Lecture 33: Local Definitions, Recursive Queries

Local Tables

- SQL provides a way to create (essentially) a temporary table for use in one select.
- Analogous to the **let** expression in Scheme.
- Here, foreigner is a one-column table local to this statement.

```
with foreigner(person) as (
   select "Martin" union
   select "Christina" union
   select "Johanna"
)
select child from people, foreigner
   where people.parent = foreigner.person;
```

What does this do?

people

parent	child
Martin	George
Christina	George
George	Martin F
Johanna	Martin F
George N	Paul
George N	Ann
George N	John
Martin F	George N
Martin F	Robert
Martin F	Donald
Donald	Peter

Example: Ancestry Relationships

- What does the program on the left do?
- (distinct removes duplicate rows.)

```
with kin(first, second) as (
    select a.child, b.child
        from people as a, people as b
        where a.parent = b.parent
        and a.child != b.child )
select distinct kin.second, child
     from people, kin
     where kin.first = parent;
```

people

parent	child
Martin	George
Christina	George
George	Martin F
Johanna	Martin F
George N	Paul
George N	Ann
George N	John
Martin F	George N
Martin F	Robert
Martin F	Donald
Donald	Peter

Recursion, Yet Again

- As with Python, Scheme, and streams, (limited) recursion is possible in SQL using the with clause.
- General form:

```
with
  table_name(column_names) as (
    select ... union     -- Base case
    select ... union     -- Base case
    select ... from ..., table_name, ...
  )
select ...
```

- The recursively defined table must appear only once in the from clause of the last select in the with clause.
- Because of these restrictions, no mutual recursions or tree recursions are allowed.

Example: Integers

- Define the table ints to contain integers from 1–30:

```
create table ints as
    with ints(n) as (
       select 1 union
       select n+1 from ints where n<=30
    )
select n from ints;
```

- Here, I've chosen to use `ints` for both the local and global tables.
- Usual sort of scope rules apply: the local `ints` is distinct from the global one, so I didn't have to make up a new name.

Defining Ancestor Recursively

- An *ancestor* is a parent or an ancestor of a parent.

```
with
    related(ancestor, descendant) as (
        select parent, child from people union
        select ancestor, child from related, people
            where descendant = parent
    )
select ancestor from related where descendant = "Paul";
```

A Famous Number

- There is a famous story about the "interesting' number 1729, the first of the "taxicab numbers."
- Given our table `ints` (numbers up to 50) how do we find them?