Announcements
Announcements

- Homework 11 due Wednesday 12/10 @ 11:59pm
Announcements

• Homework 11 due Wednesday 12/10 @ 11:59pm
  • All you have to do is vote on your favorite recursive art
Announcements

• Homework 11 due Wednesday 12/10 @ 11:59pm
  • All you have to do is vote on your favorite recursive art
• Homework 12 due Wednesday 12/10 @ 11:59pm
Announcements

• Homework 11 due Wednesday 12/10 @ 11:59pm
  ▪ All you have to do is vote on your favorite recursive art
• Homework 12 due Wednesday 12/10 @ 11:59pm
  ▪ All you have to do is complete the final online survey
Announcements

• Homework 11 due Wednesday 12/10 @ 11:59pm
  ▪ All you have to do is vote on your favorite recursive art
• Homework 12 due Wednesday 12/10 @ 11:59pm
  ▪ All you have to do is complete the final online survey
  ▪ Grading and last remarks
Announcements

• Homework 11 due Wednesday 12/10 @ 11:59pm
  ▪ All you have to do is vote on your favorite recursive art

• Homework 12 due Wednesday 12/10 @ 11:59pm
  ▪ All you have to do is complete the final online survey
  ▪ Grading and last remarks

• 30 hours of review sessions next week! Pick and choose topics.
Announcements

• Homework 11 due Wednesday 12/10 @ 11:59pm
  ▪ All you have to do is vote on your favorite recursive art
• Homework 12 due Wednesday 12/10 @ 11:59pm
  ▪ All you have to do is complete the final online survey
  ▪ Grading and last remarks
• 30 hours of review sessions next week! Pick and choose topics.
• The final exam is on Thursday 12/18 @ 3pm emphasizing:
Announcements

• Homework 11 due Wednesday 12/10 @ 11:59pm
  ▪ All you have to do is vote on your favorite recursive art
• Homework 12 due Wednesday 12/10 @ 11:59pm
  ▪ All you have to do is complete the final online survey
• Grading and last remarks
• 30 hours of review sessions next week! Pick and choose topics.
• The final exam is on Thursday 12/18 @ 3pm emphasizing:
  ▪ Functions and environments
Announcements

- Homework 11 due Wednesday 12/10 @ 11:59pm
  - All you have to do is vote on your favorite recursive art
- Homework 12 due Wednesday 12/10 @ 11:59pm
  - All you have to do is complete the final online survey
  - Grading and last remarks
- 30 hours of review sessions next week! Pick and choose topics.
- The final exam is on Thursday 12/18 @ 3pm emphasizing:
  - Functions and environments
  - Sequences, trees, and recursion
Announcements

• Homework 11 due Wednesday 12/10 @ 11:59pm
  • All you have to do is vote on your favorite recursive art
• Homework 12 due Wednesday 12/10 @ 11:59pm
  • All you have to do is complete the final online survey
  • Grading and last remarks
• 30 hours of review sessions next week! Pick and choose topics.
• The final exam is on Thursday 12/18 @ 3pm emphasizing:
  • Functions and environments
  • Sequences, trees, and recursion
  • Mutation and object-oriented programming
Announcements

- Homework 11 due Wednesday 12/10 @ 11:59pm
  - All you have to do is vote on your favorite recursive art
- Homework 12 due Wednesday 12/10 @ 11:59pm
  - All you have to do is complete the final online survey
- Grading and last remarks
- 30 hours of review sessions next week! Pick and choose topics.
- The final exam is on Thursday 12/18 @ 3pm emphasizing:
  - Functions and environments
  - Sequences, trees, and recursion
  - Mutation and object-oriented programming
  - Iterators, generators, and streams
Announcements

