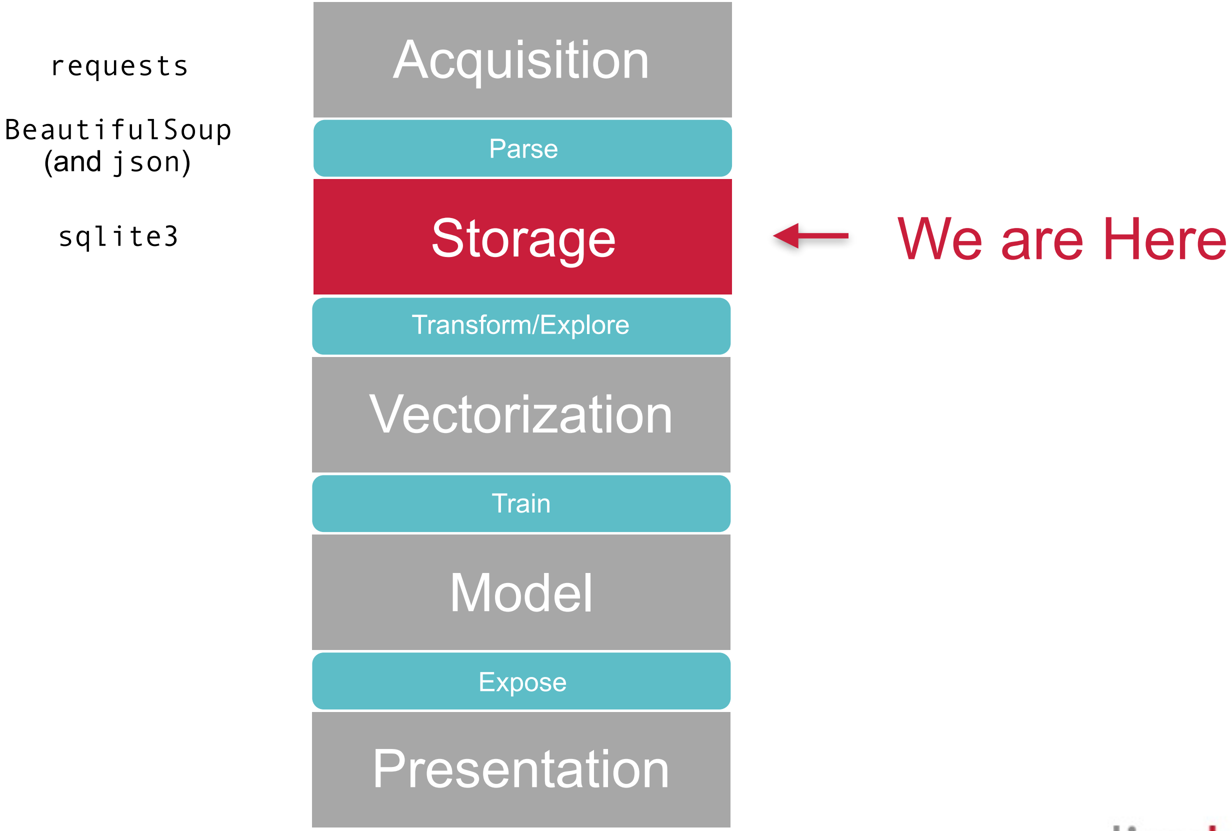
livelessons
video instruction from technology experts

# Lesson 5: Storing Data: Relational Databases (with SQLite)

livelessons

# Process + Tools



`requests` — Acquisition

Parse

`BeautifulSoup` (and `json`)

`sqlite3` — Storage ← We are Here

Transform/Explore

Vectorization

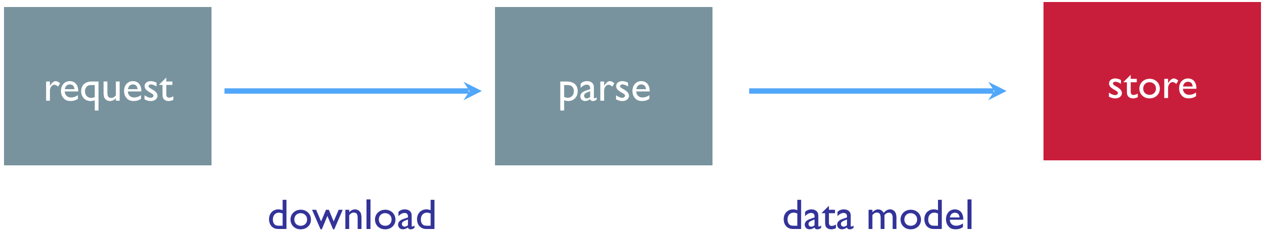
Train

Model

Expose

Presentation

# ETL Pipeline



request

download

parse

data model

store

# Database versus Flat Files

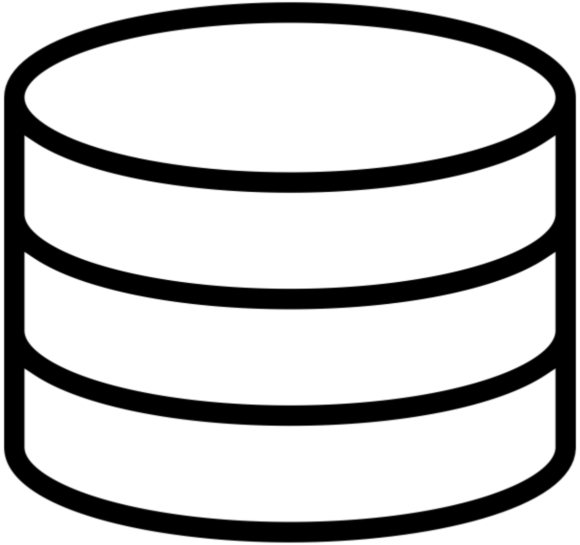
## Relation Database (SQL)

