Announcements
Announcements

• Homework 8 due Wednesday 4/17 @ 11:59pm
Announcements

• Homework 8 due Wednesday 4/17 @ 11:59pm

• Please complete the course survey on resources! http://goo.gl/ajEBkT
Announcements

• Homework 8 due Wednesday 4/17 @ 11:59pm

• Please complete the course survey on resources! http://goo.gl/ajEBkT

• Project 4 due Thursday 4/23 @ 11:59pm
Announcements

• Homework 8 due Wednesday 4/17 @ 11:59pm

• Please complete the course survey on resources! [http://goo.gl/ajEBkT](http://goo.gl/ajEBkT)

• Project 4 due Thursday 4/23 @ 11:59pm
  
  ◦ Early point #1: Questions 1–12 submitted (correctly) by Friday 4/17 @ 11:59pm
Announcements

• Homework 8 due Wednesday 4/17 @ 11:59pm

• Please complete the course survey on resources! [http://goo.gl/ajEBkT](http://goo.gl/ajEBkT)

• Project 4 due Thursday 4/23 @ 11:59pm
  • Early point #1: Questions 1–12 submitted (correctly) by Friday 4/17 @ 11:59pm
  • Early point #2: All questions (including Extra Credit) by Wednesday 4/22 @ 11:59pm
Information Hiding
Attributes for Internal Use

An attribute name that starts with one underscore is not meant to be referenced externally.
Attributes for Internal Use

An attribute name that starts with one underscore is not meant to be referenced externally.

class FibIter:
    """An iterator over Fibonacci numbers."""
    def __init__(self):
        self._next = 0
        self._addend = 1

    def __next__(self):
        result = self._next
        self._addend, self._next = self._next, self._addend + self._next
        return result
An attribute name that starts with one underscore is not meant to be referenced externally.

```
class FibIter:
    """An iterator over Fibonacci numbers."""
    def __init__(self):
        self._next = 0
        self._addend = 1

    def __next__(self):
        result = self._next
        self._addend, self._next = self._next, self._addend + self._next
        return result
```

```python
>>> fibs = FibIter()
>>> [next(fibs) for _ in range(10)]
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```
Attributes for Internal Use

An attribute name that starts with one underscore is not meant to be referenced externally.

class FibIter:
    """An iterator over Fibonacci numbers."""
    def __init__(self):
        self._next = 0
        self._addend = 1

        """Please don't reference these directly. They may change."""
    def __next__(self):
        result = self._next
        self._addend, self._next = self._next, self._addend + self._next
        return result

>>> fibs = FibIter()
>>> [next(fibs) for _ in range(10)]
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
Attributes for Internal Use

An attribute name that starts with one underscore is not meant to be referenced externally.

class FibIter:
    """An iterator over Fibonacci numbers."""
    def __init__(self):
        self._next = 0
        self._addend = 1

    def __next__(self):
        result = self._next
        self._addend, self._next = self._next, self._addend + self._next
        return result

>>> fibs = FibIter()
>>> [next(fibs) for _ in range(10)]
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]

"Please don't reference these directly. They may change."

This naming convention is not enforced, but is typically respected.
Attributes for Internal Use

An attribute name that starts with one underscore is not meant to be referenced externally.

class FibIter:
    """An iterator over Fibonacci numbers."""
    def __init__(self):
        self._next = 0
        self._addend = 1

    def __next__(self):
        result = self._next
        self._addend, self._next = self._next, self._addend + self._next
        return result

>>> fibs = FibIter()
>>> [next(fibs) for _ in range(10)]
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]

"Please don't reference these directly. They may change."

This naming convention is not enforced, but is typically respected.

A programmer who designs and maintains a public module may change internal-use names.
Attributes for Internal Use

An attribute name that starts with one underscore is not meant to be referenced externally.

class FibIter:
    """An iterator over Fibonacci numbers."""
    def __init__(self):
        self._next = 0
        self._addend = 1

    def __next__(self):
        result = self._next
        self._addend, self._next = self._next, self._addend + self._next
        return result

>>> fibs = FibIter()
>>> [next(fibs) for _ in range(10)]
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]

"Please don't reference these directly. They may change."

This naming convention is not enforced, but is typically respected

A programmer who designs and maintains a public module may change internal-use names

Starting a name with two underscores enforces restricted access from outside the class
Names in Local Scope

A name bound in a local frame is not accessible to other environments, except those that extend the frame
Names in Local Scope

A name bound in a local frame is not accessible to other environments, except those that extend the frame

def fib_generator():
    """A generator function for Fibonacci numbers."

    >>> fibs = fib_generator()
    >>> [next(fibs) for _ in range(10)]
    [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
    """

    yield 0
    previous, current = 0, 1
    while True:
        yield current
        previous, current = current, previous + current
Names in Local Scope

A name bound in a local frame is not accessible to other environments, except those that extend the frame.

def fib_generator():
    """A generator function for Fibonacci numbers."

    >>> fibs = fib_generator()
    >>> [next(fibs) for _ in range(10)]
    [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
    """
    yield 0
    previous, current = 0, 1
    while True:
        yield current
        previous, current = current, previous + current

There is no way to access values bound to "previous" and "current" externally.
Singleton Objects
Singleton Objects

A singleton class is a class that only ever has one instance
Singleton Objects

A singleton class is a class that only ever has one instance

NoneType, the class of None, is a singleton class; None is its only instance
Singleton Objects

A singleton class is a class that only ever has one instance

NoneType, the class of None, is a singleton class; None is its only instance

For user-defined singletons, some programmers re-bind the class name to the instance
Singleton Objects

A singleton class is a class that only ever has one instance

NoneType, the class of None, is a singleton class; None is its only instance

For user-defined singletons, some programmers re-bind the class name to the instance

