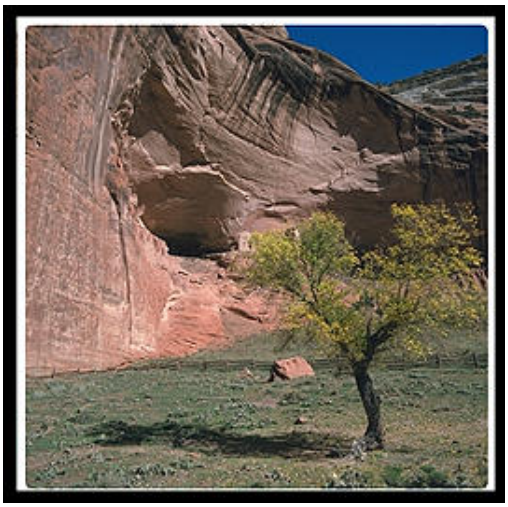
Representing Trees in Oracle SQL

15 min. read · View original



On its face, the relational database management system would appear to be a very poor tool for representing and manipulating trees. This chapter is designed to accomplish the following things:

- show you that a row in an SQL database can be thought of as an object
- show you that a pointer from one object to another can be represented by storing an integer key in a regular database column
- demonstrate the Oracle tree extensions (CONNECT BY ... PRIOR)
- show you how to work around the limitations of CONNECT BY with PL/SQL

The canonical example of trees in Oracle is the org chart.

```
create table corporate_slaves (
       slave_id                integer primary key,
       supervisor_id           references
corporate_slaves,
       name                    varchar(100)
);
```

```
insert into corporate_slaves values (1, NULL, 'Big Boss
Man');
insert into corporate_slaves values (2, 1, 'VP
Marketing');
insert into corporate_slaves values (3, 1, 'VP Sales');
insert into corporate_slaves values (4, 3, 'Joe Sales
Guy');
insert into corporate_slaves values (5, 4, 'Bill Sales
Assistant');
insert into corporate_slaves values (6, 1, 'VP
Engineering');
insert into corporate_slaves values (7, 6, 'Jane Nerd');
insert into corporate_slaves values (8, 6, 'Bob Nerd');

SQL> column name format a20
SQL> select * from corporate_slaves;

  SLAVE_ID SUPERVISOR_ID NAME
---------- ------------- --------------------
         1               Big Boss Man
         2             1 VP Marketing
         3             1 VP Sales
         4             3 Joe Sales Guy
         6             1 VP Engineering
         7             6 Jane Nerd
         8             6 Bob Nerd
         5             4 Bill Sales Assistant

8 rows selected.
```

The integers in the `supervisor_id` are actually pointers to other rows in the `corporate_slaves` table. Need to display an org chart? With only standard SQL available, you'd write a program in the client language (e.g., C, Lisp, Perl, or Tcl) to do the following:

1. query Oracle to find the employee `where supervisor_id is`

`null`, call this `$big_kahuna_id`
2. query Oracle to find those employees whose `supervisor_id = $big_kahuna_id`
3. for each subordinate, query Oracle again to find their subordinates.
4. repeat until no subordinates found, then back up one level

With the Oracle CONNECT BY clause, you can get all the rows out at once:

```
select name, slave_id, supervisor_id
from corporate_slaves
connect by prior slave_id = supervisor_id;

NAME                   SLAVE_ID SUPERVISOR_ID
-------------------- ---------- -------------
Big Boss Man                1
VP Marketing                2             1
VP Sales                    3             1
Joe Sales Guy               4             3
Bill Sales Assistant        5             4
VP Engineering              6             1
Jane Nerd                   7             6
Bob Nerd                    8             6

VP Marketing                2             1

VP Sales                    3             1
Joe Sales Guy               4             3
Bill Sales Assistant        5             4

Joe Sales Guy               4             3
Bill Sales Assistant        5             4

VP Engineering              6             1
Jane Nerd                   7             6
Bob Nerd                    8             6

Jane Nerd                   7             6

Bob Nerd                    8             6

Bill Sales Assistant        5             4
```

