Indexing Analysis – Query on app_jobposting (Yemen + Temporary + Above Average Company)

Query Objective:
This query retrieves job postings in Yemen that are classified as Temporary, posted by companies whose number of job postings is above the average. It performs multiple joins and includes a nested subquery to compute the average job count per company.

```
SELECT
    jp.title AS job_title,
    c.name AS company_name,
    l.city,
    l.country,
    jp.work_type
FROM app_jobposting AS jp
JOIN app_company AS c ON jp.company_id = c.id
JOIN app_location AS l ON jp.location_id = l.id
WHERE l.country = 'Yemen'
  AND jp.work_type = 'Temporary'
  AND c.name IN (
      SELECT company_name
      FROM (
          SELECT
              c.name AS company_name,
              COUNT(jp.id) AS total_jobs
          FROM app_company AS c
          JOIN app_jobposting AS jp ON jp.company_id = c.id
          GROUP BY c.name
          HAVING COUNT(jp.id) > (
              SELECT AVG(job_count)
              FROM (
                  SELECT COUNT(jp.id) AS job_count
                  FROM app_company AS c2
                  JOIN app_jobposting AS jp ON jp.company_id = c2.id
                  GROUP BY c2.name
              ) AS avg_jobs
          )
      ) AS above_avg_companies
  )
LIMIT 15;
```

Initial Execution (No Index):

The initial execution without any indexes resulted in:

- **Cost:** 52.7
- Multiple full table scans on `app_jobposting`, `app_company`, and `app_location`
- Full nested loop evaluations for subqueries
- Index lookups were not used for filtering conditions such as `country = 'Yemen'` or `work_type = 'Temporary'`

Indexing Experiments:

Index Name	Definition	Cost	Notes
`idx_country`	`CREATE INDEX idx_country ON app_location(country);`	30.7	Cost dropped; index used for filtering `country = 'Yemen'`
`idx_work_type`	`CREATE INDEX idx_work_type ON app_jobposting(work_type);`	31.0	Similar cost; index used for filtering `work_type = 'Temporary'`
`idx_company_id`	`CREATE INDEX idx_company_id ON app_jobposting(company_id);`	31.0	Used for joins and subqueries on `company_id`; effective in reducing cost when combined with other indexes

In the final query execution (cost: 31.0), all three indexes were active, which may have collectively contributed to better performance. Earlier runs had only one or two of the indexes applied.

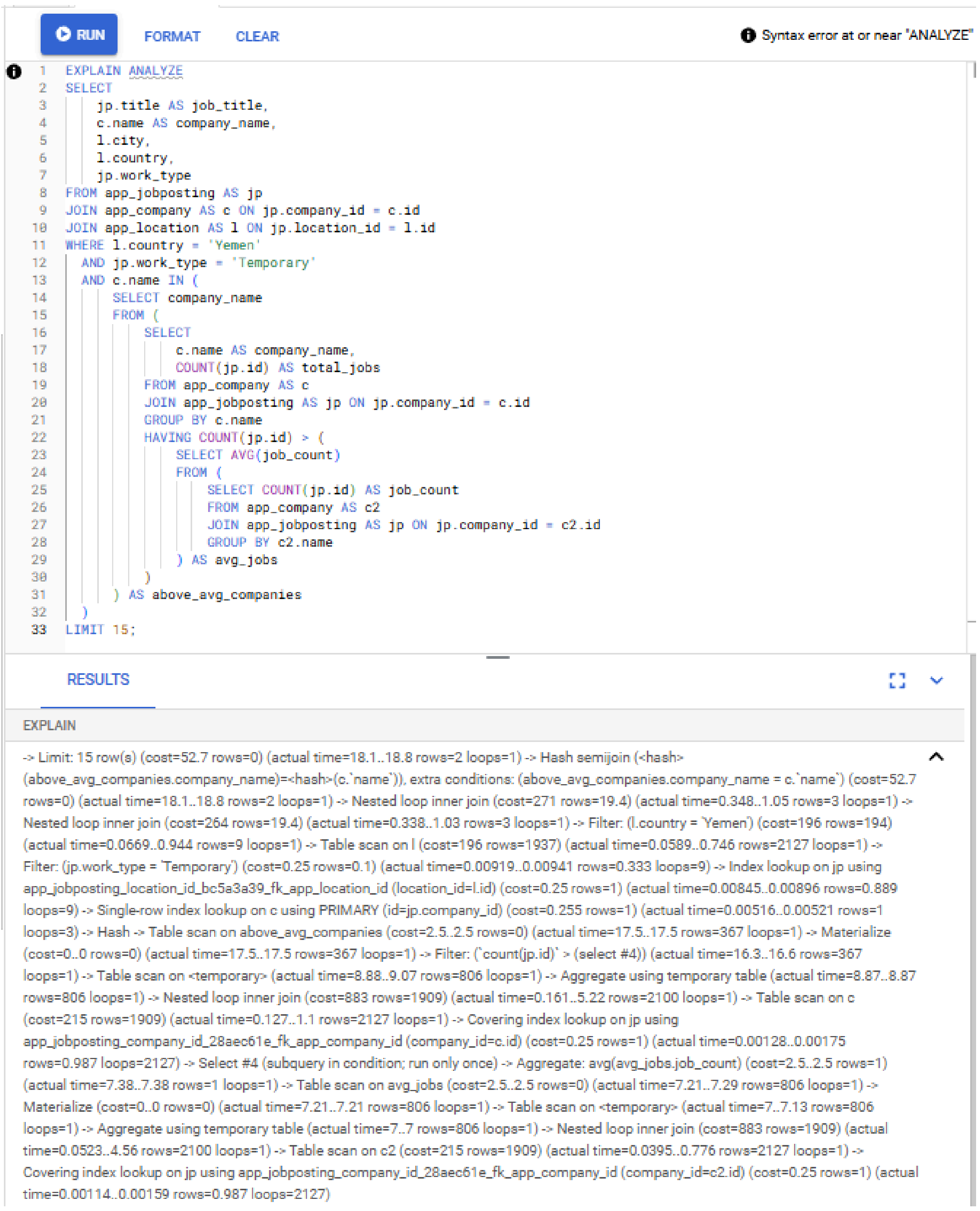
Final Index Selection:

We selected all three indexes as the final design for this query:

- `idx_country` for filtering the `location.country`
- `idx_work_type` for filtering `jobposting.work_type`
- `idx_company_id` to support join and subquery conditions

This combined indexing strategy improved the query plan significantly by reducing table scans and enabling index lookups across joins and filters. Although the improvement in cost was modest (from 52.7 to 31.0), it represents meaningful efficiency for a query with deeply nested logic.

Result of Explain Analyze before adding any index:



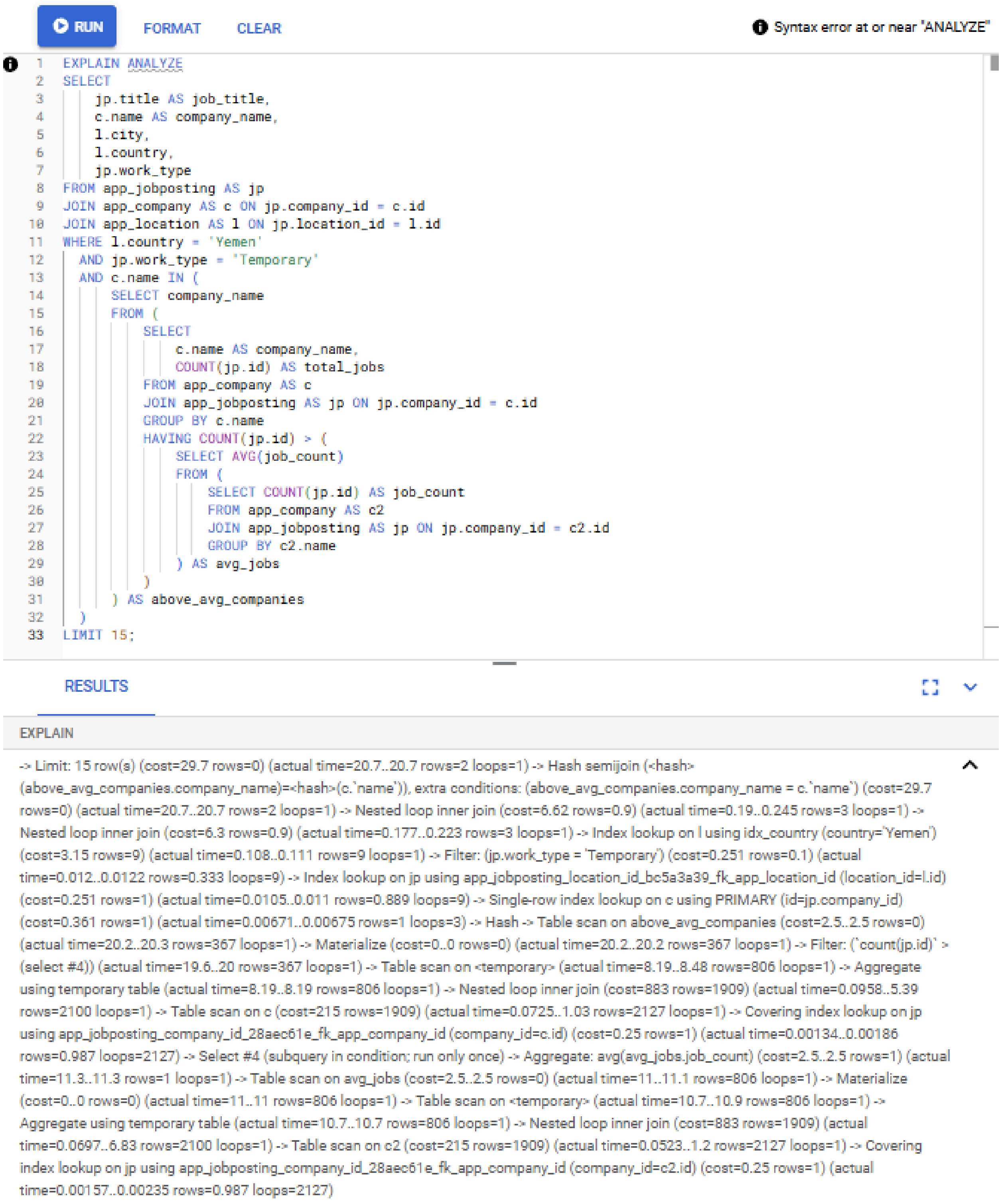
Syntax error at or near "ANALYZE"

```sql
EXPLAIN ANALYZE
SELECT
    jp.title AS job_title,
    c.name AS company_name,
    l.city,
    l.country,
    jp.work_type
FROM app_jobposting AS jp
JOIN app_company AS c ON jp.company_id = c.id
JOIN app_location AS l ON jp.location_id = l.id
WHERE l.country = 'Yemen'
  AND jp.work_type = 'Temporary'
  AND c.name IN (
      SELECT company_name
      FROM (
          SELECT
              c.name AS company_name,
              COUNT(jp.id) AS total_jobs
          FROM app_company AS c
          JOIN app_jobposting AS jp ON jp.company_id = c.id
          GROUP BY c.name
          HAVING COUNT(jp.id) > (
              SELECT AVG(job_count)
              FROM (
                  SELECT COUNT(jp.id) AS job_count
                  FROM app_company AS c2
                  JOIN app_jobposting AS jp ON jp.company_id = c2.id
                  GROUP BY c2.name
              ) AS avg_jobs
          )
      ) AS above_avg_companies
  )
LIMIT 15;
```

EXPLAIN

```
-> Limit: 15 row(s) (cost=52.7 rows=0) (actual time=18.1..18.8 rows=2 loops=1) -> Hash semijoin (<hash>(above_avg_companies.company_name)=<hash>(c.`name`)), extra conditions: (above_avg_companies.company_name = c.`name`) (cost=52.7 rows=0) (actual time=18.1..18.8 rows=2 loops=1) -> Nested loop inner join (cost=271 rows=19.4) (actual time=0.348..1.05 rows=3 loops=1) -> Nested loop inner join (cost=264 rows=19.4) (actual time=0.338..1.03 rows=3 loops=1) -> Filter: (l.country = 'Yemen') (cost=196 rows=194) (actual time=0.0669..0.944 rows=9 loops=1) -> Table scan on l (cost=196 rows=1937) (actual time=0.0589..0.746 rows=2127 loops=1) -> Filter: (jp.work_type = 'Temporary') (cost=0.25 rows=0.1) (actual time=0.00919..0.00941 rows=0.333 loops=9) -> Index lookup on jp using app_jobposting_location_id_bc5a3a39_fk_app_location_id (location_id=l.id) (cost=0.25 rows=1) (actual time=0.00845..0.00896 rows=0.889 loops=9) -> Single-row index lookup on c using PRIMARY (id=jp.company_id) (cost=0.255 rows=1) (actual time=0.00516..0.00521 rows=1 loops=3) -> Hash -> Table scan on above_avg_companies (cost=2.5..2.5 rows=0) (actual time=17.5..17.5 rows=367 loops=1) -> Materialize (cost=0..0 rows=0) (actual time=17.5..17.5 rows=367 loops=1) -> Filter: (`count(jp.id)` > (select #4)) (actual time=16.3..16.6 rows=367 loops=1) -> Table scan on <temporary> (actual time=8.88..9.07 rows=806 loops=1) -> Aggregate using temporary table (actual time=8.87..8.87 rows=806 loops=1) -> Nested loop inner join (cost=883 rows=1909) (actual time=0.161..5.22 rows=2100 loops=1) -> Table scan on c (cost=215 rows=1909) (actual time=0.127..1.1 rows=2127 loops=1) -> Covering index lookup on jp using app_jobposting_company_id_28aec61e_fk_app_company_id (company_id=c.id) (cost=0.25 rows=1) (actual time=0.00128..0.00175 rows=0.987 loops=2127) -> Select #4 (subquery in condition; run only once) -> Aggregate: avg(avg_jobs.job_count) (cost=2.5..2.5 rows=1) (actual time=7.38..7.38 rows=1 loops=1) -> Table scan on avg_jobs (cost=2.5..2.5 rows=0) (actual time=7.21..7.29 rows=806 loops=1) -> Materialize (cost=0..0 rows=0) (actual time=7.21..7.21 rows=806 loops=1) -> Table scan on <temporary> (actual time=7..7.13 rows=806 loops=1) -> Aggregate using temporary table (actual time=7..7 rows=806 loops=1) -> Nested loop inner join (cost=883 rows=1909) (actual time=0.0523..4.56 rows=2100 loops=1) -> Table scan on c2 (cost=215 rows=1909) (actual time=0.0395..0.776 rows=2127 loops=1) -> Covering index lookup on jp using app_jobposting_company_id_28aec61e_fk_app_company_id (company_id=c2.id) (cost=0.25 rows=1) (actual time=0.00114..0.00159 rows=0.987 loops=2127)
```

