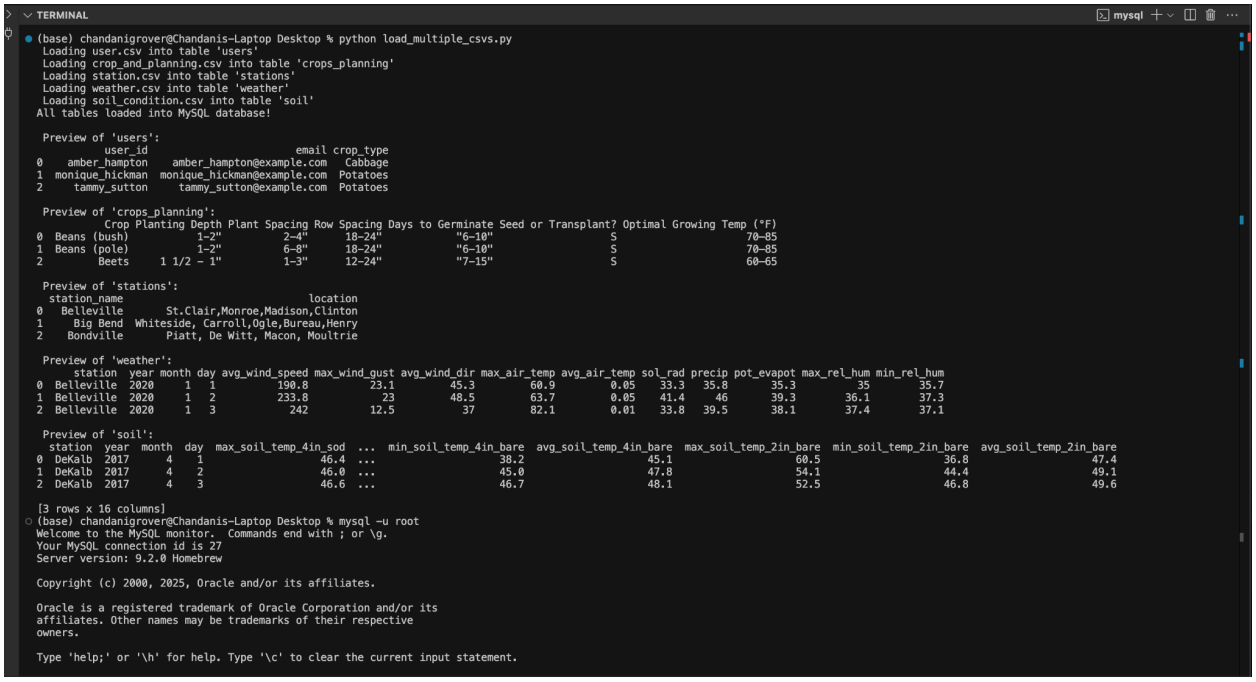
Stage 3 Database Design

Screenshot of Connection:



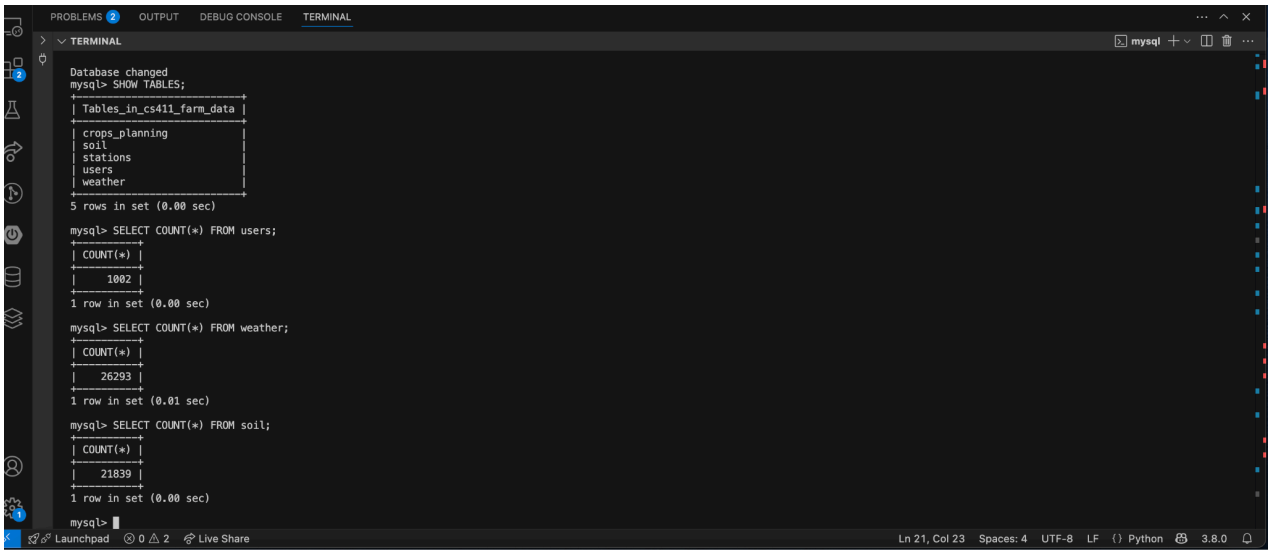
DDL Commands:

```
CREATE TABLE User (
        user_id VARCHAR (25) PRIMARY KEY,
        email_id VARCHAR(100) NOT NULL,
        crop_type VARCHAR(100) NOT NULL
);

CREATE TABLE StationName (
   station_name VARCHAR(100) PRIMARY KEY,
   location VARCHAR(100),
   user_id INT,
```

```
    FOREIGN KEY (user_id) REFERENCES User(user_id)
);

CREATE TABLE CropAndPlanning (
    crop_type VARCHAR(100) PRIMARY KEY,
    planting_depth FLOAT,
    days_to_germinate INT,
    plant_spacing FLOAT,
    seed_or_transplant VARCHAR(50)
    optimal_growing_temp FLOAT
);

CREATE TABLE Weather (
    station_name VARCHAR(100),
    day DATE,
    location VARCHAR(100),
    station VARCHAR(100),
    max_wind_speed FLOAT,
    avg_wind_speed FLOAT,
    avg_wind_direction FLOAT,
    max_air_temperature FLOAT,
    avg_air_temperature FLOAT,
    total_solar_radiation FLOAT,
    total_precipitation FLOAT,
    total_evaporation FLOAT,
    max_relative_humidity FLOAT,
    min_relative_humidity FLOAT,
    PRIMARY KEY (station_name, day)
);

CREATE TABLE SoilCondition (
    station_name VARCHAR(100),
    day DATE,
    location VARCHAR(100),
    max_2_inch_soil_temperature_under_SOD FLOAT,
    min_2_inch_soil_temperature_under_SOD FLOAT,
    max_4_inch_soil_temperature_under_SOD FLOAT,
    min_4_inch_soil_temperature_under_SOD FLOAT,
    max_8_inch_soil_temperature_under_SOD FLOAT,
    min_8_inch_soil_temperature_under_SOD FLOAT,
```

```
    max_2_inch_soil_temperature_under_bare_soil FLOAT,
    min_2_inch_soil_temperature_under_bare_soil FLOAT,
    max_4_inch_soil_temperature_under_bare_soil FLOAT,
    min_4_inch_soil_temperature_under_bare_soil FLOAT,
    max_8_inch_soil_temperature_under_bare_soil FLOAT,
    min_8_inch_soil_temperature_under_bare_soil FLOAT,
    PRIMARY KEY (station_name, day)
);

CREATE TABLE Result (
    result_id INT PRIMARY KEY,
    ideal_temperature FLOAT,
    predicted_temperature FLOAT,
    ideal_crop VARCHAR(100),
    planting_depth FLOAT,
    days_to_germinate INT
);
```