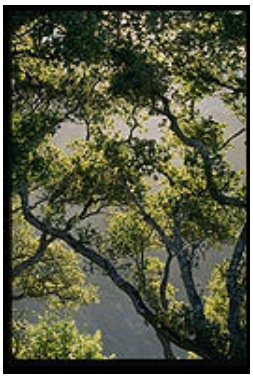
```
20 rows selected.
```

This seems a little strange. It looks as though Oracle has produced all possible trees and subtrees. Let's add a START WITH clause:

```
select name, slave_id, supervisor_id
from corporate_slaves
connect by prior slave_id = supervisor_id
start with slave_id in (select slave_id
                        from corporate_slaves
                        where supervisor_id is null);

NAME                   SLAVE_ID SUPERVISOR_ID
-------------------- ---------- -------------
Big Boss Man                  1
VP Marketing                  2             1
VP Sales                      3             1
Joe Sales Guy                 4             3
Bill Sales Assistant          5             4
VP Engineering                6             1
Jane Nerd                     7             6
Bob Nerd                      8             6

8 rows selected.
```



Notice that we've used a subquery in the START WITH clause to find out who is/are the big kahuna(s). For the rest of this example, we'll just hard-code in the `slave_id` 1 for brevity.

Though these folks are in the correct order, it is kind of tough to tell from the preceding report who works for whom. Oracle provides a magic pseudo-column that is meaningful only when a query includes a CONNECT BY. The pseudo-column is `level`:

```
select name, slave_id, supervisor_id, level
from corporate_slaves
connect by prior slave_id = supervisor_id
start with slave_id = 1;

NAME                   SLAVE_ID SUPERVISOR_ID      LEVEL
-------------------- ---------- ------------- ----------
Big Boss Man                1                         1
VP Marketing                2             1           2
VP Sales                    3             1           2
Joe Sales Guy               4             3           3
Bill Sales Assistant        5             4           4
VP Engineering              6             1           2
Jane Nerd                   7             6           3
Bob Nerd                    8             6           3

8 rows selected.
```

The `level` column can be used for indentation. Here we will use the concatenation operator (`||`) to add spaces in front of the name column:

```
column padded_name format a30

select
  lpad(' ', (level - 1) * 2) || name as padded_name,
  slave_id,
  supervisor_id,
  level
from corporate_slaves
connect by prior slave_id = supervisor_id
start with slave_id = 1;

PADDED_NAME                    SLAVE_ID SUPERVISOR_ID
LEVEL
------------------------------ ---------- -------------
----------
Big Boss Man                            1
1
  VP Marketing                          2             1
2
  VP Sales                              3             1
2
    Joe Sales Guy                       4             3
3
```

```
      Bill Sales Assistant              5             4
4
  VP Engineering                        6             1
2
    Jane Nerd                           7             6
3
    Bob Nerd                            8             6
3

8 rows selected.
```

If you want to limit your report, you can use standard WHERE clauses:

```
select
  lpad(' ', (level - 1) * 2) || name as padded_name,
  slave_id,
  supervisor_id,
  level
from corporate_slaves
where level <= 3
connect by prior slave_id = supervisor_id
start with slave_id = 1;

PADDED_NAME                     SLAVE_ID SUPERVISOR_ID
LEVEL
------------------------------ ---------- -------------
----------
Big Boss Man                            1
1
  VP Marketing                          2             1
2
  VP Sales                              3             1
2
    Joe Sales Guy                       4             3
3
  VP Engineering                        6             1
2
    Jane Nerd                           7             6
3
    Bob Nerd                            8             6
3

7 rows selected.
```

Suppose that you want people at the same level to sort alphabetically. Sadly, the ORDER BY clause doesn't work so great in conjunction with CONNECT BY:

```
select
  lpad(' ', (level - 1) * 2) || name as padded_name,
  slave_id,
  supervisor_id,
  level
from corporate_slaves
connect by prior slave_id = supervisor_id
start with slave_id = 1
order by level, name;

PADDED_NAME                     SLAVE_ID SUPERVISOR_ID
LEVEL
------------------------------ ---------- -------------
----------
Big Boss Man                            1
1
  VP Engineering                        6             1
2
  VP Marketing                          2             1
2
  VP Sales                              3             1
2
    Bob Nerd                            8             6
3
    Jane Nerd                           7             6
3
    Joe Sales Guy                       4             3
3
      Bill Sales Assistant              5             4
4

select
  lpad(' ', (level - 1) * 2) || name as padded_name,
  slave_id,
  supervisor_id,
  level
from corporate_slaves
connect by prior slave_id = supervisor_id
start with slave_id = 1
order by name;
```

```
PADDED_NAME                    SLAVE_ID SUPERVISOR_ID
LEVEL
------------------------------ ---------- -------------
----------
Big Boss Man                            1
1
      Bill Sales Assistant              5             4
4
    Bob Nerd                            8             6
3
    Jane Nerd                           7             6
3
    Joe Sales Guy                       4             3
3
  VP Engineering                        6             1
2
  VP Marketing                          2             1
2
  VP Sales                              3             1
2
```

