Public Service Announcement

"**ATTN Latina/o Coders:**

The Hispanic Heritage Foundation and the Infosys Foundations USA are excited to host a LOFT Coder Summit at Stanford University on Saturday, May 6th from 8:00 am - 5:00 pm, as hundreds of Latina/o coders gather to share ideas, energy and cultural pride! For more details and to register visit `lcsrsvp.com`.

This summit is part of HHF's broader Code as a Second Language (CSL) national initiative which has included LOFT Coder Summits in Austin at SXSW, New York, Minneapolis and Stanford University, The Rio Grande Valley and Los Angeles!

If you are a Latina/o coder, programmer, hacker, developer, and/ or a computer scientist, we invite you to be a part of this one-of-a-kind experience. The summit is a free one-day event filled with back to back workshops, discussions, and opportunities to expand your network. Please join us in redefining the landscape of computer technology through a heightened collaboration and representation of like-minded Latina/o students and professionals, all united and ignited by their endless passion for technology."

Lecture 29: SQL Aggregation and Recursion

- Abstractly, a `select` statement that lists multiple tables filters *all possible combinations of rows* from those tables.

```
> create table T1 as
   select "a" as val union select "b";
> create table T2 as
   select 1 as val union select 2;
> select T1.val, T2.val from T1, T2;
a|1
a|2
b|1
b|2
```

Comparison to Python

- This includes the case where the same table is named twice, as in

```
select A.val, B.val from T1 as A, T1 as B;
a|a
a|b
b|a
b|b
```

- Thus, the `select ... from ...` part is rather like the `for` part of a list comprehension in Python:

```
[ (A.val, B.val) for A in T1 for B in T1 ]
```

- The `where` clause is now a filter, like the `if` clause in a list comprehension.

```
> select A.val, B.val from T1 as A, T1 as B
   where A.val <= B.val;
a|a
a|b
b|b
```

...is like

```
[ (A.val, B.val) for A in T1 for B in T1 if A.val <= B.val ]
```

Expressions

- Familiar arithmetic is possible:

```
> select 3 + 4;
7
> select 3+GP from grade_values;
7
7
6.7
6
6.3
...
```

- Also string operations (not quite like Python):

```
> select First || " " || Last from students;
Jason Knowles
Valerie Chan
...
```

Aggregation

- Certain expressions *aggregate* results:

```
> select avg(GP) from grades, grade_values
    where Letter=Grade and SID = 101;
3.25

> select max(GP) from grades, grade_values
    where Letter=Grade and SID = 101;
3.7
> select count(GP) from grades, grade_values
    where Letter=Grade and SID = 101;
4
```

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."
- The story told by G. H. Hardy describes a meeting between him and Srinivasa Ramanujan:

> "I remember once going to see [Ramanujan] when he was lying ill at Putney. I had ridden in taxi-cab No. 1729, and remarked that the number seemed to be rather a dull one, and that I hoped it was not an unfavourable omen. 'No,' he replied, 'it is a very interesting number; it is the smallest [integer] number expressible as the sum of two [positive] cubes in two different ways.'"

- Given our table `ints` (numbers up to 50) how do we find such numbers?

Solution

```
with cubes(a, b, c) as (
   select x.n, y.n, x.n*x.n*x.n + y.n*y.n*y.n
   from ints as x, ints as y where x.n <= y.n
)
select left.a, left.b, right.a, right.b, left.c
   from cubes as left, cubes as right
   where left.a < right.a and left.c = right.c;
```