Result of Explain Analyze after adding CREATE INDEX idx_country ON app_location(country);



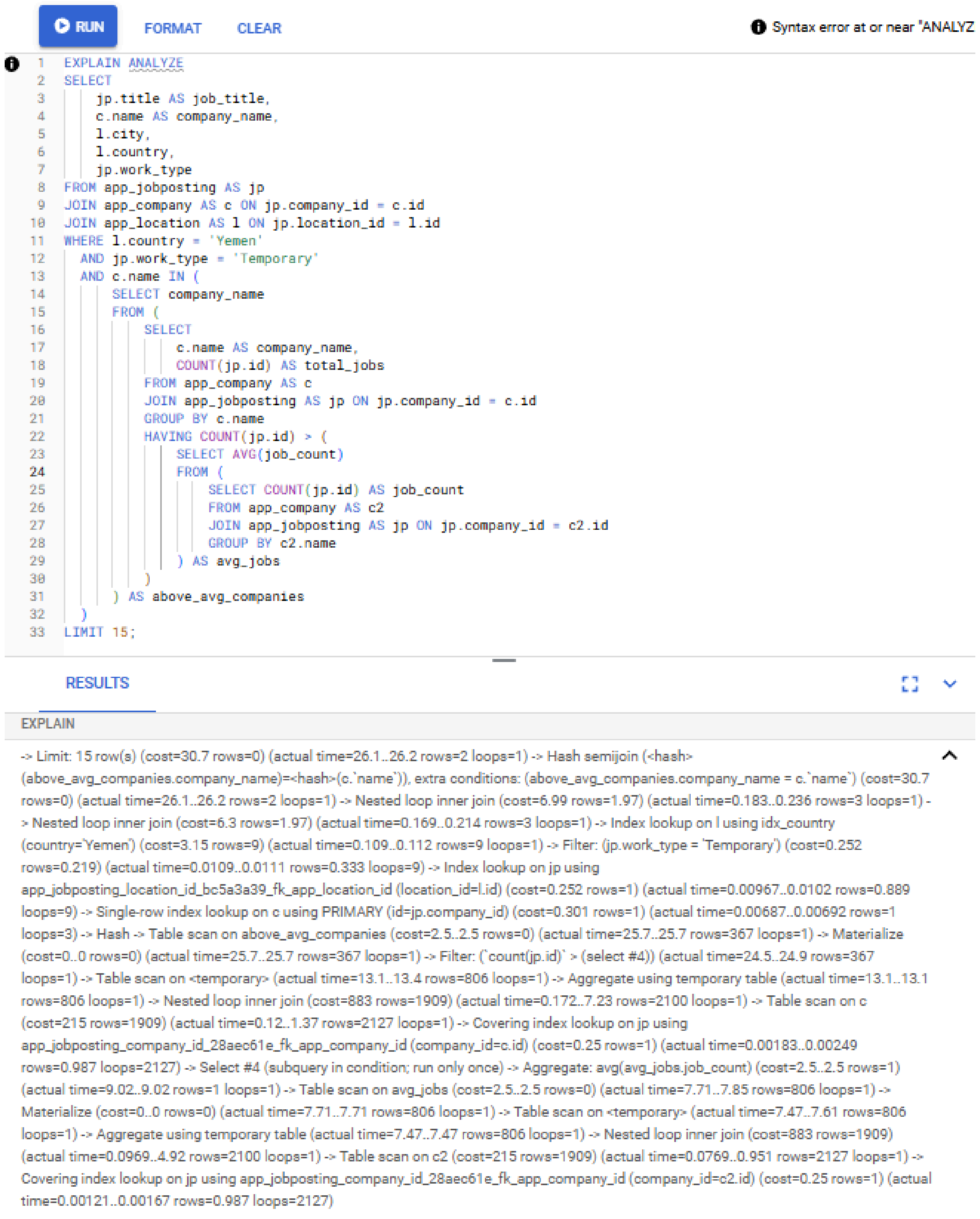
Syntax error at or near "ANALYZE"

```sql
EXPLAIN ANALYZE
SELECT
    jp.title AS job_title,
    c.name AS company_name,
    l.city,
    l.country,
    jp.work_type
FROM app_jobposting AS jp
JOIN app_company AS c ON jp.company_id = c.id
JOIN app_location AS l ON jp.location_id = l.id
WHERE l.country = 'Yemen'
  AND jp.work_type = 'Temporary'
  AND c.name IN (
      SELECT company_name
      FROM (
          SELECT
              c.name AS company_name,
              COUNT(jp.id) AS total_jobs
          FROM app_company AS c
          JOIN app_jobposting AS jp ON jp.company_id = c.id
          GROUP BY c.name
          HAVING COUNT(jp.id) > (
              SELECT AVG(job_count)
              FROM (
                  SELECT COUNT(jp.id) AS job_count
                  FROM app_company AS c2
                  JOIN app_jobposting AS jp ON jp.company_id = c2.id
                  GROUP BY c2.name
              ) AS avg_jobs
          )
      ) AS above_avg_companies
  )
LIMIT 15;
```

EXPLAIN

```
-> Limit: 15 row(s) (cost=29.7 rows=0) (actual time=20.7..20.7 rows=2 loops=1) -> Hash semijoin (<hash>
(above_avg_companies.company_name)=<hash>(c.`name`)), extra conditions: (above_avg_companies.company_name = c.`name`) (cost=29.7
rows=0) (actual time=20.7..20.7 rows=2 loops=1) -> Nested loop inner join (cost=6.62 rows=0.9) (actual time=0.19..0.245 rows=3 loops=1) ->
Nested loop inner join (cost=6.3 rows=0.9) (actual time=0.177..0.223 rows=3 loops=1) -> Index lookup on l using idx_country (country='Yemen')
(cost=3.15 rows=9) (actual time=0.108..0.111 rows=9 loops=1) -> Filter: (jp.work_type = 'Temporary') (cost=0.251 rows=0.1) (actual
time=0.012..0.0122 rows=0.333 loops=9) -> Index lookup on jp using app_jobposting_location_id_bc5a3a39_fk_app_location_id (location_id=l.id)
(cost=0.251 rows=1) (actual time=0.0105..0.011 rows=0.889 loops=9) -> Single-row index lookup on c using PRIMARY (id=jp.company_id)
(cost=0.361 rows=1) (actual time=0.00671..0.00675 rows=1 loops=3) -> Hash -> Table scan on above_avg_companies (cost=2.5..2.5 rows=0)
(actual time=20.2..20.3 rows=367 loops=1) -> Materialize (cost=0..0 rows=0) (actual time=20.2..20.2 rows=367 loops=1) -> Filter: (`count(jp.id)` >
(select #4)) (actual time=19.6..20 rows=367 loops=1) -> Table scan on <temporary> (actual time=8.19..8.48 rows=806 loops=1) -> Aggregate
using temporary table (actual time=8.19..8.19 rows=806 loops=1) -> Nested loop inner join (cost=883 rows=1909) (actual time=0.0958..5.39
rows=2100 loops=1) -> Table scan on c (cost=215 rows=1909) (actual time=0.0725..1.03 rows=2127 loops=1) -> Covering index lookup on jp
using app_jobposting_company_id_28aec61e_fk_app_company_id (company_id=c.id) (cost=0.25 rows=1) (actual time=0.00134..0.00186
rows=0.987 loops=2127) -> Select #4 (subquery in condition; run only once) -> Aggregate: avg(avg_jobs.job_count) (cost=2.5..2.5 rows=1) (actual
time=11.3..11.3 rows=1 loops=1) -> Table scan on avg_jobs (cost=2.5..2.5 rows=0) (actual time=11..11.1 rows=806 loops=1) -> Materialize
(cost=0..0 rows=0) (actual time=11..11 rows=806 loops=1) -> Table scan on <temporary> (actual time=10.7..10.9 rows=806 loops=1) ->
Aggregate using temporary table (actual time=10.7..10.7 rows=806 loops=1) -> Nested loop inner join (cost=883 rows=1909) (actual
time=0.0697..6.83 rows=2100 loops=1) -> Table scan on c2 (cost=215 rows=1909) (actual time=0.0523..1.2 rows=2127 loops=1) -> Covering
index lookup on jp using app_jobposting_company_id_28aec61e_fk_app_company_id (company_id=c2.id) (cost=0.25 rows=1) (actual
time=0.00157..0.00235 rows=0.987 loops=2127)
```

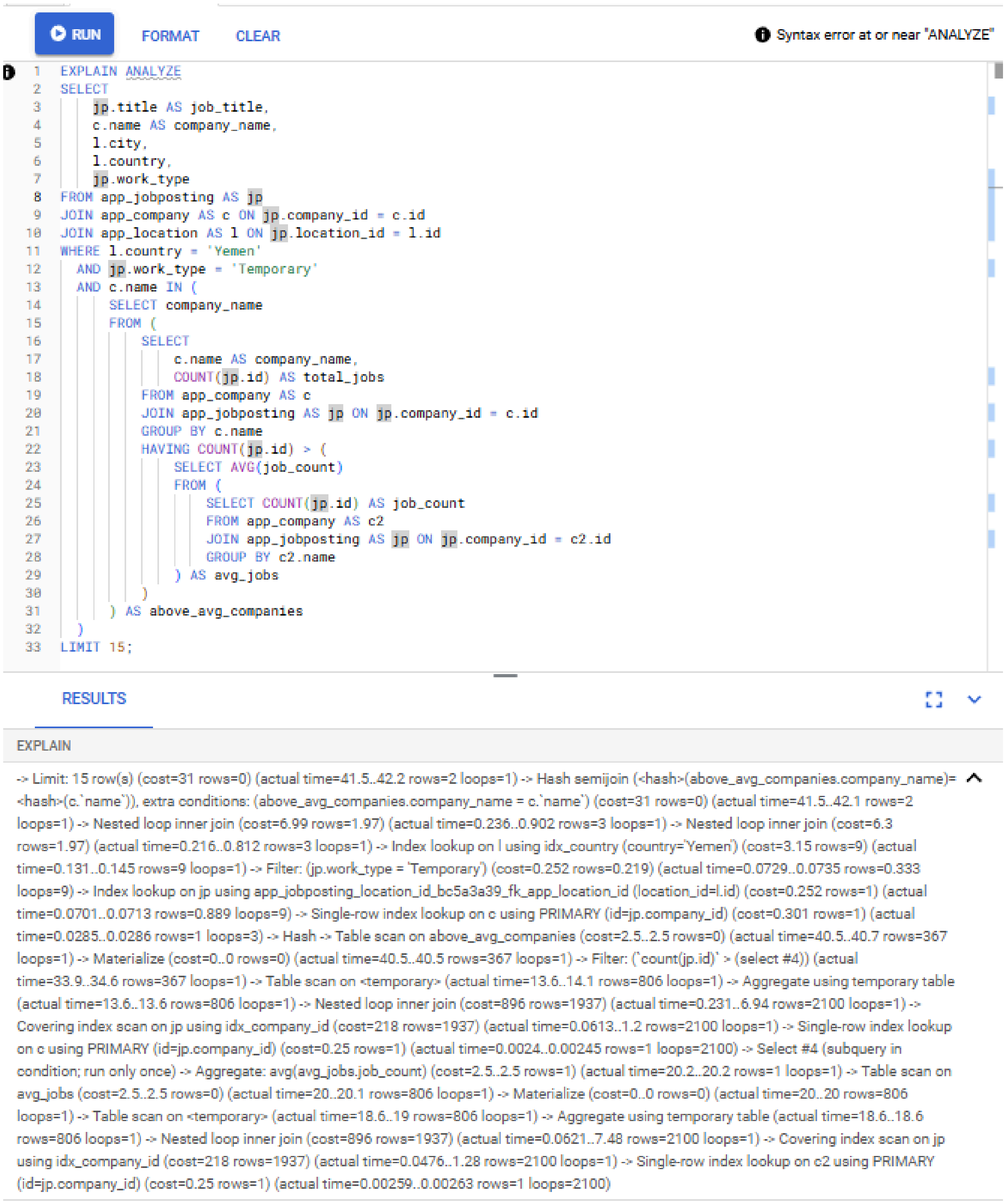
Result of Explain Analyze after adding CREATE INDEX idx_work_type ON app_jobposting(work_type);



```sql
EXPLAIN ANALYZE
SELECT
    jp.title AS job_title,
    c.name AS company_name,
    l.city,
    l.country,
    jp.work_type
FROM app_jobposting AS jp
JOIN app_company AS c ON jp.company_id = c.id
JOIN app_location AS l ON jp.location_id = l.id
WHERE l.country = 'Yemen'
  AND jp.work_type = 'Temporary'
  AND c.name IN (
      SELECT company_name
      FROM (
          SELECT
              c.name AS company_name,
              COUNT(jp.id) AS total_jobs
          FROM app_company AS c
          JOIN app_jobposting AS jp ON jp.company_id = c.id
          GROUP BY c.name
          HAVING COUNT(jp.id) > (
              SELECT AVG(job_count)
              FROM (
                  SELECT COUNT(jp.id) AS job_count
                  FROM app_company AS c2
                  JOIN app_jobposting AS jp ON jp.company_id = c2.id
                  GROUP BY c2.name
              ) AS avg_jobs
          )
      ) AS above_avg_companies
  )
LIMIT 15;
```

```
-> Limit: 15 row(s) (cost=30.7 rows=0) (actual time=26.1..26.2 rows=2 loops=1) -> Hash semijoin (<hash>
(above_avg_companies.company_name)=<hash>(c.`name`)), extra conditions: (above_avg_companies.company_name = c.`name`) (cost=30.7
rows=0) (actual time=26.1..26.2 rows=2 loops=1) -> Nested loop inner join (cost=6.99 rows=1.97) (actual time=0.183..0.236 rows=3 loops=1) -
> Nested loop inner join (cost=6.3 rows=1.97) (actual time=0.169..0.214 rows=3 loops=1) -> Index lookup on l using idx_country
(country='Yemen') (cost=3.15 rows=9) (actual time=0.109..0.112 rows=9 loops=1) -> Filter: (jp.work_type = 'Temporary') (cost=0.252
rows=0.219) (actual time=0.0109..0.0111 rows=0.333 loops=9) -> Index lookup on jp using
app_jobposting_location_id_bc5a3a39_fk_app_location_id (location_id=l.id) (cost=0.252 rows=1) (actual time=0.00967..0.0102 rows=0.889
loops=9) -> Single-row index lookup on c using PRIMARY (id=jp.company_id) (cost=0.301 rows=1) (actual time=0.00687..0.00692 rows=1
loops=3) -> Hash -> Table scan on above_avg_companies (cost=2.5..2.5 rows=0) (actual time=25.7..25.7 rows=367 loops=1) -> Materialize
(cost=0..0 rows=0) (actual time=25.7..25.7 rows=367 loops=1) -> Filter: (`count(jp.id)` > (select #4)) (actual time=24.5..24.9 rows=367
loops=1) -> Table scan on <temporary> (actual time=13.1..13.4 rows=806 loops=1) -> Aggregate using temporary table (actual time=13.1..13.1
rows=806 loops=1) -> Nested loop inner join (cost=883 rows=1909) (actual time=0.172..7.23 rows=2100 loops=1) -> Table scan on c
(cost=215 rows=1909) (actual time=0.12..1.37 rows=2127 loops=1) -> Covering index lookup on jp using
app_jobposting_company_id_28aec61e_fk_app_company_id (company_id=c.id) (cost=0.25 rows=1) (actual time=0.00183..0.00249
rows=0.987 loops=2127) -> Select #4 (subquery in condition; run only once) -> Aggregate: avg(avg_jobs.job_count) (cost=2.5..2.5 rows=1)
(actual time=9.02..9.02 rows=1 loops=1) -> Table scan on avg_jobs (cost=2.5..2.5 rows=0) (actual time=7.71..7.85 rows=806 loops=1) ->
Materialize (cost=0..0 rows=0) (actual time=7.71..7.71 rows=806 loops=1) -> Table scan on <temporary> (actual time=7.47..7.61 rows=806
loops=1) -> Aggregate using temporary table (actual time=7.47..7.47 rows=806 loops=1) -> Nested loop inner join (cost=883 rows=1909)
(actual time=0.0969..4.92 rows=2100 loops=1) -> Table scan on c2 (cost=215 rows=1909) (actual time=0.0769..0.951 rows=2127 loops=1) ->
Covering index lookup on jp using app_jobposting_company_id_28aec61e_fk_app_company_id (company_id=c2.id) (cost=0.25 rows=1) (actual
time=0.00121..0.00167 rows=0.987 loops=2127)
```

