**Complex Query 1A**

Original Query: 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. For example, species =seal, num = 3.

Text

Description automatically generated

Before optimization, the actual runtime is around 20s. Here is the original explain plan:

Text

Description automatically generated

**Query optimization:**

1. We create a materialized view to store the ranked Airbnbs based on the distance of each to all national parks, where each Airbnb has at least 150 reviews. The Airbnbs are ranked first by the distance to the park in ascending order, then by park name in alphabetical order, and lastly by ranking in ascending order. We decided to use materialized view so that we don’t have to calculate the distances between each Airbnb and national parks every time we execute the query. Since the information in this view does not require frequent updates, we can use materialized view in this case.
2. In the materialized view, we add a Hash-based index on attributes park\_name because it is used in equality search.
3. In the materialized view, we add a B+Tree indexes on attribute ranking because it is used in range search.
4. In the materialized view, we add a multi-column B+Tree index on distance\_to\_park, price, and number\_of\_reviews attributes because they are used in the ordering.
5. In the Species table, we add a Hash-based index on attribute common\_names because it is used in equality search.

The updated query is shown as the following:

Text

Description automatically generated

After optimization, the new runtime is around 550ms. Here is the explain plan after optimization:

Text

Description automatically generated

**Complex Query 1B**

Original Query: For each of the national parks in a specific state where a specific species can be found, get the top x 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, state = CA, species =seal, num = 3.

Text

Description automatically generated

Before optimization, the actual runtime is around 8.5s. Here is the original explain plan:

Graphical user interface, text

Description automatically generated

**Query optimization:**

1. We use the same materialized view we created for complex query 1A.
2. On top of all the index we added for query 1A, we add a Hash-based index on attributes state in the materialized view because it is used in equality search.

The updated query is shown as the following:

Text

Description automatically generated

After optimization, the new runtime is around 500ms. Here is the explain plan after optimization:

Text

Description automatically generated

**Complex Query 2**

Original Query: 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. For example, state = CA, neighbourhood = Hollywood, distance < 100 miles, num = 10.

Text

Description automatically generated

Before optimization, the actual runtime is around 34s. Here is the original explain plan:

Text

Description automatically generated with medium confidence

**Query optimization:**

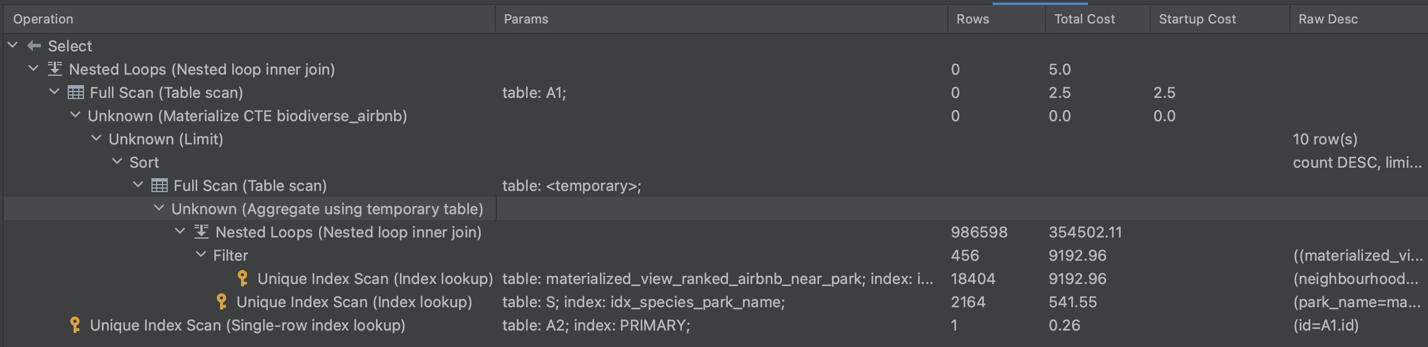
1. We use the same materialized view we created for complex query 1A.
2. On top of all the index we added for query 1A, we add
   1. A Hash-based index on attribute id in the materialized view because it is used in equality search.
   2. A Hash-based index on attribute neighbourhood in the materialized view because it is used in equality search.
   3. A B+Tree index on attribute distance\_to\_park in the materialized view because it is used in range search.

The updated query is shown as the following:

Text

Description automatically generated

After optimization, the new runtime is around 2s. Here is the explain plan after optimization:



**Complex Query 3:**

Original Query: Get the top n most popular species in each park that have a trail with popularity >= 6.5731. Popularity of a species is determined as the number of parks a species occur in. For example, n=10.

Text

Description automatically generated

Before optimization, the actual runtime is around 46s. Here is the original explain plan:

Graphical user interface, text

Description automatically generated

**Query optimization:**

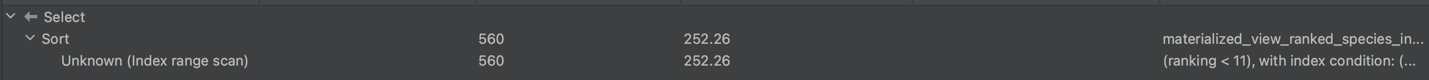
1. We create a materialized view to store all species in each park that have a trail with popularity >= 6.5731, where the species are ranked in descending order of popularity. Popularity of the species is determined as the number of parks a species occur in. We decided to use materialized view so that we don’t have to calculate the number of occurrences of each species in all park that have a trail with popularity >= 6.5731 every time we execute the query. Since the information in this view does not require frequent update, we can use materialized view in this case.
2. In the materialized view, we add a B+Tree index on attribute ranking because it is used in range search.
3. In the materialized view, we add a multi-column B+Tree index on park\_name and ranking attributes because they are used in the ordering.

The updated query is shown as the following:

Text

Description automatically generated

After optimization, the new runtime is around 500ms. Here is the explain plan after optimization:



**Complex Query 4:**

Original Query: 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). Nearby parks are defined as parks within 100 miles of an Airbnb's location. For example, n=10.

Text

Description automatically generated

Before optimization, the actual runtime is around 29s. Here is the original explain plan:

Text

Description automatically generated

**Query optimization:**

1. We create a materialized view to store all the counts of frequently appeared species in the nearby parks of the 100 top-rated Airbnbs that have trails with popularity less than or equal to 6. We decided to use materialized view so that we don’t have to calculate the occurrence count of all species near the 100 top-rated Airbnbs every time we execute the query. Since the information in this view does not require frequent updates, we can use materialized view in this case.
2. 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.

The updated query is shown as the following:

Text

Description automatically generated with medium confidence

After optimization, the new runtime is around 450ms. Here is the explain plan after optimization:

