ORACLE SQL 语句优化技术分析

一、问题的提出

在应用系统开发初期，由于开发数据库数据比较少，对于查询 SQL 语句，复杂视图的的编写等体会不出 SQL 语句各种写法的性能优劣，但是如果将应用系统提交实际应用后，随着数据库中数据的增加，系统的响应速度就成为目前系统需要解决的最主要的问题之一。系统优化中一个很重要的方面就是 SQL 语句的优化。对于海量数据，劣质 SQL 语句和优质 SQL 语句之间的速度差别可以达到上百倍，可见对于一个系统不是简单地能实现其功能就可，而是要写出高质量的 SQL 语句，提高系统的可用性。

在多数情况下，Oracle 使用索引来更快地遍历表，优化器主要根据定义的索引来提高性能。但是，如果在 SQL 语句的 where 子句中写的 SQL 代码不合理，就会造成优化器删去索引而使用全表扫描，一般就这种 SQL 语句就是所谓的劣质 SQL 语句。在编写 SQL 语句时我们应清楚优化器根据何种原则来删除索引，这有助于写出高性能的 SQL 语句。

二、SQL 语句编写注意问题

下面就某些 SQL 语句的 where 子句编写中需要注意的问题作详细介绍。在这些 where 子句中，即使某些列存在索引，但是由于编写了劣质的 SQL，系统在运行该 SQL 语句时也不能使用该索引，而同样使用全表扫描，这就造成了响应速度的极大降低。

1、 IS NULL 或IS NOT NULL操作（判断字段是否为空）

不能用 null 作索引，任何包含 null 值的列都将不会被包含在索引中。即使索引有多列这样的情况下，只要这些列中有一列含有 null，该列就会从索引中排除。也就是说如果某列存在空值，即使对该列建索引也不会提高性能。

任何在 where 子句中使用 is null 或 is not null 的语句优化器是不允许使用索引的。

推荐方案：

用其它相同功能的操作运算代替，如

a is not null 改为 a>0 或 a>'' 等。

建立位图索引（有分区的表不能建，位图索引比较难控制，如字段值太多索引会使性能下降，多人更新操作会增加数据块锁的现象）

2、 联接列 '||'

对于有联接的列，即使最后的联接值为一个静态值，优化器是不会使用索引的。我们一起来看一个例子，假定有一个职工表（employee），对于一个职工的姓和名分成两列存放（FIRST_NAME和LAST_NAME），现在要查询一个叫比尔.克林顿（Bill Cliton）的职工。

下面是一个采用联接查询的SQL语句，

```
select * from employss
where
first_name||''||last_name ='Beill Cliton';
```

上面这条语句完全可以查询出是否有Bill Cliton这个员工，但是这里需要注意，系统优化器对基于

last_name创建的索引没有使用。

当采用下面这种SQL语句的编写，Oracle系统就可以采用基于last_name创建的索引。

```
Select * from employee
where
first_name ='Beill' and last_name ='Cliton';
```

遇到下面这种情况又如何处理呢？如果一个变量（name）中存放着Bill Cliton这个员工的姓名，对于这种情况我们又如何避免全程遍历，使用索引呢？可以使用一个函数，将变量name中的姓和名分开就可以了，但是有一点需要注意，这个函数是不能作用在索引列上。下面是SQL查询脚本：

```
select * from employee
where
first_name = SUBSTR('&&name',1,INSTR('&&name',' ')-1)
and
last_name = SUBSTR('&&name',INSTR('&&name' ,' ')+1)
```

3、带通配符（%）的like语句

同样以上面的例子来看这种情况。目前的需求是这样的，要求在职工表中查询名字中包含cliton的人。可以采用如下的查询SQL语句：

```
select * from employee where last_name like '%cliton%';
```

这里由于通配符（%）在搜寻词首出现，所以Oracle系统不使用last_name的索引，而是进行全表查询。在很多情况下可能无法避免这种情况，但是一定要心中有底，通配符如此使用会降低查询速度。然而当通配符出现在字符串其他位置时，优化器就能利用索引。在下面的查询中索引得到了使用：

```
select * from employee where last_name like 'c%';
```

4、 Order by语句

ORDER BY语句决定了Oracle如何将返回的查询结果排序。Order by语句对要排序的列没有什么特别的限制，也可以将函数加入列中（象联接或者附加等）。任何在Order by语句的非索引项或者有计算表达式都将降低查询速度。

仔细检查order by语句以找出非索引项或者表达式，它们会降低性能。解决这个问题的办法就是重写order by语句以使用索引，也可以为所使用的列建立另外一个索引，同时应绝对避免在order by子句中使用表达式。

5、 NOT

我们在查询时经常在where子句使用一些逻辑表达式，如大于、小于、等于以及不等于等等，也可以使用and（与）、or（或）以及not（非）。NOT可用来对任何逻辑运算符号取反。下面是一个NOT子句的例子：

```
... where not (status ='VALID')
```

如果要使用NOT，则应在取反的短语前面加上括号，并在短语前面加上NOT运算符。NOT运算符包含在另外一个逻辑运算符中，这就是不等于（<>）运算符。换句话说，即使不在查询where子句中显式地加入NOT词，NOT仍在运算符中，见下例：

```
... where status <>'INVALID';
```

再看下面这个例子：

```
select * from employee where  salary<>3000;
```

对这个查询，可以改写为不使用NOT：

```
select * from employee where  salary<3000 or salary>3000;
```

虽然这两种查询的结果一样，但是第二种查询方案会比第一种查询方案更快些。第二种查询允许Oracle对salary列使用索引，而第一种查询则不能使用索引。

6、 IN和EXISTS (存在于….)

有时候会将一列和一系列值相比较。最简单的办法就是在where子句中使用子查询。在where子句中可以使用两种格式的子查询。

第一种格式是使用IN操作符：

```
... where column in(select * from ... where ...);
```

第二种格式是使用EXIST操作符：

```
... where exists (select 'X' from ...where ...);
```

我相信绝大多数人会使用第一种格式，因为它比较容易编写，而实际上第二种格式要远比第一种格式的效率高。在Oracle中可以几乎将所有的IN操作符子查询改写为使用EXISTS的子查询。

第二种格式中，子查询以‘select 'X’开始。运用EXISTS子句不管子查询从表中抽取什么数据它只查看where子句。这样优化器就不必遍历整个表而仅根据索引就可完成工作（这里假定在where语句中使用的列存在索引）。相对于IN子句来说，EXISTS使用相连子查询，构造起来要比IN子查询困难一些。

通过使用EXISTS，Oracle系统会首先检查主查询，然后运行子查询直到它找到第一个匹配项，这就节省了时间。Oracle系统在执行IN子查询时，首先执行子查询，并将获得的结果列表存放在在一个加了索引的临时表中。在执行子查询之前，系统先将主查询挂起，待子查询执行完毕，存放在临时表中以后再执行主查询。这也就是使用EXISTS比使用IN通常查询速度快的原因。

同时应尽可能使用NOT EXISTS来代替NOT IN，尽管二者都使用了NOT（不能使用索引而降低速度），NOT EXISTS要比NOT IN查询效率更高。

用 IN 写出来的 SQL 的优点是比较容易写及清晰易懂，这比较适合现代软件开发的风格。
但是用 IN 的 SQL 性能总是比较低的，从 ORACLE 执行的步骤来分析用 IN 的 SQL 与不用 IN 的 SQL 有以下区别：

ORACLE 试图将其转换成多个表的连接，如果转换不成功则先执行 IN 里面的子查询，再查询外层的表记录，如果转换成功则直接采用多个表的连接方式查询。由此可见用 IN 的 SQL 至少多了一个转换的过程。一般的 SQL 都可以转换成功，但对于含有分组统计等方面的 SQL 就不能转换了。

推荐方案：在业务密集的 SQL 当中尽量不采用 IN 操作符。

NOT IN 操作符

此操作是强列推荐不使用的，因为它不能应用表的索引。

推荐方案：用 NOT EXISTS 或（外连接+判断为空）方案代替

7、<> 操作符（不等于）

不等于操作符是永远不会用到索引的，因此对它的处理只会产生全表扫描。

推荐方案：用其它相同功能的操作运算代替，如

a<>0 改为 a>0 or a<0

a<>'' 改为 a>''