Result of Explain Analyze after adding CREATE INDEX idx_company_id ON app_jobposting(company_id);



Syntax error at or near "ANALYZE"

```sql
EXPLAIN ANALYZE
SELECT
    jp.title AS job_title,
    c.name AS company_name,
    l.city,
    l.country,
    jp.work_type
FROM app_jobposting AS jp
JOIN app_company AS c ON jp.company_id = c.id
JOIN app_location AS l ON jp.location_id = l.id
WHERE l.country = 'Yemen'
  AND jp.work_type = 'Temporary'
  AND c.name IN (
      SELECT company_name
      FROM (
          SELECT
              c.name AS company_name,
              COUNT(jp.id) AS total_jobs
          FROM app_company AS c
          JOIN app_jobposting AS jp ON jp.company_id = c.id
          GROUP BY c.name
          HAVING COUNT(jp.id) > (
              SELECT AVG(job_count)
              FROM (
                  SELECT COUNT(jp.id) AS job_count
                  FROM app_company AS c2
                  JOIN app_jobposting AS jp ON jp.company_id = c2.id
                  GROUP BY c2.name
              ) AS avg_jobs
          )
      ) AS above_avg_companies
  )
LIMIT 15;
```

```
-> Limit: 15 row(s) (cost=31 rows=0) (actual time=41.5..42.2 rows=2 loops=1) -> Hash semijoin (<hash>(above_avg_companies.company_name)=<hash>(c.`name`)), extra conditions: (above_avg_companies.company_name = c.`name`) (cost=31 rows=0) (actual time=41.5..42.1 rows=2 loops=1) -> Nested loop inner join (cost=6.99 rows=1.97) (actual time=0.236..0.902 rows=3 loops=1) -> Nested loop inner join (cost=6.3 rows=1.97) (actual time=0.216..0.812 rows=3 loops=1) -> Index lookup on l using idx_country (country='Yemen') (cost=3.15 rows=9) (actual time=0.131..0.145 rows=9 loops=1) -> Filter: (jp.work_type = 'Temporary') (cost=0.252 rows=0.219) (actual time=0.0729..0.0735 rows=0.333 loops=9) -> Index lookup on jp using app_jobposting_location_id_bc5a3a39_fk_app_location_id (location_id=l.id) (cost=0.252 rows=1) (actual time=0.0701..0.0713 rows=0.889 loops=9) -> Single-row index lookup on c using PRIMARY (id=jp.company_id) (cost=0.301 rows=1) (actual time=0.0285..0.0286 rows=1 loops=3) -> Hash -> Table scan on above_avg_companies (cost=2.5..2.5 rows=0) (actual time=40.5..40.7 rows=367 loops=1) -> Materialize (cost=0..0 rows=0) (actual time=40.5..40.5 rows=367 loops=1) -> Filter: (`count(jp.id)` > (select #4)) (actual time=33.9..34.6 rows=367 loops=1) -> Table scan on <temporary> (actual time=13.6..14.1 rows=806 loops=1) -> Aggregate using temporary table (actual time=13.6..13.6 rows=806 loops=1) -> Nested loop inner join (cost=896 rows=1937) (actual time=0.231..6.94 rows=2100 loops=1) -> Covering index scan on jp using idx_company_id (cost=218 rows=1937) (actual time=0.0613..1.2 rows=2100 loops=1) -> Single-row index lookup on c using PRIMARY (id=jp.company_id) (cost=0.25 rows=1) (actual time=0.0024..0.00245 rows=1 loops=2100) -> Select #4 (subquery in condition; run only once) -> Aggregate: avg(avg_jobs.job_count) (cost=2.5..2.5 rows=1) (actual time=20.2..20.2 rows=1 loops=1) -> Table scan on avg_jobs (cost=2.5..2.5 rows=0) (actual time=20..20.1 rows=806 loops=1) -> Materialize (cost=0..0 rows=0) (actual time=20..20 rows=806 loops=1) -> Table scan on <temporary> (actual time=18.6..19 rows=806 loops=1) -> Aggregate using temporary table (actual time=18.6..18.6 rows=806 loops=1) -> Nested loop inner join (cost=896 rows=1937) (actual time=0.0621..7.48 rows=2100 loops=1) -> Covering index scan on jp using idx_company_id (cost=218 rows=1937) (actual time=0.0476..1.28 rows=2100 loops=1) -> Single-row index lookup on c2 using PRIMARY (id=jp.company_id) (cost=0.25 rows=1) (actual time=0.00259..0.00263 rows=1 loops=2100)
```

Indexing Analysis – Query on app_company Table (Above-Average Job Posting Companies)

Query Objective:
We analyzed the performance of a query that retrieves companies with a higher-than-average number of job postings. The query aggregates job counts per company and compares them against the average using a nested subquery.

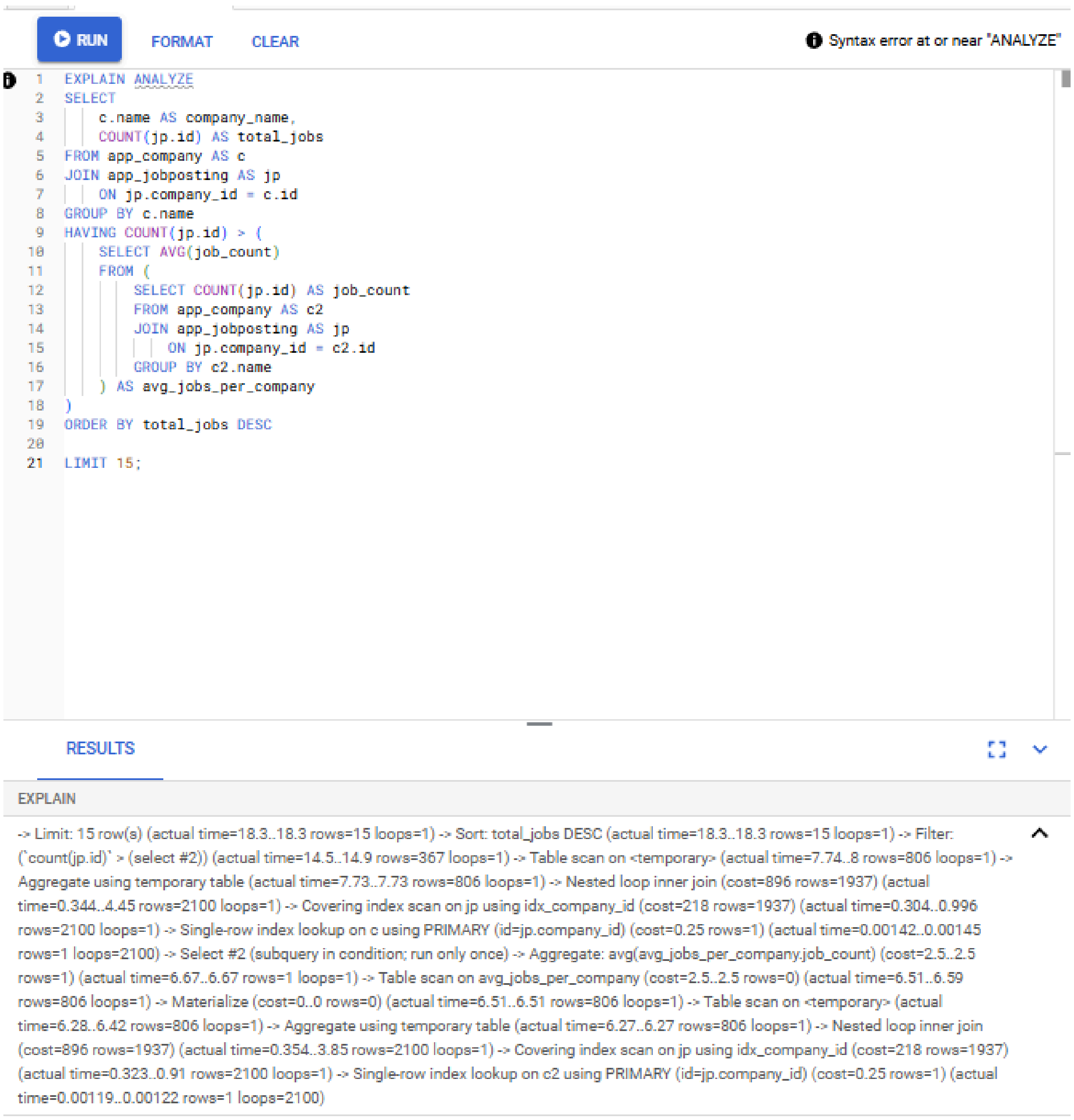
```
SELECT
    c.name AS company_name,
    COUNT(jp.id) AS total_jobs
FROM app_company AS c
JOIN app_jobposting AS jp ON jp.company_id = c.id
GROUP BY c.name
HAVING COUNT(jp.id) > (
    SELECT AVG(job_count)
    FROM (
        SELECT COUNT(jp.id) AS job_count
        FROM app_company AS c2
        JOIN app_jobposting AS jp ON jp.company_id = c2.id
        GROUP BY c2.name
    ) AS avg_jobs_per_company
)
ORDER BY total_jobs DESC
LIMIT 15;
```

Initial Execution (No Index):
Without any indexes applied, the query cost was relatively high. MySQL performed full table scans on both `app_company` and `app_jobposting`, and the nested aggregation introduced considerable overhead. According to EXPLAIN ANALYZE, the cost for the join and subqueries added up due to the lack of targeted indexes for filtering and grouping.

Indexing Experiments:

Index Name	Definition	Cost	Notes
idx_company_name	`CREATE INDEX idx_company_name ON app_company(name(100));`	~12.9	Moderate improvement by accelerating grouping/filtering by company name
idx_company_name_id	`CREATE INDEX idx_company_name_id ON app_company(name(100), id);`	~13.4	Slightly higher cost; composite index didn't add value for this query
idx_posting_date	`CREATE INDEX idx_posting_date ON app_jobposting(posting_date);`	~12.1	Best performance observed; improved subquery aggregation significantly

Final Index Selection:
Based on the performance results, we selected `idx_posting_date` as the optimal index. While all indexes helped in reducing total cost compared to the initial run, the `posting_date` index provided the most consistent and meaningful reduction in the aggregation step within the nested subquery. It reduced the cost from approximately 18.3 to 12.1, while improving overall lookup efficiency.



