Application: ParkSafari

Group 8: Yuying Fan, Bailey Lin, Botong Zhang, Jingkai Zheng

# 1 Introduction

## Our Project Goal

We want to build an application that combines the information from all national parks together with Airbnb information for travel planning. Our website will offer various filters and search options, allowing users to search for National Parks based on their location, included species, related trail difficulties, and nearby Airbnb price listings.

## ParkSafari Functionality

ParkSafari allows users who love nature and hiking to browse national parks and trails in the United States by the species they hope to see in the park. It can display the location and description of a list of popular national parks and allow the users to explore the different animals and plant species found in each park, including information about their abundance, scientific names, and levels of conservation. ParkSafari also allows users to search for trails within each park with filters for difficulty, length, popularity, etc. In addition, ParkSafari can display all the Airbnb listings in the vicinity of each National Park, providing users with convenient and appropriate accommodation options for their travel.

## The Motivation

All of our group members are national park lovers. However, information about national parks in the US are scattered across each of their websites, so planning a road trip to various parks often involves a tedious process of collecting information from several websites. Currently there is no public website that allows national park visitors to select parks and trails based on the living species they can observe onsite, while providing related accommodation recommendations at the same time.

# 2 Architecture

## List of Technologies

1. MySQL Database
2. Node.js with React
3. Bootstrap Framework
4. DynamoDB\*
5. EC2/Heroku deployment\*

## Description of System Architecture

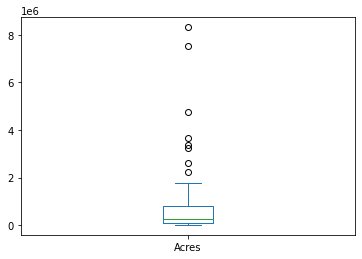
Our application has backend and frontend implementations. The backend under the “server” directory establishes the routes and SQL queries associated with each of the routes. The frontend will use the routes established in the backend to retrieve SQL query results. The frontend has three main pages: home page, discovery page, and parks page. On the home page, it allows users to search a park by park name, state, and species name. Additionally, users can also sort the search results by park name, park area, and species count. On discovery pages, some fun facts about national parks are presented. For example, users can look for the top 10 most popular species in all popular parks. On the park page, it shows all the parks with their corresponding information, such as location, area, species count, and so on.

# 3 Data

## Description of Data

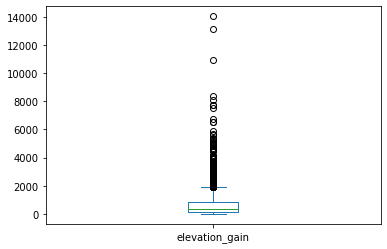
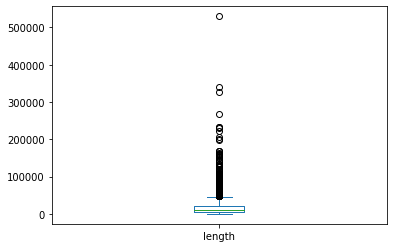
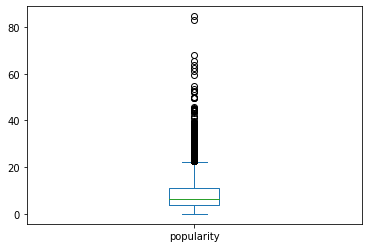
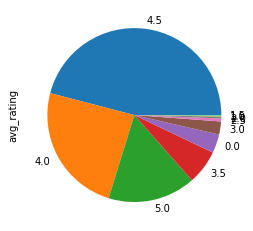
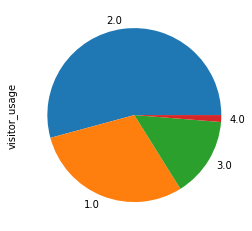
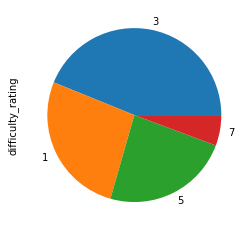
### Dataset 1: Biodiversity in National Parks (Parks and Species)

