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

Database Management Systems

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!

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

• 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
• 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`

**SQL example**

- SQL statements create tables
  - Give it a try with sqlite3 or http://kripken.github.io/sql.js/GUI/
  - Each statement ends with ‘;’

```bash
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.
sqlite> select 38 as latitude, 122 as longitude, "Berkeley" as name;
38|122|Berkeley
sqlite>
```

**A Running example from Data 8 Lec 10**

```sql
# An example of creating a Table from a list of rows. Table({"Flavor": "Strawberry", "Color": "Pink", "Price": 3.55},
{"Chocolate": "Light Brown", 4.75},
{"Chocolate": "Dark Brown", 5.25},
{"Strawberry": "Pink", 3.55},
{"Bubblegum": "Pink", 4.75})
```

**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", 4.25 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.
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];
```

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

Permanent Data Storage

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

SQL uses the following binary operators, in order from highest to lowest precedence:

```
<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>null</code></td>
<td>NULL value</td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td>Less than</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>Less than or equal to</td>
</tr>
<tr>
<td><code>=</code></td>
<td>Equal to</td>
</tr>
<tr>
<td><code>&lt;&gt;</code></td>
<td>Not equal to</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>Greater than</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td><code>BETWEEN</code></td>
<td>In range</td>
</tr>
<tr>
<td><code>IN</code></td>
<td>In set</td>
</tr>
<tr>
<td><code>LIKE</code></td>
<td>Pattern match</td>
</tr>
<tr>
<td><code>IS</code></td>
<td>Not in set</td>
</tr>
<tr>
<td><code>NOT</code></td>
<td>Not in set</td>
</tr>
</tbody>
</table>
```

Supported logical operators are these:

- `AND`  
- `OR`
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|3.25

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'))
```

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.

```
sqlite> select count(*) from cones by Flavor;
```

Unique / Distinct values

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

Joining tables

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

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

Inner Join

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

SQL: using named tables - from

```sql
select "delicious" as Taste, Flavor, Color from cones
where Flavor = "chocolate" union
select "other", 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

```sql
select TID from sales where Cashier is 'Bank1';
select * from cones
  where ID in (select TID from sales where Cashier is 'Bank1');
sqlite> select * from cones
...
  where ID in (select TID from sales where Cashier is 'Bank1');
```.triggered

11/26/18 UCB CS88 Sp18 L13

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,...);
```

Multiple clients of the database

- All of the inserts update the common repository

DB Abstraction (cont)

```python
class SQL_Table(Table):
    ''' Extend Table class with methods to read/write a Table from/to a table in a SQLite database. '''
    def read(cls, filepath, table, verbose=False):
        '''Create a SQL_Table by reading a table from a SQLite database.'''
        dbconn = sqlite3.connect(filepath, detect_types=sqlite3.PARSE_COLNAMES)
        col_names = [sqlcol_names(dbconn, table)]
        rows = sqlasarray(dbconn, SELECT * FROM %s % table, verbose).fetchall()
        return cls(col_names, with_rows=rows)
```
**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