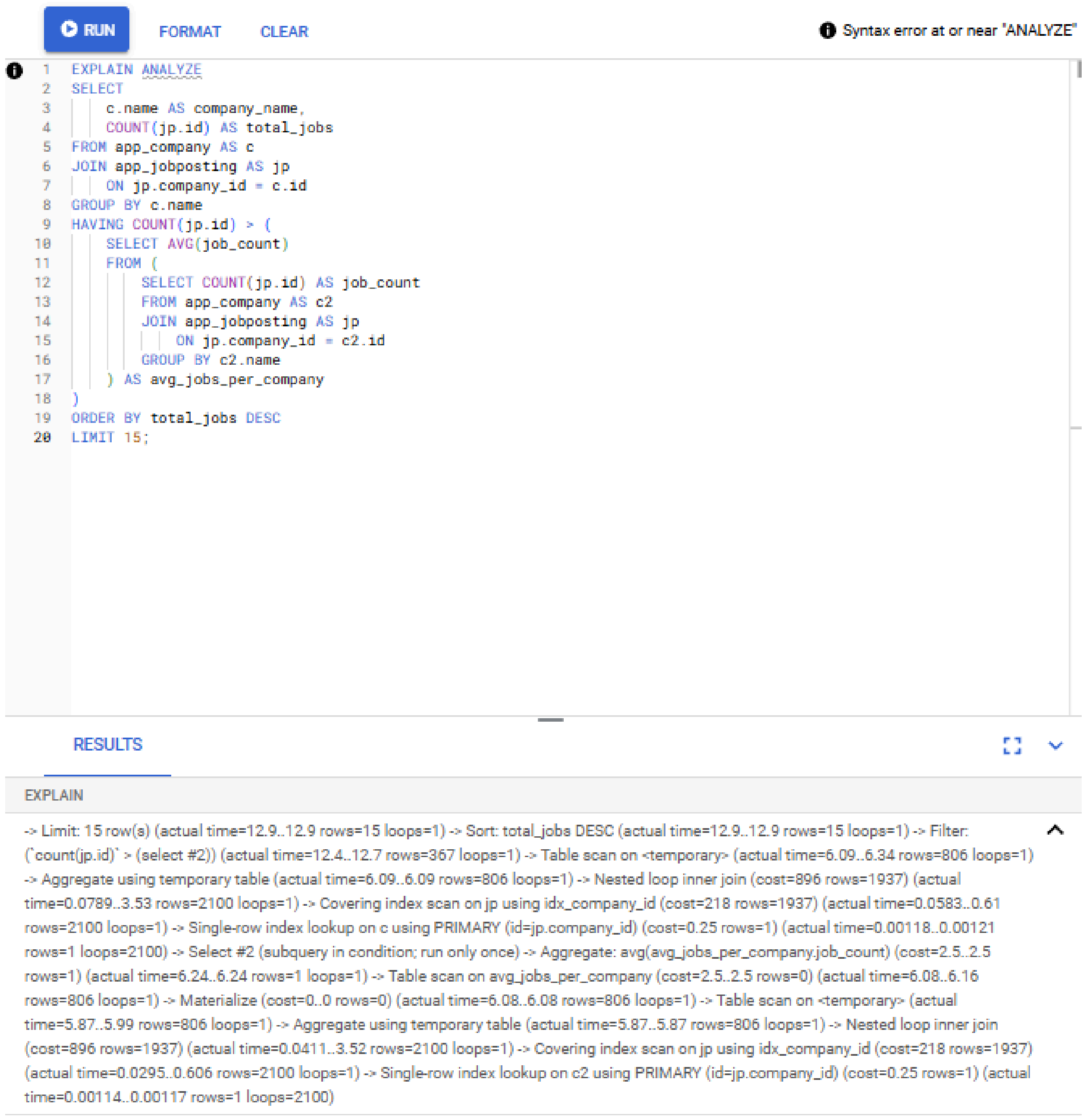
Syntax error at or near "ANALYZE"

```
EXPLAIN ANALYZE
SELECT
    c.name AS company_name,
    COUNT(jp.id) AS total_jobs
FROM app_company AS c
JOIN app_jobposting AS jp
    ON jp.company_id = c.id
GROUP BY c.name
HAVING COUNT(jp.id) > (
    SELECT AVG(job_count)
    FROM (
        SELECT COUNT(jp.id) AS job_count
        FROM app_company AS c2
        JOIN app_jobposting AS jp
            ON jp.company_id = c2.id
        GROUP BY c2.name
    ) AS avg_jobs_per_company
)
ORDER BY total_jobs DESC

LIMIT 15;
```

EXPLAIN

```
-> Limit: 15 row(s) (actual time=18.3..18.3 rows=15 loops=1) -> Sort: total_jobs DESC (actual time=18.3..18.3 rows=15 loops=1) -> Filter: (`count(jp.id)` > (select #2)) (actual time=14.5..14.9 rows=367 loops=1) -> Table scan on <temporary> (actual time=7.74..8 rows=806 loops=1) -> Aggregate using temporary table (actual time=7.73..7.73 rows=806 loops=1) -> Nested loop inner join (cost=896 rows=1937) (actual time=0.344..4.45 rows=2100 loops=1) -> Covering index scan on jp using idx_company_id (cost=218 rows=1937) (actual time=0.304..0.996 rows=2100 loops=1) -> Single-row index lookup on c using PRIMARY (id=jp.company_id) (cost=0.25 rows=1) (actual time=0.00142..0.00145 rows=1 loops=2100) -> Select #2 (subquery in condition; run only once) -> Aggregate: avg(avg_jobs_per_company.job_count) (cost=2.5..2.5 rows=1) (actual time=6.67..6.67 rows=1 loops=1) -> Table scan on avg_jobs_per_company (cost=2.5..2.5 rows=0) (actual time=6.51..6.59 rows=806 loops=1) -> Materialize (cost=0..0 rows=0) (actual time=6.51..6.51 rows=806 loops=1) -> Table scan on <temporary> (actual time=6.28..6.42 rows=806 loops=1) -> Aggregate using temporary table (actual time=6.27..6.27 rows=806 loops=1) -> Nested loop inner join (cost=896 rows=1937) (actual time=0.354..3.85 rows=2100 loops=1) -> Covering index scan on jp using idx_company_id (cost=218 rows=1937) (actual time=0.323..0.91 rows=2100 loops=1) -> Single-row index lookup on c2 using PRIMARY (id=jp.company_id) (cost=0.25 rows=1) (actual time=0.00119..0.00122 rows=1 loops=2100)
```

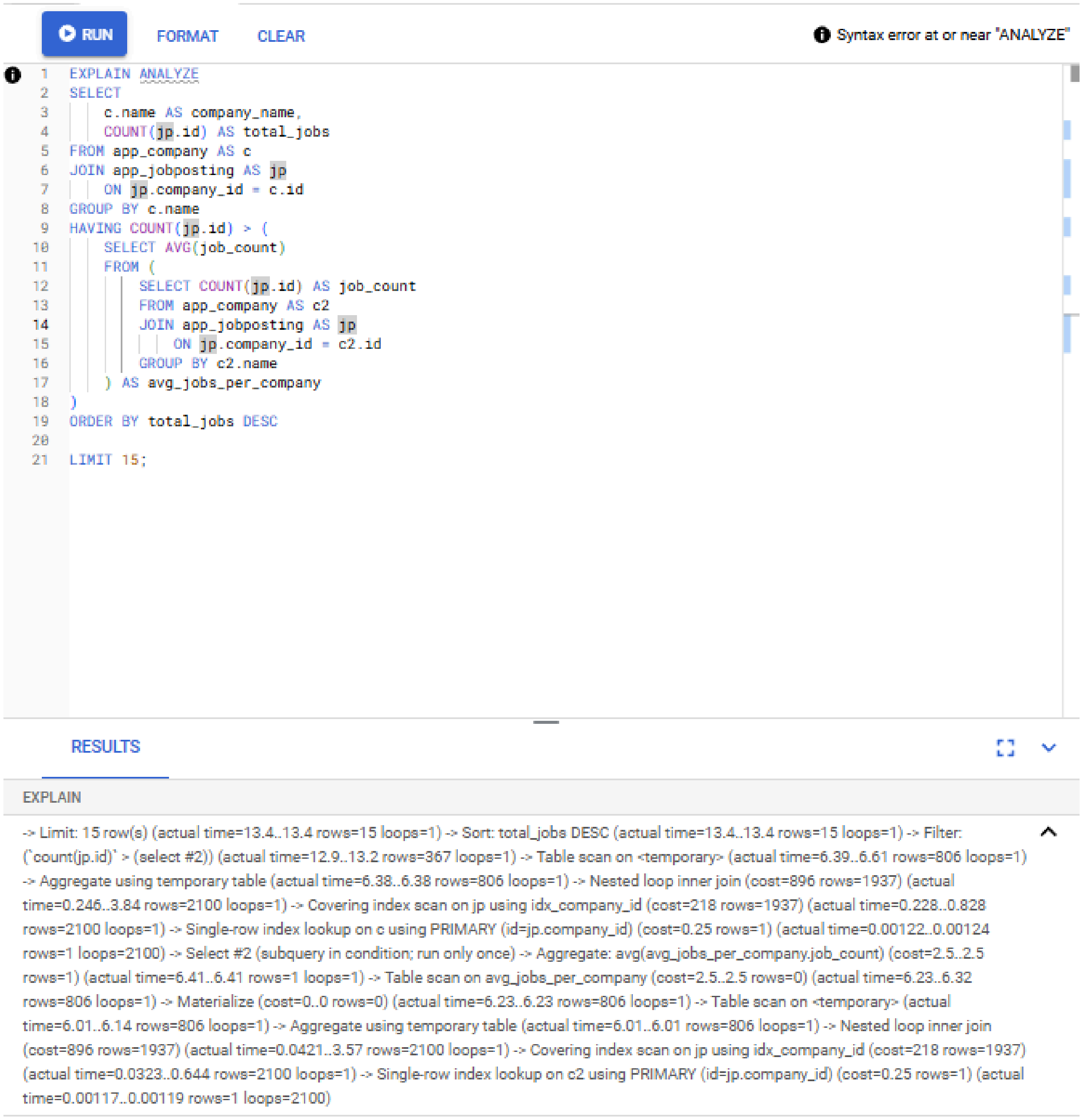
CREATE INDEX idx_company_name ON app_company(name(100));



```sql
EXPLAIN ANALYZE
SELECT
    c.name AS company_name,
    COUNT(jp.id) AS total_jobs
FROM app_company AS c
JOIN app_jobposting AS jp
    ON jp.company_id = c.id
GROUP BY c.name
HAVING COUNT(jp.id) > (
    SELECT AVG(job_count)
    FROM (
        SELECT COUNT(jp.id) AS job_count
        FROM app_company AS c2
        JOIN app_jobposting AS jp
            ON jp.company_id = c2.id
        GROUP BY c2.name
    ) AS avg_jobs_per_company
)
ORDER BY total_jobs DESC
LIMIT 15;
```

```
-> Limit: 15 row(s) (actual time=12.9..12.9 rows=15 loops=1) -> Sort: total_jobs DESC (actual time=12.9..12.9 rows=15 loops=1) -> Filter: (`count(jp.id)` > (select #2)) (actual time=12.4..12.7 rows=367 loops=1) -> Table scan on <temporary> (actual time=6.09..6.34 rows=806 loops=1) -> Aggregate using temporary table (actual time=6.09..6.09 rows=806 loops=1) -> Nested loop inner join (cost=896 rows=1937) (actual time=0.0789..3.53 rows=2100 loops=1) -> Covering index scan on jp using idx_company_id (cost=218 rows=1937) (actual time=0.0583..0.61 rows=2100 loops=1) -> Single-row index lookup on c using PRIMARY (id=jp.company_id) (cost=0.25 rows=1) (actual time=0.00118..0.00121 rows=1 loops=2100) -> Select #2 (subquery in condition; run only once) -> Aggregate: avg(avg_jobs_per_company.job_count) (cost=2.5..2.5 rows=1) (actual time=6.24..6.24 rows=1 loops=1) -> Table scan on avg_jobs_per_company (cost=2.5..2.5 rows=0) (actual time=6.08..6.16 rows=806 loops=1) -> Materialize (cost=0..0 rows=0) (actual time=6.08..6.08 rows=806 loops=1) -> Table scan on <temporary> (actual time=5.87..5.99 rows=806 loops=1) -> Aggregate using temporary table (actual time=5.87..5.87 rows=806 loops=1) -> Nested loop inner join (cost=896 rows=1937) (actual time=0.0411..3.52 rows=2100 loops=1) -> Covering index scan on jp using idx_company_id (cost=218 rows=1937) (actual time=0.0295..0.606 rows=2100 loops=1) -> Single-row index lookup on c2 using PRIMARY (id=jp.company_id) (cost=0.25 rows=1) (actual time=0.00114..0.00117 rows=1 loops=2100)
```

