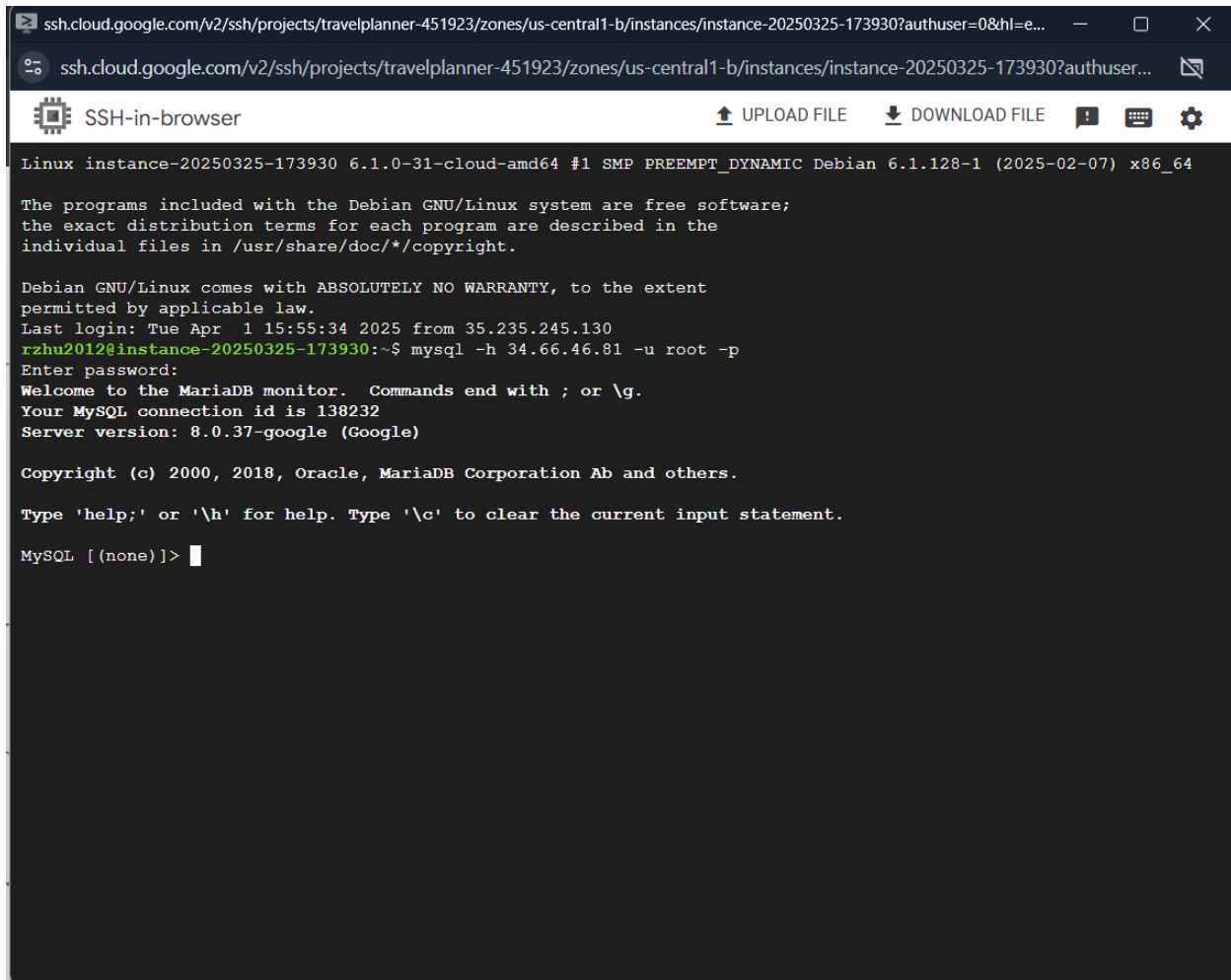


SSH Information

mysql -h 34.66.46.81 -u root -p



```
ssh.cloud.google.com/v2/ssh/projects/travelplanner-451923/zones/us-central1-b/instances/instance-20250325-173930?authuser=0&hl=e...  
ssh.cloud.google.com/v2/ssh/projects/travelplanner-451923/zones/us-central1-b/instances/instance-20250325-173930?authuser...  
SSH-in-browser  
Linux instance-20250325-173930 6.1.0-31-cloud-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.128-1 (2025-02-07) x86_64  
  
The programs included with the Debian GNU/Linux system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/copyright.  
  
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent  
permitted by applicable law.  
Last login: Tue Apr 1 15:55:34 2025 from 35.235.245.130  
rzhu2012@instance-20250325-173930:~$ mysql -h 34.66.46.81 -u root -p  
Enter password:  
Welcome to the MariaDB monitor.  Commands end with ; or \g.  
Your MySQL connection id is 138232  
Server version: 8.0.37-google (Google)  
  
Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.  
  
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.  
  
MySQL [(none)]> 
```

