UC Berkeley EECS
Lecturer
Michael Ball

Computational Structures in Data Science

Databases & SQL

Announcements

- Chat: https://go.c88c.org/chat
- Attendance: https://go.c88c.org/here
  - Passcode: ice cream

Why SQL? (Review)

- 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
- Python is a multi-paradigm language, but we haven't yet tried declarative programming.

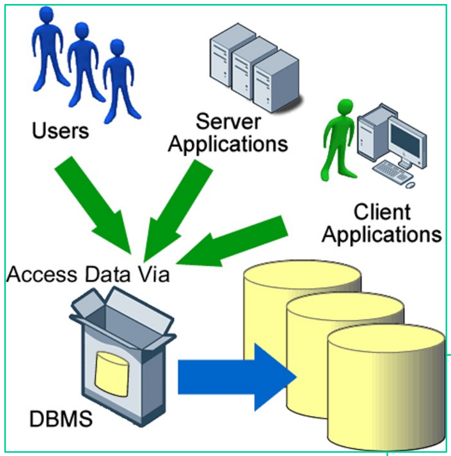
What is SQL?

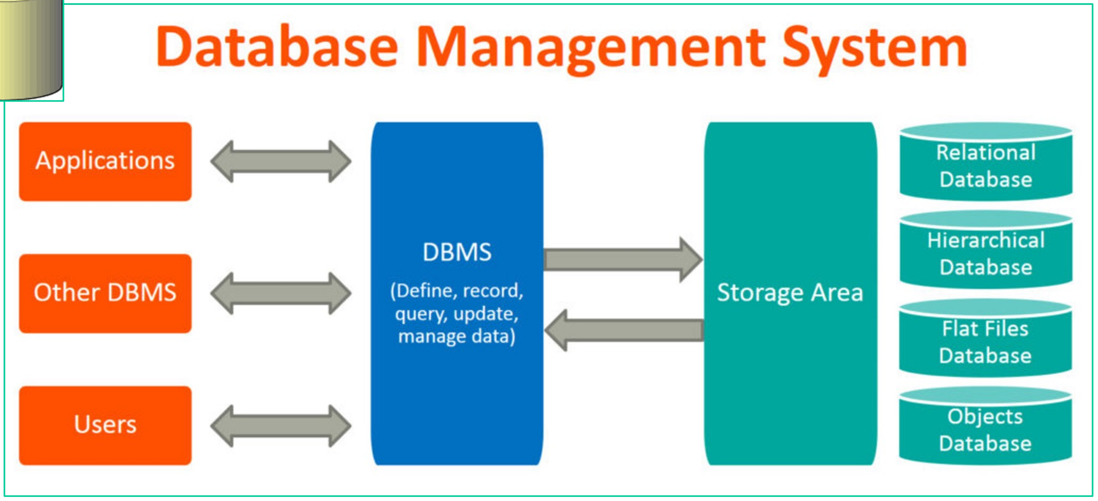
- A declarative language
  - Described *what* to compute
  - Imperative languages, like python, describe *how* to compute it
  - Query processor (interpreter) chooses which of many equivalent query plans to execute to perform the SQL statements
- ANSI and ISO standard, but many variants
  - CS88's SQL will work on nearly all relational databases—databases that use tables.
- `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,` …

Why Databases?

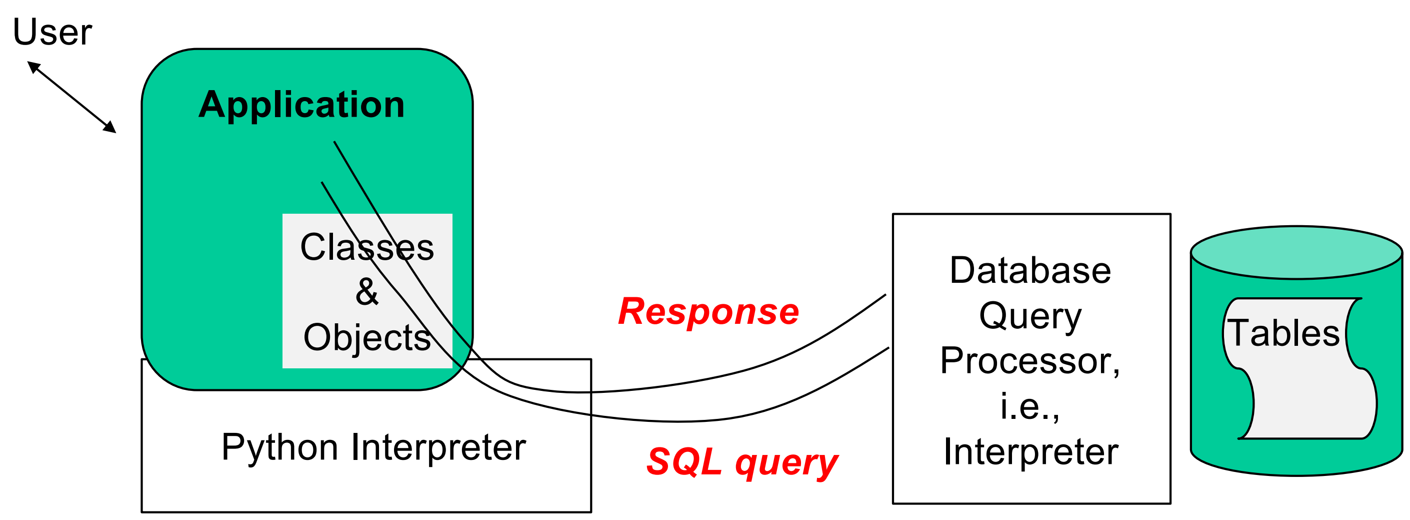
- Data lives in files: website access logs, in images, in CSVs and so on...
    - Useful, but hard to access, aggregate and compute results with.
- Databases provide a mechanism to store vast amounts of data in an *organized* manner.
    - The (often) rely on ”tables” as an abstraction.
    - There are other kinds of databases, that store “documents” or other forms of data.
- Databases is the topic of CS186
- Elsewhere: Data, it’s storage and accessing it are critical to data science.

Database Management Systems



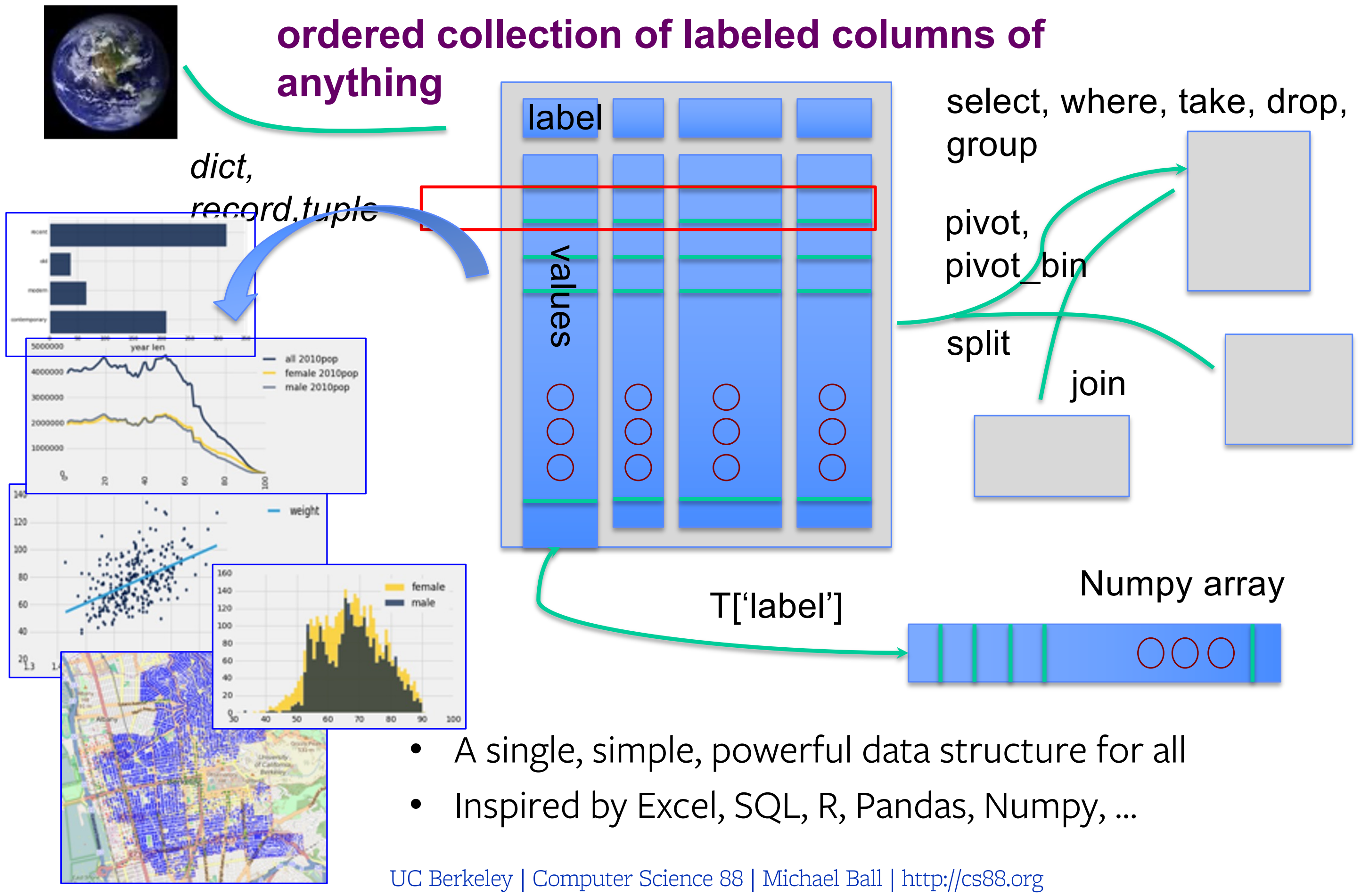


Applications Issue Queries to a Database



- 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!

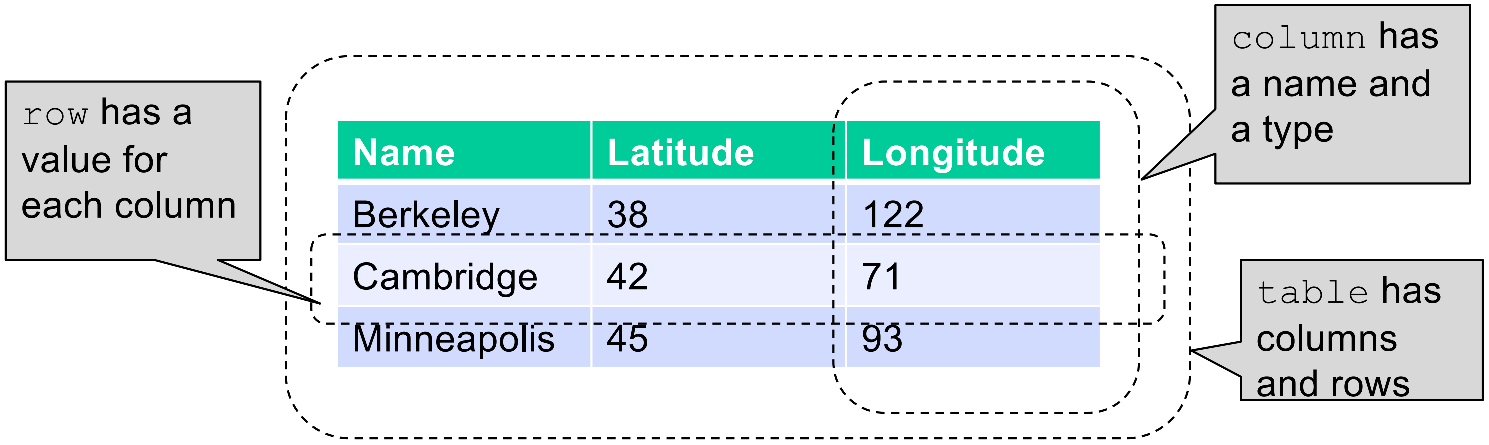
Data 8 Tables


ordered collection of labeled columns of anything
dict,
record,tuple
label
values
select, where, take, drop, group
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`



Name	Latitude	Longitude
Berkeley	38	122
Cambridge	42	71
Minneapolis	45	93

`row` has a value for each column
`column` has a name and a type
`table` has columns and rows

