

Introduction to Databases

Ivan Corneillet

Data Scientist

Final Project Countdown

Final Project, Part 4

April 26; due next session

Final Project, Part 5

April 28; due in 1 week

In the last class, we focused on exploring time series data and common statistics for time series analysis. In this class, we will advance those techniques to show how to predict or forecast forward from time series data

<i>Unit 1 – Research Design and Data Analysis</i>	<i>Research Design</i>	<i>Data Visualization in Pandas</i>	<i>Statistics</i>	<i>Exploratory Data Analysis in Pandas</i>
Unit 2 – Foundations of Modeling	Linear Regression	Classification Models	Evaluating Model Fit	Presenting Insights from Data Models
<i>Unit 3 – Data Science in the Real World</i>	<i>Decision Trees and Random Forests</i>	<i>Time Series Data</i>	<i>Natural Language Processing</i>	<i>Databases</i>

Learning Objectives

After this lesson, you should be able to:

- Understand uses and differences of databases
- Access databases from *pandas*

Outline

- Review
- Relational Databases
 - Tables
 - Schema
 - Primary Keys
- Normalized vs. Denormalized Structures
- NoSQL (Not-Only SQL) Databases
 - Key-Value databases
 - Document databases
- MongoDB Primer from Eddie
- Codealong + Independent Practice
- Office hours in class for final projects
- Review

DS

Review

Review

- Time-series models use previous values to predict future values, also known as forecasting
- AR and MA model are simple models on previous values or previous errors respectively
- ARMA combines these two types of models to account for both gradual shifts (due to AR models) and abrupt changes (MA models)
- ARIMA models train ARMA models on differenced data to account for non-stationary data
- Note that none of these models may perform well for data that has more random variation
 - For example, for something like iPhone sales (or searches) which may be sporadic, with short periods of increases, these models may not work well



DS

Pre-Work

Pre-Work

Before this lesson, you should already be able to:

- Install *sqlite* with `conda install sqlite`
- Recall and apply dataframe manipulation in *pandas* (subsetting by rows and columns; grouping aggregation with *GroupBy*)

A black circle containing the white text "DS".