- Enables powerful queries
- Performant (SQL optimizer)
- Often have to run a separate database process/server

## Flat Files

- More portable
- Easy to analyze and inspect
- Needs a programming environment

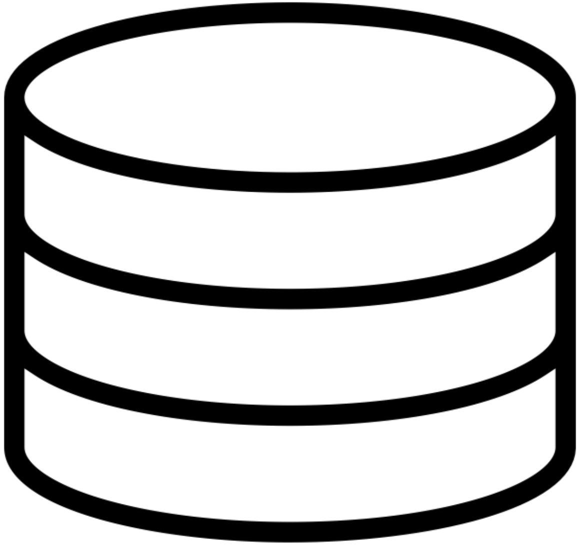
# Relational Databases 101



Created by lastspark
from Noun Project

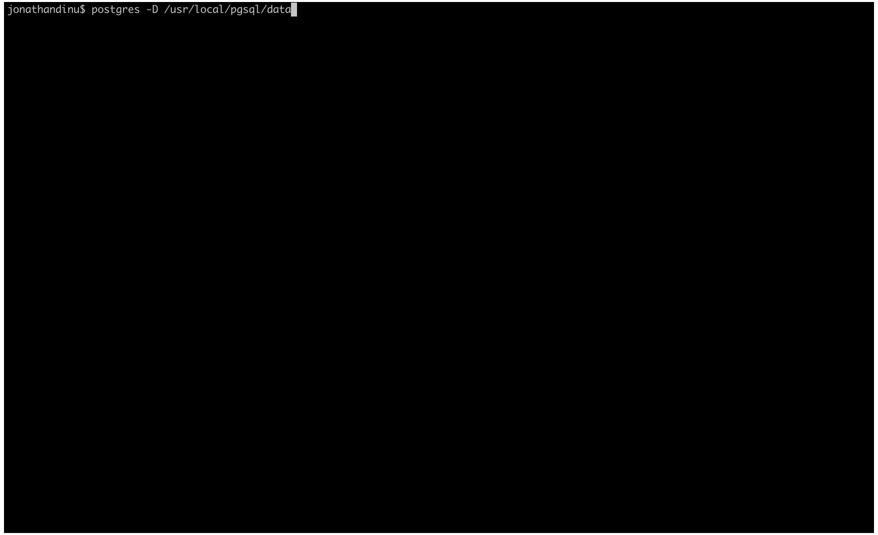
# Relational Databases 101

**Usually runs in separate process (in background)**

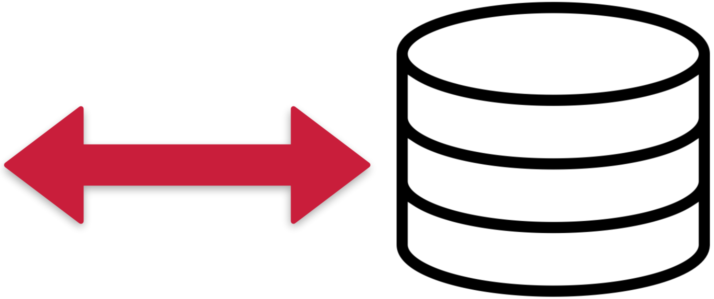


Created by lastspark
from Noun Project

# Relational Databases 101



```
jonathandinu$ postgres -D /usr/local/pgsql/data
```

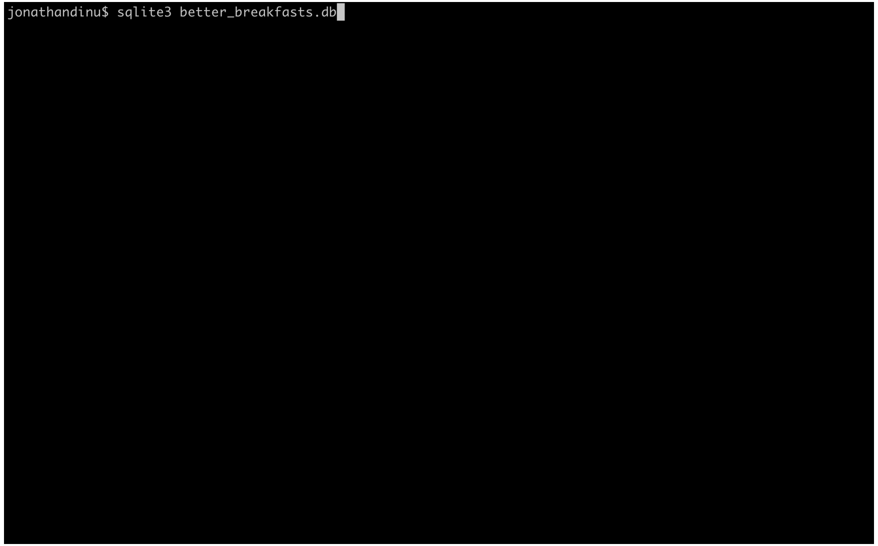


Created by lastspark
from Noun Project

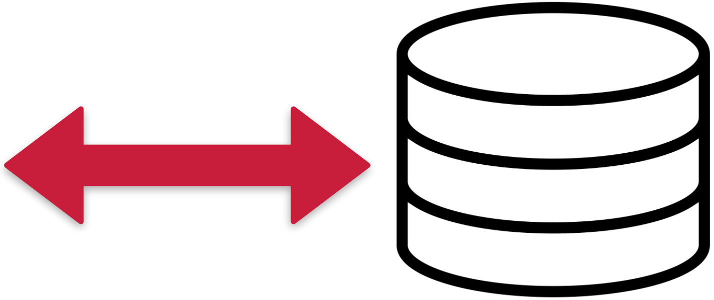
# SQLite

## A Happy Marriage Between the Two (Traditional relational DB and Flat Files)

# SQLite



```
jonathandinu$ sqlite3 better_breakfasts.db
```



Created by lastspark
from Noun Project

# SQLite

## Relation Database (SQL)

- Enables powerful queries
- Performant (SQL optimizer)
- Transactional
- Often have to run a separate database process/server

## Flat Files

- More portable
- Easy to analyze and inspect
- Needs a programming environment

# SQLite

## Relation Database (SQL)

- **Enables powerful queries**
- **Performant (SQL optimizer)**
- **Transactional**
- ~~Often have to run a separate database process/server~~

