|  |  |
| --- | --- |
| From: | 京奇電腦 |
| To: |  |
| Date: | 2016-06-07 |
| Subject: | Oracle SQL command Start with connect by prior 階層式查詢用法 |

# 目錄

內容

[目錄 1](#_Toc453083257)

[Oracle Sample 1](#_Toc453083258)

[Hierarchical Queries 2](#_Toc453083259)

[Hierarchical Query Examples 4](#_Toc453083260)

[Hierarchical Query Pseudocolumns 8](#_Toc453083261)

[CONNECT\_BY\_ISCYCLE Pseudocolumn 8](#_Toc453083262)

[CONNECT\_BY\_ISLEAF Pseudocolumn 9](#_Toc453083263)

[LEVEL Pseudocolumn 10](#_Toc453083264)

[Basic Usage 10](#_Toc453083265)

[Real case sample 21](#_Toc453083266)

[case 1 22](#_Toc453083267)

[case 2 23](#_Toc453083268)

[case 3 26](#_Toc453083269)

# Oracle Sample

ref: <https://docs.oracle.com/cd/B19306_01/server.102/b14200/queries003.htm>

## Hierarchical Queries

If a table contains hierarchical data, then you can select rows in a hierarchical order using the hierarchical query clause:

***hierarchical\_query\_clause*::=**

Description of hierarchical_query_clause.gif follows  
[Description of the illustration hierarchical\_query\_clause.gif](https://docs.oracle.com/cd/B19306_01/server.102/b14200/img_text/hierarchical_query_clause.htm)

**START** **WITH** specifies the root row(s) of the hierarchy.

**CONNECT** **BY** specifies the relationship between parent rows and child rows of the hierarchy.

* The **NOCYCLE** parameter instructs Oracle Database to return rows from a query even if a **CONNECT** **BY** **LOOP** exists in the data. Use this parameter along with the **CONNECT\_BY\_ISCYCLE** pseudocolumn to see which rows contain the loop. Please refer to [CONNECT\_BY\_ISCYCLE Pseudocolumn](https://docs.oracle.com/cd/B19306_01/server.102/b14200/pseudocolumns001.htm#i1009434) for more information.
* In a hierarchical query, one expression in ***condition*** must be qualified with the **PRIOR** operator to refer to the parent row. For example,
* ... PRIOR expr = expr
* or
* ... expr = PRIOR expr

If the **CONNECT** **BY** ***condition*** is compound, then only one condition requires the **PRIOR** operator, although you can have multiple **PRIOR** conditions. For example:

CONNECT BY last\_name != 'King' AND PRIOR employee\_id = manager\_id ...

CONNECT BY PRIOR employee\_id = manager\_id and

PRIOR account\_mgr\_id = customer\_id ...

**PRIOR** is a unary operator and has the same precedence as the unary + and - arithmetic operators. It evaluates the immediately following expression for the parent row of the current row in a hierarchical query.

**PRIOR** is most commonly used when comparing column values with the equality operator. (The **PRIOR** keyword can be on either side of the operator.) **PRIOR** causes Oracle to use the value of the parent row in the column. Operators other than the equal sign (=) are theoretically possible in **CONNECT** **BY** clauses. However, the conditions created by these other operators can result in an infinite loop through the possible combinations. In this case Oracle detects the loop at run time and returns an error.

Both the **CONNECT** **BY** condition and the **PRIOR** expression can take the form of an uncorrelated subquery. However, the **PRIOR** expression cannot refer to a sequence. That is, **CURRVAL** and **NEXTVAL** are not valid **PRIOR** expressions.

You can further refine a hierarchical query by using the **CONNECT\_BY\_ROOT** operator to qualify a column in the select list. This operator extends the functionality of the **CONNECT** **BY** [**PRIOR**] condition of hierarchical queries by returning not only the immediate parent row but all ancestor rows in the hierarchy.

**See Also:**

[CONNECT\_BY\_ROOT](https://docs.oracle.com/cd/B19306_01/server.102/b14200/operators004.htm#i1035022) for more information about this operator and ["Hierarchical Query Examples"](https://docs.oracle.com/cd/B19306_01/server.102/b14200/queries003.htm#i2060615)

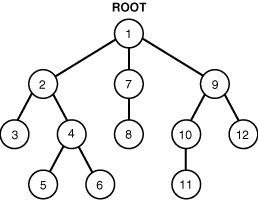
Oracle processes hierarchical queries as follows:

* A join, if present, is evaluated first, whether the join is specified in the **FROM** clause or with **WHERE** clause predicates.
* The **CONNECT** **BY** condition is evaluated.
* Any remaining **WHERE** clause predicates are evaluated.

Oracle then uses the information from these evaluations to form the hierarchy using the following steps:

1. Oracle selects the root row(s) of the hierarchy--those rows that satisfy the **START** **WITH** condition.
2. Oracle selects the child rows of each root row. Each child row must satisfy the condition of the **CONNECT** **BY** condition with respect to one of the root rows.
3. Oracle selects successive generations of child rows. Oracle first selects the children of the rows returned in step [2](https://docs.oracle.com/cd/B19306_01/server.102/b14200/queries003.htm#i2070828), and then the children of those children, and so on. Oracle always selects children by evaluating the **CONNECT** **BY** condition with respect to a current parent row.
4. If the query contains a **WHERE** clause without a join, then Oracle eliminates all rows from the hierarchy that do not satisfy the condition of the **WHERE** clause. Oracle evaluates this condition for each row individually, rather than removing all the children of a row that does not satisfy the condition.
5. Oracle returns the rows in the order shown in [Figure 9-1](https://docs.oracle.com/cd/B19306_01/server.102/b14200/queries003.htm#i2066595). In the diagram, children appear below their parents. For an explanation of hierarchical trees, see [Figure 3-1, "Hierarchical Tree"](https://docs.oracle.com/cd/B19306_01/server.102/b14200/pseudocolumns001.htm#i1009270).

