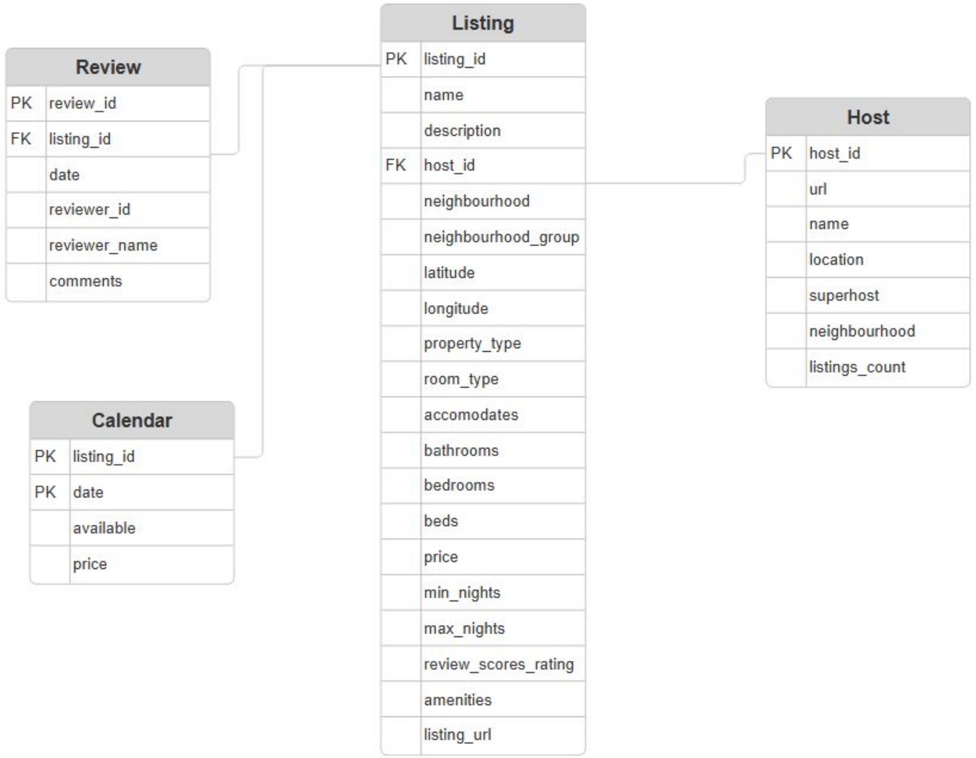
Stage 3: Prototype Implementation

By: Jeremy Lin (jl190), Justin Leong (jyleong2)

Overview: General Data Model - Spark

- We utilized Spark because of its computational speed for working with data
  - It can handle structured or unstructured data!
- Additionally, it is efficient for working with our data considering it’s “larger size”
  - Can also be used for relational when required
- Spark uses Resilient Distributed Dataset (RDD) – stores data across multiple nodes, process in parallel
- Consistency wasn’t a huge factor since we didn’t have real-time updates
  - Eventual consistency works for us in this case
- Data is also provided from Airbnb, meaning we don’t have to worry about replication or losing data from nodes going down
- Ease of use – API support for many languages (e.g. Java, Python, R)

Overview: Relational Table



Review	
PK	review_id
FK	listing_id
	date
	reviewer_id
	reviewer_name
	comments

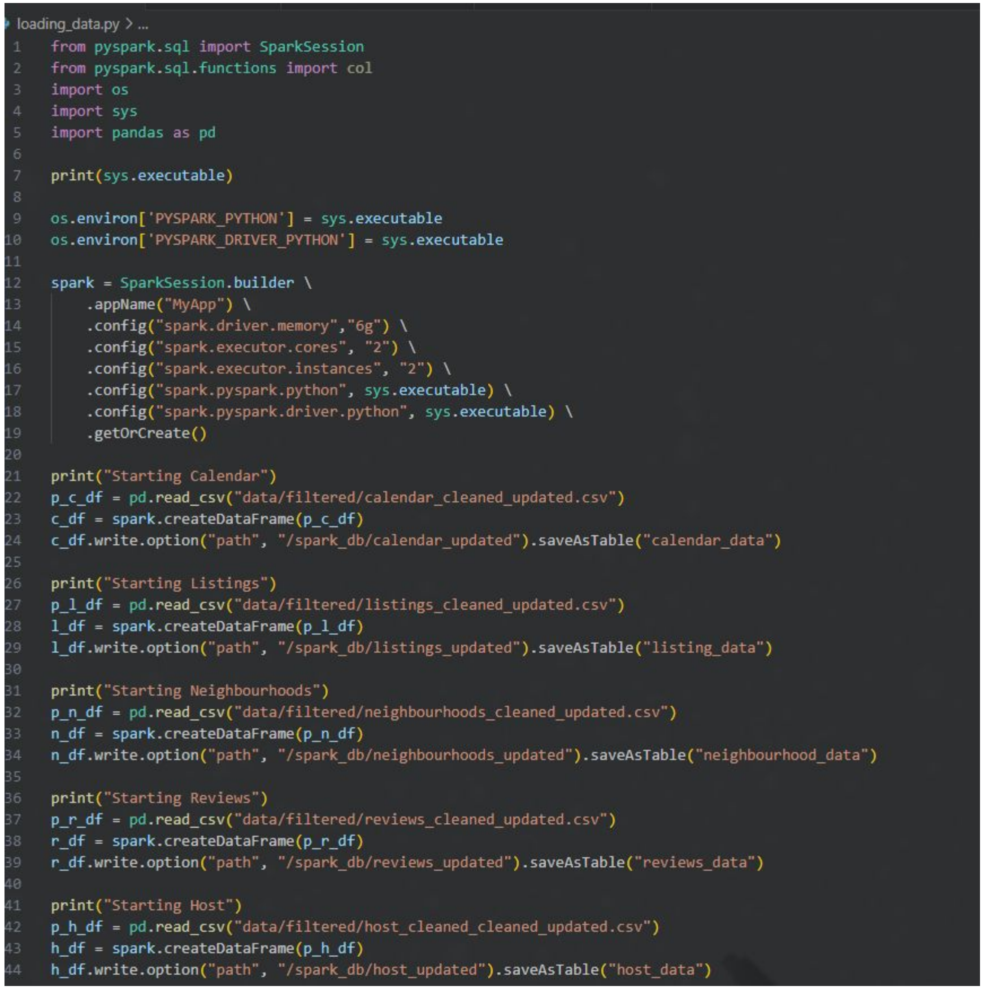
Calendar	
PK	listing_id
PK	date
	available
	price