8、> 及 < 操作符（大于或小于操作符）

大于或小于操作符一般情况下是不用调整的，因为它有索引就会采用索引查找，但有的情况下可以对它进行优化，如一个表有 100 万记录，一个数值型字段 A，30 万记录的 A=0，30 万记录的 A=1，39 万记录的 A=2，1 万记录的 A=3。那么执行 A>2 与 A>=3 的效果就有很大的区别了，因为 A>2 时 ORACLE 会先找出为 2 的记录索引再进行比较，而 A>=3 时 ORACLE 则直接找到=3 的记录索引。

9、IS NULL 或IS NOT NULL操作（判断字段是否为空）

判断字段是否为空一般是不会应用索引的，因为 B 树索引是不索引空值的。

推荐方案：

用其它相同功能的操作运算代替，如

a is not null 改为 a>0 或 a>”等。

不允许字段为空，而用一个缺省值代替空值，如业扩申请中状态字段不允许为空，缺省为申请。

建立位图索引（有分区的表不能建，位图索引比较难控制，如字段值太多索引会使性能下降，多人更新操作会增加数据块锁的现象）

10、union操作符

union 在进行表链接后会筛选掉重复的记录，所以在表链接后会对所产生的结果集进行排序运算，删除重复的记录再返回结果。实际大部分应用中是不会产生重复的记录，最常见的是过程表与历史表 union。如：

```
select * from gc_dfys
union
select * from ls_jg_dfys
```

这个 sql 在运行时先取出两个表的结果，再用排序空间进行排序删除重复的记录，最后返回结果集，如果表数据量大的话可能会导致用磁盘进行排序。

推荐方案：采用 union all 操作符替代 union，因为 union all 操作只是简单的将两个结果合并后就返回。

```
select * from gc_dfys
union all
select * from ls_jg_dfys
```

11、SELECT子句中避免使用‘*‘

当你想在SELECT子句中列出所有的COLUMN时,使用动态SQL列引用‘*’是一个方便的方法.不幸的是,这是一个非常低效的方法. 实际上,ORACLE在解析的过程中, 会将’*’依次转换成所有的列名, 这个工作是通过查询数据字典完成的, 这意味着将耗费更多的时间.

12、使用DECODE函数来减少处理时间

使用DECODE函数可以避免重复扫描相同记录或重复连接相同的表.

例如:

```
SELECT COUNT(*)，SUM(SAL)　FROM　EMP
WHERE DEPT_NO = 0020　AND ENAME LIKE　‘SMITH%’;

SELECT COUNT(*)，SUM(SAL)　FROM　EMP
WHERE DEPT_NO = 0030　AND ENAME LIKE　‘SMITH%’;
```

你可以用DECODE函数高效地得到相同结果：

```
SELECT COUNT(DECODE(DEPT_NO,0020,’X’,NULL)) D0020_COUNT,
 COUNT(DECODE(DEPT_NO,0030,’X’,NULL)) D0030_COUNT,
 SUM(DECODE(DEPT_NO,0020,SAL,NULL)) D0020_SAL,
 SUM(DECODE(DEPT_NO,0030,SAL,NULL)) D0030_SAL
FROM EMP WHERE ENAME LIKE ‘SMITH%’;
```

类似的,DECODE函数也可以运用于GROUP BY 和ORDER BY子句中.

13、计算记录条数

和一般的观点相反, count(*) 比count(1)稍快 ，当然如果可以通过索引检索,对索引列的计数仍旧是最快的. 例如 COUNT(EMPNO)

(译者按: 在 CSDN 论坛中,曾经对此有过相当热烈的讨论, 作者的观点并不十分准确,通过实际的测试,上述三种方法并没有显著的性能差别)

14、用Where子句替换HAVING子句

避免使用HAVING子句, HAVING 只会在检索出所有记录之后才对结果集进行过滤. 这个处理需要排序,总计等操作. 如果能通过WHERE子句限制记录的数目,那就能减少这方面的开销. 例如:

```
低效：SELECT REGION，AVG(LOG_SIZE)
      FROM LOCATION
      GROUP BY REGION
      HAVING REGION REGION != ‘SYDNEY’
      AND REGION != ‘PERTH’
高效：SELECT REGION，AVG(LOG_SIZE)
      FROM LOCATION
      WHERE REGION REGION != ‘SYDNEY’
      AND REGION != ‘PERTH’
      GROUP BY REGION
```

(译者按: HAVING 中的条件一般用于对一些集合函数的比较,如COUNT() 等等. 除此而外,一般的条件应该写在WHERE子句中)

15. 使用表的别名(Alias)

当在SQL语句中连接多个表时, 请使用表的别名并把别名前缀于每个Column上.这样一来,就可以减少解析的时间并减少那些由Column歧义引起的语法错误.

(译者注: Column歧义指的是由于SQL中不同的表具有相同的Column名,当SQL语句中出现这个Column时,SQL解析器无法判断这个Column的归属)

16、用EXISTS替代IN

在许多基于基础表的查询中,为了满足一个条件,往往需要对另一个表进行联接.在这种情况下, 使用EXISTS(或NOT EXISTS)通常将提高查询的效率.

低效: SELECT * FROM EMP (基础表) WHERE EMPNO > 0 AND DEPTNO IN (SELECT DEPTNO FROM DEPT WHERE LOC = 'MELB')

高效: SELECT * FROM EMP(基础表) WHERE EMPNO > 0 AND EXISTS (SELECT 'X' FROM DEPT WHERE DEPT.DEPTNO = EMP.DEPTNO AND LOC = 'MELB')

(译者按: 相对来说,用NOT EXISTS替换NOT IN 将更显著地提高效率,下一节中将指出)

17、用NOT EXISTS替代NOT IN

在子查询中,NOT IN子句将执行一个内部的排序和合并. 无论在哪种情况下,NOT IN都是最低效的 (因为它对子查询中的表执行了一个全表遍历). 为了避免使用NOT IN ,我们可以把它改写成外连接(Outer Joins)或NOT EXISTS.

例如:

```
SELECT …
FROM EMP
WHERE DEPT_NO NOT IN (SELECT DEPT_NO FROM DEPT WHERE DEPT_CAT='A');
```

为了提高效率.改写为:

(方法一: 高效)

```
SELECT ….
FROM EMP A,DEPT B
WHERE A.DEPT_NO = B.DEPT(+)
AND B.DEPT_NO IS NULL
AND B.DEPT_CAT(+) = 'A'
```

(方法二: 最高效)

```
SELECT ….
FROM EMP E
WHERE NOT EXISTS (SELECT 'X' FROM DEPT D WHERE D.DEPT_NO = E.DEPT_NO
AND DEPT_CAT = 'A');
```

18、用表连接替换EXISTS

通常来说 , 采用表连接的方式比EXISTS更有效率

```
SELECT ENAME
FROM EMP E
```

```
WHERE EXISTS (SELECT 'X' FROM DEPT WHERE DEPT_NO = E.DEPT_NO AND DEPT_CAT = 'A');
```

(更高效)

```
SELECT ENAME
FROM DEPT D,EMP E
WHERE E.DEPT_NO = D.DEPT_NO
AND DEPT_CAT = 'A' ;
```

(译者按: 在RBO的情况下,前者的执行路径包括FILTER,后者使用NESTED LOOP)

19、用EXISTS替换DISTINCT

当提交一个包含一对多表信息(比如部门表和雇员表)的查询时,避免在SELECT子句中使用DISTINCT. 一般可以考虑用EXIST替换

例如:

低效:

```
SELECT DISTINCT DEPT_NO,DEPT_NAME
FROM DEPT D,EMP E
WHERE D.DEPT_NO = E.DEPT_NO
```

高效:

```
SELECT DEPT_NO,DEPT_NAME
FROM DEPT D
WHERE EXISTS ( SELECT 'X' FROM EMP E WHERE E.DEPT_NO = D.DEPT_NO);
```

EXISTS 使查询更为迅速,因为RDBMS核心模块将在子查询的条件一旦满足后,立刻返回结果.