* Plant and animal species found in the American national park system.
* Columns (everything is string unless otherwise indicated):
  + **parks.csv:** Park Code, Park Name, State, Acres (int), Latitude (float), Longitude (float)
    - Non-null columns: All columns
    - Size: 3.29 KB, 56 rows and 6 columns
  + **species.csv:** Species ID, Park Name, Category, Order, Family, Scientific Name, Common Names, Record Status, Occurrence, Nativeness, Abundance, Seasonality, Conservation Status
    - Non-null columns: Species ID, Park Name, Category, Scientific Name, Common Names, Record Status
    - Size: 17.51 MB, 119248 rows and 13 columns
* Source: <https://www.kaggle.com/datasets/nationalparkservice/park-biodiversity>
* Relevant statistics in parks.csv: **Acres:** min: 5,550; max: 8,323,148; mean: 927,929.1; std: 1,709,258
* Relevant statistics in species.csv: **Category:** 14 distinct categories (Mammal, Bird, Reptile, Amphibian,Fish,Vascular Plant, Spider/Scorpion, Insect, Invertebrate, Fungi, Nonvascular Plant, Crab/Lobster/Shrimp, Slug/Snail,Algae)



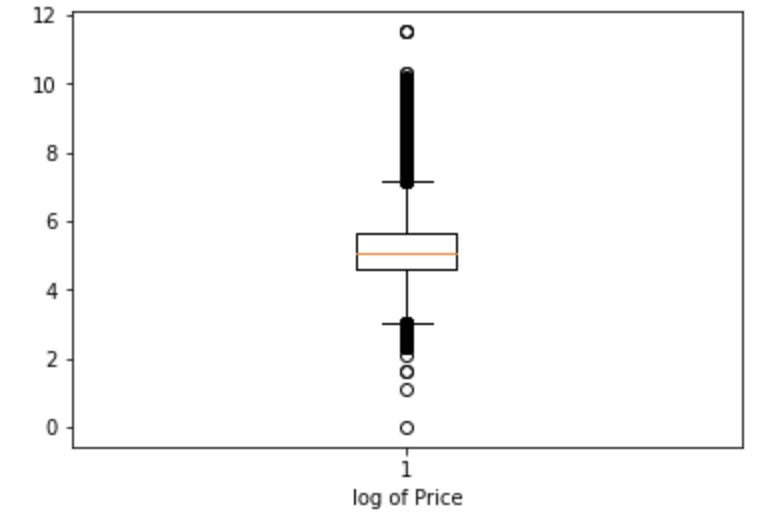
### Dataset 2: National Park Trails

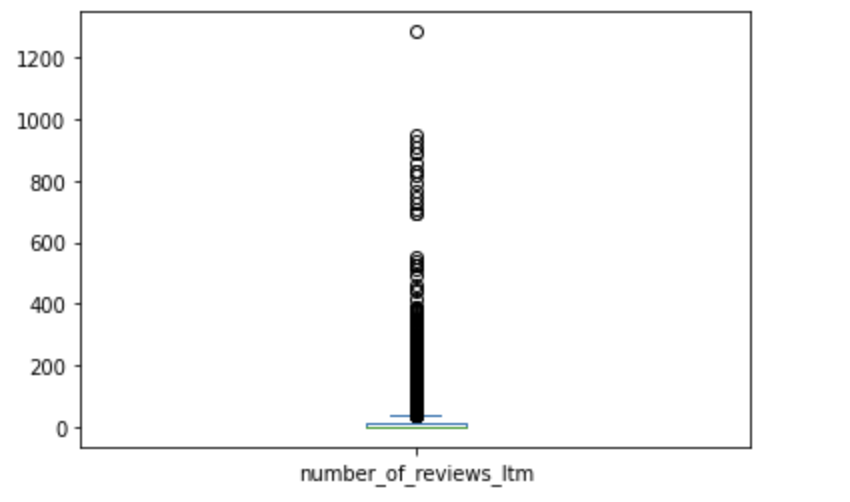
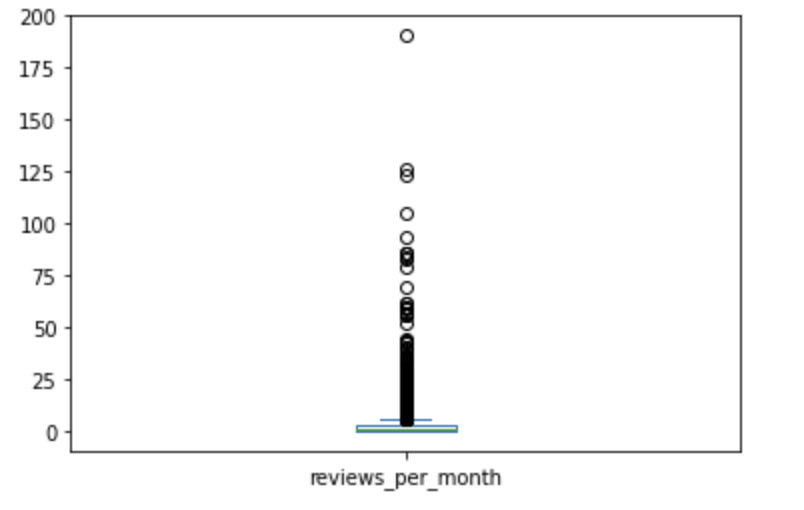
* Every trail in the National Parks Service gathered from alltrails.com.
* Columns (everything is string unless otherwise indicated): trail\_id (int), name, area\_name, city\_name, state\_name, country\_name, \_geoloc (tuple of lat and lng, both are floats), popularity (float), length (float), elevation\_gain (float), difficulty\_rating (int), route\_type, visitor\_usage (int), avg\_rating (float), num\_reviews (int), features (string list), activities (list strings), units
  + Non-null columns: All columns except column “visitor\_usage”