Attraction Table Length

```
MySQL [(none)]> use main
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MySQL [main]> SELECT COUNT(*) FROM Attraction
-> ;
+-----+
| COUNT(*) |
+-----+
|      3123 |
+-----+
1 row in set (0.005 sec)

MySQL [main]> 
```

User Table Length

```
MySQL [main]> SELECT COUNT(*) FROM User
-> ;
+-----+
| COUNT(*) |
+-----+
|      1000 |
+-----+
1 row in set (0.005 sec)

MySQL [main]> 
```

Itinerary Table Length

```
MySQL [main]> SELECT COUNT(*) FROM Itinerary
-> ;
+-----+
| COUNT(*) |
+-----+
|      2000 |
+-----+
1 row in set (0.006 sec)

MySQL [main]> 
```

Table Creation Commands

```
CREATE TABLE User (  
    user_id INT PRIMARY KEY AUTO_INCREMENT,  
    name VARCHAR(255),  
    email VARCHAR(255) UNIQUE,  
    password VARCHAR(255),  
    park_pref INT,  
    historical_landmark_pref INT,  
    historical_place_museum_pref INT,  
    museum_pref INT,  
    history_museum_pref INT,  
    tourist_attraction_pref INT,  
    wildlife_park_pref INT,  
    art_museum_pref INT,  
    aquarium_pref INT,  
    monument_pref INT,  
    hiking_area_pref INT,  
    zoo_pref INT,  
    catholic_cathedral_pref INT,  
    nature_preserve_pref INT,  
    amusement_park_pref INT,  
    garden_pref INT,  
    theme_park_pref INT,  
    water_park_pref INT,  
    scenic_spot_pref INT,  
    observatory_pref INT,  
    castle_pref INT,  
    archaeological_museum_pref INT,  
    public_beach_pref INT,  
    national_forest_pref INT,  
    catholic_church_pref INT,  
    heritage_museum_pref INT,  
    beach_pref INT,  
    synagogue_pref INT,  
    ecological_park_pref INT,  
    wax_museum_pref INT,  
    hindu_temple_pref INT,  
    wildlife_safari_park_pref INT,  
    buddhist_temple_pref INT,  
    animal_park_pref INT,  
    wildlife_refuge_pref INT,  
    heritage_building_pref INT,
```

```
vista_point_pref INT,  
national_park_pref INT,  
monastery_pref INT,  
fortress_pref INT,  
beach_pavilion_pref INT  
);
```

```
CREATE TABLE Itinerary (  
    itinerary_id INT PRIMARY KEY AUTO_INCREMENT,  
    user_id INT,  
    destination_city VARCHAR(100),  
    destination_state VARCHAR(100),  
    start_date DATE,  
    end_date DATE,  
    FOREIGN KEY (user_id) REFERENCES User(user_id) ON DELETE CASCADE  
);
```

```
CREATE TABLE Attraction (  
    attraction_id INT PRIMARY KEY AUTO_INCREMENT,  
    name VARCHAR(255) NOT NULL,  
    main_category VARCHAR(100),  
    rating FLOAT CHECK (rating >= 0 AND rating <= 5),  
    popularity INT,  
    description TEXT,  
    address VARCHAR(255),  
    city VARCHAR(100),  
    state VARCHAR(100)  
);
```

```
CREATE TABLE ItineraryItem (  
    item_id INT PRIMARY KEY AUTO_INCREMENT,  
    itinerary_id INT,  
    attraction_id INT,  
    day_number INT,  
    start_time DATETIME,  
    end_time DATETIME,  
    notes TEXT,  
    FOREIGN KEY (itinerary_id) REFERENCES Itinerary(itinerary_id) ON DELETE CASCADE,  
    FOREIGN KEY (attraction_id) REFERENCES Attraction(attraction_id)  
);
```

```
CREATE TABLE Route (  
    route_id INT PRIMARY KEY AUTO_INCREMENT,  
    from_item_id INT,  
    to_item_id INT,  
    distance DECIMAL(10,2),  
    travel_time INT, -- in minutes  
    transport_mode VARCHAR(100),  
    price_level INT,  
    FOREIGN KEY (from_item_id) REFERENCES ItineraryItem(item_id) ON DELETE  
CASCADE,  
    FOREIGN KEY (to_item_id) REFERENCES ItineraryItem(item_id) ON DELETE CASCADE  
);
```

Advanced queries:

1.

```
SELECT name FROM Attraction
WHERE city = 'Orlando'
UNION
SELECT name FROM Attraction
WHERE popularity > (
    SELECT AVG(popularity) FROM Attraction
);
```

```
MySQL [main]> SELECT name FROM Attraction
-> WHERE city = 'Orlando'
->
-> UNION
->
-> SELECT name FROM Attraction
-> WHERE popularity > (
->     SELECT AVG(popularity) FROM Attraction
-> ) LIMIT 15;
```

```
+-----+
| name                                     |
+-----+
| Urban Air Trampoline and Adventure Park |
| Magic Kingdom Park                     |
| Universal Orlando Resort                |
| Epcot                                   |
| Universal Studios Florida                |
| Disney's Animal Kingdom Theme Park      |
| SeaWorld Orlando                       |
| Universal Islands of Adventure          |
| ICON Park                               |
| Aquatica Orlando                       |
| Universal Volcano Bay                   |
| Gatorland                               |
| Lake Eola Park                          |
| The Wizarding World of Harry Potter - Diagon Alley |
| Fun Spot America                       |
+-----+
15 rows in set (0.005 sec)
```

```
EXPLAIN ANALYZE
no indices: 610 cost
```

```
| -> Table scan on <union temporary> (cost=591..610 rows=1329) (actual time=4.1..4.17 rows=554 loops=1)
|   -> Union materialize with deduplication (cost=590..590 rows=1329) (actual time=4.07..4.07 rows=554 loops=1)
|     -> Filter: (Attraction.city = 'Orlando') (cost=331 rows=307) (actual time=0.0938..1.27 rows=49 loops=1)
|       -> Table scan on Attraction (cost=331 rows=3068) (actual time=0.0468..1.01 rows=3123 loops=1)
|     -> Filter: (Attraction.popularity > (select #3)) (cost=127 rows=1023) (actual time=1.2..2.51 rows=544 loops=1)
|       -> Table scan on Attraction (cost=127 rows=3068) (actual time=0.0442..0.94 rows=3123 loops=1)
|       -> Select #3 (subquery in condition; run only once)
|         -> Aggregate: avg(Attraction.popularity) (cost=638 rows=1) (actual time=1.11..1.11 rows=1 loops=1)
|           -> Table scan on Attraction (cost=331 rows=3068) (actual time=0.0323..0.82 rows=3123 loops=1)
|
```

city: 267 cost

```
| -> Table scan on <union temporary> (cost=251..267 rows=1072) (actual time=3.8..3.87 rows=554 loops=1)
|   -> Union materialize with deduplication (cost=251..251 rows=1072) (actual time=3.8..3.8 rows=554 loops=1)
|     -> Index lookup on Attraction using idx_city (city='Orlando') (cost=17.1 rows=49) (actual time=0.672..0.758 rows=49 loops=1)
|     -> Filter: (Attraction.popularity > (select #3)) (cost=127 rows=1023) (actual time=1.44..2.74 rows=544 loops=1)
|       -> Table scan on Attraction (cost=127 rows=3068) (actual time=0.0747..1.03 rows=3123 loops=1)
|       -> Select #3 (subquery in condition; run only once)
|         -> Aggregate: avg(Attraction.popularity) (cost=638 rows=1) (actual time=1.33..1.33 rows=1 loops=1)
|           -> Table scan on Attraction (cost=331 rows=3068) (actual time=0.0579..0.943 rows=3123 loops=1)
|
```

```
| -> Table scan on <union temporary> (cost=322..331 rows=593) (actual time=5.19..5.26 rows=554 loops=1)
|   -> Union materialize with deduplication (cost=322..322 rows=593) (actual time=5.19..5.19 rows=554 loops=1)
|     -> Index lookup on Attraction using idx_city (city='Orlando') (cost=17.1 rows=49) (actual time=0.107..0.119 rows=49 loops=1)
|     -> Filter: (Attraction.popularity > (select #3)) (cost=245 rows=544) (actual time=0.0892..4.71 rows=544 loops=1)
|       -> Index range scan on Attraction using idx_popularity over (3089 <= popularity) (cost=245 rows=544) (actual time=0.0462..4.45 rows=544 loops=1)
|       -> Select #3 (subquery in condition; run only once)
|         -> Aggregate: avg(Attraction.popularity) (cost=638 rows=1) (actual time=1.07..1.07 rows=1 loops=1)
|           -> Covering index scan on Attraction using idx_popularity (cost=331 rows=3068) (actual time=0.0947..0.766 rows=3123 loops=1)
|
```