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

Announcements

- Chat: https://go.c88c.org/chat
- Attendance: https://go.c88c.org/here
  - Passcode: ice cream

UC Berkeley EECS
Lecturer
Michael Ball

Computational Structures in Data Science

Interacting With A Database

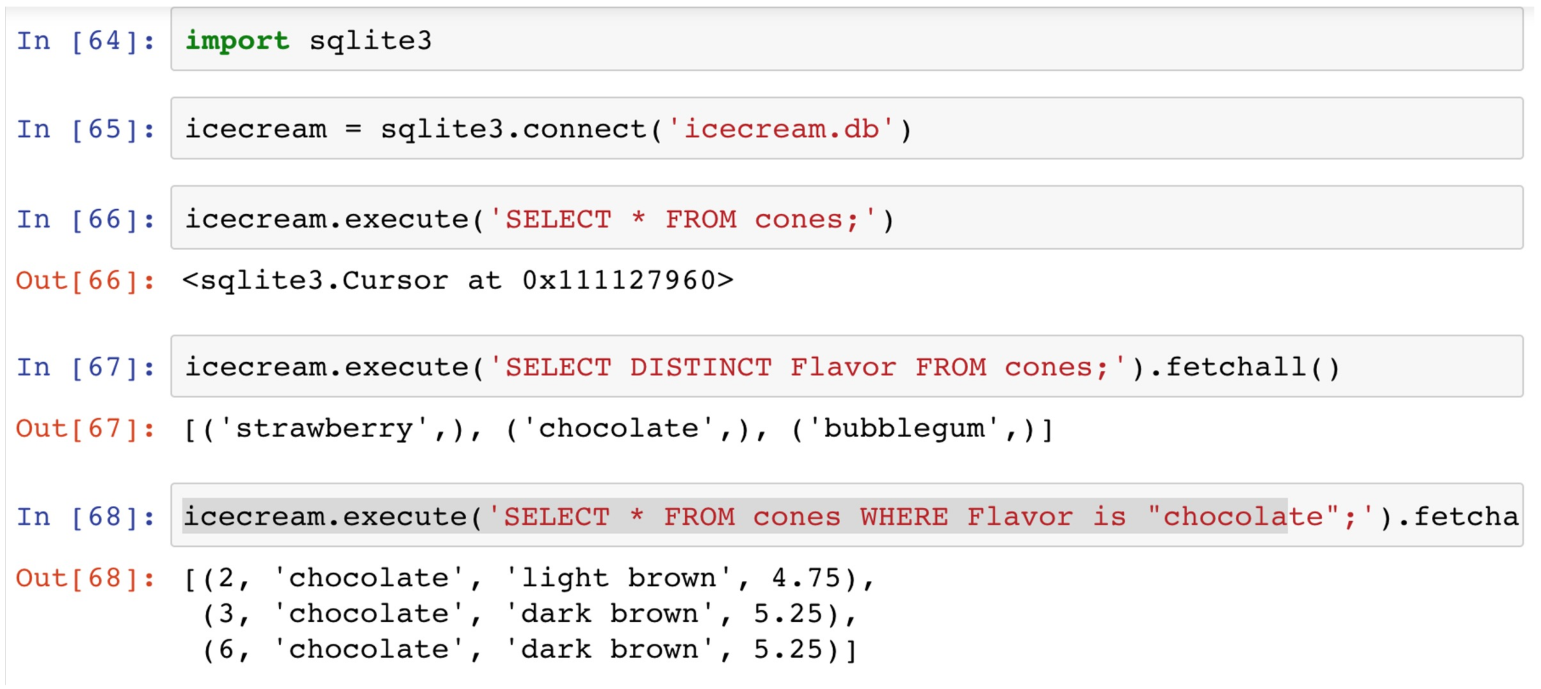
sqlite3 [Python Docs] [SQLite Docs]

- Pronounced "sequel lite"
- `sqlite3` is a Python module which connects to a very popular database!
- This is the first time you write code that really interacts with data on your computer!
- There's some "boilerplate" setup here, but it's not too bad…
- It's lightweight, fast, and works on most computers natively
  - It's incredibly popular! Such as Android, Apple apps, and even airplanes!
  - But sqlite is not setup for all applications, like such as websites like Gmail/bCourses, etc.
- A database is a `.db` file, which contains all of your data in an efficient form.
- Many people connect to sqlite through a Program like Python OR through the sqlite interpreter.

Connecting To a Database (Python 3)

```
DB_FILENAME = '25-Databases_and_SQL.db'
import sqlite3
# Talking to the database happens through a "connection"
con = sqlite3.connect(DB_FILENAME)
# A cursor is the object we use to execute a query.
cur = con.cursor()
# This returns an iterator!
result = cur.execute("YOUR QUERY")
for row in result:
    print(result) # This is a Tuple!
# Save (commit) the changes
con.commit()
# We can also close the connection if we are done with it.
# Just be sure any changes have been committed or they will be lost.
con.close()
```

SQLite python API – In a Notebook.



```
In [64]: import sqlite3

In [65]: icecream = sqlite3.connect('icecream.db')

In [66]: icecream.execute('SELECT * FROM cones;')
Out[66]: <sqlite3.Cursor at 0x111127960>

In [67]: icecream.execute('SELECT DISTINCT Flavor FROM cones;').fetchall()
Out[67]: [('strawberry',), ('chocolate',), ('bubblegum',)]

In [68]: icecream.execute('SELECT * FROM cones WHERE Flavor is "chocolate";').fetcha
Out[68]: [(2, 'chocolate', 'light brown', 4.75),
          (3, 'chocolate', 'dark brown', 5.25),
          (6, 'chocolate', 'dark brown', 5.25)]
```

The sqlite console

- Interactive console used *via the Terminal!*
- *Everything is saved automaticaly. BEWARE!*

There are many more commands than the ones shown here!, but these can be neat!

```
👉 sqlite3 23-Databases_and_SQL.db
SQLite version 3.37.0 2021-12-09 01:34:53
Enter ".help" for usage hints.
sqlite> .help
.echo on|off             Turn command echo on or off
.exit ?CODE?             Exit this program with return-code CODE
.headers on|off          Turn display of headers on or off
.help ?-all? ?PATTERN?   Show help text for PATTERN
.quit                    Exit this program
.show                    Show the current values for various settings
.tables ?TABLE?          List names of tables matching LIKE pattern TABLE
.trace ?OPTIONS?         Output each SQL statement as it is run
sqlite> .tables
cones sales
sqlite>
```

Announcements

- Chat: https://go.c88c.org/chat
- Attendance: https://go.c88c.org/here
  - Passcode: ice cream

UC Berkeley EECS
Lecturer
Michael Ball

Computational Structures in Data Science

Introduction to SQL

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
- Python is a multi-paradigm language, but we haven't yet tried declarative programming.

What is 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
  - CS88's SQL will work on nearly all relational databases—databases that use tables.
- `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,` …

SQL example

- SQL statements create tables
  - Give it a try with sqlite3 or code.cs61a.org
  - **Each statement ends with ';'**

```
cs88$ 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>
```

SQL Basics

- SQL Keywords are case-*insensitive*
  - e.g. `SELECT` and `select` do the same thing
  - I *try* to capitalize them to make it clear what's-what.
- The order of SQL keywords matters
  - e.g. `SELECT ... FROM … WHERE …`
- Every statement ends in a `;`
- Whitespace doesn't matter
  - But indentations and newlines help make queries readable!
- Despite being a standard, differences do exist between databases. We use `sqlite3`.

