START WITH and CONNECT BY in Oracle SQL

```
select ... start with initial-condition connect by nocycle recurse-condition

select ... connect by recurse-condition

select ... start with initial-condition connect by nocycle recurse-condition

select ... connect by recurse-condition
```

The **start with .. connect by** clause can be used to select data that has a hierarchical relationship (usually some sort of parent->child (boss->employee or thing->parts).

It is also being used when an sql execution plan is explained.

`recurse-condition` can make use of the keyword **prior**:

```
connect by
  prior foo = bar
```

This construct establishes the recursion. All records that are part of the next lower hierarchical level are found by having bar = foo. foo is a value found in the current hierarchical level.

A simple example

In the following example, the table from which that data is selected consists of just these attributes: parent and child. We make sure (by means of a unique constraint) that the child is uniqe within the table. This is just like in the real life where (as of yet) a child cannot have two different mothers.

The data filled into the table is such that a the sum over the children with the same parent is the value of the parent:

```
set feedback off

create table test_connect_by (
  parent     number,
  child      number,
  constraint uq_tcb unique (child)
);
```

5 = 2+3

```
insert into test_connect_by values ( 5, 2);
insert into test_connect_by values ( 5, 3);
```

18 = 11+7

```
insert into test_connect_by values (18,11);
insert into test_connect_by values (18, 7);
```

17 = 9+8

```
insert into test_connect_by values (17, 9);
insert into test_connect_by values (17, 8);
```

26 = 13+1+12

```
insert into test_connect_by values (26,13);
insert into test_connect_by values (26, 1);
insert into test_connect_by values (26,12);
```

15=10+5

```
insert into test_connect_by values (15,10);
insert into test_connect_by values (15, 5);
```

38=15+17+6

```
insert into test_connect_by values (38,15);
insert into test_connect_by values (38,17);
insert into test_connect_by values (38, 6);
```

38, 26 and 18 have no parents (the parent is null)

```
insert into test_connect_by values (null, 38);
insert into test_connect_by values (null, 26);
insert into test_connect_by values (null, 18);
```

Now, let's select the data hierarchically:

```
select lpad(' ',2*(level-1)) || to_char(child) s
  from test_connect_by
  start with parent is null
  connect by prior child = parent;
```

This select statement results in:

```
38
  15
    10
    5
      2
      3
  17
    9
    8
  6
26
  13
  1
  12
18
  11
  7
```

Interpreting connect by statements

How must a `start with ... connect by` select statement be read and interpreted? If Oracle encounters such an SQL statement, it proceeds as described in the following pseude code.

```
for rec in (select * from some_table) loop
  if FULLFILLS_START_WITH_CONDITION(rec) then
    RECURSE(rec, rec.child);
  end if;
end loop;

procedure RECURSE (rec in MATCHES_SELECT_STMT, parent_id IN field_type) is
  begin
  APPEND_RESULT_LIST(rec);
  for rec_recurse in (select * from some_table) loop
    if FULLFILLS_CONNECT_BY_CONDITION(rec_recurse.id, parent_id) then
      RECURSE(rec_recurse,rec_recurse.id);
    end if;
  end loop;
end procedure RECURSE;
```

Thanks to **Frank Trenkamp** who spotted an error in the logic in the above pseudo code and corrected it.

Thanks also to **Abhishek Ghose** who made me think about a better way to describe the logic.

