Declarative Programming
Announcements
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.

<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.

<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>

A *table* has columns and rows.

A *row* has a value for each column.

A column has a name and a type.

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

<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>

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

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

<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>
</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
```

**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>
</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
    select 45, 93, "Minneapolis";
```

<table>
<thead>
<tr>
<th>Cities:</th>
</tr>
</thead>
<tbody>
<tr>
<td>latitude</td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td>38</td>
</tr>
<tr>
<td>42</td>
</tr>
<tr>
<td>45</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
  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>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

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

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

*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
Getting Started with SQL

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

Use sqlite online: code.cs61a.org/sql
Selecting Value Literals
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.

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

```
select "delano" as parent, "herbert" as child;
```
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 "delano" as parent, "herbert" 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 "delano" as parent, "herbert" as child union
select "abraham", "barack" union
```

```
Delano -> Herbert
Barack -> Abraham
```
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 "delano" as parent, "herbert" as child union
select "abraham", "barack" 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 "delano" as parent, "herbert" as child union
select "abraham", "barack" union
select "abraham", "clinton" union
select "fillmore", "abraham" union
```

```
Delano
   \--- Fillmore
      \-- Abraham
         \-- Barack
            \-- Clinton
               \-- Herbert
```
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 "delano" as parent, "herbert" as child union
select "abraham", "barack" union
select "abraham", "clinton" union
select "fillmore", "abraham" union
select "fillmore", "delano" union
```

```
Delano

         Fillmore

      Abraham
    Barack

            Clinton

      Herbert

         Delano
```
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 "delano" as parent, "herbert" as child union
select "abraham", "barack" union
select "abraham", "clinton" 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:

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

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

```sql
select "delano" as parent, "herbert" as child union
select "abraham", "barack" union
select "abraham", "clinton" 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

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

```
select "delano" as parent, "herbert" as child union
select "abraham", "barack" union
select "abraham", "clinton" 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 "delano" as parent, "herbert" as child union
select "abraham" , "barack" union
select "abraham" , "clinton" 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 "delano" as parent, "herbert" as child union
select "abraham", "barack" union
select "abraham", "clinton" 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 "delano" as parent, "herbert" as child union
    select "abraham", "barack" union
    select "abraham", "clinton" 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
create table [name] as [select statement];
```

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

Parents:

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

```
select [expression] as [name], [expression] as [name], ... ;
```
A `select` statement can specify an input table using a `from` clause.

- `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.

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

```sql
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] ;
```
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.

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

```sql
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.

```sql
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];
```
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 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";
```
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.

```plaintext
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 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;
```
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;
```

<table>
<thead>
<tr>
<th>Child</th>
<th>Parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>barack</td>
<td>fillmore</td>
</tr>
<tr>
<td>clinton</td>
<td>fillmore</td>
</tr>
</tbody>
</table>
Arithmetic
Arithmetic in Select Expressions
Arithmetic in Select Expressions

In a select expression, column names evaluate to row values

Arithmetic expressions can combine row values and constants
Arithmetic in Select Expressions

In a select expression, column names evaluate to row values.

Arithmetic expressions can combine row values and constants.
Arithmetic in Select Expressions

In a select expression, column names evaluate to row values.

Arithmetic expressions can combine row values and constants.

```sql
create table lift as
    select 101 as chair, 2 as single, 2 as couple union
select 102 , 0 , 3 union
select 103 , 4 , 1;
```
Arithmetic in Select Expressions

In a select expression, column names evaluate to row values.

Arithmetic expressions can combine row values and constants.

```sql
create table lift as
    select 101 as chair, 2 as single, 2 as couple union
    select 102, 0, 3 union
    select 103, 4, 1;
```

101
Arithmetic in Select Expressions

In a select expression, column names evaluate to row values. Arithmetic expressions can combine row values and constants.