## Flat Files

- **More portable**
- **Easy to analyze and inspect**
- ~~Needs a programming environment~~

# SQLite

- Portable (cross platform, single `.db` file)
- Easy to analyze and inspect: sqlite shell, programming APIs, human readable DB file format
- Powerful, performant (SQL optimizer), and transactional queries
- Zero-Configuration and Serverless
- Compact and Embeddable
- Public Domain (open source)

# Interfacing with a Database

## SQL

- Command Line Shell
- Graphical Browser

## Programmatic

- Database Adapter
- Object Relational Mapper
- REST interface*

*not every database has a built-in REST interface

# Database Types

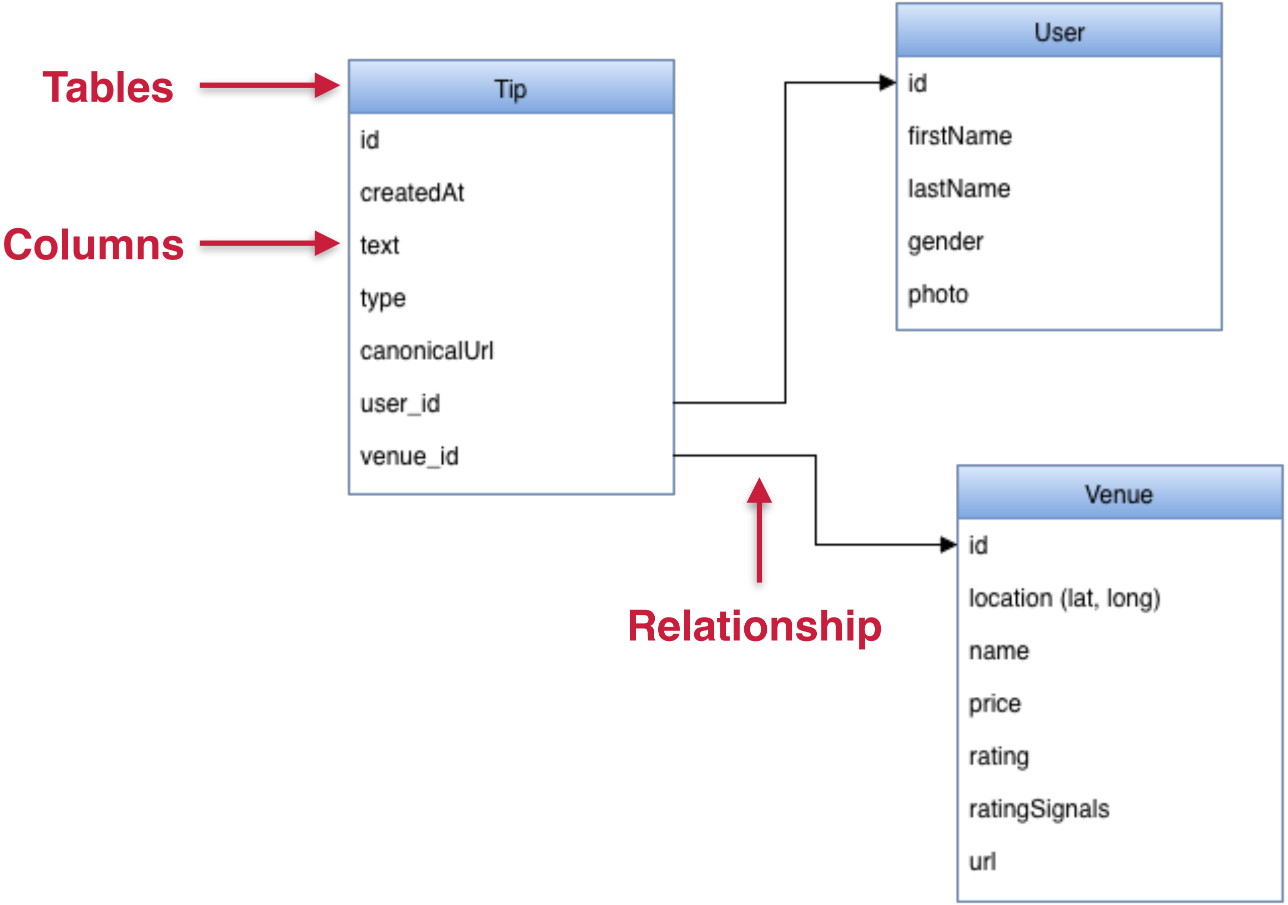
| Type | Advantage | Disadvantage | Example |
|---|---|---|---|
| Graph | Complex relations and optimized graph & network operations | Can be overkill for independent (or unrelated) data | Neo4j, OrientDB |
| Relational | Transactions, joins, & expressive queries | Inflexible schemas & can be difficult to distribute/scale | Postgres, MySQL |
| Document | Flexible data model, hierarchical, and often full-text search | No (true) schema, joins or transactions | MongoDB, CouchDB |
| Columnar | Scalable with optimized analytic operations (OLAP) | Can be complex and often distributed | HBase, Bigtable |
| Key Value | Performant and simple to work with | Limited data model and querying | Redis, Dynamo |

