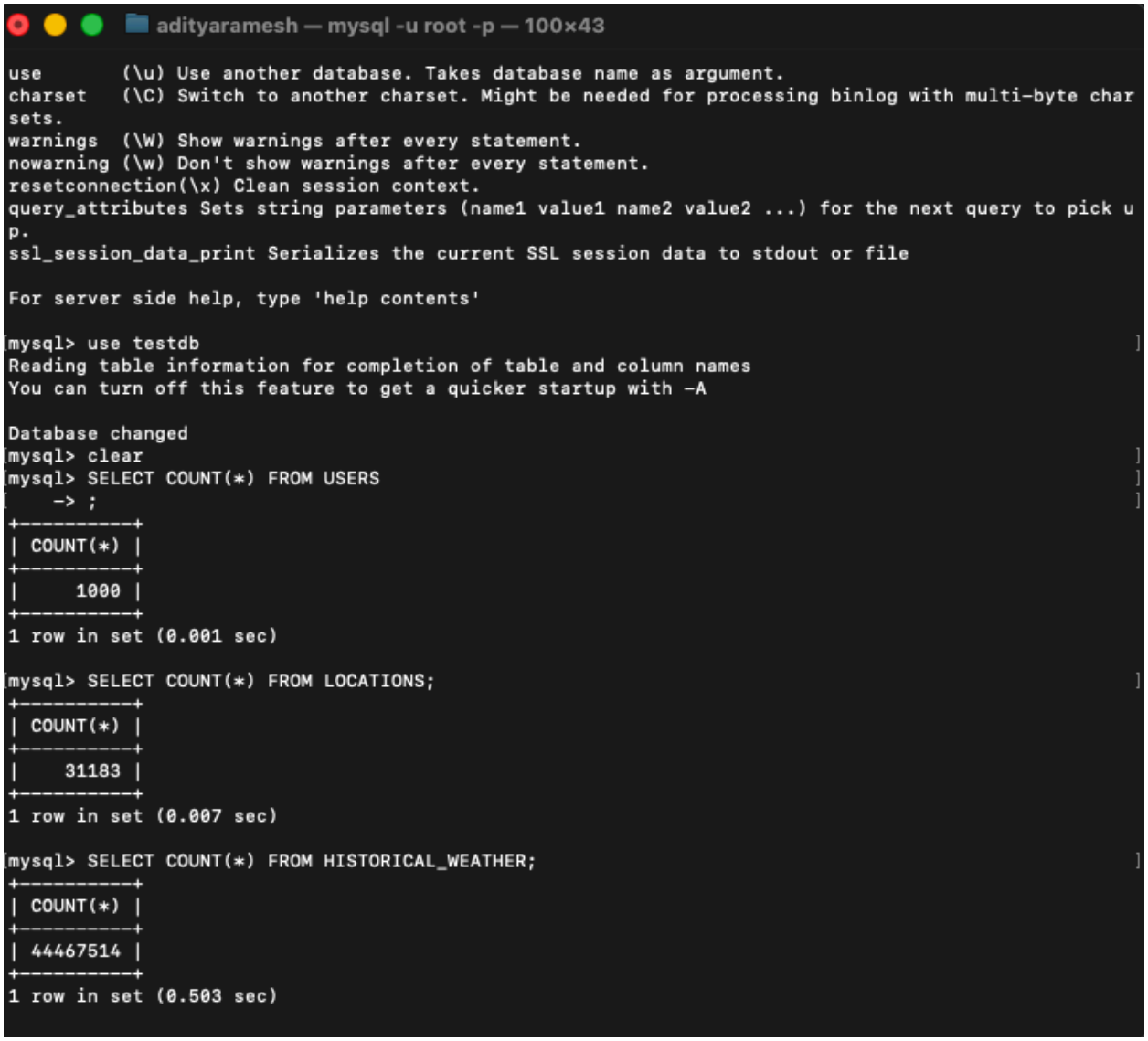
**DDL Commands**

```
CREATE TABLE USERS (
    userID INT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(255) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL
);

CREATE TABLE LOCATIONS (
    CITY VARCHAR(255) NOT NULL,
    STATE VARCHAR(100) NOT NULL,
    LAT DECIMAL(8, 4) NOT NULL,
    LNG DECIMAL(9, 4) NOT NULL,
    RND_LAT DECIMAL(8, 4) NOT NULL,
    RND_LNG DECIMAL(9, 4) NOT NULL,

    PRIMARY KEY (CITY, STATE)
);

CREATE TABLE TRIP_PLANS (
    tripID INT PRIMARY KEY AUTO_INCREMENT,
    userID INT NOT NULL,
    planName VARCHAR(255) NOT NULL,
    startDate DATE,
    endDate DATE,
    FOREIGN KEY (userID) REFERENCES USERS(userID) ON DELETE CASCADE
);

CREATE TABLE TRIP_LOCATIONS (
    tripID INT NOT NULL,
    CITY VARCHAR(255) NOT NULL,
    STATE VARCHAR(100) NOT NULL,
    PRIMARY KEY (tripID, CITY, STATE),
    FOREIGN KEY (tripID) REFERENCES TRIP_PLANS(tripID) ON DELETE CASCADE,
    FOREIGN KEY (CITY, STATE) REFERENCES LOCATIONS(CITY, STATE) ON DELETE
CASCADE
);

CREATE TABLE PREFERENCES (
    preferenceID INT PRIMARY KEY AUTO_INCREMENT,
    tripID INT NOT NULL,
    featureName VARCHAR(255) NOT NULL,
    preferredValue INT NOT NULL,
    FOREIGN KEY (tripID) REFERENCES TRIP_PLANS(tripID) ON DELETE CASCADE);
```

```
CREATE TABLE HISTORICAL_WEATHER (
    LAT DECIMAL(8, 4) NOT NULL,
    LNG DECIMAL(9, 4) NOT NULL,
    DATE DATE NOT NULL,
    TAVG DECIMAL(5, 1) NOT NULL,
    RND_LAT DECIMAL(8, 4) NOT NULL,
    RND_LNG DECIMAL(9, 4) NOT NULL,
    H_MONTH INT NOT NULL,
    H_DAY INT NOT NULL,
    PRCP INT NOT NULL,
    PRIMARY KEY (LAT, LNG, DATE)
);
```

**Local DB and Count Proof:**



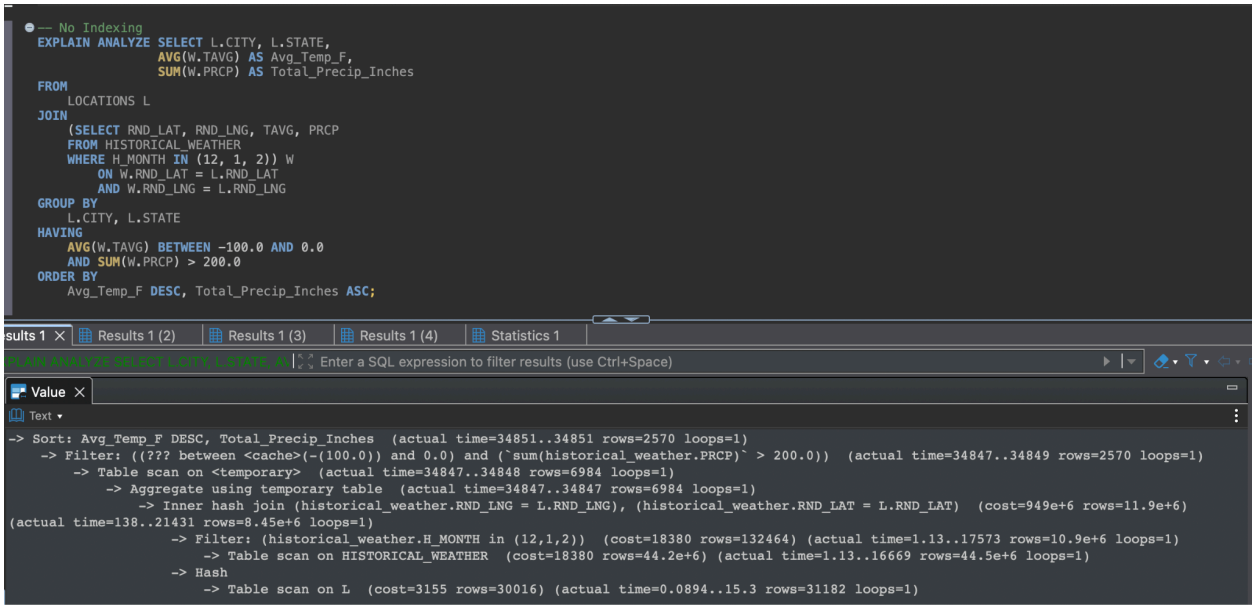
**Query 1: Locations with most Snowfall**

```
EXPLAIN ANALYZE SELECT L.CITY, L.STATE,
                       AVG(W.TAVG) AS Avg_Temp_F,
                          SUM(W.PRCP) AS Total_Precip_Inches
FROM
   LOCATIONS L
JOIN
   (SELECT RND_LAT, RND_LNG, TAVG, PRCP
   FROM HISTORICAL_WEATHER
   WHERE H_MONTH IN (12, 1, 2)) W
         ON W.RND_LAT = L.RND_LAT
                  AND W.RND_LNG = L.RND_LNG
GROUP BY
   L.CITY, L.STATE
HAVING
   AVG(W.TAVG) BETWEEN -100.0 AND 0.0
   AND SUM(W.PRCP) > 200.0
ORDER BY
   Avg_Temp_F DESC, Total_Precip_Inches ASC;
```