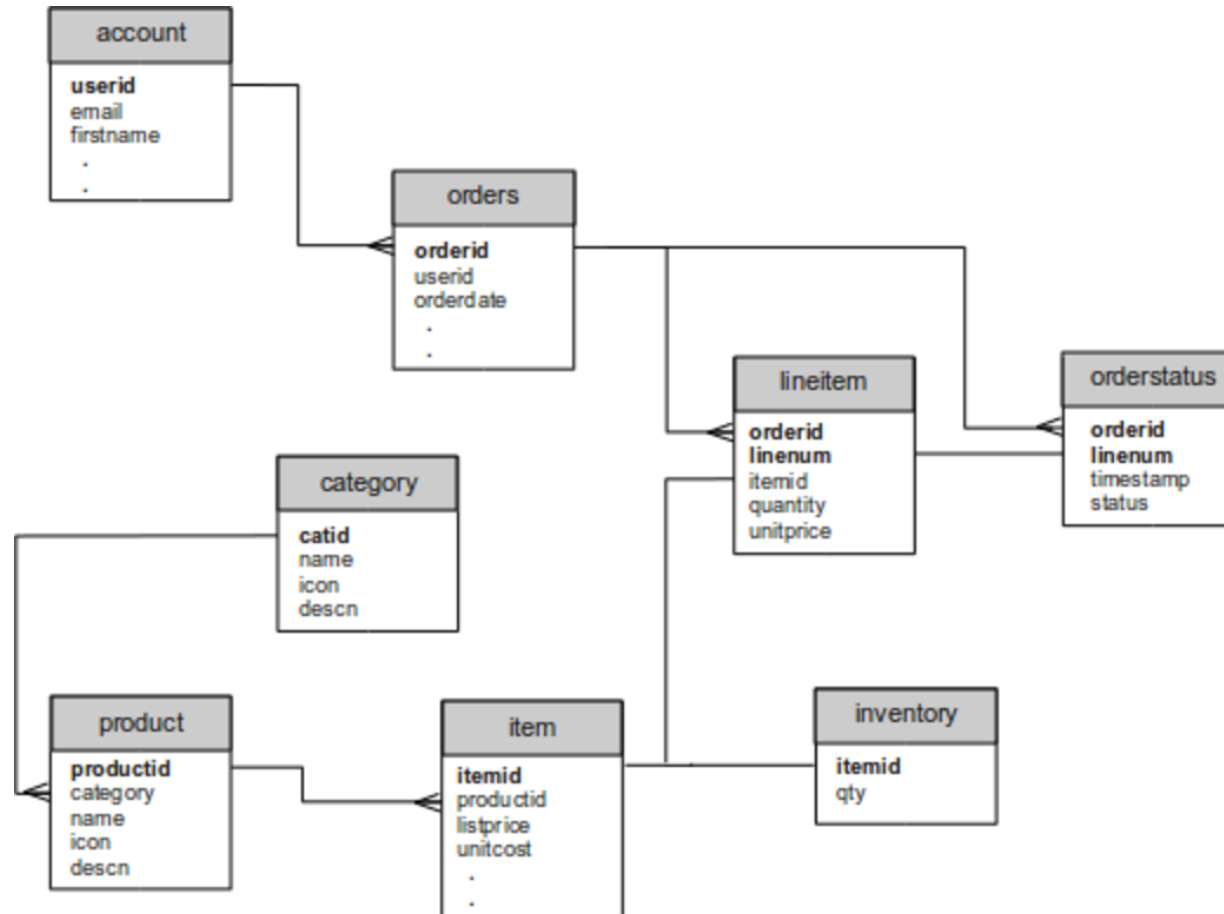
DS

Relational Databases

Databases are the standard solution for data storage

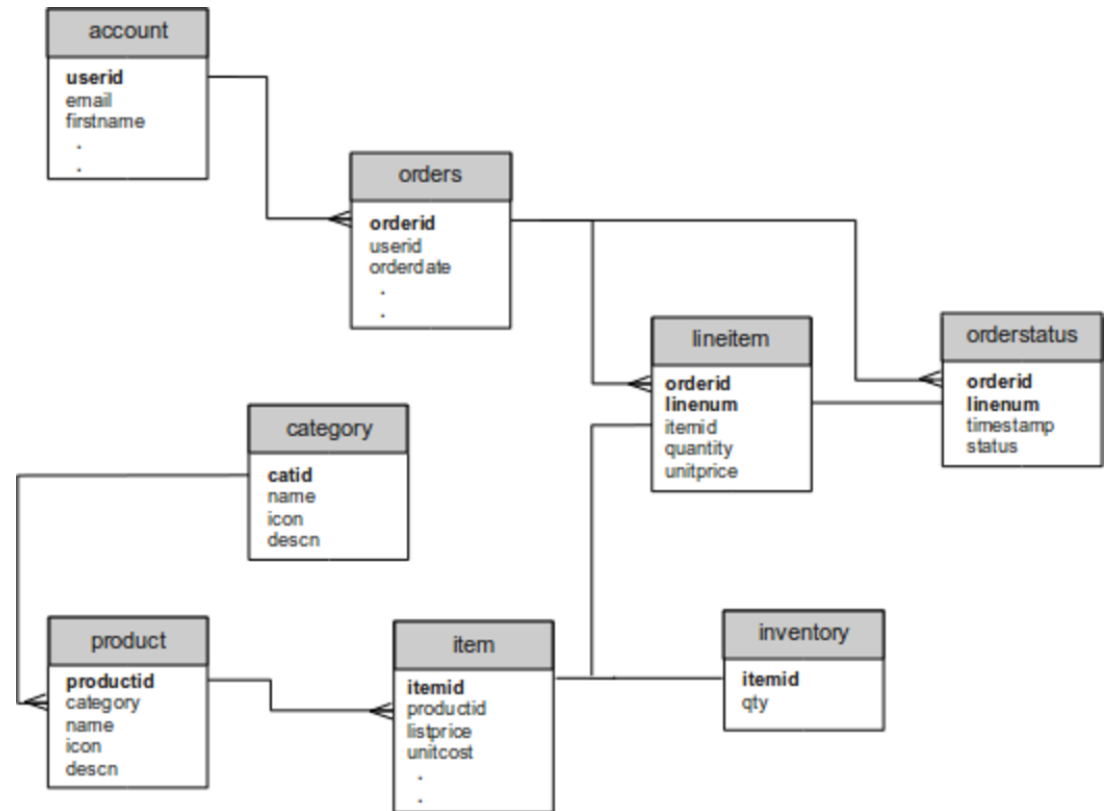
- They provide a way to organize the data on permanent storage (e.g., hard drive and SSD) and efficient methods to retrieve information. They're far more robust and efficient than text and CSV files
- Most analyses typically involve pulling data from a database
- Typically, retrieval is performed using a query language, a mini programming language with a few basic operators for data transformation
- Databases come in many flavors, but we'll explore the most common: relational databases
- Relational databases also come in different varieties, but almost all use SQL (Structured Query Language) as a basis for querying/retrieving data