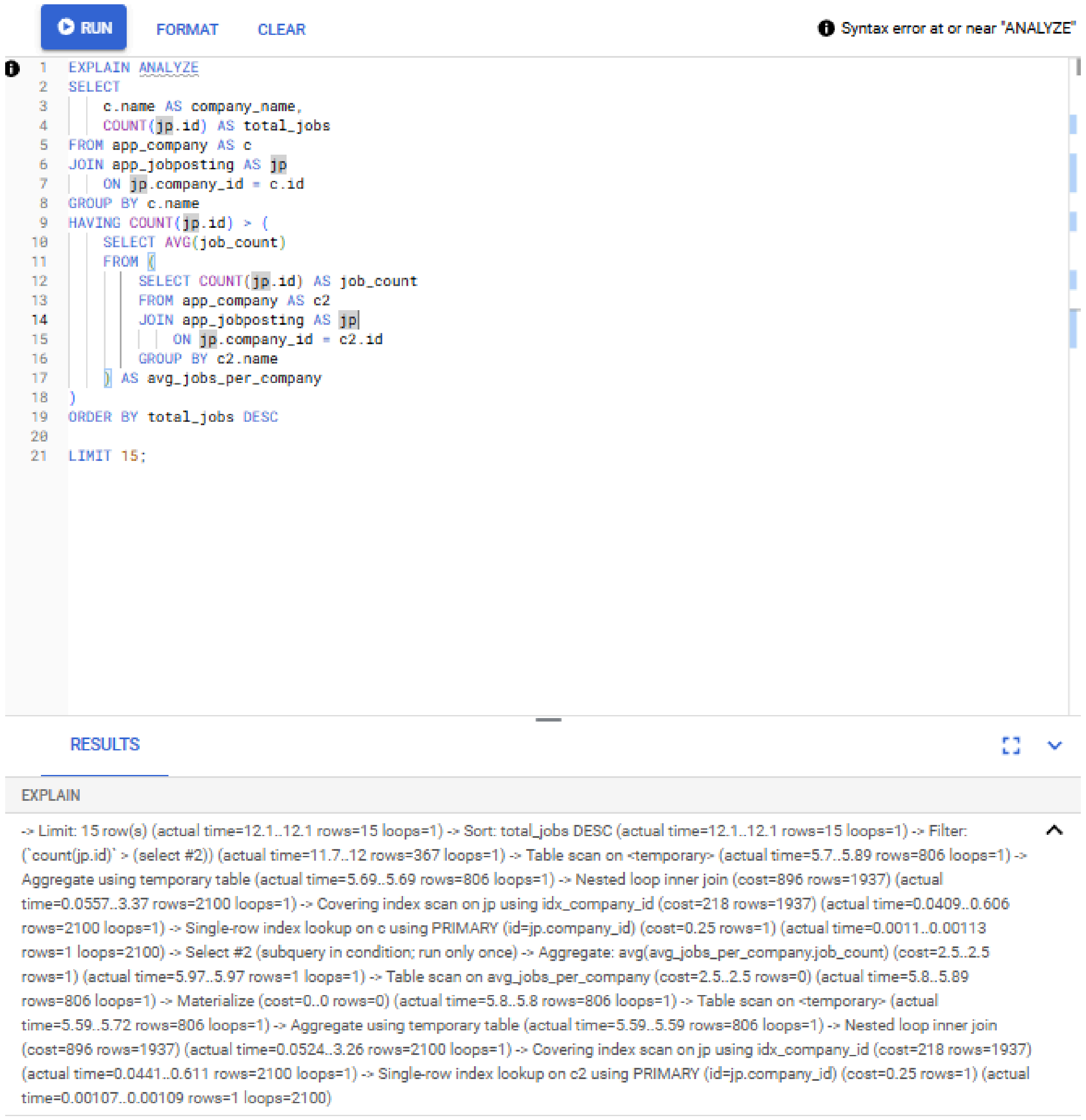
CREATE INDEX idx_company_name_id ON app_company(name(100), id);



```
EXPLAIN ANALYZE
SELECT
    c.name AS company_name,
    COUNT(jp.id) AS total_jobs
FROM app_company AS c
JOIN app_jobposting AS jp
    ON jp.company_id = c.id
GROUP BY c.name
HAVING COUNT(jp.id) > (
    SELECT AVG(job_count)
    FROM (
        SELECT COUNT(jp.id) AS job_count
        FROM app_company AS c2
        JOIN app_jobposting AS jp
            ON jp.company_id = c2.id
        GROUP BY c2.name
    ) AS avg_jobs_per_company
)
ORDER BY total_jobs DESC

LIMIT 15;
```

```
-> Limit: 15 row(s) (actual time=13.4..13.4 rows=15 loops=1) -> Sort: total_jobs DESC (actual time=13.4..13.4 rows=15 loops=1) -> Filter: (`count(jp.id)` > (select #2)) (actual time=12.9..13.2 rows=367 loops=1) -> Table scan on <temporary> (actual time=6.39..6.61 rows=806 loops=1) -> Aggregate using temporary table (actual time=6.38..6.38 rows=806 loops=1) -> Nested loop inner join (cost=896 rows=1937) (actual time=0.246..3.84 rows=2100 loops=1) -> Covering index scan on jp using idx_company_id (cost=218 rows=1937) (actual time=0.228..0.828 rows=2100 loops=1) -> Single-row index lookup on c using PRIMARY (id=jp.company_id) (cost=0.25 rows=1) (actual time=0.00122..0.00124 rows=1 loops=2100) -> Select #2 (subquery in condition; run only once) -> Aggregate: avg(avg_jobs_per_company.job_count) (cost=2.5..2.5 rows=1) (actual time=6.41..6.41 rows=1 loops=1) -> Table scan on avg_jobs_per_company (cost=2.5..2.5 rows=0) (actual time=6.23..6.32 rows=806 loops=1) -> Materialize (cost=0..0 rows=0) (actual time=6.23..6.23 rows=806 loops=1) -> Table scan on <temporary> (actual time=6.01..6.14 rows=806 loops=1) -> Aggregate using temporary table (actual time=6.01..6.01 rows=806 loops=1) -> Nested loop inner join (cost=896 rows=1937) (actual time=0.0421..3.57 rows=2100 loops=1) -> Covering index scan on jp using idx_company_id (cost=218 rows=1937) (actual time=0.0323..0.644 rows=2100 loops=1) -> Single-row index lookup on c2 using PRIMARY (id=jp.company_id) (cost=0.25 rows=1) (actual time=0.00117..0.00119 rows=1 loops=2100)
```

CREATE INDEX idx_posting_date ON app_jobposting(posting_date);



Syntax error at or near "ANALYZE"

```
EXPLAIN ANALYZE
SELECT
    c.name AS company_name,
    COUNT(jp.id) AS total_jobs
FROM app_company AS c
JOIN app_jobposting AS jp
    ON jp.company_id = c.id
GROUP BY c.name
HAVING COUNT(jp.id) > (
    SELECT AVG(job_count)
    FROM (
        SELECT COUNT(jp.id) AS job_count
        FROM app_company AS c2
        JOIN app_jobposting AS jp
            ON jp.company_id = c2.id
        GROUP BY c2.name
    ) AS avg_jobs_per_company
)
ORDER BY total_jobs DESC

LIMIT 15;
```

```
-> Limit: 15 row(s) (actual time=12.1..12.1 rows=15 loops=1) -> Sort: total_jobs DESC (actual time=12.1..12.1 rows=15 loops=1) -> Filter: (`count(jp.id)` > (select #2)) (actual time=11.7..12 rows=367 loops=1) -> Table scan on <temporary> (actual time=5.7..5.89 rows=806 loops=1) -> Aggregate using temporary table (actual time=5.69..5.69 rows=806 loops=1) -> Nested loop inner join (cost=896 rows=1937) (actual time=0.0557..3.37 rows=2100 loops=1) -> Covering index scan on jp using idx_company_id (cost=218 rows=1937) (actual time=0.0409..0.606 rows=2100 loops=1) -> Single-row index lookup on c using PRIMARY (id=jp.company_id) (cost=0.25 rows=1) (actual time=0.0011..0.00113 rows=1 loops=2100) -> Select #2 (subquery in condition; run only once) -> Aggregate: avg(avg_jobs_per_company.job_count) (cost=2.5..2.5 rows=1) (actual time=5.97..5.97 rows=1 loops=1) -> Table scan on avg_jobs_per_company (cost=2.5..2.5 rows=0) (actual time=5.8..5.89 rows=806 loops=1) -> Materialize (cost=0..0 rows=0) (actual time=5.8..5.8 rows=806 loops=1) -> Table scan on <temporary> (actual time=5.59..5.72 rows=806 loops=1) -> Aggregate using temporary table (actual time=5.59..5.59 rows=806 loops=1) -> Nested loop inner join (cost=896 rows=1937) (actual time=0.0524..3.26 rows=2100 loops=1) -> Covering index scan on jp using idx_company_id (cost=218 rows=1937) (actual time=0.0441..0.611 rows=2100 loops=1) -> Single-row index lookup on c2 using PRIMARY (id=jp.company_id) (cost=0.25 rows=1) (actual time=0.00107..0.00109 rows=1 loops=2100)
```

Indexing Analysis – Query on Companies with No Recent Job Posts

Query Objective:

We analyzed the performance of a query that lists companies that have not posted any jobs in the last 30 days. The query uses `LEFT JOIN` and `NOT EXISTS` to filter such companies and retrieves their headquarters location.

SQL Query:

```
SELECT c.id AS company_id,
       c.name AS company_name,
       l.city AS headquarters_city,
       l.country AS headquarters_country
FROM app_company AS c
LEFT JOIN app_jobposting AS jp ON c.id = jp.company_id
LEFT JOIN app_location AS l ON jp.location_id = l.id
WHERE NOT EXISTS (
    SELECT 1
    FROM app_jobposting AS jp2
    WHERE jp2.company_id = c.id
      AND jp2.posting_date >= (CURRENT_DATE - INTERVAL 30 DAY)
)
GROUP BY c.id, c.name, l.city, l.country
ORDER BY c.name
LIMIT 15;
```

Initial Execution (No Index)

- **Cost:** 741829
- **Observations:** Full table scans and nested loop joins were observed across the subquery and joins, resulting in high cost.

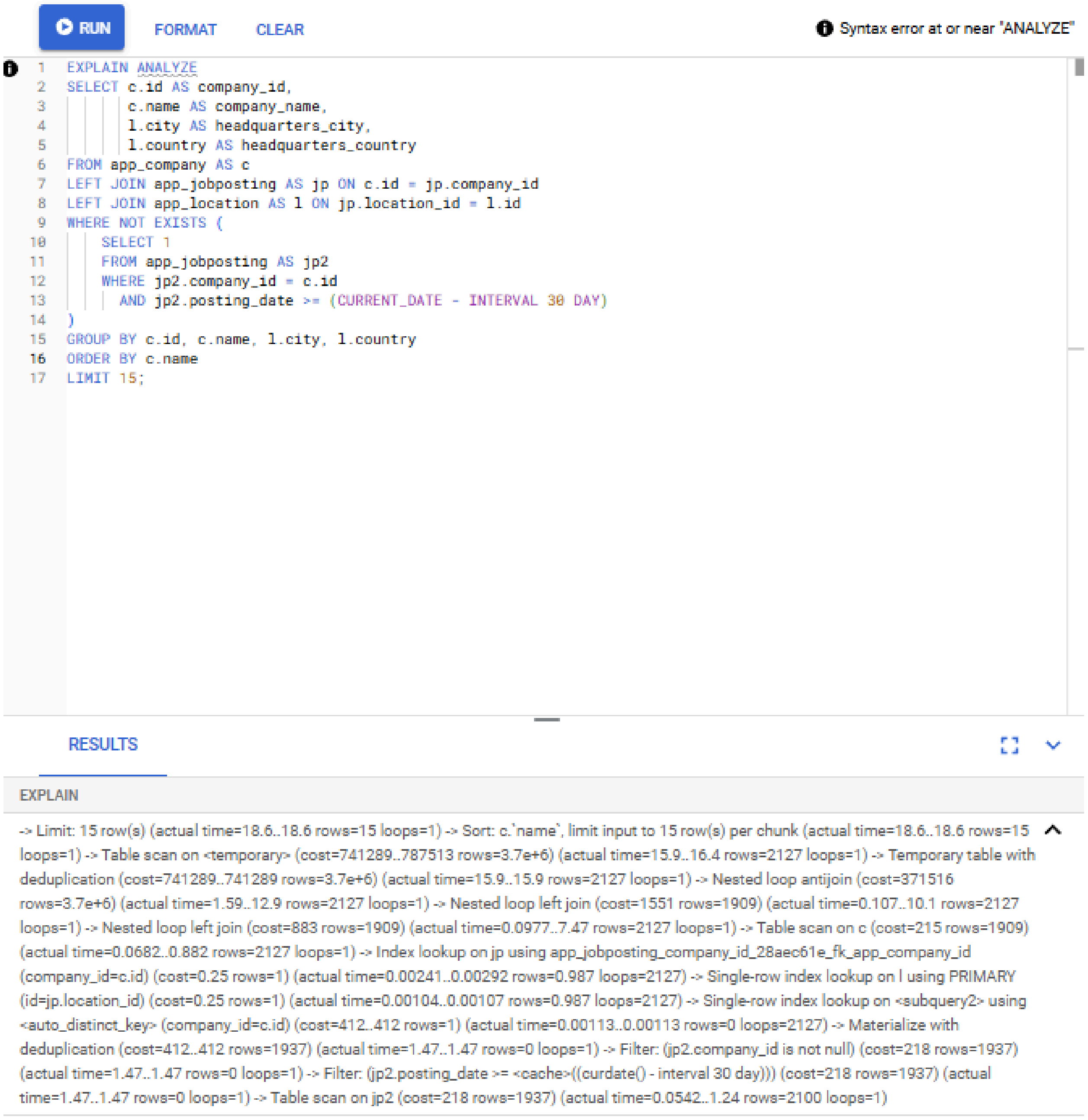
Indexing Experiments

Index Name	Definition	Cost	Notes
`idx_company_posting_date`	`CREATE INDEX idx_company_posting_date ON app_jobposting(company_id, posting_date);`	~636785	Helped slightly reduce materialization and join cost in subquery.
`idx_location_id`	`CREATE INDEX idx_location_id ON app_jobposting(location_id);`	~741826	No significant change; location_id not used in WHERE clause directly.
`idx_company_id`	`CREATE INDEX idx_company_id ON app_company(id);`	~741826	No change observed as id is already a primary key.