SQL is a set-oriented language. In the result of a CONNECT BY query, it is precisely the order that has value. Thus it doesn't make much sense to also have an ORDER BY clause.

JOIN doesn't work with CONNECT BY

If we try to build a report showing each employee and his or her supervisor's name, we are treated to one of Oracle's few informative error messages:

```
select
  lpad(' ', (level - 1) * 2) || cs1.name as padded_name,
  cs2.name as supervisor_name
from corporate_slaves cs1, corporate_slaves cs2
where cs1.supervisor_id = cs2.slave_id(+)
connect by prior cs1.slave_id = cs1.supervisor_id
start with cs1.slave_id = 1;
```

```
ERROR at line 4:
ORA-01437: cannot have join with CONNECT BY
```

We can work around this particular problem by creating a view:

```
create or replace view connected_slaves
as
select
  lpad(' ', (level - 1) * 2) || name as padded_name,
  slave_id,
  supervisor_id,
  level as the_level
from corporate_slaves
connect by prior slave_id = supervisor_id
start with slave_id = 1;
```

Notice that we've had to rename `level` so that we didn't end up with a view column named after a reserved word. The view works just like the raw query:

```
select * from connected_slaves;

PADDED_NAME                    SLAVE_ID SUPERVISOR_ID
THE_LEVEL
------------------------------ ---------- -------------
----------
Big Boss Man                            1
1
  VP Marketing                          2             1
2
  VP Sales                              3             1
2
    Joe Sales Guy                       4             3
3
      Bill Sales Assistant              5             4
4
  VP Engineering                        6             1
2
    Jane Nerd                           7             6
3
    Bob Nerd                            8             6
3

8 rows selected.
```

but we can JOIN now

```
select padded_name, corporate_slaves.name as
supervisor_name
from connected_slaves, corporate_slaves
where connected_slaves.supervisor_id =
corporate_slaves.slave_id(+);

PADDED_NAME                    SUPERVISOR_NAME
------------------------------ --------------------
Big Boss Man
  VP Marketing                 Big Boss Man
  VP Sales                     Big Boss Man
    Joe Sales Guy              VP Sales
      Bill Sales Assistant     Joe Sales Guy
  VP Engineering               Big Boss Man
    Jane Nerd                  VP Engineering
    Bob Nerd                   VP Engineering

8 rows selected.
```

If you have sharp eyes, you'll notice that we've actually OUTER JOINed so that our results don't exclude the big boss.

Select-list subqueries *do* work with CONNECT BY

Instead of the VIEW and JOIN, we could have added a subquery to the select list:

```
select
  lpad(' ', (level - 1) * 2) || name as padded_name,
  (select name
   from corporate_slaves cs2
   where cs2.slave_id = cs1.supervisor_id) as
supervisor_name
from corporate_slaves cs1
connect by prior slave_id = supervisor_id
start with slave_id = 1;

PADDED_NAME                    SUPERVISOR_NAME
------------------------------ --------------------
Big Boss Man
  VP Marketing                 Big Boss Man
  VP Sales                     Big Boss Man
    Joe Sales Guy              VP Sales
```

```
      Bill Sales Assistant     Joe Sales Guy
  VP Engineering               Big Boss Man
    Jane Nerd                  VP Engineering
    Bob Nerd                   VP Engineering

8 rows selected.
```

The general rule in Oracle is that you can have a subquery that returns a single row anywhere in the select list.

Does this person work for me?