- Homework 11 due Wednesday 12/10 @ 11:59pm
  - All you have to do is vote on your favorite recursive art
- Homework 12 due Wednesday 12/10 @ 11:59pm
  - All you have to do is complete the final online survey
- Grading and last remarks
- 30 hours of review sessions next week! Pick and choose topics.
- The final exam is on Thursday 12/18 @ 3pm emphasizing:
  - Functions and environments
  - Sequences, trees, and recursion
  - Mutation and object-oriented programming
  - Iterators, generators, and streams
  - Scheme and SQL
Implementing Recursive Tables
Reminder: a Select Class

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

Simplified version of http://composingprograms.com/examples/sql/sql_exec.py
Reminder: a Select Class

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

```python
>>> class Select:
    """select [columns] from [tables] where [condition].""
```

Simplified version of http://composingprograms.com/examples/sql/sql_exec.py
Reminder: a Select Class

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

```python
>>> 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)
```

Simplified version of http://composingprograms.com/examples/sql/sql_exec.py
Reminder: a Select Class

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

```python
>>> 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):
        """Join, filter, and map rows from tables to columns.""
        from_rows = join(self.tables, env)
        filtered_rows = filter(self.filter_fn, from_rows)
        return map(self.make_row, filtered_rows)
```

Reminder: a Select Class

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

```python
>>> 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):
        """Join, filter, and map rows from tables to columns.""
        from_rows = join(self.tables, env)
        filtered_rows = filter(self.filter_fn, from_rows)
        return map(self.make_row, filtered_rows)
    def filter_fn(self, row):
        if self.condition:
            return eval(self.condition, row)
        else:
            return True
```

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

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

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

**HW 10:** 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.*
Recursive Tables

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

HW 10: 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
Recursive Tables

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

HW 10: 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 (
Recursive Tables

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

HW 10: 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

```sql
with stacks(names, n, total, tallest) as (  
    select name, 1, height, height from dogs union
```
Recursive Tables

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

HW 10: 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 name, 1, height, height from dogs union  
    select names || ',' || name, n+1, total+height, height  
    from stacks, dogs  
    where n < 4 and tallest < height  
)
Recursive Tables

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

HW 10: 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 name, 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 ...;
Recursive Tables

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

HW 10: 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 name, 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 ...;
```

<table>
<thead>
<tr>
<th>names</th>
<th>n</th>
<th>total</th>
<th>tallest</th>
</tr>
</thead>
<tbody>
<tr>
<td>abraham</td>
<td>1</td>
<td>26</td>
<td>26</td>
</tr>
</tbody>
</table>
Recursive Tables

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

**HW 10:** 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*

```sql
with
    stacks(names, n, total, tallest) as (  
        select name, 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 ...;
```

<table>
<thead>
<tr>
<th>names</th>
<th>total</th>
<th>tallest</th>
</tr>
</thead>
<tbody>
<tr>
<td>abraham</td>
<td>1</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>26</td>
<td></td>
</tr>
</tbody>
</table>
Recursive Tables

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

**HW 10:** 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*

```sql
WITH stacks(names, n, total, tallest) AS (
    SELECT name, 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
```

<table>
<thead>
<tr>
<th>names</th>
<th>n</th>
<th>total</th>
<th>tallest</th>
</tr>
</thead>
<tbody>
<tr>
<td>abraham</td>
<td>1</td>
<td>26</td>
<td>26</td>
</tr>
</tbody>
</table>
Recursive Tables

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

**HW 10:** 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 name, 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 ...;
```
Recursive Tables

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

**HW 10:** 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.

```sql
with
    stacks(names, n, total, tallest) as (  
        select name, 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 ...
```

<table>
<thead>
<tr>
<th>names</th>
<th>n</th>
<th>total</th>
<th>tallest</th>
</tr>
</thead>
<tbody>
<tr>
<td>abraham</td>
<td>1</td>
<td>26</td>
<td>26</td>
</tr>
</tbody>
</table>
Recursive Tables

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

**HW 10:** 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*

```sql
with
    stacks(names, n, total, tallest) as (  
        select name, 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 ...;
```