***Figure 9-1 Hierarchical Queries***

  
[Description of "Figure 9-1 Hierarchical Queries"](https://docs.oracle.com/cd/B19306_01/server.102/b14200/img_text/sqlrf002.htm)

To find the children of a parent row, Oracle evaluates the **PRIOR** expression of the **CONNECT** **BY** condition for the parent row and the other expression for each row in the table. Rows for which the condition is true are the children of the parent. The **CONNECT** **BY** condition can contain other conditions to further filter the rows selected by the query. The **CONNECT** **BY** condition cannot contain a subquery.

If the **CONNECT** **BY** condition results in a loop in the hierarchy, then Oracle returns an error. A loop occurs if one row is both the parent (or grandparent or direct ancestor) and a child (or a grandchild or a direct descendent) of another row.

**Note:**

In a hierarchical query, do not specify either **ORDER** **BY** or **GROUP** **BY**, as they will destroy the hierarchical order of the **CONNECT** **BY** results. If you want to order rows of siblings of the same parent, then use the **ORDER** **SIBLINGS** **BY** clause. See [*order\_by\_clause*](https://docs.oracle.com/cd/B19306_01/server.102/b14200/statements_10002.htm#i2171079) .

## Hierarchical Query Examples

Hierarchical Query Examples

**CONNECT BY Example**The following hierarchical query uses the **CONNECT** **BY** clause to define the relationship between employees and managers:

SELECT employee\_id, last\_name, manager\_id

FROM employees

CONNECT BY PRIOR employee\_id = manager\_id;

EMPLOYEE\_ID LAST\_NAME MANAGER\_ID

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

101 Kochhar 100

108 Greenberg 101

109 Faviet 108

110 Chen 108

111 Sciarra 108

112 Urman 108

113 Popp 108

200 Whalen 101

...

**LEVEL Example**The next example is similar to the preceding example, but uses the **LEVEL** pseudocolumn to show parent and child rows:

SELECT employee\_id, last\_name, manager\_id, LEVEL

FROM employees

CONNECT BY PRIOR employee\_id = manager\_id;

EMPLOYEE\_ID LAST\_NAME MANAGER\_ID LEVEL

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

101 Kochhar 100 1

108 Greenberg 101 2

109 Faviet 108 3

110 Chen 108 3

111 Sciarra 108 3

112 Urman 108 3

113 Popp 108 3

...

**START WITH Examples**The next example adds a **START** **WITH** clause to specify a root row for the hierarchy and an **ORDER** **BY** clause using the **SIBLINGS** keyword to preserve ordering within the hierarchy:

SELECT last\_name, employee\_id, manager\_id, LEVEL

FROM employees

START WITH employee\_id = 100

CONNECT BY PRIOR employee\_id = manager\_id

ORDER SIBLINGS BY last\_name;

LAST\_NAME EMPLOYEE\_ID MANAGER\_ID LEVEL

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

King 100 1

Cambrault 148 100 2

Bates 172 148 3

Bloom 169 148 3

Fox 170 148 3

Kumar 173 148 3

Ozer 168 148 3

Smith 171 148 3

De Haan 102 100 2

Hunold 103 102 3

Austin 105 103 4

Ernst 104 103 4

Lorentz 107 103 4

Pataballa 106 103 4

Errazuriz 147 100 2

Ande 166 147 3

Banda 167 147 3

...

In the **hr.employees** table, the employee Steven King is the head of the company and has no manager. Among his employees is John Russell, who is the manager of department 80. If we update the **employees** table to set Russell as King's manager, we will create a loop in the data:

UPDATE employees SET manager\_id = 145

WHERE employee\_id = 100;

SELECT last\_name "Employee",

LEVEL, SYS\_CONNECT\_BY\_PATH(last\_name, '/') "Path"

FROM employees

WHERE level <= 3 AND department\_id = 80

START WITH last\_name = 'King'

CONNECT BY PRIOR employee\_id = manager\_id AND LEVEL <= 4;

2 3 4 5 6 7 ERROR:

ORA-01436: CONNECT BY loop in user data

The **NOCYCLE** parameter in the **CONNECT** **BY** condition causes Oracle to return the rows in spite of the loop. The **CONNECT\_BY\_ISCYCLE** pseudocolumn shows you which rows contain the cycle:

SELECT last\_name "Employee", CONNECT\_BY\_ISCYCLE "Cycle",

LEVEL, SYS\_CONNECT\_BY\_PATH(last\_name, '/') "Path"

FROM employees

WHERE level <= 3 AND department\_id = 80

START WITH last\_name = 'King'

CONNECT BY NOCYCLE PRIOR employee\_id = manager\_id AND LEVEL <= 4;

Employee Cycle LEVEL Path

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

Russell 1 2 /King/Russell

Tucker 0 3 /King/Russell/Tucker

Bernstein 0 3 /King/Russell/Bernstein

Hall 0 3 /King/Russell/Hall

Olsen 0 3 /King/Russell/Olsen

Cambrault 0 3 /King/Russell/Cambrault

Tuvault 0 3 /King/Russell/Tuvault

Partners 0 2 /King/Partners

King 0 3 /King/Partners/King

Sully 0 3 /King/Partners/Sully

McEwen 0 3 /King/Partners/McEwen

...