Suppose that you've built an intranet Web service. There are things that your software should show to an employee's boss (or boss's boss) that it shouldn't show to a subordinate or peer. Here we try to figure out if the VP Marketing (#2) has supervisory authority over Jane Nerd (#7):

```
select count(*)
from corporate_slaves
where slave_id = 7
and level > 1
start with slave_id = 2
connect by prior slave_id = supervisor_id;

  COUNT(*)
----------
         0
```

Apparently not. Notice that we start with the VP Marketing (#2) and stipulate `level > 1` to be sure that we will never conclude that someone supervises him or herself. Let's ask if the Big Boss Man (#1) has authority over Jane Nerd:

```
select count(*)
from corporate_slaves
where slave_id = 7
and level > 1
start with slave_id = 1
connect by prior slave_id = supervisor_id;

  COUNT(*)
----------
         1
```

Even though Big Boss Man isn't Jane Nerd's direct supervisor, asking Oracle to compute the relevant subtree yields us the correct result. In

the ArsDigita Community System Intranet module, we decided that this computation was too important to be left as a query in individual Web pages. We centralized the question in a PL/SQL procedure:

```
create or replace function intranet_supervises_p
  (query_supervisor IN integer, query_user_id IN
integer)
return varchar
is
  n_rows_found integer;
BEGIN
  select count(*) into n_rows_found
   from intranet_users
   where user_id = query_user_id
   and level > 1
   start with user_id = query_supervisor
   connect by supervisor = PRIOR user_id;
  if n_rows_found > 0 then
        return 't';
  else
        return 'f';
  end if;
END intranet_supervises_p;
```

Family trees

What if the graph is a little more complicated than employee-supervisor? For example, suppose that you are representing a family tree. Even without allowing for divorce and remarriage, exotic South African fertility clinics, etc., we still need more than one pointer for each node:

```
create table family_relatives (
        relative_id     integer primary key,
        spouse          references family_relatives,
        mother          references family_relatives,
        father          references family_relatives,
        -- in case they don't know the exact birthdate
        birthyear       integer,
        birthday        date,
        -- sadly, not everyone is still with us
        deathyear       integer,
        first_names     varchar(100) not null,
        last_name       varchar(100) not null,
        sex             char(1) check (sex in
```

```
('m','f')),
        -- note the use of multi-column check
constraints
        check ( birthyear is not null or birthday is not
null)
);

-- some test data


insert into family_relatives
(relative_id, first_names, last_name, sex, spouse,
mother, father, birthyear)
values
(1, 'Nick', 'Gittes', 'm', NULL, NULL, NULL, 1902);

insert into family_relatives
(relative_id, first_names, last_name, sex, spouse,
mother, father, birthyear)
values
(2, 'Cecile', 'Kaplan', 'f', 1, NULL, NULL, 1910);

update family_relatives
set spouse = 2
where relative_id = 1;

insert into family_relatives
(relative_id, first_names, last_name, sex, spouse,
mother, father, birthyear)
values
(3, 'Regina', 'Gittes', 'f', NULL, 2, 1, 1934);

insert into family_relatives
(relative_id, first_names, last_name, sex, spouse,
mother, father, birthyear)
values
(4, 'Marjorie', 'Gittes', 'f', NULL, 2, 1, 1936);

insert into family_relatives
(relative_id, first_names, last_name, sex, spouse,
mother, father, birthyear)
values
(5, 'Shirley', 'Greenspun', 'f', NULL, NULL, NULL,
1901);
```

```
insert into family_relatives
(relative_id, first_names, last_name, sex, spouse,
mother, father, birthyear)
values
(6, 'Jack', 'Greenspun', 'm', 5, NULL, NULL, 1900);

update family_relatives
set spouse = 6
where relative_id = 5;

insert into family_relatives
(relative_id, first_names, last_name, sex, spouse,
mother, father, birthyear)
values
(7, 'Nathaniel', 'Greenspun', 'm', 3, 5, 6, 1930);

update family_relatives
set spouse = 7
where relative_id = 3;

insert into family_relatives
(relative_id, first_names, last_name, sex, spouse,
mother, father, birthyear)
values
(8, 'Suzanne', 'Greenspun', 'f', NULL, 3, 7, 1961);

insert into family_relatives
(relative_id, first_names, last_name, sex, spouse,
mother, father, birthyear)
values
(9, 'Philip', 'Greenspun', 'm', NULL, 3, 7, 1963);

insert into family_relatives
(relative_id, first_names, last_name, sex, spouse,
mother, father, birthyear)
values
(10, 'Harry', 'Greenspun', 'm', NULL, 3, 7, 1965);
```