<table>
<thead>
<tr>
<th>names</th>
<th>n</th>
<th>total</th>
<th>tallest</th>
</tr>
</thead>
<tbody>
<tr>
<td>abraham</td>
<td>1</td>
<td>26</td>
<td>26</td>
</tr>
</tbody>
</table>
Recursive Tables

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

**HW 10:** 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*.

```sql
with stacks(names, n, total, tallest) as (  
    select name, 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 ...;
```

<table>
<thead>
<tr>
<th>names</th>
<th>n</th>
<th>total</th>
<th>tallest</th>
</tr>
</thead>
<tbody>
<tr>
<td>abraham</td>
<td>1</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>abraham,barack</td>
<td>2</td>
<td>78</td>
<td>52</td>
</tr>
</tbody>
</table>
Recursive Tables

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

HW 10: 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 name, 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 ...;
```
Recursive Tables

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

**HW 10:** 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*

```sql
with
    stacks(names, n, total, tallest) as (  
        select name, 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 ...;
```

<table>
<thead>
<tr>
<th>names</th>
<th>n</th>
<th>total</th>
<th>tallest</th>
</tr>
</thead>
<tbody>
<tr>
<td>abraham</td>
<td>1</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>abraham,barack</td>
<td>2</td>
<td>78</td>
<td>52</td>
</tr>
<tr>
<td>abraham,clinton</td>
<td>2</td>
<td>73</td>
<td>47</td>
</tr>
</tbody>
</table>
Recursive Tables

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

**HW 10:** 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*

```sql
with
  stacks(names, n, total, tallest) as (
    select name, 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 ...;
```

<table>
<thead>
<tr>
<th>names</th>
<th>n</th>
<th>total</th>
<th>tallest</th>
</tr>
</thead>
<tbody>
<tr>
<td>abraham</td>
<td>1</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>abraham,barack</td>
<td>2</td>
<td>78</td>
<td>52</td>
</tr>
<tr>
<td>abraham,clinton</td>
<td>2</td>
<td>73</td>
<td>47</td>
</tr>
</tbody>
</table>
Recursive Tables

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

**HW 10**: 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

```sql
with
  stacks(names, n, total, tallest) as (  
    select name, 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 ...;
```

<table>
<thead>
<tr>
<th>names</th>
<th>n</th>
<th>total</th>
<th>tallest</th>
</tr>
</thead>
<tbody>
<tr>
<td>abraham</td>
<td>1</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>abraham,barack</td>
<td>2</td>
<td>78</td>
<td>52</td>
</tr>
<tr>
<td>abraham,clinton</td>
<td>2</td>
<td>73</td>
<td>47</td>
</tr>
<tr>
<td>abraham,clinton,barack</td>
<td>3</td>
<td>125</td>
<td>52</td>
</tr>
</tbody>
</table>
Recursive Tables

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

**HW 10:** 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*

```sql
with
    stacks(names, n, total, tallest) as (  
        select name, 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 ...;
```

<table>
<thead>
<tr>
<th>names</th>
<th>n</th>
<th>total</th>
<th>tallest</th>
</tr>
</thead>
<tbody>
<tr>
<td>abraham</td>
<td>1</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>abraham,barack</td>
<td>2</td>
<td>78</td>
<td>52</td>
</tr>
<tr>
<td>abraham,clinton</td>
<td>2</td>
<td>73</td>
<td>47</td>
</tr>
<tr>
<td>abraham,clinton,barack</td>
<td>3</td>
<td>125</td>
<td>52</td>
</tr>
</tbody>
</table>
Recursive Tables

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

**HW 10:** 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*

