Computational Structures in Data Science

UC Berkeley EECS
Lecturer Michael Ball

Lecture #23 & 24: Databases & SQL

April 20, 2020

https://cs88.org/

Computing In the News [Link]

- **NASA Using Red and Blue 3D Glasses to Drive Mars Rover While Working From Home** ***Gizmodo*** *Andrew Liszewski April 17, 2020*

  Planners at the National Aeronautics and Space Administration's Jet Propulsion Laboratory (JPL) are remotely piloting the Curiosity rover on Mars while working from home. Without access to JPL's powerful workstations and special three-dimensional (3D) goggles due to quarantine orders, the team must rely on red and blue 3D glasses. While antiquated by today's 3D standards, the cardboard glasses are essentially the same anaglyph 3D technology as the special goggles normally used by the team to plan the rover's movements and more accurately target its robotic arm and probes. The team successfully executed Curiosity's first mission planned outside of JPL's facilities just two days after relocating to home offices.

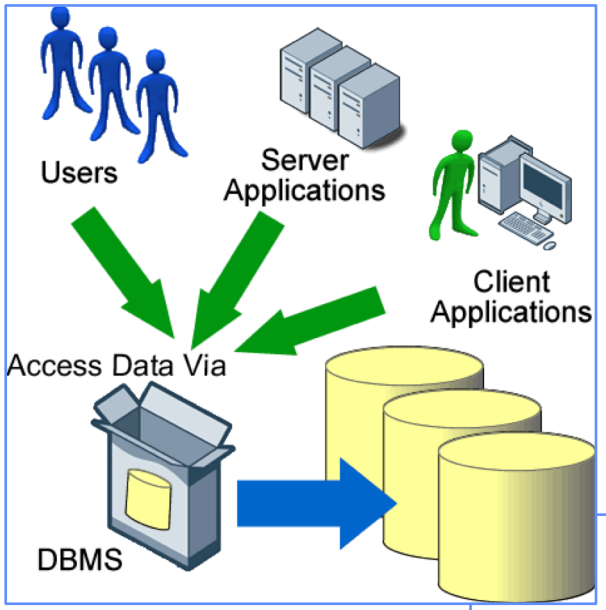
Why Databases?

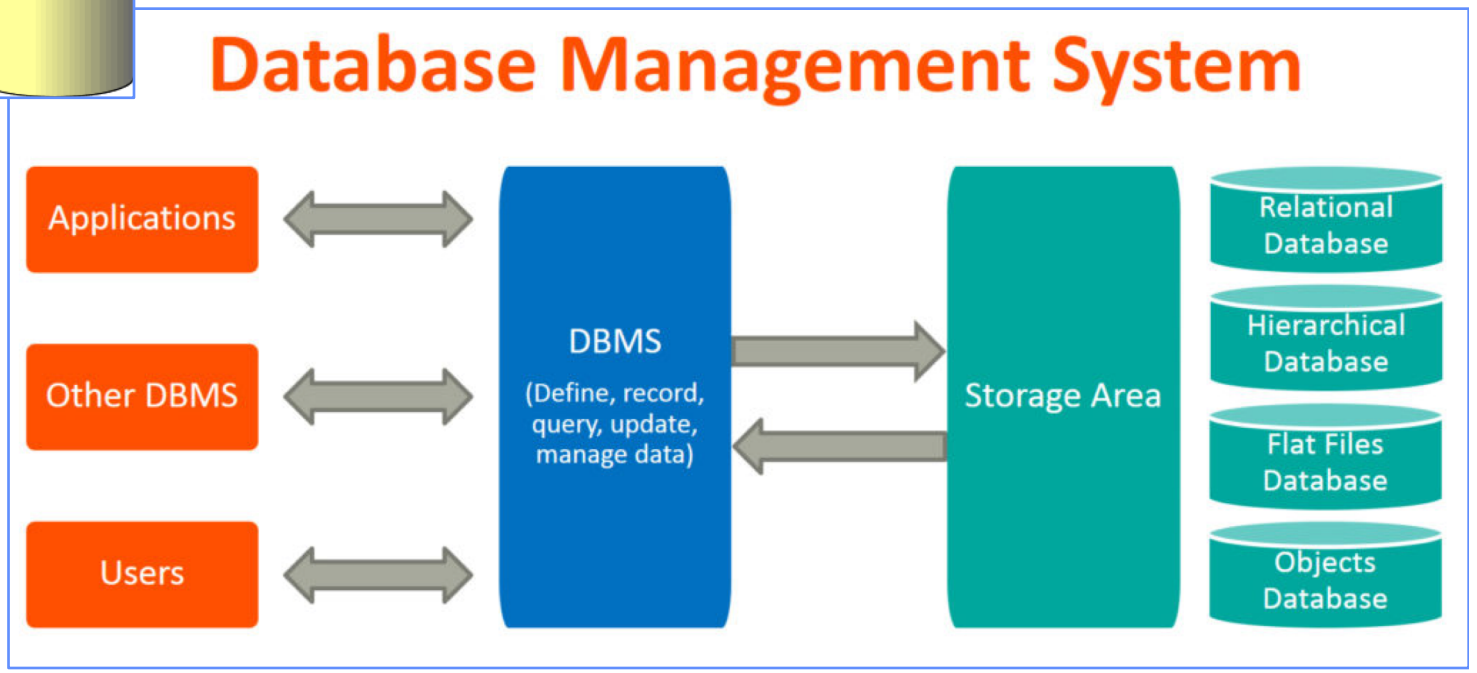
- Data lives in files: website access logs, in images, in CSVs and so on…
    - This is an amazing source, but hard to access, aggregate and compute results with.
- Databases provide a mechanism to store vast amounts of data in an *organized* manner.
    - The (often) rely on ”tables” as an abstraction. We
    - There are other kinds of databases, that store “documents” or other forms of data.
- This stuff is the topic of CS186