In applying the lessons from the employee examples, the most obvious problem that we face now is whether to follow the mother or the father pointers:

```


column full_name format a25

-- follow patrilineal (start with my mom's father)
```

```
select lpad(' ', (level - 1) * 2) || first_names || ' '
|| last_name as full_name
from family_relatives
connect by prior relative_id = father
start with relative_id = 1;

FULL_NAME
-------------------------
Nick Gittes
  Regina Gittes
  Marjorie Gittes

-- follow matrilineal (start with my mom's mother)
select lpad(' ', (level - 1) * 2) || first_names || ' '
|| last_name as full_name
from family_relatives
connect by prior relative_id = mother
start with relative_id = 2;

FULL_NAME
-------------------------
Cecile Kaplan
  Regina Gittes
    Suzanne Greenspun
    Philip Greenspun
    Harry Greenspun
  Marjorie Gittes
```

Here's what the official Oracle docs have to say about CONNECT BY:

> specifies the relationship between parent rows and child rows of the hierarchy. condition can be any condition as described in "Conditions". However, some part of the condition must use the PRIOR operator to refer to the parent row. The part of the condition containing the PRIOR operator must have one of the following forms:
>
> `PRIOR expr comparison_operator expr`
>
> `expr comparison_operator PRIOR expr`

There is nothing that says `comparison_operator` has to be merely the equals sign. Let's start again with my mom's father but CONNECT BY more than one column:

```
-- follow both
select lpad(' ', (level - 1) * 2) || first_names || ' '
|| last_name as full_name
```

```
from family_relatives
connect by prior relative_id in (mother, father)
start with relative_id = 1;

FULL_NAME
-------------------------
Nick Gittes
  Regina Gittes
    Suzanne Greenspun
    Philip Greenspun
    Harry Greenspun
  Marjorie Gittes
```

Instead of arbitrarily starting with Grandpa Nick, let's ask Oracle to show us all the trees that start with a person whose parents are unknown:

```
select lpad(' ', (level - 1) * 2) || first_names || ' '
|| last_name as full_name
from family_relatives
connect by prior relative_id in (mother, father)
start with relative_id in (select relative_id from
family_relatives
                           where mother is null
                           and father is null);

FULL_NAME
-------------------------
Nick Gittes
  Regina Gittes
    Suzanne Greenspun
    Philip Greenspun
    Harry Greenspun
  Marjorie Gittes
Cecile Kaplan
  Regina Gittes
    Suzanne Greenspun
    Philip Greenspun
    Harry Greenspun
  Marjorie Gittes
Shirley Greenspun
  Nathaniel Greenspun
    Suzanne Greenspun
    Philip Greenspun
    Harry Greenspun
```

```
Jack Greenspun
  Nathaniel Greenspun
    Suzanne Greenspun
    Philip Greenspun
    Harry Greenspun

22 rows selected.
```

PL/SQL instead of JOIN

The preceding report is interesting but confusing because it is hard to tell where the trees meet in marriage. As noted above, you can't do a JOIN with a CONNECT BY. We demonstrated the workaround of burying the CONNECT BY in a view. A more general workaround is using PL/SQL:

```
create or replace function family_spouse_name
  (v_relative_id family_relatives.relative_id%TYPE)
return varchar
is
  v_spouse_id integer;
  spouse_name varchar(500);
BEGIN
  select spouse into v_spouse_id
    from family_relatives
    where relative_id = v_relative_id;
  if v_spouse_id is null then
    return null;
  else
    select (first_names || ' ' || last_name) into
spouse_name
      from family_relatives
      where relative_id = v_spouse_id;
    return spouse_name;
  end if;
END family_spouse_name;
/
show errors

column spouse format a20
```

```
select
  lpad(' ', (level - 1) * 2) || first_names || ' ' ||
last_name as full_name,
  family_spouse_name(relative_id) as spouse
from family_relatives
connect by prior relative_id in (mother, father)
start with relative_id in (select relative_id from
family_relatives
                           where mother is null
                           and father is null);

FULL_NAME                 SPOUSE
------------------------- --------------------
Nick Gittes               Cecile Kaplan
  Regina Gittes           Nathaniel Greenspun
    Suzanne Greenspun
    Philip Greenspun
    Harry Greenspun
  Marjorie Gittes
Cecile Kaplan             Nick Gittes
  Regina Gittes           Nathaniel Greenspun
    Suzanne Greenspun
    Philip Greenspun
    Harry Greenspun
  Marjorie Gittes
Shirley Greenspun         Jack Greenspun
  Nathaniel Greenspun     Regina Gittes
    Suzanne Greenspun
    Philip Greenspun
    Harry Greenspun
Jack Greenspun            Shirley Greenspun
  Nathaniel Greenspun     Regina Gittes
    Suzanne Greenspun
    Philip Greenspun
    Harry Greenspun
```