Pruning branches

Sometimes, it might be a requirement to only partially retrieve a hierarchical tree and to prune branches. Here, a tree is filled. Each child is the number of its parent plus a new digit on the right side.

```
create table prune_test (
  parent  number,
  child   number
);

insert into prune_test values (null,   1);
insert into prune_test values (null,   6);
insert into prune_test values (null,   7);

insert into prune_test values (   1,  12);
insert into prune_test values (   1,  14);
insert into prune_test values (   1,  15);

insert into prune_test values (   6,  61);
insert into prune_test values (   6,  63);
insert into prune_test values (   6,  65);
insert into prune_test values (   6,  69);

insert into prune_test values (   7,  71);
insert into prune_test values (   7,  74);

insert into prune_test values (  12, 120);
insert into prune_test values (  12, 124);
insert into prune_test values (  12, 127);

insert into prune_test values (  65, 653);

insert into prune_test values (  71, 712);
insert into prune_test values (  71, 713);
insert into prune_test values (  71, 715);

insert into prune_test values (  74, 744);
insert into prune_test values (  74, 746);
insert into prune_test values (  74, 748);

insert into prune_test values ( 712,7122);
insert into prune_test values ( 712,7125);
insert into prune_test values ( 712,7127);

insert into prune_test values ( 748,7481);
insert into prune_test values ( 748,7483);
insert into prune_test values ( 748,7487);
```

Now, we want to retrieve the tree, but prune everything below the branch 1 and 71. It would be false to put these into a where clause of the sql statement, rather, it belongs to the connect by clause:

```
select
  lpad(' ', 2*level) || child
from
```

```
  prune_test
start with
  parent is null
connect by
  prior child=parent
  and parent not in (1, 71);
```

This returns:

```
  1
  6
    61
    63
    65
      653
    69
  7
    71
    74
      744
      746
      748
        7481
        7483
        7487
```

See also another example for pruning.

Do two items stand in a ancestor descendant relationship

Sometimes, one want's to know if two items are in an ancestor descendant relationship, that is if XYZ as grandfather, or grand-grandfather, or ... of ABC. The following template of a query can be used to determine that.

```
set feedback off

drop table parent_child;

create table parent_child(parent_ varchar2(20), child_ varchar2(20));

insert into parent_child values (null,  'a')

insert into parent_child values (  'a',  'af');
insert into parent_child values (  'a',  'ab');
insert into parent_child values (  'a',  'ax');

insert into parent_child values ( 'ab', 'abc');
insert into parent_child values ( 'ab', 'abd');
insert into parent_child values ( 'ab', 'abe');
```

```
insert into parent_child values ('abe','abes');
insert into parent_child values ('abe','abet');

insert into parent_child values ( null,   'b');

insert into parent_child values (  'b',  'bg');
insert into parent_child values (  'b',  'bh');
insert into parent_child values (  'b',  'bi');

insert into parent_child values ( 'bi', 'biq');
insert into parent_child values ( 'bi', 'biv');
insert into parent_child values ( 'bi', 'biw');
```

The following query 'asks' for a parent and a supposed child (grand child, grand grand child) and answers the question if the are indeed in an ancester successor relationship.

```
set verify off

select
  case when count(*) > 0 then
    '&&parent is an ancestor of &&child' else
    '&&parent is no ancestor of &&child' end
    "And here's the answer"
from
  parent_child
where
  child_ = '&&child'
start with
  parent_ = '&&parent'
connect by
  prior child_ = parent_;

undefine child
undefine parent
```

Features of 9i

sys_connect_by_path

With sys_connect_by_path it is possible to show the entire path from the top level down to the 'actual' child:

Using hierarchical result sets

With this technique, it is possible to show all kind of hierarchical data relations. Here is an example that lists privileges, roles and users in their hierarchical relation.

See also flat hiearchy.

connect_by_root

connect_by_root is a new operator that comes with Oracle 10g and enhances the ability to perform hierarchical queries.

I have yet to dig into this subject and will write about it when things become clearer.

connect_by_is_leaf

connect_by_isleaf is a new operator that comes with Oracle 10g and enhances the ability to perform hierarchical queries.

I have yet to dig into this subject and will write about it when things become clearer.

connect_by_iscycle

connect_by_is_cycle is a new operator that comes with Oracle 10g and enhances the ability to perform hierarchical queries.

I have yet to dig into this subject and will write about it when things become clearer.

Thanks

Thanks to **Peter Bruhn**, **Jonathan Schmalze**, **Jeff Jones**, **Keith Britch** and **Fabian Iturralde** who each pointed out an error, misstake or typo on this page.

Further links

On storing hierarchical data

On summing up values in nodes of a hierachical query.