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]; . . .
```

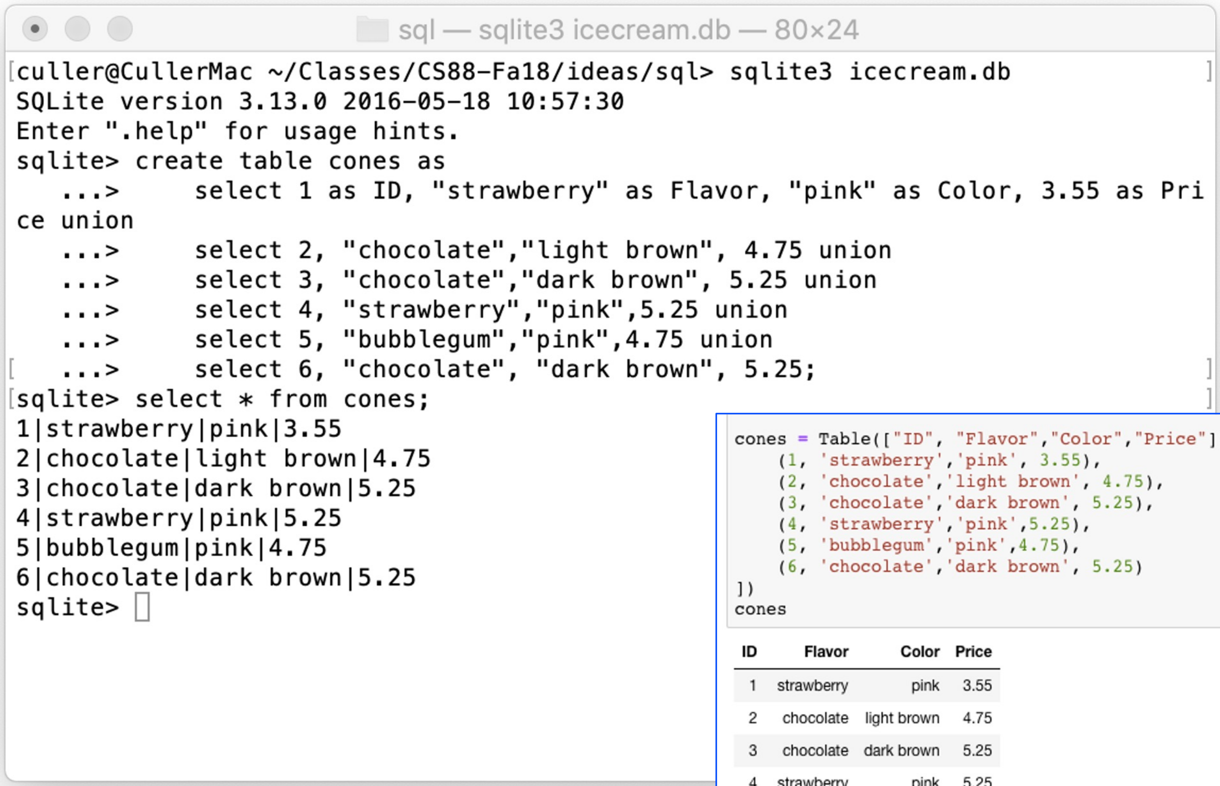
- Selecting literals creates a one-row table

```
select "strawberry" as Flavor, "pink" as Color, 3.55 as Price;
```

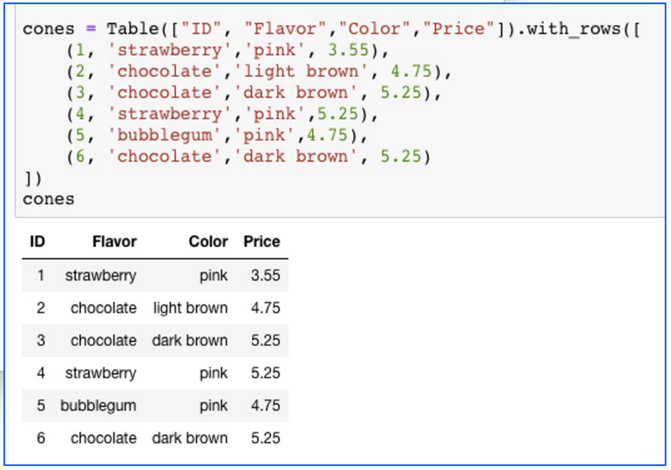
- `union` of select statements is a table containing the union of the rows

```
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;
```

Select ...



```
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> create table cones as
   ...>     select 1 as ID, "strawberry" as Flavor, "pink" as Color, 3.55 as Pri
ce 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;
sqlite> select * from cones;
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
sqlite>
```