Top 15 Results

| | CITY | STATE | Avg_Temp_F | Total_Precip |
|---|---|---|---|---|
| 1 | Castle Valley | UT | -0.01453 | 7,079 |
| 2 | Hoonah | AK | -0.08042 | 67,878 |
| 3 | Whitestone Loggi | AK | -0.08042 | 67,878 |
| 4 | Enterprise | UT | -0.09698 | 18,407 |
| 5 | Cheswick | PA | -0.11301 | 37,530 |
| 6 | Oakmont | PA | -0.11301 | 37,530 |
| 7 | Springdale | PA | -0.11301 | 37,530 |
| 8 | Verona | PA | -0.11301 | 37,530 |
| 9 | Evans | CO | -0.17905 | 6,252 |
| 10 | Garden City | CO | -0.17905 | 6,252 |
| 11 | New Philadelphia | OH | -0.21251 | 28,787 |
| 12 | Cheraw | CO | -0.24206 | 7,315 |
| 13 | Springer | NM | -0.24427 | 462 |
| 14 | Franklin | IN | -0.27667 | 4,073 |
| 15 | Osage City | KS | -0.30883 | 16,268 |

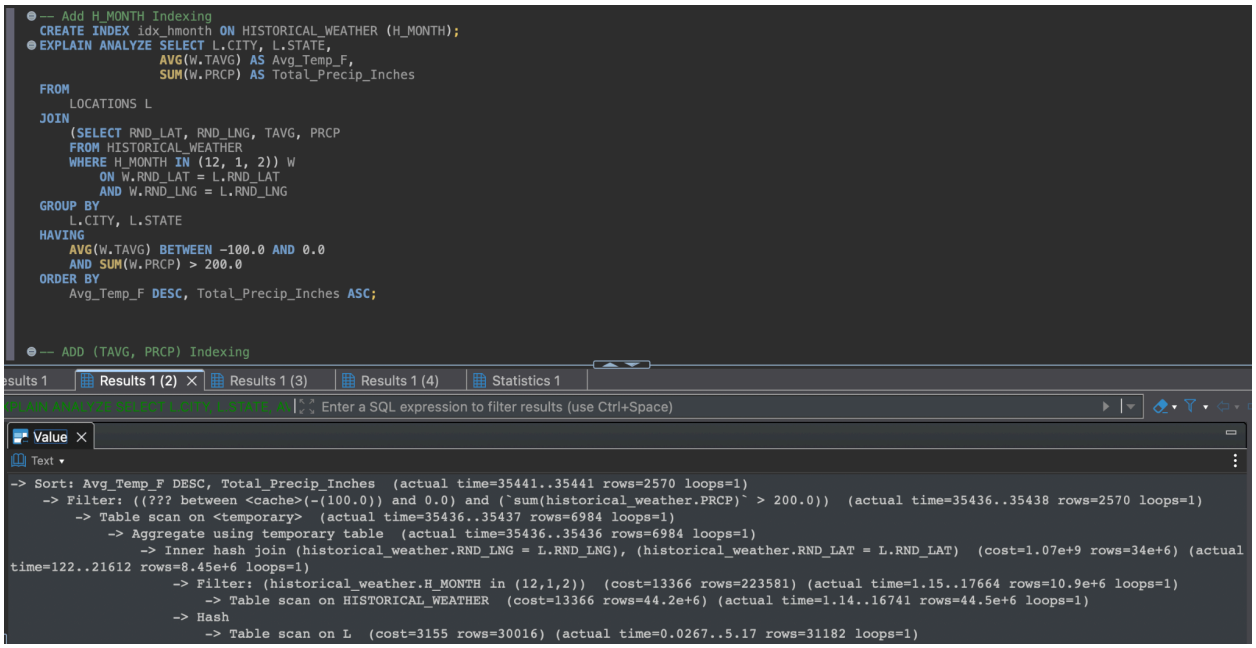
Before Indexing:



```
-- No Indexing
EXPLAIN ANALYZE SELECT L.CITY, L.STATE,
                AVG(W.TAVG) AS Avg_Temp_F,
                SUM(W.PRCP) AS Total_Precip_Inches
FROM
    LOCATIONS L
JOIN
    (SELECT RND_LAT, RND_LNG, TAVG, PRCP
    FROM HISTORICAL_WEATHER
    WHERE H_MONTH IN (12, 1, 2)) W
        ON W.RND_LAT = L.RND_LAT
        AND W.RND_LNG = L.RND_LNG
GROUP BY
    L.CITY, L.STATE
HAVING
    AVG(W.TAVG) BETWEEN -100.0 AND 0.0
    AND SUM(W.PRCP) > 200.0
ORDER BY
    Avg_Temp_F DESC, Total_Precip_Inches ASC;
```

```
sults 1 | Results 1 (2) | Results 1 (3) | Results 1 (4) | Statistics 1
Enter a SQL expression to filter results (use Ctrl+Space)
Value
Text
-> Sort: Avg_Temp_F DESC, Total_Precip_Inches  (actual time=34851..34851 rows=2570 loops=1)
    -> Filter: ((??? between <cache>(-(100.0)) and 0.0) and (`sum(historical_weather.PRCP)` > 200.0))  (actual time=34847..34849 rows=2570 loops=1)
        -> Table scan on <temporary>  (actual time=34847..34848 rows=6984 loops=1)
            -> Aggregate using temporary table  (actual time=34847..34847 rows=6984 loops=1)
                -> Inner hash join (historical_weather.RND_LNG = L.RND_LNG), (historical_weather.RND_LAT = L.RND_LAT)  (cost=949e+6 rows=11.9e+6)
(actual time=138..21431 rows=8.45e+6 loops=1)
                    -> Filter: (historical_weather.H_MONTH in (12,1,2))  (cost=18380 rows=132464) (actual time=1.13..17573 rows=10.9e+6 loops=1)
                        -> Table scan on HISTORICAL_WEATHER  (cost=18380 rows=44.2e+6) (actual time=1.13..16669 rows=44.5e+6 loops=1)
                    -> Hash
                        -> Table scan on L  (cost=3155 rows=30016) (actual time=0.0894..15.3 rows=31182 loops=1)
```

Adding Index on H_MONTH

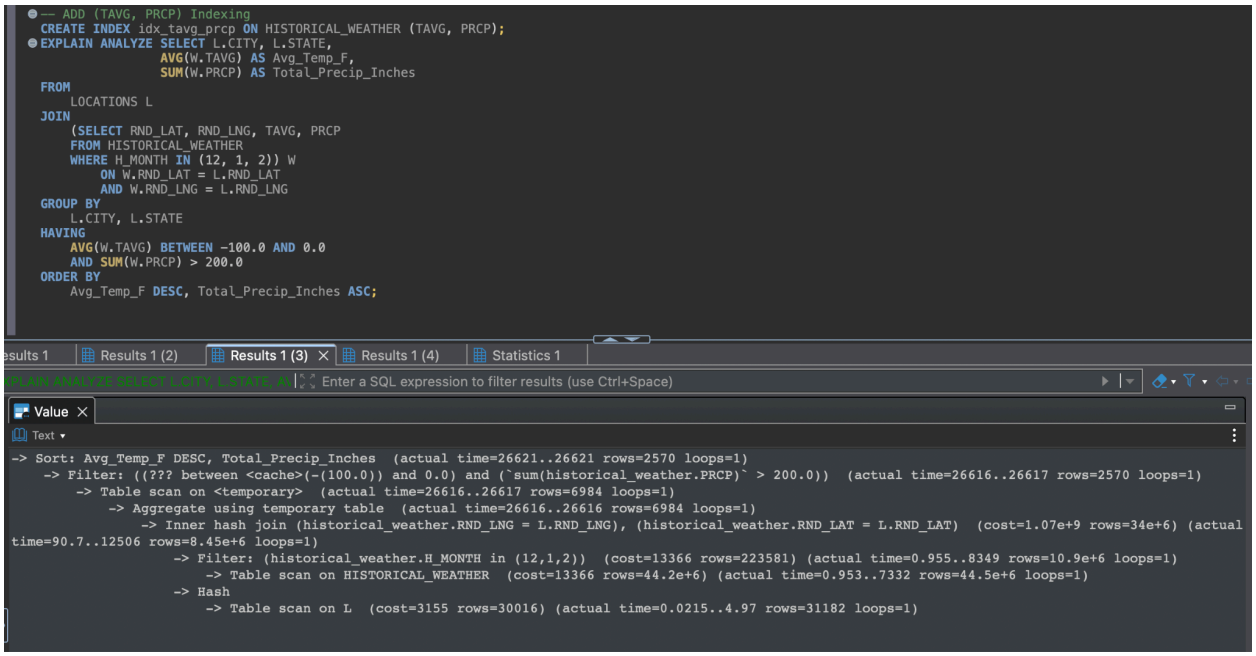


```
-- Add H_MONTH Indexing
CREATE INDEX idx_hmonth ON HISTORICAL_WEATHER (H_MONTH);
EXPLAIN ANALYZE SELECT L.CITY, L.STATE,
                AVG(W.TAVG) AS Avg_Temp_F,
                SUM(W.PRCP) AS Total_Precip_Inches
FROM
    LOCATIONS L
JOIN
    (SELECT RND_LAT, RND_LNG, TAVG, PRCP
    FROM HISTORICAL_WEATHER
    WHERE H_MONTH IN (12, 1, 2)) W
        ON W.RND_LAT = L.RND_LAT
        AND W.RND_LNG = L.RND_LNG
GROUP BY
    L.CITY, L.STATE
HAVING
    AVG(W.TAVG) BETWEEN -100.0 AND 0.0
    AND SUM(W.PRCP) > 200.0
ORDER BY
    Avg_Temp_F DESC, Total_Precip_Inches ASC;


-- ADD (TAVG, PRCP) Indexing
```

```
sults 1 | Results 1 (2) | Results 1 (3) | Results 1 (4) | Statistics 1
Enter a SQL expression to filter results (use Ctrl+Space)
Value
Text
-> Sort: Avg_Temp_F DESC, Total_Precip_Inches  (actual time=35441..35441 rows=2570 loops=1)
    -> Filter: ((??? between <cache>(-(100.0)) and 0.0) and (`sum(historical_weather.PRCP)` > 200.0))  (actual time=35436..35438 rows=2570 loops=1)
        -> Table scan on <temporary>  (actual time=35436..35437 rows=6984 loops=1)
            -> Aggregate using temporary table  (actual time=35436..35436 rows=6984 loops=1)
                -> Inner hash join (historical_weather.RND_LNG = L.RND_LNG), (historical_weather.RND_LAT = L.RND_LAT)  (cost=1.07e+9 rows=34e+6) (actual
time=122..21612 rows=8.45e+6 loops=1)
                    -> Filter: (historical_weather.H_MONTH in (12,1,2))  (cost=13366 rows=223581) (actual time=1.15..17664 rows=10.9e+6 loops=1)
                        -> Table scan on HISTORICAL_WEATHER  (cost=13366 rows=44.2e+6) (actual time=1.14..16741 rows=44.5e+6 loops=1)
                    -> Hash
                        -> Table scan on L  (cost=3155 rows=30016) (actual time=0.0267..5.17 rows=31182 loops=1)
```

After we index on H_MONTH, our filtering for the winter months becomes faster as our cost goes from 18380 to 13366. This is because we now have an index for our months allowing us to scan for fewer rows. However our biggest costs (e9 costs) have still not shrunken, making the gains with this indexing quite small.