**CONNECT\_BY\_ROOT Examples**The following example returns the last name of each employee in department 110, each manager above that employee in the hierarchy, the number of levels between manager and employee, and the path between the two:

SELECT last\_name "Employee", CONNECT\_BY\_ROOT last\_name "Manager",

LEVEL-1 "Pathlen", SYS\_CONNECT\_BY\_PATH(last\_name, '/') "Path"

FROM employees

WHERE LEVEL > 1 and department\_id = 110

CONNECT BY PRIOR employee\_id = manager\_id;

Employee Manager Pathlen Path

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

Higgins Kochhar 1 /Kochhar/Higgins

Gietz Kochhar 2 /Kochhar/Higgins/Gietz

Gietz Higgins 1 /Higgins/Gietz

Higgins King 2 /King/Kochhar/Higgins

Gietz King 3 /King/Kochhar/Higgins/Gietz

The following example uses a **GROUP** **BY** clause to return the total salary of each employee in department 110 and all employees below that employee in the hierarchy:

SELECT name, SUM(salary) "Total\_Salary" FROM (

SELECT CONNECT\_BY\_ROOT last\_name as name, Salary

FROM employees

WHERE department\_id = 110

CONNECT BY PRIOR employee\_id = manager\_id)

GROUP BY name;

NAME Total\_Salary

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

Gietz 8300

Higgins 20300

King 20300

Kochhar 20300

Hierarchical Query Pseudocolumns

The hierarchical query pseudocolumns are valid only in hierarchical queries. The hierarchical query pseudocolumns are:

* [CONNECT\_BY\_ISCYCLE Pseudocolumn](https://docs.oracle.com/cd/B19306_01/server.102/b14200/pseudocolumns001.htm#i1009434)
* [CONNECT\_BY\_ISLEAF Pseudocolumn](https://docs.oracle.com/cd/B19306_01/server.102/b14200/pseudocolumns001.htm#i1007332)
* [LEVEL Pseudocolumn](https://docs.oracle.com/cd/B19306_01/server.102/b14200/pseudocolumns001.htm#i1009261)

CONNECT\_BY\_ISCYCLE Pseudocolumn

The **CONNECT\_BY\_ISCYCLE** pseudocolumn returns 1 if the current row has a child which is also its ancestor. Otherwise it returns 0.

You can specify **CONNECT\_BY\_ISCYCLE** only if you have specified the **NOCYCLE** parameter of the **CONNECT** **BY** clause. **NOCYCLE** enables Oracle to return the results of a query that would otherwise fail because of a **CONNECT** **BY** loop in the data.

**See Also:**

["Hierarchical Queries"](https://docs.oracle.com/cd/B19306_01/server.102/b14200/queries003.htm#i2053935) for more information about the **NOCYCLE** parameter and ["Hierarchical Query Examples"](https://docs.oracle.com/cd/B19306_01/server.102/b14200/queries003.htm#i2060615) for an example that uses the **CONNECT\_BY\_ISCYCLE** pseudocolumn

CONNECT\_BY\_ISLEAF Pseudocolumn

The **CONNECT\_BY\_ISLEAF** pseudocolumn returns 1 if the current row is a leaf of the tree defined by the **CONNECT** **BY** condition. Otherwise it returns 0. This information indicates whether a given row can be further expanded to show more of the hierarchy.

**CONNECT\_BY\_ISLEAF Example**The following example shows the first three levels of the **hr.employees** table, indicating for each row whether it is a leaf row (indicated by 1 in the **IsLeaf** column) or whether it has child rows (indicated by 0 in the **IsLeaf** column):

SELECT last\_name "Employee", CONNECT\_BY\_ISLEAF "IsLeaf",

LEVEL, SYS\_CONNECT\_BY\_PATH(last\_name, '/') "Path"

FROM employees

WHERE LEVEL <= 3 AND department\_id = 80

START WITH employee\_id = 100

CONNECT BY PRIOR employee\_id = manager\_id AND LEVEL <= 4;

Employee IsLeaf LEVEL Path

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

Russell 0 2 /King/Russell

Tucker 1 3 /King/Russell/Tucker

Bernstein 1 3 /King/Russell/Bernstein

Hall 1 3 /King/Russell/Hall

Olsen 1 3 /King/Russell/Olsen

Cambrault 1 3 /King/Russell/Cambrault

Tuvault 1 3 /King/Russell/Tuvault

Partners 0 2 /King/Partners

King 1 3 /King/Partners/King

Sully 1 3 /King/Partners/Sully

McEwen 1 3 /King/Partners/McEwen

Smith 1 3 /King/Partners/Smith

Doran 1 3 /King/Partners/Doran

Sewall 1 3 /King/Partners/Sewall

Errazuriz 0 2 /King/Errazuriz

Vishney 1 3 /King/Errazuriz/Vishney

...

34 rows selected.

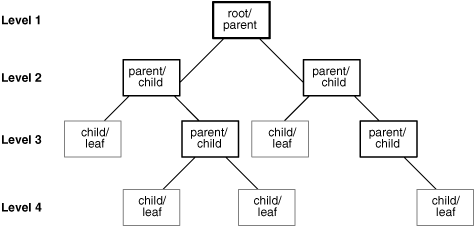
**See Also:**

["Hierarchical Queries"](https://docs.oracle.com/cd/B19306_01/server.102/b14200/queries003.htm#i2053935) and [SYS\_CONNECT\_BY\_PATH](https://docs.oracle.com/cd/B19306_01/server.102/b14200/functions164.htm#i1038266)

LEVEL Pseudocolumn

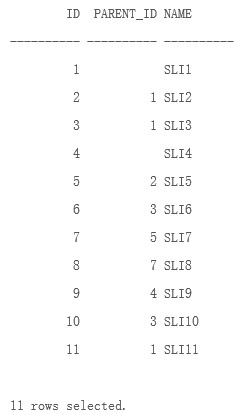
For each row returned by a hierarchical query, the **LEVEL** pseudocolumn returns 1 for a root row, 2 for a child of a root, and so on. A **root row** is the highest row within an inverted tree. A **child row** is any nonroot row. A **parent row** is any row that has children. A **leaf row** is any row without children. [Figure 3-1](https://docs.oracle.com/cd/B19306_01/server.102/b14200/pseudocolumns001.htm#i1009270) shows the nodes of an inverted tree with their **LEVEL** values.