# What’s in a Schema

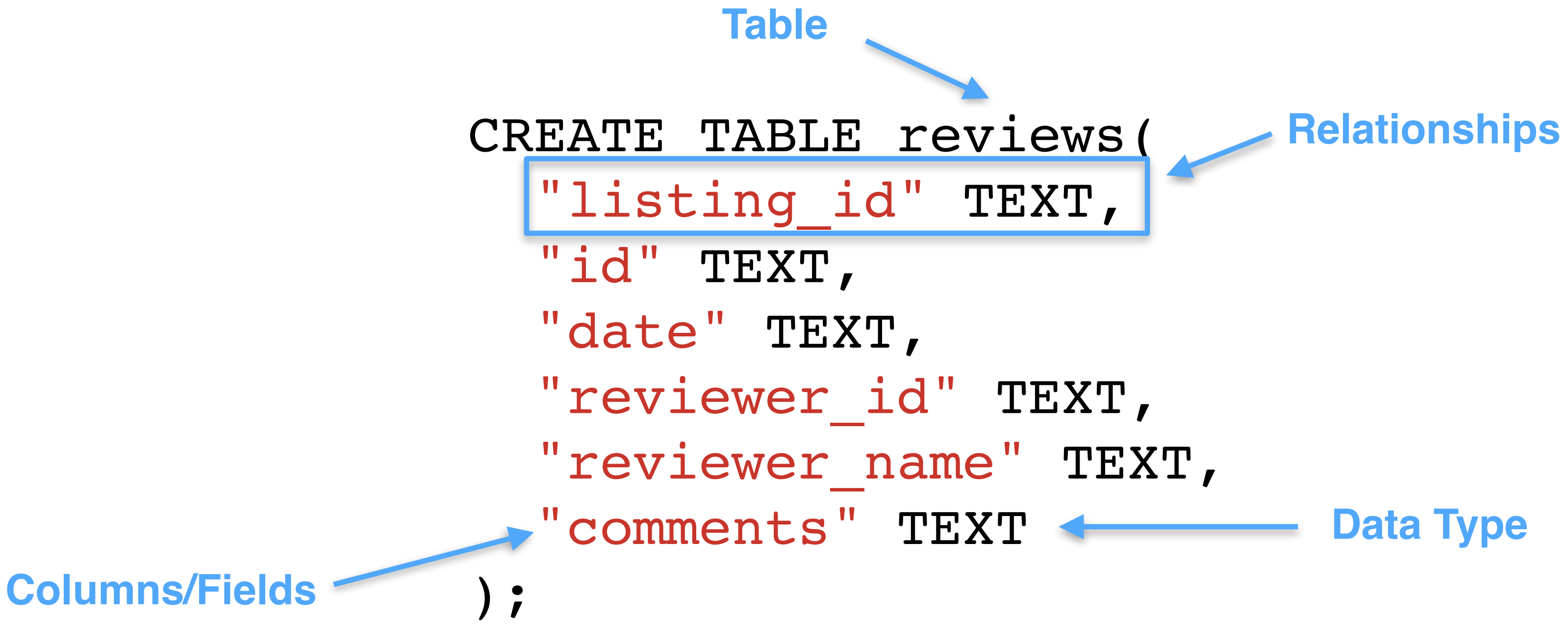
> *The term "schema" refers to the organization of data as a blueprint of how the database is constructed (divided into database tables in the case of relational databases).*

- Wikipedia

# Schema


Tables
Columns
Relationship
Tip
id
createdAt
text
type
canonicalUrl
user_id
venue_id
User
id
firstName
lastName
gender
photo
Venue
id
location (lat, long)
name
price
rating
ratingSignals
url

# What’s in a Schema



Table

Relationships

```
CREATE TABLE reviews(
   "listing_id" TEXT,
   "id" TEXT,
   "date" TEXT,
   "reviewer_id" TEXT,
   "reviewer_name" TEXT,
   "comments" TEXT
);
```

Data Type

Columns/Fields

# Data Types

## Discrete

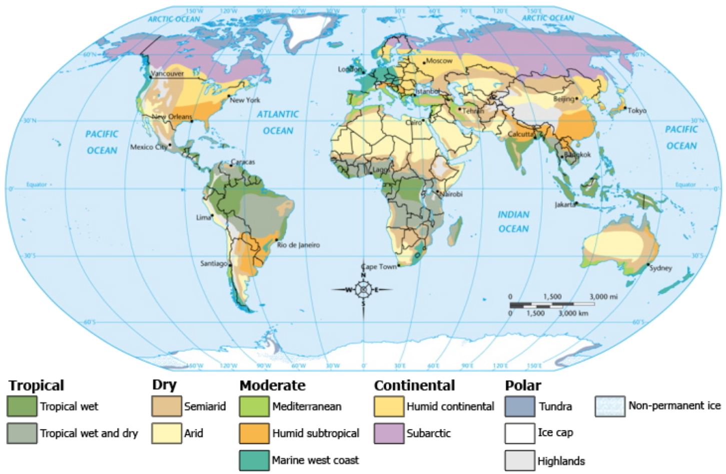
- Countably Finite
- Often human artifacts
- Finite Granularity
- Ex: weight class, climate, date