(city, popularity): 331 cost

popularity: 674

```
| -> Table scan on <union temporary> (cost=661..674 rows=851) (actual time=2.26..2.32 rows=554 loops=1)
|   -> Union materialize with deduplication (cost=661..661 rows=851) (actual time=2.26..2.26 rows=554 loops=1)
|     -> Filter: (Attraction.city = 'Orlando') (cost=331 rows=307) (actual time=0.0787..1.23 rows=49 loops=1)
|       -> Table scan on Attraction (cost=331 rows=3068) (actual time=0.0317..0.985 rows=3123 loops=1)
|     -> Filter: (Attraction.popularity > (select #3)) (cost=245 rows=544) (actual time=0.0194..0.792 rows=544 loops=1)
|       -> Index range scan on Attraction using idx_popularity over (3089 <= popularity) (cost=245 rows=544) (actual time=0.017..0.714 rows=544 loops=1)
|       -> Select #3 (subquery in condition; run only once)
|         -> Aggregate: avg(Attraction.popularity) (cost=638 rows=1) (actual time=0.986..0.986 rows=1 loops=1)
|           -> Covering index scan on Attraction using idx_popularity (cost=331 rows=3068) (actual time=0.0532..0.63 rows=3123 loops=1)
|
```

Index Configuration	Cost
None	610
city	267
popularity	674
city, popularity	331

We chose to use the city index configuration, because it was the most efficient

2.

```
SELECT A.name, A.main_category, A.rating
FROM Attraction A
JOIN (
    SELECT main_category, AVG(rating) AS avg_rating
    FROM Attraction
    GROUP BY main_category
) AS CategoryAverages ON A.main_category = CategoryAverages.main_category
WHERE A.rating > CategoryAverages.avg_rating;
```

```
MySQL [main]> SELECT A.name, A.main_category, A.rating
-> FROM Attraction A
-> JOIN (
->   SELECT main_category, AVG(rating) AS avg_rating
->   FROM Attraction
->   GROUP BY main_category
-> ) AS CategoryAverages ON A.main_category = CategoryAverages.main_category
-> WHERE A.rating > CategoryAverages.avg_rating LIMIT 15;
+-----+-----+-----+
| name | main_category | rating |
+-----+-----+-----+
| Forsyth Park | Park | 4.8 |
| Fountain at Forsyth Park | Historical landmark | 4.8 |
| Old Fort Jackson | Historical landmark | 4.7 |
| Hearse Ghost Tours | Tourist attraction | 4.9 |
| Juliette Gordon Low Birthplace Museum | History museum | 4.7 |
| Oglethorpe Square | Park | 4.7 |
| Emmet Park | Park | 4.6 |
| Graveface Museum | Museum | 4.8 |
| UGA Marine Education Center and Aquarium | Aquarium | 4.6 |
| Pin Point Heritage Museum | Museum | 4.9 |
| Ralph Mark Gilbert Civil Rights Museum | Museum | 4.7 |
| Savannah's Waterfront | Tourist attraction | 4.8 |
| Telfair Square | Park | 4.7 |
| Beach Institute African American Cultural Center | Museum | 4.8 |
| Pulaski Square | Park | 4.8 |
+-----+-----+-----+
15 rows in set (0.008 sec)
```

EXPLAIN ANALYZE

No config: Cost of 32545

```
| -> Nested loop inner join (cost=32545 rows=30677) (actual time=3.13..9.2 rows=1671 loops=1)
-> Filter: (A.main_category is not null) (cost=331 rows=3068) (actual time=0.0635..1.7 rows=3123 loops=1)
-> Table scan on A (cost=331 rows=3068) (actual time=0.0629..1.45 rows=3123 loops=1)
-> Filter: (A.rating > CategoryAverages.avg_rating) (cost=0.75..7.5 rows=10) (actual time=0.00209..0.00223 rows=0.535 loops=3123)
-> Covering index lookup on CategoryAverages using <auto_key0> (main_category=A.main_category) (cost=0.25..7.5 rows=30) (actual time=0.00181..0.002 rows=1 loops=3123)
-> Materialize (cost=0..0 rows=0) (actual time=3.05..3.05 rows=42 loops=1)
-> Table scan on <temporary> (actual time=3..3.01 rows=42 loops=1)
-> Aggregate using temporary table (actual time=3..3 rows=42 loops=1)
-> Table scan on Attraction (cost=331 rows=3068) (actual time=0.0333..1.17 rows=3123 loops=1)
|
```

city: (the same, city is not used): Cost of 32545

```
| -> Nested loop inner join (cost=32545 rows=30677) (actual time=3..8.49 rows=1671 loops=1)
-> Filter: (A.main_category is not null) (cost=331 rows=3068) (actual time=0.0695..1.48 rows=3123 loops=1)
-> Table scan on A (cost=331 rows=3068) (actual time=0.0688..1.25 rows=3123 loops=1)
-> Filter: (A.rating > CategoryAverages.avg_rating) (cost=0.75..7.5 rows=10) (actual time=0.00196..0.00209 rows=0.535 loops=3123)
-> Covering index lookup on CategoryAverages using <auto_key0> (main_category=A.main_category) (cost=0.25..7.5 rows=30) (actual time=0.0017..0.00188 rows=1 loops=3123)
-> Materialize (cost=0..0 rows=0) (actual time=2.91..2.91 rows=42 loops=1)
-> Table scan on <temporary> (actual time=2.84..2.85 rows=42 loops=1)
-> Aggregate using temporary table (actual time=2.84..2.84 rows=42 loops=1)
-> Table scan on Attraction (cost=331 rows=3068) (actual time=0.0389..1.18 rows=3123 loops=1)
|
```