* Size: 992.6 KB, 3313 rows and 18 columns
* Source: <https://www.kaggle.com/datasets/planejane/national-park-trails>
* Relevant statistics:
  + **Popularity:** min:0; max: 84.622900; mean: 8.953441; std: 8.138323
  + **Length:** min: 0; max: 529794.728; mean: 17676.848717; std: 25497.376640
  + **Elevation\_gain:** min: 0; max: 14029.944; mean: 641.805943; std: 901.506642
  + **Difficulty\_Rating:** min: 1; max: 7; mean: 3.167824; std: 1.702752
  + **Visitor\_Usage:** min: 1; max: 4; mean: 1.877124; std: 0.947039
  + **Avg\_rating:** min: 0; max: 5; mean: 4.173106; std: 1709258

### Dataset 3: Airbnb USA Listings

* Airbnb listings from over 15 states and 30 unique cities/counties.
* Columns (everything is string unless otherwise indicated): id (int), name, host\_id (int), host\_name, neighbourhood\_group, neighbourhood, latitude (float), longitude (float), room\_type, price (int), minimum\_nights (int), number\_of\_reviews (int), last\_review (date-formatted string), reviews\_per\_month (float), calculated\_host\_listings\_count (int), availability\_365 (int), number\_of\_reviews\_ltm (int), license, state, city
  + Non-null columns: id, host\_id, latitude, longitude, room\_type, price, minimum\_nights, number\_of\_reviews, calculated\_host\_listings\_count, availability\_365, number\_of\_reviews\_ltm, state, city
* Size: 60.36 MB, 325858 rows and 21 columns
* Source: <https://www.kaggle.com/datasets/tamle507/airbnb-listings-usa>
* Relevant statistics:
  + **Price:** min:0.00; max: 100000; mean: 284.92; std: 835.57
  + **Number\_of\_reviews:** min:0.00; max: 2600; mean: 39.46; std: 75.72
  + **Reviews\_per\_month:** min:0.01; max: 190.48; mean: 1.69; std: 2.01
  + **Number\_of\_reviews\_ltm:** min:0.00; max: 1284.00; mean: 11.89; std: 20.67





