和降序索引（Ascending and Descending Indexes）

在升序索引中，Oracle数据库按升序存储数据。默认情况下，字符数据由值的每个字节所包含的二进制值排序，数字数据按照从小到大排序，日期按照从早到晚排序。

降序索引通过在create index语句中指定desc创建，此时，索引按照一个指定的列或多个列以降序的方式存储数据。当查询按某些列升序，并且其他列按照降序排序时，这种情况可以使用降序索引。

3.2 位图索引（Bitmap Indexes）

在位图索引中，数据库为每个索引键存储一个位图。在传统的B-tree索引中，一个索引条目（index entry）指向一行，而在位图索引中，每个索引键存储指向多行的指针。

位图索引主要为数据仓库或引用许多列的即席查询环境设计，使用位图索引的情况包括：

- 被索引的列具有低基数（cardinality），也就说，与表的行数相比，不同值的数量很小；
- 被索引的表要么是只读的，要么不会被DML做重大修改；

位图中的每个位对应一个可能的rowid，如果设置了位，则对应rowid的行包含键值。映射函数将位位转换为一个实际的rowid，因此，位图索引提供了与B-tree索引同样的功能，虽然它使用了不同的内部呈现机制。

如果更新单个行的索引列，那么数据库会锁定索引键条目，而不是映射到更新行的各个位，因为一个键指向许多行，在被索引的数据进行DML操作通常会锁定所有这些行，处于这个原因，位图索引不适用于OLTP应用程序。

3.3 位图连接索引（Bitmap Join Indexes)

位图连接索引是两个或多个表连接的位图索引，对于表列的每个值，索引在索引表中存储相应行的rowid。相反，在单个表上创建标准的位图索引。
在此不对位图连接索引做过多的介绍，更多关于它的内容可参阅官网。

3.4 基于函数的索引（Function-Based Indexes)

可以基于函数或者表达式在一个或者多列创建基于函数的索引，基于函数的索引可以是B-tree索引，也可以是位图索引。

4 管理索引

4.1 索引创建

1）创建唯一索引

```
SQL> conn hr/hr@orcl
Connected.
```

SQL> create table t_emp as select * from employees;

Table created.

SQL> create unique index uk_t_emp on t_emp(employee_id);

Index created.

2）创建组合索引

SQL> create index idx_emp_last_first_name on t_emp(last_name,first_name);

Index created.

3）创建函数索引

SQL> create index idx_emp_email on t_emp (lower(email));

Index created.

4）创建反向键索引

SQL> create index idx_emp_last_name on t_emp(last_name) reverse;

Index created.

5）创建降序索引

SQL> create index idx_emp_salary on t_emp(salary desc);

Index created.

6）练习

- 创建两个表空间，分别为ts_data和ts_index；

- 创建一张表t_test，表结构和dba_objects相同，并存放其数据，数据存放于ts_data中；
- 创建一个索引idx_test，索引列为owner、object_name和object_type，索引存放于ts_index中；

4.2 查看索引

```
SQL> select t.table_owner,t.table_name,t.index_name,t.index_type FROM
user_indexes t where t.table_name='T_EMP';

TABLE_OWNER                    TABLE_NAME                INDEX_NAME
INDEX_TYPE
------------------------------ ------------------------- ---------------------------
---------------------------
HR                             T_EMP                     IDX_EMP_SALARY
FUNCTION-BASED NORMAL
HR                             T_EMP                     UK_T_EMP                  NORMAL
HR                             T_EMP                     IDX_EMP_LAST_FIRST_NAME
NORMAL
HR                             T_EMP                     IDX_EMP_EMAIL
FUNCTION-BASED NORMAL
HR                             T_EMP                     IDX_EMP_LAST_NAME
NORMAL/REV

SQL> select t.table_name,t.index_name,t.column_name,t.column_position FROM
user_ind_columns t where t.table_name='T_EMP';

TABLE_NAME                INDEX_NAME                COLUMN_NAME
COLUMN_POSITION
------------------------- ------------------------- ---------------------------
--------------
T_EMP                     UK_T_EMP                  EMPLOYEE_ID                    1
T_EMP                     IDX_EMP_LAST_FIRST_NAME   LAST_NAME
1
```

```
T_EMP                    IDX_EMP_LAST_FIRST_NAME      FIRST_NAME
2
T_EMP                    IDX_EMP_EMAIL                SYS_NC00012$
1
T_EMP                    IDX_EMP_LAST_NAME            LAST_NAME
1
T_EMP                    IDX_EMP_SALARY               SYS_NC00013$
1

6 rows selected.

SQL> select *  FROM user_ind_expressions t where t.table_name='T_EMP';

INDEX_NAME                TABLE_NAME                COLUMN_EXPRESSION
COLUMN_POSITION
------------------------------ ------------------------------ --------------------------------
--------------------------------
IDX_EMP_EMAIL                  T_EMP                     LOWER("EMAIL")
1
IDX_EMP_SALARY                 T_EMP                     "SALARY"
```

4.3 监控索引

```
SQL> alter index UK_T_EMP monitoring usage;

Index altered.

SQL> SELECT * FROM t_emp WHERE employee_id=100;

EMPLOYEE_ID FIRST_NAME            LAST_NAME            EMAIL
PHONE_NUMBER      HIRE_DATE JOB_ID          SALARY COMMISSION_PCT
MANAGER_ID DEPARTMENT_ID
----------- -------------------- ------------------------- ------------------------- --------
------------ --------- ---------- ---------- -------------- ---------- -------------
```

```
       100 Steven               King                      SKING               515.123.4567
17-JUN-03 AD_PRES          24000                          90
```

```
SQL> alter index UK_T_EMP nomonitoring usage;

Index altered.

SQL> select * FROM v$object_usage t;

INDEX_NAME                     TABLE_NAME                     MON USE START_MONITORING
END_MONITORING
------------------------------ ------------------------------ --- --- ------------------ ----
----------------
UK_T_EMP                       T_EMP                          NO  YES 08/02/2018 16:53:45
08/02/2018 16:54:31
```

4.4 重建索引

1）重建索引

```
SQL> alter index IDX_EMP_LAST_FIRST_NAME rebuild;

Index altered.
```

> 可以使用索引，但不能进行DML和DDL操作。

2）重建索引并迁移表空间

```
SQL> alter index IDX_EMP_LAST_FIRST_NAME rebuild tablespace test;

Index altered.
```

3）联机重建索引

```
SQL> alter index IDX_EMP_LAST_FIRST_NAME rebuild online;
```

Index altered.

> 可以执行DML操作，但不可进行DDL操作。

4.5 维护索引

1）合并索引碎片

SQL> alter index IDX_EMP_LAST_FIRST_NAME coalesce;

Index altered.

2）删除索引

SQL> drop index IDX_EMP_LAST_FIRST_NAME;

Index dropped.