main_category, (main_category, rating): Cost of 32545


```
| -> Nested loop inner join (cost=32545 rows=30677) (actual time=1.86..7.57 rows=1671 loops=1)
  -> Filter: (A.main_category is not null) (cost=331 rows=3068) (actual time=0.0919..1.68 rows=3123 loops=1)
    -> Table scan on A (cost=331 rows=3068) (actual time=0.0901..1.46 rows=3123 loops=1)
    -> Filter: (A.rating > CategoryAverages.avg_rating) (cost=579..7.5 rows=10) (actual time=0.00158..0.00173 rows=0.535 loops=3123)
      -> Covering index lookup on CategoryAverages using <auto_key0> (main_category=A.main_category) (cost=642..650 rows=30) (actual time=0.00132..0.00152 rows=1 loops=3123)
        -> Materialize (cost=642..642 rows=42) (actual time=1.74..1.74 rows=42 loops=1)
          -> Group aggregate: avg(Attraction.rating) (cost=638 rows=42) (actual time=0.0765..1.68 rows=42 loops=1)
      -> Covering index scan on Attraction using idx_category_rating (cost=331 rows=3068) (actual time=0.0511..0.91 rows=3123 loops=1)
    |
```

(main_category, rating): Cost of 32545

```
| -> Nested loop inner join (cost=32545 rows=30677) (actual time=1.82..7.62 rows=1671 loops=1)
  -> Filter: (A.main_category is not null) (cost=331 rows=3068) (actual time=0.0961..1.67 rows=3123 loops=1)
    -> Table scan on A (cost=331 rows=3068) (actual time=0.0948..1.44 rows=3123 loops=1)
    -> Filter: (A.rating > CategoryAverages.avg_rating) (cost=579..7.5 rows=10) (actual time=0.00161..0.00175 rows=0.535 loops=3123)
      -> Covering index lookup on CategoryAverages using <auto_key0> (main_category=A.main_category) (cost=642..650 rows=30) (actual time=0.00134..0.00154 rows=1 loops=3123)
        -> Materialize (cost=642..642 rows=42) (actual time=1.71..1.71 rows=42 loops=1)
          -> Group aggregate: avg(Attraction.rating) (cost=638 rows=42) (actual time=0.0628..1.66 rows=42 loops=1)
      -> Covering index scan on Attraction using idx_category_rating (cost=331 rows=3068) (actual time=0.0393..0.841 rows=3123 loops=1)
    |
```

main_category: Cost of 32545

```
| -> Nested loop inner join (cost=32545 rows=30677) (actual time=5.08..10.6 rows=1671 loops=1)
  -> Filter: (A.main_category is not null) (cost=331 rows=3068) (actual time=0.0666..1.54 rows=3123 loops=1)
    -> Table scan on A (cost=331 rows=3068) (actual time=0.0654..1.31 rows=3123 loops=1)
    -> Filter: (A.rating > CategoryAverages.avg_rating) (cost=579..7.5 rows=10) (actual time=0.00262..0.00274 rows=0.535 loops=3123)
      -> Covering index lookup on CategoryAverages using <auto_key0> (main_category=A.main_category) (cost=642..650 rows=30) (actual time=0.00237..0.00255 rows=1 loops=3123)
        -> Materialize (cost=642..642 rows=42) (actual time=4.99..4.99 rows=42 loops=1)
          -> Group aggregate: avg(Attraction.rating) (cost=638 rows=42) (actual time=0.239..4.93 rows=42 loops=1)
      -> Index scan on Attraction using idx_main_category (cost=331 rows=3068) (actual time=0.214..4.11 rows=3123 loops=1)
    |
```

Rating: Cost of 32545

```
| -> Nested loop inner join (cost=32545 rows=30677) (actual time=4.86..10.3 rows=1671 loops=1)
  -> Filter: (A.main_category is not null) (cost=331 rows=3068) (actual time=0.079..1.51 rows=3123 loops=1)
    -> Table scan on A (cost=331 rows=3068) (actual time=0.0776..1.3 rows=3123 loops=1)
    -> Filter: (A.rating > CategoryAverages.avg_rating) (cost=579..7.5 rows=10) (actual time=0.00252..0.00265 rows=0.535 loops=3123)
      -> Covering index lookup on CategoryAverages using <auto_key0> (main_category=A.main_category) (cost=642..650 rows=30) (actual time=0.00227..0.00245 rows=1 loops=3123)
        -> Materialize (cost=642..642 rows=42) (actual time=4.76..4.76 rows=42 loops=1)
          -> Group aggregate: avg(Attraction.rating) (cost=638 rows=42) (actual time=0.205..4.7 rows=42 loops=1)
      -> Index scan on Attraction using idx_main_category (cost=331 rows=3068) (actual time=0.181..3.93 rows=3123 loops=1)
    |
```