```python
class empty_iterator:
    """An iterator over no values."""
    def __next__(self):
        raise StopIteration
    empty_iterator = empty_iterator()
```
Singleton Objects

A singleton class is a class that only ever has one instance

NoneType, the class of None, is a singleton class; None is its only instance

For user-defined singletons, some programmers re-bind the class name to the instance

class empty_iterator:
    """An iterator over no values."""
    def __next__(self):
        raise StopIteration
    empty_iterator = empty_iterator()
Singleton Objects

A singleton class is a class that only ever has one instance

`NoneType`, the class of `None`, is a singleton class; `None` is its only instance

For user-defined singletons, some programmers re-bind the class name to the instance

```python
class empty_iterator:
    """An iterator over no values."""
    def __next__(self):
        raise StopIteration
empty_iterator = empty_iterator()
```

The instance  The class
Stream Implementation
Stream Implementation
Stream Implementation

A stream is a linked list with an *explicit* first element and a rest-of-the-list that is computed lazily
Stream Implementation

A stream is a linked list with an *explicit* first element and a rest-of-the-list that is computed lazily

```python
class Stream:
    """A lazily computed linked list."""
```
Stream Implementation

A stream is a linked list with an explicit first element and a rest-of-the-list that is computed lazily

```python
class Stream:
    """A lazily computed linked list."""
    class empty:
        def __repr__(self):
            return 'Stream.empty'
    empty = empty()
```
A stream is a linked list with an explicit first element and a rest-of-the-list that is computed lazily

```python
class Stream:
    """A lazily computed linked list."""

class empty:
    def __repr__(self):
        return 'Stream.empty'
empty = empty()

