AirBnB Prototype Presentation

Julio Cornejo

Data Model Overview

- MongoDB: Document-oriented, flexible schema (BSON)
    - Every document can vary fields; easy to evolve
- Referencing vs. Embedding -> chose referencing for scalability
    - Referencing: separate collections linked by IDs -> keeps documents small
    - Embedding: would group reviews/calendar inside listings -> risk of 16 MB limit
- Collections: listings, calendar, reviews, neighbourhoods

Example listings document


```
{
    "id": "75212",
    "name": "Sunny Room",
    "host_id”: "343761",
    "host_name": "Kris",
    "neighbourhood_group": "Other Cities",
    "neighbourhood": "Lakewood",
    "latitude": "33.8465",
    "longitude": "-118.08244",
    "room_type": "Private room",
    "price": "65",
    "minimum_nights": "2",
    "number_of_reviews": "77",
    "last_review": "1/12/25",
    "reviews_per_month": "0.46",
    "calculated_host_listings_count": "1",
    "availability_365": "330",
    "number_of_reviews_ltm": "7"
}
```

Query 1 Code (Q1)

```python
from pymongo import MongoClient

client = MongoClient("mongodb+srv://cellistkyle:yo@cluster0.jtzdio9.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0")
db = client["airbnb"]

# Dates to check
start_date = "2025-03-05"
end_date = "2025-03-06"

sample_vals = db["calendar"].distinct("available")
print("Distinct 'available' values:", sample_vals)

# Check what date formats exist
sample_dates = db["calendar"].distinct("date")
print("Sample 'date' values:", sample_dates[:10])  # Just print 10
# Step 1: Find listings available on both days
available_cursor = db["calendar"].aggregate([
    {
        "$match": {
            "date": {"$in": [start_date, end_date]},
            "available": True
        }
    },
    {
        "$group": {
            "_id": "$listing_id",
            "count": {"$sum": 1}
        }
    },
    {
        "$match": {"count": 2}  # Ensure both dates are available
    }
])
```

1) (AirBnB search) Display the listings available for a particular two-day period for Portland, OR with details: name, neighborhood, room type, how many guests it accommodates, property type and amenities, and per night's cost, in descending order of average rating

```python
# Collect matching listing_ids

listing_ids = [doc["_id"] for doc in available_cursor]
print(f"Found {len(listing_ids)} listings available on both dates.")
print(listing_ids[:10])  # Preview first few

results = db["listings"].find(
    {
        "id": {"$in": listing_ids},
        "loc": "Portland",

    },
    {
        "name": 1,
        "neighbourhood_cleansed": 1,
        "room_type": 1,
        "accommodates": 1,
        "property_type": 1,
        "amenities": 1,
        "price": 1,
        "review_scores_rating": 1
    }
).sort("review_
```

```python
# Step 3: Display
with open("top_listings.txt", "w", encoding="utf-8") as f:
    f.write("Top 10 Listings for 2025-03-05 and 2025-03-06\n\n")
    for i, listing in enumerate(results, 1):
        f.write(f"{i}. {listing.get('name', 'Unnamed')}\n")
        f.write(f"   Neighborhood: {listing.get('neighbourhood_cleansed', 'N/A')}\n")
        f.write(f"   Room Type: {listing.get('room_type', 'N/A')}\n")
        f.write(f"   Accommodates: {listing.get('accommodates', 'N/A')}\n")
        f.write(f"   Property Type: {listing.get('property_type', 'N/A')}\n")
        f.write(f"   Price: ${listing.get('price', 'N/A')}\n")
        f.write(f"   Rating: {listing.get('review_scores_rating', 'N/A')}\n")
        amenities = listing.get("amenities", [])
        if isinstance(amenities, list):
            amenities_str = ", ".join(str(item).strip() for item in amenities)
        else:
            amenities_str = str(amenities).strip()
        f.write(f"   Amenities: {amenities_str}\n\n")

print("Exported top 10 listings to top_listings.txt")
```

Query 1 Results (Q1)

```
Top 10 Listings for 2025-03-05 and 2025-03-06

1. N Portland Adventure Basecamp!
   Neighborhood: Kenton
   Room Type: Private room
   Accommodates: 2
   Property Type: Private room in home
   Price: $55.0
   Rating: 5.0
   Amenities: ["Hair dryer", "Hot water", "Exercise equipment", "Wine glasses", "First aid kit", "Dedicated workspa

2. Charming & Cozy Retreat in Portland
   Neighborhood: Hayhurst
   Room Type: Private room
   Accommodates: 2
   Property Type: Private room in home
   Price: $54.0
   Rating: 5.0
   Amenities: ["Hair dryer", "Conditioner", "Hot water", "Exercise equipment", "Exterior security cameras on proper

3. Exec. 3bm condo centrally located to tram 'n shops
   Neighborhood: Arbor Lodge
   Room Type: Entire home/apt
   Accommodates: 6
   Property Type: Entire condo
   Price: $115.0
   Rating: 5.0
   Amenities: ["Hot water", "Private entrance", "Wine glasses", "First aid kit", "Dedicated workspace", "Dishes and

4. Downtown PDX Condo Indoor Pool Mt Hood Views
   Neighborhood: Portland Downtown
   Room Type: Entire home/apt
   Accommodates: 2
```