Why SQL?

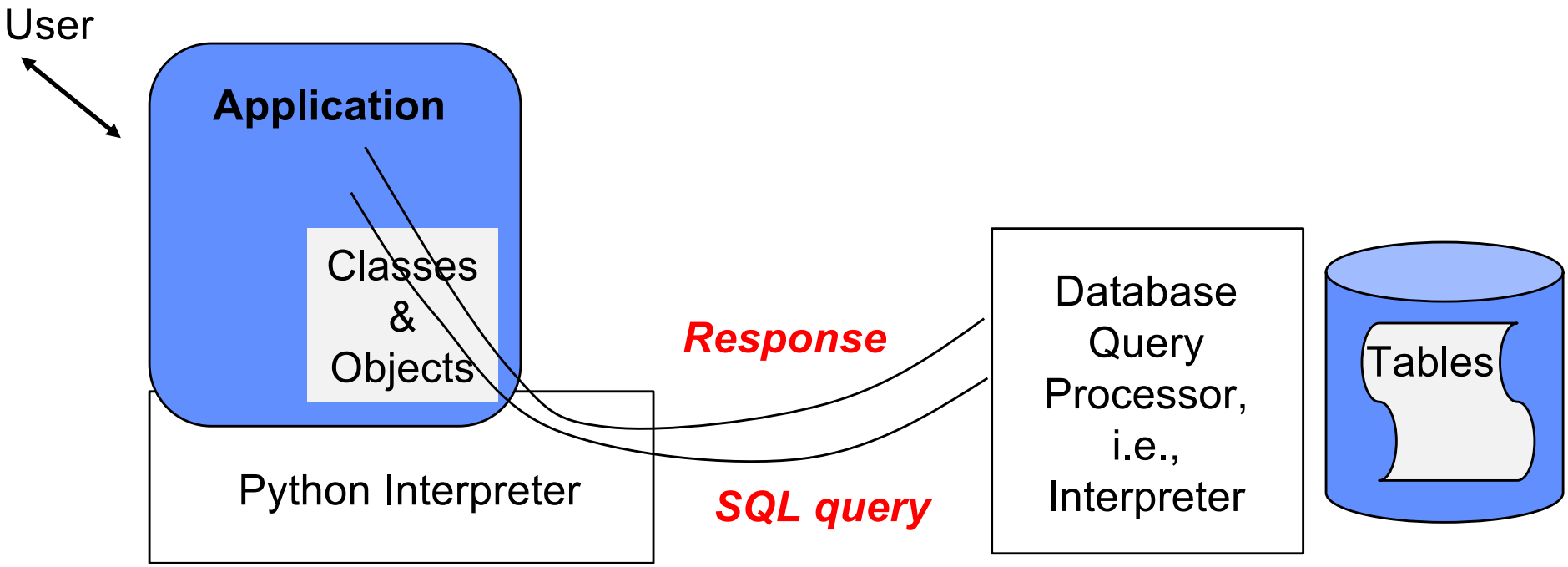
- SQL is a *declarative* programming language for accessing and modifying data in a relational database.
- It is an entirely new way of thinking ("new" in 1970, and new to you now!) that specifies *what* should happen, but not *how* it should happen.
- One of a few major programming paradigms
  - Imperative/Procedural
  - Object Oriented
  - Functional
  - Declarative