## Explanation of Data Usage

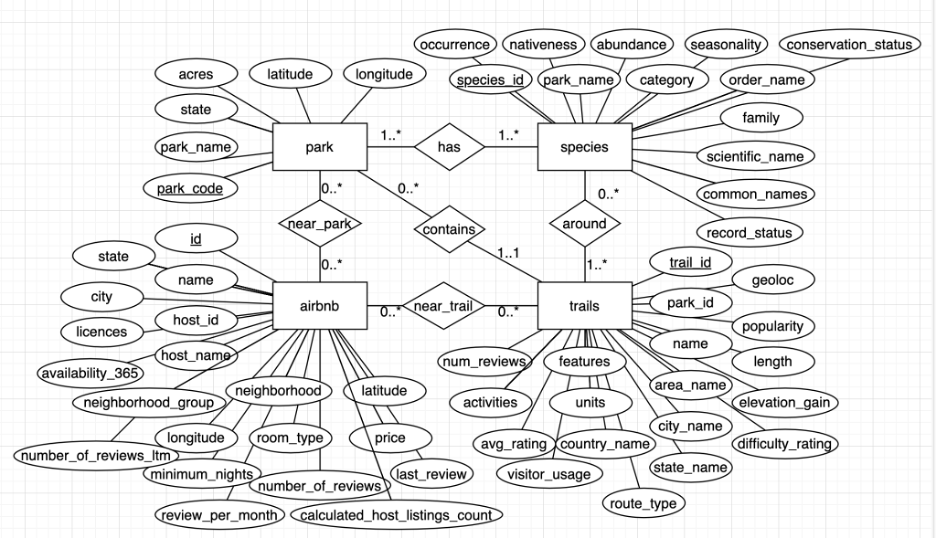
* With the park.csv, we created a Park table to store all the park relevant information. With SQL queries, we can display the list of all national parks, sort parks by area, search for a park by name or by state.
* With the species.csv, we created a Species table to store all the species relevant information. With SQL queries, we can display all species at a specific national park, display the species count at a park, sort parks by species count, and search for parks having a specific species.
* With the national park trail data, we created a Trail table to store all the trails relevant information. With SQL queries, we can display all trails at a specific national park, filter trails by length, difficulty, and average ratings, and get all trails where a specific species can be observed by joining the table with the Species table.
* With the Airbnb data, we created a Airbnb table to store all the relevant information. With SQL queries, we can sort Airbnb by distance, price, and number of reviews, and get all Airbnb listings near a specific national park or trail within [x] distance away.
* By joining different tables together, we can achieve some special functionalities described in the previous section, like providing suggestions for top Airbnbs that are surrounded by the highest abundance of species.

# 4 Database

## Data Preprocessing and entity resolution efforts

* Format Column Names:
  + Change column names to lowercase and substitute space with "\_";
  + Change column names that start with "\_" in Trails;
  + Change column name of "order" to "order\_name" in Species dataset;
* Drop unnamed and unrelated columns;
* Make parks name stay consistent across all datasets:
  + Since the number of national parks in the Trail dataset is initially larger than the one in the Park dataset, we kept only the national parks that appear in the Park dataset.
* Parse Geoloc into longitude and latitude.

## ER diagram



## Number of instances in each table

* Park: 56 rows, 6 columns;
* Species: 119,175 rows, 13 columns;
* Trails: 3,137, 18 columns;
* Airbnb: 120, 212, 20 columns.

## Normal Form and Justification

### Park Table

Consider the relation of schema Park is R(park code, park name, state, acres, latitude, longitude), where park code is the primary key. Considering there are no dependencies between non-key attributes, the minimum cover of functional dependencies are the following: F = {park\_code → park\_name, park\_code → state, park\_code→acres, park\_code → latitude, park\_code → longitude}. The superkey is park\_code, so it is in both BCNF and 3NF.

Species Table

Consider the relation of schema Species is R(species\_id, park\_name, category, order\_name, family, scientific\_name , common\_names, record\_status, occurrence, nativeness, abundance, seasonality, conservation\_status), where species\_id is the primary key. The minimum cover of functional dependencies are the following: F = {species\_id → park\_name, species\_id → scientific\_name, scientific\_name → common\_names, scientific\_name → family, family → order\_name, order\_name → category, species\_id → record\_status, species\_id → occurrence, species\_id → nativeness, species\_id → abundance, species\_id → seasonality, species\_id → conservation\_status}.

Since there are dependencies between non-key attributes like {scientific\_name → common\_names, scientific\_name → family, family → order\_name, order\_name → category}, we can decompose these transitive dependencies into smaller schema in order to achieve 3NF such as the following: R(scientific\_name, common\_names), R(scientific\_name, family), R(family, order\_name), R(order\_name, category).

However, we aim to fetch comprehensive data on species, like scientific\_name, common\_names, family, order, and category. Splitting these attributes into separate schemas to meet 3NF would increase query runtime due to the required joins. Thus, we're maintaining our original schema to boost query performance, even though it doesn't comply with 3NF.