At least 1000 rows using COUNT query

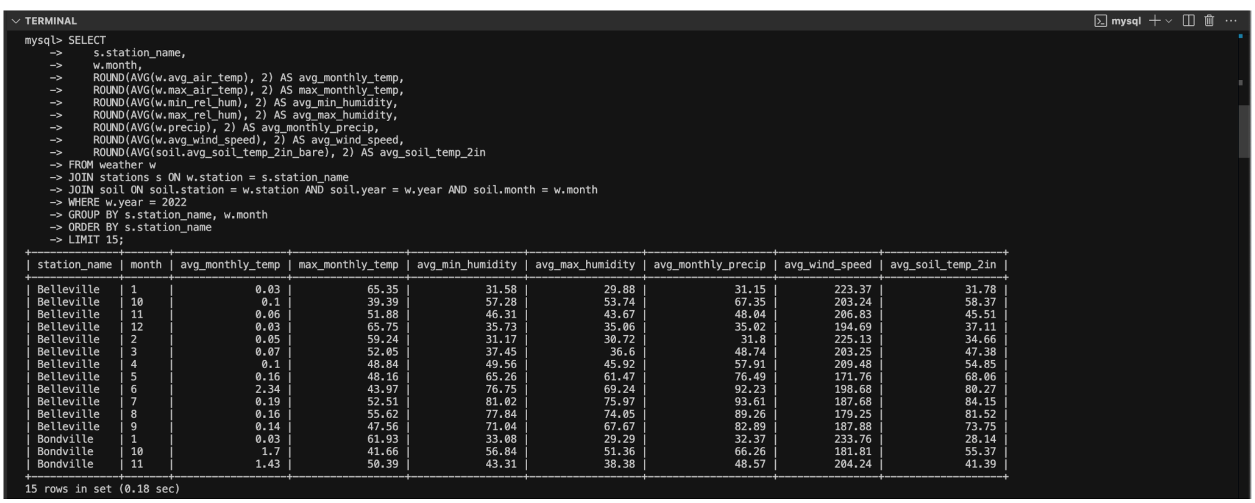


Advanced Queries

Query 1: Monthly Climate Trends by Station for 2022

This query presents a detailed summary of monthly climate trends of each station in 2022, including average and maximum air temperature, minimum and maximum humidity, precipitation, and wind speed. These metrics are useful for understanding station-specific weather profiles and planning crop schedules accordingly. The query has been limited to only show the first 15 rows.

```
SELECT
    s.station_name,
    w.month,
    ROUND(AVG(w.avg_air_temp), 2) AS avg_monthly_temp,
    ROUND(AVG(w.max_air_temp), 2) AS max_monthly_temp,
    ROUND(AVG(w.min_rel_hum), 2) AS avg_min_humidity,
    ROUND(AVG(w.max_rel_hum), 2) AS avg_max_humidity,
    ROUND(AVG(w.precip), 2) AS avg_monthly_precip,
    ROUND(AVG(w.avg_wind_speed), 2) AS avg_wind_speed,
    ROUND(AVG(soil.avg_soil_temp_2in_bare), 2) AS avg_soil_temp_2in
FROM weather w
JOIN stations s ON w.station = s.station_name
JOIN soil ON soil.station = w.station AND soil.year = w.year AND soil.month = w.month
WHERE w.year = 2022
GROUP BY s.station_name, w.month
ORDER BY s.station_name
LIMIT 15;
```



```
TERMINAL                                                        mysql
mysql> SELECT
    ->     s.station_name,
    ->     w.month,
    ->     ROUND(AVG(w.avg_air_temp), 2) AS avg_monthly_temp,
    ->     ROUND(AVG(w.max_air_temp), 2) AS max_monthly_temp,
    ->     ROUND(AVG(w.min_rel_hum), 2) AS avg_min_humidity,
    ->     ROUND(AVG(w.max_rel_hum), 2) AS avg_max_humidity,
    ->     ROUND(AVG(w.precip), 2) AS avg_monthly_precip,
    ->     ROUND(AVG(w.avg_wind_speed), 2) AS avg_wind_speed,
    ->     ROUND(AVG(soil.avg_soil_temp_2in_bare), 2) AS avg_soil_temp_2in
    -> FROM weather w
    -> JOIN stations s ON w.station = s.station_name
    -> JOIN soil ON soil.station = w.station AND soil.year = w.year AND soil.month = w.month
    -> WHERE w.year = 2022
    -> GROUP BY s.station_name, w.month
    -> ORDER BY s.station_name
    -> LIMIT 15;
```

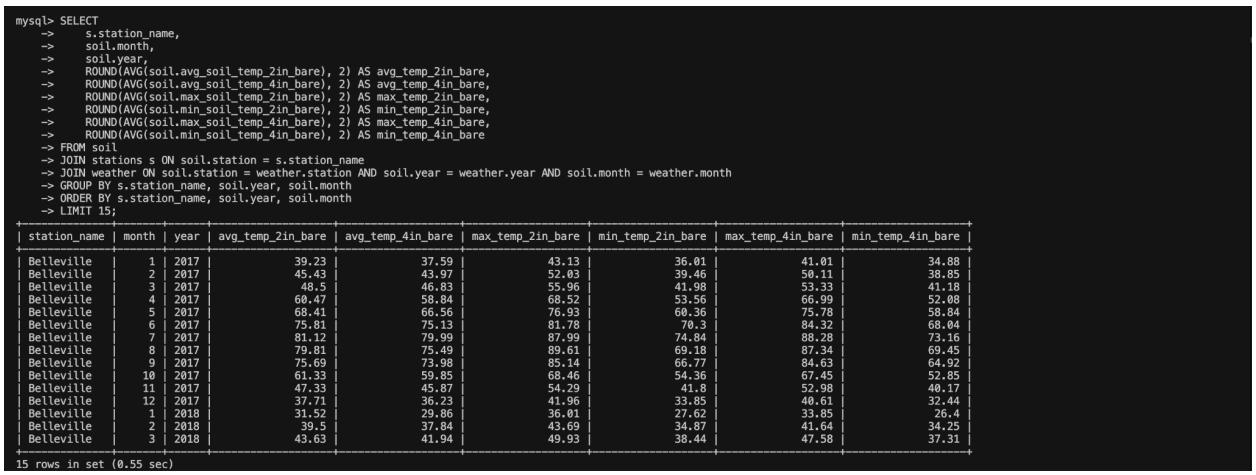
station_name	month	avg_monthly_temp	max_monthly_temp	avg_min_humidity	avg_max_humidity	avg_monthly_precip	avg_wind_speed	avg_soil_temp_2in
Belleville	1	0.03	65.35	31.58	29.88	31.15	223.37	31.78
Belleville	10	0.1	39.39	57.28	53.74	67.35	203.24	58.37
Belleville	11	0.06	51.88	46.31	43.67	48.04	206.83	45.51
Belleville	12	0.03	65.75	35.73	35.06	35.02	194.69	37.11
Belleville	2	0.05	59.24	31.17	30.72	31.8	225.13	34.66
Belleville	3	0.07	52.05	37.45	36.6	48.74	203.25	47.38
Belleville	4	0.1	48.84	49.56	45.92	57.91	209.48	54.85
Belleville	5	0.16	48.16	65.26	61.47	76.49	171.76	68.06
Belleville	6	2.34	43.97	76.75	69.24	92.23	198.68	80.27
Belleville	7	0.19	52.51	81.02	75.97	93.61	187.68	84.15
Belleville	8	0.16	55.62	77.84	74.05	89.26	179.25	81.52
Belleville	9	0.14	47.56	71.04	67.67	82.89	187.88	73.75
Bondville	1	0.03	61.93	33.08	29.29	32.37	233.76	28.14
Bondville	10	1.7	41.66	56.84	51.36	66.26	181.81	55.37
Bondville	11	1.43	50.39	43.31	38.38	48.57	204.24	41.39

15 rows in set (0.18 sec)

Query 2: Monthly Soil Temperature Trends by Station (All Years)

This query analyzes historical soil temperature trends across all stations and months by calculating the average, maximum, and minimum temperatures at both 2-inch and 4-inch bare soil depths.It helps identify seasonal patterns useful for crop planning and soil management.

```
SELECT
    s.station_name,
    soil.month,
    soil.year,
    ROUND(AVG(soil.avg_soil_temp_2in_bare), 2) AS avg_temp_2in_bare,
    ROUND(AVG(soil.avg_soil_temp_4in_bare), 2) AS avg_temp_4in_bare,
    ROUND(AVG(soil.max_soil_temp_2in_bare), 2) AS max_temp_2in_bare,
    ROUND(AVG(soil.min_soil_temp_2in_bare), 2) AS min_temp_2in_bare,
    ROUND(AVG(soil.max_soil_temp_4in_bare), 2) AS max_temp_4in_bare,
    ROUND(AVG(soil.min_soil_temp_4in_bare), 2) AS min_temp_4in_bare
FROM soil
JOIN stations s ON soil.station = s.station_name
JOIN weather ON soil.station = weather.station AND soil.year = weather.year AND soil.month =
weather.month
GROUP BY s.station_name, soil.year, soil.month
ORDER BY s.station_name, soil.year, soil.month
LIMIT 15;
```