None of the index configurations we tried improved the performance, because we use AVG(rating), which requires a full scan of the whole table anyway. Therefore, we won't use any index for this query.

3.

```
SELECT A.name, A.popularity, A.state
FROM Attraction A
JOIN (
  SELECT state, MAX(popularity) AS max_popularity
  FROM Attraction
  GROUP BY state
) AS PopularAttractions ON A.state = PopularAttractions.state AND A.popularity =
PopularAttractions.max_popularity;
```

```
MySQL [main]> SELECT A.name, A.popularity, A.state
-> FROM Attraction A
-> JOIN (
->   SELECT state, MAX(popularity) AS max_popularity
->   FROM Attraction
->   GROUP BY state
-> ) AS PopularAttractions ON A.state = PopularAttractions.state AND A.popularity = PopularAttractions.max_popularity
-> LIMIT 15;

+-----+-----+-----+
| name                | popularity | state |
+-----+-----+-----+
| name                |           | state |
| Adventureland Park  | 7837      | IA    |
| Magic Kingdom Park  | 221929    | FL    |
| New England Aquarium | 23253     | MA    |
| Waterfront Park     | 3348      | VT    |
| National Aquarium   | 28759     | MD    |
| Temple Square       | 18601     | UT    |
| Houston Zoo         | 38124     | TX    |
| Liberty Bell        | 21681     | PA    |
| Phoenix Zoo         | 20865     | AZ    |
| Jenkinson's Aquarium | 4446      | NJ    |
| Story Land          | 4279      | NH    |
| New Castle Battery Park | 3448     | DE    |
| Palace Playland     | 2397      | ME    |
| Newport Aquarium    | 17994     | KY    |
+-----+-----+-----+
15 rows in set (0.008 sec)
```

EXPLAIN ANALYZE

Base case: Cost of 8021

```
| -> Nested loop inner join (cost=8021 rows=0) (actual time=5.81..10.2 rows=52 loops=1)
-> Filter: ((A.state is not null) and (A.popularity is not null)) (cost=331 rows=3068) (actual time=0.453..1.
91 rows=3123 loops=1)
-> Table scan on A (cost=331 rows=3068) (actual time=0.451..1.63 rows=3123 loops=1)
-> Covering index lookup on PopularAttractions using <auto_key0> (state=A.state, max_popularity=A.popularity)
(cost=0.25..2.51 rows=10) (actual time=0.00251..0.00252 rows=0.0167 loops=3123)
-> Materialize (cost=0..0 rows=0) (actual time=5.33..5.33 rows=51 loops=1)
-> Table scan on <temporary> (actual time=5.24..5.26 rows=51 loops=1)
-> Aggregate using temporary table (actual time=5.24..5.24 rows=51 loops=1)
-> Table scan on Attraction (cost=331 rows=3068) (actual time=0.329..3.69 rows=3123 loops=1)
|
```

State: Cost of 23668

```
| -> Nested loop inner join (cost=23668 rows=156468) (actual time=12..16.4 rows=52 loops=1)
-> Filter: ((A.state is not null) and (A.popularity is not null)) (cost=331 rows=3068) (actual time=0.121..1.
59 rows=3123 loops=1)
-> Table scan on A (cost=331 rows=3068) (actual time=0.119..1.29 rows=3123 loops=1)
-> Covering index lookup on PopularAttractions using <auto_key0> (state=A.state, max_popularity=A.popularity)
(cost=643..645 rows=10) (actual time=0.0046..0.00461 rows=0.0167 loops=3123)
-> Materialize (cost=643..643 rows=51) (actual time=11.9..11.9 rows=51 loops=1)
-> Group aggregate: max(Attraction.popularity) (cost=638 rows=51) (actual time=0.525..11.7 rows=51 lo
ops=1)
-> Index scan on Attraction using idx_state (cost=331 rows=3068) (actual time=0.488..11.1 rows=31
23 loops=1)
|
```

