Logic Programming

In SQL's limited form of recursion, a table can be constructed by iterating through rows.

**WM 18:** When dogs are stacked on top of one another, the total height of the stack is the sum of the heights of the dogs; list dogs in increasing order of height within a stack.

```
with
  stacks(names, n, total, tallest) as ( 
    select names, 1, height, height from dogs 
    union
    select names || '.' || name, n-1, total-height, height 
    from stacks, dogs 
    where n < 4 and tallest < height
  )
select names, total from stacks where ...;
```

**Demo**

```
abraham,clinton,barack
abraham,clinton
abraham,barack
abraham
clinton
```

**HW 10**

In SQL's limited form of recursion, a table can be constructed by iterating through rows.

```
union
```

**Demo**

```
abraham,clinton,barack
abraham,clinton
abraham,barack
abraham
clinton
```

Implementing Recursive Tables

**Reminder: a Select Class**

The SQL parser creates an instance of the `Select` class for each `select` statement.

```
>>> class Select:
    ...   """select \[columns\] from \[tables\] where \[condition\]."""
    ...   def __init__(self, columns, tables, condition):
    ...       self.columns = columns
    ...       self.tables = tables
    ...       self.condition = condition
    ...       self.make_row = create_make_row(self.columns)
    ...       def execute(self, env):
    ...           return eval(self.make_row, filtered_rows)
    ...   return filter(filter_fn, from_rows)
    ...   def __call__(self, row):
    ...       return eval(self.condition, row)
    ...   return True
```

Simplified version of http://composingprograms.com/examples/sql/sql_exec.py

Logic Programming

Logic programming languages (such as Prolog) are more powerful declarative languages.

In Prolog:
- All rows in a table have the same number of columns, restricted to primitive values.
- No mutual recursion: two or more tables cannot be defined in terms of each other.
- No tree recursion: the table being defined can only appear once in a from clause.

In SQL:
- Output can contain structured data (sequences, trees, etc.)
- Any kind of recursion is allowed.
- Scaling challenges.
- Programs may not terminate, even if the output is finite.

```
integers(n) = ¬n < 1 
ints(n) = integers(n-1) + n 
ints(1) = 1 
```

```
sums(exp, value) = values
sums(a.exp, value + a.value) = values
```

**Demo**

```
1
2
3
4
```

```
1
2
3
4
```

```
1
2
3
4
```

Addition

What if we could write a tree-recursive `select` statement?

```
select sum expressions that evaluate to a number less than 3
```

```
integers() as {
  select 1 union select 2 union select 3
},
sums(exp, value) as {
  select n from integers
  where n < 4 and n = a.value + b.value <= 3
  from sums(a.exp, b.value)
}
```

```
exp value
1 1
2 2
3 3
```

```
exp value
1 1
2 2
3 3
```

```
exp value
1 1
2 2
3 3
```

```
exp value
1 1
2 2
3 3
```
Thanks for being amazing!
Please stay for the HKN survey