Query 2 Code (Q4)

```
from pymongo import MongoClient
from pprint import pprint

client = MongoClient("mongodb+srv://cellistkyle:yo@cluster0.jtzdio9.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0")
db = client["airbnb"]

# Step 1: Get IDs of "Entire home/apt" listings
entire_home_ids = db["listings"].distinct("id", {"room_type": "Entire home/apt"})

# Step 2: Aggregate calendar availability for those listings
pipeline = [
    {
        "$match": {
            "listing_id": {"$in": entire_home_ids},
            "available": True,
            "date": {"$regex": "^2025-(03|04|05|06|07|08)"}  # March–August
        }
    },
    {
        "$group": {
            "_id": {"$substr": ["$date", 0, 7]},  # Group by "YYYY-MM"
            "available_nights": {"$sum": 1}
        }
    },
    {"$sort": {"_id": 1}}
]

result = list(db["calendar"].aggregate(pipeline))
```

Booking trend, by month) For “Entire home/apt” type listings in
Portland provide the total number of available nights (as per Query
3) for each month March through August of a given year.

Query 2 Results (Q4)

```
(venv) PS C:\Users\jcorn\OneDrive\Desktop\498dbb> python query2.py
Available Nights per Month (Entire home/apt):

2025-03: 50259 nights
2025-04: 69606 nights
2025-05: 77831 nights
2025-06: 68549 nights
2025-07: 72294 nights
2025-08: 75765 nights
```

Query 3 Code

```
from pymongo import MongoClient

client = MongoClient("mongodb+srv://cellistkyle:yo@cluster0.jtzdio9.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0")
db = client["airbnb"]

pipeline = [
    {"$match": {"date": {"$regex": "-12-"}}},
    {
        "$group": {
            "_id": {
                "city": "$rloc",
                "year": {"$substr": ["$date", 0, 4]}
            },
            "review_count": {"$sum": 1}
        }
    },
    {"$sort": {"_id.year": 1, "_id.city": 1}}
]
```

For each city, how many reviews are received for December of each year?

Query 3 Results

```
2022 - Portland: 3621 reviews
2022 - SD: 8164 reviews
2022 - Salem: 209 reviews
2023 - LA: 21250 reviews
2023 - Portland: 4195 reviews
2023 - SD: 11198 reviews
2023 - Salem: 250 reviews
2024 - LA: 29001 reviews
2024 - Portland: 5283 reviews
2024 - SD: 6718 reviews
2024 - Salem: 183 reviews
```

QUERY 4 Code

```
neighborhood_docs = db["neighbourhoods"].find({}, {"neighbourhood": 1})
all_neighborhoods = set(doc["neighbourhood"] for doc in neighborhood_docs)

target_month = "2025-07"  # July 2025

available_listing_ids = db["calendar"].distinct(
    "listing_id",
    {
        "available": True,
        "date": {"$regex": f"^{target_month}"}
    }
)

available_listings = db["listings"].find(
    {"id": {"$in": available_listing_ids}},
    {"neighbourhood_cleansed": 1}
)
active_neighborhoods = set(
    doc["neighbourhood_cleansed"] for doc in available_listings if "neighbourhood_cleansed" in doc
)
neighborhoods_with_no_listings = all_neighborhoods - active_neighborhoods

# Output
print(f"Portland Neighborhoods with NO available listings in {target_month}:\n")
for n in sorted(neighborhoods_with_no_listings):
    print(f"- {n}")
```

What neighborhoods in any of the cities that have no listings for a
given month?

Query 4 Result

```
PS C:\Users\jcorn\OneDrive\Desktop\498dbb> python query4.py
Portland Neighborhoods with NO available listings in 2025-07:

- Maywood Park
- Northwest Industrial
- Woodland Park
```

Critique of Data Model / Design Changes

- Embedding vs. Referencing
  - Initial idea: embed **calendar** & **reviews** in **listings** for single-query fetch
  - Issue: embedding calendar and review data could cause individual listings documents to grow very large, which might exceed 16 MB BSON limit
  - Change: switched to referencing
- Indexing
  - Initial: simple single-field indices
  - Change: added compound indices (**neighbourhoods_group** + **price**, **room_type** + **host_id**) for common filters

(continued)

- Transaction Usage
  - Initial: considered multi-document transactions for referential updates
  - Change: abandoned transactions for per-document operations to maximize throughput
- Shard Key Choice
  - Initial: considered sharding on **host_id** (uneven distribution)
  - Change: decided on **listing_id** hashed shard key for uniform distribution

Overview and Insight Gained

- MongoDB Flexibility is a Double-Edged Sword
  - MongoDB's schemaless nature made it easy to ingest raw CSVs and prototype quickly.
  - However, without a strict schema, we had to do a lot of pre-processing to ensure consistency
- Importance of Indexing
  - Indexes significantly improved performance for heavy operations like filtering availability by date or joining reviews with listings.
  - Advice: Design your indexes based on expected query workload
- Data Cleaning is Crucial
  - Similar to what we did in Assignment 3, it was really crucial to clean and uniformize the data.
  - Had to normalize different formats across the CSVs to maintain consistency (e.g., listing_id types across calendar and reviews).
- Embedding vs Referencing: Tradeoffs Realized

Advice for New Users

- Prototype schema with sample data early
- Use MongoDB University’s free courses for best practices
- Design your indexes based on expected query workload
  Don't wait until your queries start lagging
- For high-performance needs, consider denormalization or embedding when appropriate.

Final Slide

We ended up loading over 512MB of data as we had to upgrade our ATLAS plan to successfully keep querying what we needed to and add data. One of the largest imports was the calendar with over a million records.