## Adding Index on TAVG, PRCP

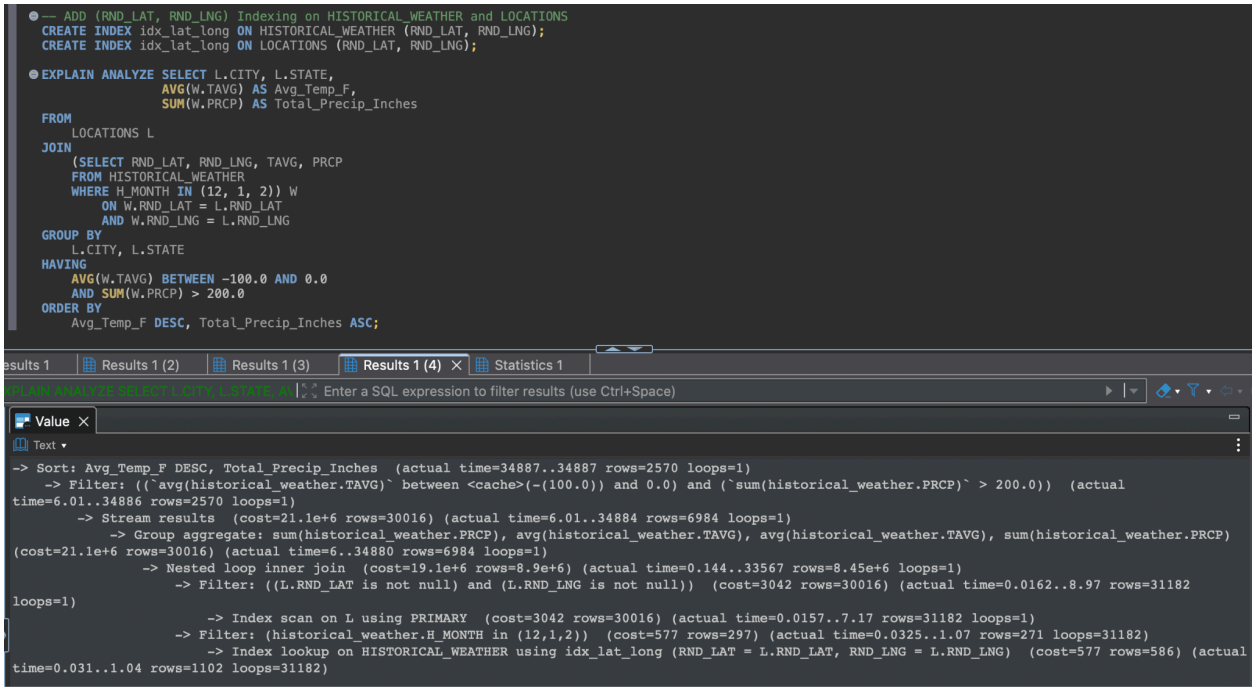


```
-- ADD (TAVG, PRCP) Indexing
CREATE INDEX idx_tavg_prcp ON HISTORICAL_WEATHER (TAVG, PRCP);
EXPLAIN ANALYZE SELECT L.CITY, L.STATE,
                AVG(W.TAVG) AS Avg_Temp_F,
                SUM(W.PRCP) AS Total_Precip_Inches
FROM
    LOCATIONS L
JOIN
    (SELECT RND_LAT, RND_LNG, TAVG, PRCP
    FROM HISTORICAL_WEATHER
    WHERE H_MONTH IN (12, 1, 2)) W
        ON W.RND_LAT = L.RND_LAT
        AND W.RND_LNG = L.RND_LNG
GROUP BY
    L.CITY, L.STATE
HAVING
    AVG(W.TAVG) BETWEEN -100.0 AND 0.0
    AND SUM(W.PRCP) > 200.0
ORDER BY
    Avg_Temp_F DESC, Total_Precip_Inches ASC;
```

```
esults 1 | Results 1 (2) | Results 1 (3) | Results 1 (4) | Statistics 1
Enter a SQL expression to filter results (use Ctrl+Space)
Value
Text
-> Sort: Avg_Temp_F DESC, Total_Precip_Inches  (actual time=26621..26621 rows=2570 loops=1)
    -> Filter: ((??? between <cache>(-(100.0)) and 0.0) and (`sum(historical_weather.PRCP)` > 200.0))  (actual time=26616..26617 rows=2570 loops=1)
        -> Table scan on <temporary>  (actual time=26616..26617 rows=6984 loops=1)
            -> Aggregate using temporary table  (actual time=26616..26616 rows=6984 loops=1)
                -> Inner hash join (historical_weather.RND_LNG = L.RND_LNG), (historical_weather.RND_LAT = L.RND_LAT)  (cost=1.07e+9 rows=34e+6) (actual
time=90.7..12506 rows=8.45e+6 loops=1)
                    -> Filter: (historical_weather.H_MONTH in (12,1,2))  (cost=13366 rows=223581) (actual time=0.955..8349 rows=10.9e+6 loops=1)
                        -> Table scan on HISTORICAL_WEATHER  (cost=13366 rows=44.2e+6) (actual time=0.953..7332 rows=44.5e+6 loops=1)
                    -> Hash
                        -> Table scan on L  (cost=3155 rows=30016) (actual time=0.0215..4.97 rows=31182 loops=1)
```

Unfortunately, we did not gain any major performance here. This is because although TAVG and PRCP are in our HAVING clause, it doesn't help filtering our rows down in the WHERE clause which is where we have our big costs to performance. It seems like indexing our aggregate functions isn't providing much performance benefit.

Adding Index on HW.RND_LAT, HW.RND_LNG, L.RND_LAT, L.RND_LNG



```
-- ADD (RND_LAT, RND_LNG) Indexing on HISTORICAL_WEATHER and LOCATIONS
CREATE INDEX idx_lat_long ON HISTORICAL_WEATHER (RND_LAT, RND_LNG);
CREATE INDEX idx_lat_long ON LOCATIONS (RND_LAT, RND_LNG);

EXPLAIN ANALYZE SELECT L.CITY, L.STATE,
                AVG(W.TAVG) AS Avg_Temp_F,
                SUM(W.PRCP) AS Total_Precip_Inches
FROM
    LOCATIONS L
JOIN
    (SELECT RND_LAT, RND_LNG, TAVG, PRCP
    FROM HISTORICAL_WEATHER
    WHERE H_MONTH IN (12, 1, 2)) W
        ON W.RND_LAT = L.RND_LAT
        AND W.RND_LNG = L.RND_LNG
GROUP BY
    L.CITY, L.STATE
HAVING
    AVG(W.TAVG) BETWEEN -100.0 AND 0.0
    AND SUM(W.PRCP) > 200.0
ORDER BY
    Avg_Temp_F DESC, Total_Precip_Inches ASC;
```

esults 1 | Results 1 (2) | Results 1 (3) | Results 1 (4) | Statistics 1

Enter a SQL expression to filter results (use Ctrl+Space)

Value

Text

```
-> Sort: Avg_Temp_F DESC, Total_Precip_Inches  (actual time=34887..34887 rows=2570 loops=1)
    -> Filter: ((`avg(historical_weather.TAVG)` between <cache>(-(100.0)) and 0.0) and (`sum(historical_weather.PRCP)` > 200.0))  (actual
time=6.01..34886 rows=2570 loops=1)
        -> Stream results  (cost=21.1e+6 rows=30016) (actual time=6.01..34884 rows=6984 loops=1)
            -> Group aggregate: sum(historical_weather.PRCP), avg(historical_weather.TAVG), avg(historical_weather.TAVG), sum(historical_weather.PRCP)
(cost=21.1e+6 rows=30016) (actual time=6..34880 rows=6984 loops=1)
                -> Nested loop inner join  (cost=19.1e+6 rows=8.9e+6) (actual time=0.144..33567 rows=8.45e+6 loops=1)
                    -> Filter: ((L.RND_LAT is not null) and (L.RND_LNG is not null))  (cost=3042 rows=30016) (actual time=0.0162..8.97 rows=31182
loops=1)
                        -> Index scan on L using PRIMARY  (cost=3042 rows=30016) (actual time=0.0157..7.17 rows=31182 loops=1)
                    -> Filter: (historical_weather.H_MONTH in (12,1,2))  (cost=577 rows=297) (actual time=0.0325..1.07 rows=271 loops=31182)
                        -> Index lookup on HISTORICAL_WEATHER using idx_lat_long (RND_LAT = L.RND_LAT, RND_LNG = L.RND_LNG)  (cost=577 rows=586) (actual
time=0.031..1.04 rows=1102 loops=31182)
```

Finally when we add in indexing for our lat/lng location parameters, we see our costs fall significantly. This is because we are now doing indexed lookups for each of our rows instead of a giant hash table without it being indexed. This reduces each lookup to 577 as opposed to 3155, which also reduces our total costs. For example our aggregate function goes from 946 e9 to just 21.6 e6, which is orders of magnitudes more efficient.

This query is the final indexing design we will use, since it had the greatest impact on our query performance. Just indexing by H_MONTHS or as well as TAVG and PRCP did give us small benefits, but they were so marginal that it didn't make a real difference to our overall performance.

