ORACLE递归查询（适用于ID,PARENTID结构数据表） - 业精于勤而荒于嬉 - 博客园

oracle树查询的最重要的就是select…start with…connect by…prior语法了。依托于该语法，我们可以将一个表形结构的以树的顺序列出来。在下面列述了oracle中树型查询的常用查询方式以及经常使用的与树查询相关的oracle特性函数等，在这里只涉及到一张表中的树查询方式而不涉及多表中的关联等。

1、准备测试表和测试数据

```
--菜单目录结构表
createtabletb_menu(
   id number(10) not null, --主键id
   title varchar2(50), --标题
   parent number(10) --parent id
)
--父菜单
insertintotb_menu(id, title, parent) values(1, '父菜单1', null);
insertintotb_menu(id, title, parent) values(2, '父菜单2', null);
insertintotb_menu(id, title, parent) values(3, '父菜单3', null);
insertintotb_menu(id, title, parent) values(4, '父菜单4', null);
insertintotb_menu(id, title, parent) values(5, '父菜单5', null);
--一级菜单
insertintotb_menu(id, title, parent)
```

```
values(
6, '一
级菜单6
',1);
inserti
ntotb_m
enu(id,
title,
parent)
values(
7, '一
级菜单7
',1);
inserti
ntotb_m
enu(id,
title,
parent)
values(
8, '一
级菜单8
',1);
inserti
ntotb_m
enu(id,
title,
parent)
values(
9, '一
级菜单9
',2);
inserti
ntotb_m
enu(id,
title,
parent)
values(
10, '一
级菜单1
0',2);
inserti
ntotb_m
enu(id,
title,
parent)
values(
11, '一
级菜单1
1',2);
inserti
ntotb_m
enu(id,
title,
parent)
values(
12, '一
级菜单1
2',3);
inserti
ntotb_m
enu(id,
title,
parent)
values(
13, '一
级菜单1
3',3);
inserti
ntotb_m
enu(id,
title,
parent)
values(
14, '一
级菜单1
4',3);
inserti
ntotb_m
enu(id,
title,
parent)
values(
15, '一
级菜单1
5',4);
inserti
ntotb_m
enu(id,
title,
parent)
values(
```

```
values(
16, '一
级菜单1
6',4);
inserti
ntotb_m
enu(id,
title,
parent)
values(
17, '一
级菜单1
7',4);
inserti
ntotb_m
enu(id,
title,
parent)
values(
18, '一
级菜单1
8',5);
inserti
ntotb_m
enu(id,
title,
parent)
values(
19, '一
级菜单1
9',5);
inserti
ntotb_m
enu(id,
title,
parent)
values(
20, '一
级菜单2
0',5);
--二级
菜单
inserti
ntotb_m
enu(id,
title,
parent)
values(
21, '二
级菜单2
1',6);
inserti
ntotb_m
enu(id,
title,
parent)
values(
22, '二
级菜单2
2',6);
inserti
ntotb_m
enu(id,
title,
parent)
values(
23, '二
级菜单2
3',7);
inserti
ntotb_m
enu(id,
title,
parent)
values(
24, '二
级菜单2
4',7);
inserti
ntotb_m
enu(id,
title,
parent)
values(
25, '二
级菜单2
5',8);
inserti
ntotb_m
enu(id,
title,
parent)
```

```
parent)
values(
26, '二
级菜单2
6',9);
inserti
ntotb_m
enu(id,
title,
parent)
values(
27, '二
级菜单2
7',10);
inserti
ntotb_m
enu(id,
title,
parent)
values(
28, '二
级菜单2
8',11);
inserti
ntotb_m
enu(id,
title,
parent)
values(
29, '二
级菜单2
9',12);
inserti
ntotb_m
enu(id,
title,
parent)
values(
30, '二
级菜单3
0',13);
inserti
ntotb_m
enu(id,
title,
parent)
values(
31, '二
级菜单3
1',14);
inserti
ntotb_m
enu(id,
title,
parent)
values(
32, '二
级菜单3
2',15);
inserti
ntotb_m
enu(id,
title,
parent)
values(
33, '二
级菜单3
3',16);
inserti
ntotb_m
enu(id,
title,
parent)
values(
34, '二
级菜单3
4',17);
inserti
ntotb_m
enu(id,
title,
parent)
values(
35, '二
级菜单3
5',18);
inserti
ntotb_m
enu(id,
title,
parent)
```

```
values(
36, '二
级菜单3
6',19);
inserti
ntotb_m
enu(id,
title,
parent)
values(
37, '二
级菜单3
7',20);
--三级
菜单
inserti
ntotb_m
enu(id,
title,
parent)
values(
38, '三
级菜单3
8',21);
inserti
ntotb_m
enu(id,
title,
parent)
values(
39, '三
级菜单3
9',22);
inserti
ntotb_m
enu(id,
title,
parent)
values(
40, '三
级菜单4
0',23);
inserti
ntotb_m
enu(id,
title,
parent)
values(
41, '三
级菜单4
1',24);
inserti
ntotb_m
enu(id,
title,
parent)
values(
42, '三
级菜单4
2',25);
inserti
ntotb_m
enu(id,
title,
parent)
values(
43, '三
级菜单4
3',26);
inserti
ntotb_m
enu(id,
title,
parent)
values(
44, '三
级菜单4
4',27);
inserti
ntotb_m
enu(id,
title,
parent)
values(
45, '三
级菜单4
5',28);
inserti
ntotb_m
enu(id,
title,
```

```
parent)
values(
46, '三
级菜单4
6',28);
inserti
ntotb_m
enu(id,
title,
parent)
values(
47, '三
级菜单4
7',29);
inserti
ntotb_m
enu(id,
title,
parent)
values(
48, '三
级菜单4
8',30);
inserti
ntotb_m
enu(id,
title,
parent)
values(
49, '三
级菜单4
9',31);
inserti
ntotb_m
enu(id,
title,
parent)
values(
50, '三
级菜单5
0',31);
commit;
select*
fromtb_
menu;
```

