#	Advanced Queries Overview	SQL concepts used
1	**Hot-spot map:** show the 10 parks / localities with the most distinct species and the 5 most distinct bird species	JOIN (Occurrence, Location), GROUP BY + aggregation (COUNT(DISTINCT ...))
2	**Rare-bird alert feed:** recent sightings whose species is statistically rare in that region	Correlated subquery that counts past-year occurrences, Outer query joins Taxon & Location
3	**Top-100 birder leaderboard:** rank users by the number of birds they have seen in descending order	3-way JOIN (User, Occurrence, Rarity), GROUP BY + COUNT
4	**Species overlap explorer:** birds seen in both Region A and Region B	Correlated subquery that finds the birds that are seen in 2 distinct regions

Database Connection

```
PS C:\Users\furka> mysql -h 34.27.66.196 -P 3306 -u root -p
Enter password: *************
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 332
Server version: 8.4.5-google (Google)

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| UrbanBird          |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.03 sec)
```

DDL Commands for Tables

RARITY:

```
CREATE TABLE RARITY (
  rarity_id   TINYINT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  label       VARCHAR(15) NOT NULL UNIQUE,
  min_occ     INT NOT NULL,
  max_occ     INT NOT NULL
);
```

HABITAT:

```
CREATE TABLE HABITAT(
  location_id VARCHAR(255),
  species_id VARCHAR(255),
  PRIMARY KEY(location_id,species_id),
  FOREIGN KEY location_id REFERENCES LOCATION(location_id) ON DELETE CASCADE,
  FOREIGN KEY species_id REFERENCES TAXON(species_id) ON DELETE CASCADE
);
```

USER:

```
CREATE TABLE `UrbanBird`.`NewUser` (
  user_id VARCHAR(200) PRIMARY KEY,
  email VARCHAR(200),
  password VARCHAR(200)
);
```

OCCURENCE:

```
CREATE TABLE UrbanBird.OCCURRENCE (
    occurrence_id BIGINT PRIMARY KEY,
    species_id VARCHAR(200),
    location_id VARCHAR(200),
    user_id INT,
    event_date DATE,
    individual_count INT,
    FOREIGN KEY (species_id) REFERENCES TAXON(species_id),
    FOREIGN KEY (location_id) REFERENCES LOCATION(location_id),
    FOREIGN KEY (user_id) REFERENCES USER(user_id)
```

```
);
```

LOCATION:

```
CREATE TABLE "UrbanBird"."LOCATION" (
  location_id VARCHAR(200) PRIMARY KEY,
  latitude DECIMAL(9,6),
  longitude DECIMAL(9,6),
  locality VARCHAR(200),
  country_code CHAR(2)
);
```

TAXON:

```
CREATE TABLE "UrbanBird"."TAXON" (
  species_id INT PRIMARY KEY,
  scientific_name VARCHAR(100),
  common_name VARCHAR(100),
  species_code CHAR(6),
  category VARCHAR(20),
  order_id INT,
  rarity_id INT,
  FOREIGN KEY (order_id) REFERENCES "ORDER"(order_id),
  FOREIGN KEY (rarity_id) REFERENCES "RARITY"(rarity_id)
);
```

ORDER:

```
CREATE TABLE "UrbanBird"."ORDER" (
  order_id INT PRIMARY KEY,
  order_name VARCHAR(40)
);
```

