**1.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’)

**2. 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，对子查询中的表执行一个全表遍历，因此是非常低效的。

为了避免使用NOT IN，可以把它改写成外连接（Outer Joins）或者NOT EXISTS。

低效：

SELECT …

FROM EMP

WHERE DEPT\_NO NOT IN （SELECT DEPT\_NO

FROM DEPT

WHERE DEPT\_CAT=’A’）

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’）

**3.用索引提高效率**

**（1）特点**

优点： 提高效率 主键的唯一性验证

代价： 需要空间存储 定期维护

重构索引：

ALTER INDEX <INDEXNAME> REBUILD <TABLESPACENAME>

**（2）Oracle对索引有两种访问模式**

* 索引唯一扫描 (Index Unique Scan)
* 索引范围扫描 (Index Range Scan)

**（3）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。

**（4）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。
* 如果不同表中两个相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。
* 如果相同表中两个相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。

**（5）等式比较优先于范围比较**

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

即使是唯一性索引，如果做范围比较，其优先级也低于非唯一性索引的等式比较。

**（6）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。

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

**（7）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) 。

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\*/

**（8）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。

低效：

SELECT …

FROM DEPT

WHERE SAL \* **12** > **25000**;

高效：

SELECT …

FROM DEPT

WHERE SAL > **25000**/**12**;

**（9）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，而其他是非唯一性索引。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。

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

**（10）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，它就会停止使用索引转而执行全表扫描。

低效: (这里，不使用索引)

SELECT …

FROM DEPT

WHERE NOT DEPT\_CODE = **0**

高效：(这里，使用了索引)

SELECT …

FROM DEPT

WHERE DEPT\_CODE > **0**

**4.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等于3的记录并且向前扫描到第一个DEPT大于3的记录.

**5.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。

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**;

**6.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。

[复制代码](javascript:void(0);)

SQL> create index multindex on multiindexusage(inda,indb);

Index created.

SQL> select \* from multiindexusage where indb = **1**;

Execution Plan

----------------------------------------------------------

**0** SELECT STATEMENT Optimizer=CHOOSE

**1** **0** TABLE ACCESS (FULL) OF 'MULTIINDEXUSAGE‘

[复制代码](javascript:void(0);)

**7.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。

[复制代码](javascript:void(0);)

/\*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**

[复制代码](javascript:void(0);)

因为内部发生的类型转换，这个索引将不会被用到。

几点注意：

* 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。
* 如果在索引列上面进行了隐式类型转换，在查询的时候将不会用到索引。
* 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。
* 为了避免ORACLE对SQL进行隐式的类型转换，最好把类型转换用显式表现出来。

**8.几种不能使用索引的WHERE子句**

（1）下面的例子中，‘!=’ 将不使用索引 ，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**；

（2）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’；

（3）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** ；

（4）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, ’%’)