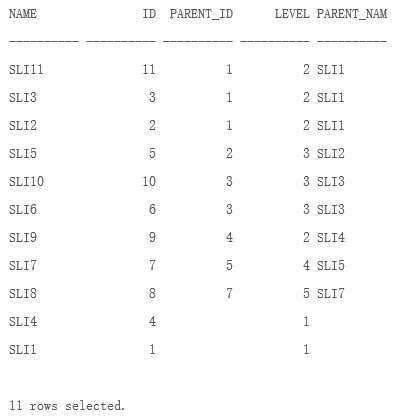
***Figure 3-1 Hierarchical Tree***

  
[Description of "Figure 3-1 Hierarchical Tree "](https://docs.oracle.com/cd/B19306_01/server.102/b14200/img_text/sqlrf001.htm)

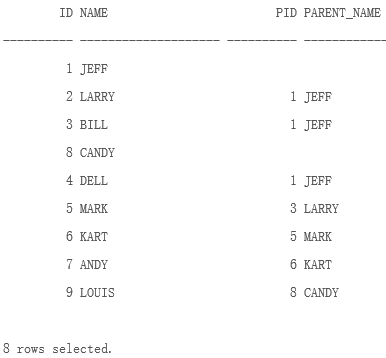
To define a hierarchical relationship in a query, you must use the **START** **WITH** and **CONNECT** **BY** clauses.

# Basic Usage

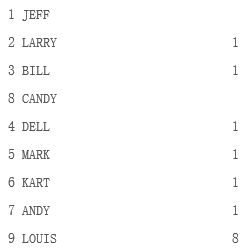
最基本的connect by的用法:  
  
需求1: 我需要下钻所有的树得到level和parent\_name  
create table test\_lvl1 (id number, parent\_id number, name varchar2(10));  
insert into test\_lvl1 values (1,null,'SLI1');  
insert into test\_lvl1 values (2,1,'SLI2');  
insert into test\_lvl1 values (3,1,'SLI3');  
insert into test\_lvl1 values (4,null,'SLI4');  
insert into test\_lvl1 values (5,2,'SLI5');  
insert into test\_lvl1 values (6,3,'SLI6');  
insert into test\_lvl1 values (7,5,'SLI7');  
insert into test\_lvl1 values (8,7,'SLI8');  
insert into test\_lvl1 values (9,4,'SLI9');  
insert into test\_lvl1 values (10,3,'SLI10');  
insert into test\_lvl1 values (11,1,'SLI11');  
  
kl@k01> select \* from test\_lvl1;  
  
select a.\*, b.name parent\_name  
from  
(select name, ID, PARENT\_ID, LEVEL  
from test\_lvl1  
start with parent\_id is null  
connect by prior id=PARENT\_ID) a,  
test\_lvl1 b  
where a.parent\_id=b.id(+)



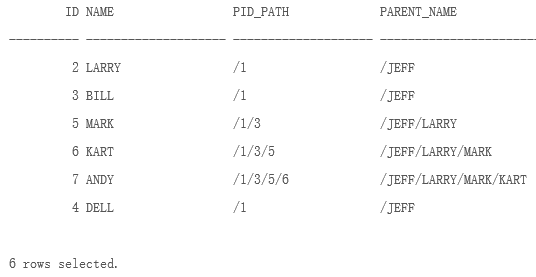
\* 解释一下:  
\* 1. Start with表示从那一层开始的，后面跟表达式,如: pid=0,  
寻找继承关系时，指定的顶点，如果需要对整个表进行整理，比较常用  
的作法是:  PID is null. 例如这种情况顶点的Parent\_ID显然是NULL,  
所以从PID is null开始无疑是最完整的。验证如下:  
\* 2. prior 表示返回所以符合这种条件(如id=pid)的connect by操作结果.  
即以循环的显示所有的记录和继承关系。可以试一下如果把prior去掉，那么  
只有两条记录,验证如下:  
  
select name, ID, PARENT\_ID, LEVEL  
from test\_lvl1  
start with parent\_id is null  
connect by prior id=PARENT\_ID;  
  
--- 如果用Chile ID或者其他列作为开始列，会丢掉分支, 如下查询，可以自己验证  
select name, ID, PARENT\_ID, LEVEL  
from test\_lvl1  
start with id =1  
connect by prior id=PARENT\_ID(+);  
  
select name, ID, PARENT\_ID, LEVEL  
from test\_lvl1  
start with NAME='SLI1'  
connect by prior id(+)=PARENT\_ID;  
  
\* 另外一个很有趣的现象，对上述start with条件不同，而且结果记录数也不同的查询查看执行计划  
竟是完全相同的。测试版本是Oracle 10.2.0.1  
测试过程如下，节约篇幅，结果不述:  
1. Set autotrace on;  
2. 执行上面的query,查看返回值和执行计划;  
  
  
需求2，我需要每个从根节点到当前孩子节点的路径:  
create table test\_parent1 (id number, name varchar2(20), pid number, parent\_name varchar2(20));  
insert into test\_parent1 values (1,'JEFF',null,null);  
insert into test\_parent1 values (2,'LARRY',1,'JEFF');  
insert into test\_parent1 values (3,'BILL',1,'JEFF');  
insert into test\_parent1 values (4,'DELL',1,'JEFF');  
insert into test\_parent1 values (5,'MARK',3,'LARRY');  
insert into test\_parent1 values (6,'KART',5,'MARK');  
insert into test\_parent1 values (7,'ANDY',6,'KART');  
insert into test\_parent1 values (8,'CANDY',null,null);  
insert into test\_parent1 values (9,'LOUIS',8,'CANDY');  
commit;  
  
--- 得到如下表:  
kl@k01> select \* from test\_parent1;