**Query 2: Hotter than Avg Summer**

```
SELECT L.CITY,L.STATE,AVG(S.TAVG) AS City_Avg_Summer_Temp
FROM LOCATIONS L
JOIN (SELECT RND_LAT,RND_LNG,TAVG FROM HISTORICAL_WEATHER WHERE
H_MONTH IN (6,7,8)) S
  ON S.RND_LAT=L.RND_LAT AND S.RND_LNG=L.RND_LNG
GROUP BY L.CITY,L.STATE
HAVING AVG(S.TAVG) > (
  SELECT AVG(T.TAVG) FROM (SELECT TAVG FROM HISTORICAL_WEATHER WHERE
H_MONTH IN (6,7,8)) T
)
ORDER BY City_Avg_Summer_Temp DESC;
```

Top 15 Results

| | CITY | STATE | City_Avg_Summer_Temp |
|---|---|---|---|
| 1 | Guadalupe | AZ | 352.4576 |
| 2 | Laughlin | NV | 349.73701 |
| 3 | Katherine | AZ | 348.22572 |
| 4 | Bluewater | AZ | 346.07022 |
| 5 | Bluewater | CA | 346.07022 |
| 6 | Mesa Verde | CA | 344.81887 |
| 7 | Quartzsite | AZ | 344.6159 |
| 8 | Indio | CA | 343.99825 |
| 9 | Parker Strip | AZ | 342.51993 |
| 10 | Ehrenberg | AZ | 341.97886 |
| 11 | Needles | CA | 340.472 |
| 12 | Gila Crossing | AZ | 339.72583 |
| 13 | Komatke | AZ | 339.72583 |
| 14 | Winterhaven | CA | 339.23982 |
| 15 | El Mirage | AZ | 338.74237 |

## Before Indexing

```
-- Before Indexing
EXPLAIN ANALYZE SELECT L.CITY,L.STATE,AVG(S.TAVG) AS City_Avg_Summer_Temp
FROM LOCATIONS L
JOIN (SELECT RND_LAT,RND_LNG,TAVG FROM HISTORICAL_WEATHER WHERE H_MONTH IN (6,7,8)) S
  ON S.RND_LAT=L.RND_LAT AND S.RND_LNG=L.RND_LNG
GROUP BY L.CITY,L.STATE
HAVING AVG(S.TAVG) > (
  SELECT AVG(T.TAVG) FROM (SELECT TAVG FROM HISTORICAL_WEATHER WHERE H_MONTH IN (6,7,8)) T
)
ORDER BY City_Avg_Summer_Temp DESC;
```

Results 1 | Results 1 (2) | Results 1 (3) | Results 1 (4) | Statistics 1

Enter a SQL expression to filter results (use Ctrl+Space)

Value

Text

```
-> Sort: City_Avg_Summer_Temp DESC  (actual time=49307..49307 rows=4868 loops=1)
    -> Filter: (??? > (select #3))  (actual time=49303..49304 rows=4868 loops=1)
        -> Table scan on <temporary>  (actual time=33288..33289 rows=6947 loops=1)
            -> Aggregate using temporary table  (actual time=33288..33288 rows=6947 loops=1)
                -> Inner hash join (historical_weather.RND_LNG = L.RND_LNG), (historical_weather.RND_LAT = L.RND_LAT)  (cost=949e+6 rows=11.9e+6)
(actual time=129..22127 rows=8.73e+6 loops=1)
                    -> Filter: (historical_weather.H_MONTH in (6,7,8))  (cost=18380 rows=132464) (actual time=1.96..18009 rows=11.3e+6 loops=1)
                        -> Table scan on HISTORICAL_WEATHER  (cost=18380 rows=44.2e+6) (actual time=1.95..17060 rows=44.5e+6 loops=1)
                    -> Hash
                        -> Table scan on L  (cost=3155 rows=30016) (actual time=1.49..23.4 rows=31182 loops=1)
        -> Select #3 (subquery in condition; run only once)
            -> Aggregate: avg(historical_weather.TAVG)  (cost=7.72e+6 rows=1) (actual time=16015..16015 rows=1 loops=1)
                -> Filter: (historical_weather.H_MONTH in (6,7,8))  (cost=4.66e+6 rows=13.2e+6) (actual time=1.58..15731 rows=11.3e+6 loops=1)
                    -> Table scan on HISTORICAL_WEATHER  (cost=4.66e+6 rows=44.2e+6) (actual time=1.57..14739 rows=44.5e+6 loops=1)
```

## Adding Indexing on H_MONTH, RND_LAT, RND_LNG, TAVG

```
-- Add H_MONTH, RND_LAT, RND_LNG, TAVG Indexing
CREATE INDEX idx_hw_month_lat_lng ON HISTORICAL_WEATHER (H_MONTH, RND_LAT, RND_LNG, TAVG);
EXPLAIN ANALYZE SELECT L.CITY,L.STATE,AVG(S.TAVG) AS City_Avg_Summer_Temp
FROM LOCATIONS L
JOIN (SELECT RND_LAT,RND_LNG,TAVG FROM HISTORICAL_WEATHER WHERE H_MONTH IN (6,7,8)) S
  ON S.RND_LAT=L.RND_LAT AND S.RND_LNG=L.RND_LNG
GROUP BY L.CITY,L.STATE
HAVING AVG(S.TAVG) > (
  SELECT AVG(T.TAVG) FROM (SELECT TAVG FROM HISTORICAL_WEATHER WHERE H_MONTH IN (6,7,8)) T
)
ORDER BY City_Avg_Summer_Temp DESC;
```

sults 1 | Results 1 (2) | Results 1 (3) | Results 1 (4) | Statistics 1

Enter a SQL expression to filter results (use Ctrl+Space)

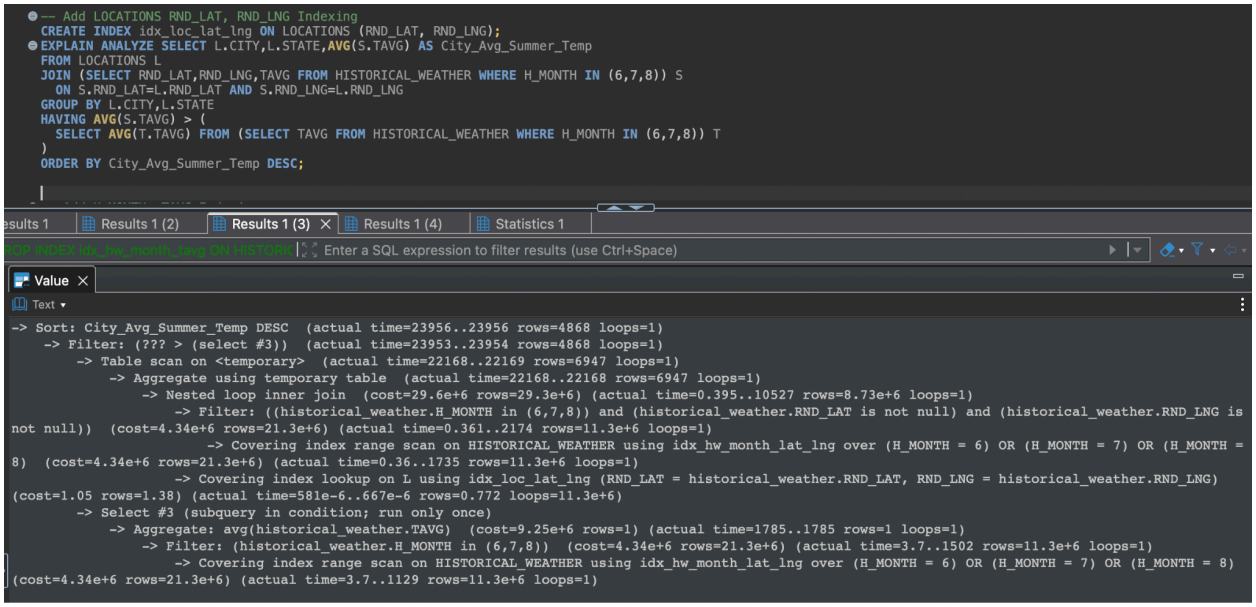
Value

Text

```
-> Sort: City_Avg_Summer_Temp DESC  (actual time=19722..19722 rows=4868 loops=1)
    -> Filter: (??? > (select #3))  (actual time=19718..19719 rows=4868 loops=1)
        -> Table scan on <temporary>  (actual time=17942..17943 rows=6947 loops=1)
            -> Aggregate using temporary table  (actual time=17942..17942 rows=6947 loops=1)
                -> Inner hash join (historical_weather.RND_LNG = L.RND_LNG), (historical_weather.RND_LAT = L.RND_LAT)  (cost=67e+9 rows=64e+6) (actual
time=33.7..6586 rows=8.73e+6 loops=1)
                    -> Filter: (historical_weather.H_MONTH in (6,7,8))  (cost=2.21e+6 rows=213112) (actual time=0.403..2098 rows=11.3e+6 loops=1)
                        -> Covering index range scan on HISTORICAL_WEATHER using idx_hw_month_lat_lng over (H_MONTH = 6) OR (H_MONTH = 7) OR (H_MONTH =
8)  (cost=2.21e+6 rows=21.3e+6) (actual time=0.402..1725 rows=11.3e+6 loops=1)
                    -> Hash
                        -> Table scan on L  (cost=3155 rows=30016) (actual time=1.31..18.5 rows=31182 loops=1)
        -> Select #3 (subquery in condition; run only once)
            -> Aggregate: avg(historical_weather.TAVG)  (cost=9.25e+6 rows=1) (actual time=1775..1775 rows=1 loops=1)
                -> Filter: (historical_weather.H_MONTH in (6,7,8))  (cost=4.34e+6 rows=21.3e+6) (actual time=0.367..1493 rows=11.3e+6 loops=1)
                    -> Covering index range scan on HISTORICAL_WEATHER using idx_hw_month_lat_lng over (H_MONTH = 6) OR (H_MONTH = 7) OR (H_MONTH = 8)
(cost=4.34e+6 rows=21.3e+6) (actual time=0.365..1121 rows=11.3e+6 loops=1)
```

Adding this indexing to our query reduced the subquery filter cost from 18380 to 2126. The join cost was reduced from 949e6 to 676e6. The filtering cost decreased because we're now able to perform an index range scan to directly locate rows for H_MONTH rather than scanning the entire HISTORICAL_WEATHER table. The total join cost decreased because by indexing RND_LAT and RND_LNG, we reduced the number of intermediate tuples joined with the LOCATIONS table.