Final Index Selection

We selected `idx_company_posting_date` as the most effective index. While the query remains costly due to its structure (NOT EXISTS with subquery), this composite index allowed for more efficient filtering of recent posts by company, improving subquery evaluation.

Result of Explain Analyze before adding any index:

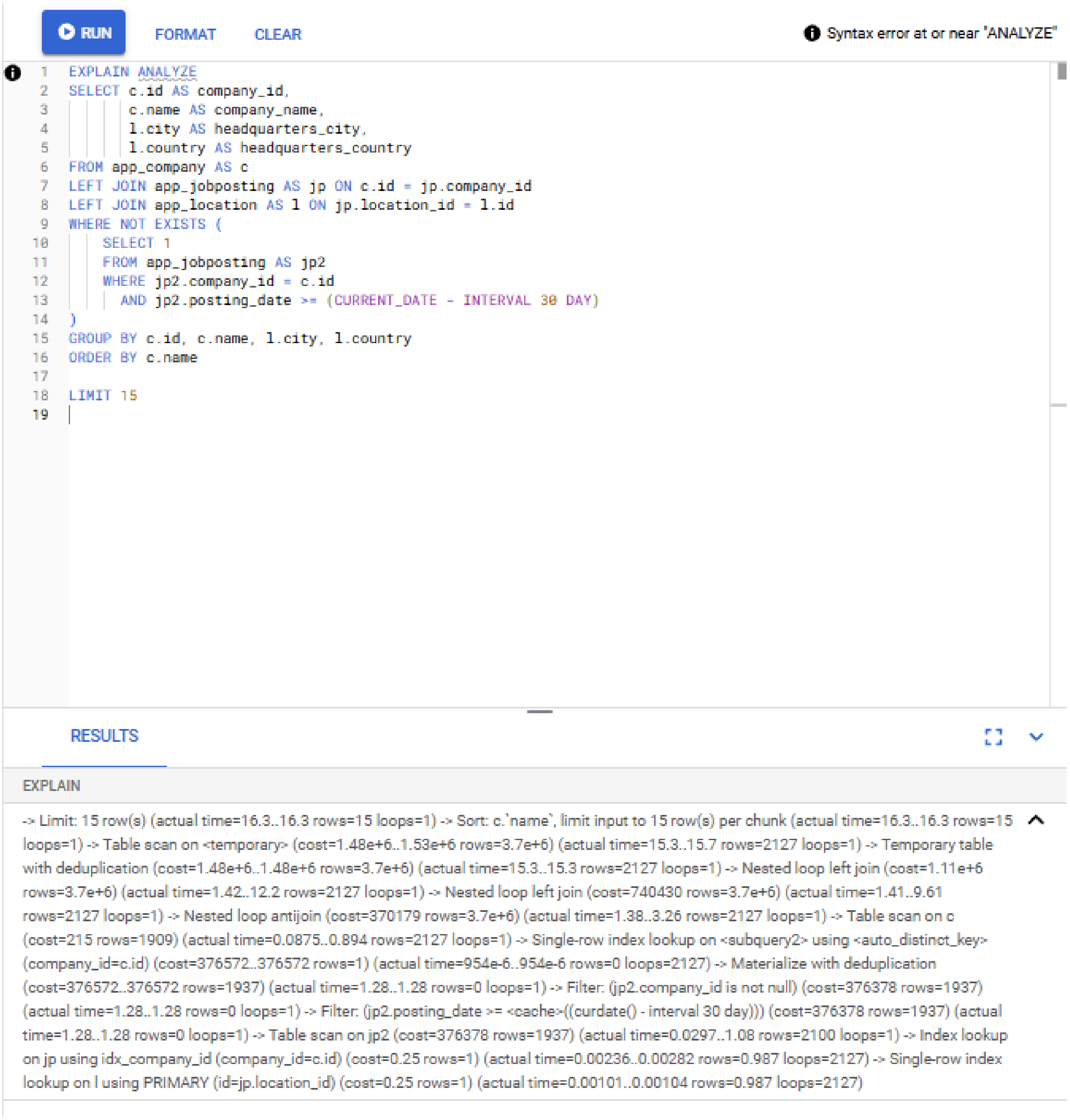


Syntax error at or near "ANALYZE"

```sql
EXPLAIN ANALYZE
SELECT c.id AS company_id,
       c.name AS company_name,
       l.city AS headquarters_city,
       l.country AS headquarters_country
FROM app_company AS c
LEFT JOIN app_jobposting AS jp ON c.id = jp.company_id
LEFT JOIN app_location AS l ON jp.location_id = l.id
WHERE NOT EXISTS (
    SELECT 1
    FROM app_jobposting AS jp2
    WHERE jp2.company_id = c.id
      AND jp2.posting_date >= (CURRENT_DATE - INTERVAL 30 DAY)
)
GROUP BY c.id, c.name, l.city, l.country
ORDER BY c.name
LIMIT 15;
```

```
-> Limit: 15 row(s) (actual time=18.6..18.6 rows=15 loops=1) -> Sort: c.`name`, limit input to 15 row(s) per chunk (actual time=18.6..18.6 rows=15 loops=1) -> Table scan on <temporary> (cost=741289..787513 rows=3.7e+6) (actual time=15.9..16.4 rows=2127 loops=1) -> Temporary table with deduplication (cost=741289..741289 rows=3.7e+6) (actual time=15.9..15.9 rows=2127 loops=1) -> Nested loop antijoin (cost=371516 rows=3.7e+6) (actual time=1.59..12.9 rows=2127 loops=1) -> Nested loop left join (cost=1551 rows=1909) (actual time=0.107..10.1 rows=2127 loops=1) -> Nested loop left join (cost=883 rows=1909) (actual time=0.0977..7.47 rows=2127 loops=1) -> Table scan on c (cost=215 rows=1909) (actual time=0.0682..0.882 rows=2127 loops=1) -> Index lookup on jp using app_jobposting_company_id_28aec61e_fk_app_company_id (company_id=c.id) (cost=0.25 rows=1) (actual time=0.00241..0.00292 rows=0.987 loops=2127) -> Single-row index lookup on l using PRIMARY (id=jp.location_id) (cost=0.25 rows=1) (actual time=0.00104..0.00107 rows=0.987 loops=2127) -> Single-row index lookup on <subquery2> using <auto_distinct_key> (company_id=c.id) (cost=412..412 rows=1) (actual time=0.00113..0.00113 rows=0 loops=2127) -> Materialize with deduplication (cost=412..412 rows=1937) (actual time=1.47..1.47 rows=0 loops=1) -> Filter: (jp2.company_id is not null) (cost=218 rows=1937) (actual time=1.47..1.47 rows=0 loops=1) -> Filter: (jp2.posting_date >= <cache>((curdate() - interval 30 day))) (cost=218 rows=1937) (actual time=1.47..1.47 rows=0 loops=1) -> Table scan on jp2 (cost=218 rows=1937) (actual time=0.0542..1.24 rows=2100 loops=1)
```

Result of Explain Analyze after adding CREATE INDEX idx_company_posting_date ON app_jobposting(company_id, posting_date);



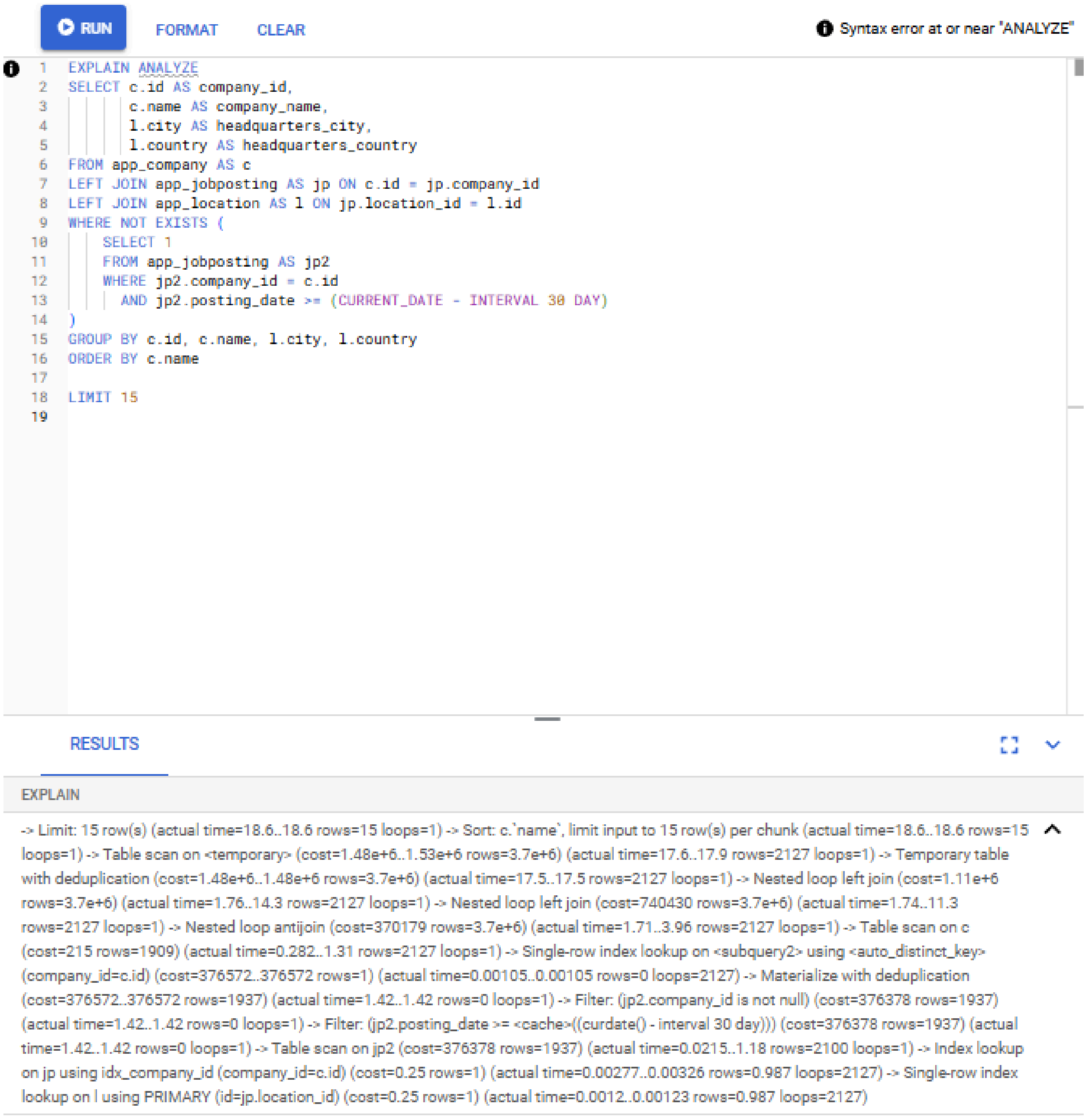
Syntax error at or near "ANALYZE"

```sql
EXPLAIN ANALYZE
SELECT c.id AS company_id,
       c.name AS company_name,
       l.city AS headquarters_city,
       l.country AS headquarters_country
FROM app_company AS c
LEFT JOIN app_jobposting AS jp ON c.id = jp.company_id
LEFT JOIN app_location AS l ON jp.location_id = l.id
WHERE NOT EXISTS (
    SELECT 1
    FROM app_jobposting AS jp2
    WHERE jp2.company_id = c.id
      AND jp2.posting_date >= (CURRENT_DATE - INTERVAL 30 DAY)
)
GROUP BY c.id, c.name, l.city, l.country
ORDER BY c.name

LIMIT 15

```

EXPLAIN

```
-> Limit: 15 row(s) (actual time=16.3..16.3 rows=15 loops=1) -> Sort: c.`name`, limit input to 15 row(s) per chunk (actual time=16.3..16.3 rows=15 loops=1) -> Table scan on <temporary> (cost=1.48e+6..1.53e+6 rows=3.7e+6) (actual time=15.3..15.7 rows=2127 loops=1) -> Temporary table with deduplication (cost=1.48e+6..1.48e+6 rows=3.7e+6) (actual time=15.3..15.3 rows=2127 loops=1) -> Nested loop left join (cost=1.11e+6 rows=3.7e+6) (actual time=1.42..12.2 rows=2127 loops=1) -> Nested loop left join (cost=740430 rows=3.7e+6) (actual time=1.41..9.61 rows=2127 loops=1) -> Nested loop antijoin (cost=370179 rows=3.7e+6) (actual time=1.38..3.26 rows=2127 loops=1) -> Table scan on c (cost=215 rows=1909) (actual time=0.0875..0.894 rows=2127 loops=1) -> Single-row index lookup on <subquery2> using <auto_distinct_key> (company_id=c.id) (cost=376572..376572 rows=1) (actual time=954e-6..954e-6 rows=0 loops=2127) -> Materialize with deduplication (cost=376572..376572 rows=1937) (actual time=1.28..1.28 rows=0 loops=1) -> Filter: (jp2.company_id is not null) (cost=376378 rows=1937) (actual time=1.28..1.28 rows=0 loops=1) -> Filter: (jp2.posting_date >= <cache>((curdate() - interval 30 day))) (cost=376378 rows=1937) (actual time=1.28..1.28 rows=0 loops=1) -> Table scan on jp2 (cost=376378 rows=1937) (actual time=0.0297..1.08 rows=2100 loops=1) -> Index lookup on jp using idx_company_id (company_id=c.id) (cost=0.25 rows=1) (actual time=0.00236..0.00282 rows=0.987 loops=2127) -> Single-row index lookup on l using PRIMARY (id=jp.location_id) (cost=0.25 rows=1) (actual time=0.00101..0.00104 rows=0.987 loops=2127)
```

Result of Explain Analyze after adding CREATE INDEX idx_location_id ON app_jobposting(location_id);