def __init__(self, first, compute_rest=lambda: Stream.empty):
    assert callable(compute_rest), 'compute_rest must be callable.'
    self.first = first
    self._compute_rest = compute_rest
```
Stream Implementation

A stream is a linked list with an explicit first element and a rest-of-the-list that is computed lazily

class Stream:
    """A lazily computed linked list."""
    class empty:
        def __repr__(self):
            return 'Stream.empty'
    empty = empty()

def __init__(self, first, compute_rest=lambda: Stream.empty):
    assert callable(compute_rest), 'compute_rest must be callable.'
    self.first = first
    self._compute_rest = compute_rest

@property
def rest(self):
    """Return the rest of the stream, computing it if necessary."""
    if self._compute_rest is not None:
        self._rest = self._compute_rest()
        self._compute_rest = None
    return self._rest
Declarative Languages
Database Management Systems
Database Management Systems

Database management systems (DBMS) are important, heavily used, and interesting!
Database Management Systems

Database management systems (DBMS) are important, heavily used, and interesting!

A table is a collection of records, which are rows that have a value for each column.
Database Management Systems

Database management systems (DBMS) are important, heavily used, and interesting!

A table is a collection of records, which are rows that have a value for each column.

<table>
<thead>
<tr>
<th>Latitude</th>
<th>Longitude</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>38</td>
<td>122</td>
<td>Berkeley</td>
</tr>
<tr>
<td>42</td>
<td>71</td>
<td>Cambridge</td>
</tr>
<tr>
<td>45</td>
<td>93</td>
<td>Minneapolis</td>
</tr>
</tbody>
</table>
Database Management Systems

Database management systems (DBMS) are important, heavily used, and interesting!

A table is a collection of records, which are rows that have a value for each column.
Database Management Systems

Database management systems (DBMS) are important, heavily used, and interesting!

A table is a collection of records, which are rows that have a value for each column.
**Database Management Systems**

Database management systems (DBMS) are important, heavily used, and interesting!

A table is a collection of records, which are rows that have a value for each column.
Database Management Systems

Database management systems (DBMS) are important, heavily used, and interesting!

A table is a collection of records, which are rows that have a value for each column.

The Structured Query Language (SQL) is perhaps the most widely used programming language.
Database Management Systems

Database management systems (DBMS) are important, heavily used, and interesting!

A table is a collection of records, which are rows that have a value for each column.

The Structured Query Language (SQL) is perhaps the most widely used programming language. SQL is a declarative programming language.
Declarative Programming
Declarative Programming

In *declarative languages* such as SQL & Prolog:
Declarative Programming

In **declarative languages** such as SQL & Prolog:

- A "program" is a description of the desired result
Declarative Programming

In declarative languages such as SQL & Prolog:

• A "program" is a description of the desired result
• The interpreter figures out how to generate the result
Declarative Programming

In declarative languages such as SQL & Prolog:
• A "program" is a description of the desired result
• The interpreter figures out how to generate the result

In imperative languages such as Python & Scheme:
Declarative Programming

In **declarative languages** such as SQL & Prolog:
- A "program" is a description of the desired result
- The interpreter figures out how to generate the result

In **imperative languages** such as Python & Scheme:
- A "program" is a description of computational processes
Declarative Programming

In **declarative languages** such as SQL & Prolog:
• A "program" is a description of the desired result
• The interpreter figures out how to generate the result

In **imperative languages** such as Python & Scheme:
• A "program" is a description of computational processes
• The interpreter carries out execution/evaluation rules
Declarative Programming

In declarative languages such as SQL & Prolog:
• A "program" is a description of the desired result
• The interpreter figures out how to generate the result

In imperative languages such as Python & Scheme:
• A "program" is a description of computational processes
• The interpreter carries out execution/evaluation rules

create table cities as
Declarative Programming

In **declarative languages** such as SQL & Prolog:
- A "program" is a description of the desired result
- The interpreter figures out how to generate the result

In **imperative languages** such as Python & Scheme:
- A "program" is a description of computational processes
- The interpreter carries out execution/evaluation rules

```sql
create table cities as

select 38 as latitude, 122 as longitude, "Berkeley" as name union
```

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Latitude</td>
<td>Longitude</td>
<td>Name</td>
</tr>
<tr>
<td>38</td>
<td>122</td>
<td>Berkeley</td>
</tr>
</tbody>
</table>
Declarative Programming

In declarative languages such as SQL & Prolog:
• A "program" is a description of the desired result
• The interpreter figures out how to generate the result

In imperative languages such as Python & Scheme:
• A "program" is a description of computational processes
• The interpreter carries out execution/evaluation rules

```
create table cities as
  select 38 as latitude, 122 as longitude, "Berkeley" as name union
select 42, 71, "Cambridge" union
```

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Latitude</td>
<td>Longitude</td>
<td>Name</td>
</tr>
<tr>
<td>38</td>
<td>122</td>
<td>Berkeley</td>
</tr>
<tr>
<td>42</td>
<td>71</td>
<td>Cambridge</td>
</tr>
</tbody>
</table>
Declarative Programming

In **declarative languages** such as SQL & Prolog:
  • A "program" is a description of the desired result
  • The interpreter figures out how to generate the result

In **imperative languages** such as Python & Scheme:
  • A "program" is a description of computational processes
  • The interpreter carries out execution/evaluation rules