Adding Indexing on LOCATIONS RND_LAT, RND_LNG



```
-- Add LOCATIONS RND_LAT, RND_LNG Indexing
CREATE INDEX idx_loc_lat_lng ON LOCATIONS (RND_LAT, RND_LNG);
EXPLAIN ANALYZE SELECT L.CITY,L.STATE,AVG(S.TAVG) AS City_Avg_Summer_Temp
FROM LOCATIONS L
JOIN (SELECT RND_LAT,RND_LNG,TAVG FROM HISTORICAL_WEATHER WHERE H_MONTH IN (6,7,8)) S
  ON S.RND_LAT=L.RND_LAT AND S.RND_LNG=L.RND_LNG
GROUP BY L.CITY,L.STATE
HAVING AVG(S.TAVG) > (
  SELECT AVG(T.TAVG) FROM (SELECT TAVG FROM HISTORICAL_WEATHER WHERE H_MONTH IN (6,7,8)) T
)
ORDER BY City_Avg_Summer_Temp DESC;
```

esults 1 | Results 1 (2) | Results 1 (3) | Results 1 (4) | Statistics 1

Enter a SQL expression to filter results (use Ctrl+Space)

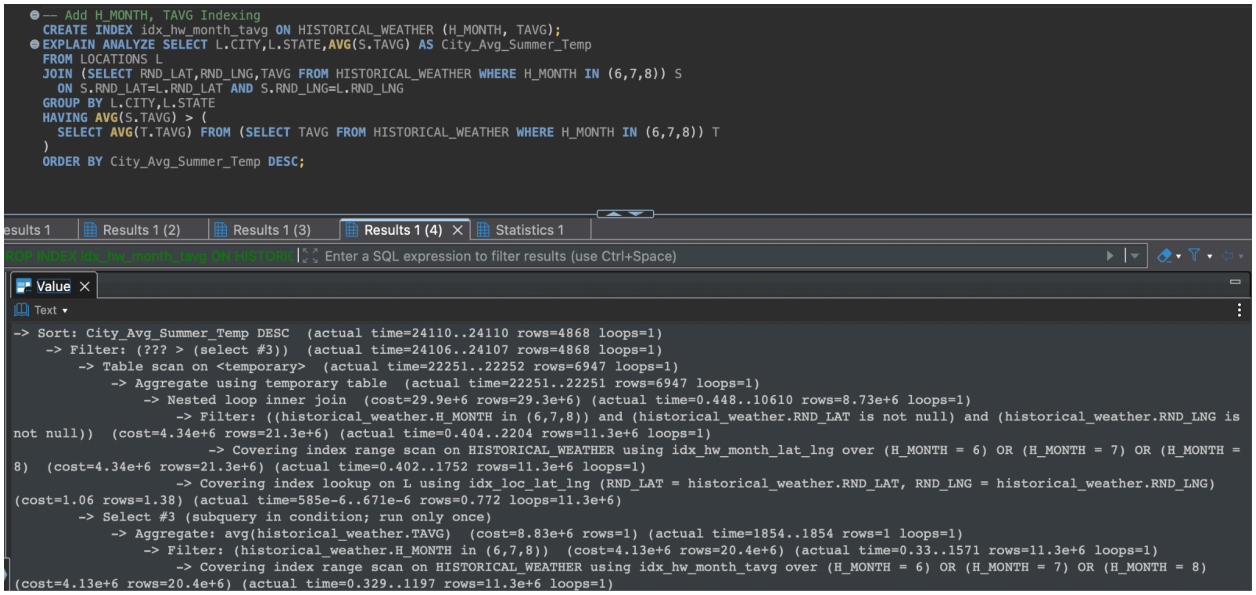
Value

Text

```
-> Sort: City_Avg_Summer_Temp DESC  (actual time=23956..23956 rows=4868 loops=1)
    -> Filter: (??? > (select #3))  (actual time=23953..23954 rows=4868 loops=1)
        -> Table scan on <temporary>  (actual time=22168..22169 rows=6947 loops=1)
            -> Aggregate using temporary table  (actual time=22168..22168 rows=6947 loops=1)
                -> Nested loop inner join  (cost=29.6e+6 rows=29.3e+6) (actual time=0.395..10527 rows=8.73e+6 loops=1)
                    -> Filter: ((historical_weather.H_MONTH in (6,7,8)) and (historical_weather.RND_LAT is not null) and (historical_weather.RND_LNG is
not null))  (cost=4.34e+6 rows=21.3e+6) (actual time=0.361..2174 rows=11.3e+6 loops=1)
                        -> Covering index range scan on HISTORICAL_WEATHER using idx_hw_month_lat_lng over (H_MONTH = 6) OR (H_MONTH = 7) OR (H_MONTH =
8)  (cost=4.34e+6 rows=21.3e+6) (actual time=0.36..1735 rows=11.3e+6 loops=1)
                    -> Covering index lookup on L using idx_loc_lat_lng (RND_LAT = historical_weather.RND_LAT, RND_LNG = historical_weather.RND_LNG)
(cost=1.05 rows=1.38) (actual time=581e-6..667e-6 rows=0.772 loops=11.3e+6)
        -> Select #3 (subquery in condition; run only once)
            -> Aggregate: avg(historical_weather.TAVG)  (cost=9.25e+6 rows=1) (actual time=1785..1785 rows=1 loops=1)
                -> Filter: (historical_weather.H_MONTH in (6,7,8))  (cost=4.34e+6 rows=21.3e+6) (actual time=3.7..1502 rows=11.3e+6 loops=1)
                    -> Covering index range scan on HISTORICAL_WEATHER using idx_hw_month_lat_lng over (H_MONTH = 6) OR (H_MONTH = 7) OR (H_MONTH = 8)
(cost=4.34e+6 rows=21.3e+6) (actual time=3.7..1129 rows=11.3e+6 loops=1)
```

The index significantly decreased the total join cost from 676e+6 to 29.6e+6. However, the filtering cost on H_MONTH increased from 2126 to 4340. The join cost sees a massive decrease because both sides of the join can now directly match using latitude and longitude indexes. The filtering cost increases slightly since we switched from a broad join with less overhead to a granular index lookup.

Adding Indexing on H_MONTH, TAVG




The index caused a join cost increase from 26.6e+6 to 30.4+6. However, we do see a minor filtering cost decrease from 4340 to 4130. We see the minor filtering cost decrease because we're now able to use the index to retrieve summer temperature values directly when evaluating the subquery's average. The total join cost significantly increases because RND_LAT and RND_LNG aren't included in the index, so the same join operations as before have to be performed, with the extra cost of managing another index for TAVG, causing the slight increase in cost.

The final indexing design we will use will be the second query. Although we double the filter cost, the massive performance gains we get in the total join cost significantly decreases the overall cost.

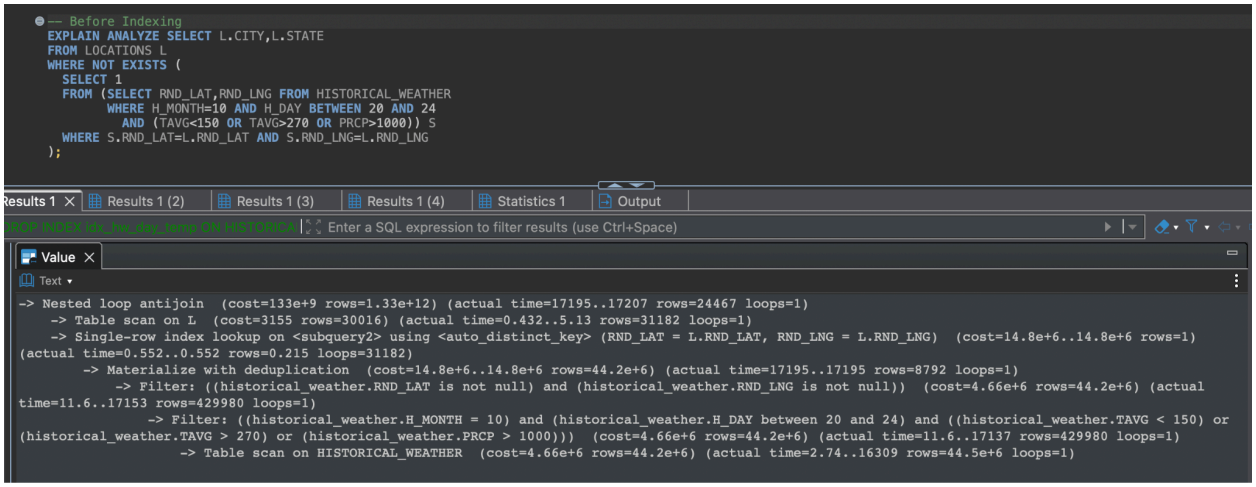
**Query 3: Finds locations that have NEVER had a 'bad' day between Oct 20th and Oct 24th**

```
SELECT L.CITY,L.STATE
FROM LOCATIONS L
WHERE NOT EXISTS (
  SELECT 1
  FROM (SELECT RND_LAT,RND_LNG FROM HISTORICAL_WEATHER
        WHERE H_MONTH=10 AND H_DAY BETWEEN 20 AND 24
          AND (TAVG<150 OR TAVG>270 OR PRCP>1000)) S
  WHERE S.RND_LAT=L.RND_LAT AND S.RND_LNG=L.RND_LNG
);
```