```sql
with
    stacks(names, n, total, tallest) as (  
        select name, 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 ...;
```

<table>
<thead>
<tr>
<th>names</th>
<th>n</th>
<th>total</th>
<th>tallest</th>
</tr>
</thead>
<tbody>
<tr>
<td>abraham</td>
<td>1</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>abraham,barack</td>
<td>2</td>
<td>78</td>
<td>52</td>
</tr>
<tr>
<td>abraham,clinton</td>
<td>2</td>
<td>73</td>
<td>47</td>
</tr>
<tr>
<td>abraham,clinton,barack</td>
<td>3</td>
<td>125</td>
<td>52</td>
</tr>
<tr>
<td>barack</td>
<td>1</td>
<td>52</td>
<td>52</td>
</tr>
</tbody>
</table>
Recursive Tables

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

**HW 10:** 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 name, 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 ...;
```
Recursive Tables

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

**HW 10:** 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*

```sql
with
  stacks(names, n, total, tallest) as (
    select name, 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 ...
```

<table>
<thead>
<tr>
<th>names</th>
<th>n</th>
<th>total</th>
<th>tallest</th>
</tr>
</thead>
<tbody>
<tr>
<td>abraham</td>
<td>1</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>abraham,barack</td>
<td>2</td>
<td>78</td>
<td>52</td>
</tr>
<tr>
<td>abraham,clinton</td>
<td>2</td>
<td>73</td>
<td>47</td>
</tr>
<tr>
<td>abraham,clinton,barack</td>
<td>3</td>
<td>125</td>
<td>52</td>
</tr>
<tr>
<td>barack</td>
<td>1</td>
<td>52</td>
<td>52</td>
</tr>
<tr>
<td>clinton</td>
<td>1</td>
<td>47</td>
<td>47</td>
</tr>
</tbody>
</table>
Recursive Tables

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

**HW 10:** 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*

```sql
with
  stacks(names, n, total, tallest) as (
    select name, 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 ...;
```

<table>
<thead>
<tr>
<th>names</th>
<th>n</th>
<th>total</th>
<th>tallest</th>
</tr>
</thead>
<tbody>
<tr>
<td>abraham</td>
<td>1</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>abraham,barack</td>
<td>2</td>
<td>78</td>
<td>52</td>
</tr>
<tr>
<td>abraham,clinton</td>
<td>2</td>
<td>73</td>
<td>47</td>
</tr>
<tr>
<td>abraham,clinton,barack</td>
<td>3</td>
<td>125</td>
<td>52</td>
</tr>
<tr>
<td>barack</td>
<td>1</td>
<td>52</td>
<td>52</td>
</tr>
<tr>
<td>clinton</td>
<td>1</td>
<td>47</td>
<td>47</td>
</tr>
</tbody>
</table>
Recursive Tables

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

**HW 10:** 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

```sql
with
    stacks(names, n, total, tallest) as (  
        select name, 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 ...;
```

<table>
<thead>
<tr>
<th>names</th>
<th>n</th>
<th>total</th>
<th>tallest</th>
</tr>
</thead>
<tbody>
<tr>
<td>abraham</td>
<td>1</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>abraham,barack</td>
<td>2</td>
<td>78</td>
<td>52</td>
</tr>
<tr>
<td>abraham,clinton</td>
<td>2</td>
<td>73</td>
<td>47</td>
</tr>
<tr>
<td>abraham,clinton,barack</td>
<td>3</td>
<td>125</td>
<td>52</td>
</tr>
<tr>
<td>barack</td>
<td>1</td>
<td>52</td>
<td>52</td>
</tr>
<tr>
<td>clinton</td>
<td>1</td>
<td>47</td>
<td>47</td>
</tr>
<tr>
<td>clinton,barack</td>
<td>1</td>
<td>99</td>
<td>52</td>
</tr>
</tbody>
</table>
Recursive Tables

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

**HW 10**: 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

```sql
with
  stacks(names, n, total, tallest) as (
    select name, 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 ...;
```