20、识别低效执行的SQL语句

用下列SQL工具找出低效SQL:

```
SELECT EXECUTIONS , DISK_READS, BUFFER_GETS,
ROUND((BUFFER_GETS-DISK_READS)/BUFFER_GETS,2) Hit_radio,
ROUND(DISK_READS/EXECUTIONS,2) Reads_per_run,
SQL_TEXT
FROM V$SQLAREA
WHERE EXECUTIONS>0
AND BUFFER_GETS > 0
AND (BUFFER_GETS-DISK_READS)/BUFFER_GETS < 0.8
ORDER BY 4 DESC;
```

(译者按: 虽然目前各种关于SQL优化的图形化工具层出不穷,但是写出自己的SQL工具来解决问题始终是一个最好的方法)

21、使用TKPROF 工具来查询SQL性能状态

SQL trace 工具收集正在执行的SQL的性能状态数据并记录到一个跟踪文件中. 这个跟踪文件提供了许多有用的信息,例如解析次数.执行次数,CPU使用时间等.这些数据将可以用来优化你的系统.

设置SQL TRACE在会话级别：有效
ALTER SESSION SET SQL_TRACE TRUE

设置SQL TRACE 在整个数据库有效，你必须将SQL_TRACE参数在init.ora中设为TRUE, USER_DUMP_DEST参数说明了生成跟踪文件的目录

(译者按: 这一节中,作者并没有提到TKPROF的用法, 对SQL TRACE的用法也不够准确, 设置SQL TRACE首先要在init.ora中设定TIMED_STATISTICS, 这样才能得到那些重要的时间状态. 生成的trace文件是不可读的,所以要用TKPROF工具对其进行转换,TKPROF有许多执行参数. 大家可以参考ORACLE手册来了解具体的配置.)

22、用EXPLAIN PLAN 分析SQL语句

EXPLAIN PLAN 是一个很好的分析SQL语句的工具,它甚至可以在不执行SQL的情况下分析语句. 通过分析,我们就可以知道ORACLE是怎么样连接表,使用什么方式扫描表(索引扫描或全表扫描)以及使用到的索引名称.

你需要按照从里到外,从上到下的次序解读分析的结果. EXPLAIN PLAN分析的结果是用缩进的格式排列的, 最内部的操作将被最先解读, 如果两个操作处于同一层中,带有最小操作号的将被首先执行.

NESTED LOOP是少数不按照上述规则处理的操作, 正确的执行路径是检查对NESTED LOOP提供数据的操作,其中操作号最小的将被最先处理.

译者按:
通过实践, 感到还是用SQLPLUS中的SET TRACE 功能比较方便.
举例:

```
SQL> list
1 SELECT *
2 FROM dept, emp
3* WHERE emp.deptno = dept.deptno
SQL> set autotrace traceonly /*traceonly 可以不显示执行结果*/
SQL> /
14 rows selected.
Execution Plan
----------------------------------------------------------
0 SELECT STATEMENT Optimizer=CHOOSE
1 0 NESTED LOOPS
2 1 TABLE ACCESS (FULL) OF 'EMP'
```

3 1 TABLE ACCESS (BY INDEX ROWID) OF 'DEPT'
4 3 INDEX (UNIQUE SCAN) OF 'PK_DEPT' (UNIQUE)
Statistics
----
0 recursive calls
2 db block gets
30 consistent gets
0 physical reads
0 redo size
2598 bytes sent via SQL*Net to client
503 bytes received via SQL*Net from client
2 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
14 rows processed

通过以上分析,可以得出实际的执行步骤是:
1. TABLE ACCESS (FULL) OF 'EMP'
2. INDEX (UNIQUE SCAN) OF 'PK_DEPT' (UNIQUE)
3. TABLE ACCESS (BY INDEX ROWID) OF 'DEPT'
4. NESTED LOOPS (JOINING 1 AND 3)

注: 目前许多第三方的工具如TOAD和ORACLE本身提供的工具如OMS的SQL Analyze都提供了极其方便的EXPLAIN PLAN工具.也许喜欢图形化界面的朋友们可以选用它们.

23、用索引提高效率

索引是表的一个概念部分,用来提高检索数据的效率. 实际上,ORACLE使用了一个复杂的自平衡B-tree结构. 通常,通过索引查询数据比全表扫描要快. 当ORACLE找出执行查询和Update语句的最佳路径时, ORACLE优化器将使用索引. 同样在联结多个表时使用索引也可以提高效率. 另一个使用索引的好处是,它提供了主键(primary key)的唯一性验证.

除了那些LONG或LONG RAW数据类型, 你可以索引几乎所有的列. 通常, 在大型表中使用索引特别有效. 当然,你也会发现, 在扫描小表时,使用索引同样能提高效率.

虽然使用索引能得到查询效率的提高,但是我们也必须注意到它的代价. 索引需要空间来
存储,也需要定期维护, 每当有记录在表中增减或索引列被修改时, 索引本身也会被修改. 这意味着每条记录的INSERT , DELETE , UPDATE将为此多付出 4 , 5 次的磁盘I/O . 因为索引需要额外的存储空间和处理, 那些不必要的索引反而会使查询反应时间变慢.

译者按:
定期的重构索引是有必要的.
ALTER INDEX <INDEXNAME> REBUILD <TABLESPACENAME>

24、索引的操作

ORACLE对索引有两种访问模式.

索引唯一扫描 (INDEX UNIQUE SCAN)

大多数情况下, 优化器通过WHERE子句访问INDEX.

例如:

表LODGING有两个索引 : 建立在LODGING列上的唯一性索引LODGING_PK和建立在MANAGER列上的非唯一性索引LODGING$MANAGER.

```
SELECT *
FROM LODGING
WHERE LODGING = ‘ROSE HILL';
```

在内部 , 上述SQL将被分成两步执行, 首先 , LODGING_PK 索引将通过索引唯一扫描的方式被访问 , 获得相对应的ROWID, 通过ROWID访问表的方式 执行下一步检索.

如果被检索返回的列包括在INDEX列中,ORACLE将不执行第二步的处理(通过ROWID访问表). 因为检索数据保存在索引中, 单单访问索引就可以完全满足查询结果.

下面SQL只需要INDEX UNIQUE SCAN 操作.

```
SELECT LODGING
FROM LODGING
WHERE LODGING = ‘ROSE HILL';
```

索引范围查询(INDEX RANGE SCAN)

适用于两种情况:

1. 基于一个范围的检索
2. 基于非唯一性索引的检索

例 1:

```
SELECT LODGING
FROM LODGING
WHERE LODGING LIKE ‘M%';
```

WHERE子句条件包括一系列值, ORACLE将通过索引范围查询的方式查询LODGING_PK . 由于索引范围查询将返回一组值, 它的效率就要比索引唯一扫描低一些.

例 2:

```
SELECT LODGING
FROM LODGING
WHERE MANAGER = ‘BILL GATES';
```

这个SQL的执行分两步, LODGING$MANAGER的索引范围查询(得到所有符合条件记录的ROWID) 和下一步同过ROWID访问表得到LODGING列的值. 由于LODGING$MANAGER是一个非唯一性的索引,数据库不能对它执行索引唯一扫描.

由于SQL返回LODGING列,而它并不存在于LODGING$MANAGER索引中, 所以在索引范围查询后会执行

一个通过ROWID访问表的操作.
WHERE子句中，如果索引列所对应的值的第一个字符由通配符(WILDCARD)开始，索引将不被采用.

SELECT LODGING
FROM LODGING
WHERE MANAGER LIKE ‘%HANMAN';
在这种情况下，ORACLE将使用全表扫描.

25、基础表的选择

基础表(Driving Table)是指被最先访问的表(通常以全表扫描的方式被访问). 根据优化器的不同，SQL语句中基础表的选择是不一样的.

如果你使用的是CBO (COST BASED OPTIMIZER),优化器会检查SQL语句中的每个表的物理大小,索引的状态,然后选用花费最低的执行路径.

如果你用RBO (RULE BASED OPTIMIZER)，并且所有的连接条件都有索引对应，在这种情况下，基础表就是FROM 子句中列在最后的那个表.

举例:
SELECT A.NAME , B.MANAGER
FROM WORKER A,
LODGING B
WHERE A.LODGING = B.LODING;
由于LODGING表的LODING列上有一个索引，而且WORKER表中没有相比较的索引，WORKER表将被作为查询中的基础表.

26、多个平等的索引

当SQL语句的执行路径可以使用分布在多个表上的多个索引时，ORACLE会同时使用多个索引并在运行时对它们的记录进行合并，检索出仅对全部索引有效的记录.

在ORACLE选择执行路径时,唯一性索引的等级高于非唯一性索引. 然而这个规则只有
当WHERE子句中索引列和常量比较才有效.如果索引列和其他表的索引类相比较. 这种子句在优化器中的等级是非常低的.

