Relations and SQL (Reading: Ch 9)

Learning goals:

- Learn the vocabulary used to describe data stored in databases
- Learn SQL joins, predicates, sampling, and subqueries
- Understand when to use SQL and when to use Python

UC Berkeley Data 100 Summer 2019
Sam Lau

(Slides adapted from John DeNero)

Announcements

- HW2 due tomorrow (July 9)
  - We will not mark points off if your graph looks slightly different, only if the overall pattern is different
- HW3 due Friday (July 12)
- **Midterm next Tues!**
  - 9:30am - 11am Tues, July 16 in 105 North Gate
  - Single hand-written two-sided cheat sheet allowed
  - Covers weeks 1-3 of class
  - DSP students: We will email you today
- Small group tutoring this week: bit.ly/d100-tutor

Database Schemas

Relations

A table with rows and named columns:

- Each row is called a *record* or *tuple*.
- Each column is called an *attribute* or *field* and has a name and data type.
- Software enforces a fixed set of generic types (strings, integers, etc.), but conceptually a data type describes a semantic domain (names, ages, etc.)
- Classically, a relation is an unordered set of tuples, and the attributes of a tuple are unordered, but in SQL both rows and columns are ordered.

Relations

Pandas Dataframes allow more than just relational operations. E.g., transpose.

Tables in a relational database (e.g., sqlite, mysql, postgres) represent relations.

- Maximizes efficiency of relational operators (group, sort, select, join, etc.)
- Allows for concurrent access and updates from multiple users.

Schemas

A database schema describes all relations and their attribute names & types.

- Determines **granularity**: what does one record in each table represent?
- Determines **primary and foreign keys**: what tables are linked?
- Determines **representation**: what data types will be used to store attributes?

```
CREATE TABLE users(
    id INTEGER PRIMARY KEY,
    name TEXT
)
CREATE TABLE orders(
    item TEXT PRIMARY KEY,
    price NUMERIC,
    name TEXT
)
```

You should know SELECT, FROM, WHERE

- We will skip SQL basics since it was covered in CS61A!
- For review, see the CS61A textbook.
- For example, you should understand the following query:

```
SELECT name, latitude, temp
FROM cities, temps
WHERE name = city;
```

You should know GROUP BY and HAVING

And this query should feel familiar:

```
SELECT max(name), legs, weight FROM animals
GROUP BY legs, weight
HAVING max(weight) < 100;
```

Pandas equivalent:

```
(animals.groupby(['legs', 'weight'])
 .filter(lambda g: g['weight'].max() < 100)
 .loc[:, ['name', 'legs', 'weight']]
 .groupby(['legs', 'weight'])
 .max())
```

Types of Joins

Cross Join

All pairs of rows appear in the result.

s

M	U
1	W
2	X
3	X
4	Y

t

N	V
A	X
B	X
C	Y
D	Z

M	U	N	V
1	W	A	X
2	X	A	X
3	X	A	X
4	Y	A	X
1	W	B	X
2	X	B	X
3	X	B	X
4	Y	B	X

(to be continued …)

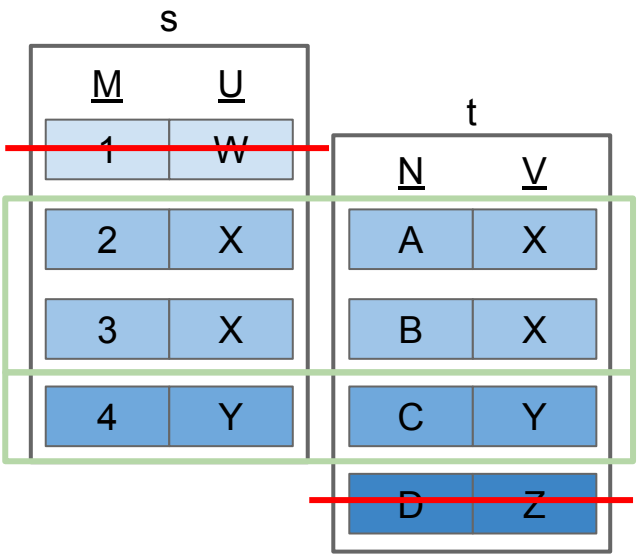
(... continued)

M	U	N	V
1	W	C	Y
2	X	C	Y
3	X	C	Y
4	Y	C	Y
1	W	D	Z
2	X	D	Z
3	X	D	Z
4	Y	D	Z

```
SELECT * FROM s, t;
```

Inner Join

Only pairs of matching rows appear in the result.