PL/SQL instead of JOIN and GROUP BY

Suppose that in addition to displaying the family tree in a Web page, we also want to show a flag when a story about a family member is available. First we need a way to represent stories:

```
create table family_stories (
        family_story_id         integer primary key,
```

```
        story                   clob not null,
        item_date               date,
        item_year               integer,
        access_control          varchar(20)
             check (access_control in ('public',
'family', 'designated')),
        check (item_date is not null or item_year is not
null)
);

-- a story might be about more than one person
create table family_story_relative_map (
        family_story_id         references
family_stories,
        relative_id             references
family_relatives,
        primary key (relative_id, family_story_id)
);

-- put in a test story
insert into family_stories
(family_story_id, story, item_year, access_control)
values
(1, 'After we were born, our parents stuck the Wedgwood
in a cabinet
and bought indestructible china.  Philip and his father
were sitting at
the breakfast table one morning.  Suzanne came
downstairs and, without
saying a word, took a cereal bowl from the cupboard,
walked over to
Philip and broke the bowl over his head.  Their father
immediately
started laughing hysterically.', 1971, 'public');

insert into family_story_relative_map
(family_story_id, relative_id)
values
(1, 8);

insert into family_story_relative_map
(family_story_id, relative_id)
values
(1, 9);
```

```
insert into family_story_relative_map
(family_story_id, relative_id)
values
(1, 7);
```

To show the number of stories alongside a family member's listing, we would typically do an OUTER JOIN and then GROUP BY the columns other than the `count(family_story_id)`. In order not to disturb the CONNECT BY, however, we create another PL/SQL function:

```
create or replace function family_n_stories
(v_relative_id family_relatives.relative_id%TYPE)
return integer
is
  n_stories integer;
BEGIN
  select count(*) into n_stories
    from family_story_relative_map
    where relative_id = v_relative_id;
  return n_stories;
END family_n_stories;
/
show errors

select
  lpad(' ', (level - 1) * 2) || first_names || ' ' ||
last_name as full_name,
  family_n_stories(relative_id) as n_stories
from family_relatives
connect by prior relative_id in (mother, father)
start with relative_id in (select relative_id from
family_relatives
                           where mother is null
                           and father is null);

FULL_NAME                  N_STORIES
-------------------------- ----------
Nick Gittes                         0
...
Shirley Greenspun                   0
  Nathaniel Greenspun               1
    Suzanne Greenspun               1
    Philip Greenspun                1
    Harry Greenspun                 0
```

```
...
```

Working Backwards

What does it look like to start at the youngest generation and work back?

```
select
  lpad(' ', (level - 1) * 2) || first_names || ' ' ||
last_name as full_name,
  family_spouse_name(relative_id) as spouse
from family_relatives
connect by relative_id in (prior mother, prior father)
start with relative_id = 9;

FULL_NAME                 SPOUSE
------------------------- --------------------
Philip Greenspun
  Regina Gittes           Nathaniel Greenspun
    Nick Gittes           Cecile Kaplan
    Cecile Kaplan         Nick Gittes
  Nathaniel Greenspun     Regina Gittes
    Shirley Greenspun     Jack Greenspun
    Jack Greenspun        Shirley Greenspun
```

We ought to be able to view all the trees starting from all the leaves but Oracle seems to be exhibiting strange behavior:

```
select
  lpad(' ', (level - 1) * 2) || first_names || ' ' ||
last_name as full_name,
  family_spouse_name(relative_id) as spouse
from family_relatives
connect by relative_id in (prior mother, prior father)
start with relative_id not in (select mother from
family_relatives
                               union
                               select father from
```

```
family_relatives);

no rows selected
```

What's wrong? If we try the subquery by itself, we get a reasonable result. Here are all the `relative_id`s that appear in the `mother` or `father` column at least once.

```
select mother from family_relatives
union
select father from family_relatives

    MOTHER
----------
         1
         2
         3
         5
         6
         7


7 rows selected.
```