```sql
create table lift as
  select 101 as chair, 2 as single, 2 as couple union
  select 102 , 0 , 3 union
  select 103 , 4 , 1;
```
Arithmetic in Select Expressions

In a select expression, column names evaluate to row values

Arithmetic expressions can combine row values and constants

```sql
create table lift as
    select 101 as chair, 2 as single, 2 as couple union
select 102, 0, 3 union
select 103, 4, 1;
```
Arithmetic in Select Expressions

In a select expression, column names evaluate to row values
Arithmetic expressions can combine row values and constants

```sql
create table lift as
    select 101 as chair, 2 as single, 2 as couple union
    select 102 , 0 , 3 union
    select 103 , 4 , 1;

select chair, single + 2 * couple as total from lift;
```
Arithmetic in Select Expressions

In a select expression, column names evaluate to row values.
Arithmetic expressions can combine row values and constants.

```sql
create table lift as
    select 101 as chair, 2 as single, 2 as couple union
    select 102     , 0     , 3 union
    select 103     , 4     , 1;
```

```sql
select chair, single + 2 * couple as total from lift;
```

<table>
<thead>
<tr>
<th>chair</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>6</td>
</tr>
<tr>
<td>102</td>
<td>6</td>
</tr>
<tr>
<td>103</td>
<td>6</td>
</tr>
</tbody>
</table>
Discussion Question

Given the table `ints` that describes how to sum powers of 2 to form various integers

```sql
create table ints as
    select "zero" as word, 0 as one, 0 as two, 0 as four, 0 as eight union
select "one" , 1 , 0 , 0 , 0 union
select "two" , 0 , 2 , 0 , 0 union
select "three" , 1 , 2 , 0 , 0 union
select "four" , 0 , 0 , 4 , 0 union
select "five" , 1 , 0 , 4 , 0 union
select "six" , 0 , 2 , 4 , 0 union
select "seven" , 1 , 2 , 4 , 0 union
select "eight" , 0 , 0 , 0 , 8 union
select "nine" , 1 , 0 , 0 , 8;
```
**Discussion Question**

Given the table **ints** that describes how to sum powers of 2 to form various integers

```sql
create table ints as
    select "zero" as word, 0 as one, 0 as two, 0 as four, 0 as eight union
    select "one", 1, 0, 0, 0 union
    select "two", 0, 2, 0, 0 union
    select "three", 1, 2, 0, 0 union
    select "four", 0, 0, 4, 0 union
    select "five", 1, 0, 4, 0 union
    select "six", 0, 2, 4, 0 union
    select "seven", 1, 2, 4, 0 union
    select "eight", 0, 0, 0, 8 union
    select "nine", 1, 0, 0, 8;
```

(A) Write a select statement for a two-column table of the **word** and **value** for each integer
Discussion Question

Given the table `ints` that describes how to sum powers of 2 to form various integers

```sql
create table ints as
    select "zero" as word, 0 as one, 0 as two, 0 as four, 0 as eight union
    select "one", 1, 0, 0, 0 union
    select "two", 0, 2, 0, 0 union
    select "three", 1, 2, 0, 0 union
    select "four", 0, 0, 4, 0 union
    select "five", 1, 0, 4, 0 union
    select "six", 0, 2, 4, 0 union
    select "seven", 1, 2, 4, 0 union
    select "eight", 0, 0, 0, 8 union
    select "nine", 1, 0, 0, 8;
```

(A) Write a select statement for a two-column table of the word and value for each integer

<table>
<thead>
<tr>
<th>word</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>zero</td>
<td>0</td>
</tr>
<tr>
<td>one</td>
<td>1</td>
</tr>
<tr>
<td>two</td>
<td>2</td>
</tr>
<tr>
<td>three</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Discussion Question

Given the table `ints` that describes how to sum powers of 2 to form various integers