```sql
create table cities as

    select 38 as latitude, 122 as longitude, "Berkeley" as name union
    select 42, 71, "Cambridge" union
    select 45, 93, "Minneapolis";
```

<table>
<thead>
<tr>
<th>Latitude</th>
<th>Longitude</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>38</td>
<td>122</td>
<td>Berkeley</td>
</tr>
<tr>
<td>42</td>
<td>71</td>
<td>Cambridge</td>
</tr>
<tr>
<td>45</td>
<td>93</td>
<td>Minneapolis</td>
</tr>
</tbody>
</table>
Declarative Programming

In **declarative languages** such as SQL & Prolog:

- A "program" is a description of the desired result
- The interpreter figures out how to generate the result

In **imperative languages** such as Python & Scheme:

- A "program" is a description of computational processes
- The interpreter carries out execution/evaluation rules

```sql
create table cities as
    select 38 as latitude, 122 as longitude, "Berkeley" as name union
    select 42, 71, "Cambridge" union
    select 45, 93, "Minneapolis";

select "west coast" as region, name from cities where longitude >= 115 union
select "other", name from cities where longitude < 115;
```

<table>
<thead>
<tr>
<th></th>
<th>Latitude</th>
<th>Longitude</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>38</td>
<td>122</td>
<td></td>
<td>Berkeley</td>
</tr>
<tr>
<td>42</td>
<td>71</td>
<td></td>
<td>Cambridge</td>
</tr>
<tr>
<td>45</td>
<td>93</td>
<td></td>
<td>Minneapolis</td>
</tr>
</tbody>
</table>
Declarative Programming

In **declarative languages** such as SQL & Prolog:

- A "program" is a description of the desired result
- The interpreter figures out how to generate the result

In **imperative languages** such as Python & Scheme:

- A "program" is a description of computational processes
- The interpreter carries out execution/evaluation rules

```sql
create table cities as
    select 38 as latitude, 122 as longitude, "Berkeley" as name union
    select 42, 71, "Cambridge" union
    select 45, 93, "Minneapolis";