(state, popularity): Cost of 27

```
| -> Nested loop inner join (cost=27 rows=53.1) (actual time=0.494..0.829 rows=52 loops=1)
    -> Filter: ((PopularAttractions.state is not null) and (PopularAttractions.max_popularity is not null)) (cost=64.5..8.35 rows=52) (actual time=0.47..0.486 rows=51 loops=1)
        -> Table scan on PopularAttractions (cost=65.7..68.8 rows=52) (actual time=0.467..0.476 rows=51 loops=1)
            -> Materialize (cost=65.7..65.7 rows=52) (actual time=0.465..0.465 rows=51 loops=1)
                -> Covering index skip scan for grouping on Attraction using idx_state_popularity (cost=60.5 rows=52) (actual time=0.103..0.435 rows=51 loops=1)
            -> Index lookup on A using idx_state_popularity (state=PopularAttractions.state, popularity=PopularAttractions.max_popularity) (cost=0.257 rows=1.02) (actual time=0.00592..0.00646 rows=1.02 loops=51)
        |
```

Popularity: Cost of 2199

```
| -> Nested loop inner join (cost=2199 rows=529) (actual time=5.03..5.28 rows=52 loops=1)
    -> Filter: (PopularAttractions.max_popularity is not null) (cost=0.113..348 rows=3068) (actual time=4.99..5 rows=51 loops=1)
        -> Table scan on PopularAttractions (cost=2.5..2.5 rows=0) (actual time=4.99..4.99 rows=51 loops=1)
            -> Materialize (cost=0..0 rows=0) (actual time=4.99..4.99 rows=51 loops=1)
                -> Table scan on <temporary> (actual time=4.95..4.96 rows=51 loops=1)
                    -> Aggregate using temporary table (actual time=4.95..4.95 rows=51 loops=1)
                        -> Table scan on Attraction (cost=331 rows=3068) (actual time=0.0769..2.1 rows=3123 loops=1)
                    -> Filter: (A.state = PopularAttractions.state) (cost=0.431 rows=0.172) (actual time=0.00461..0.00511 rows=1.02 loops=51)
                        -> Index lookup on A using idx_popularity (popularity=PopularAttractions.max_popularity) (cost=0.431 rows=1.72) (actual time=0.00432..0.00475 rows=1.04 loops=51)
                    |
```

state + popularity: Cost of 2199

```
| -> Nested loop inner join (cost=2199 rows=153) (actual time=4.4..4.61 rows=52 loops=1)
    -> Filter: (PopularAttractions.max_popularity is not null) (cost=643..348 rows=3068) (actual time=4.37..4.39 rows=51 loops=1)
        -> Table scan on PopularAttractions (cost=643..646 rows=51) (actual time=4.37..4.38 rows=51 loops=1)
            -> Materialize (cost=643..643 rows=51) (actual time=4.37..4.37 rows=51 loops=1)
                -> Group aggregate: max(Attraction.popularity) (cost=638 rows=51) (actual time=0.124..4.33 rows=51 loops=1)
                    -> Index scan on Attraction using idx_state (cost=331 rows=3068) (actual time=0.108..3.9 rows=3123 loops=1)
                -> Filter: (A.state = PopularAttractions.state) (cost=0.431 rows=0.05) (actual time=0.00366..0.00417 rows=1.02 loops=51)
                    -> Index lookup on A using idx_popularity (popularity=PopularAttractions.max_popularity) (cost=0.431 rows=1.72) (actual time=0.00317..0.0036 rows=1.04 loops=51)
                |
```

Since indexing with the composite index of (state, popularity) is clearly the lowest cost, we will use the index (state, popularity)

4.

```
SELECT A.city, A.state, AVG(A.rating) AS avg_city_rating
FROM Attraction A
GROUP BY A.city, A.state
HAVING AVG(A.rating) > (
    SELECT AVG(B.rating)
    FROM Attraction B
    WHERE B.state = A.state
);
```

```
MySQL [main]> SELECT A.city, A.state, AVG(A.rating) AS avg_city_rating
-> FROM Attraction A
-> GROUP BY A.city, A.state
-> HAVING AVG(A.rating) > (
->     SELECT AVG(B.rating)
->     FROM Attraction B
->     WHERE B.state = A.state
-> ) LIMIT 15;
```

city	state	avg_city_rating
Savannah	GA	4.66999997138977
Lafayette	LA	4.609999942779541
Orlando	FL	4.635555511050754
Orlando	OR	4.599999904632568
Boston	MA	4.617777707841661
Burlington	VT	4.564285618918283
Greenville	SC	4.628571442195347
Houston	TX	4.58269228385045
Philadelphia	PA	4.609835992093946
New-haven	CT	4.583333253860474
West-palm-beach	FL	4.613043432650358
Hartford	CT	4.588888830608791
Raleigh	NC	4.637777784135607
Dover	MA	4.8500001430511475
Dover	NY	4.6000001430511475

15 rows in set (0.140 sec)