A relational database links data entities and concepts. E.g., a database for storing orders



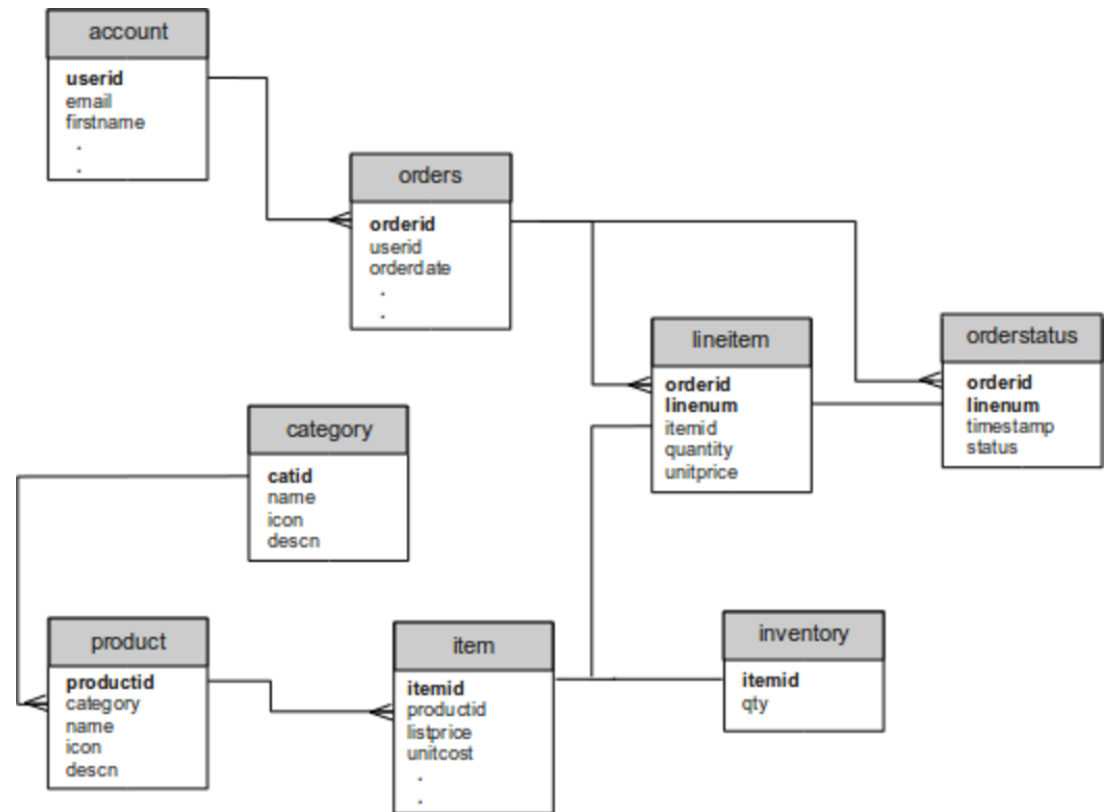
Relational databases are organized into *tables*

- Each table corresponding to one entity or concept
- A table is made up of rows and columns, similar to a *pandas* dataframe