如果不同表中两个想同等级的索引将被引用, FROM子句中表的顺序将决定哪个会被率先使用. FROM子句中最后的表的索引将有最高的优先级.

如果相同表中两个想同等级的索引将被引用, WHERE子句中最先被引用的索引将有最高的优先级.
举例:
DEPTNO上有一个非唯一性索引,EMP_CAT也有一个非唯一性索引.
SELECT ENAME, FROM EMP WHERE DEPT_NO = 20 AND EMP_CAT = ‘A';
这里,DEPTNO索引将被最先检索,然后同EMP_CAT索引检索出的记录进行合并. 执行路径如下:

TABLE ACCESS BY ROWID ON EMP
AND-EQUAL
INDEX RANGE SCAN ON DEPT_IDX
INDEX RANGE SCAN ON CAT_IDX

27、等式比较和范围比较

当WHERE子句中有索引列, ORACLE不能合并它们,ORACLE将用范围比较.
举例:
DEPTNO上有一个非唯一性索引,EMP_CAT也有一个非唯一性索引.

```
SELECT ENAME
FROM EMP
WHERE DEPTNO > 20
AND EMP_CAT = ‘A';
```

这里只有EMP_CAT索引被用到,然后所有的记录将逐条与DEPTNO条件进行比较. 执行路径如下:

```
TABLE ACCESS BY ROWID ON EMP
INDEX RANGE SCAN ON CAT_IDX
```

28、不明确的索引等级

当ORACLE无法判断索引的等级高低差别,优化器将只使用一个索引,它就是在WHERE子句中被列在最前面的.
举例:
DEPTNO上有一个非唯一性索引,EMP_CAT也有一个非唯一性索引.

```
SELECT ENAME
FROM EMP
WHERE DEPTNO > 20
AND EMP_CAT > ‘A';
```

这里, ORACLE只用到了DEPT_NO索引. 执行路径如下:

```
TABLE ACCESS BY ROWID ON EMP
INDEX RANGE SCAN ON DEPT_IDX
```

译者按:
我们来试一下以下这种情况:

```
SQL> select index_name, uniqueness from user_indexes where table_name = 'EMP';
INDEX_NAME UNIQUENES
------------------------------ ---------
EMPNO UNIQUE
EMPTYPE NONUNIQUE
SQL> select * from emp where empno >= 2 and emp_type = 'A' ;
```

no rows selected

Execution Plan
----
0 SELECT STATEMENT Optimizer=CHOOSE
1 0 TABLE ACCESS (BY INDEX ROWID) OF 'EMP'
2 1 INDEX (RANGE SCAN) OF 'EMPTYPE' (NON-UNIQUE)

虽然EMPNO是唯一性索引,但是由于它所做的是范围比较, 等级要比非唯一性索引的等式比较低!

29、强制索引失效

如果两个或以上索引具有相同的等级,你可以强制命令ORACLE优化器使用其中的一个(通过它,检索出的记录数量少) .
举例:

```
SELECT ENAME
FROM EMP
WHERE EMPNO = 7935
AND DEPTNO + 0 = 10 /*DEPTNO上的索引将失效*/
AND EMP_TYPE || '' = 'A' /*EMP_TYPE上的索引将失效*/
```

这是一种相当直接的提高查询效率的办法. 但是你必须谨慎考虑这种策略,一般来说,只有在你希望单独优化几个SQL时才能采用它.
这里有一个例子关于何时采用这种策略,
假设在EMP表的EMP_TYPE列上有一个非唯一性的索引而EMP_CLASS上没有索引.

```
SELECT ENAME
FROM EMP
WHERE EMP_TYPE = 'A'
AND EMP_CLASS = 'X';
```

优化器会注意到EMP_TYPE上的索引并使用它. 这是目前唯一的选择. 如果,一段时间以后, 另一个非唯一性建立在EMP_CLASS上,优化器必须对两个索引进行选择,在通常情况下,优化器将使用两个索引并在他们的结果集合上执行排序及合并. 然而,如果其中一个索引（EMP_TYPE）接近于唯一性而另一个索引（EMP_CLASS）上有几千个重复的值. 排序及合并就会成为一种不必要的负担. 在这种情况下,你希望使优化器屏蔽掉EMP_CLASS索引.

用下面的方案就可以解决问题.

```
SELECT ENAME
FROM EMP
WHERE EMP_TYPE = 'A'
AND EMP_CLASS||'' = 'X';
```

30、避免在索引列上使用计算.

WHERE子句中，如果索引列是函数的一部分．优化器将不使用索引而使用全表扫描．
举例:
低效：
SELECT …
FROM DEPT
WHERE SAL * 12 > 25000;
高效:
SELECT …
FROM DEPT
WHERE SAL > 25000/12;
译者按:
这是一个非常实用的规则，请务必牢记

31、自动选择索引

如果表中有两个以上（包括两个）索引，其中有一个唯一性索引，而其他是非唯一性．
在这种情况下，ORACLE将使用唯一性索引而完全忽略非唯一性索引．
举例:
SELECT ENAME
FROM EMP
WHERE EMPNO = 2326
AND DEPTNO = 20 ;

这里，只有EMPNO上的索引是唯一性的，所以EMPNO索引将用来检索记录．
TABLE ACCESS BY ROWID ON EMP
INDEX UNIQUE SCAN ON EMP_NO_IDX

32、避免在索引列上使用NOT

通常，　我们要避免在索引列上使用NOT, NOT会产生在和在索引列上使用函数相同的影响．当ORACLE”遇到”NOT,他就会停止使用索引转而执行全表扫描.
举例:
低效: (这里,不使用索引)
SELECT …
FROM DEPT
WHERE DEPT_CODE NOT = 0;
高效: (这里,使用了索引)
SELECT …
FROM DEPT
WHERE DEPT_CODE > 0;

需要注意的是,在某些时候, ORACLE优化器会自动将NOT转化成相对应的关系操作符.
NOT > to <=

NOT >= to <

NOT < to >=

NOT <= to >
译者按:
在这个例子中,作者犯了一些错误. 例子中的低效率SQL是不能被执行的.
我做了一些测试:
SQL> select * from emp where NOT empno > 1;
no rows selected
Execution Plan
----
0 SELECT STATEMENT Optimizer=CHOOSE
1 0 TABLE ACCESS (BY INDEX ROWID) OF 'EMP'
2 1 INDEX (RANGE SCAN) OF 'EMPNO' (UNIQUE)

SQL> select * from emp where empno <= 1;
no rows selected
Execution Plan
----
0 SELECT STATEMENT Optimizer=CHOOSE
1 0 TABLE ACCESS (BY INDEX ROWID) OF 'EMP'
2 1 INDEX (RANGE SCAN) OF 'EMPNO' (UNIQUE)

两者的效率完全一样，也许这符合作者关于”在某些时候，ORACLE优化器会自动将NOT转化成相对应的关系操作符”的观点.

33、用>=替代>

如果DEPTNO上有一个索引,
高效:
SELECT *
FROM EMP
WHERE DEPTNO >=4

低效:
SELECT *
FROM EMP
WHERE DEPTNO >3

两者的区别在于，前者 DBMS 将直接跳到第一个 DEPT 等于 4 的记录而后者将首先定位到 DEPTNO=3 的记录并且向前扫描到第一个 DEPT 大于 3 的记录.

