Computational Structures in Data Science

Databases & SQL
“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.
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

```sql
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*

```sql
select [expression] as [name], [expression] as [name];
```

- Selecting literals creates a one-row table

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

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

```sql
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

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

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

<table>
<thead>
<tr>
<th>ID</th>
<th>Flavor</th>
<th>Color</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>strawberry</td>
<td>pink</td>
<td>3.55</td>
</tr>
<tr>
<td>2</td>
<td>chocolate</td>
<td>light brown</td>
<td>4.75</td>
</tr>
<tr>
<td>5</td>
<td>bubblegum</td>
<td>pink</td>
<td>4.75</td>
</tr>
<tr>
<td>3</td>
<td>chocolate</td>
<td>dark brown</td>
<td>5.25</td>
</tr>
<tr>
<td>4</td>
<td>strawberry</td>
<td>pink</td>
<td>5.25</td>
</tr>
<tr>
<td>6</td>
<td>chocolate</td>
<td>dark brown</td>
<td>5.25</td>
</tr>
</tbody>
</table>
SELECT

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>
SQL: Filtering Queries
Filtering rows - WHERE

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

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

<table>
<thead>
<tr>
<th>ID</th>
<th>Flavor</th>
<th>Color</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>strawberry</td>
<td>pink</td>
<td>3.55</td>
</tr>
<tr>
<td>4</td>
<td>strawberry</td>
<td>pink</td>
<td>5.25</td>
</tr>
</tbody>
</table>
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

```sql
create table [name] as [select statement];
```
SQL: creating a named table

```sql
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)

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

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

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

<table>
<thead>
<tr>
<th>ID</th>
<th>Flavor</th>
<th>Color</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>strawberry</td>
<td>pink</td>
<td>3.55</td>
</tr>
<tr>
<td>2</td>
<td>chocolate</td>
<td>light brown</td>
<td>4.75</td>
</tr>
<tr>
<td>3</td>
<td>chocolate</td>
<td>dark brown</td>
<td>5.25</td>
</tr>
<tr>
<td>4</td>
<td>strawberry</td>
<td>pink</td>
<td>5.25</td>
</tr>
<tr>
<td>5</td>
<td>bubblegum</td>
<td>pink</td>
<td>4.75</td>
</tr>
<tr>
<td>6</td>
<td>chocolate</td>
<td>dark brown</td>
<td>5.25</td>
</tr>
<tr>
<td>7</td>
<td>Vanilla</td>
<td>White</td>
<td>3.95</td>
</tr>
</tbody>
</table>

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

UC Berkeley | Computer Science 88 | Michael Ball | http://cs88.org
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!
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.

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

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

```sql
sqlite> select avg(Price), Flavor from cones group by Flavor;
avg(Price)|Flavor
4.75|bubblemum
5.0|chocolate
4.4|strawberry
```
**UNIQUE / Distinct values**

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

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

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

<table>
<thead>
<tr>
<th>Flavor</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>bubblegum</td>
<td>pink</td>
</tr>
<tr>
<td>chocolate</td>
<td>dark brown</td>
</tr>
<tr>
<td>chocolate</td>
<td>light brown</td>
</tr>
<tr>
<td>strawberry</td>
<td>pink</td>
</tr>
</tbody>
</table>
```

```python
In [7]: np.unique(cones[\'Flavor\'])
Out[7]: array([\'bubblegum\', \'chocolate\', \'strawberry\'], dtype=\'U10\')
```
SQL: Joins
Joining tables

- Two tables are joined by a comma to yield all combinations of a row from each

```sql
  select * from sales, cones;
```

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

<table>
<thead>
<tr>
<th>TID</th>
<th>Cashier</th>
<th>Flavor</th>
<th>Color</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Baskin</td>
<td>strawberry</td>
<td>pink</td>
<td>3.55</td>
</tr>
<tr>
<td>2</td>
<td>Robin</td>
<td>chocolate</td>
<td>light brown</td>
<td>4.75</td>
</tr>
<tr>
<td>3</td>
<td>Baskin</td>
<td>chocolate</td>
<td>dark brown</td>
<td>5.25</td>
</tr>
<tr>
<td>4</td>
<td>Baskin</td>
<td>strawberry</td>
<td>pink</td>
<td>5.25</td>
</tr>
<tr>
<td>5</td>
<td>Robin</td>
<td>bubblegum</td>
<td>pink</td>
<td>4.75</td>
</tr>
<tr>
<td>6</td>
<td>Robin</td>
<td>chocolate</td>
<td>dark brown</td>
<td>5.25</td>
</tr>
</tbody>
</table>

```sql
<table>
<thead>
<tr>
<th>TID</th>
<th>Cashier</th>
<th>Flavor</th>
<th>Color</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Baskin</td>
<td>strawberry</td>
<td>pink</td>
<td>3.55</td>
</tr>
<tr>
<td>2</td>
<td>Robin</td>
<td>chocolate</td>
<td>light brown</td>
<td>4.75</td>
</tr>
<tr>
<td>3</td>
<td>Baskin</td>
<td>chocolate</td>
<td>dark brown</td>
<td>5.25</td>
</tr>
<tr>
<td>4</td>
<td>Baskin</td>
<td>strawberry</td>
<td>pink</td>
<td>5.25</td>
</tr>
<tr>
<td>5</td>
<td>Robin</td>
<td>bubblegum</td>
<td>pink</td>
<td>4.75</td>
</tr>
<tr>
<td>6</td>
<td>Robin</td>
<td>chocolate</td>
<td>dark brown</td>
<td>5.25</td>
</tr>
</tbody>
</table>
```
Inner Join

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

```sql
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

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

```sql
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

```sql
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
```
Computational Structures in Data Science

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;').fetchall()
Out[66]: <sqlite3.Cursor at 0x11117960>

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";').fetchall()
Out[68]: [(2, 'chocolate', 'light brown', 4.75),
(3, 'chocolate', 'dark brown', 5.25),
(6, 'chocolate', 'dark brown', 5.25)]
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 ;