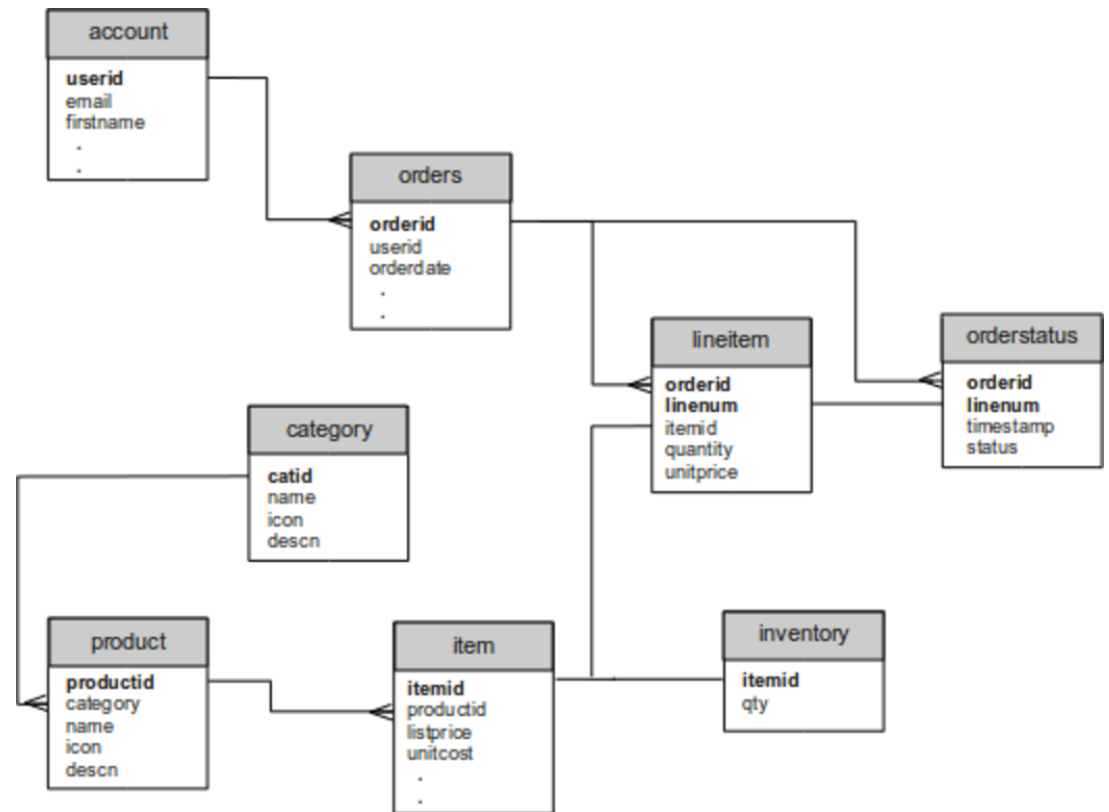
Each table has a specific *schema*, a set of rules for what goes in each table

- These specify which columns are contained in the table and what type of data is in each column (e.g., text, integer, or date). This means you can't add text data to an integer column
- For this reason and many others, databases allow for stronger consistency of the data and are often a better solution for data storage



Each table typically has a *primary key* column

- This column has a unique value per row and serves as the identifier for the row
- A table can have many foreign keys as well. A foreign key is a column that contains values to link the table to the other tables
- These keys that link the table together define the relational database