34 用UNION替换OR (适用于索引列)

通常情况下，用UNION替换WHERE子句中的OR将会起到较好的效果．对索引列使用OR将造成全表扫描．注意，以上规则只针对多个索引列有效．如果有column没有被索引，查询效率可能会因为你没有选择OR而降低．

在下面的例子中, LOC_ID 和REGION上都建有索引．
高效:
SELECT LOC_ID , LOC_DESC , REGION
FROM LOCATION
WHERE LOC_ID = 10
UNION
SELECT LOC_ID , LOC_DESC , REGION
FROM LOCATION
WHERE REGION = “MELBOURNE”

低效:
SELECT LOC_ID , LOC_DESC , REGION
FROM LOCATION
WHERE LOC_ID = 10 OR REGION = “MELBOURNE”

如果你坚持要用OR，那就需要返回记录最少的索引列写在最前面．

注意:
WHERE KEY1 = 10 (返回最少记录)
OR KEY2 = 20 (返回最多记录)
ORACLE 内部将以上转换为
WHERE KEY1 = 10 AND
((NOT KEY1 = 10) AND KEY2 = 20)
译者按:
下面的测试数据仅供参考: (a = 1003 返回一条记录 , b = 1 返回 1003 条记录)
SQL> select * from unionvsor /*1st test*/
2 where a = 1003 or b = 1;
1003 rows selected.
Execution Plan
----
0 SELECT STATEMENT Optimizer=CHOOSE
1 0 CONCATENATION
2 1 TABLE ACCESS (BY INDEX ROWID) OF 'UNIONVSOR'
3 2 INDEX (RANGE SCAN) OF 'UB' (NON-UNIQUE)
4 1 TABLE ACCESS (BY INDEX ROWID) OF 'UNIONVSOR'
5 4 INDEX (RANGE SCAN) OF 'UA' (NON-UNIQUE)
Statistics
----
0 recursive calls

```
0 db block gets
144 consistent gets
0 physical reads
0 redo size
63749 bytes sent via SQL*Net to client
7751 bytes received via SQL*Net from client
68 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
1003 rows processed
SQL> select * from unionvsor /*2nd test*/
2 where b = 1 or a = 1003 ;
1003 rows selected.
Execution Plan
----------------------------------------------------------
0 SELECT STATEMENT Optimizer=CHOOSE
1 0 CONCATENATION
2 1 TABLE ACCESS (BY INDEX ROWID) OF 'UNIONVSOR'
3 2 INDEX (RANGE SCAN) OF 'UA' (NON-UNIQUE)
4 1 TABLE ACCESS (BY INDEX ROWID) OF 'UNIONVSOR'
5 4 INDEX (RANGE SCAN) OF 'UB' (NON-UNIQUE)
Statistics
----------------------------------------------------------
0 recursive calls
0 db block gets
143 consistent gets
0 physical reads
0 redo size
63749 bytes sent via SQL*Net to client
7751 bytes received via SQL*Net from client
68 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
1003 rows processed

SQL> select * from unionvsor /*3rd test*/
2 where a = 1003
3 union
4 select * from unionvsor
5 where b = 1;
1003 rows selected.
Execution Plan
----------------------------------------------------------
0 SELECT STATEMENT Optimizer=CHOOSE
```

1 0 SORT (UNIQUE)
2 1 UNION-ALL
3 2 TABLE ACCESS (BY INDEX ROWID) OF 'UNIONVSOR'
4 3 INDEX (RANGE SCAN) OF 'UA' (NON-UNIQUE)
5 2 TABLE ACCESS (BY INDEX ROWID) OF 'UNIONVSOR'
6 5 INDEX (RANGE SCAN) OF 'UB' (NON-UNIQUE)
Statistics
----
0 recursive calls
0 db block gets
10 consistent gets
0 physical reads
0 redo size
63735 bytes sent via SQL*Net to client
7751 bytes received via SQL*Net from client
68 SQL*Net roundtrips to/from client
1 sorts (memory)
0 sorts (disk)
1003 rows processed
用UNION的效果可以从consistent gets和 SQL*NET的数据交换量的减少看出

35、用IN来替换OR

下面的查询可以被更有效率的语句替换:
低效:
SELECT….
FROM LOCATION
WHERE LOC_ID = 10
OR LOC_ID = 20
OR LOC_ID = 30

高效
SELECT…
FROM LOCATION
WHERE LOC_IN IN (10,20,30);

译者按:
这是一条简单易记的规则，但是实际的执行效果还须检验，在ORACLE8i下，两者的执行路径似乎是相同的.

36、避免在索引列上使用IS NULL和IS NOT NULL

避免在索引中使用任何可以为空的列，ORACLE将无法使用该索引 . 对于单列索引，如果列包含空值，索引中将不存在此记录. 对于复合索引，如果每个列都为空，索引中同样不存在此记录. 如果至少有一个列不为空，则记录存在于索引中.

举例:
如果唯一性索引建立在表的A列和B列上，并且表中存在一条记录的A,B值为(123,null) ，ORACLE将不接受下一条具有相同A,B值（123,null）的记录(插入). 然而如果 所有的索引列都为空，ORACLE将认为整个键值为空而空不等于空. 因此你可以插入 1000 条具有相同键值的记录,当然它们都是空!
因为空值不存在于索引列中,所以WHERE子句中对索引列进行空值比较将使ORACLE停用该索引.

举例:
低效: (索引失效)
SELECT …
FROM DEPARTMENT
WHERE DEPT_CODE IS NOT NULL;

高效: (索引有效)
SELECT …
FROM DEPARTMENT
WHERE DEPT_CODE >=0;

37、总是使用索引的第一个列

如果索引是建立在多个列上，只有在它的第一个列(leading column)被where子句引用时,优化器才会选择使用该索引.
译者按:
这也是一条简单而重要的规则. 见以下实例.

```
SQL> create table multiindexusage ( inda number , indb number , descr varchar2(10));
Table created.
SQL> create index multindex on multiindexusage(inda,indb);
Index created.
SQL> set autotrace traceonly

SQL> select * from multiindexusage where inda = 1;
Execution Plan
----------------------------------------------------------
0 SELECT STATEMENT Optimizer=CHOOSE

1 0 TABLE ACCESS (BY INDEX ROWID) OF 'MULTIINDEXUSAGE'

2 1 INDEX (RANGE SCAN) OF 'MULTINDEX' (NON-UNIQUE)

SQL> select * from multiindexusage where indb = 1;

Execution Plan
----------------------------------------------------------

0 SELECT STATEMENT Optimizer=CHOOSE
```

1 0 TABLE ACCESS (FULL) OF 'MULTIINDEXUSAGE'

很明显，当仅引用索引的第二个列时,优化器使用了全表扫描而忽略了索引

38、ORACLE内部操作

当执行查询时,ORACLE采用了内部的操作．下表显示了几种重要的内部操作.

ORACLE Clause
内部操作

ORDER BY
SORT ORDER BY

UNION
UNION-ALL

MINUS
MINUS

INTERSECT
INTERSECT

DISTINCT,MINUS,INTERSECT,UNION
SORT UNIQUE

MIN,MAX,COUNT
SORT AGGREGATE

GROUP BY
SORT GROUP BY

ROWNUM
COUNT or COUNT STOPKEY

Queries involving Joins
SORT JOIN,MERGE JOIN,NESTED LOOPS

CONNECT BY
CONNECT BY

39、用UNION-ALL 替换UNION (如果有可能的话)

当SQL语句需要UNION两个查询结果集合时,这两个结果集合会以UNION-ALL的方式被合并, 然后在输出最终结果前进行排序.
如果用UNION ALL替代UNION, 这样排序就不是必要了. 效率就会因此得到提高.

举例:
低效:

```
    SELECT ACCT_NUM, BALANCE_AMT FROM DEBIT_TRANSACTIONS
WHERE TRAN_DATE = '31-DEC-95'
UNION
SELECT ACCT_NUM, BALANCE_AMT
FROM DEBIT_TRANSACTIONS
WHERE TRAN_DATE = '31-DEC-95'
```

高效:

```
SELECT ACCT_NUM, BALANCE_AMT
FROM DEBIT_TRANSACTIONS
WHERE TRAN_DATE = '31-DEC-95'
UNION ALL
SELECT ACCT_NUM, BALANCE_AMT
FROM DEBIT_TRANSACTIONS
WHERE TRAN_DATE = '31-DEC-95'
```

译者按:
需要注意的是, UNION ALL 将重复输出两个结果集合中相同记录. 因此各位还是要从业务需求分析使用UNION ALL的可行性.
UNION 将对结果集合排序,这个操作会使用到SORT_AREA_SIZE这块内存. 对于这块内存的优化也是相当重要的. 下面的SQL可以用来查询排序的消耗量

```
Select substr(name,1,25) "Sort Area Name",
substr(value,1,15) "Value"
from v$sysstat
where name like 'sort%'
```

40、使用提示(Hints)

对于表的访问,可以使用两种Hints.
FULL 和 ROWID
FULL hint 告诉ORACLE使用全表扫描的方式访问指定表.
例如:

```
SELECT /*+ FULL(EMP) */ *
FROM EMP
WHERE EMPNO = 7893;
```

ROWID hint 告诉ORACLE使用TABLE ACCESS BY ROWID的操作访问表.

