

a Taste of Declarative Programming in SQL

David E. Culler
 CS8 – Computational Structures in Data Science
<http://inst.eecs.berkeley.edu/~cs88>

Lecture 13
 November 19, 2018

Administrative

- Thanks to all of you who help so much
- Regular lecture and lab this week
 - Brief introduction to SQL providing review of what you've learned
 - Course evaluation in last 10 mins
 - Read 4.3
- Deferred project 2 due Wednesday
- Monday RRR lecture provides review
 - Regular place and time
- Additional review session
- Regular Final Exam: Th 12/13 3-6 pm
- Alternative Final by request
<http://bit.ly/cs88-fa18-L13>

Database Management Systems

Database Management System

Applications ↔ DBMS (Define, record, query, update, manage data) ↔ Storage Area

Other DBMS ↔ DBMS

Users ↔ DBMS

Relational Database
 Hierarchical Database
 Flat Files Database
 Object Database

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

- 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

- 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
- **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

11/26/18

UCB CS88 Sp18 L13

7

SQL example

- SQL statements create tables
 - Give it a try with `sqlite3` or <http://kripken.github.io/sql.is/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>
```

11/26/18

UCB CS88 Sp18 L13

8

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>
```

11/26/18

UCB CS88 Sp18 L13

9

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;
```

11/26/18

UCB CS88 Sp18 L13

10

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];
```

11/26/18

UCB CS88 Sp18 L13

11

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.

11/26/18

UCB CS88 Sp18 L13

12

Select ...

```

culler@CullerMac ~/Classes/CS88-Fa18/ideas/sql> sqlite3 icecream.db
SQLite version 3.13.0 2016-05-18 18:57:30
Enter ".help" for usage hints.
sqlite> create table cones as
...> select 1 as ID, "strawberry" as Flavor, "pink" as Color, 3.55 as 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;
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>

```

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

11/26/18 UCB CS88 Sp18 L13 13

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

11/26/18 UCB CS88 Sp18 L13 14

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

```

- Select versus indexing a column?

11/26/18 UCB CS88 Sp18 L13 15

Permanent Data Storage



```

sqlite> .quit
culler@CullerMac ~/Classes/CS88-Fa18/ideas/sql> sqlite3 icecream.db
SQLite version 3.13.0 2016-05-18 18:57:30
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>

```

11/26/18 UCB CS88 Sp18 L13 16

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

```

11/26/18 UCB CS88 Sp18 L13 17

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

```

11/26/18 UCB CS88 Sp18 L13 18

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

11/26/18

UCB CS88 Sp18 L13

19

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 5.00000
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
```

11/26/18

UCB CS88 Sp18 L13

20

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>
```

```
cones.groupby(['Flavor', 'Color']).drop('count')
Flavor Color
bubblegum pink
chocolate dark brown
chocolate light brown
strawberry pink
```

```
is (7) np.unique(cones['Flavor'])
out(7) array(['bubblegum', 'chocolate', 'strawberry'], dtype=object)
```

- Built in to the language or a composable tool?

11/26/18

UCB CS88 Sp18 L13

21

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", 3 union
select "Baskin", 4 union
select "Robin", 2 union
select "Robin", 5 union
select "Robin", 6;
```

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

11/26/18

UCB CS88 Sp18 L13

22

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|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>
```

11/26/18

UCB CS88 Sp18 L13

23

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>
```

11/26/18

UCB CS88 Sp18 L13

24

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
```

11/26/18

UCB CS88 Sp18 L13

25

Inserting new records (rows)

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

```
sqlite> insert into cones(ID, Flavor, Color, Price) values (7, "Vanilla", "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|Vanilla|White|3.95
sqlite>
```

```
cones.append(7, "Vanilla", "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	Vanilla	White	3.95

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

11/26/18

UCB CS88 Sp18 L13

26

Multiple clients of the database

```
sqlite> insert into cones(ID, Flavor, Color, Price) values (9, "Fudge", "Dark", 7.95);
sqlite>
```

```
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|Vanilla|White|3.95
8|Vanilla|Light brown|4.75
9|Fudge|Dark|7.95
sqlite>
```

- All of the inserts update the common repository

11/26/18

UCB CS88 Sp18 L13

27

SQLite python API

```
In [64]: import sqlite3

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

In [66]: icream.execute('SELECT * FROM cones;')
Out[66]: <sqlite3.Cursor at 0x11127960>

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

In [68]: icream.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)]
```

11/26/18

UCB CS88 Sp18 L13

28

Creating DB Abstractions

```
class SQL_Table(Table):
    """ Extend Table class with methods to read/write a Table
    from/to a table in a SQLite3 database. """
    @classmethod
    def read(cls, filepath, table, verbose=False):
        """ Create a SQL_Table by reading a table from a SQL database. """
        dbconn = sqlite3.connect(filepath,
            detect_types=sqlite3.PARSE_COLNAMES)

        col_names = sqlcol_names(dbconn, table)
        rows = sqllexec(dbconn, 'SELECT * from %s;' % table, verbose).fetchall()
        dbconn.close()
        return cls(col_names).with_rows(rows)
```

11/26/18

UCB CS88 Sp18 L13

29

DB Abstraction (cont)

```
class SQL_Table(Table):
    ...
    def write(self, filepath, table, verbose=False, overwrite=True):
        """ Write a Table into a SQL database as a SQL table. """
        dbconn = sqlite3.connect(filepath)
        # Create table and insert each row
        cols = build_list(self.labels)
        sqlexec(dbconn, "CREATE TABLE %s %s;" % (table, cols), verbose)
        for row in self.rows:
            sqllexec(dbconn, 'INSERT INTO %s VALUES %s;' % (table, tuple(row)
            dbconn.commit()
            dbconn.close()

    @classmethod
    def cast(cls, table):
        """ Return a SQL_Table version of a Table. """
        return cls().with_columns(zip(table.labels, table.columns))
```

11/26/18

UCB CS88 Sp18 L13

30

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;
```

11/26/18

UCB CS88 Sp18 L13

31

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**

11/26/18

UCB CS88 Sp18 L13

32