Top 15 Results

| | CITY | STATE |
|---|---|---|
| 1 | Boqueron | PR |
| 2 | El Combate | PR |
| 3 | Pole Ojea | PR |
| 4 | Betances | PR |
| 5 | Lajas | PR |
| 6 | Maguayo | PR |
| 7 | Palmarejo | PR |
| 8 | La Parguera | PR |
| 9 | El Tumbao | PR |
| 10 | Fuig | PR |
| 11 | Guanica | PR |
| 12 | Liborio Negron To | PR |
| 13 | Lluveras | PR |
| 14 | Maria Antonia | PR |
| 15 | Palomas | PR |

## Before Indexing



```
-- Before Indexing
EXPLAIN ANALYZE SELECT L.CITY,L.STATE
FROM LOCATIONS L
WHERE NOT EXISTS (
  SELECT 1
  FROM (SELECT RND_LAT,RND_LNG FROM HISTORICAL_WEATHER
        WHERE H_MONTH=10 AND H_DAY BETWEEN 20 AND 24
          AND (TAVG<150 OR TAVG>270 OR PRCP>1000)) S
  WHERE S.RND_LAT=L.RND_LAT AND S.RND_LNG=L.RND_LNG
);
```

Results 1 | Results 1 (2) | Results 1 (3) | Results 1 (4) | Statistics 1 | Output

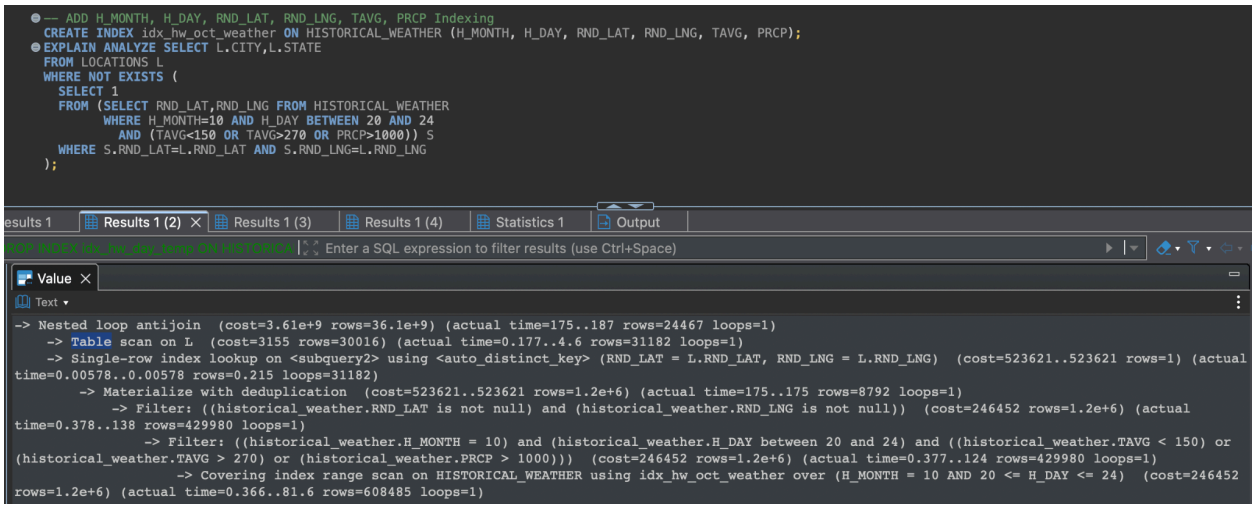
Enter a SQL expression to filter results (use Ctrl+Space)

Value

Text

```
-> Nested loop antijoin  (cost=133e+9 rows=1.33e+12) (actual time=17195..17207 rows=24467 loops=1)
    -> Table scan on L  (cost=3155 rows=30016) (actual time=0.432..5.13 rows=31182 loops=1)
    -> Single-row index lookup on <subquery2> using <auto_distinct_key> (RND_LAT = L.RND_LAT, RND_LNG = L.RND_LNG)  (cost=14.8e+6..14.8e+6 rows=1)
(actual time=0.552..0.552 rows=0.215 loops=31182)
        -> Materialize with deduplication  (cost=14.8e+6..14.8e+6 rows=44.2e+6) (actual time=17195..17195 rows=8792 loops=1)
            -> Filter: ((historical_weather.RND_LAT is not null) and (historical_weather.RND_LNG is not null))  (cost=4.66e+6 rows=44.2e+6) (actual
time=11.6..17153 rows=429980 loops=1)
                -> Filter: ((historical_weather.H_MONTH = 10) and (historical_weather.H_DAY between 20 and 24) and ((historical_weather.TAVG < 150) or
(historical_weather.TAVG > 270) or (historical_weather.PRCP > 1000)))  (cost=4.66e+6 rows=44.2e+6) (actual time=11.6..17137 rows=429980 loops=1)
                    -> Table scan on HISTORICAL_WEATHER  (cost=4.66e+6 rows=44.2e+6) (actual time=2.74..16309 rows=44.5e+6 loops=1)
```

## Adding Indexing on H_MONTH, H_DAY, RND_LAT, RND_LNG, TAVG, PRCP



```
-- ADD H_MONTH, H_DAY, RND_LAT, RND_LNG, TAVG, PRCP Indexing
CREATE INDEX idx_hw_oct_weather ON HISTORICAL_WEATHER (H_MONTH, H_DAY, RND_LAT, RND_LNG, TAVG, PRCP);
EXPLAIN ANALYZE SELECT L.CITY,L.STATE
FROM LOCATIONS L
WHERE NOT EXISTS (
  SELECT 1
  FROM (SELECT RND_LAT,RND_LNG FROM HISTORICAL_WEATHER
        WHERE H_MONTH=10 AND H_DAY BETWEEN 20 AND 24
          AND (TAVG<150 OR TAVG>270 OR PRCP>1000)) S
  WHERE S.RND_LAT=L.RND_LAT AND S.RND_LNG=L.RND_LNG
);
```

esults 1 | Results 1 (2) | Results 1 (3) | Results 1 (4) | Statistics 1 | Output

Enter a SQL expression to filter results (use Ctrl+Space)

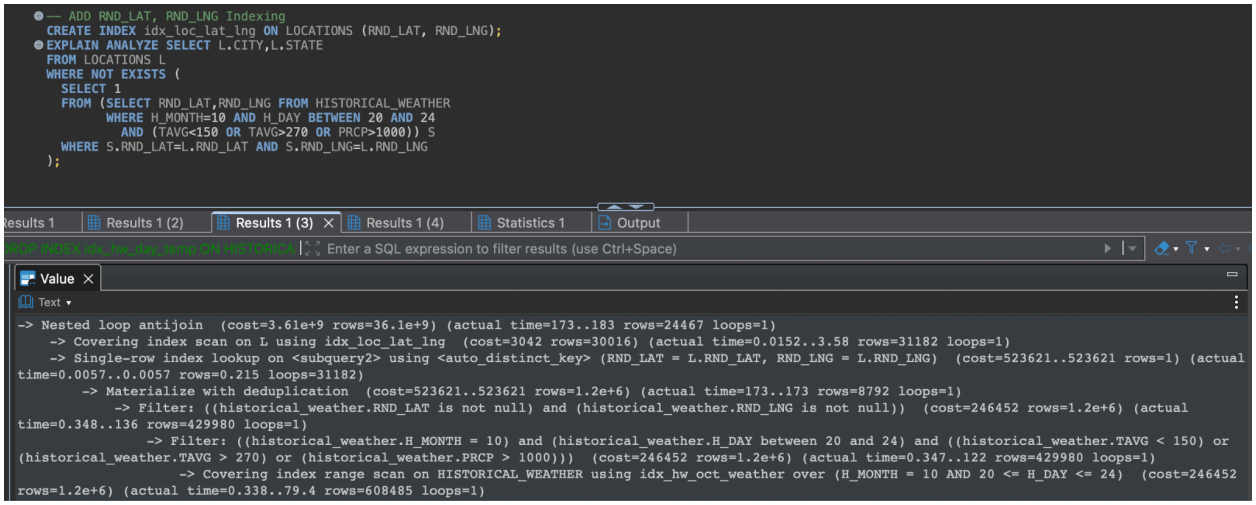
Value

Text

```
-> Nested loop antijoin  (cost=3.61e+9 rows=36.1e+9) (actual time=175..187 rows=24467 loops=1)
    -> Table scan on L  (cost=3155 rows=30016) (actual time=0.177..4.6 rows=31182 loops=1)
    -> Single-row index lookup on <subquery2> using <auto_distinct_key> (RND_LAT = L.RND_LAT, RND_LNG = L.RND_LNG)  (cost=523621..523621 rows=1) (actual
time=0.00578..0.00578 rows=0.215 loops=31182)
        -> Materialize with deduplication  (cost=523621..523621 rows=1.2e+6) (actual time=175..175 rows=8792 loops=1)
            -> Filter: ((historical_weather.RND_LAT is not null) and (historical_weather.RND_LNG is not null))  (cost=246452 rows=1.2e+6) (actual
time=0.378..138 rows=429980 loops=1)
                -> Filter: ((historical_weather.H_MONTH = 10) and (historical_weather.H_DAY between 20 and 24) and ((historical_weather.TAVG < 150) or
(historical_weather.TAVG > 270) or (historical_weather.PRCP > 1000)))  (cost=246452 rows=1.2e+6) (actual time=0.377..124 rows=429980 loops=1)
                    -> Covering index range scan on HISTORICAL_WEATHER using idx_hw_oct_weather over (H_MONTH = 10 AND 20 <= H_DAY <= 24)  (cost=246452
rows=1.2e+6) (actual time=0.366..81.6 rows=608485 loops=1)
```