通常，你需要采用TABLE ACCESS BY ROWID的方式特别是当访问大表的时候，使用这种方式，你需要知道ROIWD的值或者使用索引.

如果一个大表没有被设定为缓存(CACHED)表而你希望它的数据在查询结束是仍然停留在SGA中,你就可以使用CACHE hint 来告诉优化器把数据保留在SGA中. 通常CACHE hint 和 FULL hint 一起使用.
例如:

```
SELECT /*+ FULL(WORKER) CACHE(WORKER)*/ *
FROM WORK;
```

索引hint 告诉ORACLE使用基于索引的扫描方式. 你不必说明具体的索引名称
例如:

```
SELECT /*+ INDEX(LODGING) */ LODGING
FROM LODGING
WHERE MANAGER = ‘BILL GATES';
```

在不使用hint的情况下，以上的查询应该也会使用索引,然而,如果该索引的重复值过多而你的优化器是CBO,优化器就可能忽略索引. 在这种情况下，你可以用INDEX hint强制ORACLE使用该索引.

ORACLE hints 还包括ALL_ROWS, FIRST_ROWS, RULE,USE_NL, USE_MERGE, USE_HASH 等等.
译者按:
使用hint，表示我们对ORACLE优化器缺省的执行路径不满意,需要手工修改.
这是一个很有技巧性的工作. 我建议只针对特定的,少数的SQL进行hint的优化.
对ORACLE的优化器还是要有信心(特别是CBO)

41、用WHERE替代ORDER BY

ORDER BY 子句只在两种严格的条件下使用索引.
ORDER BY中所有的列必须包含在相同的索引中并保持在索引中的排列顺序.
ORDER BY中所有的列必须定义为非空.
WHERE子句使用的索引和ORDER BY子句中所使用的索引不能并列.
例如:
表DEPT包含以下列:

```
DEPT_CODE PK NOT NULL
DEPT_DESC NOT NULL
DEPT_TYPE NULL
```

非唯一性的索引(DEPT_TYPE)

低效: (索引不被使用)

```
SELECT DEPT_CODE
FROM DEPT
ORDER BY DEPT_TYPE
EXPLAIN PLAN:
```

SORT ORDER BY
TABLE ACCESS FULL

高效: (使用索引)
SELECT DEPT_CODE
FROM DEPT
WHERE DEPT_TYPE > 0
EXPLAIN PLAN:
TABLE ACCESS BY ROWID ON EMP
INDEX RANGE SCAN ON DEPT_IDX
译者按:
ORDER BY 也能使用索引! 这的确是个容易被忽视的知识点. 我们来验证一下:

SQL> select * from emp order by empno;

Execution Plan
---

0 SELECT STATEMENT Optimizer=CHOOSE

1 0 TABLE ACCESS (BY INDEX ROWID) OF 'EMP'

2 1 INDEX (FULL SCAN) OF 'EMPNO' (UNIQUE)

42、避免改变索引列的类型.

当比较不同数据类型的数据时, ORACLE自动对列进行简单的类型转换.

假设 EMPNO是一个数值类型的索引列.
SELECT …
FROM EMP
WHERE EMPNO = '123'
实际上,经过ORACLE类型转换, 语句转化为:
SELECT …
FROM EMP
WHERE EMPNO = TO_NUMBER('123')

幸运的是,类型转换没有发生在索引列上,索引的用途没有被改变.

现在,假设EMP_TYPE是一个字符类型的索引列.
SELECT …
FROM EMP
WHERE EMP_TYPE = 123

这个语句被ORACLE转换为:
SELECT …
FROM EMP
WHERE TO_NUMBER(EMP_TYPE)=123

因为内部发生的类型转换, 这个索引将不会被用到!
译者按:
为了避免ORACLE对你的SQL进行隐式的类型转换, 最好把类型转换用显式表现出来. 注意当字符和数值比较时, ORACLE会优先转换数值类型到字符类型.

43、需要当心的WHERE子句

某些SELECT 语句中的WHERE子句不使用索引. 这里有一些例子.

在下面的例子里, '!=' 将不使用索引. 记住, 索引只能告诉你什么存在于表中, 而不能告诉你什么不存在于表中.
不使用索引:
SELECT ACCOUNT_NAME
FROM TRANSACTION
WHERE AMOUNT !=0;
使用索引:
SELECT ACCOUNT_NAME
FROM TRANSACTION
WHERE AMOUNT >0;

下面的例子中, '||'是字符连接函数. 就象其他函数那样, 停用了索引.
不使用索引:
SELECT ACCOUNT_NAME,AMOUNT
FROM TRANSACTION
WHERE ACCOUNT_NAME||ACCOUNT_TYPE='AMEXA';
使用索引:
SELECT ACCOUNT_NAME,AMOUNT
FROM TRANSACTION
WHERE ACCOUNT_NAME = 'AMEX'
AND ACCOUNT_TYPE=' A';

下面的例子中, '+'是数学函数. 就象其他数学函数那样, 停用了索引.
不使用索引:
SELECT ACCOUNT_NAME, AMOUNT
FROM TRANSACTION
WHERE AMOUNT + 3000 >5000;
使用索引:

SELECT ACCOUNT_NAME, AMOUNT
FROM TRANSACTION
WHERE AMOUNT > 2000 ;
下面的例子中,相同的索引列不能互相比较,这将会启用全表扫描.

不使用索引:
SELECT ACCOUNT_NAME, AMOUNT
FROM TRANSACTION
WHERE ACCOUNT_NAME = NVL(:ACC_NAME,ACCOUNT_NAME);
使用索引:
SELECT ACCOUNT_NAME, AMOUNT
FROM TRANSACTION
WHERE ACCOUNT_NAME LIKE NVL(:ACC_NAME,'%');
译者按:

如果一定要对使用函数的列启用索引, ORACLE新的功能: 基于函数的索引(Function-Based Index) 也许是一个较好的方案.
CREATE INDEX EMP_I ON EMP (UPPER(ename)); /*建立基于函数的索引*/

SELECT * FROM emp WHERE UPPER(ename) = 'BLACKSNAIL'; /*将使用索引*/
44、连接多个扫描

如果你对一个列和一组有限的值进行比较, 优化器可能执行多次扫描并对结果进行合并连接.

举例:
SELECT *
FROM LODGING
WHERE MANAGER IN ('BILL GATES','KEN MULLER');

优化器可能将它转换成以下形式
SELECT *
FROM LODGING
WHERE MANAGER = 'BILL GATES'
OR MANAGER = 'KEN MULLER';

当选择执行路径时, 优化器可能对每个条件采用LODGING$MANAGER上的索引范围扫描. 返回的ROWID用来访问LODGING表的记录 (通过TABLE ACCESS BY ROWID 的方式). 最后两组记录以连接(CONCATENATION)的形式被组合成一个单一的集合.

Explain Plan :
SELECT STATEMENT Optimizer=CHOOSE
CONCATENATION
TABLE ACCESS (BY INDEX ROWID) OF LODGING

INDEX (RANGE SCAN) OF LODGING$MANAGER (NON-UNIQUE)
TABLE ACCESS (BY INDEX ROWID) OF LODGING
INDEX (RANGE SCAN) OF LODGING$MANAGER (NON-UNIQUE)
译者按:
本节和第 37 节似乎有矛盾之处.

45、CBO下使用更具选择性的索引

基于成本的优化器(CBO, Cost-Based Optimizer)对索引的选择性进行判断来决定索引的使用是否能提高效率.
如果索引有很高的选择性, 那就是说对于每个不重复的索引键值,只对应数量很少的记录.
比如, 表中共有 100 条记录而其中有 80 个不重复的索引键值. 这个索引的选择性就是 80/100 = 0.8 . 选择性越高, 通过索引键值检索出的记录就越少.
如果索引的选择性很低, 检索数据就需要大量的索引范围查询操作和ROWID 访问表的操作. 也许会比全表扫描的效率更低.

译者按:
下列经验请参阅:

a. 如果检索数据量超过 30%的表中记录数.使用索引将没有显著的效率提高.
b. 在特定情况下, 使用索引也许会比全表扫描慢, 但这是同一个数量级上的区别. 而通常情况下,使用索引比全表扫描要块几倍乃至几千倍!