## Continuous

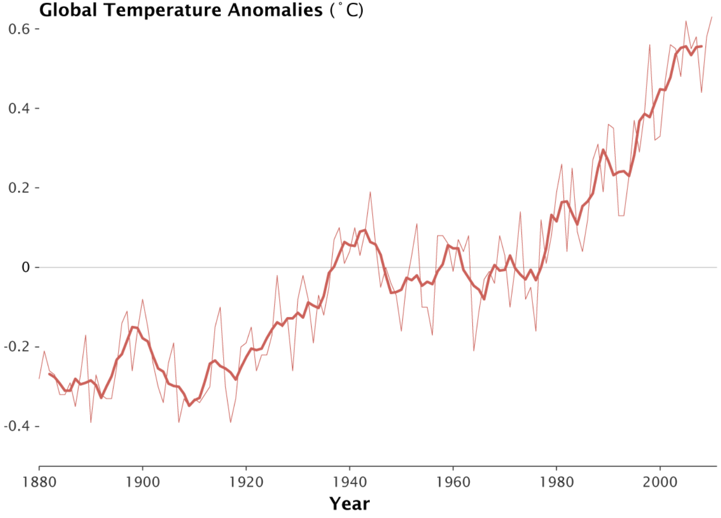
- Uncountably (even in finite interval) real valued.
- Typically represent natural phenomenon
- Can always get more precise
- Ex: mass, temperature, time