```sql
CREATE TABLE ints AS
SELECT "zero" AS word, 0 AS one, 0 AS two, 0 AS four, 0 AS eight UNION
SELECT "one" , 1 , 0 , 0 , 0 UNION
SELECT "two" , 0 , 2 , 0 , 0 UNION
SELECT "three" , 1 , 2 , 0 , 0 UNION
SELECT "four" , 0 , 0 , 4 , 0 UNION
SELECT "five" , 1 , 0 , 4 , 0 UNION
SELECT "six" , 0 , 2 , 4 , 0 UNION
SELECT "seven" , 1 , 2 , 4 , 0 UNION
SELECT "eight" , 0 , 0 , 0 , 8 UNION
SELECT "nine" , 1 , 0 , 0 , 8;
```

(A) Write a select statement for a two-column table of the `word` and `value` for each integer

<table>
<thead>
<tr>
<th>word</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>zero</td>
<td>0</td>
</tr>
<tr>
<td>one</td>
<td>1</td>
</tr>
<tr>
<td>two</td>
<td>2</td>
</tr>
<tr>
<td>three</td>
<td>3</td>
</tr>
</tbody>
</table>

(B) Write a select statement for the `word` names of the powers of two
Discussion Question

Given the table `ints` that describes how to sum powers of 2 to form various integers

```sql
create table ints as
    select "zero" as word, 0 as value, 0 as two, 0 as four, 0 as eight union
    select "one" , 1 , 0 , 0 , 0 union
    select "two" , 0 , 2 , 0 , 0 union
    select "three" , 1 , 2 , 0 , 0 union
    select "four" , 0 , 0 , 4 , 0 union
    select "five" , 1 , 0 , 4 , 0 union
    select "six" , 0 , 2 , 4 , 0 union
    select "seven" , 1 , 2 , 4 , 0 union
    select "eight" , 0 , 0 , 0 , 8 union
    select "nine" , 1 , 0 , 0 , 8;
```

(A) Write a select statement for a two-column table of the `word` and `value` for each integer

<table>
<thead>
<tr>
<th>word</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>zero</td>
<td>0</td>
</tr>
<tr>
<td>one</td>
<td>1</td>
</tr>
<tr>
<td>two</td>
<td>2</td>
</tr>
<tr>
<td>three</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

(B) Write a select statement for the `word` names of the powers of two

<table>
<thead>
<tr>
<th>word</th>
</tr>
</thead>
<tbody>
<tr>
<td>one</td>
</tr>
<tr>
<td>two</td>
</tr>
<tr>
<td>four</td>
</tr>
<tr>
<td>eight</td>
</tr>
</tbody>
</table>
Discussion Question

Given the table `ints` that describes how to sum powers of 2 to form various integers

```sql
create table ints as
    select "zero" as word, 0 as one, 0 as two, 0 as four, 0 as eight union
    select "one" , 1 , 0 , 0 , 0 union
    select "two" , 0 , 2 , 0 , 0 union
    select "three" , 1 , 2 , 0 , 0 union
    select "four" , 0 , 0 , 4 , 0 union
    select "five" , 1 , 0 , 4 , 0 union
    select "six" , 0 , 2 , 4 , 0 union
    select "seven" , 1 , 2 , 4 , 0 union
    select "eight" , 0 , 0 , 0 , 8 union
    select "nine" , 1 , 0 , 0 , 8;
```

(A) Write a select statement for a two-column table of the `word` and `value` for each integer

<table>
<thead>
<tr>
<th>word</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>zero</td>
<td>0</td>
</tr>
<tr>
<td>one</td>
<td>1</td>
</tr>
<tr>
<td>two</td>
<td>2</td>
</tr>
<tr>
<td>three</td>
<td>3</td>
</tr>
</tbody>
</table>
| ...   | ...   | (Demo)

(B) Write a select statement for the `word` names of the powers of two

<table>
<thead>
<tr>
<th>word</th>
</tr>
</thead>
<tbody>
<tr>
<td>one</td>
</tr>
<tr>
<td>two</td>
</tr>
<tr>
<td>four</td>
</tr>
<tr>
<td>eight</td>
</tr>
</tbody>
</table>

(Demo)