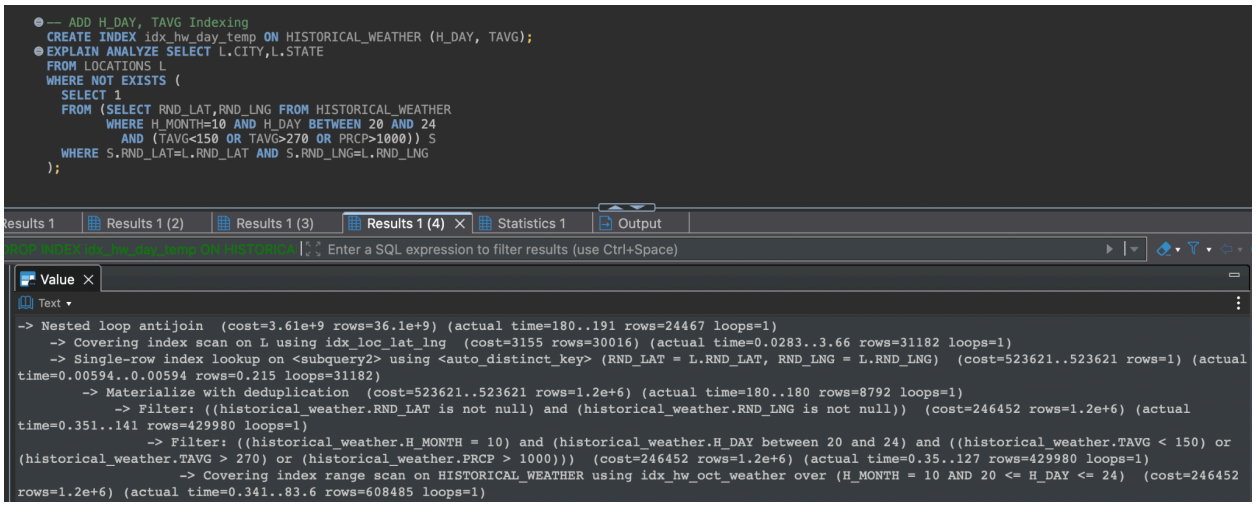
We see a massive decrease in cost from 133e9 to 3.61e9 in the nested loop anti join cost because we're now able to use the new index to locate "bad" day rows, avoiding the need to evaluate irrelevant rows. There is also a decrease in the weather conditions filter cost from 4.66e6 to 2.46e5 because we're not scanning the entire HISTORICAL_WEATHER table anymore. There's also a decrease in single-row index lookup from 14.8e6 to 5.23e5 because we're able to do direct coordinate lookups instead of a full table scan.

## Adding Indexing on RND_LAT, RND_LNG




The nested loop anti join had a very small decrease in cost from 3.61e9 to 3.6e9 because of the reduction in join comparison between LOCATIONS and the indexed subset of HISTORICAL_WEATHER. The filter on weather conditions and date range and the covering index scan on locations remain basically unchanged since the new location index isn't relevant for these operations.

## Adding Indexing on H_DAY, TAVG




All costs remained basically unchanged. The index on H_DAY and TAVG lacks critical filtering and join columns.

The final indexing design we will use will be the first query because it's the only option that results in any tangible performance gains. The other two queries had insignificant impact.

**Query 4: Locations with Above-Average Precipitation**

```
SELECT L.CITY, L.STATE, AVG(H.PRCP) AS LOCATION_AVG_PRECIP
FROM LOCATIONS L
JOIN HISTORICAL_WEATHER H
  ON L.RND_LAT = H.RND_LAT AND L.RND_LNG = H.RND_LNG
GROUP BY L.CITY, L.STATE
HAVING AVG(H.PRCP) > (
    SELECT AVG(PRCP)
    FROM HISTORICAL_WEATHER
)
ORDER BY LOCATION_AVG_PRECIP DESC;
```

Top 15 Results

| | CITY | STATE | LOCATION_AVG_PRECIP |
|---|---|---|---|
| 1 | Frazee | MN | 622 |
| 2 | Eden Roc | HI | 126.6276 |
| 3 | Fern Acres | HI | 126.6276 |
| 4 | Fern Forest | HI | 126.6276 |
| 5 | Pelican | AK | 105.5653 |
| 6 | Ketchikan | AK | 105.4434 |
| 7 | Barnum Island | NY | 98.25 |
| 8 | Lido Beach | NY | 98.25 |
| 9 | Oceanside | NY | 98.25 |
| 10 | Point Lookout | NY | 98.25 |
| 11 | Grays River | WA | 91.7947 |
| 12 | Hilo | HI | 84.687 |
| 13 | Wainaku | HI | 84.687 |
| 14 | Snoqualmie Pass | WA | 83.7363 |
| 15 | Forks | WA | 82.0057 |

## Before Indexing

```
-- Before Indexing
EXPLAIN ANALYZE SELECT L.CITY, L.STATE, AVG(H.PRCP) AS LOCATION_AVG_PRECIP
FROM LOCATIONS L
JOIN HISTORICAL_WEATHER H
  ON L.RND_LAT = H.RND_LAT AND L.RND_LNG = H.RND_LNG
GROUP BY L.CITY, L.STATE
HAVING AVG(H.PRCP) > (
    SELECT AVG(PRCP)
    FROM HISTORICAL_WEATHER
)
ORDER BY LOCATION_AVG_PRECIP DESC;
```

Results 1 (2) | Results 1 (3) | Results 1 (4) | Statistics 1

Enter a SQL expression to filter results (use Ctrl+Space)

Value

Text

```
-> Sort: LOCATION_AVG_PRECIP DESC  (actual time=113548..113548 rows=4503 loops=1)
    -> Filter: (??? > (select #2))  (actual time=113544..113545 rows=4503 loops=1)
        -> Table scan on <temporary>  (actual time=96098..96099 rows=7020 loops=1)
            -> Aggregate using temporary table  (actual time=96098..96098 rows=7020 loops=1)
                -> Inner hash join (H.RND_LNG = L.RND_LNG), (H.RND_LAT = L.RND_LAT)  (cost=133e+9 rows=13.3e+9) (actual time=131..35908 rows=34.4e+6
loops=1)
                    -> Table scan on H  (cost=1343 rows=44.2e+6) (actual time=1.74..19132 rows=44.5e+6 loops=1)
                    -> Hash
                        -> Table scan on L  (cost=3155 rows=30016) (actual time=1.71..23.9 rows=31182 loops=1)
        -> Select #2 (subquery in condition; run only once)
            -> Aggregate: avg(historical_weather.PRCP)  (cost=14.8e+6 rows=1) (actual time=17445..17445 rows=1 loops=1)
                -> Table scan on HISTORICAL_WEATHER  (cost=4.66e+6 rows=44.2e+6) (actual time=3.38..16443 rows=44.5e+6 loops=1)
```

## Adding Indexing on RND_LAT, RND_LNG, PRCP

```
-- Add RND_LAT, RND_LNG, PRCP Indexing
CREATE INDEX idx_hw_lat_lng_prcp ON HISTORICAL_WEATHER (RND_LAT, RND_LNG, PRCP);
EXPLAIN ANALYZE SELECT L.CITY, L.STATE, AVG(H.PRCP) AS LOCATION_AVG_PRECIP
FROM LOCATIONS L
JOIN HISTORICAL_WEATHER H
  ON L.RND_LAT = H.RND_LAT AND L.RND_LNG = H.RND_LNG
GROUP BY L.CITY, L.STATE
HAVING AVG(H.PRCP) > (
    SELECT AVG(PRCP)
    FROM HISTORICAL_WEATHER
)
ORDER BY LOCATION_AVG_PRECIP DESC;
```

Results 1 (2) | Results 1 (3) | Results 1 (4) | Statistics 1

Enter a SQL expression to filter results (use Ctrl+Space)

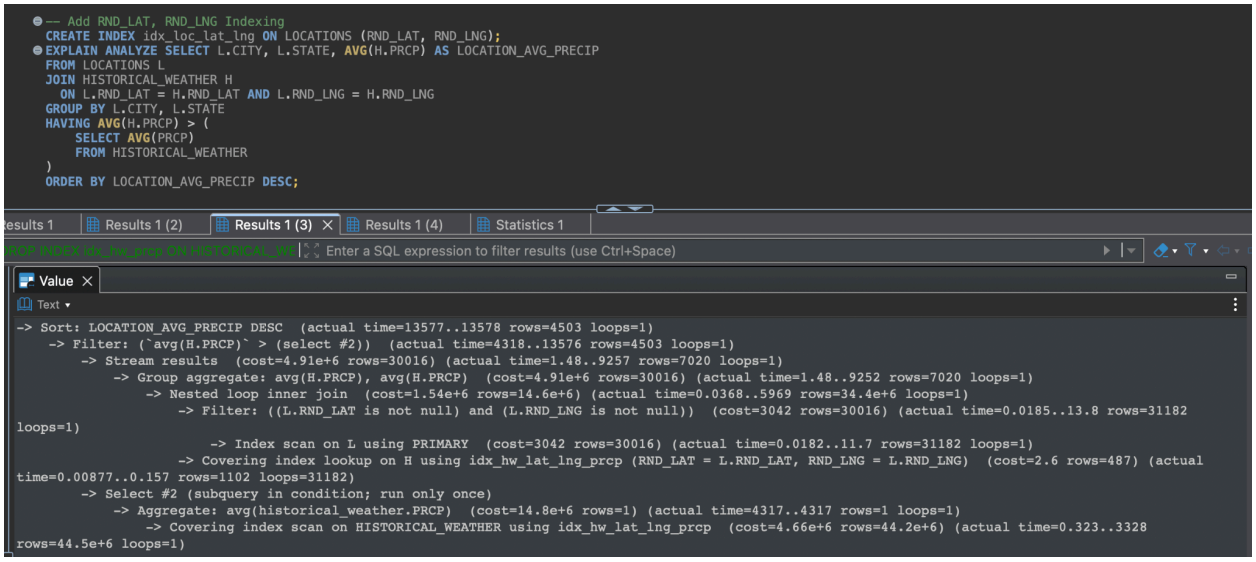
Value

Text

```
-> Sort: LOCATION_AVG_PRECIP DESC  (actual time=13536..13537 rows=4503 loops=1)
    -> Filter: (`avg(H.PRCP)` > (select #2))  (actual time=4318..13535 rows=4503 loops=1)
        -> Stream results  (cost=4.91e+6 rows=30016) (actual time=2.25..9218 rows=7020 loops=1)
            -> Group aggregate: avg(H.PRCP), avg(H.PRCP)  (cost=4.91e+6 rows=30016) (actual time=2.25..9214 rows=7020 loops=1)
                -> Nested loop inner join  (cost=1.54e+6 rows=14.6e+6) (actual time=0.716..5932 rows=34.4e+6 loops=1)
                    -> Filter: ((L.RND_LAT is not null) and (L.RND_LNG is not null))  (cost=3155 rows=30016) (actual time=0.687..48.3 rows=31182
loops=1)
                        -> Index scan on L using PRIMARY  (cost=3155 rows=30016) (actual time=0.686..46.4 rows=31182 loops=1)
                    -> Covering index lookup on H using idx_hw_lat_lng_prcp (RND_LAT = L.RND_LAT, RND_LNG = L.RND_LNG)  (cost=2.6 rows=487) (actual
time=0.0085..0.155 rows=1102 loops=31182)
        -> Select #2 (subquery in condition; run only once)
            -> Aggregate: avg(historical_weather.PRCP)  (cost=14.8e+6 rows=1) (actual time=4316..4316 rows=1 loops=1)
                -> Covering index scan on HISTORICAL_WEATHER using idx_hw_lat_lng_prcp  (cost=4.66e+6 rows=44.2e+6) (actual time=0.322..3321
rows=44.5e+6 loops=1)
```