# Data Types

## Discrete

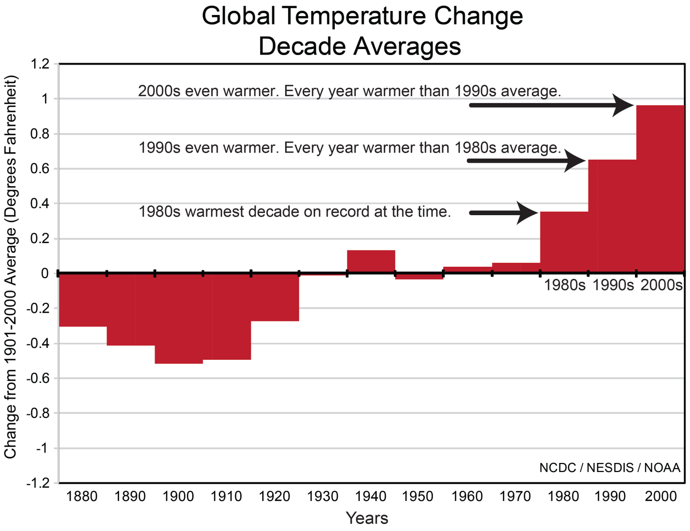


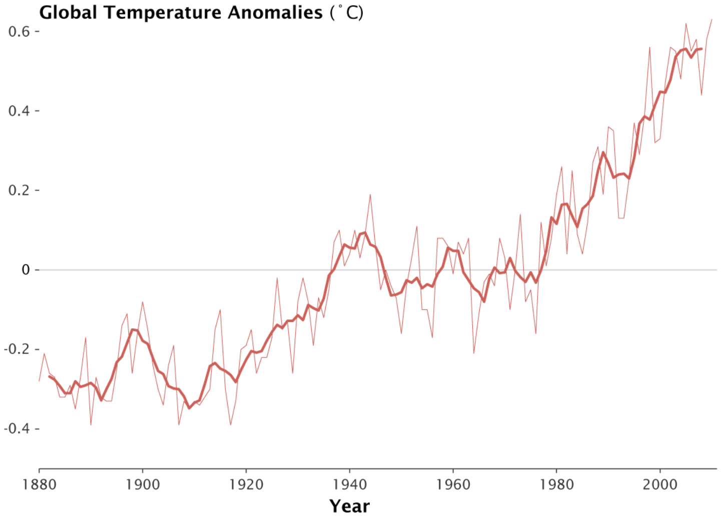
## Continuous



# Data Types

Discrete ← always some loss of information — Continuous





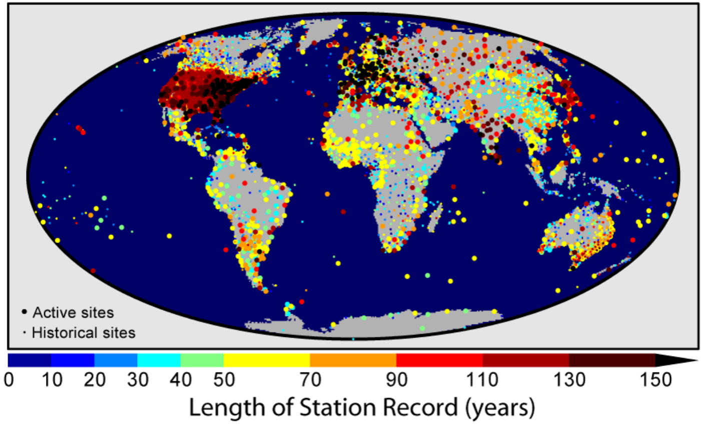
# Data Types

Discrete

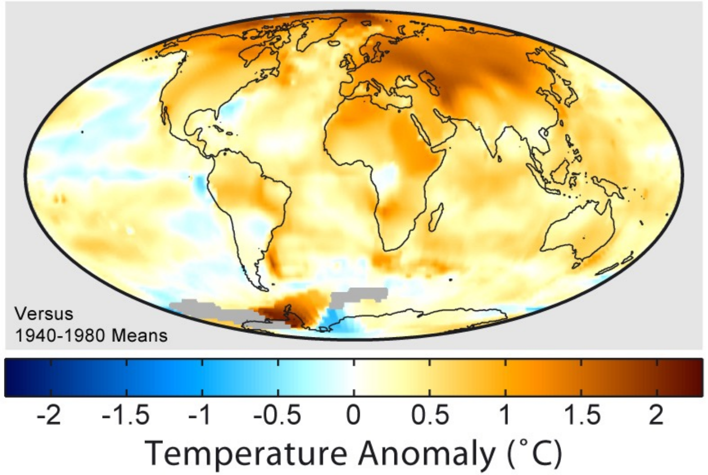
always error/bias in estimation

Continuous

Global Climate Network Temperature Stations



1999-2008 Mean Temperatures



# Data Types

## Discrete

- integer
- date
- varchar
- text
- uuid
- boolean
- NULL*

## Continuous

- real
- float
- decimal/numeric
- timestamp/time

livelessons

# Object Relational Mappers

```
class Venue:
    def __init__(self, **data):
        self.id = data['id']
        self.name = data['name']
        self.price = data['price']
        self.rating = data['rating']
        self.ratingSignals = data['ratingSignals']
        self.url = data['url']
        self.latitude = data['latitude']
        self.longitude = data['longitude']
```

## Venue Table

| id | name | price | rating | rating Signals | url | lat | long |
|---|---|---|---|---|---|---|---|
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |

```
class User:
    def __init__(self, **data):
        self.id = data['id']
        self.firstName = data['firstName']
        self.lastName = data['lastName']
        self.gender = data['gender']
        self.photo = data['photo']
```

## User Table

| id | firstName | lastName | gender | photo |
|---|---|---|---|---|
| | | | | |
| | | | | |
| | | | | |
| | | | | |

# Object Relational Mappers

```
cafe = Venue(**v).save()
bar = Venue(**w).save()
```

### Venue Table

| id | name | price | rating | rating Signals | url | lat | long |
|---|---|---|---|---|---|---|---|
| 1 | cafe | ... | ... | ... | ... | ... | ... |
| 2 | bar | ... | ... | ... | ... | ... | ... |
| | | | | | | | |
| | | | | | | | |

```
jessica = User(**u).save()
```

### User Table

| id | firstName | lastName | gender | photo |
|---|---|---|---|---|
| 1 | Jessica | ... | ... | ... |
| | | | | |
| | | | | |
| | | | | |

# Object Relational Mappers (ORM)

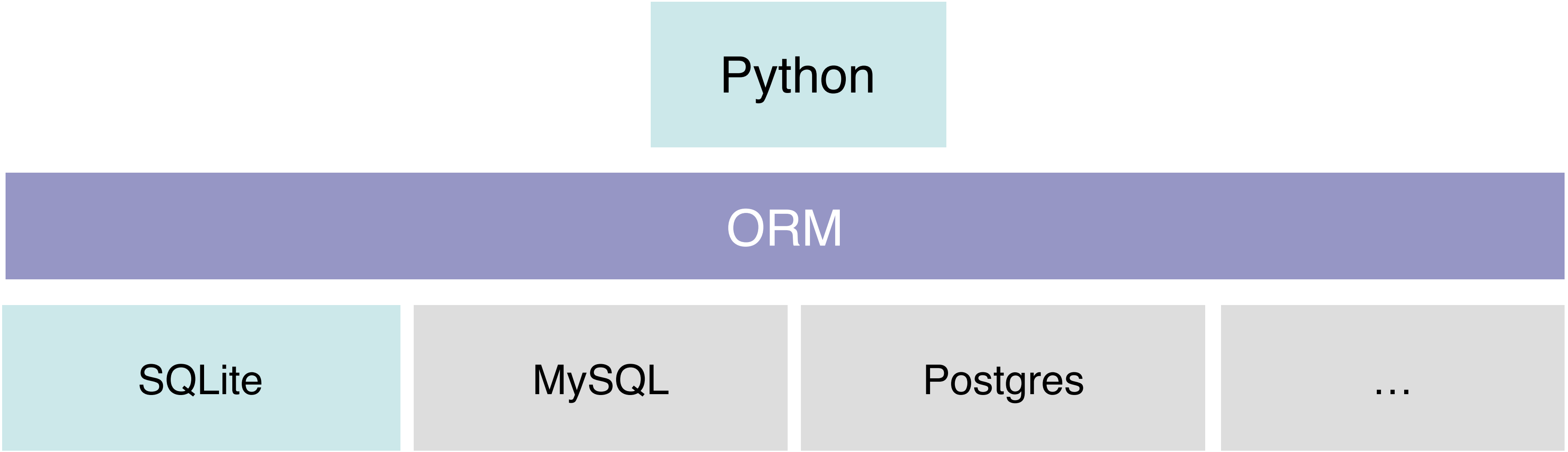
## Advantages

- Abstracts Underlying Processes
- Separation of Concerns (storage infrastructure from application)
- Single Language (and common interface)
- Custom Functions Possible

## Disadvantages

- Abstracts Underlying Processes
- Additional Complexity (more moving parts)
- Object-relational Impedance Mismatch