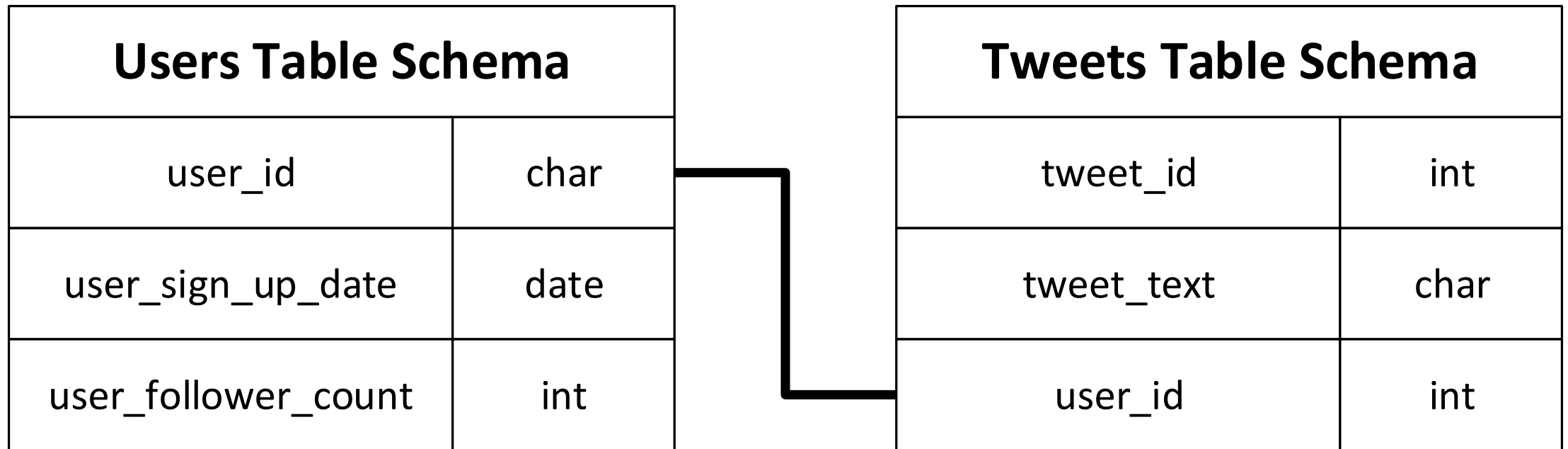
DS

Normalized vs. Denormalized Structures

Once we start organizing our data into tables, we start to separate it into *normalized* and *denormalized* setups

- *Normalized* structures have a single table per entity and use many foreign keys or link tables to connect the entities
- *Denormalized* structures have fewer tables that combine different entities

E.g., if we were Twitter, using a *normalized* structure, we would place users and tweets in different tables



A denormalized structure would put them both in one table

Twitter Table Schema	
tweet_id	int
tweet_text	char
user_id	char
user_sign_up_date	date
user_follower_count	int

Normalized vs. Denormalized Data

Normalized

- Save storage space by separating information
- Requires joining of table to access information about two different entities, a slow operation

Denormalized

- Duplicates a lot of information
- Makes data easy to access since it's all in one table



DS

NoSQL (Not-Only SQL) Databases

“NoSQL” (Not-Only SQL) databases are those that don’t rely on a traditional relational table setup and more flexible in their data organization

- While relational databases are the most popular and broadly used, specific applications may require different data organization
- You don’t need to know every variety, but it’s good to know some overall themes
 - Key-Value Stores
 - Document Databases

Key-Value databases are nothing more than very large and very fast dictionaries

- These are useful for storing key based data (e.g., the last time a customer made a transaction at your e-commerce website)
- Every entry in these databases has two values, a key and a value. We can retrieve any value based upon its key
- This is exactly like a python dictionary, but it can be larger than your computer main memory (i.e., RAM). So these systems use smart caching algorithms to ensure frequently or recently accessed items are quickly accessible
- Popular key-value stores include Cassandra and MemcacheDB (pronounced “mem-cash-dee-bee”)

Document databases may organize data on an entity level, but often have denormalized and nested data setups

NoSQL Data Structure

```
{  
  "user_id": 13123,  
  "user_name": "robby_g",  
  "user_hobbies": ["guitar", "cars"],  
  "user_age": 25  
},  
{  
  "user_id": 19423,  
  "user_name": "jt1235",  
  "user_hobbies": ["football"],  
  "user_age": 31  
}
```

- This nested data layout is often similar to that in JSON documents
- Popular databases include MongoDB and CouchDB

Relational Structure

user_id	user_name	user_hobby_1	user_hobby_2	user_age
13123	robby_g	guitar	cars	25
18423	jt1235	football		31