Listing	
PK	listing_id
	name
	description
FK	host_id
	neighbourhood
	neighbourhood_group
	latitude
	longitude
	property_type
	room_type
	accomodates
	bathrooms
	bedrooms
	beds
	price
	min_nights
	max_nights
	review_scores_rating
	amenities
	listing_url

Host	
PK	host_id
	url
	name
	location
	superhost
	neighbourhood
	listings_count

Data Loading

- Data transformations through pandas
- Loading .csv into spark
- Original csv → pandas → altered csv → spark
- Why Local?
  - Resources
  - Compute Power


loading_data.py > ...

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import col
import os
import sys
import pandas as pd

print(sys.executable)

os.environ['PYSPARK_PYTHON'] = sys.executable
os.environ['PYSPARK_DRIVER_PYTHON'] = sys.executable

spark = SparkSession.builder \
    .appName("MyApp") \
    .config("spark.driver.memory","6g") \
    .config("spark.executor.cores", "2") \
    .config("spark.executor.instances", "2") \
    .config("spark.pyspark.python", sys.executable) \
    .config("spark.pyspark.driver.python", sys.executable) \
    .getOrCreate()

print("Starting Calendar")
p_c_df = pd.read_csv("data/filtered/calendar_cleaned_updated.csv")
c_df = spark.createDataFrame(p_c_df)
c_df.write.option("path", "/spark_db/calendar_updated").saveAsTable("calendar_data")

print("Starting Listings")
p_l_df = pd.read_csv("data/filtered/listings_cleaned_updated.csv")
l_df = spark.createDataFrame(p_l_df)
l_df.write.option("path", "/spark_db/listings_updated").saveAsTable("listing_data")

print("Starting Neighbourhoods")
p_n_df = pd.read_csv("data/filtered/neighbourhoods_cleaned_updated.csv")
n_df = spark.createDataFrame(p_n_df)
n_df.write.option("path", "/spark_db/neighbourhoods_updated").saveAsTable("neighbourhood_data")

print("Starting Reviews")
p_r_df = pd.read_csv("data/filtered/reviews_cleaned_updated.csv")
r_df = spark.createDataFrame(p_r_df)
r_df.write.option("path", "/spark_db/reviews_updated").saveAsTable("reviews_data")

print("Starting Host")
p_h_df = pd.read_csv("data/filtered/host_cleaned_cleaned_updated.csv")
h_df = spark.createDataFrame(p_h_df)
h_df.write.option("path", "/spark_db/host_updated").saveAsTable("host_data")
```

Query Demonstration (Pseudo code provided)

1. Assuming we are given the “particular two-day period” as an input of two different dates, we can filter the calendar table to obtain all the listings (listing_id) that are available on both of those days (removing listings that only fall on one of the days). Using all the listing_id that we have obtained, it can then be used in the listing table to obtain and display the name, neighborhood, room type, accommodates (# of guests), property type, amenities, price (per night’s cost), and review_scores_rating. The display method will sort to display by descending order of review_scores_rating.

2. Obtain all the neighborhoods of the particular dataset/city from the listing table “neighbourhood” column along with all the “listing_id” that are in that specific neighborhood. Using all the neighborhood names with their group of listing_id that are in that area, use the calendar table and filter by month. We can use this filter to then see which months have no listings for listing_id inside the neighborhoods. If all the listing_id inside a neighborhood don’t exist for a particular month, that means it would fall under the category “neighborhoods in any of the cities that have no listings for a given month”.

5. (With the assumption that each dataset is specifically for that city). For each dataset/city, under the review table, filter and sort only for the reviews that are given in the month of December in the “date” column. With all the filtered values that are only given in the month of December, then filter by the year and sum up all of the filtered values that are given in that specific year. The remaining product will give us the number of reviews received in December for each year.

Query Demonstration!

Critique of plans? What could we change?

- Our data model seemed to be pretty solid at first
- Only thing to change would most likely be the amount of attributes
  - To load data faster, we could reduce the amount of unused attributes, thus reducing size and increasing efficiency
  - To expand scopes of queries, add attributes when needed
- If we were to redo our design, maybe we would be able to add more functionality (adding other tables) to better searchability
- We mostly stuck to our original design!

Lessons Learned

- Primary lesson learned: Setting up and creating a functional database utilizing our ER diagram
  - In our case, on a local machine as well (not too different from cloud hosting)
- Spark - optimized for fast, in-memory distributed processing
  - Not a regular database, not really a storage solution
- Spark - also requires a lot of memory! (hungry for resources)
- How to structure/format data to alter the performance of queries

Advice to others?

- First and foremost, make sure to have a system with a lot of resources (i.e. ram, cores) for Spark if locally hosting
- Have a clear understanding of what you want to do
    - Spark is more of a processing tool rather than a storage tool!
- Make sure to set up system in a way for faster efficiency and querying, so processing will also be fast