# Separation of Concerns



Python

ORM

SQLite

MySQL

Postgres

…

# Extract Transform Load (ETL)

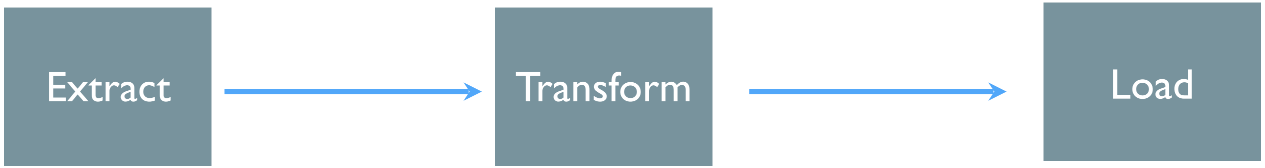
- **Extract** data from homogeneous or heterogeneous data sources

- **Transform** the data for storing it in the proper format or structure for the purposes of querying and analysis

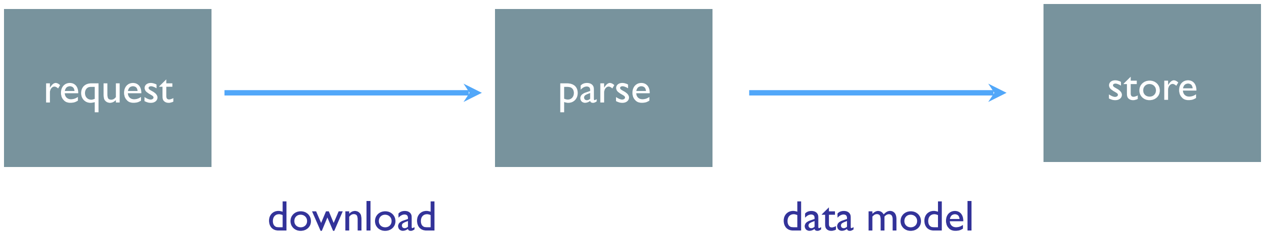
- **Load** it into the final target (database, more specifically, operational data store, data mart, or data warehouse)