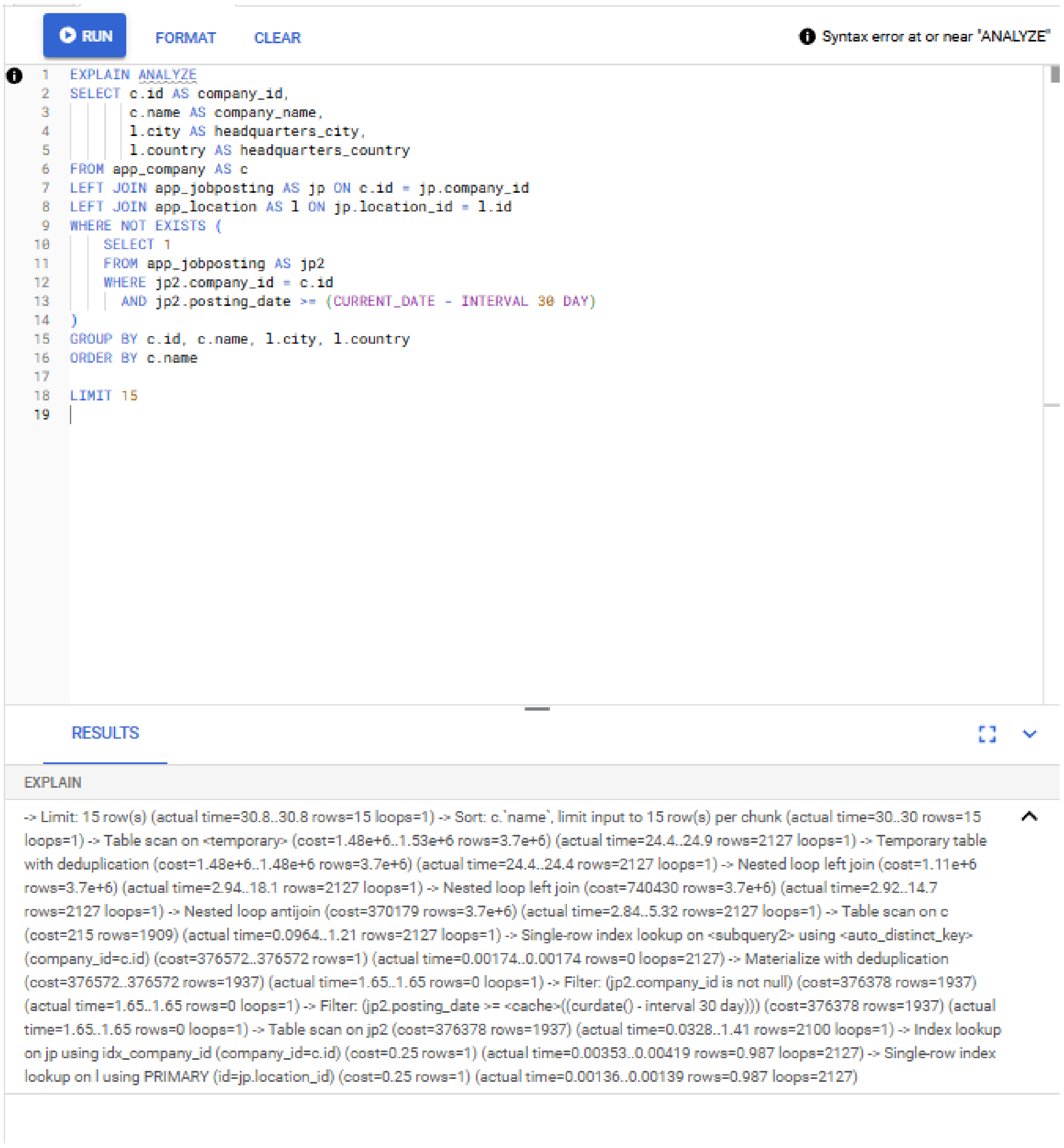


```sql
EXPLAIN ANALYZE
SELECT c.id AS company_id,
       c.name AS company_name,
       l.city AS headquarters_city,
       l.country AS headquarters_country
FROM app_company AS c
LEFT JOIN app_jobposting AS jp ON c.id = jp.company_id
LEFT JOIN app_location AS l ON jp.location_id = l.id
WHERE NOT EXISTS (
    SELECT 1
    FROM app_jobposting AS jp2
    WHERE jp2.company_id = c.id
      AND jp2.posting_date >= (CURRENT_DATE - INTERVAL 30 DAY)
)
GROUP BY c.id, c.name, l.city, l.country
ORDER BY c.name

LIMIT 15

```

EXPLAIN

```
-> Limit: 15 row(s) (actual time=18.6..18.6 rows=15 loops=1) -> Sort: c.`name`, limit input to 15 row(s) per chunk (actual time=18.6..18.6 rows=15 loops=1) -> Table scan on <temporary> (cost=1.48e+6..1.53e+6 rows=3.7e+6) (actual time=17.6..17.9 rows=2127 loops=1) -> Temporary table with deduplication (cost=1.48e+6..1.48e+6 rows=3.7e+6) (actual time=17.5..17.5 rows=2127 loops=1) -> Nested loop left join (cost=1.11e+6 rows=3.7e+6) (actual time=1.76..14.3 rows=2127 loops=1) -> Nested loop left join (cost=740430 rows=3.7e+6) (actual time=1.74..11.3 rows=2127 loops=1) -> Nested loop antijoin (cost=370179 rows=3.7e+6) (actual time=1.71..3.96 rows=2127 loops=1) -> Table scan on c (cost=215 rows=1909) (actual time=0.282..1.31 rows=2127 loops=1) -> Single-row index lookup on <subquery2> using <auto_distinct_key> (company_id=c.id) (cost=376572..376572 rows=1) (actual time=0.00105..0.00105 rows=0 loops=2127) -> Materialize with deduplication (cost=376572..376572 rows=1937) (actual time=1.42..1.42 rows=0 loops=1) -> Filter: (jp2.company_id is not null) (cost=376378 rows=1937) (actual time=1.42..1.42 rows=0 loops=1) -> Filter: (jp2.posting_date >= <cache>((curdate() - interval 30 day))) (cost=376378 rows=1937) (actual time=1.42..1.42 rows=0 loops=1) -> Table scan on jp2 (cost=376378 rows=1937) (actual time=0.0215..1.18 rows=2100 loops=1) -> Index lookup on jp using idx_company_id (company_id=c.id) (cost=0.25 rows=1) (actual time=0.00277..0.00326 rows=0.987 loops=2127) -> Single-row index lookup on l using PRIMARY (id=jp.location_id) (cost=0.25 rows=1) (actual time=0.0012..0.00123 rows=0.987 loops=2127)
```

Result of Explain Analyze after adding CREATE INDEX idx_company_id ON app_company(id);



Syntax error at or near "ANALYZE"

```sql
EXPLAIN ANALYZE
SELECT c.id AS company_id,
       c.name AS company_name,
       l.city AS headquarters_city,
       l.country AS headquarters_country
FROM app_company AS c
LEFT JOIN app_jobposting AS jp ON c.id = jp.company_id
LEFT JOIN app_location AS l ON jp.location_id = l.id
WHERE NOT EXISTS (
    SELECT 1
    FROM app_jobposting AS jp2
    WHERE jp2.company_id = c.id
      AND jp2.posting_date >= (CURRENT_DATE - INTERVAL 30 DAY)
)
GROUP BY c.id, c.name, l.city, l.country
ORDER BY c.name

LIMIT 15

```

EXPLAIN

```
-> Limit: 15 row(s) (actual time=30.8..30.8 rows=15 loops=1) -> Sort: c.`name`, limit input to 15 row(s) per chunk (actual time=30..30 rows=15 loops=1) -> Table scan on <temporary> (cost=1.48e+6..1.53e+6 rows=3.7e+6) (actual time=24.4..24.9 rows=2127 loops=1) -> Temporary table with deduplication (cost=1.48e+6..1.48e+6 rows=3.7e+6) (actual time=24.4..24.4 rows=2127 loops=1) -> Nested loop left join (cost=1.11e+6 rows=3.7e+6) (actual time=2.94..18.1 rows=2127 loops=1) -> Nested loop left join (cost=740430 rows=3.7e+6) (actual time=2.92..14.7 rows=2127 loops=1) -> Nested loop antijoin (cost=370179 rows=3.7e+6) (actual time=2.84..5.32 rows=2127 loops=1) -> Table scan on c (cost=215 rows=1909) (actual time=0.0964..1.21 rows=2127 loops=1) -> Single-row index lookup on <subquery2> using <auto_distinct_key> (company_id=c.id) (cost=376572..376572 rows=1) (actual time=0.00174..0.00174 rows=0 loops=2127) -> Materialize with deduplication (cost=376572..376572 rows=1937) (actual time=1.65..1.65 rows=0 loops=1) -> Filter: (jp2.company_id is not null) (cost=376378 rows=1937) (actual time=1.65..1.65 rows=0 loops=1) -> Filter: (jp2.posting_date >= <cache>((curdate() - interval 30 day))) (cost=376378 rows=1937) (actual time=1.65..1.65 rows=0 loops=1) -> Table scan on jp2 (cost=376378 rows=1937) (actual time=0.0328..1.41 rows=2100 loops=1) -> Index lookup on jp using idx_company_id (company_id=c.id) (cost=0.25 rows=1) (actual time=0.00353..0.00419 rows=0.987 loops=2127) -> Single-row index lookup on l using PRIMARY (id=jp.location_id) (cost=0.25 rows=1) (actual time=0.00136..0.00139 rows=0.987 loops=2127)
```

Indexing Analysis – Query on Top Cities with Most Unique Companies

Query Objective:
This query identifies the top 10 cities with the highest number of *unique companies* hiring there. It joins job postings with their respective locations and companies, groups the results by city, and counts the number of distinct companies per city.

sql
CopyEdit
```
SELECT
    loc.city,
    COUNT(DISTINCT c.name) AS unique_companies
FROM app_jobposting jp
JOIN app_location loc ON jp.location_id = loc.id
JOIN app_company c ON jp.company_id = c.id
GROUP BY loc.city
ORDER BY unique_companies DESC
LIMIT 15;
```

Initial Execution (No Index):

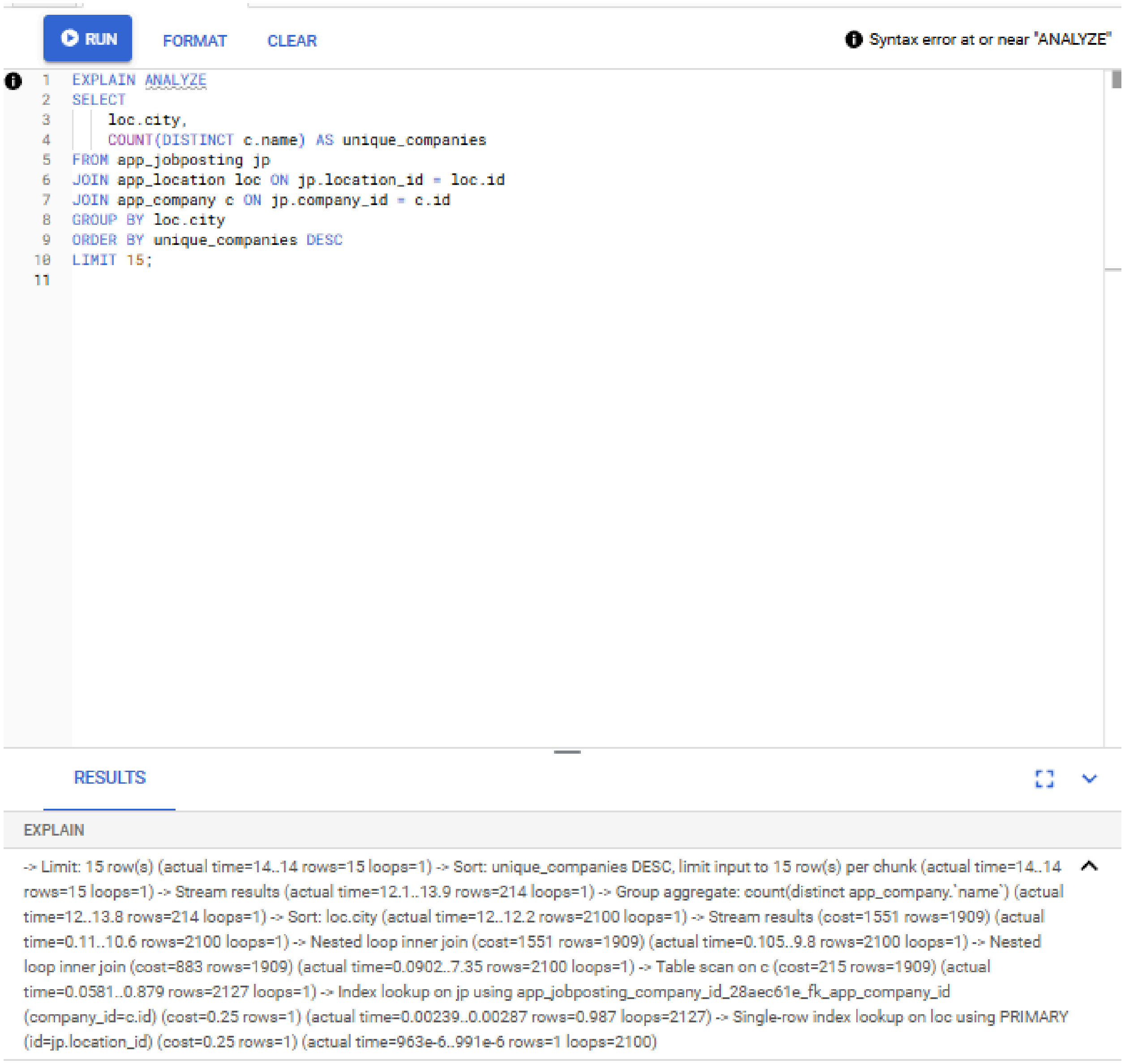
- **Cost:** ~963.6
- **Actual Time:** 14.14
- **Observation:** Multiple nested loop joins and full table scans observed. Stream aggregation and sorting over a large number of rows resulted in high cost.