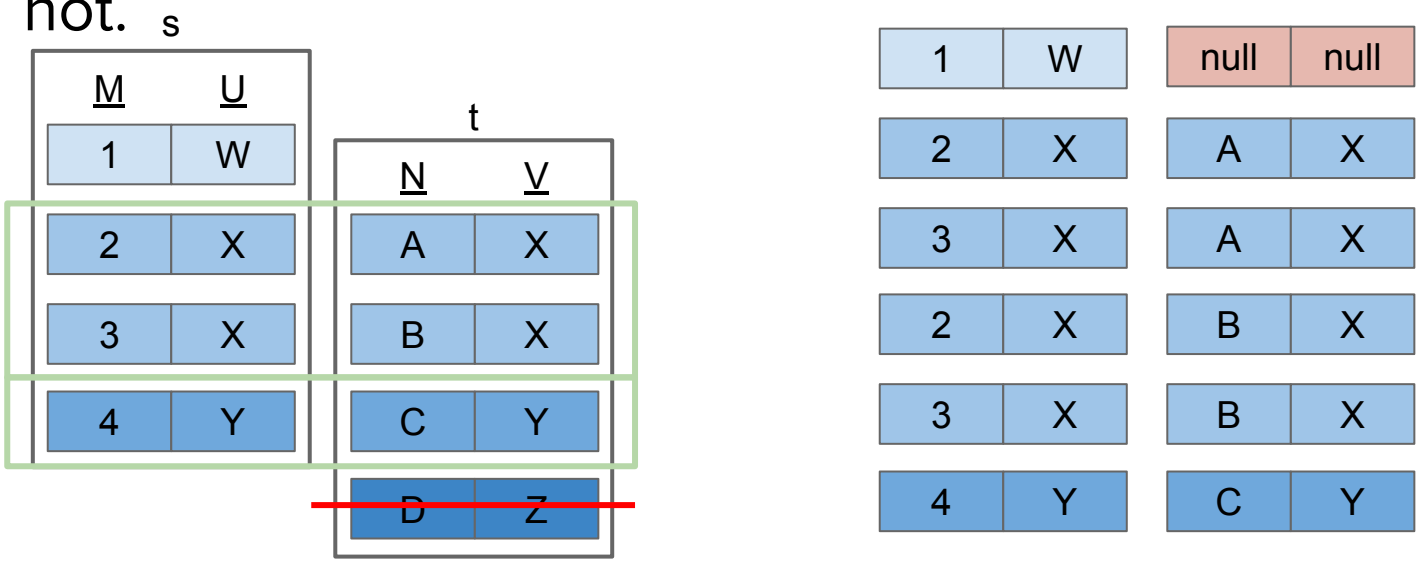


M	U	N	V
2	X	A	X
3	X	A	X
2	X	B	X
3	X	B	X
4	Y	C	Y

```
SELECT * FROM s JOIN t ON s.u = t.v;
SELECT * FROM s INNER JOIN t ON s.u = t.v;
SELECT * FROM s, t WHERE s.u = t.v;
```

Left Outer Join

Every row in the first table appears in the result, matching or not.




```
SELECT * FROM s LEFT JOIN t ON s.u = t.v;
```

(Demo)

Break!
Fill out Attendance:
http://bit.ly/at-d100

SQL and Python

SQL Predicates

In addition to numerical comparisons (=, <, >), SQL has built-in predicates.

The `IN` operator tests whether a value is in a list.

E.g., select rows whose month is either January, March, or May:

```
SELECT * FROM t WHERE t.month IN ('January', 'March', 'May')
```

SQL Predicates

The LIKE operator tests whether a string matches a pattern (baby regex):

E.g. select rows where the time string is on the hour, such as 8:00 or 12:00 pm:

```
SELECT * FROM t WHERE t.time LIKE ‘%:00%’;
```

Large Datasets

Datasets that are too large to fit in a Pandas DataFrame can often fit in a local database, such as Sqlite

- Databases are highly optimized for storage efficiency.
- **Databases are limited by disk size, Python is limited by memory size**

Common pattern: large datasets are manipulated in SQL until the domain of interest is reached; remaining analysis is carried out in another language (such as Python).

(Demo)

Sampling Rows

Sampling Rows in SQL

In SQLite, a simple random sample of 10 rows can be drawn like this:

```
SELECT * FROM t ORDER BY RANDOM() LIMIT 10;
```

What about drawing a cluster sample?

(Demo)

More SQL Syntax

(Demo)

CASE Expressions

A CASE expression chooses among alternative values.

Without a base expression:

```
CASE WHEN born < 1980 THEN 'old'
     WHEN born < 2000 THEN 'not too old'
     ELSE 'young'
     END
```

With a base expression:

```
CASE year % 10 WHEN 0 THEN 'start of decade'
               WHEN 5 THEN 'middle of decade'
               END
```

Subqueries

A query within another query can be used to create a temporary table.

In a FROM clause: Describe a table instead of naming it.

E.g., join table u with a simple random sample from table t:

```
SELECT *
FROM
  (SELECT * FROM t ORDER BY RANDOM() LIMIT 10),
  u;
```

Subqueries

In a WHERE clause: Describe a one-column table instead of a list; used with IN.

E.g., select rows in a top-3 most popular month:

```
SELECT *
FROM t
WHERE t.month IN
  (SELECT month
   FROM months
   ORDER BY popularity DESC);
```

Summary

- SQL is a programming language designed for data queries
- SQL databases enable large-scale data processing
  - Databases are limited by disk size while Python is limited by memory size (disk usually much larger)
- Sampling procedures can be directly implemented in SQL
- Subqueries allow composing complex queries