<table>
<thead>
<tr>
<th>names</th>
<th>n</th>
<th>total</th>
<th>tallest</th>
</tr>
</thead>
<tbody>
<tr>
<td>abraham</td>
<td>1</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>abraham,barack</td>
<td>2</td>
<td>78</td>
<td>52</td>
</tr>
<tr>
<td>abraham,clinton</td>
<td>2</td>
<td>73</td>
<td>47</td>
</tr>
<tr>
<td>abraham,clinton,barack</td>
<td>3</td>
<td>125</td>
<td>52</td>
</tr>
<tr>
<td>barack</td>
<td>1</td>
<td>52</td>
<td>52</td>
</tr>
<tr>
<td>clinton</td>
<td>1</td>
<td>47</td>
<td>47</td>
</tr>
<tr>
<td>clinton,barack</td>
<td>1</td>
<td>99</td>
<td>52</td>
</tr>
</tbody>
</table>
Recursive Tables

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

**HW 10**: 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*.

```sql
with
    stacks(names, n, total, tallest) as (  
      select name, 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)

<table>
<thead>
<tr>
<th>names</th>
<th>n</th>
<th>total</th>
<th>tallest</th>
</tr>
</thead>
<tbody>
<tr>
<td>abraham</td>
<td>1</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>abraham,barack</td>
<td>2</td>
<td>78</td>
<td>52</td>
</tr>
<tr>
<td>abraham,clinton</td>
<td>2</td>
<td>73</td>
<td>47</td>
</tr>
<tr>
<td>abraham,clinton,barack</td>
<td>3</td>
<td>125</td>
<td>52</td>
</tr>
<tr>
<td>barack</td>
<td>1</td>
<td>52</td>
<td>52</td>
</tr>
<tr>
<td>clinton</td>
<td>1</td>
<td>47</td>
<td>47</td>
</tr>
<tr>
<td>clinton,barack</td>
<td>1</td>
<td>99</td>
<td>52</td>
</tr>
</tbody>
</table>
Logic Programming
Logic Programming Languages
Logic Programming Languages

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

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

In SQL:
Logic Programming Languages

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

In SQL:

• All rows in a table have the same number of columns, restricted to primitive values
Logic Programming Languages

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

In SQL:

- 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.
Logic Programming Languages

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

In SQL:

- 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.
Logic Programming Languages

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

In SQL:

- 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 Prolog:
Logic Programming Languages

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

In SQL:

- 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 Prolog:

- Output can contain structured data (sequences, trees, etc.).
Logic Programming Languages

Logic programming languages (such as Prolog) are more powerful declarative languages compared to SQL:

In SQL:

• 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 Prolog:

• Output can contain structured data (sequences, trees, etc.)
• Any kind of recursion is allowed
Logic Programming Languages

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

In SQL:

- 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 Prolog:

- Output can contain structured data (sequences, trees, etc.)
- Any kind of recursion is allowed
- Scaling challenges
Logic Programming Languages

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

In SQL:

- 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 Prolog:

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

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

Select ll sum expressions that evaluate to a number less than 3
Addition

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

Select all sum expressions that evaluate to a number less than 3

with
Addition

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

Select all sum expressions that evaluate to a number less than 3

```sql
with
  ints(n) as (
    select 1 union select 2 union select 3
  ),
```
What if we could write a tree-recursive `select` statement?

Select all sum expressions that evaluate to a number less than 3

```sql
with
  ints(n) as (
    select 1 union select 2 union select 3
  ),
  sums(exp, value) as (  
    select n, n from ints union
```
Addition

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

Select all sum expressions that evaluate to a number less than 3

```sql
with
  ints(n) as (  
    select 1 union select 2 union select 3
  ),
  sums(exp, value) as (  
    select n, n from ints union  
    select '(' || a.exp || '+' || b.exp || ')', a.value + b.value  
      from sums as a, sums as b where a.value + b.value <= 3
  )
```
What if we could write a tree-recursive select statement?

Select all sum expressions that evaluate to a number less than 3

with

  ints(n) as (  
    select 1 union select 2 union select 3  
  ),

sums(exp, value) as (  
  select n, n from ints union  
  select '( ' || a.exp || ' + ' || b.exp || ' )', a.value + b.value  
    from sums as a, sums as b where a.value + b.value <= 3  
  )

select ...;
Addition

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

Select all sum expressions that evaluate to a number less than 3