select "west coast" as region, name from cities where longitude >= 115 union
select "other", name from cities where longitude < 115;
```

**Cities:**

<table>
<thead>
<tr>
<th>Latitude</th>
<th>Longitude</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>38</td>
<td>122</td>
<td>Berkeley</td>
</tr>
<tr>
<td>42</td>
<td>71</td>
<td>Cambridge</td>
</tr>
<tr>
<td>45</td>
<td>93</td>
<td>Minneapolis</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Region</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>west coast</td>
<td>Berkeley</td>
</tr>
<tr>
<td>other</td>
<td>Minneapolis</td>
</tr>
<tr>
<td>other</td>
<td>Cambridge</td>
</tr>
</tbody>
</table>
Structured Query Language (SQL)
SQL Overview
SQL Overview

The SQL language is an ANSI and ISO standard, but DBMS's implement custom variants.
The SQL language is an ANSI and ISO standard, but DBMS's implement custom variants

- A `select` statement creates a new table, either from scratch or by projecting a table
SQL Overview

The SQL language is an ANSI and ISO standard, but DBMS's implement custom variants

- A `select` statement creates a new table, either from scratch or by projecting a table
- A `create table` statement gives a global name to a table
SQL Overview

The SQL language is an ANSI and ISO standard, but DBMS's implement custom variants

- A `select` statement creates a new table, either from scratch or by projecting a table
- A `create table` statement gives a global name to a table
- Lots of other statements exist: `analyze`, `delete`, `explain`, `insert`, `replace`, `update`, etc.
SQL Overview

The SQL language is an ANSI and ISO standard, but DBMS's implement custom variants.

- A **select** statement creates a new table, either from scratch or by projecting a table.
- A **create table** statement gives a global name to a table.
- Lots of other statements exist: **analyze**, **delete**, **explain**, **insert**, **replace**, **update**, etc.
- Most of the important action is in the **select** statement.
SQL Overview

The SQL language is an ANSI and ISO standard, but DBMS's implement custom variants

• A **select** statement creates a new table, either from scratch or by projecting a table
• A **create table** statement gives a global name to a table
• Lots of other statements exist: **analyze**, **delete**, **explain**, **insert**, **replace**, **update**, etc.
• Most of the important action is in the **select** statement
• The code for executing **select** statements fits on a single sheet of paper (next lecture)
SQL Overview

The SQL language is an ANSI and ISO standard, but DBMS's implement custom variants

- A `select` statement creates a new table, either from scratch or by projecting a table
- A `create table` statement gives a global name to a table
- Lots of other statements exist: `analyze`, `delete`, `explain`, `insert`, `replace`, `update`, etc.
- Most of the important action is in the `select` statement
- The code for executing `select` statements fits on a single sheet of paper (next lecture)

Today's theme:
The SQL language is an ANSI and ISO standard, but DBMS's implement custom variants

* A `select` statement creates a new table, either from scratch or by projecting a table
* A `create table` statement gives a global name to a table
* Lots of other statements exist: `analyze`, `delete`, `explain`, `insert`, `replace`, `update`, etc.
* Most of the important action is in the `select` statement
* The code for executing `select` statements fits on a single sheet of paper (next lecture)

Today's theme:
Getting Started with SQL

Install sqlite (version 3.8.3 or later): http://sqlite.org/download.html

Use sqlite online: http://kripken.github.io/sql.js/GUI/

Use the SQL example from the textbook: http://composingprograms.com/examples/sql/sql.zip
Selecting Value Literals

A `select` statement always includes a comma-separated list of column descriptions.
Selecting Value Literals

A `select` statement always includes a comma-separated list of column descriptions.

A column description is an expression, optionally followed by `as` and a column name.
Selecting Value Literals

A `select` statement always includes a comma-separated list of column descriptions.

A column description is an expression, optionally followed by `as` and a column name.

```
select [expression] as [name]
```
Selecting Value Literals

A `select` statement always includes a comma-separated list of column descriptions. A column description is an expression, optionally followed by `as` and a column name:

```
select [expression] as [name], [expression] as [name]
```
Selecting Value Literals

A `select` statement always includes a comma-separated list of column descriptions. A column description is an expression, optionally followed by `as` and a column name:

```
select [expression] as [name], [expression] as [name], ...
```
Selecting Value Literals

A **select** statement always includes a comma-separated list of column descriptions. A column description is an expression, optionally followed by **as** and a column name.

```sql
select [expression] as [name], [expression] as [name];
```
Selecting Value Literals

A `select` statement always includes a comma-separated list of column descriptions. A column description is an expression, optionally followed by `as` and a column name:

```
select [expression] as [name], [expression] as [name];
```

Selecting literals creates a one-row table.
Selecting Value Literals

A `select` statement always includes a comma-separated list of column descriptions

A column description is an expression, optionally followed by `as` and a column name

```
select [expression] as [name], [expression] as [name];
```

Selecting literals creates a one-row table

The union of two select statements is a table containing the rows of both of their results
Selecting Value Literals

A `select` statement always includes a comma-separated list of column descriptions.

A column description is an expression, optionally followed by `as` and a column name:

```
select [expression] as [name], [expression] as [name];
```

Selecting literals creates a one-row table:

```
select "abraham" as parent, "barack" as child;
```

The union of two select statements is a table containing the rows of both of their results.
Selecting Value Literals

A `select` statement always includes a comma-separated list of column descriptions.

A column description is an expression, optionally followed by `as` and a column name.

```plaintext
select [expression] as [name], [expression] as [name];
```

Selecting literals creates a one-row table.

The union of two select statements is a table containing the rows of both of their results.

```plaintext
select "abraham" as parent, "barack" as child union
```
Selecting Value Literals

A `select` statement always includes a comma-separated list of column descriptions.

A column description is an expression, optionally followed by `as` and a column name:

```
select [expression] as [name], [expression] as [name];
```

Selecting literals creates a one-row table.

The union of two select statements is a table containing the rows of both of their results.