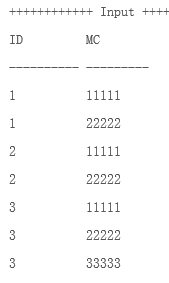
可以看出来一个简单的层次关系，如果我希望得到每个人的直接主管，应该就足够了，  
但是要得到每个人的顶级BOSS，即变成如下:

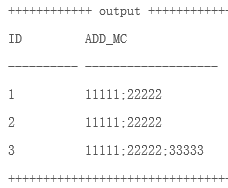


我们还需要做如下处理,使用SYS\_CONNECT\_BY\_PATH,这个函数只使用于9i和9i以上的版本，  
她可以帮我返回一个字符串，这个字符串是一个from-root-to-node的字串。如下所示:  
col PID\_PATH for a20  
col parent\_name for a30  
select ID, NAME, SYS\_CONNECT\_BY\_PATH(pid,'/') as pid\_path, SYS\_CONNECT\_BY\_PATH(parent\_name,'/') as parent\_name  
from test\_parent1  
start with pid=1 connect by prior id=pid;

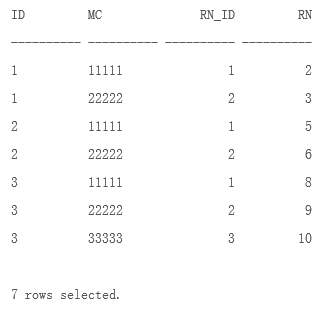


需求3, 字符串分组方法(这里参考了zhouwf0726在ITPUB里面的研究结果 http://www.itpub.net/thread-614563-1-1.html):

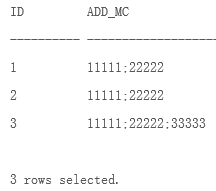




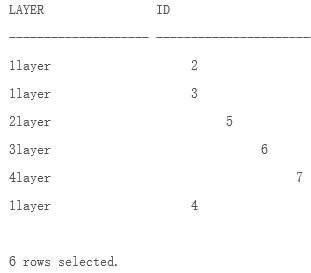
实现方法和试验步骤如下:  
drop table test purge;  
create table test(id varchar2(10),mc varchar2(50));  
insert into test values('1','11111');  
insert into test values('1','22222');  
insert into test values('2','11111');  
insert into test values('2','22222');  
insert into test values('3','11111');  
insert into test values('3','22222');  
insert into test values('3','33333');  
  
commit;  
\* 先看看怎么利用connect by, 如果要利用connect by，必须考虑使用sys\_connect\_by\_path(上面有详细介绍)  
关键是如何选择connect by的条件,为什么要制定rn\_id，我们需要对ID进行分组，从而把同组的MC组合起来，  
而对于RN,  
col MC for a10  
select id, mc, row\_number() over(partition by id order by id) rn\_id,  
               row\_number() over(order by id)+id rn  
from test;



col add\_mc for a50  
select id,ltrim(max(sys\_connect\_by\_path(mc,';')),';') add\_mc from (  
select id, mc, row\_number() over(partition by id order by id) rn\_id,  
row\_number() over(order by id)+id rn from test)  
start with rn\_id=1 connect by rn-1=prior rn  
group by id  
order by id  
/



## 解释:  首先要熟悉分析函数 row\_number() over(partition by id order by id)  
表示以id进行分组，显示每条记录中id的数量，order by id是对id进行排序。  
rn是顺序的，如果直接connect by一组顺序的数，将一直连续下去，所以因为一直满足rn - 1= rn,  
所以要使rn为 row\_number() over(order by id)+id，因为每组的id都不一样，这样才能避免连续的数字出现。  
  
  
  
  
  
需求4，我需要在输出的时候就对结果进行格式化:  
  
lpad和level配合决定错行缩进显示:  
  
select level||'layer' layer,lpad(' ',level\*5)||id id  
from test\_parent1  
start with pid=1 connect by prior id=pid;

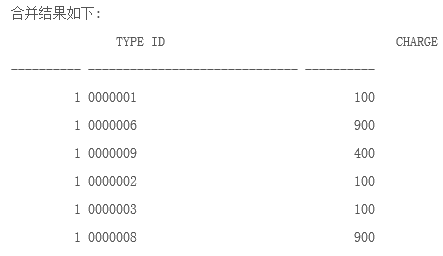


需求4, 如果需要构建一种循环处理机制，比如把一个字符的每个字母输出，不用通过  
PLSQL来实现，直接用connect by即可，当着这里已经有点跑题了，权当兴趣吧:  
  
select substr('JEFFREY',rownum, 1) Letter from dual

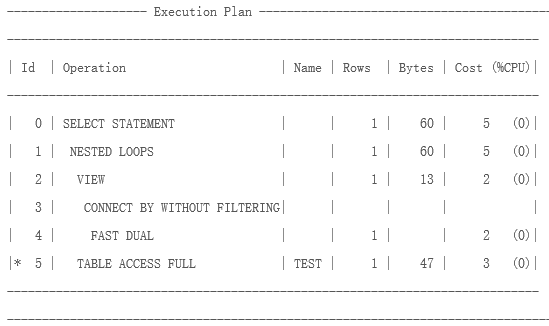


介绍这个例子也是为了引申出两个简单常用的技巧:  
1) 构建大数据量的测试数据:  
   select rownum a from dual connect by rownum<=10000;   --- 10000条顺序的纪录就轻松产生了。  
   或者:  select level a from dual connect by level<=10000;  
2) 结合case ... when优化query, 通过这种方法在列转行的时候，减少全表扫描的次数:  
比如，我要把ID1,ID2,ID3合并起来，他们对应的CHARGE不变:





++++++++++++  test table ++++++++++++++++++  
drop table test;  
create table test  
( type number,  
  id1 varchar2(10) default 0 not null,  
  id2 varchar2(10) default 0 not null,  
  id3 varchar2(10) default 0 not null,  
  charge number  
);  
insert into test values (1,'0000001','0000002','0000003',100);  
insert into test values (1,'0000006',0,'0000008',900);  
insert into test values (1,'0000009',0,0,400);  
  
commit;  
+++++++++++++++++++++++++++++++++++++++++++  
常规方法可能是:  
select type, id1 id, charge from test where id1<>0  
union all  
select type, id2 id, charge from test where id2<>0  
union all  
select type, id3 id, charge from test where id3<>0  
;  
  
但如果test纪录size很大，三次FTS肯定cost很高，所以我们必须考虑通过case .. when的方法减少FTS的次数:  
select type,  
       case a when 1 then id1  
              when 2 then id2  
              when 3 then id3  
       end  
       id,charge  
from test, (select level a from dual connect by level<=3) lvl  
where case a when 1 then id1  
             when 2 then id2  
             when 3 then id3  
             end <>0;  
从下面的执行计划看，只有一次FTS，而不是之前的三次



# Real case sample

## case 1

SELECT AA.AGENT\_ID,

AA.AGENT\_CODE, --車商代碼

CPAB.BRANCH\_CODE, --營業所代碼

CPAB.OCCUPY\_SP\_CODE, --據點代碼PO.REGISTRATION\_NAME, --全名

PO.NAME, -- 簡稱

AA.PARTY\_ID,

PTYR1.PTY\_ID PTY\_ID1, -- ROOT ID

PTYR2.PTY\_ID PTY\_ID2,

T.\*

FROM

(SELECT connect\_by\_root TO\_PTYR\_ID,

R.FROM\_PTYR\_ID,

R.RELATION\_TYPE

FROM T\_PTYR\_RELA R

START WITH R.to\_Ptyr\_Id IN

(SELECT RR.PTYR\_ID

FROM T\_AGT\_AGENT AG,

T\_PTYR RR

WHERE AG.PARTY\_ID = RR.PTY\_ID

)

CONNECT BY PRIOR R.FROM\_PTYR\_ID = R.TO\_PTYR\_ID

) T,

T\_CHT\_PTY\_AGENCY\_BRANCH CPAB,

T\_PTYR PTYR1, -- 關聯至T\_CHT\_PTY\_AGENCY\_BRANCH的PTYR

T\_AGT\_AGENT AA,

T\_PTY\_ORG PO,

T\_PTYR PTYR2 -- 關聯至CONNECT\_BY\_ROOTTO\_PTYR\_ID的PTYR

WHERE CPAB.PTYR\_ID = T.FROM\_PTYR\_ID

--AND T.RELATION\_TYPE IN (109, 110) -- 109=Company-Agent, 110=Company-Agent-Branch, 111=同業891101

AND AA.PARTY\_ID = PTYR1.PTY\_ID

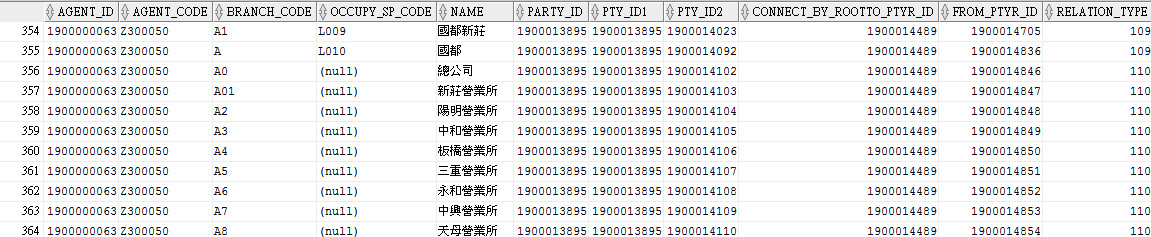
AND PO.PTY\_ID = PTYR2.PTY\_ID

AND PTYR2.PTYR\_ID = CPAB.PTYR\_ID

AND PTYR1.PTYR\_ID = CONNECT\_BY\_ROOTTO\_PTYR\_ID

-- 2016-06-01, Honda, Create.

;



## case 2

Reference: https://dotblogs.com.tw/jeff-yeh/2009/05/20/8489

在Oracle裡有一個不錯的階層式查詢功能,可以用很簡短的Script來達成目標,不用一堆子查詢或是temp Table在那join來join去囉.

　　在Oracle裡有一個不錯的階層式查詢功能,可以用很簡短的Script來達成目標,階層的意思就有點像是(總經理->副總->協理->經理->副理…..)的這種層級關係,這樣看可能還是很難懂階層或查詢的用法關係,以下是Demo的資料表:

TableName : UserMgrInfo

|  |  |  |  |
| --- | --- | --- | --- |
| UserNo  [人員編號] | UserName  [人員名稱] | UserMgrNo  [主管代號] | UserActive  [人員生失效] |
| Z001 | Jeff | X059 | 1 |
| X059 | Kevin | X043 | 1 |
| X040 | Judy | B001 | 1 |
| X043 | David | X040 | 0 |
| B001 | Andy | A001 | 1 |
| A001 | Frank |  | 1 |
| C001 | Cindy | B001 | 1 |

　　上面這個表格的關係很簡單,每筆資料都有一個主管代號,這個主管代號就是他的上一階主管,當UserMgrNo為空時,代表為最上階主管,階層結束,之中也有人員生失效的控制,現在要做的是如何下一個查詢,找出該員的主管清單?而且只要找出生效的人員。

　　在Oracle裡有個Start with connect by prior可以使用,以下的語法就是找出該員的生效主管清單:

Select UserNo,UserName from

(

        Select UserNo,UserName,UserActive from UserMgrInfo

        Start with UserNo=’Z001’

        Connect by UserNo=prior UserMgrNo

)