### Trail Table

Consider the relation of schema Trail is R(trail\_id, park\_code, name, park\_name, city\_name, state\_name, country\_name, popularity, length, elevation\_gain, difficulty\_rating, route\_type, avg\_rating, num\_reviews, features, activities, latitude, longitude), where trail\_id is the primary key. The minimum cover of functional dependencies are the following: F = {trail\_id → park\_code, trail\_id → name, park\_code → park\_name, park\_name → city\_name, city\_name → state\_name, state\_name → country\_name, trail\_id → popularity, trail\_id → length, trail\_id → elevation\_gain, trail\_id → difficulty\_rating, trail\_id → route\_type, trail\_id → avg\_rating, trail\_id → num\_reviews, trail\_id → features, trail\_id → activities, trail\_id → latitude, trail\_id → longitude}.

Since there are dependencies between non-key attributes like {park\_code → park\_name, park\_name → city\_name, city\_name → state\_name, state\_name → country\_name}, we can decompose these transitive dependencies into smaller schema in order to achieve 3NF such as the following: R(park\_code, park\_name), R(park\_name, city\_name), R(city\_name, state\_name), R(state\_name, country\_name).

However, our trail-related queries require detailed information such as park, city, state, and country\_names. Applying 3NF and dividing these attributes into separate schemas would require extra joins, increasing query time. Thus, to enhance query performance, we're maintaining our original schema, even though it doesn't comply with 3NF.

Species Table

Consider the relation of schema Airbnb is R(id, name, host\_id, host\_name, neighbourhood\_group, neighbourhood, latitude, longitude, room\_type, price, minimum\_nights, number\_of\_reviews, last\_review, reviews\_per\_month, calculated\_host\_listings\_count, availability\_365, number\_of\_reviews\_ltm, license, state, city), where id is the primary key. The minimum cover of functional dependencies are the following: F = {id → name, id → host\_id, host\_id → host\_name, latitude, longitude → neighbourhood\_group, latitude, longitude → neighbourhood, id → latitude, id → longitude, id → room\_type, id → minimum\_nights, id → number\_of\_reviews, id → last\_review, id → reviews\_per\_month, id → calculated\_host\_listings\_count, id → availability\_365, id → number\_of\_reviews\_ltm, id → license, id → state, latitude, longitude → city}.

Since there are dependencies between non-key attributes like {latitude, longitude → neighbourhood\_group; latitude, longitude → neighbourhood; latitude, longitude → city}, we can decompose these transitive dependencies into smaller schema in order to achieve 3NF such as the following: R(latitude, longitude, neighbourhood\_group), R(latitude, longitude, neighbourhood), R(latitude, longitude, city).

However, for Airbnb-related queries, we need to fetch details such as neighbourhood, neighbourhood\_group, and city. Segregating these non-key attributes to meet 3NF would cause an upswing in query time due to the needed joins. So, we're sticking with our original schema for better query performance, even though it doesn't conform to 3NF.

# 5 Web App description

ParkSafari has three pages: the Home Page, the Discovery Page, and the Park Page.

On the Home Page, ParkSafari allows users to search a park by park name, state, and species name. Additionally, users can also sort the search results by park name, park area, and species count.

In the Discovery Page, we have incorporated some special features that satisfy the needs of certain travelers’ special interests. First, for photographers, ParkSafari helps to find the most frequently appeared species in parks that are rarely visited and with top-rated Airbnb nearby. Users could choose the number of species displayed. Secondly, ParkSafari provides a list of the most popular species in each park with popular trails for park visitors. For travelers who hope to find a certain species, ParkSafari can provide the top airbnb listings that are closest to the parks where one can find the species. For animal lovers, ParkSafari provides recommendations for the top Airbnbs that are surrounded by the highest abundance of species.

On the Park Page, all national parks in the US are displayed in the form of a card and each card shows all the park’s detailed information, such as location, area, species count, and so on.

