UC Berkeley EECS
Lecturer
Michael Ball

UC Berkeley EECS
Adj. Ass. Prof.
Dr. Gerald Friedland

Computational Structures in Data Science

Databases & SQL

Computing In the News

- ″Pandemic has forced producers to bring new technology to sets″ (LATimes)
- After the pandemic shut down live entertainment, Guzzetta, 56, created a new company, Safe Haus Group, and adapted some of his safety technology for use on film sets.
- “It allowed me to put the live event business over on a shelf and not be depressed by that,” he said. “I’ve taken bits and pieces of different tech that I used in the other spaces and wrote some new software with my development team and we came up with Safe Set.”
- Safe Set is one of a group of new and existing technology companies capitalizing on the demand for safe productions. These businesses, which supply everything from remote-operated robotic cameras to tracking technology that helps enforce social distancing, have emerged in response to new safety protocols on sets.

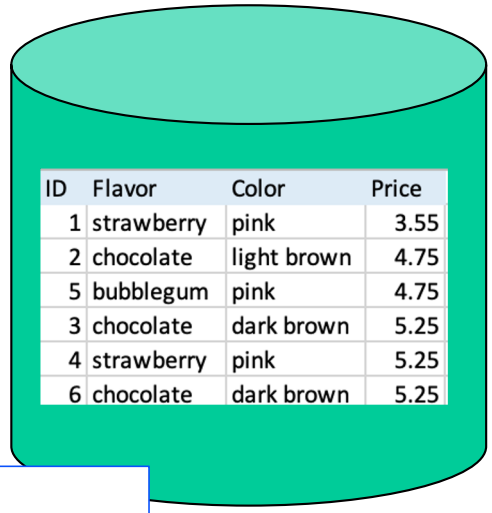
UC Berkeley EECS
Lecturer
Michael Ball

SQL: SELECT Statements

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

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

```
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>
```

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;
```

Projecting existing tables

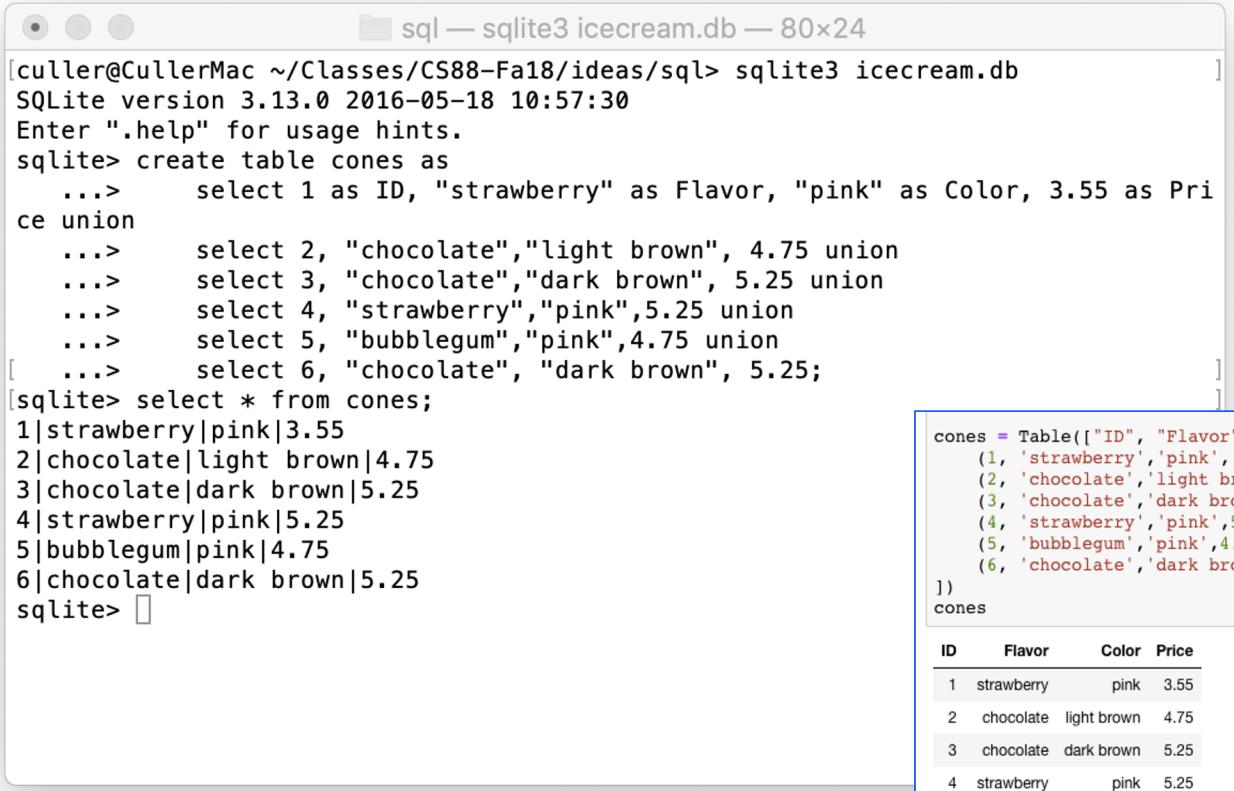
- Input table specified by `from` clause
- Subset of rows selected using a `where` clause
- Ordering of the selected rows declared using an `order by` clause