The answer lies in that extra blank line at the bottom. There is a NULL in this result set. Experimentation reveals that Oracle behaves asymmetrically with NULLs and IN and NOT IN:

```
SQL> select * from dual where 1 in (1,2,3,NULL);

D
-
X

SQL> select * from dual where 1 not in (2,3,NULL);

no rows selected
```

The answer is buried in the Oracle documentation of NOT IN: "Evaluates to FALSE if any member of the set is NULL." The correct query in this case?

```
select
  lpad(' ', (level - 1) * 2) || first_names || ' ' ||
last_name as full_name,
```

```
  family_spouse_name(relative_id) as spouse
from family_relatives
connect by relative_id in (prior mother, prior father)
start with relative_id not in (select mother
                               from family_relatives
                               where mother is not null
                               union
                               select father
                               from family_relatives
                               where father is not
null);

FULL_NAME                 SPOUSE
------------------------- --------------------
Marjorie Gittes
  Nick Gittes             Cecile Kaplan
  Cecile Kaplan           Nick Gittes
Suzanne Greenspun
  Regina Gittes           Nathaniel Greenspun
    Nick Gittes           Cecile Kaplan
    Cecile Kaplan         Nick Gittes
  Nathaniel Greenspun     Regina Gittes
    Shirley Greenspun     Jack Greenspun
    Jack Greenspun        Shirley Greenspun
Philip Greenspun
  Regina Gittes           Nathaniel Greenspun
    Nick Gittes           Cecile Kaplan
    Cecile Kaplan         Nick Gittes
  Nathaniel Greenspun     Regina Gittes
    Shirley Greenspun     Jack Greenspun
    Jack Greenspun        Shirley Greenspun
Harry Greenspun
  Regina Gittes           Nathaniel Greenspun
    Nick Gittes           Cecile Kaplan
    Cecile Kaplan         Nick Gittes
  Nathaniel Greenspun     Regina Gittes
    Shirley Greenspun     Jack Greenspun
    Jack Greenspun        Shirley Greenspun

24 rows selected.
```

Performance and Tuning

Oracle is not getting any help from the Tree Fairy in producing results from a CONNECT BY. If you don't want tree queries to take O(N^2) time, you need to build indices that let Oracle very quickly answer questions of the form "What are all the children of Parent X?"

For the corporate slaves table, you'd want two concatenated indices:

```
create index corporate_slaves_idx1
  on corporate_slaves (slave_id, supervisor_id);
create index corporate_slaves_idx2
  on corporate_slaves (supervisor_id, slave_id);
```

Reference

Gratuitous Photos

Next: dates

philg@mit.edu

Reader's Comments

> Oracle9i does CONNECT BY on joins. It also adds an "ORDER SIBLINGS BY" clause, fixing the omission that prevents you from ordering each level of the query.
>
> Couldn't find the article at Dartmouth :(, it looked really interesting!
>
> -- Andrew Wolfe, March 24, 2004

> Interested readers should check out Joe Celko's nested set model for representing trees in SQL. No need to be locked into proprietary SQL dialects and probably a couple of orders of magnitude faster to query!
>
> Here's some links...
>
> + http://www.intelligententerprise.com/001020/celko.jhtml
> + http://www.dbmsmag.com/9603d06.html
> + http://www.dbmsmag.com/9604d06.html
> + http://www.dbmsmag.com/9605d06.html
> + http://www.dbmsmag.com/9606d06.html
> + http://www.sqlteam.com/Forums/topic.asp?TOPIC_ID=14099
> + http://www.dbazine.com/oracle/or-articles/tropashko4
> + http://mrnaz.com/static/articles/trees_in_sql_tutorial/mptt_overview.php
>
> Regards, Mattster
>
> -- Matt Anon, April 3, 2007

Add a comment

Related Links

- representing an m-ary tree in sql- This method allows for very fast retrieval of descendants and modification of an m-ary tree. no self-referencing or nested select statements are necessary to retrieve some or all

descendants. the labelling of nodes is such that it allows very simple and fast querying for DFS order of nodes. it was partially inspired by huffman encoding. (contributed by Anthony D'Auria)

- Dead link- The link above to Dartmouth college appears to be dead, but Web Archive kept a copy of the page (contributed by Tom Lebr)

Add a link