# 6 API Specification

* **allParks:** get all national parks sorted by a given criterion; @param sortBy: the criterion to sort by, must be name (alphabetical), area (descending), or species\_count (descending); @return{string}: the SQL query string for this search.
* **allSpeciesAtPark:** get all species at the given park; @param parkName: the name of the park to search for; @return {string}: the SQL query string for this search.
* **airbnbInfo:** get the information about a specific Airbnb listing; @param id The id of the Airbnb listing to search for; @return {string} The SQL query string for this search.
* **airbnbsNearPark:** get the 50 closest Airbnb listings to the park specified by the given park code sorted by distance, price, and number of reviews; @param parkCode: The code of the park to search for; @return {string}: The SQL query string for this search.
* **searches:** search for a park by name, state, or species observed sorted by the given criterion; @param searchBy: the criterion to search by, must be park\_name, state, or species; @param sortBy: the criterion to sort by, must be name (alphabetical), area (descending), or species\_count (descending); @param searchTerm: the term to search for; @return {string}: the SQL query string for this search.
* **trailsForPark:** get the top 20 best rated trails at the given park. @param parkCode: the code of the park to search for; @return {string}: The SQL query string for this search.
* **recommendedAirbnbForSpecies:** for each of the national parks where a specific species can be found, get the top 3 best-valued Airbnb listings that are the closest to this park. Best-valued listing is defined as the Airbnb that is the closest and has at least 150 user reviews. @param species: a common name for the species of interest; @param num: The number of Airbnbs to return for each park; @return {string}: The SQL query string for this search.
* **recommendedAirbnbForSpeciesOptimized:** optimized version of recommendedAirbnbForSpecies; it has the same request and return parameters.
* **recommendedAirbnbInStateForSpecies:** gor each of the national parks in a specific state where a specific species can be found, get the top 3 best-valued Airbnb listings that are the closest to this park. Best-valued listing is defined as the Airbnb that is the closest and has at least 150 user reviews; @param species: a common name for the species of interest; @param num: the number of Airbnbs to return for each park; @param state: the 2-letter code of the state of interest; @return {string}: the SQL query for this search.
* **recommendedAirbnbInStateForSpeciesOptimized:** optimized version of recommendedAirbnbInStateForSpecies; it has the same request and return parameters.
* **mostBiodiverseAirbnbs:** in a specified state and neighbourhood, get the top n Airbnbs that have the highest species count from parks within [x] miles of radius from it; @param state: the 2-letter code of the state of interest; @param neighbourhood: the neighbourhood of interest; @param distance: the maximum distance from the Airbnb to the park in miles; @param num: the number of Airbnbs to return; @return {string}: the SQL query for this search.
* **mostBiodiverseAirbnbsOptimized:** optimized version of recommendedAirbnbInStateForSpecies; it has the same request and return parameters.
* **popularSpecies:** get the top n most popular species in each park that have a trail with popularity >= 6.5731; @param num: the number of species to return for each park; @return {string}: the SQL query for this search.
* **popularSpeciesOptimized:** optimized version of popularSpecies; it has the same request and return parameters.
* **speciesForPhotographers:** get top n most frequently appeared species in the nearby parks of the 100 top-rated Airbnbs that have trails with popularity less than or equal to 6 (e.g. Photography routes recommendation with more species and fewer people); @param num: the number of species to return; @return {string}: the SQL query for this search.
* **speciesForPhotographersOptimized:** optimized version of speciesForPhotographers; it has the same request and return parameters.

# 7 Queries

### **Complex Query 1: Recommended Airbnb for Species**

### For each of the national parks where a specific species can be found, the query gets the top n best-valued Airbnb listings that are the closest to this park. Best-valued listing is defined as the Airbnb that is the closest and has at least 150 user reviews. For example, for travelers who hope to find a certain species like “seal”, ParkSafari can provide the top airbnb listings that are closest to the parks where one can find the species.

### **Complex Query 2: Most Biodiverse Airbnb**

### In a specified state and neighbourhood, the query gets the top n Airbnbs that have the highest species count from parks within [x] miles of radius from it. For example, once users enter the state, neighbourhood information, and distance criteria, ParkSafari provides recommendations for the top Airbnbs that are surrounded by the highest abundance of species.

### **Complex Query 3: Popular Species:**

### The query gets the top n most popular species in each park that have a trail with popularity >= 6.5731. ParkSafari provides a list of the most popular species in each park for park visitors.