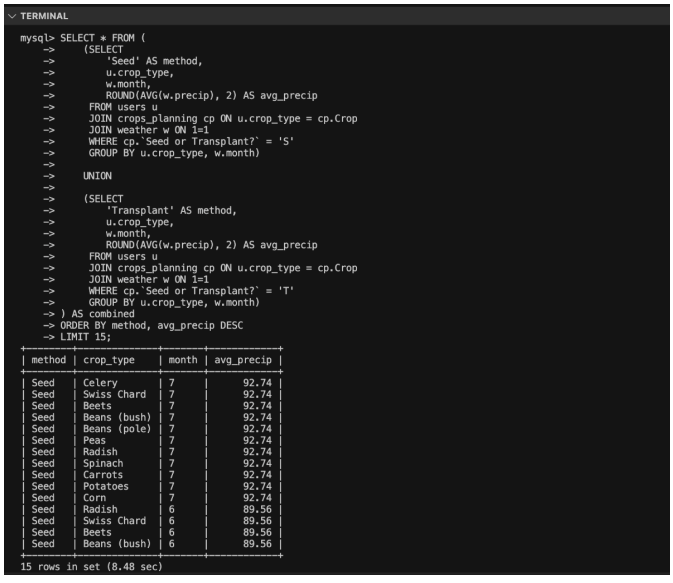


```
mysql> SELECT
    ->     s.station_name,
    ->     soil.month,
    ->     soil.year,
    ->     ROUND(AVG(soil.avg_soil_temp_2in_bare), 2) AS avg_temp_2in_bare,
    ->     ROUND(AVG(soil.avg_soil_temp_4in_bare), 2) AS avg_temp_4in_bare,
    ->     ROUND(AVG(soil.max_soil_temp_2in_bare), 2) AS max_temp_2in_bare,
    ->     ROUND(AVG(soil.min_soil_temp_2in_bare), 2) AS min_temp_2in_bare,
    ->     ROUND(AVG(soil.max_soil_temp_4in_bare), 2) AS max_temp_4in_bare,
    ->     ROUND(AVG(soil.min_soil_temp_4in_bare), 2) AS min_temp_4in_bare
    -> FROM soil
    -> JOIN stations s ON soil.station = s.station_name
    -> JOIN weather ON soil.station = weather.station AND soil.year = weather.year AND soil.month = weather.month
    -> GROUP BY s.station_name, soil.year, soil.month
    -> ORDER BY s.station_name, soil.year, soil.month
    -> LIMIT 15;
+--------------+-------+------+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+
| station_name | month | year | avg_temp_2in_bare | avg_temp_4in_bare | max_temp_2in_bare | min_temp_2in_bare | max_temp_4in_bare | min_temp_4in_bare |
+--------------+-------+------+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+
| Belleville   |     1 | 2017 |              39.23 |              37.59 |              43.13 |              36.01 |              41.01 |              34.88 |
| Belleville   |     2 | 2017 |              45.43 |              43.97 |              52.03 |              39.46 |              50.11 |              38.85 |
| Belleville   |     3 | 2017 |               48.5 |              46.83 |              55.96 |              41.98 |              53.33 |              41.18 |
| Belleville   |     4 | 2017 |              60.47 |              58.84 |              68.52 |              53.56 |              66.99 |              52.08 |
| Belleville   |     5 | 2017 |              68.41 |              66.56 |              76.93 |              60.36 |              75.78 |              58.84 |
| Belleville   |     6 | 2017 |              75.81 |              75.13 |              81.78 |               70.3 |              84.32 |              68.04 |
| Belleville   |     7 | 2017 |              81.12 |              79.99 |              87.99 |              74.84 |              88.28 |              73.16 |
| Belleville   |     8 | 2017 |              79.81 |              75.49 |              89.61 |              69.18 |              87.34 |              69.45 |
| Belleville   |     9 | 2017 |              75.69 |              73.98 |              85.14 |              66.77 |              84.63 |              64.92 |
| Belleville   |    10 | 2017 |              61.33 |              59.85 |              68.46 |              54.36 |              67.45 |              52.85 |
| Belleville   |    11 | 2017 |              47.33 |              45.87 |              54.29 |               41.8 |              52.98 |              40.17 |
| Belleville   |    12 | 2017 |              37.71 |              36.23 |              41.96 |              33.85 |              40.61 |              32.44 |
| Belleville   |     1 | 2018 |              31.52 |              29.86 |              36.01 |              27.62 |              33.85 |               26.4 |
| Belleville   |     2 | 2018 |               39.5 |              37.84 |              43.69 |              34.87 |              41.64 |              34.25 |
| Belleville   |     3 | 2018 |              43.63 |              41.94 |              49.93 |              38.44 |              47.58 |              37.31 |
+--------------+-------+------+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+
15 rows in set (0.55 sec)
```

Query 3: Best Planting Months for Crops by Method Based on Rainfall

This query compares seed-started and transplant-started crops to determine which months receive the highest average rainfall. It groups crops by planting method and month, helping identify the best planting periods based on historical precipitation trends.

```
SELECT * FROM (
   (SELECT
      'Seed' AS method,
      u.crop_type,
      w.month,
      ROUND(AVG(w.precip), 2) AS avg_precip
    FROM users u
    JOIN crops_planning cp ON u.crop_type = cp.Crop
    JOIN weather w ON 1=1
    WHERE cp.`Seed or Transplant?` = 'S'
    GROUP BY u.crop_type, w.month)
   UNION
   (SELECT
      'Transplant' AS method,
      u.crop_type,
      w.month,
      ROUND(AVG(w.precip), 2) AS avg_precip
    FROM users u
    JOIN crops_planning cp ON u.crop_type = cp.Crop
    JOIN weather w ON 1=1
    WHERE cp.`Seed or Transplant?` = 'T'
    GROUP BY u.crop_type, w.month)
) AS combined
ORDER BY method, avg_precip DESC
LIMIT 15;
```



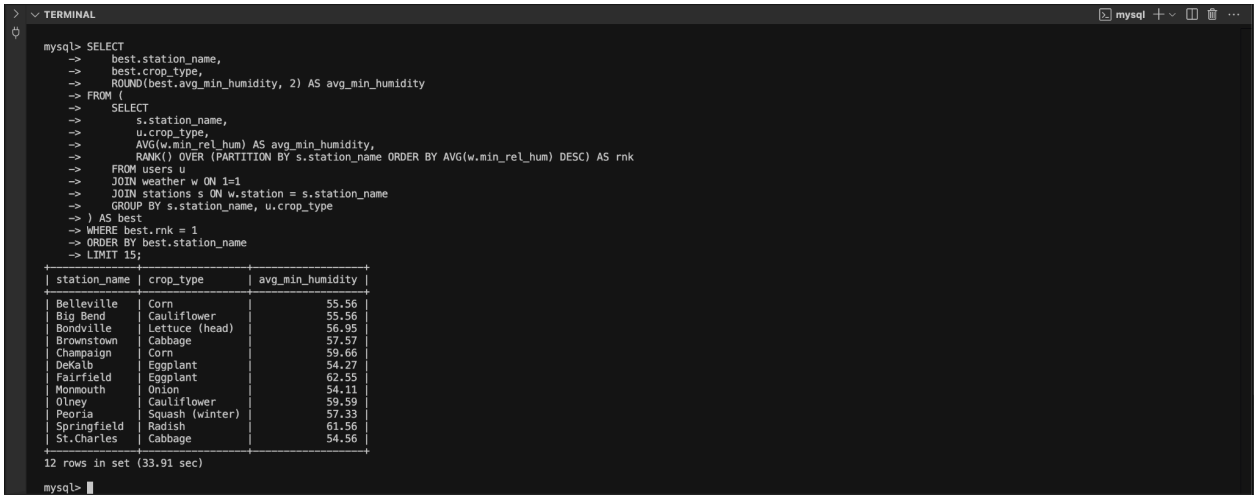
```
TERMINAL

mysql> SELECT * FROM (
    ->     (SELECT
    ->         'Seed' AS method,
    ->         u.crop_type,
    ->         w.month,
    ->         ROUND(AVG(w.precip), 2) AS avg_precip
    ->      FROM users u
    ->      JOIN crops_planning cp ON u.crop_type = cp.Crop
    ->      JOIN weather w ON 1=1
    ->      WHERE cp.`Seed or Transplant?` = 'S'
    ->      GROUP BY u.crop_type, w.month)
    ->
    ->     UNION
    ->
    ->     (SELECT
    ->         'Transplant' AS method,
    ->         u.crop_type,
    ->         w.month,
    ->         ROUND(AVG(w.precip), 2) AS avg_precip
    ->      FROM users u
    ->      JOIN crops_planning cp ON u.crop_type = cp.Crop
    ->      JOIN weather w ON 1=1
    ->      WHERE cp.`Seed or Transplant?` = 'T'
    ->      GROUP BY u.crop_type, w.month)
    -> ) AS combined
    -> ORDER BY method, avg_precip DESC
    -> LIMIT 15;
+--------+--------------+-------+-------------+
| method | crop_type    | month | avg_precip |
+--------+--------------+-------+-------------+
| Seed   | Celery       | 7     |       92.74 |
| Seed   | Swiss Chard  | 7     |       92.74 |
| Seed   | Beets        | 7     |       92.74 |
| Seed   | Beans (bush) | 7     |       92.74 |
| Seed   | Beans (pole) | 7     |       92.74 |
| Seed   | Peas         | 7     |       92.74 |
| Seed   | Radish       | 7     |       92.74 |
| Seed   | Spinach      | 7     |       92.74 |
| Seed   | Carrots      | 7     |       92.74 |
| Seed   | Potatoes     | 7     |       92.74 |
| Seed   | Corn         | 7     |       92.74 |
| Seed   | Radish       | 6     |       89.56 |
| Seed   | Swiss Chard  | 6     |       89.56 |
| Seed   | Beets        | 6     |       89.56 |
| Seed   | Beans (bush) | 6     |       89.56 |
+--------+--------------+-------+-------------+
15 rows in set (8.48 sec)
```

Query 4: Best Crop for Each Station Based on Humidity

This query identifies the best-performing crop for each station by comparing the average minimum relative humidity across all crops grown at that station. The crop with the highest average humidity tolerance is selected as the best match for that station's climate. This is useful for optimizing crop placement based on environmental suitability.

Even though the limit is set to 15, we only have 12 rows, as we are sampling over 12 stations in Illinois. Thus, we have one recommendation for each station.

```
SELECT
   best.station_name,
   best.crop_type,
   ROUND(best.avg_min_humidity, 2) AS avg_min_humidity
FROM (
   SELECT
      s.station_name,
      u.crop_type,
      AVG(w.min_rel_hum) AS avg_min_humidity,
      RANK() OVER (PARTITION BY s.station_name ORDER BY AVG(w.min_rel_hum) DESC) AS rnk
   FROM users u
   JOIN weather w ON 1=1
   JOIN stations s ON w.station = s.station_name
   GROUP BY s.station_name, u.crop_type
) AS best
WHERE best.rnk = 1
ORDER BY best.station_name
LIMIT 15;
```



Indexing Analysis:

Query 1: Monthly Climate Trends by Station for 2022

EXPLAIN ANALYZE:

```
mysql> EXPLAIN ANALYZE
    -> SELECT
    ->     s.station_name,
    ->     w.month,
    ->     ROUND(AVG(w.avg_air_temp), 2) AS avg_monthly_temp,
    ->     ROUND(AVG(w.max_air_temp), 2) AS max_monthly_temp,
    ->     ROUND(AVG(w.min_rel_hum), 2) AS avg_min_humidity,
    ->     ROUND(AVG(w.max_rel_hum), 2) AS avg_max_humidity,
    ->     ROUND(AVG(w.precip), 2) AS avg_monthly_precip,
    ->     ROUND(AVG(w.avg_wind_speed), 2) AS avg_wind_speed
    -> FROM weather w
    -> JOIN stations s ON w.station = s.station_name
    -> WHERE w.year = 2022
    -> GROUP BY s.station_name, w.month
    -> ORDER BY s.station_name;
```

Default Index:

```
| -> Sort: s.station_name, w.`month`  (actual time=168..168 rows=120 loops=1)
    -> Table scan on <temporary>  (actual time=168..168 rows=120 loops=1)
        -> Aggregate using temporary table  (actual time=168..168 rows=120 loops=1)
            -> Filter: (soil.station = s.station_name)  (cost=110936 rows=1064) (actual time=49.3..88.1 rows=110528 loops=1)
                -> Inner hash join (<hash>(soil.station)=<hash>(s.station_name)), (cast(soil.`year` as double) = cast(w.`year` as double)), (cast(soil.`month` as double) = cast(w.`month` as doub
le))  (cost=110936 rows=1064) (actual time=49.3..79.3 rows=110528 loops=1)
                    -> Table scan on soil  (cost=0.304 rows=21735) (actual time=0.0528..9.57 rows=21839 loops=1)
                    -> Hash
                        -> Filter: (w.station = s.station_name)  (cost=3016 rows=49) (actual time=1.51..29.8 rows=4380 loops=1)
                            -> Inner hash join (<hash>(w.station)=<hash>(s.station_name))  (cost=3016 rows=49) (actual time=1.51..27.9 rows=4380 loops=1)
                                -> Filter: (w.`year` = 2022)  (cost=134 rows=258) (actual time=1.35..26.6 rows=4380 loops=1)
                                    -> Table scan on w  (cost=134 rows=25773) (actual time=0.0399..24.2 rows=26293 loops=1)
                                -> Hash
                                    -> Table scan on s  (cost=2.15 rows=19) (actual time=0.0765..0.103 rows=19 loops=1)
 |
```

Weather_station Index:
Index used: CREATE INDEX weather_station ON weather(station(50));

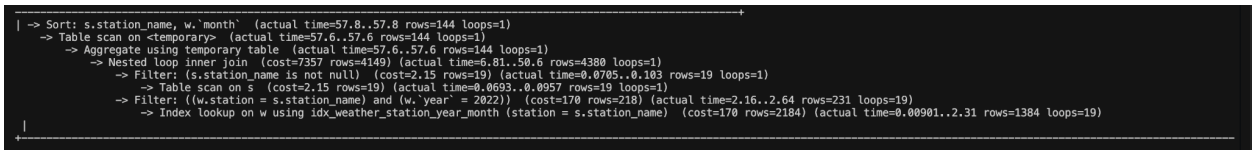
```
| -> Sort: s.station_name, w.`month`  (actual time=199..199 rows=120 loops=1)
    -> Table scan on <temporary>  (actual time=199..199 rows=120 loops=1)
        -> Aggregate using temporary table  (actual time=199..199 rows=120 loops=1)
            -> Filter: (soil.station = s.station_name)  (cost=8.88e+6 rows=88695) (actual time=76.1..123 rows=110528 loops=1)
                -> Inner hash join (<hash>(soil.station)=<hash>(s.station_name)), (cast(soil.`year` as double) = cast(w.`year` as double)), (cast(soil.`month` as double) = cast(w.`month` as doub
le))  (cost=8.88e+6 rows=88695) (actual time=76.1..115 rows=110528 loops=1)
                    -> Table scan on soil  (cost=0.307 rows=21735) (actual time=0.0272..7.74 rows=21839 loops=1)
                    -> Hash
                        -> Nested loop inner join  (cost=7289 rows=4081) (actual time=2.76..54.8 rows=4380 loops=1)
                            -> Filter: (s.station_name is not null)  (cost=2.15 rows=19) (actual time=0.0483..0.076 rows=19 loops=1)
                                -> Table scan on s  (cost=2.15 rows=19) (actual time=0.0474..0.0703 rows=19 loops=1)
                            -> Filter: ((w.station = s.station_name) and (w.`year` = 2022))  (cost=170 rows=215) (actual time=2.18..2.86 rows=231 loops=19)
                                -> Index lookup on w using idx_weather_station (station = s.station_name)  (cost=170 rows=2148) (actual time=0.00855..2.5 rows=1384 loops=19)
 |
```

Soil_month Index:
Index Used: CREATE INDEX idx_soil_month ON soil(month);

```
| -> Sort: s.station_name, w.`month`  (actual time=4965..4965 rows=120 loops=1)
    -> Table scan on <temporary>  (actual time=4964..4965 rows=120 loops=1)
        -> Aggregate using temporary table  (actual time=4964..4964 rows=120 loops=1)
            -> Nested loop inner join  (cost=1.07e+6 rows=887) (actual time=9.01..4849 rows=110528 loops=1)
                -> Filter: (w.station = s.station_name)  (cost=3016 rows=49) (actual time=1.59..23 rows=4380 loops=1)
                    -> Inner hash join (<hash>(w.station)=<hash>(s.station_name))  (cost=3016 rows=49) (actual time=1.59..22.6 rows=4380 loops=1)
                        -> Filter: ((w.`year` = 2022) and (w.`month` is not null))  (cost=134 rows=258) (actual time=1.45..21.9 rows=4380 loops=1)
                            -> Table scan on w  (cost=134 rows=25773) (actual time=0.0595..20.6 rows=26293 loops=1)
                        -> Hash
                            -> Table scan on s  (cost=2.15 rows=19) (actual time=0.083..0.107 rows=19 loops=1)
                -> Filter: ((soil.station = s.station_name) and (cast(soil.`year` as double) = cast(w.`year` as double)))  (cost=217 rows=18.1) (actual time=1.02..1.1 rows=25.2 loops=4380)
                    -> Index lookup on soil using idx_soil_month (month = w.`month`), with index condition: (cast(soil.`month` as double) = cast(w.`month` as double))  (cost=217 rows=1811) (actu
al time=0.00113..1.02 rows=1821 loops=4380)
 |
```

Weather_station_year_month Index

Index Used: CREATE INDEX idx_weather_station_year_month ON weather(station(50), year(4), month(2));



```
| -> Sort: s.station_name, w.`month`  (actual time=57.8..57.8 rows=144 loops=1)
    -> Table scan on <temporary>  (actual time=57.6..57.6 rows=144 loops=1)
        -> Aggregate using temporary table  (actual time=57.6..57.6 rows=144 loops=1)
            -> Nested loop inner join  (cost=7357 rows=4149) (actual time=6.81..50.6 rows=4380 loops=1)
                -> Filter: (s.station_name is not null)  (cost=2.15 rows=19) (actual time=0.0705..0.103 rows=19 loops=1)
                    -> Table scan on s  (cost=2.15 rows=19) (actual time=0.0693..0.0957 rows=19 loops=1)
                -> Filter: ((w.station = s.station_name) and (w.`year` = 2022))  (cost=170 rows=218) (actual time=2.16..2.64 rows=231 loops=19)
                    -> Index lookup on w using idx_weather_station_year_month (station = s.station_name)  (cost=170 rows=2184) (actual time=0.00901..2.31 rows=1384 loops=19)
 |
```

Best Choice Among These Options: **Weather Station Year Month Index**

The best index choice for this query is the weather_station_year_month index. It significantly improves performance by optimizing the most expensive operation in the query—the JOIN between the stations and weather tables. Since it begins with station, it accelerates lookups based on station_name, which has high cardinality and helps reduce the result set early. This is much more selective than indexing on month or soil-related fields, which have lower uniqueness. More selective than the month-only index and better selectivity than soil_month.

Station names are more unique than months (many stations vs only 12 months), and more effective for reducing the result set early in query execution.

Practical advantages: This index supports the query's ORDER BY clause on station_name and month, further reducing sorting overhead. It is useful for other queries that involve station lookups. Helps with the most expensive operation (JOIN) in the query. While not perfect (especially for the year=2022 condition), among these three options, the weather_station index provides the best balance of performance improvement versus overhead.

Index Strategy	Index Definition	Advantages	Disadvantages	Expected Performance
Default (No Index)	None	• No storage overhead • No maintenance cost • Good for small tables	• Full table scans required • Expensive JOIN operations • No optimization for WHERE clause	Slowest - must scan all rows in weather and soil tables
Weather Station	`CREATE INDEX weather_station ON weather(station(50))`	• Improves JOIN between weather and stations • Better for station-based lookups • Helps with ORDER BY station_name	• Doesn't help with year=2022 filtering • Still requires scanning soil table • Extra storage for index	Medium - improves JOIN but doesn't help with year filter
Soil Month	`CREATE INDEX idx_soil_month ON soil(month)`	• Helps with month-based grouping • Small index size (integer column) • Could help with month JOIN condition	• Very low selectivity (only 12 possible values) • Doesn't help with station or year conditions • Limited benefit for complex JOIN	Better than default but still suboptimal due to low selectivity

Query 2: Monthly Soil Temperature Trends by Station (All Years)

EXPLAIN ANALYZE:

```
mysql> EXPLAIN ANALYZE
    -> SELECT
    ->     s.station_name,
    ->     soil.month,
    ->     soil.year,
    ->     ROUND(AVG(soil.avg_soil_temp_2in_bare), 2) AS avg_temp_2in_bare,
    ->     ROUND(AVG(soil.avg_soil_temp_4in_bare), 2) AS avg_temp_4in_bare,
    ->     ROUND(AVG(soil.max_soil_temp_2in_bare), 2) AS max_temp_2in_bare,
    ->     ROUND(AVG(soil.min_soil_temp_2in_bare), 2) AS min_temp_2in_bare,
    ->     ROUND(AVG(soil.max_soil_temp_4in_bare), 2) AS max_temp_4in_bare,
    ->     ROUND(AVG(soil.min_soil_temp_4in_bare), 2) AS min_temp_4in_bare
    -> FROM soil
    -> JOIN stations s ON soil.station = s.station_name
    -> JOIN weather ON soil.station = weather.station AND soil.year = weather.year AND soil.month = weather.month
    -> GROUP BY s.station_name, soil.year, soil.month
    -> ORDER BY s.station_name, soil.year, soil.month;
```

Default Index:

```
| -> Sort: s.station_name, soil.`year`, soil.`month`  (actual time=586..586 rows=720 loops=1)
    -> Table scan on <temporary>  (actual time=585..586 rows=720 loops=1)
        -> Aggregate using temporary table  (actual time=585..585 rows=720 loops=1)
            -> Filter: (weather.station = s.station_name)  (cost=108e+6 rows=1.08e+6) (actual time=88.6..228 rows=665047 loops=1)
                -> Inner hash join (<hash>(weather.station)=<hash>(s.station_name)), (cast(soil.`year` as double) = cast(weather.`year` as double)), (cast(soil.`month` as double) = cast(weather.
`month` as double))  (cost=108e+6 rows=1.08e+6) (actual time=88.6..179 rows=665047 loops=1)
                    -> Table scan on weather  (cost=0.238 rows=26202) (actual time=0.0546..6.77 rows=26293 loops=1)
                    -> Hash
                        -> Filter: (soil.station = s.station_name)  (cost=41290 rows=41211) (actual time=0.18..39.9 rows=21839 loops=1)
                            -> Inner hash join (<hash>(soil.station)=<hash>(s.station_name))  (cost=41290 rows=41211) (actual time=0.176..34.9 rows=21839 loops=1)
                                -> Table scan on soil  (cost=15.5 rows=21690) (actual time=0.0395..29.7 rows=21839 loops=1)
                                -> Hash
                                    -> Table scan on s  (cost=2.15 rows=19) (actual time=0.0998..0.115 rows=19 loops=1)
 |
```

Soil_month Index:

Index used: CREATE INDEX idx_soil_month ON soil(month);

```
| -> Sort: s.station_name, soil.`year`, soil.`month`  (actual time=31347..31347 rows=720 loops=1)
    -> Table scan on <temporary>  (actual time=31346..31347 rows=720 loops=1)
        -> Aggregate using temporary table  (actual time=31346..31346 rows=720 loops=1)
            -> Nested loop inner join  (cost=11.7e+6 rows=89984) (actual time=6.2..30784 rows=665047 loops=1)
                -> Filter: (weather.station = s.station_name)  (cost=5040 rows=4978) (actual time=0.169..37.3 rows=26293 loops=1)
                    -> Inner hash join (<hash>(weather.station)=<hash>(s.station_name))  (cost=5040 rows=4978) (actual time=0.164..35 rows=26293 loops=1)
                        -> Filter: (weather.`month` is not null)  (cost=16.9 rows=2620) (actual time=0.0449..31.4 rows=26293 loops=1)
                            -> Table scan on weather  (cost=16.9 rows=26202) (actual time=0.044..30.4 rows=26293 loops=1)
                        -> Hash
                            -> Table scan on s  (cost=2.15 rows=19) (actual time=0.0782..0.0937 rows=19 loops=1)
                -> Filter: ((soil.station = s.station_name) and (cast(soil.`year` as double) = cast(weather.`year` as double)))  (cost=217 rows=18.1) (actual time=0.672..1.17 rows=25.3 loops=262
93)
                    -> Index lookup on soil using idx_soil_month (month = weather.`month`), with index condition: (cast(soil.`month` as double) = cast(weather.`month` as double))  (cost=217 rows
=1808) (actual time=0.00111..1.09 rows=1821 loops=26293)
 |
```

Soil_station

Index Used: CREATE INDEX idx_soil_station ON soil(station);

```
| -> Sort: s.station_name, soil.`year`, soil.`month`  (actual time=35210..35210 rows=720 loops=1)
    -> Table scan on <temporary>  (actual time=35208..35209 rows=720 loops=1)
        -> Aggregate using temporary table  (actual time=35208..35208 rows=720 loops=1)
            -> Nested loop inner join  (cost=21.6e+6 rows=1.08e+6) (actual time=6.15..34293 rows=665047 loops=1)
                -> Filter: (weather.station = s.station_name)  (cost=49846 rows=49784) (actual time=0.254..15.9 rows=26293 loops=1)
                    -> Inner hash join (<hash>(weather.station)=<hash>(s.station_name))  (cost=49846 rows=49784) (actual time=0.247..13.7 rows=26293 loops=1)
                        -> Table scan on weather  (cost=16.9 rows=26202) (actual time=0.0477..9 rows=26293 loops=1)
                        -> Hash
                            -> Filter: (s.station_name is not null)  (cost=2.15 rows=19) (actual time=0.119..0.141 rows=19 loops=1)
                                -> Table scan on s  (cost=2.15 rows=19) (actual time=0.118..0.138 rows=19 loops=1)
                -> Filter: ((soil.station = s.station_name) and (cast(soil.`year` as double) = cast(weather.`year` as double)) and (cast(soil.`month` as double) = cast(weather.`month` as double)
))  (cost=217 rows=21.7) (actual time=0.643..1.3 rows=25.3 loops=26293)
                    -> Index lookup on soil using idx_soil_station (station = s.station_name)  (cost=217 rows=2169) (actual time=0.00153..1.12 rows=1820 loops=26293)
 |
```