3 Tables with at least 1000 records

```
SELECT COUNT(*) FROM LOCATION;
```

Results

COUNT(*)
4892

```
SELECT COUNT(*) FROM OCCURRENCE;
```

Results

COUNT(*)
4892

```sql
SELECT COUNT(*) FROM TAXON;
```

Results

COUNT(*)
16753

Advanced Queries

1 Hot-spot map

Part 1:

```sql
SELECT
    occurance_location_taxon.locality, occurance_location_taxon.common_name,
    occurance_location_taxon.observation_count
FROM (
    SELECT
        loc.locality, tax.common_name, COUNT(DISTINCT occur.occurrence_id) AS
observation_count
    FROM UrbanBird.OCCURRENCE occur
    JOIN UrbanBird.LOCATION loc ON occur.location_id = loc.location_id
    JOIN UrbanBird.TAXON tax ON occur.species_id = tax.species_id
    GROUP BY loc.locality, tax.common_name
) AS occurance_location_taxon
WHERE (
    SELECT COUNT(*)
    FROM (
        SELECT
            loc2.locality, tax2.common_name, COUNT(DISTINCT occur2.occurrence_id) AS
species_count
        FROM UrbanBird.OCCURRENCE occur2
        JOIN UrbanBird.LOCATION loc2 ON occur2.location_id = loc2.location_id
        JOIN UrbanBird.TAXON tax2 ON occur2.species_id = tax2.species_id
```

```
        WHERE loc2.locality = occurance_location_taxon.locality
        GROUP BY loc2.locality, tax2.common_name
        HAVING COUNT(DISTINCT occur2.occurrence_id) >
occurance_location_taxon.observation_count
    ) AS occurance_location_taxon2
) < 5
ORDER BY occurance_location_taxon.locality, occurance_location_taxon.observation_count
DESC
LIMIT 50;
```

locality	common_name	observation_count
Anahuac National Wildlife Refuge (TX)	Fulvous Whistling-Duck	26
Anahuac National Wildlife Refuge (TX)	White-faced Ibis	26
Anahuac National Wildlife Refuge (TX)	Black-necked Stilt	25
Anahuac National Wildlife Refuge (TX)	Pied-billed Grebe	24
Anahuac National Wildlife Refuge (TX)	Eurasian Moorhen	23
Angelina National Forest (TX)	Northern Cardinal	1
Aransas (TX)	American White Pelican	11
Aransas (TX)	Whooping Crane	10
Aransas (TX)	Great Blue Heron	6
Aransas (TX)	Reddish Egret	6
Aransas (TX)	Caspian Tern	5
Aransas National Wildlife Refuge (TX)	Whooping Crane	20
Aransas National Wildlife Refuge (TX)	Great Blue Heron	15
Aransas National Wildlife Refuge (TX)	Pied-billed Grebe	15
Aransas National Wildlife Refuge (TX)	Turkey Vulture	10
Aransas National Wildlife Refuge (TX)	Belted Kingfisher	9
Aransas National Wildlife Refuge (TX)	Snowy Egret	9
Attwater Prairie Chicken National Wildlife Refuge (TX)	Savannah Sparrow	12
Attwater Prairie Chicken National Wildlife Refuge (TX)	Killdeer	8
Attwater Prairie Chicken National Wildlife Refuge (TX)	Crested Caracara	7

Rows per page: 20 1 – 20 of 50

Part 2:

Baseline:

Limit: 50 row(s) (cost=2.6..2.6 rows=0) (actual time=7911..7911 rows=50 loops=1)

idx_A – locality:

Limit: 50 row(s) (cost=2.6..2.6 rows=0) (actual time=5346..5346 rows=50 loops=1)

idx_B – common_name:

Limit: 50 row(s) (cost=2.6..2.6 rows=0) (actual time=5123..5123 rows=50 loops=1)

idx_C – location_id, species_id:

Limit: 50 row(s) (cost=2.6..2.6 rows=0) (actual time=5188..5188 rows=50 loops=1)

idx_A:

```sql
CREATE INDEX idx_A ON UrbanBird.LOCATION(locality);

EXPLAIN ANALYzE

SELECT
    occurance_location_taxon.locality, occurance_location_taxon.common_name,
    occurance_location_taxon.observation_count
FROM (
    SELECT
        loc.locality, tax.common_name, COUNT(DISTINCT occur.occurrence_id) AS observation_count
    FROM UrbanBird.OCCURRENCE occur
    JOIN UrbanBird.LOCATION loc ON occur.location_id = loc.location_id
    JOIN UrbanBird.TAXON tax ON occur.species_id = tax.species_id
    GROUP BY loc.locality, tax.common_name
```