Adding this index sees the cost of the query reduce significantly going from 133e+9 to 1.54e+6. This is largely due to the inclusion of an index on the rounded latitude and longitude which makes the Inner Hash Join become a Nester Loop join. As a result the cost of the join command is significantly less, and the overall time it takes to run the query is reduced.

## Adding Indexing on RND_LAT, RND_LNG



```
-- Add RND_LAT, RND_LNG Indexing
CREATE INDEX idx_loc_lat_lng ON LOCATIONS (RND_LAT, RND_LNG);
EXPLAIN ANALYZE SELECT L.CITY, L.STATE, AVG(H.PRCP) AS LOCATION_AVG_PRECIP
  FROM LOCATIONS L
  JOIN HISTORICAL_WEATHER H
    ON L.RND_LAT = H.RND_LAT AND L.RND_LNG = H.RND_LNG
  GROUP BY L.CITY, L.STATE
  HAVING AVG(H.PRCP) > (
      SELECT AVG(PRCP)
      FROM HISTORICAL_WEATHER
  )
  ORDER BY LOCATION_AVG_PRECIP DESC;
```

Results 1 | Results 1 (2) | Results 1 (3) | Results 1 (4) | Statistics 1

Enter a SQL expression to filter results (use Ctrl+Space)

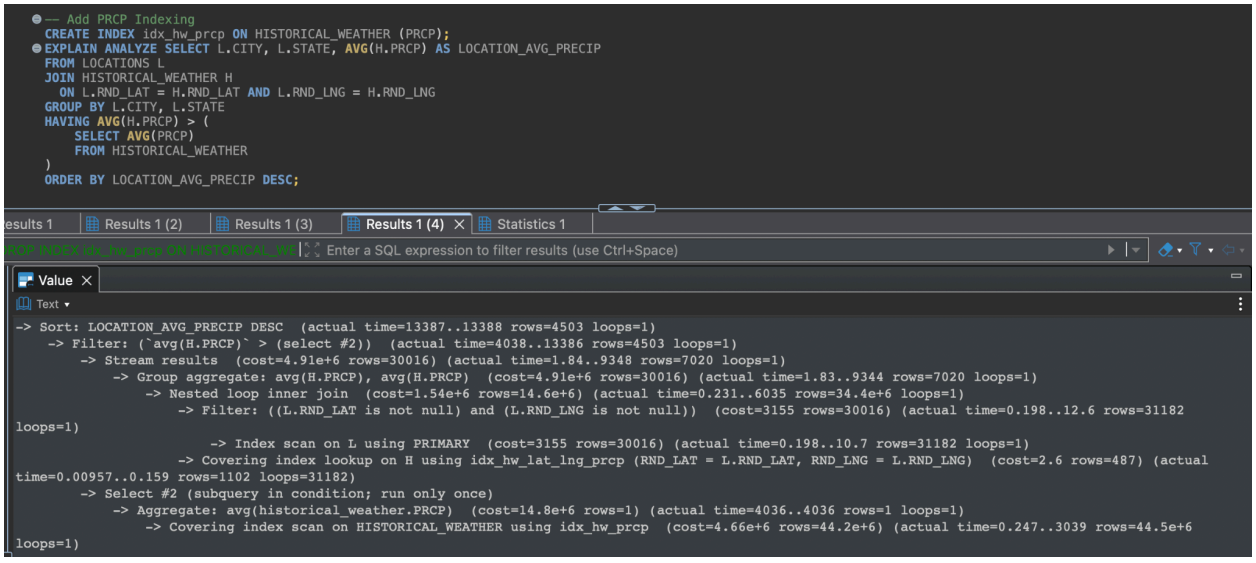
Value

Text

```
-> Sort: LOCATION_AVG_PRECIP DESC  (actual time=13577..13578 rows=4503 loops=1)
    -> Filter: (`avg(H.PRCP)` > (select #2))  (actual time=4318..13576 rows=4503 loops=1)
        -> Stream results  (cost=4.91e+6 rows=30016) (actual time=1.48..9257 rows=7020 loops=1)
            -> Group aggregate: avg(H.PRCP), avg(H.PRCP)  (cost=4.91e+6 rows=30016) (actual time=1.48..9252 rows=7020 loops=1)
                -> Nested loop inner join  (cost=1.54e+6 rows=14.6e+6) (actual time=0.0368..5969 rows=34.4e+6 loops=1)
                    -> Filter: ((L.RND_LAT is not null) and (L.RND_LNG is not null))  (cost=3042 rows=30016) (actual time=0.0185..13.8 rows=31182
loops=1)
                        -> Index scan on L using PRIMARY  (cost=3042 rows=30016) (actual time=0.0182..11.7 rows=31182 loops=1)
                    -> Covering index lookup on H using idx_hw_lat_lng_prcp (RND_LAT = L.RND_LAT, RND_LNG = L.RND_LNG)  (cost=2.6 rows=487) (actual
time=0.00877..0.157 rows=1102 loops=31182)
        -> Select #2 (subquery in condition; run only once)
            -> Aggregate: avg(historical_weather.PRCP)  (cost=14.8e+6 rows=1) (actual time=4317..4317 rows=1 loops=1)
                -> Covering index scan on HISTORICAL_WEATHER using idx_hw_lat_lng_prcp  (cost=4.66e+6 rows=44.2e+6) (actual time=0.323..3328
rows=44.5e+6 loops=1)
```

This index was created to isolate RND_LAT and RND_LNG and see if it would make a difference. What we found, however, is that we had the same performance as the previous indexing step. Our rationale was to isolate these two attributes in order to see if it would make any benefit over creating a composite index of all 3, however from the analysis we can see there is no meaningful difference.

## Adding Indexing on PRCP



```
-- Add PRCP Indexing
CREATE INDEX idx_hw_prcp ON HISTORICAL_WEATHER (PRCP);
EXPLAIN ANALYZE SELECT L.CITY, L.STATE, AVG(H.PRCP) AS LOCATION_AVG_PRECIP
  FROM LOCATIONS L
  JOIN HISTORICAL_WEATHER H
    ON L.RND_LAT = H.RND_LAT AND L.RND_LNG = H.RND_LNG
  GROUP BY L.CITY, L.STATE
  HAVING AVG(H.PRCP) > (
      SELECT AVG(PRCP)
      FROM HISTORICAL_WEATHER
  )
  ORDER BY LOCATION_AVG_PRECIP DESC;
```

Results 1 | Results 1 (2) | Results 1 (3) | Results 1 (4) | Statistics 1

Enter a SQL expression to filter results (use Ctrl+Space)

Value

Text

```
-> Sort: LOCATION_AVG_PRECIP DESC  (actual time=13387..13388 rows=4503 loops=1)
    -> Filter: (`avg(H.PRCP)` > (select #2))  (actual time=4038..13386 rows=4503 loops=1)
        -> Stream results  (cost=4.91e+6 rows=30016) (actual time=1.84..9348 rows=7020 loops=1)
            -> Group aggregate: avg(H.PRCP), avg(H.PRCP)  (cost=4.91e+6 rows=30016) (actual time=1.83..9344 rows=7020 loops=1)
                -> Nested loop inner join  (cost=1.54e+6 rows=14.6e+6) (actual time=0.231..6035 rows=34.4e+6 loops=1)
                    -> Filter: ((L.RND_LAT is not null) and (L.RND_LNG is not null))  (cost=3155 rows=30016) (actual time=0.198..12.6 rows=31182
loops=1)
                        -> Index scan on L using PRIMARY  (cost=3155 rows=30016) (actual time=0.198..10.7 rows=31182 loops=1)
                    -> Covering index lookup on H using idx_hw_lat_lng_prcp (RND_LAT = L.RND_LAT, RND_LNG = L.RND_LNG)  (cost=2.6 rows=487) (actual
time=0.00957..0.159 rows=1102 loops=31182)
        -> Select #2 (subquery in condition; run only once)
            -> Aggregate: avg(historical_weather.PRCP)  (cost=14.8e+6 rows=1) (actual time=4036..4036 rows=1 loops=1)
                -> Covering index scan on HISTORICAL_WEATHER using idx_hw_prcp  (cost=4.66e+6 rows=44.2e+6) (actual time=0.247..3039 rows=44.5e+6
loops=1)
```

After adding this index, there was no meaningful change in performance. The nested loop inner join cost, the group aggregation cost, the AVG(PRCP) cost, and total cost all remained exactly the same. This indicates that adding the PRCP index isn't necessary, likely because the precipitation column is being used in aggregation rather than in filtering, so indexing on precipitation doesn't reduce the amount of row comparisons we're doing.

The option we’re going with for the final indexing design will be the first query. This is because the first query is the only one with any significant performance gains,as the other two queries had no meaningful impact.