### **Complex Query 4: Species for Photographer:**

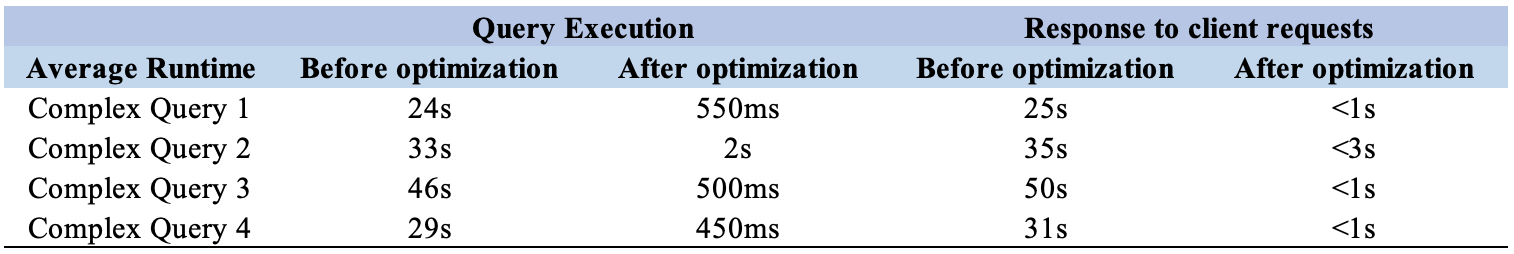
### This query gets top n most frequently appeared species in the nearby parks of the 100 top-rated Airbnbs that have trails with popularity less than or equal to 6. Nearby parks are defined as parks within 100 miles of an Airbnb's location. This feature is recommended for Photographers who prefer more species and fewer people in the park. ParkSafari helps to find the most frequently appeared species in parks that are rarely visited and with top-rated Airbnb nearby.

### **Query 5: Find Nearby Valued Airbnbs Around Parks:**

### This query gets the 50 closest Airbnb listings to the specified park sorted first by distance then by price and number of reviews. ParkSafari can recommend best-value Airbnbs that are closest to specific parks.

* **All queries are stored in the Github repository → server folder → queries.js**

# 8 Performance Evaluation



### Optimization explanation for complex query 1:

1. Restructuring the query by pushing projections and selections to as far down to the base relations as possible;
2. Caching: create a materialized view to store the ranked Airbnbs that are closest to any national park and have at least 150 reviews. In this way, we don’t have to calculate the distances between each Airbnb and national parks every time we execute the query.
3. Indexing: in the materialized view, we add
   1. A Hash-based index on attributes park\_name because it is used in equality search.
   2. A B+Tree indexes on attribute ranking because it is used in range search.
   3. A multi-column B+Tree index on distance\_to\_park, price, and number\_of\_reviews attributes because they are used in the ordering.

### Optimization explanation for complex query 2:

1. Restructuring the query by pushing projections and selections to as far down to the base relations as possible;
2. Caching: use the same materialized view created for complex query 1.
3. Indexing: in the materialized view, we add
   1. Hash-based indexes on attributes id neighbourhood because they are used in equality search.
   2. B+Tree index on attribute distance\_to\_park because it is used in range search.

### Optimization explanation for complex query 3:

1. Caching: create a materialized view to store all species in each park, where the species are ranked in descending order of popularity. In this way, we don’t have to calculate the number of occurrences of each species in all parks every time we execute the query.
2. Indexing: In the materialized view, we add
   1. A B+Tree index on attribute ranking because it is used in range search.
   2. A B+Tree index on park\_name and ranking attributes because they are used in the ordering.

### Optimization explanation for complex query 4:

1. Caching: create a materialized view to store the number of occurrences of all species in the nearby parks of 100 top-rated Airbnbs that have trails with popularity less than or equal to 6.
2. Indexing: In the materialized view, we add a multi-column B+Tree index on occurrence\_count and species\_id attributes because they are used in the ordering.

# 9 Technical Challenges

### Challenge 1: Difficulty of coming up with complex query

Since two of our tables are relatively small and most of the regular functionality only requires relatively simple queries, it took us a while to come up with complex queries that have real-life applications. After consulting with Professor Davidson, we decided to create joins between tables that can have many-to-many relationships, such as queries that involve calculating the distance between parks and Airbnbs.

### Challenge 2: Calculation of the distance between two locations

For three of our complex queries, we need to calculate the distance between an Airbnb and a park or trail using the longitude and latitude information. It is difficult as Earth is a globe and irregular sphere. After doing some research, we found that we can use the Haversine formula to solve the problem.