Where UserActive=’1’

這個語法所下的條件,就是找出編號Z001這個人的主管清單,這個語法所輸出的結果如下 :

|  |  |
| --- | --- |
| UserNo  [人員編號] | UserName  [人員名稱] |
| Z001 | Jeff |
| X059 | Kevin |
| X040 | Judy |
| B001 | Andy |
| A001 | Frank |

　　而編號X043的David並不會輸出,因為他是失效人員,為什麼要用子查詢的方式去下UserActive=’1’,而不直接下在裡面就好? 因為如果直接下在裡面,這個查詢結果就只會到X059就停下來了,這個情況就很像rownum :

Select \* from UserMgrInfo where rownum=1

這時會帶回第一筆資料

Select \* from UserMgrInfo where rownum=2

這時並不是帶回二筆資料,而是一筆也沒有,如果要帶回第二筆,就要用子查詢的方式

Select \* from

(

Select rownum,\* from UserMgrInfo

)

Where rownum=2

回正題,如果此時要查Cindy編號C001的主管清單,同樣的語法所帶回的結果如下 :

|  |  |
| --- | --- |
| UserNo  [人員編號] | UserName  [人員名稱] |
| C001 | Cindy |
| B001 | Andy |
| A001 | Frank |

所以用這個Start wit connect by prior就可以很快的達成階層式查詢囉~

## case 3

Reference: http://fecbob.pixnet.net/blog/post/43278427-oracle-start-with-connect-by-%E7%94%A8%E6%B3%95

oracle 提供了start with connect by 語法結構可以實現遞迴查詢。

1. 一個簡單舉例:

SQL> select \* from test;

BILL\_MONTH DAY\_NUMBER MSISDN

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

200803 1 13800

200803 3 13800

200803 2 13800

200803 2 13801

200803 4 13804

200803 5 13804

200803 7 13804

200803 8 13804

200803 6 13802

200803 6 13801

200803 7 13801

200803 8 13801

12 rows selected

SQL>

SQL> select \* from test

2 start with day\_number=1

3 connect by prior day\_number=day\_number-1 and prior msisdn= msisdn

4 ;

BILL\_MONTH DAY\_NUMBER MSISDN

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

200803 1 13800

200803 2 13800

200803 3 13800

SQL>

上面的語句查找出了從1開始，並且day\_number 逐漸+1 遞增的，並且 msisdn 相同的哪些個資料.

2. start with connect by 語法結構

如上面說看到的 例子， 其語法結構為 start with condition connect by condition （含 prior 關鍵字)

start with conditon 給出的seed 資料的範圍, connect by 後面給出了遞迴查詢的條件,prior 關鍵字表示父資料，prior 條件表示子資料需要滿足父資料的什麼條件。

在下面的這個start with connect by 結構中，就表示 查找出了從1開始，並且day\_number 逐漸+1 遞增的，並且 msisdn 相同的那些個資料.

start with day\_number=1

connect by prior day\_number=day\_number-1 and prior msisdn= msisdn

3. 執行計畫

對於這個特殊的語法結構，我們來看看它的執行計畫。

通過下面的執行計畫，我們可以看出，對於簡單的訪問一個物件的遞迴查詢，實際上oracle 要三次訪問要查詢的物件。因此，這一個告訴我們，在使用遞迴查詢時，一定要謹慎，因為即使原表資料不多，但是三倍的訪問喜愛來，代價也會很大。

SQL> explain plan for

2

2 select \* from test

3 --where bill\_month='200803'

4 start with day\_number=1

5 connect by prior day\_number=day\_number-1 and prior msisdn= msisdn

6 ;

Explained

SQL> select \* from table( dbms\_xplan.display);

PLAN\_TABLE\_OUTPUT

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

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

| Id | Operation | Name | Rows | Bytes | Cost |

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

| 0 | SELECT STATEMENT | | | | |

|\* 1 | CONNECT BY WITH FILTERING| | | | |

|\* 2 | FILTER | | | | |

| 3 | TABLE ACCESS FULL | TEST | | | |

| 4 | NESTED LOOPS | | | | |

| 5 | BUFFER SORT | | | | |

| 6 | CONNECT BY PUMP | | | | |

|\* 7 | TABLE ACCESS FULL | TEST | | | |

| 8 | TABLE ACCESS FULL | TEST | | | |

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

Predicate Information (identified by operation id):

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

1 - filter("TEST"."DAY\_NUMBER"=1)

2 - filter("TEST"."DAY\_NUMBER"=1)

PLAN\_TABLE\_OUTPUT

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

7 - filter("TEST"."MSISDN"=Null AND "TEST"."DAY\_NUMBER"-1=Null)

Note: rule based optimization

23 rows selected

SQL>

另外，發現了在含有其他條件的遞迴中，是先處理所有的遞迴查詢，最後才用加入的條件過濾.

請看下面的例子。

和上面的執行計畫對比下我們可以知道，加入條件 where bill\_month='200803' 後，實際上卻是在遞迴完成後，最後才執行的 1 - filter("TEST"."BILL\_MONTH"='200803') 。

所以，為了確保語句的性能，不要直接加入條件在start with connect by 結構中，而是要想辦法將原表的資料控制住。這個可以採用子查詢的辦法，或者使用臨時表等（最好採用臨時表，將資料量從本源上控制住；因為從子查詢的執行計畫我們可以看到，它每次也都是訪問全表，再用條件過濾，要重複三次，不是一次過濾就夠了).