Database Management Systems





App in program language issues queries to a database interpreter

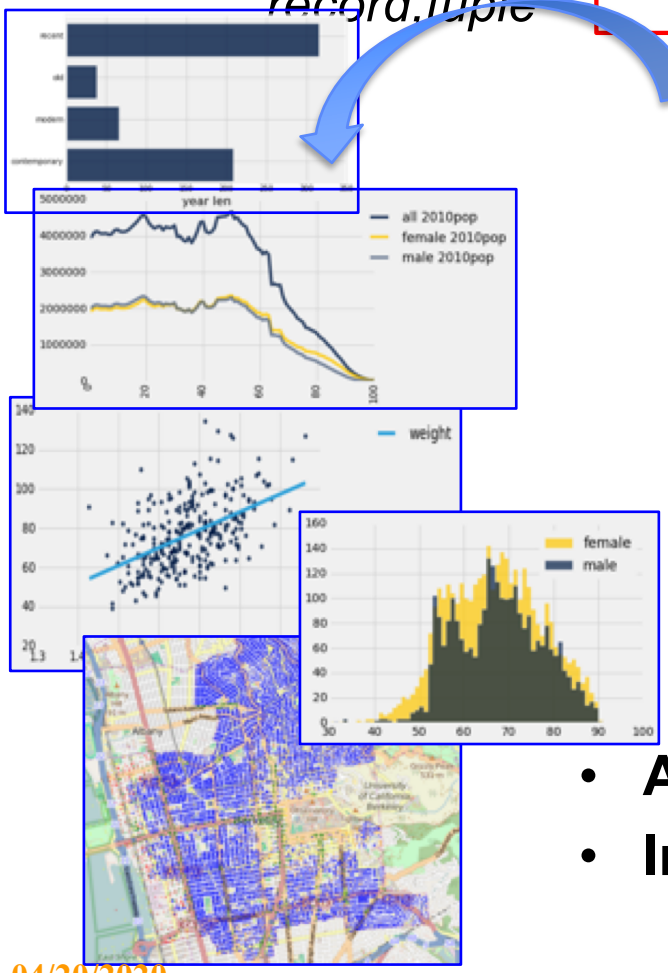


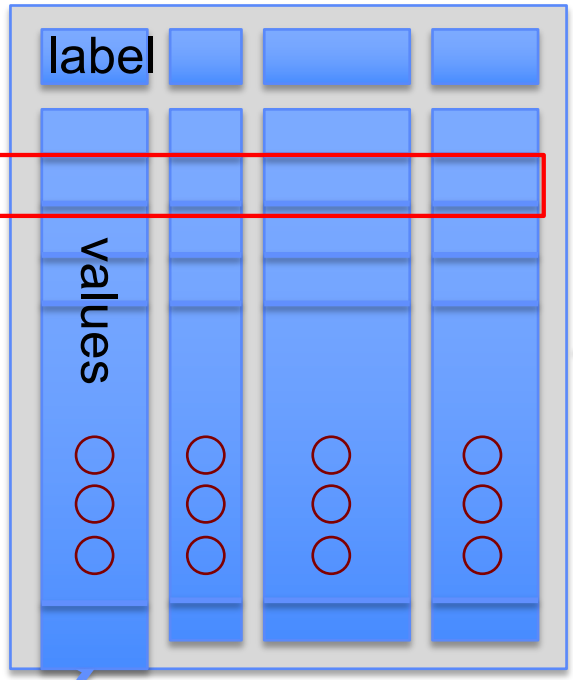
- **The SQL language is represented in query strings delivered to a DB backend.**
- **Use the techniques learned here to build clean abstractions.**
- **You have already learned the relational operators!**