```
select "abraham" as parent, "barack" as child union
select "abraham", "clinton" union
```
Selecting Value Literals

A `select` statement always includes a comma-separated list of column descriptions. A column description is an expression, optionally followed by `as` and a column name:

```
select [expression] as [name], [expression] as [name];
```

Selecting literals creates a one-row table.

The union of two select statements is a table containing the rows of both of their results:

```
select "abraham" as parent, "barack" as child union
select "abraham", "clinton" union
select "delano", "herbert" union
```
Selecting Value Literals

A **select** statement always includes a comma-separated list of column descriptions. A column description is an expression, optionally followed by **as** and a column name.

```sql
select [expression] as [name], [expression] as [name];
```

Selecting literals creates a one-row table.

The union of two select statements is a table containing the rows of both of their results:

```sql
select "abraham" as parent, "barack" as child union
select "abraham", "clinton" union
select "delano", "herbert" union
select "fillmore", "abraham" union
```
Selecting Value Literals

A `select` statement always includes a comma-separated list of column descriptions.

A column description is an expression, optionally followed by `as` and a column name.

```
select [expression] as [name], [expression] as [name];
```

Selecting literals creates a one-row table.

The union of two select statements is a table containing the rows of both of their results.

```
select "abraham" as parent, "barack" as child union
select "abraham", "clinton" union
select "delano", "herbert" union
select "fillmore", "abraham" union
select "fillmore", "delano" union
```
Selecting Value Literals

A `select` statement always includes a comma-separated list of column descriptions.

A column description is an expression, optionally followed by `as` and a column name:

```
select [expression] as [name], [expression] as [name];
```

Selecting literals creates a one-row table.

The union of two select statements is a table containing the rows of both of their results:

```
select "abraham" as parent, "barack" as child union
select "abraham", "clinton" union
select "delano", "herbert" union
select "fillmore", "abraham" union
select "fillmore", "delano" union
select "fillmore", "grover" union
```
Selecting Value Literals

A `select` statement always includes a comma-separated list of column descriptions. A column description is an expression, optionally followed by `as` and a column name:

\[ \text{select} \ [\text{expression}] \ as \ [\text{name}], \ [\text{expression}] \ as \ [\text{name}]; \]

Selecting literals creates a one-row table.

The union of two select statements is a table containing the rows of both of their results:

- `select "abraham" as parent, "barack" as child union`
- `select "abraham", "clinton" union`
- `select "delano", "herbert" union`
- `select "fillmore", "abraham" union`
- `select "fillmore", "delano" union`
- `select "fillmore", "grover" union`
- `select "eisenhower", "fillmore";`
Naming Tables

```sql
select "abraham" as parent, "barack" as child union
select "abraham", "clinton" union
select "delano", "herbert" union
select "fillmore", "abraham" union
select "fillmore", "delano" union
select "fillmore", "grover" union
select "eisenhower", "fillmore";
```
Naming Tables

SQL is often used as an interactive language

```
select "abraham" as parent, "barack" as child union
select "abraham" , "clinton" union
select "delano" , "herbert" union
select "fillmore" , "abraham" union
select "fillmore" , "delano" union
select "fillmore" , "grover" union
select "eisenhower" , "fillmore";
```
Naming Tables

SQL is often used as an interactive language.

The result of a `select` statement is displayed to the user, but not stored.

```sql
select "abraham" as parent, "barack" as child union
select "abraham", "clinton" union
select "delano", "herbert" union
select "fillmore", "abraham" union
select "fillmore", "delano" union
select "fillmore", "grover" union
select "eisenhower", "fillmore";
```
Naming Tables

SQL is often used as an interactive language.

The result of a **select** statement is displayed to the user, but not stored.

A **create table** statement gives the result a name:

```sql
select "abraham" as parent, "barack" as child union
select "abraham", "clinton" union
select "delano", "herbert" union
select "fillmore", "abraham" union
select "fillmore", "delano" union
select "fillmore", "grover" union
select "eisenhower", "fillmore";
```
Naming Tables

SQL is often used as an interactive language
The result of a `select` statement is displayed to the user, but not stored
A `create table` statement gives the result a name

```
cREATE TABLE [name] AS [select statement];
```

```
SELECT "abraham" AS parent, "barack" AS child UNION
SELECT "abraham", "clinton" UNION
SELECT "delano", "herbert" UNION
SELECT "fillmore", "abraham" UNION
SELECT "fillmore", "delano" UNION
SELECT "fillmore", "grover" UNION
SELECT "eisenhower", "fillmore";
```
Naming Tables

SQL is often used as an interactive language.

The result of a `select` statement is displayed to the user, but not stored.

A `create table` statement gives the result a name.