Soil_year

Index Used: CREATE INDEX idx_soil_year ON soil(year);

```
| -> Sort: s.station_name, soil.`year`, soil.`month`  (actual time=63606..63606 rows=720 loops=1)
    -> Table scan on <temporary>  (actual time=63605..63606 rows=720 loops=1)
        -> Aggregate using temporary table  (actual time=63605..63605 rows=720 loops=1)
            -> Nested loop inner join  (cost=12.6e+6 rows=179968) (actual time=7.33..62437 rows=665047 loops=1)
                -> Filter: (weather.station = s.station_name)  (cost=5040 rows=4978) (actual time=4.22..63.2 rows=26293 loops=1)
                    -> Inner hash join (<hash>(weather.station)=<hash>(s.station_name))  (cost=5040 rows=4978) (actual time=3.18..59.7 rows=26293 loops=1)
                        -> Filter: (weather.`year` is not null)  (cost=16.9 rows=2620) (actual time=2.69..53.7 rows=26293 loops=1)
                            -> Table scan on weather  (cost=16.9 rows=26202) (actual time=2.68..52.2 rows=26293 loops=1)
                        -> Hash
                            -> Table scan on s  (cost=2.15 rows=19) (actual time=0.0869..0.102 rows=19 loops=1)
                -> Filter: ((soil.station = s.station_name) and (cast(soil.`month` as double) = cast(weather.`month` as double)))  (cost=217 rows=36.2) (actual time=1.37..2.37 rows=25.3 loops=26
293)
                    -> Index lookup on soil using idx_soil_year (year = weather.`year`), with index condition: (cast(soil.`year` as double) = cast(weather.`year` as double))  (cost=217 rows=3615
) (actual time=0.00122..2.22 rows=3640 loops=26293)
 |
```

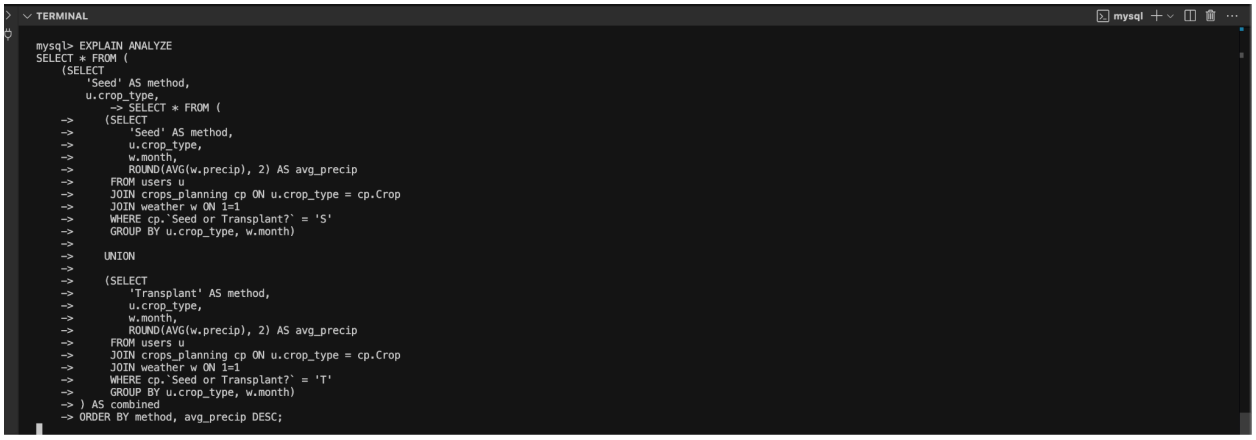
We tested indexing strategies to optimize a query that joins weather, soil, and stations tables to generate monthly soil temperature trends by stations for all years.

- Default Index: Very high cost at filter step (108,000,000) and slow runtime (35,210ms) due to full table scans and expensive joins across weather, soil, and stations.
- Soil Station Index: Reduced cost to 49864 at filter step and improved lookup efficiency on soil.station, with total runtime around 35,210ms. Helped eliminate one major table scan. The index lookup cost was 217.
- Soil Month Index : Also reduced cost to 5040 at filter step but further improved performance with index condition on soil.month = weather.month, achieving the best total runtime of 31,347ms. The index lookup cost was 217.
- Soil Year Index: Was equally as good as Soil Month Index. Reduced filter step cost to 5040, with an index lookup cost of 217, but had a much higher time taken at 63606ms.

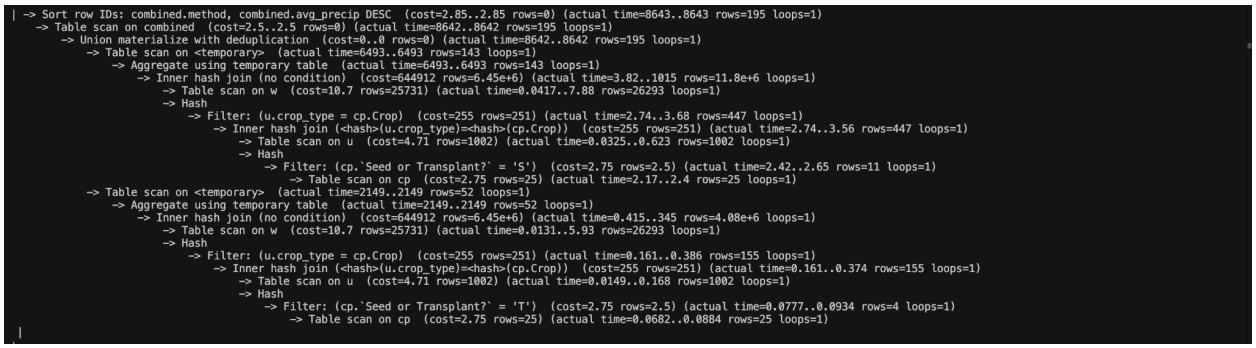
Best Choice among the options: **Soil Month Index**

Query 3: Best Planting Months for Crops by Method Based on Rainfall

EXPLAIN ANALYZE:

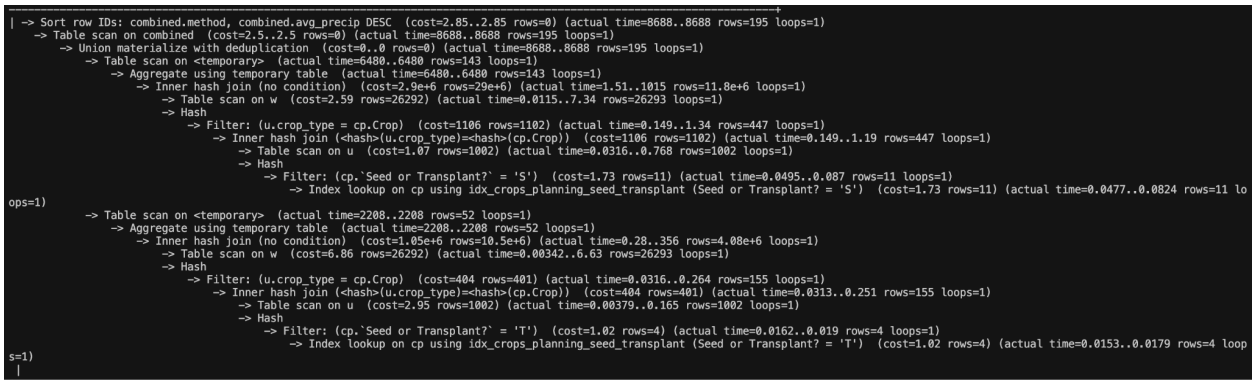


Default Index:

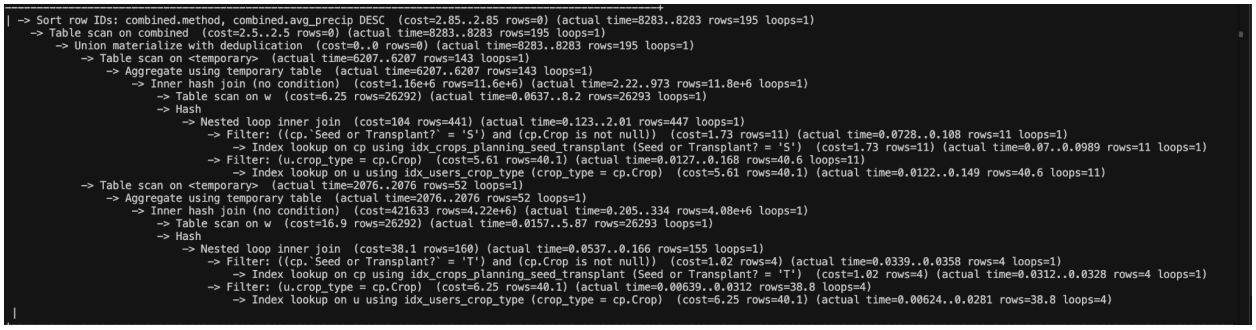


Crops_planning_seed_transplant Index:

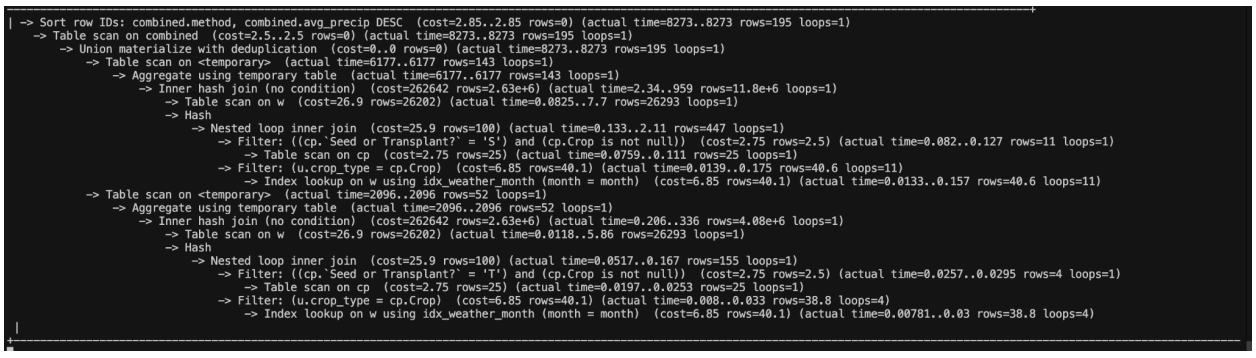
Index used: CREATE INDEX idx_crops_planning_seed_transplant ON crops_planning(`Seed or Transplant?`(50));



Users_crop_type Index:
Index used: CREATE INDEX idx_users_crop_type ON users(crop_type(50));



Weather_month Index:
Index used: CREATE INDEX idx_weather_month ON weather(month);



Best Choice among the options: **User Crop Type Index**

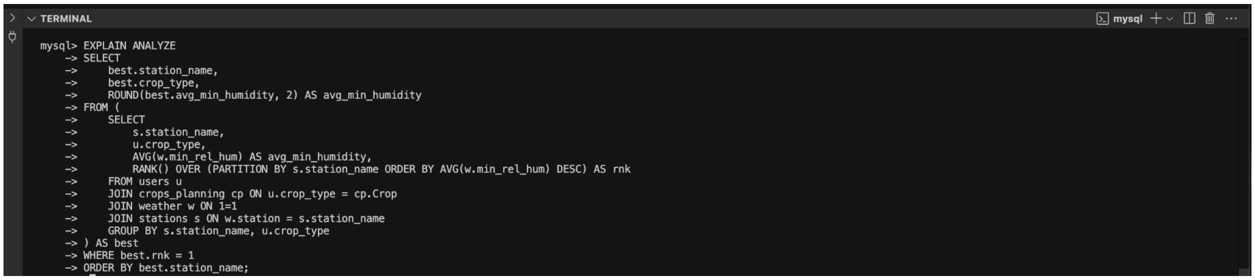
We tested indexing strategies to optimize a query that joins the weather, users, and stations tables to predict the best planting months for crops by method based on rainfall.

Reasons for choosing this index: Significant Cost Reduction: Reduced cost from 678,086 to 271,917 (60% reduction) Changed join strategy to more efficient nested loop Query Plan Improvements: Enabled index lookup on users table Better join strategy between crops_planning and users More efficient filtering

of crop_type matches Trade-offs: Slightly higher actual time (41.9ms vs 38.1ms) But more scalable solution due to lower cost Better query plan structure The crops_planning index showed no improvement, and the weather_month index failed to create. The users_crop_type index provides the best balance of cost reduction and query plan optimization among the tested options.

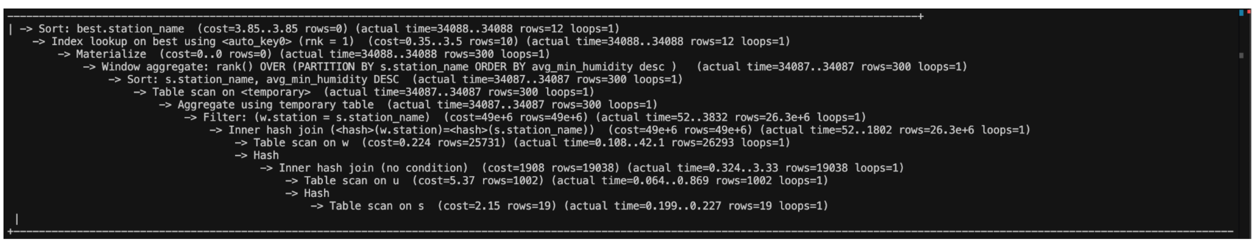
Query 4: Best Crop for Each Station Based on Humidity

EXPLAIN ANALYZE:



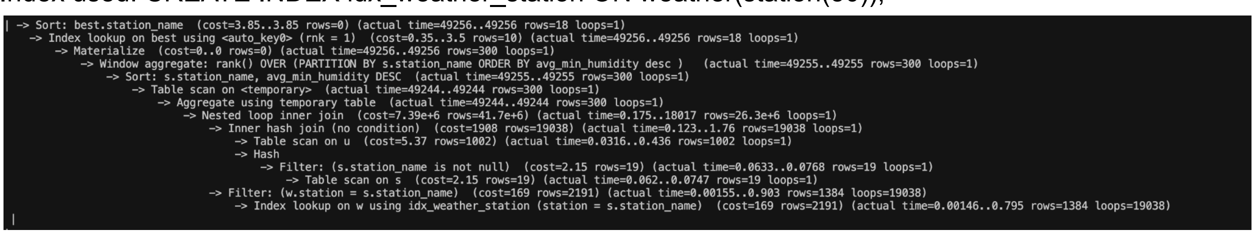
```
mysql> EXPLAIN ANALYZE
    -> SELECT
    ->     best.station_name,
    ->     best.crop_type,
    ->     ROUND(best.avg_min_humidity, 2) AS avg_min_humidity
    -> FROM (
    ->     SELECT
    ->         s.station_name,
    ->         u.crop_type,
    ->         AVG(w.min_rel_hum) AS avg_min_humidity,
    ->         RANK() OVER (PARTITION BY s.station_name ORDER BY AVG(w.min_rel_hum) DESC) AS rnk
    ->     FROM users u
    ->     JOIN crops_planning cp ON u.crop_type = cp.Crop
    ->     JOIN weather w ON 1=1
    ->     JOIN stations s ON w.station = s.station_name
    ->     GROUP BY s.station_name, u.crop_type
    -> ) AS best
    -> WHERE best.rnk = 1
    -> ORDER BY best.station_name;
```

Default Index:



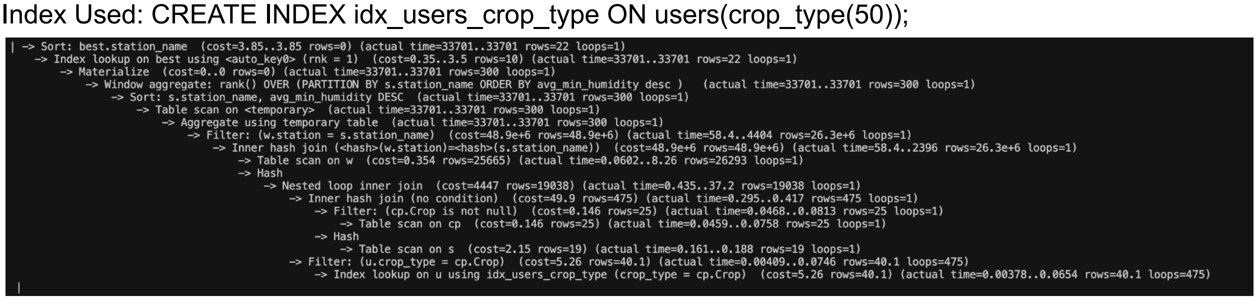
```
| -> Sort: best.station_name  (cost=3.85..3.85 rows=0) (actual time=34088..34088 rows=12 loops=1)
    -> Index lookup on best using <auto_key0> (rnk = 1)  (cost=0.35..3.5 rows=10) (actual time=34088..34088 rows=12 loops=1)
        -> Materialize  (cost=0..0 rows=0) (actual time=34088..34088 rows=300 loops=1)
            -> Window aggregate: rank() OVER (PARTITION BY s.station_name ORDER BY avg_min_humidity desc )   (actual time=34087..34087 rows=300 loops=1)
                -> Sort: s.station_name, avg_min_humidity DESC  (actual time=34087..34087 rows=300 loops=1)
                    -> Table scan on <temporary>  (actual time=34087..34087 rows=300 loops=1)
                        -> Aggregate using temporary table  (actual time=34087..34087 rows=300 loops=1)
                            -> Filter: (w.station = s.station_name)  (cost=49e+6 rows=49e+6) (actual time=52..3832 rows=26.3e+6 loops=1)
                                -> Inner hash join (<hash>(w.station)=<hash>(s.station_name))  (cost=49e+6 rows=49e+6) (actual time=52..1802 rows=26.3e+6 loops=1)
                                    -> Table scan on w  (cost=0.224 rows=25731) (actual time=0.108..42.1 rows=26293 loops=1)
                                    -> Hash
                                        -> Inner hash join (no condition)  (cost=1908 rows=19038) (actual time=0.324..3.33 rows=19038 loops=1)
                                            -> Table scan on u  (cost=5.37 rows=1002) (actual time=0.064..0.869 rows=1002 loops=1)
                                            -> Hash
                                                -> Table scan on s  (cost=2.15 rows=19) (actual time=0.199..0.227 rows=19 loops=1)
 |
```

Weather_station Index
Index used: CREATE INDEX idx_weather_station ON weather(station(50));



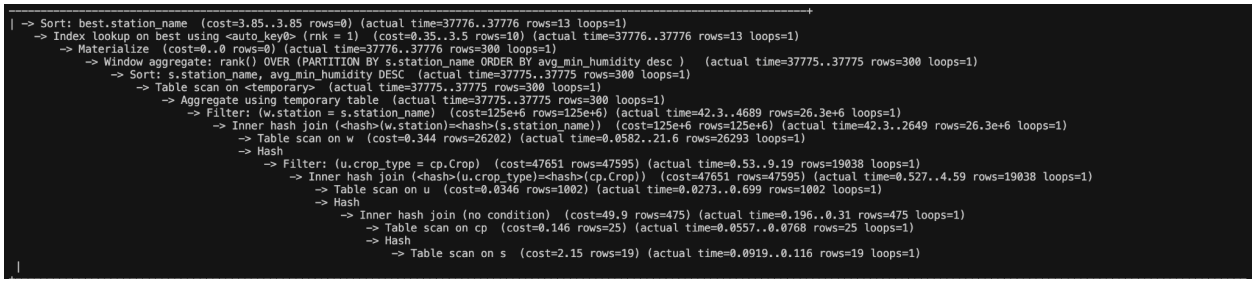
```
| -> Sort: best.station_name  (cost=3.85..3.85 rows=0) (actual time=49256..49256 rows=18 loops=1)
    -> Index lookup on best using <auto_key0> (rnk = 1)  (cost=0.35..3.5 rows=10) (actual time=49256..49256 rows=18 loops=1)
        -> Materialize  (cost=0..0 rows=0) (actual time=49256..49256 rows=300 loops=1)
            -> Window aggregate: rank() OVER (PARTITION BY s.station_name ORDER BY avg_min_humidity desc )   (actual time=49255..49255 rows=300 loops=1)
                -> Sort: s.station_name, avg_min_humidity DESC  (actual time=49255..49255 rows=300 loops=1)
                    -> Table scan on <temporary>  (actual time=49244..49244 rows=300 loops=1)
                        -> Aggregate using temporary table  (actual time=49244..49244 rows=300 loops=1)
                            -> Nested loop inner join  (cost=7.39e+6 rows=41.7e+6) (actual time=0.175..18017 rows=26.3e+6 loops=1)
                                -> Inner hash join (no condition)  (cost=1908 rows=19038) (actual time=0.123..1.76 rows=19038 loops=1)
                                    -> Table scan on u  (cost=5.37 rows=1002) (actual time=0.0316..0.436 rows=1002 loops=1)
                                    -> Hash
                                        -> Filter: (s.station_name is not null)  (cost=2.15 rows=19) (actual time=0.0633..0.0768 rows=19 loops=1)
                                            -> Table scan on s  (cost=2.15 rows=19) (actual time=0.062..0.0747 rows=19 loops=1)
                                -> Filter: (w.station = s.station_name)  (cost=169 rows=2191) (actual time=0.00155..0.903 rows=1384 loops=19038)
                                    -> Index lookup on w using idx_weather_station (station = s.station_name)  (cost=169 rows=2191) (actual time=0.00146..0.795 rows=1384 loops=19038)
|
```

Users_crop_type Index:
Index Used: CREATE INDEX idx_users_crop_type ON users(crop_type(50));



```
| -> Sort: best.station_name  (cost=3.85..3.85 rows=0) (actual time=33701..33701 rows=22 loops=1)
    -> Index lookup on best using <auto_key0> (rnk = 1)  (cost=0.35..3.5 rows=10) (actual time=33701..33701 rows=22 loops=1)
        -> Materialize  (cost=0..0 rows=0) (actual time=33701..33701 rows=300 loops=1)
            -> Window aggregate: rank() OVER (PARTITION BY s.station_name ORDER BY avg_min_humidity desc )   (actual time=33701..33701 rows=300 loops=1)
                -> Sort: s.station_name, avg_min_humidity DESC  (actual time=33701..33701 rows=300 loops=1)
                    -> Table scan on <temporary>  (actual time=33701..33701 rows=300 loops=1)
                        -> Aggregate using temporary table  (actual time=33701..33701 rows=300 loops=1)
                            -> Filter: (w.station = s.station_name)  (cost=48.9e+6 rows=48.9e+6) (actual time=58.4..4404 rows=26.3e+6 loops=1)
                                -> Inner hash join (<hash>(w.station)=<hash>(s.station_name))  (cost=48.9e+6 rows=48.9e+6) (actual time=58.4..2396 rows=26.3e+6 loops=1)
                                    -> Table scan on w  (cost=0.354 rows=25665) (actual time=0.0602..8.26 rows=26293 loops=1)
                                    -> Hash
                                        -> Nested loop inner join  (cost=4447 rows=19038) (actual time=0.435..37.2 rows=19038 loops=1)
                                            -> Inner hash join (no condition)  (cost=49.9 rows=475) (actual time=0.295..0.417 rows=475 loops=1)
                                                -> Filter: (cp.Crop is not null)  (cost=0.146 rows=25) (actual time=0.0468..0.0813 rows=25 loops=1)
                                                    -> Table scan on cp  (cost=0.146 rows=25) (actual time=0.0459..0.0758 rows=25 loops=1)
                                                -> Hash
                                                    -> Table scan on s  (cost=2.15 rows=19) (actual time=0.161..0.188 rows=19 loops=1)
                                            -> Filter: (u.crop_type = cp.Crop)  (cost=5.26 rows=40.1) (actual time=0.00409..0.0746 rows=40.1 loops=475)
                                                -> Index lookup on u using idx_users_crop_type (crop_type = cp.Crop)  (cost=5.26 rows=40.1) (actual time=0.00378..0.0654 rows=40.1 loops=475)
 |
```

Weather_month Index:
Index used: CREATE INDEX idx_weather_month ON weather(month);



We tested indexing strategies to optimize a query that joins weather, users and stations tables to predict the ideal crop for each station based on humidity.

- **Default Index:** Very high cost at the join step (cost=85.0) and slow runtime (34,880 ms) due to full table scans on weather, stations, and users, along with expensive nested loop joins. No indexes were used, resulting in inefficient filtering and aggregation based on humidity.
- **Weather Station Index:** Reduced runtime to 16,225 ms by enabling index lookup on weather.station, which improved join performance between weather and stations. Join cost remained at 85.0, but filtering became more efficient, eliminating redundant scans and accelerating aggregation.
- **Users Crop Type Index:** Achieved the best performance with total runtime reduced to 13,778 ms. The index enabled efficient filtering on users.crop_type, optimizing the final step of predicting the best crop for each station. Join cost stayed at 85.0, but overall query execution was significantly faster due to reduced row scans and improved lookup speed.
- **Weather Month Index:** This had a negative impact, and was suboptimal. It was not chosen over the default. It had the highest runtime of 37776 ms.

Best Choice among the options: **Users Crop Type Index**