parent字段存储的是上级id，如果是顶级父节点，该parent为null(得补充一句，当初的确是这样设计的，不过现在知道，表中最好别有null记录，这会引起全文扫描，建议改成0代替)。

2、树操作

我们从最基本的操作，逐步列出树查询中常见的操作，所有查询出来的节点以家族中的辈份作比方。

1)、查找树中的所有顶级父节点（辈份最长的人）。 假设这个树是个目录结构，那么第一个操作总是找出所有的顶级节点，再根据该节点找到其下属节点。

```
1
select
* from
tb_menu
m where
m.paren
t is nu
ll;
```

2)、查找一个节点的直属子节点（所有儿子）。 如果查找的是直属子类节点，也是不用用到树型查询的。

```
1
select
* from
tb_menu
m where
m.paren
t=1;
```

3)、查找一个节点的所有直属子节点（所有后代）。

```
1
select
* from
tb_menu
m start
with m.
id=1con
nect by
m.paren
t=prior
m.id;
```

这个查找的是id为1的节点下的所有直属子类节点，包括子辈的和孙子辈的所有直属节点。

4)、查找一个节点的直属父节点（父亲）。 如果查找的是节点的直属父节点，也是不用用到树型查询的。

```
--c-->child, p->parent
select c.id, c.title, p.id parent_id, p.title parent_title
from tb_menu c, tb_menu p
where c.parent=p.id and c.id=6
```

5)、查找一个节点的所有直属父节点（祖宗）。

```
select * from tb_menu m start with m.id=38connect by prior m.parent=m.id;
```

这里查找的就是id为1的所有直属父节点，打个比方就是找到一个人的父亲、祖父等。但是值得注意的是这个查询出来的结果的顺序是先列出子类节点再列出父类节点，姑且认为是个倒序吧。
上面列出两个树型查询方式，第3条语句和第5条语句，这两条语句之间的区别在于prior关键字的位置不同，所以决定了查询的方式不同。 当parent = prior id时，数据库会根据当前的id迭代出parent与该id相同的记录，所以查询的结果是迭代出了所有的子类记录；而prior parent = id时，数据库会跟据当前的parent来迭代出与当前的parent相同的id的记录，所以查询出来的结果就是所有的父类结果。
以下是一系列针对树结构的更深层次的查询，这里的查询不一定是最优的查询方式，或许只是其中的一种实现而已。
6)、查询一个节点的兄弟节点（亲兄弟）。

```
--m.parent=m2.parent-->同一个父亲
select * from tb_menu m
where exists ( select * from tb_menu m2 where m.parent=m2.parent and m2.id=6)
```

7)、查询与一个节点同级的节点（族兄弟）。 如果在表中设置了级别的字段，那么在做这类查询时会很轻松，同一级别的就是与那个节点同级的，在这里列出不使用该字段时的实现!

```
with tmp as(
select a.*, level leaf
from tb_menu a
start with a.parent is null
connect by a.parent = prior a.id)
select *
from tmp
where leaf = (select leaf from tmp where id = 50);
```