```
create table [name] as [select statement];
```

create table parents as
select "abraham" as parent, "barack" as child union
select "abraham", "clinton" union
select "delano", "herbert" union
select "fillmore", "abraham" union
select "fillmore", "delano" union
select "fillmore", "grover" union
select "eisenhower", "fillmore";

```
Eisenhower
  \  /  Fillmore
 /    \  \
Abraham  Delano  Grover
  /   \   \
Barack  Clinton  Herbert
```
Naming Tables

SQL is often used as an interactive language.
The result of a `select` statement is displayed to the user, but not stored.
A `create table` statement gives the result a name:

```sql
create table [name] as [select statement];
```

```sql
create table parents as
  select "abraham" as parent, "barack" as child union
  select "abraham", "clinton" union
  select "delano", "herbert" union
  select "fillmore", "abraham" union
  select "fillmore", "delano" union
  select "fillmore", "grover" union
  select "eisenhower", "fillmore";
```

![Diagram of family tree]
Naming Tables

SQL is often used as an interactive language

The result of a `select` statement is displayed to the user, but not stored

A `create table` statement gives the result a name

```sql
CREATE TABLE [name] AS [select statement];
```

```sql
cREATE TABLE parents AS
  SELECT "abraham" AS parent, "barack" AS child UNION
  SELECT "abraham", "clinton" UNION
  SELECT "delano", "herbert" UNION
  SELECT "fillmore", "abraham" UNION
  SELECT "fillmore", "delano" UNION
  SELECT "fillmore", "grover" UNION
  SELECT "eisenhower", "fillmore";
```

<table>
<thead>
<tr>
<th>Parent</th>
<th>Child</th>
</tr>
</thead>
<tbody>
<tr>
<td>abraham</td>
<td>barack</td>
</tr>
<tr>
<td>abraham</td>
<td>clinton</td>
</tr>
<tr>
<td>delano</td>
<td>herbert</td>
</tr>
<tr>
<td>fillmore</td>
<td>abraham</td>
</tr>
<tr>
<td>fillmore</td>
<td>delano</td>
</tr>
<tr>
<td>fillmore</td>
<td>grover</td>
</tr>
<tr>
<td>eisenhower</td>
<td>fillmore</td>
</tr>
</tbody>
</table>
Projecting Tables
Select Statements Project Existing Tables
Select Statements Project Existing Tables

A `select` statement can specify an input table using a `from` clause.
Select Statements Project Existing Tables

A `select` statement can specify an input table using a `from` clause

```
select [expression] as [name], [expression] as [name], ... ;
```
Select Statements Project Existing Tables

A `select` statement can specify an input table using a `from` clause

```sql
select [expression] as [name], [expression] as [name], ...;
select [columns]  
```
Select Statements Project Existing Tables

A `select` statement can specify an input table using a `from` clause

```
select [expression] as [name], [expression] as [name], ...;
select [columns] from [table] ;
```
Select Statements Project Existing Tables

A `select` statement can specify an input table using a `from` clause.
A subset of the rows of the input table can be selected using a `where` clause.

```sql
select [expression] as [name], [expression] as [name], ...;
select [columns] from [table];
```
Select Statements Project Existing Tables

A `select` statement can specify an input table using a `from` clause.

A subset of the rows of the input table can be selected using a `where` clause.

```
select [expression] as [name], [expression] as [name], ...;

select [columns] from [table] where [condition];
```
Select Statements Project Existing Tables

A `select` statement can specify an input table using a `from` clause.
A subset of the rows of the input table can be selected using a `where` clause.
An ordering over the remaining rows can be declared using an `order by` clause.

```
select [expression] as [name], [expression] as [name], ...;
select [columns] from [table] where [condition] ;
```
Select Statements Project Existing Tables

A **select** statement can specify an input table using a **from** clause.

A subset of the rows of the input table can be selected using a **where** clause.

An ordering over the remaining rows can be declared using an **order by** clause.