46、避免使用耗费资源的操作

带有DISTINCT,UNION,MINUS,INTERSECT,ORDER BY的SQL语句会启动SQL引擎执行耗费资源的排序(SORT)功能. DISTINCT需要一次排序操作, 而其他的至少需要执行两次排序.
例如,一个UNION查询,其中每个查询都带有GROUP BY子句, GROUP BY会触发嵌入排序(NESTED SORT) ; 这样, 每个查询需要执行一次排序, 然后在执行UNION时, 又一个唯一排序(SORT UNIQUE)操作被执行而且它只能在前面的嵌入排序结束后才能开始执行. 嵌入的排序的深度会大大影响查询的效率.

通常, 带有UNION, MINUS , INTERSECT的SQL语句都可以用其他方式重写.
译者按:
如果你的数据库的SORT_AREA_SIZE调配得好, 使用UNION , MINUS, INTERSECT也是可以考虑的, 毕竟它们的可读性很强

47、优化GROUP BY

提高GROUP BY 语句的效率, 可以通过将不需要的记录在GROUP BY 之前过滤掉.下面两个查询返回相同结果但第二个明显就快了许多.

低效:
SELECT JOB , AVG(SAL)

FROM EMP
GROUP JOB
HAVING JOB = ‘PRESIDENT'
OR JOB = ‘MANAGER'
高效:
SELECT JOB , AVG(SAL)
FROM EMP
WHERE JOB = ‘PRESIDENT'
OR JOB = ‘MANAGER'
GROUP JOB

译者按:
本节和 14 节相同. 可略过.

48、使用日期

当使用日期是,需要注意如果有超过 5 位小数加到日期上, 这个日期会进到下一天!

例如:
1.
SELECT TO_DATE(‘01-JAN-93'+.99999)
FROM DUAL;
Returns:
'01-JAN-93 23:59:59'
2.
SELECT TO_DATE(‘01-JAN-93'+.999999)
FROM DUAL;
Returns:
'02-JAN-93 00:00:00'

译者按:
虽然本节和SQL性能优化没有关系, 但是作者的功力可见一斑

49、使用显式的游标(CURSORS)

使用隐式的游标,将会执行两次操作. 第一次检索记录, 第二次检查TOO MANY ROWS 这个exception . 而显式游标不执行第二次操作.

50、优化EXPORT和IMPORT

使用较大的BUFFER(比如 10MB , 10,240,000)可以提高EXPORT和IMPORT的速度.

ORACLE将尽可能地获取你所指定的内存大小,即使在内存不满足,也不会报错.这个值至少要和表中最大的列相当,否则列值会被截断.

译者按:

可以肯定的是, 增加BUFFER会大大提高EXPORT , IMPORT的效率. (曾经碰到过一个CASE, 增加BUFFER后,IMPORT/EXPORT快了 10 倍!)

作者可能犯了一个错误: “这个值至少要和表中最大的列相当,否则列值会被截断. “

其中最大的列也许是指最大的记录大小.

关于EXPORT/IMPORT的优化,CSDN论坛中有一些总结性的贴子,比如关于BUFFER参数, COMMIT参数等等, 详情请查.

51、分离表和索引

总是将你的表和索引建立在不同的表空间内(TABLESPACES). 决不要将不属于ORACLE内部系统的对象存放到SYSTEM表空间里. 同时,确保数据表空间和索引表空间置于不同的硬盘上.

译者按:
“同时,确保数据表空间和索引表空间置与不同的硬盘上.”可能改为如下更为准确 “同时,确保数据表空间和索引表空间置与不同的硬盘控制卡控制的硬盘上.”

三、SQL书写的影响

同一功能同一性能不同写法SQL的影响

如一个 SQL 在 A 程序员写的为

Select * from zl_yhjbqk

B 程序员写的为

Select * from dlyx.zl_yhjbqk（带表所有者的前缀）

C 程序员写的为

Select * from DLYX.ZLYHJBQK（大写表名）

D 程序员写的为

Select * from DLYX.ZLYHJBQK（中间多了空格）

以上四个 SQL 在 ORACLE 分析整理之后产生的结果及执行的时间是一样的,但是从 ORACLE 共享内存 SGA 的原理，可以得出 ORACLE 对每个 SQL 都会对其进行一次分析，并且占用共享内存，如果将 SQL 的字符串及格式写得完全相同则 ORACLE 只会分析一次，共享内存也只会留下一次的分析结果，这不仅可以减少分析 SQL 的时间，而且可以减少共享内存重复的信息，ORACLE 也可以准确统计 SQL 的执行频率。

四、WHERE后面的条件顺序影响

WHERE 子句后面的条件顺序对大数据量表的查询会产生直接的影响，如：

Select * from zl_yhjbqk where dy_dj = '1KV以下' and xh_bz=1

Select * from zl_yhjbqk where xh_bz=1 and dy_dj = '1KV以下'

以上两个SQL中dy_dj（电压等级）及xh_bz（销户标志）两个字段都没进行索引，所以执行的时候都是全表扫描，第一条SQL的**dy_dj = '1KV以下'**条件在记录集内比率为 99%，而**xh_bz=1**的比率只为 0.5%，在进行第一条SQL的时候 99%条记录都进行**dy_dj及xh_bz**的比较，而在进行第二条SQL的时候 0.5%条记录都进行**dy_dj及xh_bz**的比较，以此可以得出第二条SQL的CPU占用率明显比第一条低。

五、查询表顺序的影响

在 FROM 后面的表中的列表顺序会对 SQL 执行性能影响，在没有索引及 ORACLE 没有对表进行统计分析的情况下 ORACLE 会按表出现的顺序进行链接，由此因为表的顺序不对会产生十分耗服务器资源的数据交叉。（注：如果对表进行了统计分析，ORACLE 会自动先进小表的链接，再进行大表的链接）

六、SQL语句索引的利用

对操作符的优化（见上节）

对条件字段的一些优化

采用函数处理的字段不能利用索引，如：

substr(hbs_bh,1,4)=’5400’，优化处理：hbs_bh like ‘5400%’

trunc(sk_rq)=trunc(sysdate)， 优化处理：sk_rq>=trunc(**sysdate**) **and** sk_rq<trunc(**sysdate+1**)

进行了显式或隐式的运算的字段不能进行索引，如： ss_df+20>50，优化处理：ss_df>30

‘X’||hbs_bh>’X5400021452’，优化处理：hbs_bh>’5400021542’

sk_rq+5=sysdate，优化处理：sk_rq=sysdate-5

hbs_bh=5401002554，优化处理：hbs_bh=’ 5401002554’，**注：**此条件对hbs_bh 进行隐式的to_number转换，因为hbs_bh字段是字符型。

条件内包括了多个本表的字段运算时不能进行索引，如：

ys_df>cx_df，无法进行优化

qc_bh||kh_bh=’5400250000’，优化处理：qc_bh=’5400’ and kh_bh=’250000’

七、**应用ORACLE的HINT（提示）处理**

提示处理是在 ORACLE 产生的 SQL 分析执行路径不满意的情况下要用到的。它可以对 SQL 进行以下方面的提示

目标方面的提示：

COST（按成本优化）

RULE（按规则优化）

CHOOSE（缺省）（ORACLE 自动选择成本或规则进行优化）

ALL_ROWS（所有的行尽快返回）

FIRST_ROWS（第一行数据尽快返回）

执行方法的提示：

USE_NL（使用 NESTED LOOPS 方式联合）

USE_MERGE（使用 MERGE JOIN 方式联合）

USE_HASH（使用 HASH JOIN 方式联合）

索引提示：

INDEX（TABLE INDEX）（使用提示的表索引进行查询）

其它高级提示（如并行处理等等）

ORACLE 的提示功能是比较强的功能，也是比较复杂的应用，并且提示只是给 ORACLE 执行的一个建议，有时如果出于成本方面的考虑 ORACLE 也可能不会按提示进行。根据实践应用，一

般不建议开发人员应用 ORACLE 提示，因为各个数据库及服务器性能情况不一样，很可能一个地方性能提升了，但另一个地方却下降了，ORACLE 在 SQL 执行分析方面已经比较成熟，如果分析执行的路径不对首先应在数据库结构（主要是索引）、服务器当前性能（共享内存、磁盘文件碎片）、数据库对象（表、索引）统计信息是否正确这几方面分析。