source: Wikipedia

# ETL Pipeline



# ETL Pipeline



request

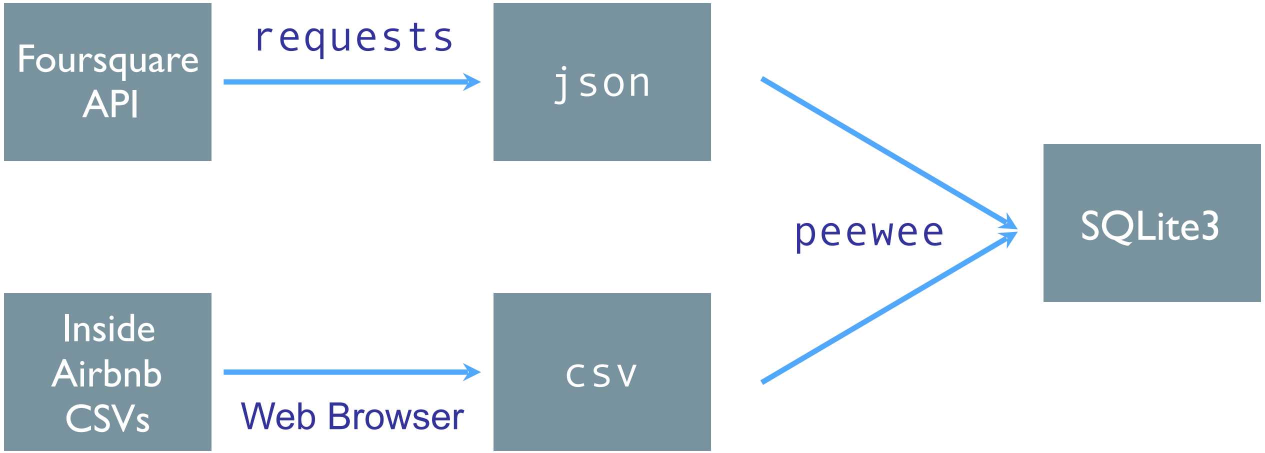
parse

store

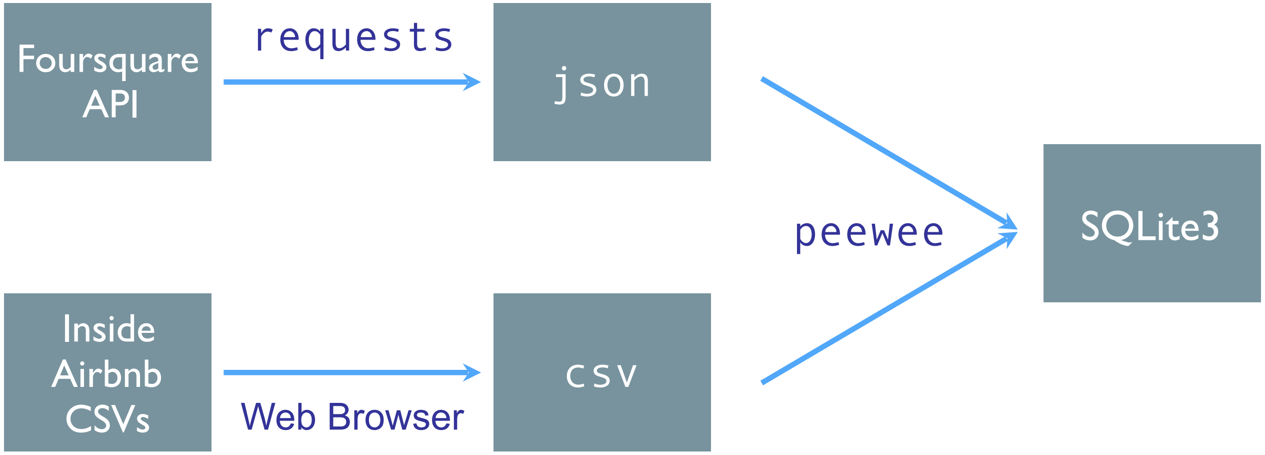
download

data model

# ETL Pipeline



# Data Integration



Foursquare API

requests

json

Inside Airbnb CSVs

Web Browser

csv

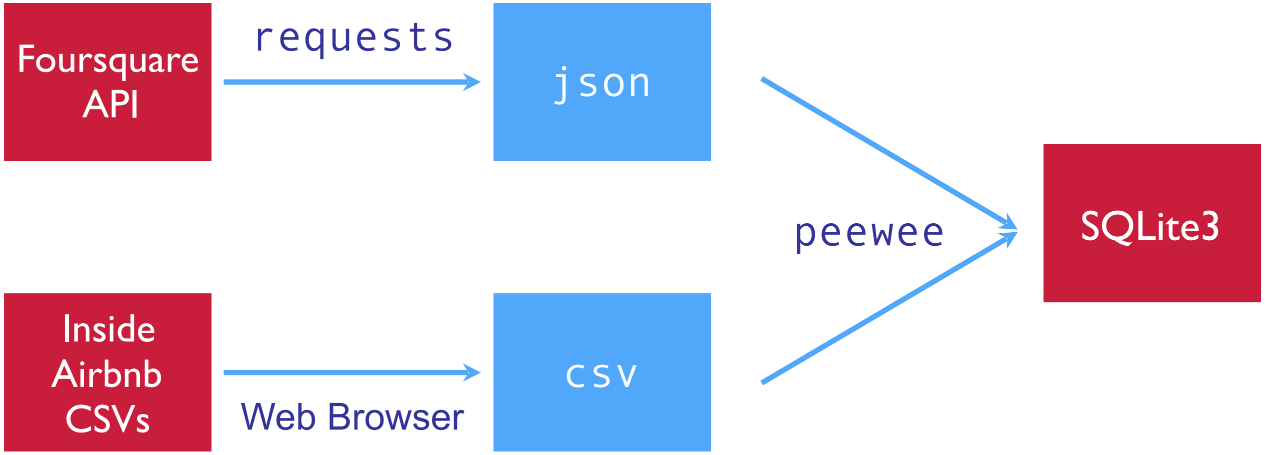
peewee

SQLite3

# Paradigms

🟦 functional

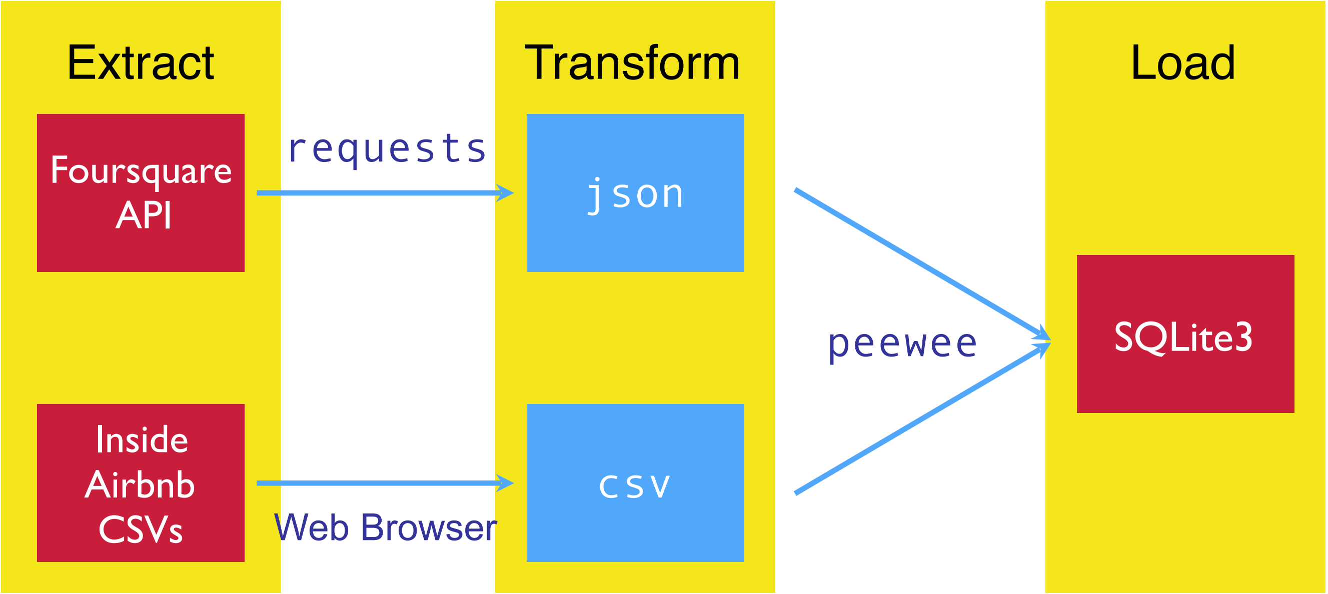
🟥 OOP



# Modules

🟦 functional

🟥 OOP



Extract

Foursquare API

Inside Airbnb CSVs

requests

Web Browser

Transform

json

csv

peewee

Load

SQLite3