```
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

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

UC Berkeley EECS
Lecturer
Michael Ball

SQL: Filtering Queries

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
```

Approximate Matching ...

Regular expression matches are so common that they are built in in SQL.

```
sqlite> select * from cones where Flavor like "%berry%";
Flavor|Color|Price
strawberry|pink|3.55
strawberry|pink|5.25
sqlite>
```

On the other hand, you have the full power of Python to express what you mean.

```
cones.where(cones.apply(lambda x:'berry' in x, 'Flavor'))
```

ID	Flavor	Color	Price
1	strawberry	pink	3.55
4	strawberry	pink	5.25

UC Berkeley EECS
Lecturer
Michael Ball

Computational Structures in Data Science

SQL: CREATE and INSERT and UPDATE

`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.

Inserting new records (rows)

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

```
sqlite> insert into cones(ID, Flavor, Color, Price) values (7, "Vanila", "White", 3.95);
sqlite> select * from 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
7|Vanila|White|3.95
sqlite>
```

```
cones.append((7, "Vanila", "White", 3.95))
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
7	Vanila	White	3.95

- A database table is typically a shared, durable repository shared by multiple applications

UPDATING new records (rows)

```
UPDATE table SET column1 = value1, column2 =
value2 [WHERE condition];
```

- If you don't specify a WHERE, you'll update all rows!

UC Berkeley EECS
Lecturer
Michael Ball

SQL: Aggregations

Group and Aggregate

- The GROUP BY clause is used to group rows returned by SELECT statement into a set of summary rows or groups based on values of columns or expressions.
- Apply an aggregate function, such as SUM, AVG, MIN, MAX or COUNT, to each group to output the summary information.

```
cones.group('Flavor')
```

Flavor	count
bubblegum	1
chocolate	3
strawberry	2

```
sqlite> select count(Price), Flavor from cones group by Flavor;
count(Price)|Flavor
1|bubblegum
2|chocolate
2|strawberry
```

```
cones.select(['Flavor', 'Price']).group('Flavor', np.mean)
```

Flavor	Price mean
bubblegum	4.75
chocolate	5.08333
strawberry	4.4

```
sqlite> select avg(Price), Flavor from cones group by Flavor;
avg(Price)|Flavor
4.75|bubblegum
5.0|chocolate
4.4|strawberry
```

UNIQUE / Distinct values

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

```
sqlite> select distinct Flavor, Color from cones;
strawberry|pink
chocolate|light brown
chocolate|dark brown
bubblegum|pink
sqlite>
```

```
In [8]: cones.groups(['Flavor', 'Color']).drop('count')
```

Out[8]:

Flavor	Color
bubblegum	pink
chocolate	dark brown
chocolate	light brown
strawberry	pink

```
In [7]: np.unique(cones['Flavor'])
Out[7]: array(['bubblegum', 'chocolate', 'strawberry'], dtype='<U10')
```

UC Berkeley EECS
Lecturer
Michael Ball

Computational Structures in Data Science

SQL: Joins

Joining tables

- Two tables are joined by a comma to yield all combinations of a row from each
  - `select * from sales, cones;`

```
create table sales as
    select "Baskin" as Cashier, 1 as TID union
    select "Baskin", 3 union
    select "Baskin", 4 union
    select "Robin", 2 union
    select "Robin", 5 union
    select "Robin", 6;
```

Cashier	TID
Baskin	1
Robin	2
Baskin	3
Baskin	4
Robin	5
Robin	6

```
sales.join('TID', cones, 'ID')
```

TID	Cashier	Flavor	Color	Price
1	Baskin	strawberry	pink	3.55
2	Robin	chocolate	light brown	4.75
3	Baskin	chocolate	dark brown	5.25
4	Baskin	strawberry	pink	5.25
5	Robin	bubblegum	pink	4.75
6	Robin	chocolate	dark brown	5.25

```
sqlite> select * from sales, cones;
Baskin|1|1|strawberry|pink|3.55
Baskin|1|2|chocolate|light brown|4.75
Baskin|1|3|chocolate|dark brown|5.25
Baskin|1|4|strawberry|pink|5.25
Baskin|1|5|bubblegum|pink|4.75
Baskin|1|6|chocolate|dark brown|5.25
Baskin|3|1|strawberry|pink|3.55
Baskin|3|2|chocolate|light brown|4.75
Baskin|3|3|chocolate|dark brown|5.25
Baskin|3|4|strawberry|pink|5.25
Baskin|3|5|bubblegum|pink|4.75
Baskin|3|6|chocolate|dark brown|5.25
Baskin|4|1|strawberry|pink|3.55
Baskin|4|2|chocolate|light brown|4.75
Baskin|4|3|chocolate|dark brown|5.25
Baskin|4|4|strawberry|pink|5.25
Baskin|4|5|bubblegum|pink|4.75
Baskin|4|6|chocolate|dark brown|5.25
Robin|2|1|strawberry|pink|3.55
Robin|2|2|chocolate|light brown|4.75
Robin|2|3|chocolate|dark brown|5.25
Robin|2|4|strawberry|pink|5.25
Robin|2|5|bubblegum|pink|4.75
Robin|2|6|chocolate|dark brown|5.25
Robin|5|1|strawberry|pink|3.55
Robin|5|2|chocolate|light brown|4.75
Robin|5|3|chocolate|dark brown|5.25
Robin|5|4|strawberry|pink|5.25
Robin|5|5|bubblegum|pink|4.75
Robin|5|6|chocolate|dark brown|5.25
Robin|6|1|strawberry|pink|3.55
Robin|6|2|chocolate|light brown|4.75
Robin|6|3|chocolate|dark brown|5.25
Robin|6|4|strawberry|pink|5.25
Robin|6|5|bubblegum|pink|4.75
Robin|6|6|chocolate|dark brown|5.25
```

Inner Join

```
select * from sales, cones where TID=ID;
```

```
sales.join('TID', cones, 'ID')
```

TID	Cashier	Flavor	Color	Price
1	Baskin	strawberry	pink	3.55
2	Robin	chocolate	light brown	4.75
3	Baskin	chocolate	dark brown	5.25
4	Baskin	strawberry	pink	5.25
5	Robin	bubblegum	pink	4.75
6	Robin	chocolate	dark brown	5.25

```
sqlite> select * from sales, cones where TID=ID;
Baskin|1|1|strawberry|pink|3.55
Baskin|3|3|chocolate|dark brown|5.25
Baskin|4|4|strawberry|pink|5.25
Robin|2|2|chocolate|light brown|4.75
Robin|5|5|bubblegum|pink|4.75
Robin|6|6|chocolate|dark brown|5.25
sqlite>
```

SQL: using named tables - `from`

```
select "delicious" as Taste, Flavor, Color from cones
       where Flavor is "chocolate" union
select "other", Flavor, Color from cones
       where Flavor is not "chocolate";
```

```
sqlite> select "delicious" as Taste, Flavor, Color from cones where Flavor is "chocolate" union
   ...> select "other", Flavor, Color from cones where Flavor is not "chocolate";
Taste|Flavor|Color
delicious|chocolate|dark brown
delicious|chocolate|light brown
other|bubblegum|pink
other|strawberry|pink
sqlite>
```

Queries within queries

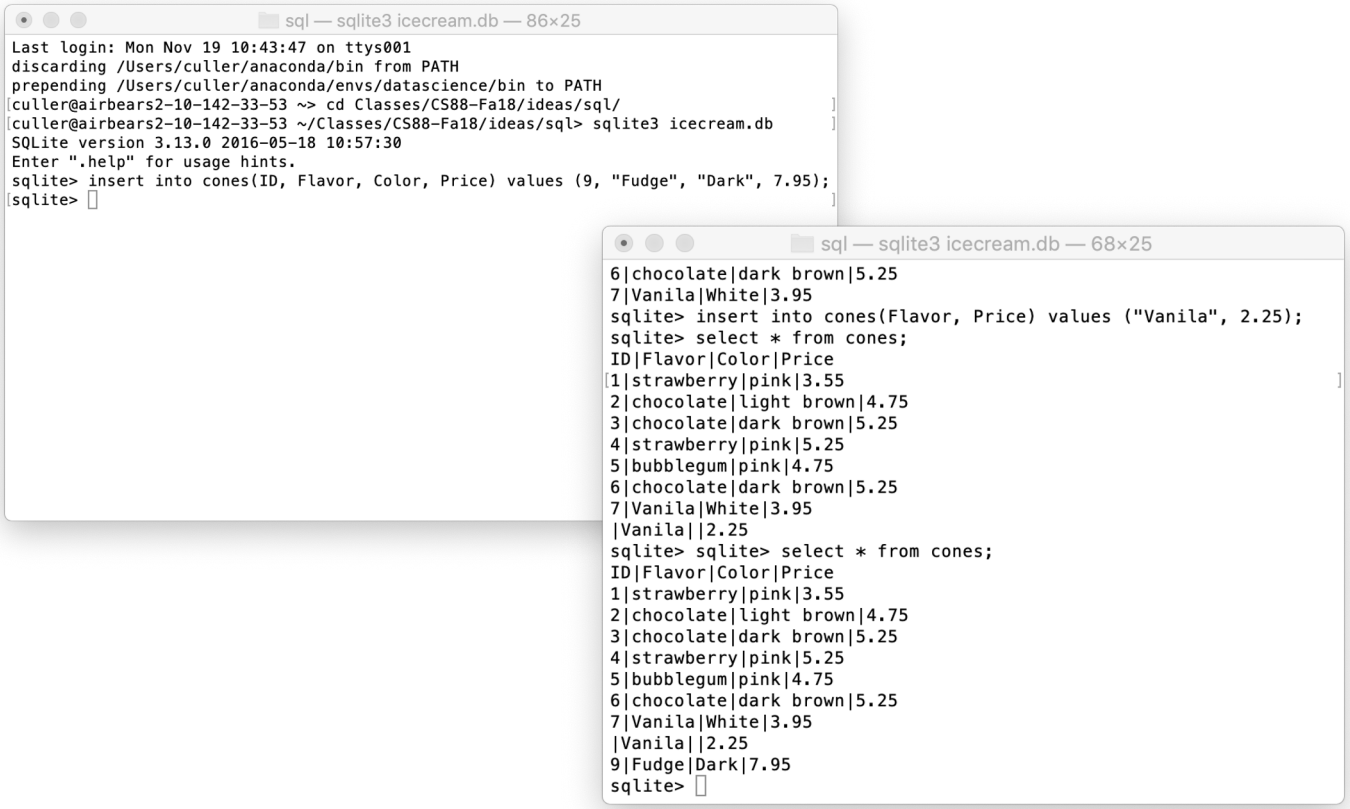
- Any place that a table is named within a select statement, a table could be computed
  - As a sub-query

```
select TID from sales where Cashier is "Baskin";

select * from cones
    where ID in (select TID from sales where Cashier is "Baskin");

sqlite> select * from cones
   ...>     where ID in (select TID from sales where Cashier is "Baskin");
ID|Flavor|Color|Price
1|strawberry|pink|3.55
3|chocolate|dark brown|5.25
4|strawberry|pink|5.25
```

UC Berkeley EECS
Lecturer
Michael Ball

Bonus: SQL From Python

Multiple clients of the database



- All of the inserts update the common repository

SQLite python API

```
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)]
```

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 ;
```