Lecture #12: SQL

Btw. CNN criticizing my work…


Question

Next lecture…
A) Q&A Session
B) Fun Lecture
C) Review Lecture
D) All of the above

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 ‘;’

```
create table
```

A Running example from Data 8 Lec 10

# An example of creating a Table from a list of rows.

```
create table
```

```c
CREATE TABLE flavors (Flavor TEXT, Color TEXT, Price REAL);
```

```
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;
```
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.

Projecting existing tables

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

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

Permanent Data Storage

```
sqlite> select Flavor, Price from cones;
Flavor|Price
-----|-----
strawberry|3.55
chocolate|4.75
chocolate|5.25
bubblegum|4.75
```

Filtering rows - `where`

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

04/22/19  UCB CS66 Sp19 L12

04/22/19  UCB CS66 Sp19 L12
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.

<table>
<thead>
<tr>
<th>SQL operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
</tr>
<tr>
<td>&lt;</td>
</tr>
<tr>
<td>&lt;=</td>
</tr>
<tr>
<td>IS NULL</td>
</tr>
<tr>
<td>AND</td>
</tr>
</tbody>
</table>

Supported unary operators are:
- + | - | NOT

Approximate Matching ...

Regular expression matches are so common that they are built in to SQL. Update or modify the columns to express what you mean:
```
sales where coins 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>

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.

Unique / Distinct values

- Built in to the language or a composable tool?

Joining tables

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

Inner Join

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

**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 "Rashid";
```

**Inserting new records (rows)**

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

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

**Multiple clients of the database**

- All of the inserts update the common repository

**SQLite python API**

```
import sqlite3

icecream = sqlite3.connect('icecream.db')

icecream.execute('SELECT * FROM cones;')

for row in icecream.execute('SELECT DISTINCT Flavor FROM cones;').fetchall():
    print(row)

icecream.execute('SELECT * FROM cones WHERE Flavor is "chocolate";').fetchall()
```

**Creating DB Abstractions**

```python
class SQL_Table(Table):
    """
    Extend Table class with methods to read/write a Table from/to a table in a SQLite database.
    """

class_table = SQL_Table()

def read_sql(filepath, table, verbose=False):
    """Create a SQL_Table by reading a table from a SQL database."""

dbcnn = sqlite3.connect(filepath, detect_types=sqlite3.PARSE_COLNAMES)

col_name = sqlcol_names(dbcnn, table)

col = sqlgetColumn(dbcnn, SELECT + from col_name)  # table, verbose).fetchall()
dbcnn.close()
return col[col_name] + rows(rows)
```
DB Abstraction (cont)

```python
class SQL_Table(Table):
    ... """Write a Table into a SQL database as a SQL table."""
    def write(self, filepath, table, overwrite=False, overwrite=True):
        # Create table and insert each row
        cols = build_list(self.labels)
        sql = sql.get()  # CREATE TABLE % % WHERE % %
        for row in self.rows:
            sql()  # INSERT INTO % VALUES %
            dconn.commit()
        dconn.close()

# Example:
def cast(cols, table):
    return cols, with_columns(zip(table.labels, table.columns))
```

Summary – Part 1

- **SQL** a declarative programming language on
  relational tables
  - largely familiar to you from data
  - 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

Summary

- SQL a declarative programming language on relational tables
  - largely familiar to you from data
  - 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

Solutions for the Wandering Mind

1) Adding two n-bit integers, how many bits can the result have?
   Maximally n+1 bits (1 overflow bit).
2) Multiplying two n bit integers, how many bits can the result have?
   log a*b=log a+log b=>log n*n=2*log n. 2*n bits.
   Assume:
   a) Exceptions don’t exist
   b) We only reserve 8bit for an integer variable (0–255)
   Questions:
   1) What would be the result of an addition 255+255?
      (255+255) modulo 255=0. One entire summand is lost!
   2) What would be the result of a multiplication 255*255?
      (255*255) modulo 255=0. The error is: 64770!
3) Assume l additions of 8bit integers into the same 8bit variable. Can you formulate the maximum error that can occur as a function of l?
   Each addition can maximally lose the entire 8bits. The error accumulates. This is, each addition loses maximally 8bits. So it’s l*8bits of loss. So the maximum error 255*l.
   For multiplication it’s 255. This is, exponential error!
   (Also compare Lyapunov Exponent in physics)
   This is taught in CS61C