Computational Structures in Data Science

SQL
Announcements

- Ants Project due Weds!
  - Bonus point for submitting by Tuesday night
- Be on the lookout for a final survey
  - Extra credit for everyone if enough people submit. (TBD on numbers, points)
- Final Exam logistics out soon, but just like the midterm.
  - Tuesday **8AM! (Sorry!)**
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

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;
```

<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 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;
```

```
<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>
</tbody>
</table>
```
SQL: Filtering Queries
Filtering rows - WHERE

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

\[
\text{SELECT [columns] FROM [table] WHERE [condition] [ORDER BY order];}
\]
SQL Operators for predicates

- 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.

```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.

```python
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>
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
• Can create a table in many ways
  – Often may just supply a list of columns without data.
• 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);
ssqlite> 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
```

- 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!
SQL: Aggregations
Grouping and Aggregations

- 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')
<table>
<thead>
<tr>
<th>Flavor</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>bubblegum</td>
<td>1</td>
</tr>
<tr>
<td>chocolate</td>
<td>3</td>
</tr>
<tr>
<td>strawberry</td>
<td>2</td>
</tr>
</tbody>
</table>
```

```
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)
<table>
<thead>
<tr>
<th>Flavor</th>
<th>Price mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>bubblegum</td>
<td>4.75</td>
</tr>
<tr>
<td>chocolate</td>
<td>5.083333</td>
</tr>
<tr>
<td>strawberry</td>
<td>4.4</td>
</tr>
</tbody>
</table>
```

```
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**

```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", 3 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>
**Inner Join**

```
SELECT * FROM sales, cones WHERE cone_id = cones.id;
```

When column names conflict we write: `table_name.column_name` in a query.

```
sqlite> SELECT * FROM cones, sales WHERE cone_id=cones.id;
Id|Flavor|Color|Price|Cashier|id|cone_id
1|strawberry|pink|3.55|Baskin|3|1
1|strawberry|pink|3.55|Robin|6|1
2|chocolate|light brown|4.75|Baskin|1|2
2|chocolate|light brown|4.75|Baskin|4|2
2|chocolate|light brown|4.75|Robin|5|2
3|chocolate|dark brown|5.25|Robin|2|3
```
Putting It All Together:

- Which of our cashiers sold the highest value of ice cream?
- First we need to find which cones were sold by whom, then we SUM() the results!

```sql
sqlite> SELECT Cashier, SUM(Price) as 'Total Sold' FROM sales, cones WHERE sales.cone_id = cones.id GROUP BY Cashier;
Cashier|Total Sold
Baskin|13.3
Robin|13.8
```
SQL: using named tables - FROM

```
SELECT "delicious" as Taste, Flavor, Color FROM cones
    WHERE Flavor is "chocolate"
UNION
SELECT "also tasty", Flavor, Color FROM cones
    WHERE Flavor is not "chocolate";
```
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
```
Summary

```
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> ;
```