--直接加入條件後的執行計畫

SQL> explain plan for

2

2 select \* from test

3 where bill\_month='200803'

4 start with day\_number=1

5 connect by prior day\_number=day\_number-1 and prior msisdn= msisdn

6 ;

Explained

SQL> select \* from table( dbms\_xplan.display);

PLAN\_TABLE\_OUTPUT

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

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

| Id | Operation | Name | Rows | Bytes | Cost |

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

| 0 | SELECT STATEMENT | | | | |

|\* 1 | FILTER | | | | |

|\* 2 | CONNECT BY WITH FILTERING| | | | |

|\* 3 | FILTER | | | | |

| 4 | TABLE ACCESS FULL | TEST | | | |

| 5 | NESTED LOOPS | | | | |

| 6 | BUFFER SORT | | | | |

| 7 | CONNECT BY PUMP | | | | |

|\* 8 | TABLE ACCESS FULL | TEST | | | |

| 9 | TABLE ACCESS FULL | TEST | | | |

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

Predicate Information (identified by operation id):

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

1 - filter("TEST"."BILL\_MONTH"='200803')

PLAN\_TABLE\_OUTPUT

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

2 - filter("TEST"."DAY\_NUMBER"=1)

3 - filter("TEST"."DAY\_NUMBER"=1)

8 - filter("TEST"."MSISDN"=Null AND "TEST"."DAY\_NUMBER"-1=Null)

Note: rule based optimization

25 rows selected

SQL>

--使用子查詢，將過濾條件嵌在子查詢中

SQL> explain plan for

2

2 select \* from (select \* from test

3 where bill\_month='200803')

4 start with day\_number=1

5 connect by prior day\_number=day\_number-1 and prior msisdn= msisdn

6 ;

Explained

SQL> select \* from table(dbms\_xplan.display);

PLAN\_TABLE\_OUTPUT

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

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

| Id | Operation | Name | Rows | Bytes | Cost |

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

| 0 | SELECT STATEMENT | | | | |

|\* 1 | CONNECT BY WITH FILTERING| | | | |

|\* 2 | FILTER | | | | |

|\* 3 | TABLE ACCESS FULL | TEST | | | |

| 4 | NESTED LOOPS | | | | |

| 5 | BUFFER SORT | | | | |

| 6 | CONNECT BY PUMP | | | | |

|\* 7 | TABLE ACCESS FULL | TEST | | | |

|\* 8 | TABLE ACCESS FULL | TEST | | | |

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

Predicate Information (identified by operation id):

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

1 - filter("TEST"."DAY\_NUMBER"=1)

2 - filter("TEST"."DAY\_NUMBER"=1)

PLAN\_TABLE\_OUTPUT

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

3 - filter("TEST"."BILL\_MONTH"='200803')

7 - filter("TEST"."BILL\_MONTH"='200803' AND "TEST"."MSISDN"=Null AND

"TEST"."DAY\_NUMBER"-1=Null)

8 - filter("TEST"."BILL\_MONTH"='200803')

Note: rule based optimization

26 rows selected

SQL>

4. 實際中 遞迴查詢的使用。

問題：

資料庫裡有欄位day\_number，msisdn。如何寫月度連續3天有記錄的手機號？表結構如下：

id bill\_month day\_number msisdn

1 200803 1 13800000000

2 200803 1 130137.....

3 200803 2 13800000000

4 200803 3 13800000000

..............................

表中3月份連續3天有記錄的紀錄就是1380000000。請問如何寫這樣的sql？

解決方案：

SQL> create table test ( bill\_month Varchar2(20),day\_number number ,msisdn Varchar2(20));

Table created

SQL> insert into test values ( '200803',1,'13800');

1 row inserted

SQL> insert into test values ( '200803',3,'13800');

1 row inserted

SQL> insert into test values ( '200803',2,'13800');

1 row inserted

SQL> insert into test values ( '200803',2,'13801');

1 row inserted

SQL> insert into test values ( '200803',4,'13804');

1 row inserted

SQL> insert into test values ( '200803',5,'13804');

1 row inserted

SQL> commit;

Commit complete

SQL> select \* from test;

BILL\_MONTH DAY\_NUMBER MSISDN

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

200803 1 13800

200803 3 13800

200803 2 13800

200803 2 13801

200803 4 13804

200803 5 13804

6 rows selected

SQL>

SQL> select distinct msisdn from test a

2 where bill\_month='200803'

3 and exists

4 ( select msisdn from test

5 where bill\_month='200803' and msisdn=a.msisdn

6 start with day\_number=a.day\_number

7 connect by prior day\_number=day\_number-1 and prior msisdn= msisdn

8 group by msisdn

9 having count(\*)>=3

10 );

MSISDN

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

13800

SQL> select \* from test;

BILL\_MONTH DAY\_NUMBER MSISDN

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

200803 1 13800

200803 3 13800

200803 2 13800

200803 2 13801

200803 4 13804

200803 5 13804

6 rows selected

SQL> insert into test values ( '200803',7,'13804');

1 row inserted

SQL> insert into test values ( '200803',8,'13804');

1 row inserted

SQL> insert into test values ( '200803',6,'13802');

1 row inserted

SQL> insert into test values ( '200803',6,'13801');

1 row inserted

SQL> insert into test values ( '200803',7,'13801');

1 row inserted

SQL> insert into test values ( '200803',8,'13801');

1 row inserted

SQL> select \* from test;

BILL\_MONTH DAY\_NUMBER MSISDN

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

200803 1 13800

200803 3 13800

200803 2 13800

200803 2 13801

200803 4 13804

200803 5 13804

200803 7 13804

200803 8 13804

200803 6 13802

200803 6 13801

200803 7 13801

200803 8 13801

12 rows selected

SQL> commit;

Commit complete

SQL>

SQL> select distinct msisdn from test a

2 where bill\_month='200803'

3 and exists

4 ( select msisdn from test

5 where bill\_month='200803' and msisdn=a.msisdn

6 start with day\_number=a.day\_number

7 connect by prior day\_number=day\_number-1 and prior msisdn= msisdn

8 group by msisdn

9 having count(\*)>=3

10 );

MSISDN

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

13800

13801

SQL>