Data 8 Tables

ordered collection of labeled columns of anything

dict,
record,tuple





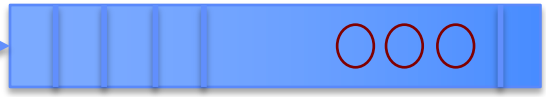
select, where, take, drop,
stats, bin
group
sample
pivot,
pivot_bin

split

join

T['label']

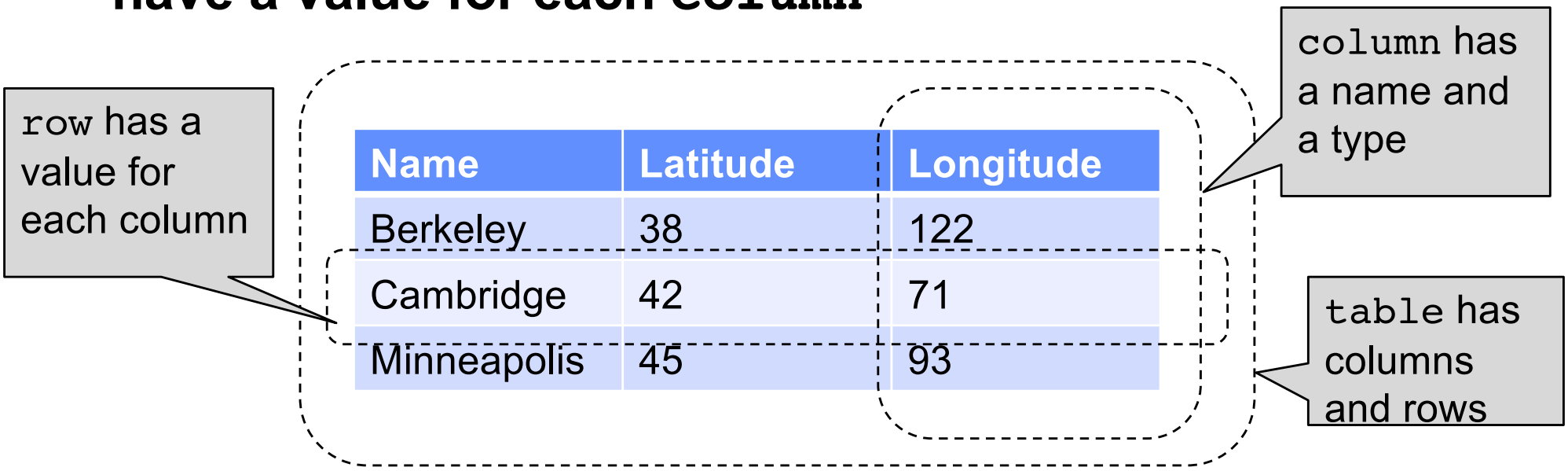
Numpy array



- **A single, simple, powerful data structure for all**
- **Inspired by Excel, SQL, R, Pandas, Numpy, …**

Database Management Systems

- **DBMS are persistent tables with powerful relational operators**
  - **Important, heavily used, interesting!**
- **A `table` is a collection of `records`, which are `rows` that have a value for each `column`**



- **Structure Query Language (SQL) is a declarative programming language describing operations on tables**

SQL

- A declarative language
  - Described *what* to compute
  - Imperative languages, like python, describe *how* to compute it
  - Query processor (interpreter) chooses which of many equivalent query plans to execute to perform the SQL statements
- ANSI and ISO standard, but many variants
  - This SQL will work on most databases.
- `SELECT` statement creates a new table, either from scratch or by projecting a table
- `create table` statement gives a global name to a table
- Lots of other statements
  - `analyze, delete, explain, insert, replace, update, …`
- The action is in select