```sql
with
  ints(n) as (  
    select 1 union select 2 union select 3
  ),
  sums(exp, value) as (  
    select n, n from ints union  
    select '(' || a.exp || '+' || b.exp || ')', a.value + b.value  
    from sums as a, sums as b where a.value + b.value <= 3
  )
select ...;
```
What if we could write a tree-recursive `select` statement?

Select all sum expressions that evaluate to a number less than 3

```
with
  ints(n) as (  
    select 1 union select 2 union select 3  
  ),
  sums(exp, value) as (  
    select n, n from ints union
    select '(' || a.exp || '+' || b.exp || ')', a.value + b.value
    from sums as a, sums as b where a.value + b.value <= 3
  )
select ...;
```
Addition

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

Select ll sum expressions that evaluate to a number less than 3

```
with
  ints(n) as (  
    select 1 union select 2 union select 3 
  ),
  sums(exp, value) as ( 
    select n, n from ints union 
    select '(' || a.exp || '+' || b.exp || ')', a.value + b.value 
      from sums as a, sums as b 
      where a.value + b.value <= 3
  )
select ...;
```
Addition

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

Select all sum expressions that evaluate to a number less than 3

```sql
with
    ints(n) as (select 1 union select 2 union select 3),
    sums(exp, value) as (select n, n from ints union
                          select '(' || a.exp || '+' || b.exp || ')', a.value + b.value
                          from sums as a, sums as b where a.value + b.value <= 3)

select ...;
```

<table>
<thead>
<tr>
<th>exp</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>(1+1)</td>
<td>2</td>
</tr>
<tr>
<td>(1+2)</td>
<td>3</td>
</tr>
</tbody>
</table>
Addition

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

Select all sum expressions that evaluate to a number less than 3

```sql
with
  ints(n) as (  
    select 1 union select 2 union select 3  
  ),

  sums(exp, value) as (  
    select n, n from ints union  
    select '(': || a.exp || '|' || '+' || '|' || b.exp || '|')', a.value + b.value  
    from sums as a, sums as b where a.value + b.value <= 3  
  )

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

Select all sum expressions that evaluate to a number less than 3

```
with
  ints(n) as (  
    select 1 union select 2 union select 3
  ),
  sums(exp, value) as (  
    select n, n from ints union
    select '(' || a.exp || '+' || b.exp || ')', a.value + b.value
    from sums as a, sums as b where a.value + b.value <= 3
  )
select ...;
```
Addition

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

Select ll sum expressions that evaluate to a number less than 3

```
with
  ints(n) as (  
    select 1 union select 2 union select 3  
  ),
  sums(exp, value) as (  
    select n, n from ints union
    select '(' || a.exp || '+' || b.exp || ')', a.value + b.value
    from sums as a, sums as b where a.value + b.value <= 3  
  )
select ...;
```

<table>
<thead>
<tr>
<th>exp</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>(1+1)</td>
<td>2</td>
</tr>
<tr>
<td>(1+2)</td>
<td>3</td>
</tr>
<tr>
<td>(2+1)</td>
<td>3</td>
</tr>
<tr>
<td>((1+1)+1)</td>
<td>3</td>
</tr>
<tr>
<td>(1+(1+1))</td>
<td>3</td>
</tr>
</tbody>
</table>
Addition

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

Select ll sum expressions that evaluate to a number less than 3

```sql
with
    ints(n) as (  
      select 1 union select 2 union select 3
    ),
    sums(exp, value) as (  
      select n, n from ints union  
      select '(' || a.exp || ' + ' || b.exp || ')' as exp, a.value + b.value  
        from sums as a, sums as b where a.value + b.value <= 3
    )
select ...;
```

(Demo)
Life
Thanks for being amazing!

Please stay for the HKN survey