Declarative Languages
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

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>

<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)
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:*
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

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
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 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

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

```
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;
```

<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 \texttt{ints} that describes how to sum powers of 2 to form various integers

\begin{verbatim}
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   union
\end{verbatim}

(A) Write a select statement for a two-column table of the \textbf{word} and \textbf{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 \textbf{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>
Joining Tables
Reminder: John the Patriotic Dog Breeder

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";

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>
Joining Two Tables

Two tables \(A\) & \(B\) are joined by a comma to yield all combos of a row from \(A\) & a row from \(B\):

```
CREATE TABLE dogs AS
    SELECT "abraham" AS name, "long" AS fur UNION
    SELECT "barack" , "short" UNION
    SELECT "clinton" , "long" UNION
    SELECT "delano" , "long" UNION
    SELECT "eisenhower" , "short" UNION
    SELECT "fillmore" , "curly" UNION
    SELECT "grover" , "short" UNION
    SELECT "herbert" , "curly";
```

```
CREATE TABLE parents AS
    SELECT "abraham" AS parent, "barack" AS child UNION
    SELECT "abraham" , "clinton" UNION
    ...;
```

Select the parents of curly-furred dogs:

```
SELECT parent FROM parents, dogs
    WHERE child = name AND fur = "curly";
```

(Demo)
Aliases and Dot Expressions
Joining a Table with Itself

Two tables may share a column name; dot expressions and aliases disambiguate column values

```
SELECT [columns] FROM [table] WHERE [condition] ORDER BY [order];
```

[table] is a comma-separated list of table names with optional aliases

Select all pairs of siblings

```
SELECT a.child AS first, b.child AS second
FROM parents AS a, parents AS b
WHERE a.parent = b.parent AND a.child < b.child;
```

<table>
<thead>
<tr>
<th>First</th>
<th>Second</th>
</tr>
</thead>
<tbody>
<tr>
<td>barack</td>
<td>clinton</td>
</tr>
<tr>
<td>abraham</td>
<td>delano</td>
</tr>
<tr>
<td>abraham</td>
<td>grover</td>
</tr>
<tr>
<td>delano</td>
<td>grover</td>
</tr>
</tbody>
</table>
Example: Grandparents

Which select statement evaluates to all grandparent, grandchild pairs?

1. SELECT a.grandparent, b.child FROM parents AS a, parents AS b
   WHERE b.parent = a.child;

2. SELECT a.parent, b.child FROM parents AS a, parents AS b
   WHERE a.parent = b.child;

3. SELECT a.parent, b.child FROM parents AS a, parents AS b
   WHERE b.parent = a.child;

4. SELECT a.grandparent, b.child FROM parents AS a, parents AS b
   WHERE a.parent = b.child;

5. None of the above
Joining Multiple Tables

Multiple tables can be joined to yield all combinations of rows from each

```
CREATE TABLE grandparents AS
    SELECT a.parent AS grandog, b.child AS granpup
    FROM parents AS a, parents AS b
    WHERE b.parent = a.child;
```

Select all grandparents with the same fur as their grandchildren

```
SELECT grandog FROM grandparents, dogs AS c, dogs AS d
    WHERE grandog = c.name AND
        granpup = d.name AND
        c.fur = d.fur;
```
Example: Dog Triples
Write a SQL query that selects all possible combinations of three different dogs with the same fur and lists each triple in *inverse* alphabetical order.

```sql
CREATE TABLE dogs AS
    SELECT "abraham" AS name, "long" AS fur UNION
    SELECT "barack" , "short" UNION
    ...;

CREATE TABLE parents AS
    SELECT "abraham" AS parent, "barack" AS child UNION
    SELECT "abraham" , "clinton" UNION
    ...;

Expected output:

delano|clinton|abraham
grover|eisenhower|barack

(Demo)
Numerical Expressions
Numerical Expressions

Expressions can contain function calls and arithmetic operators

```
[expression] AS [name], [expression] AS [name], ...
```

```
SELECT [columns] FROM [table] WHERE [expression] ORDER BY [expression];
```

Combine values: +, −, *, /, %, and, or

Transform values: abs, round, not, −

Compare values: ‹, ‹=, ‡, ‡=, †, †=, =

(Demo)
String Expressions
String Expressions

String values can be combined to form longer strings

```sql
sqlite> SELECT "hello," || " world";
hello, world
```

Basic string manipulation is built into SQL, but differs from Python

```sql
sqlite> CREATE TABLE phrase AS SELECT "hello, world" AS s;
sqlite> SELECT substr(s, 4, 2) || substr(s, instr(s, " ")+1, 1) FROM phrase;
low
```

Strings can be used to represent structured values, but doing so is rarely a good idea

```sql
sqlite> CREATE TABLE lists AS SELECT "one" AS car, "two,three,four" AS cdr;
sqlite> SELECT substr(cdr, 1, instr(cdr, ",",)-1) AS cadr FROM lists;
two
```

(Demo)