这里使用两个技巧，一个是使用了level来标识每个节点在表中的级别，还有就是使用with语法模拟出了一张带有级别的临时表。

8)、查询一个节点的父节点的的兄弟节点（伯父与叔父）。

```
with tmp as(
select tb_menu.*, level lev
from tb_menu
start with parent is null
connect by parent = prior id)
select b.*
from tmp b,(select * from tmp
where id = 21and lev = 2) a
where b.lev = 1
union all
select *
from tmp
where parent = (select distinct x.id
from tmp x, --祖父
tmp y, --父亲
(select *
from tmp
where id = 21and lev > 2) z --儿子
where y.id = z.parent and x.id = y.parent);
```

这里查询分成以下几步。

首先，将第7个一样，将全表都使用临时表加上级别；

其次，根据级别来判断有几种类型，以上文中举的例子来说，有三种情况：

（1）当前节点为顶级节点，即查询出来的lev值为1，那么它没有上级节点，不予考虑。

（2）当前节点为2级节点，查询出来的lev值为2，那么就只要保证lev级别为1的就是其上级节点的兄弟节点。

（3）其它情况就是3以及以上级别，那么就要选查询出来其上级的上级节点（祖父），再来判断祖父的下级节点都是属于该节点的上级节点的兄弟节点。

最后，就是使用union将查询出来的结果进行结合起来，形成结果集。

9)、查询一个节点的父节点的同级节点（族叔）。

这个其实跟第7种情况是相同的。

```
with tmp as(
select a.*, level leaf
from tb_menu a
start with a.parent is null
connect by a.parent = prior a.id)
select *
from tmp
where leaf = (select leaf from tmp where id = 6) - 1;
```

基本上，常见的查询在里面了，不常见的也有部分了。其中，查询的内容都是节点的基本信息，都是数据表中的基本字段，但是在树查询中还有些特殊需求，是对查询数据进行了处理的，常见的包括列出树路径等。
补充一个概念，对于数据库来说，根节点并不一定是在数据库中设计的顶级节点，对于数据库来说，根节点就是start with开始的地方。
下面列出的是一些与树相关的特殊需求。
10)、名称要列出名称全部路径。
这里常见的有两种情况，一种是从顶级列出，直到当前节点的名称（或者其它属性）；一种是从当前节点列出，直到顶级节点的名称（或其它属性）。举地址为例：国内的习惯是从省开始、到市、到县、到居委会的，而国外的习惯正好相反（老师说的，还没接过国外的邮件，谁能寄个瞅瞅）。
从顶部开始：

```
select sys_connect_by_path (title, '/')
from tb_menu
where id = 50
start with parent is null
connect by parent = prior id;
```

从当前节点开始：

```
select sys_connect_by_path (title, '/')
from tb_menu
start with id = 50
connect by prior parent = id;
```

在这里我又不得不放个牢骚了。oracle只提供了一个sys_connect_by_path函数，却忘了字符串的连接的顺序。在上面的例子中，第一个sql是从根节点开始遍历，而第二个sql是直接找到当前节点，从效率上来说已经是千差万别，更关键的是第一个sql只能选择一个节点，而第二个sql却是遍历出了一颗树来。再次ps一下。
sys_connect_by_path函数就是从start with开始的地方开始遍历，并记下其遍历到的节点，start with开始的地方被视为根节点，将遍历到的路径根据函数中的分隔符，组成一个新的字符串，这个功能还是很强大的。

11)、列出当前节点的根节点。
在前面说过，根节点就是start with开始的地方。

```
1
2
3
4
```

```
select
connect
_by_roo
t title
, tb_me
nu.*
from tb
_menu
start w
ith id
= 50
connect
by prio
r paren
t = id;
```

connect_by_root函数用来列的前面，记录的是当前节点的根节点的内容。
12)、列出当前节点是否为叶子。
这个比较常见，尤其在动态目录中，在查出的内容是否还有下级节点时，这个函数是很适用的。

```
1
2
3
4
```

```
select
connect
_by_isl
eaf, tb
_menu.*
from tb
_menu
start w
ith par
ent is
null
connect
by pare
nt = pr
ior id;
```

connect_by_isleaf函数用来判断当前节点是否包含下级节点，如果包含的话，说明不是叶子节点，这里返回0；反之，如果不包含下级节点，这里返回1。
至此，oracle树型查询基本上讲完了，以上的例子中的数据是使用到做过的项目中的数据，因为里面的内容可能不好理解，所以就全部用一些新的例子来进行阐述。以上所有sql都在本机上测试通过，也都能实现相应的功能，但是并不能保证是解决这类问题的最优方案（如第8条明显写成存储过程会更好）.