```
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

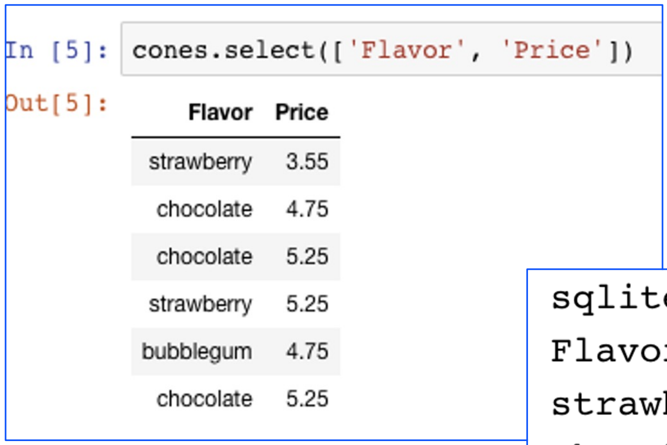
- Input table specified by `from` clause
- Subset of rows selected using a `where` clause
- Ordering of the selected rows declared using an `order by` clause

```
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.

UC Berkeley EECS
Lecturer
Michael Ball

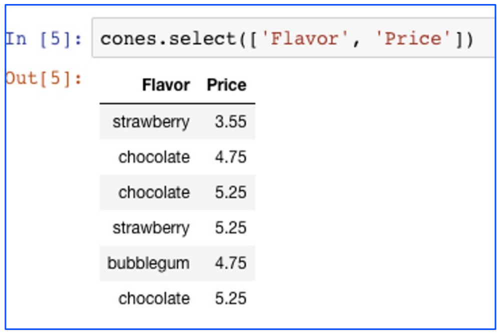
Computational Structures in Data Science

Filtering in SQL

Filtering rows - where

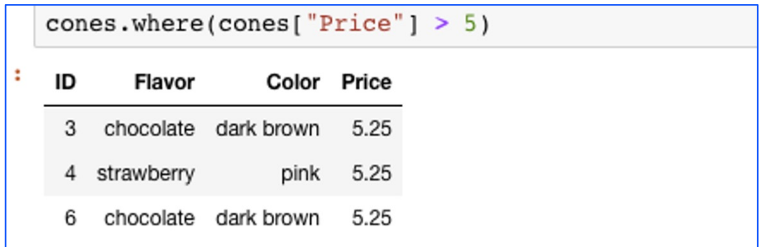
- Set of Table records (rows) that satisfy a condition

```
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



```
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

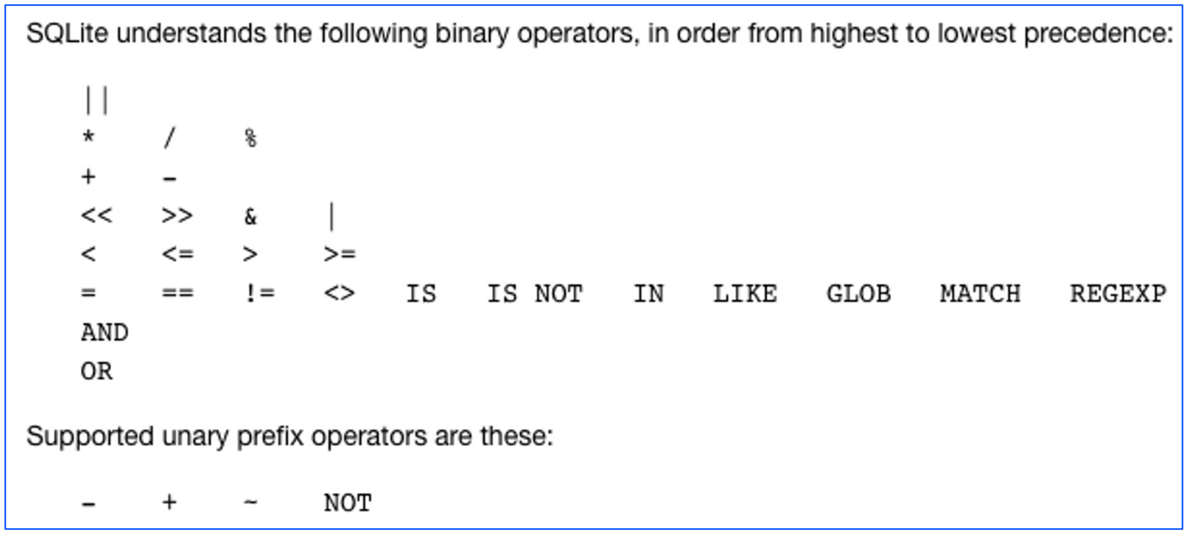
SQL:

```
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
```

```
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
```

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

- 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

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.

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