操作符优化

IN 操作符

用 IN 写出来的 SQL 的优点是比较容易写及清晰易懂，这比较适合现代软件开发的风格。

但是用 IN 的 SQL 性能总是比较低的，从 ORACLE 执行的步骤来分析用 IN 的 SQL 与不用 IN 的 SQL 有以下区别：

ORACLE 试图将其转换成多个表的连接，如果转换不成功则先执行 IN 里面的子查询，再查询外层的表记录，如果转换成功则直接采用多个表的连接方式查询。由此可见用 IN 的 SQL 至少多了一个转换的过程。一般的 SQL 都可以转换成功，但对于含有分组统计等方面的 SQL 就不能转换了。

推荐方案：在业务密集的 SQL 当中尽量不采用 IN 操作符。

NOT IN操作符

此操作是强列推荐不使用的，因为它不能应用表的索引。

推荐方案：用 NOT EXISTS 或（外连接+判断为空）方案代替

<> 操作符（不等于）

不等于操作符是永远不会用到索引的，因此对它的处理只会产生全表扫描。

推荐方案：用其它相同功能的操作运算代替，如

a<>0 改为 a>0 or a<0

a<>” 改为 a>”

IS NULL 或IS NOT NULL操作（判断字段是否为空）

判断字段是否为空一般是不会应用索引的，因为 B 树索引是不索引空值的。

推荐方案：

用其它相同功能的操作运算代替，如

a is not null 改为 a>0 或 a>”等。

不允许字段为空，而用一个缺省值代替空值，如业扩申请中状态字段不允许为空，缺省为申请。

建立位图索引（有分区的表不能建，位图索引比较难控制，如字段值太多索引会使性能下降，多人更新操作会增加数据块锁的现象）

> 及 < 操作符（大于或小于操作符）

大于或小于操作符一般情况下是不用调整的，因为它有索引就会采用索引查找，但有的情况下可以对它进行优化，如一个表有 100 万记录，一个数值型字段 A，30 万记录的 A=0，30 万记录的 A=1，39 万记录的 A=2，1 万记录的 A=3。那么执行 A>2 与 A>=3 的效果就有很大的区别了，因为 A>2 时 ORACLE 会先找出为 2 的记录索引再进行比较，而 A>=3 时 ORACLE 则直接找到=3 的记录索引。

LIKE操作符

LIKE 操作符可以应用通配符查询，里面的通配符组合可能达到几乎是任意的查询，但是如果用得不

好则会产生性能上的问题，如 LIKE ‘%5400%’ 这种查询不会引用索引，而 LIKE ‘X5400%’则会引用范围索引。一个实际例子：用 YW_YHJBQK 表中营业编号后面的户标识号可来查询营业编号 YY_BH LIKE ‘%5400%’ 这个条件会产生全表扫描，如果改成 YY_BH LIKE ’X5400%’ OR YY_BH LIKE ’B5400%’ 则会利用 YY_BH 的索引进行两个范围的查询，性能肯定大大提高。

UNION操作符

UNION 在进行表链接后会筛选掉重复的记录，所以在表链接后会对所产生的结果集进行排序运算，删除重复的记录再返回结果。实际大部分应用中是不会产生重复的记录，最常见的是过程表与历史表UNION。如：

```
select * from gc_dfys
union
select * from ls_jg_dfys
```

这个 SQL 在运行时先取出两个表的结果，再用排序空间进行排序删除重复的记录，最后返回结果集，如果表数据量大的话可能会导致用磁盘进行排序。

推荐方案：采用 UNION ALL 操作符替代 UNION，因为 UNION ALL 操作只是简单的将两个结果合并后就返回。

```
select * from gc_dfys
union all
select * from ls_jg_dfys
```

SQL 书写的影响

同一功能同一性能不同写法SQL的影响

如一个 SQL 在 A 程序员写的为

Select * from zl_yhjbqk

B 程序员写的为

Select * from dlyx.zl_yhjbqk（带表所有者的前缀）

C 程序员写的为

Select * from DLYX.ZLYHJBQK（大写表名）

D 程序员写的为

Select * from DLYX.ZLYHJBQK（中间多了空格）

以上四个 SQL 在 ORACLE 分析整理之后产生的结果及执行的时间是一样的，但是从 ORACLE 共享内存 SGA 的原理，可以得出 ORACLE 对每个 SQL 都会对其进行一次分析，并且占用共享内存，如果将 SQL 的字符串及格式写得完全相同则 ORACLE 只会分析一次，共享内存也只会留下一次的分析结果，这不仅可以减少分析 SQL 的时间，而且可以减少共享内存重复的信息，ORACLE 也可以准确统计 SQL 的执行频率。

WHERE后面的条件顺序影响

WHERE 子句后面的条件顺序对大数据量表的查询会产生直接的影响，如

Select * from zl_yhjbqk where dy_dj = '1KV以下' and xh_bz=1

Select * from zl_yhjbqk where xh_bz=1 and dy_dj = '1KV以下'

以上两个SQL中dy_dj（电压等级）及xh_bz（销户标志）两个字段都没进行索引，所以执行的时候都是全表扫描，第一条SQL的**dy_dj = '1KV以下'**条件在记录集内比率为 99%，而**xh_bz=1**的比率只为 0.5%，在进行第一条SQL的时候 99%条记录都进行**dy_dj及xh_bz**的比较，而在进行第二条SQL的时候 0.5%条记录都进行**dy_dj及xh_bz**的比较，以此可以得出第二条SQL的CPU占用率明显比第一条低。

查询表顺序的影响

在 FROM 后面的表中的列表顺序会对 SQL 执行性能影响，在没有索引及 ORACLE 没有对表进行统计分析的情况下 ORACLE 会按表出现的顺序进行链接，由此因为表的顺序不对会产生十分耗服务器资源的数据交叉。（注：如果对表进行了统计分析，ORACLE 会自动先进小表的链接，再进行大表的链接）

SQL语句索引的利用

对操作符的优化（见上节）

对条件字段的一些优化

采用函数处理的字段不能利用索引，如：

substr(hbs_bh,1,4)=’5400’，优化处理：hbs_bh like ‘5400%’

trunc(sk_rq)=trunc(sysdate)， 优化处理：

sk_rq>=trunc(**sysdate**) **and** sk_rq<trunc(**sysdate+1**)

进行了显式或隐式的运算的字段不能进行索引，如：

ss_df+20>50，优化处理：ss_df>30

‘X’||hbs_bh>’X5400021452’，优化处理：hbs_bh>’5400021542’

sk_rq+5=sysdate，优化处理：sk_rq=sysdate-5

hbs_bh=5401002554，优化处理：hbs_bh=’ 5401002554’，**注：**此条件对hbs_bh 进行隐式的to_number转换，因为hbs_bh字段是字符型。

条件内包括了多个本表的字段运算时不能进行索引，如：

ys_df>cx_df，无法进行优化

qc_bh||kh_bh=’5400250000’，优化处理：qc_bh=’5400’ and kh_bh=’250000’

应用ORACLE的HINT（提示）处理

提示处理是在 ORACLE 产生的 SQL 分析执行路径不满意的情况下要用到的。它可以对 SQL 进行以下方面的提示

目标方面的提示：

COST（按成本优化）

RULE（按规则优化）

CHOOSE（缺省）（ORACLE 自动选择成本或规则进行优化）

ALL_ROWS（所有的行尽快返回）

FIRST_ROWS（第一行数据尽快返回）

执行方法的提示：

USE_NL（使用 NESTED LOOPS 方式联合）

USE_MERGE（使用 MERGE JOIN 方式联合）

USE_HASH（使用 HASH JOIN 方式联合）

索引提示：

INDEX（TABLE INDEX）（使用提示的表索引进行查询）

其它高级提示（如并行处理等等）

ORACLE 的提示功能是比较强的功能，也是比较复杂的应用，并且提示只是给 ORACLE 执行的一个建议，有时如果出于成本方面的考虑 ORACLE 也可能不会按提示进行。根据实践应用，一般不建议开发人员应用 ORACLE 提示，因为各个数据库及服务器性能情况不一样，很可能一个地方性能提升了，但另一个地方却下降了，ORACLE 在 SQL 执行分析方面已经比较成熟，如果分析执行的路径不对首先应在数据库结构（主要是索引）、服务器当前性能（共享内存、磁盘文件碎片）、数据库对象（表、索引）统计信息是否正确这几方面分析。