SQL example

- **SQL statements create tables**
  - **Give it a try with sqlite3 or http://kripken.github.io/sql.js/GUI/**
  - **Each statement ends with ';'**

```
culler$ sqlite3
SQLite version 3.9.2 2015-11-02 18:31:45
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> select 38 as latitude, 122 as longitude, "Berkeley" as
name;
38|122|Berkeley
sqlite>
```

A Running example from Data 8 Lec 10

```
# An example of creating a Table from a list of rows.
Table(["Flavor","Color","Price"]).with_rows([
    ('strawberry','pink', 3.55),
    ('chocolate','light brown', 4.75),
    ('chocolate','dark brown', 5.25),
    ('strawberry','pink',5.25),
    ('bubblegum','pink',4.75)])
```

Flavor	Color	Price
strawberry	pink	3.55
chocolate	light brown	4.75
chocolate	dark brown	5.25
strawberry	pink	5.25
bubblegum	pink	4.75


```
culler@CullerMac ~/Classes/CS88-Fa18/ideas/sql> sqlite3 icecream.db
SQLite version 3.13.0 2016-05-18 10:57:30
Enter ".help" for usage hints.
sqlite>
```

select

- **Comma-separated list of *column descriptions***
- **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 "strawberry" as Flavor, "pink" as Color, 3.55 as Price;
```

- **`union` of select statements is a table containing the union of the rows**

```
select "strawberry" as Flavor, "pink" as Color, 3.55 as Price union
select "chocolate","light brown", 4.75 union
select "chocolate","dark brown", 5.25 union
select "strawberry","pink",5.25 union
select "bubblegum","pink",4.75;
```

`create table`

- **SQL often used interactively**
  - **Result of select displayed to the user, but not stored**
- **Create table statement gives the result a name**
  - **Like a variable, but for a permanent object**

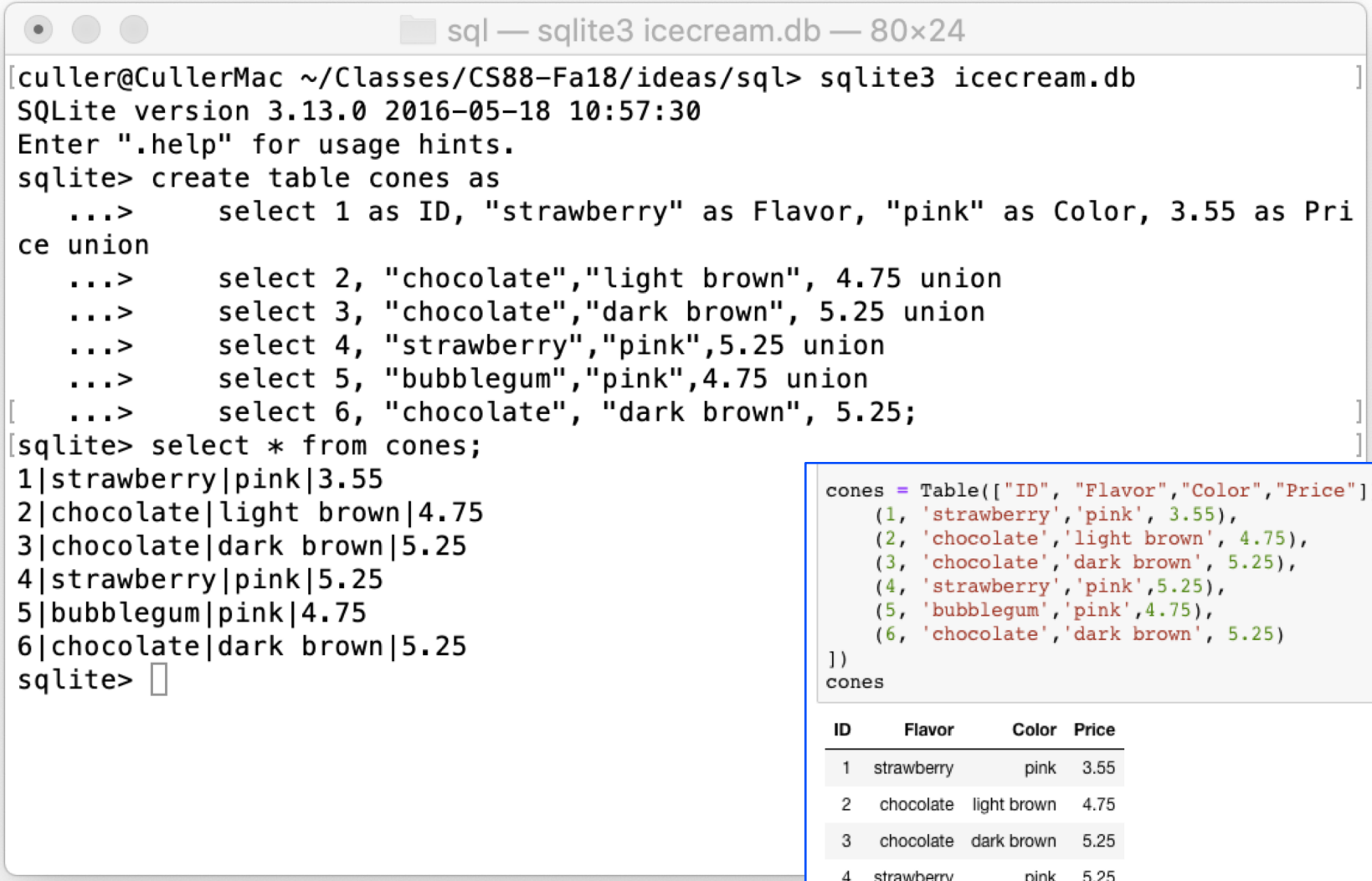
```
create table [name] as [select statement];
```

SQL: creating a named table