Relational vs. Document Data Model

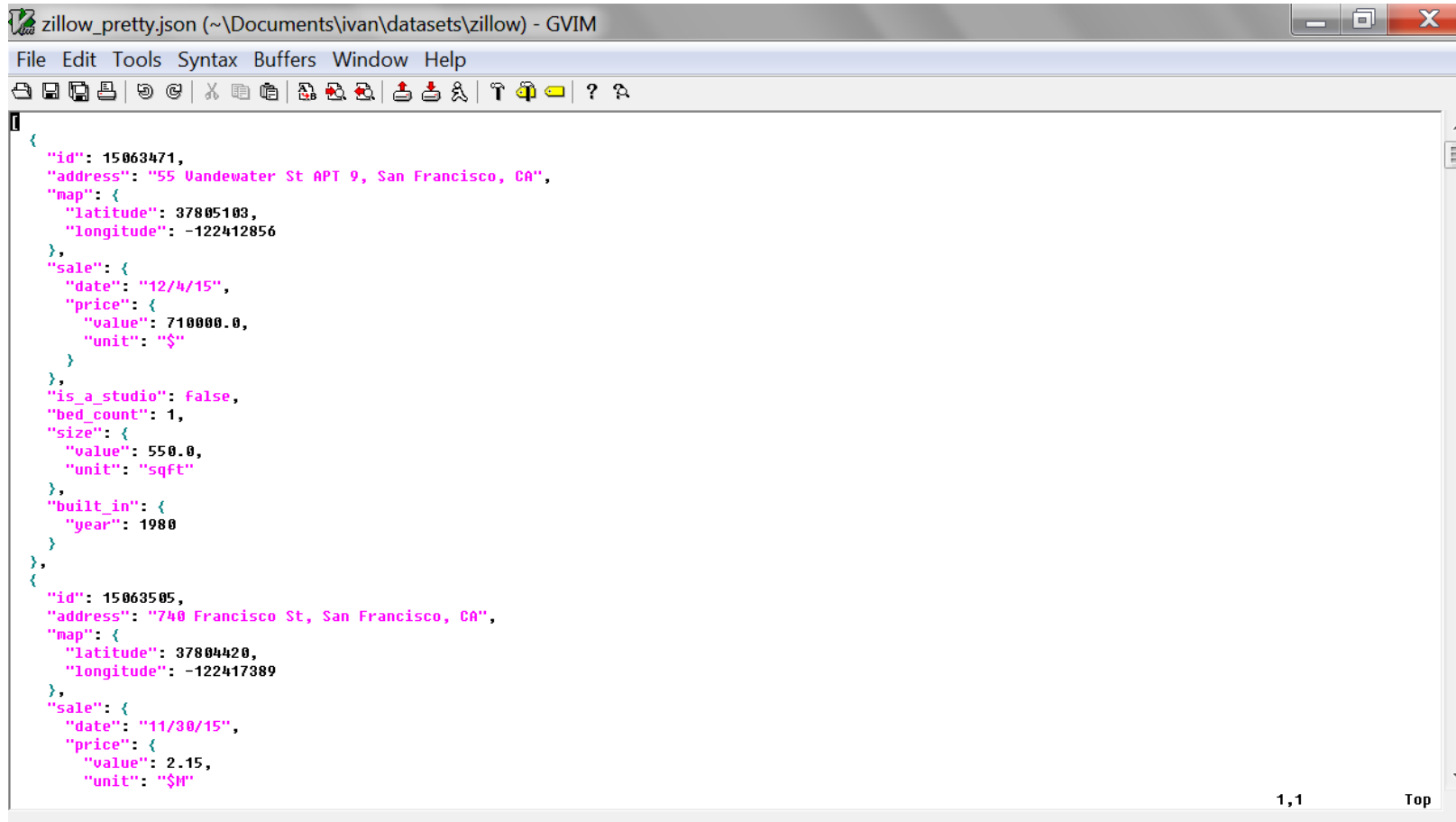
Relational Data Model

- Highly-structured table organization with rigidly-defined data formats and record structure

Document Data Model

- Collection of complex documents with arbitrary, nested data formats and varying “record” format

Recall our JSON file format for our Zillow Real Estate Dataset (session 2)



```
{
  "id": 15063471,
  "address": "55 Vandewater St APT 9, San Francisco, CA",
  "map": {
    "latitude": 37805103,
    "longitude": -122412856
  },
  "sale": {
    "date": "12/4/15",
    "price": {
      "value": 710000.0,
      "unit": "$"
    }
  },
  "is_a_studio": false,
  "bed_count": 1,
  "size": {
    "value": 550.0,
    "unit": "sqft"
  },
  "built_in": {
    "year": 1980
  }
},
{
  "id": 15063505,
  "address": "740 Francisco St, San Francisco, CA",
  "map": {
    "latitude": 37804420,
    "longitude": -122417389
  },
  "sale": {
    "date": "11/30/15",
    "price": {
      "value": 2.15,
      "unit": "$M"
    }
  }
}
```

Activity: Knowledge Check



EXERCISE

ANSWER THE FOLLOWING QUESTIONS (5 minutes)

1. In the following examples, which might be the best storage or database solution? Why?
 - a) An application where a user can create a profile
 - b) An online store
 - c) Storing the last visit date of a user

DELIVERABLE

Answers to the above questions

Activity: Knowledge Check



EXERCISE

ANSWER THE FOLLOWING QUESTIONS (5 minutes)

1. Consider a dataset from Uber with the following fields:

- User ID
- User Name
- Driver ID
- Drive Name
- Ride ID
- Ride Time
- Pickup Latitude
- Pickup Longitude
- Pickup Location Entity
- Dropoff Longitude
- Dropoff Latitude

Activity: Knowledge Check (cont.)



EXERCISE

ANSWER THE FOLLOWING QUESTIONS (cont.)

- Dropoff Location Entity
 - Miles
 - Travel Time
 - Fare
 - CC Number
2. In your table, discuss how you would design a relational database to support this data? List the tables you would create, the fields they would contain, and how they would link to other tables

DELIVERABLE

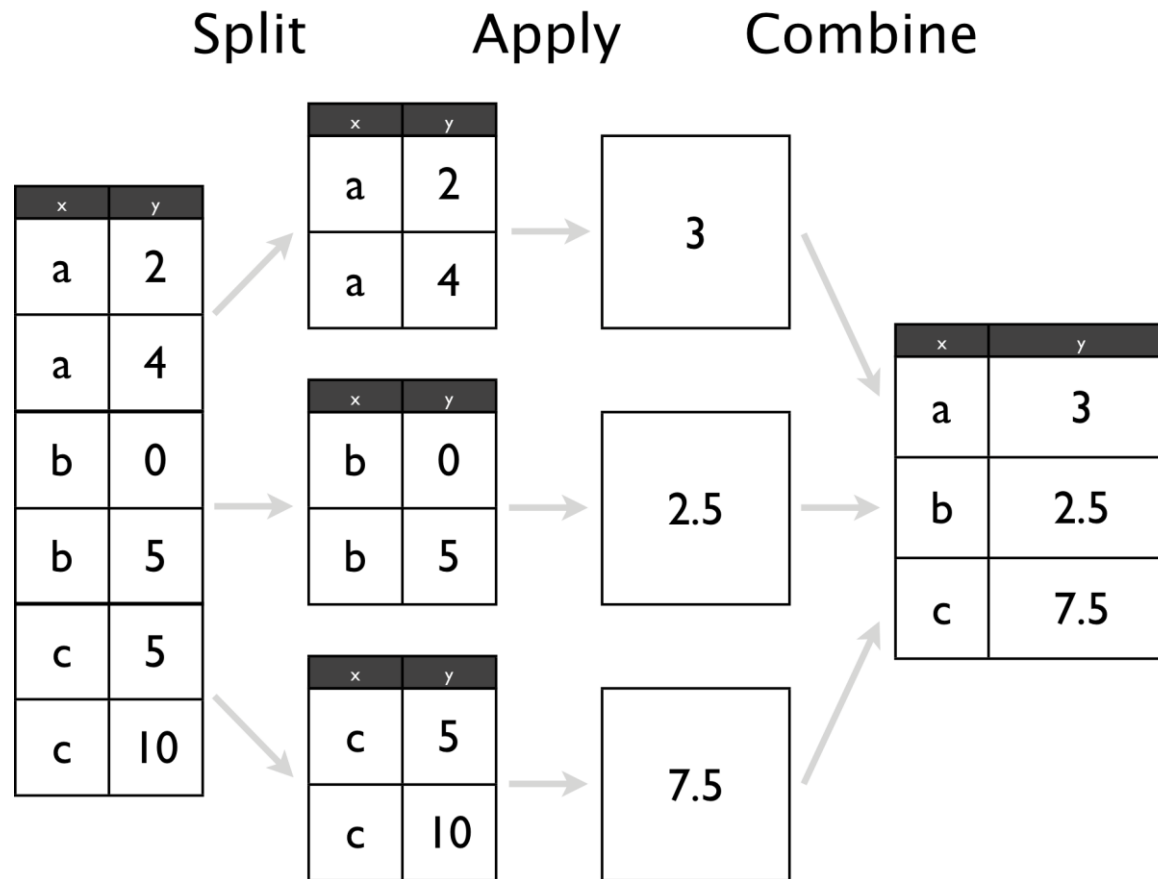
Your database schema design



DS

Codealong

Split-Apply-Combine



DS

Review

Review

- While this was a brief introduction, databases are often at the core of any data analysis. Most analysis starts with retrieving data from a database
- SQL (Structured Query Language) is a key language that any data scientist should understand
 - SELECT: Used in every query to define the resulting columns
 - WHERE: Filters rows based on a given condition
 - GROUP BY: Groups rows for aggregation
 - JOIN: Combines two tables based upon a given condition
- *pandas* can be used to access data from databases as well. The result of the queries will end up in a *pandas* dataframe
- There is much more to learn about query optimization if one dives further!



DS

Q & A



DS

Exit Ticket

Don't forget to fill out your exit ticket [here](#)