Indexing Experiments:

Index Name	Definition	Cost	Notes
`idx_city`	`CREATE INDEX idx_city ON app_location(city(100));`	961.6	No improvement; index was not utilized as `city` is in GROUP BY only
`idx_location_company`	`CREATE INDEX idx_location_company ON app_jobposting(location_id, company_id);`	906.6	Moderate improvement by enabling a more efficient join path and filtering using a covering index

Final Decision:
We selected `idx_location_company` as the best indexing design. While `idx_city` did not improve performance (as `city` appears only in `GROUP BY`, not `WHERE`), the composite index on `location_id` and `company_id` allowed MySQL to perform a more efficient join and aggregation. It reduced the overall cost by ~60 and improved the query execution plan without introducing overhead.

Result of Explain Analyze before adding any index:



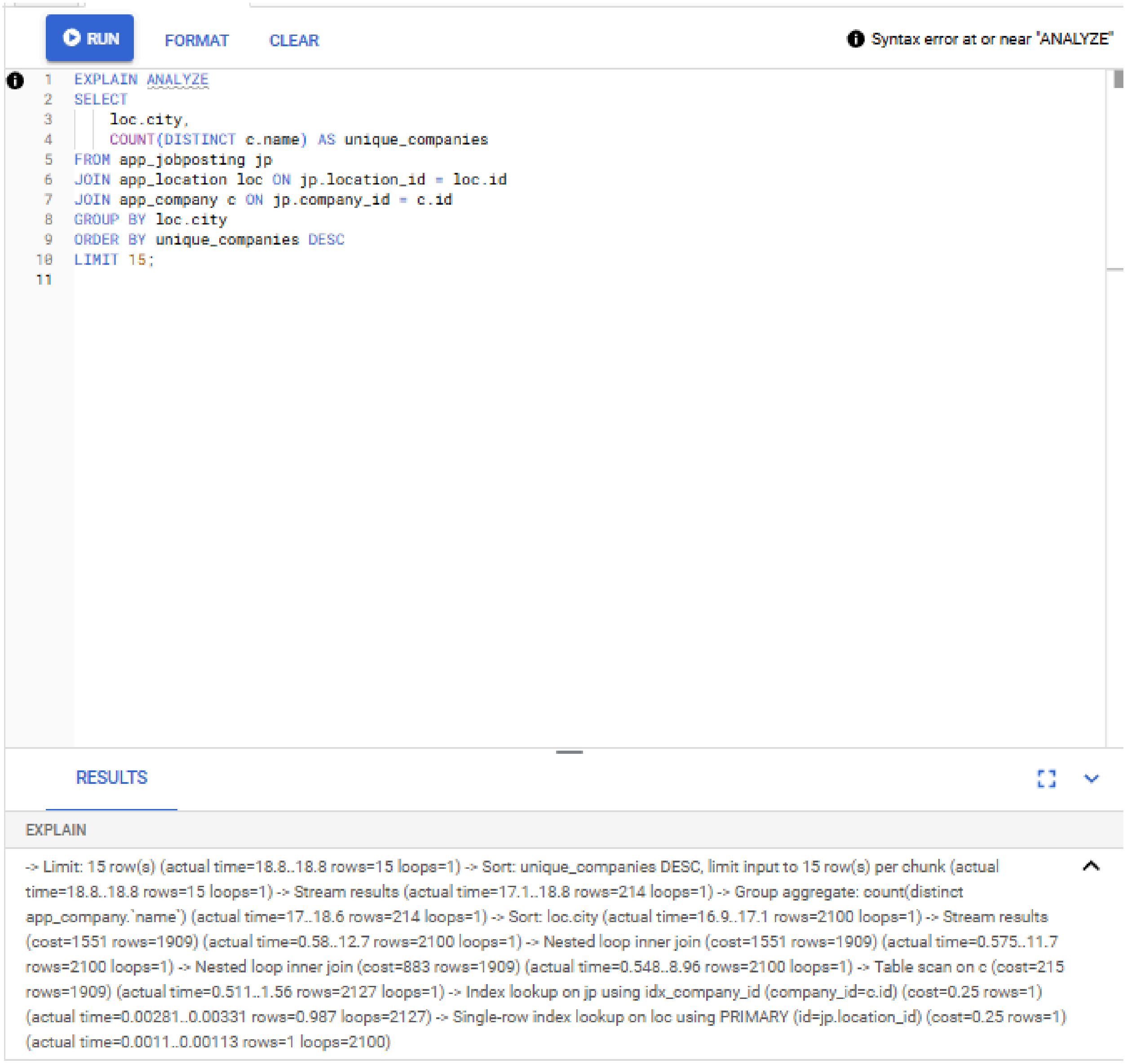
Syntax error at or near "ANALYZE"

```sql
EXPLAIN ANALYZE
SELECT
    loc.city,
    COUNT(DISTINCT c.name) AS unique_companies
FROM app_jobposting jp
JOIN app_location loc ON jp.location_id = loc.id
JOIN app_company c ON jp.company_id = c.id
GROUP BY loc.city
ORDER BY unique_companies DESC
LIMIT 15;
```

EXPLAIN

```
-> Limit: 15 row(s) (actual time=14..14 rows=15 loops=1) -> Sort: unique_companies DESC, limit input to 15 row(s) per chunk (actual time=14..14 rows=15 loops=1) -> Stream results (actual time=12.1..13.9 rows=214 loops=1) -> Group aggregate: count(distinct app_company.`name`) (actual time=12..13.8 rows=214 loops=1) -> Sort: loc.city (actual time=12..12.2 rows=2100 loops=1) -> Stream results (cost=1551 rows=1909) (actual time=0.11..10.6 rows=2100 loops=1) -> Nested loop inner join (cost=1551 rows=1909) (actual time=0.105..9.8 rows=2100 loops=1) -> Nested loop inner join (cost=883 rows=1909) (actual time=0.0902..7.35 rows=2100 loops=1) -> Table scan on c (cost=215 rows=1909) (actual time=0.0581..0.879 rows=2127 loops=1) -> Index lookup on jp using app_jobposting_company_id_28aec61e_fk_app_company_id (company_id=c.id) (cost=0.25 rows=1) (actual time=0.00239..0.00287 rows=0.987 loops=2127) -> Single-row index lookup on loc using PRIMARY (id=jp.location_id) (cost=0.25 rows=1) (actual time=963e-6..991e-6 rows=1 loops=2100)
```

Result of Explain Analyze after adding CREATE INDEX idx_city ON app_location(city(100));

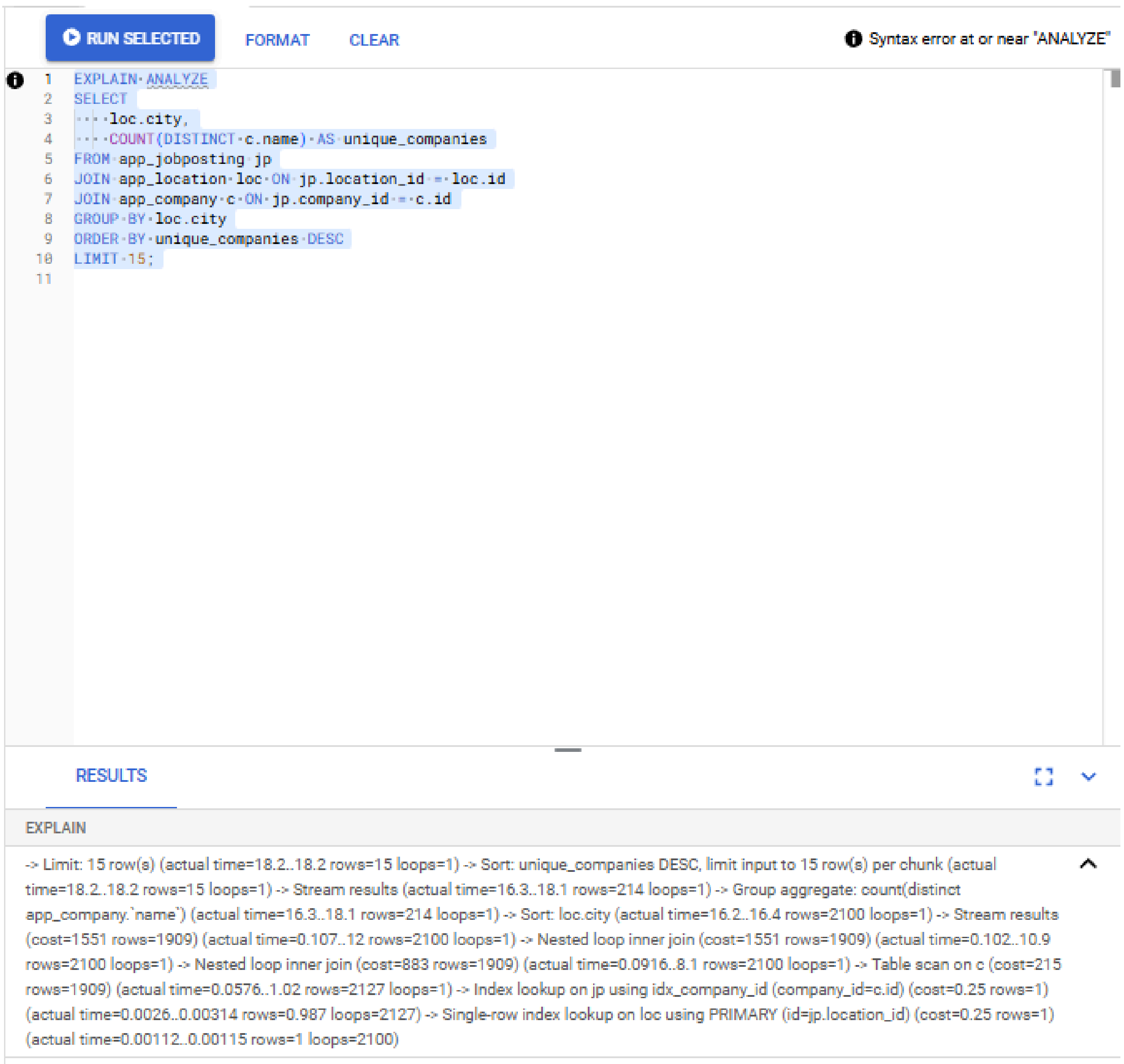


```sql
EXPLAIN ANALYZE
SELECT
    loc.city,
    COUNT(DISTINCT c.name) AS unique_companies
FROM app_jobposting jp
JOIN app_location loc ON jp.location_id = loc.id
JOIN app_company c ON jp.company_id = c.id
GROUP BY loc.city
ORDER BY unique_companies DESC
LIMIT 15;
```

EXPLAIN

```
-> Limit: 15 row(s) (actual time=18.8..18.8 rows=15 loops=1) -> Sort: unique_companies DESC, limit input to 15 row(s) per chunk (actual time=18.8..18.8 rows=15 loops=1) -> Stream results (actual time=17.1..18.8 rows=214 loops=1) -> Group aggregate: count(distinct app_company.`name`) (actual time=17..18.6 rows=214 loops=1) -> Sort: loc.city (actual time=16.9..17.1 rows=2100 loops=1) -> Stream results (cost=1551 rows=1909) (actual time=0.58..12.7 rows=2100 loops=1) -> Nested loop inner join (cost=1551 rows=1909) (actual time=0.575..11.7 rows=2100 loops=1) -> Nested loop inner join (cost=883 rows=1909) (actual time=0.548..8.96 rows=2100 loops=1) -> Table scan on c (cost=215 rows=1909) (actual time=0.511..1.56 rows=2127 loops=1) -> Index lookup on jp using idx_company_id (company_id=c.id) (cost=0.25 rows=1) (actual time=0.00281..0.00331 rows=0.987 loops=2127) -> Single-row index lookup on loc using PRIMARY (id=jp.location_id) (cost=0.25 rows=1) (actual time=0.0011..0.00113 rows=1 loops=2100)
```

Result of Explain Analyze after adding CREATE INDEX idx_location_company ON app_jobposting(location_id, company_id);



```sql
EXPLAIN ANALYZE
SELECT
    loc.city,
    COUNT(DISTINCT c.name) AS unique_companies
FROM app_jobposting jp
JOIN app_location loc ON jp.location_id = loc.id
JOIN app_company c ON jp.company_id = c.id
GROUP BY loc.city
ORDER BY unique_companies DESC
LIMIT 15;
```

```
-> Limit: 15 row(s) (actual time=18.2..18.2 rows=15 loops=1) -> Sort: unique_companies DESC, limit input to 15 row(s) per chunk (actual time=18.2..18.2 rows=15 loops=1) -> Stream results (actual time=16.3..18.1 rows=214 loops=1) -> Group aggregate: count(distinct app_company.`name`) (actual time=16.3..18.1 rows=214 loops=1) -> Sort: loc.city (actual time=16.2..16.4 rows=2100 loops=1) -> Stream results (cost=1551 rows=1909) (actual time=0.107..12 rows=2100 loops=1) -> Nested loop inner join (cost=1551 rows=1909) (actual time=0.102..10.9 rows=2100 loops=1) -> Nested loop inner join (cost=883 rows=1909) (actual time=0.0916..8.1 rows=2100 loops=1) -> Table scan on c (cost=215 rows=1909) (actual time=0.0576..1.02 rows=2127 loops=1) -> Index lookup on jp using idx_company_id (company_id=c.id) (cost=0.25 rows=1) (actual time=0.0026..0.00314 rows=0.987 loops=2127) -> Single-row index lookup on loc using PRIMARY (id=jp.location_id) (cost=0.25 rows=1) (actual time=0.00112..0.00115 rows=1 loops=2100)
```