```
create table cones as
    select 1 as ID, "strawberry" as Flavor, "pink" as Color,
3.55 as Price union
    select 2, "chocolate","light brown", 4.75 union
    select 3, "chocolate","dark brown", 5.25 union
    select 4, "strawberry","pink",5.25 union
    select 5, "bubblegum","pink",4.75 union
    select 6, "chocolate", "dark brown", 5.25;
```

Notice how column names are introduced and implicit later on.

Select …




```
cones = Table(["ID", "Flavor","Color","Price"]).with_rows([
    (1, 'strawberry','pink', 3.55),
    (2, 'chocolate','light brown', 4.75),
    (3, 'chocolate','dark brown', 5.25),
    (4, 'strawberry','pink',5.25),
    (5, 'bubblegum','pink',4.75),
    (6, 'chocolate','dark brown', 5.25)
])
cones
```

ID	Flavor	Color	Price
1	strawberry	pink	3.55
2	chocolate	light brown	4.75
3	chocolate	dark brown	5.25
4	strawberry	pink	5.25
5	bubblegum	pink	4.75
6	chocolate	dark brown	5.25

Projecting existing tables

- Input table specified by `from` clause
- Subset of rows selected using a `where` clause
- Ordering of the selected rows declared using an `order by` clause

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

```
select * from cones order by Price;
```

ID	Flavor	Color	Price
1	strawberry	pink	3.55
2	chocolate	light brown	4.75
5	bubblegum	pink	4.75
3	chocolate	dark brown	5.25
4	strawberry	pink	5.25
6	chocolate	dark brown	5.25

Projection