Baseline cost: Cost of 331 + (85.6 * 135) + (54.9 * 3068) = 180320.2

```
| -> Filter: (??? > `(select #2)`) (actual time=161..161 rows=62 loops=1)
|   -> Table scan on <temporary> (actual time=161..161 rows=135 loops=1)
|     -> Aggregate using temporary table (actual time=161..161 rows=135 loops=1)
|       -> Table scan on A (cost=331 rows=3068) (actual time=0.079..1.51 rows=3123 loops=1)
| -> Select #2 (subquery in projection; dependent)
|   -> Aggregate: avg(B.rating) (cost=85.6 rows=1) (actual time=1.15..1.15 rows=1 loops=135)
|     -> Filter: (B.state = A.state) (cost=54.9 rows=307) (actual time=0.291..1.14 rows=105 loops=135)
|       -> Table scan on B (cost=54.9 rows=3068) (actual time=0.0105..0.858 rows=3123 loops=135)
|
```

(state, popularity): Cost of 331 + (27.1 * 135) + 21.1 * 135 = 6838

```
| -> Filter: (??? > `(select #2)`) (actual time=28..28.1 rows=62 loops=1)
    -> Table scan on <temporary> (actual time=28..28 rows=135 loops=1)
        -> Aggregate using temporary table (actual time=28..28 rows=135 loops=1)
            -> Table scan on A (cost=331 rows=3068) (actual time=0.423..1.91 rows=3123 loops=1)
-> Select #2 (subquery in projection; dependent)
    -> Aggregate: avg(B.rating) (cost=27.1 rows=1) (actual time=0.168..0.168 rows=1 loops=135)
        -> Index lookup on B using idx_state_popularity (state=A.state) (cost=21.1 rows=60.2) (actual time=0.0595
..0.159 rows=105 loops=135)
```

(state): Cost of 331 + (27.1 * 135) + 21.1 * 135 = 6838

```
| -> Filter: (avg_city_rating > `(select #2)`) (actual time=10.2..10.3 rows=62 loops=1)
    -> Table scan on <temporary> (actual time=10.2..10.3 rows=135 loops=1)
        -> Aggregate using temporary table (actual time=10.2..10.2 rows=135 loops=1)
            -> Table scan on A (cost=331 rows=3068) (actual time=0.0718..1.28 rows=3123 loops=1)
-> Select #2 (subquery in projection; dependent)
    -> Aggregate: avg(B.rating) (cost=15.2 rows=1) (actual time=0.0463..0.0463 rows=1 loops=135)
        -> Covering index lookup on B using idx_state_rating (state=A.state) (cost=9.23 rows=60.2) (actual time=0.0143..0.0393 rows=105 loops=135)
```

(state, rating): Cost of 331 + 15.2 * 135 + 9.23 * 135 = 3629.05

```
| EXPLAIN
+-----+
|
+-----+
| -> Filter: (??? > `(select #2)`) (actual time=10.7..10.8 rows=62 loops=1)
    -> Table scan on <temporary> (actual time=10.7..10.7 rows=135 loops=1)
        -> Aggregate using temporary table (actual time=10.7..10.7 rows=135 loops=1)
            -> Table scan on A (cost=331 rows=3068) (actual time=0.0519..1.22 rows=3123 loops=1)
-> Select #2 (subquery in projection; dependent)
    -> Aggregate: avg(B.rating) (cost=15.2 rows=1) (actual time=0.0488..0.0489 rows=1 loops=135)
        -> Covering index lookup on B using idx_state_rating (state=A.state) (cost=9.23 rows=60.2) (actual time=0.0161..0.0417 rows=105 loops=135)
|
+-----+
+-----+
```

(state, city, rating): Cost of 331 + 11.6 * 135 = 1897

```
| -> Filter: (??? > `(select #2)`) (actual time=10.5..10.5 rows=62 loops=1)
    -> Table scan on <temporary> (actual time=10.5..10.5 rows=135 loops=1)
        -> Aggregate using temporary table (actual time=10.5..10.5 rows=135 loops=1)
            -> Covering index scan on A using idx_state_city_rating (cost=331 rows=3068) (actual time=0.0476..1.07 rows=3123 loops=1)
-> Select #2 (subquery in projection; dependent)
    -> Aggregate: avg(B.rating) (cost=17.6 rows=1) (actual time=0.0476..0.0476 rows=1 loops=135)
        -> Covering index lookup on B using idx_state_city_rating (state=A.state) (cost=11.6 rows=60.2) (actual time=0.0124..0.0394 rows=105 loops=135)
|
+-----+
+-----+
```

It looks like the cost is most optimal with the (state, city, rating) composite index. In addition, we predict we will usually be getting the state, city, and rating together. Therefore, we will use the (state, city, rating) index.

Final Index Design

Index: city, (state, popularity), (state, city, rating)