```
select [expression] as [name], [expression] as [name], ...;
```

```
select [columns] from [table] where [condition] order by [order];
```
Select Statements Project Existing Tables

A `select` statement can specify an input table using a `from` clause.

A subset of the rows of the input table can be selected using a `where` clause.

An ordering over the remaining rows can be declared using an `order by` clause.

Column descriptions determine how each input row is projected to a result row.

```
select [expression] as [name], [expression] as [name], ...;
```

```
select [columns] from [table] where [condition] order by [order];
```
A `select` statement can specify an input table using a `from` clause.

A subset of the rows of the input table can be selected using a `where` clause.

An ordering over the remaining rows can be declared using an `order by` clause.

Column descriptions determine how each input row is projected to a result row.

```
select [expression] as [name], [expression] as [name], ...;
```

```
select [columns] from [table] where [condition] order by [order];
```
Select Statements Project Existing Tables

A `select` statement can specify an input table using a `from` clause
A subset of the rows of the input table can be selected using a `where` clause
An ordering over the remaining rows can be declared using an `order by` clause
Column descriptions determine how each input row is projected to a result row

```sql
select {[expression] as [name], [expression] as [name], ...};
select [columns] from [table] where [condition] order by [order];
select child from parents where parent = "abraham";
```
Select Statements Project Existing Tables

A `select` statement can specify an input table using a `from` clause.

A subset of the rows of the input table can be selected using a `where` clause.

An ordering over the remaining rows can be declared using an `order by` clause.

Column descriptions determine how each input row is projected to a result row.

```
select [expression] as [name], [expression] as [name], ...

select [columns] from [table] where [condition] order by [order];

select child from parents where parent = "abraham";
```
Select Statements Project Existing Tables

A `select` statement can specify an input table using a `from` clause.
A subset of the rows of the input table can be selected using a `where` clause.
An ordering over the remaining rows can be declared using an `order by` clause.
Column descriptions determine how each input row is projected to a result row.

```
select [expression] as [name], [expression] as [name], ...;
select [columns] from [table] where [condition] order by [order];
select child from parents where parent = "abraham";
```
Select Statements Project Existing Tables

A select statement can specify an input table using a from clause.

A subset of the rows of the input table can be selected using a where clause.

An ordering over the remaining rows can be declared using an order by clause.

Column descriptions determine how each input row is projected to a result row.

```sql
select [expression] as [name], [expression] as [name], ...;
```

```sql
select [columns] from [table] where [condition] order by [order];
```

```sql
select child from parents where parent = "abraham";
```

```sql
select parent from parents where parent > child;
```

<table>
<thead>
<tr>
<th>Child</th>
</tr>
</thead>
<tbody>
<tr>
<td>barack</td>
</tr>
<tr>
<td>clinton</td>
</tr>
</tbody>
</table>

Diagram:

```
  Eisenhower -> Fillmore
    /     
  Abraham  Delano  Grover
   |       |
  Barack  Clinton  Herbert
```

18
Select Statements Project Existing Tables

A `select` statement can specify an input table using a `from` clause.
A subset of the rows of the input table can be selected using a `where` clause.
An ordering over the remaining rows can be declared using an `order by` clause.
Column descriptions determine how each input row is projected to a result row.

```
select [expression] as [name], [expression] as [name], ...;
select [columns] from [table] where [condition] order by [order];
select child from parents where parent = "abraham";
select parent from parents where parent > child;
```
Select Statements Project Existing Tables

A **select** statement can specify an input table using a **from** clause

A subset of the rows of the input table can be selected using a **where** clause

An ordering over the remaining rows can be declared using an **order by** clause

Column descriptions determine how each input row is projected to a result row

```sql
select [expression] as [name], [expression] as [name], ...;
select [columns] from [table] where [condition] order by [order];
select child from parents where parent = "abraham";
select parent from parents where parent > child;
```

<table>
<thead>
<tr>
<th>Child</th>
<th>Parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>barack</td>
<td>fillmore</td>
</tr>
</tbody>
</table>
| clinton    | fillmore    | (Demo)