```
In [5]: cones.select(['Flavor', 'Price'])
```

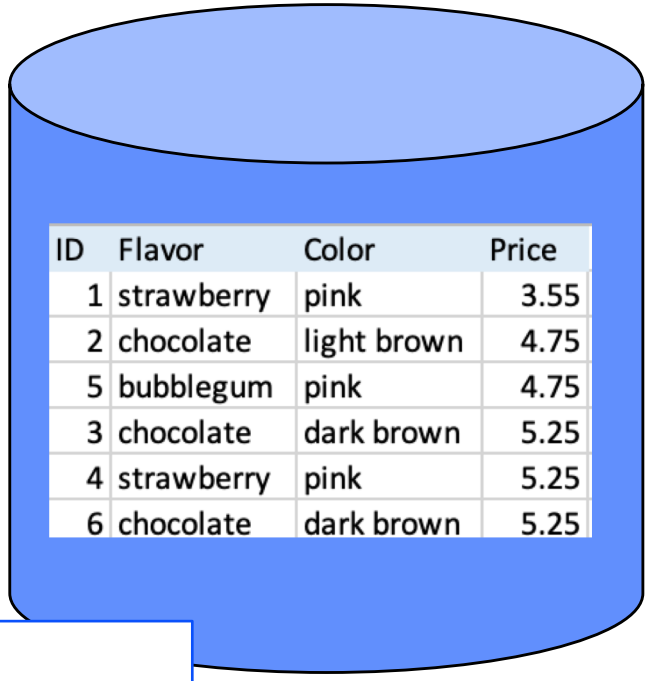
Out[5]:

Flavor	Price
strawberry	3.55
chocolate	4.75
chocolate	5.25
strawberry	5.25
bubblegum	4.75
chocolate	5.25

```
sqlite> select Flavor, Price from cones;
Flavor|Price
strawberry|3.55
chocolate|4.75
chocolate|5.25
strawberry|5.25
bubblegum|4.75
chocolate|5.25
```

- **A "projection" is a view of a table, it doesn't alter the state of the table.**

Permanent Data Storage



ID	Flavor	Color	Price
1	strawberry	pink	3.55
2	chocolate	light brown	4.75
5	bubblegum	pink	4.75
3	chocolate	dark brown	5.25
4	strawberry	pink	5.25
6	chocolate	dark brown	5.25

```
sqlite> .quit
culler@CullerMac ~/Classes/CS88-Fa18/ideas/sql> sqlite3 icecream.db
SQLite version 3.13.0 2016-05-18 10:57:30
Enter ".help" for usage hints.
sqlite> .tables
cones
sqlite> select * from cones where Color is "dark brown";
3|chocolate|dark brown|5.25
6|chocolate|dark brown|5.25
sqlite>
```

Filtering rows - where

- **Set of Table records (rows) that satisfy a condition**

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

```
In [5]: cones.select(['Flavor', 'Price'])
```

Out[5]:

Flavor	Price
strawberry	3.55
chocolate	4.75
chocolate	5.25
strawberry	5.25
bubblegum	4.75
chocolate	5.25

```
sqlite> select * from cones where Flavor = "chocolate";
ID|Flavor|Color|Price
2|chocolate|light brown|4.75
3|chocolate|dark brown|5.25
6|chocolate|dark brown|5.25
```

```
cones.where(cones["Price"] > 5)
```

:

ID	Flavor	Color	Price
3	chocolate	dark brown	5.25
4	strawberry	pink	5.25
6	chocolate	dark brown	5.25

SQL:

```
sqlite> select * from cones where Price > 5;
ID|Flavor|Color|Price
3|chocolate|dark brown|5.25
4|strawberry|pink|5.25
6|chocolate|dark brown|5.25
```

SQL Operators for predicate

- use the **WHERE** clause in the SQL statements such as SELECT, UPDATE and DELETE to filter rows that do not meet a specified condition

SQLite understands the following binary operators, in order from highest to lowest precedence:

```
||
*     /     %
+     -
<<    >>    &     |
<     <=    >     >=
=     ==    !=    <>    IS    IS NOT    IN    LIKE    GLOB    MATCH    REGEXP
AND
OR
```

Supported unary prefix operators are these:

```
-     +     ~     NOT
```

Summary – Part 1

```
SELECT <col spec> FROM <table spec> WHERE <cond spec>
    GROUP BY <group spec> ORDER BY <order spec>;
```

```
INSERT INTO table(column1, column2,...)
       VALUES (value1, value2,...);
```

CREATE TABLE name (<columns>) ;

CREATE TABLE name **AS** <select statement> ;

DROP TABLE name ;

Summary

- SQL a declarative programming language on relational tables
  - largely familiar to you from data8
  - create, select, where, order, group by, join
- Databases are accessed through Applications
  - e.g., all modern web apps have Database backend
  - Queries are issued through API
    - » Be careful about app corrupting the database
- Data analytics tend to draw database into memory and operate on it as a data structure
  - e.g., Tables
- More in lab