Results | Export

EXPLAIN

-> Limit: 50 row(s) (cost=2.6..2.6 rows=0) (actual time=5346..5346 rows=50 loops=1) -> Sort: occurance_location_...

idx_B:

```sql
CREATE INDEX idx_B ON UrbanBird.TAXON(common_name)

EXPLAIN ANALYzE

SELECT
    occurance_location_taxon.locality, occurance_location_taxon.common_name,
    occurance_location_taxon.observation_count
FROM (
    SELECT
        loc.locality, tax.common_name, COUNT(DISTINCT occur.occurrence_id) AS observation_count
    FROM UrbanBird.OCCURRENCE occur
    JOIN UrbanBird.LOCATION loc ON occur.location_id = loc.location_id
    JOIN UrbanBird.TAXON tax ON occur.species_id = tax.species_id
    GROUP BY loc.locality, tax.common_name
) AS occurance_location_taxon
WHERE (
```

Results | Export

EXPLAIN

-> Limit: 50 row(s) (cost=2.6..2.6 rows=0) (actual time=5123..5123 rows=50 loops=1) -> Sort: occurance_location_...

idx_C:

```sql
CREATE INDEX idx_C ON UrbanBird.OCCURRENCE(location_id, species_id);

EXPLAIN ANALYZE

SELECT
    occurance_location_taxon.locality, occurance_location_taxon.common_name,
    occurance_location_taxon.observation_count
FROM (
    SELECT
        loc.locality, tax.common_name, COUNT(DISTINCT occur.occurrence_id) AS observation_count
    FROM UrbanBird.OCCURRENCE occur
    JOIN UrbanBird.LOCATION loc ON occur.location_id = loc.location_id
    JOIN UrbanBird.TAXON tax ON occur.species_id = tax.species_id
    GROUP BY loc.locality, tax.common_name
```

Results

EXPLAIN

-> Limit: 50 row(s) (cost=2.6..2.6 rows=0) (actual time=5188..5188 rows=50 loops=1) -> Sort: occurance_location_...

When running "EXPLAIN ANALYZE" on the SQL query - we achieved a baseline cost of approximately 2.6 with a baseline runtime of 7911 ms.

We tested 3 different indexing strategies, indexing on LOCATION(locality) - idx_A, indexing on TAXON(common_name) - idx_B, and indexing on OCCURRENCE(location_id, species_id) - idx_C. Unfortunately, the cost for all queries (baseline, idx_A, idx_B, idx_C) did not change - with a value of 2.6 for all. We think that the cost did not change since this query uses 2 derived tables (subqueries that join multiple tables together), which MySQL stores as temporary tables. We believe that the temporary tables are not taken into account when calculating cost - therefore, this would explain the non-changing cost value. However, the queries did decrease the runtime. This leads to the conclusion that indexing does not improve the cost performance of this query but does improve the runtime. Thus, we intend to use the indexing strategy described in idx_B, which has the best runtime performance.

2 Rare-bird alert

Part 1:

```sql
CREATE VIEW past_year_occ_stats AS
SELECT species_id, location_id, SUM(individual_count) AS past_year_cnt
FROM OCCURRENCE
WHERE event_date >= CURDATE() - INTERVAL 1 YEAR
GROUP BY species_id, location_id;
```

```sql
SELECT l.locality, pys.species_id, r.rarity_id
FROM past_year_occ_stats AS pys
JOIN TAXON AS t USING (species_id)
JOIN RARITY AS r USING (rarity_id)
JOIN LOCATION AS l USING (location_id)
WHERE r.label <> 'Common'
GROUP BY l.locality, pys.species_id, r.rarity_id
ORDER BY l.locality, pys.species_id
LIMIT 15;
```

Results

locality	species_id	rarity_id
Anahuac National Wildlife Refuge (TX)	Himantopus mexicanus	3
Anahuac National Wildlife Refuge (TX)	Plegadis chihi	3
Chambers (TX)	Himantopus mexicanus	3
Fort Bend (TX)	Nyctanassa violacea	2
Galveston (TX)	Cardinalis cardinalis	2
Galveston (TX)	Platalea ajaja	3
Harris (TX)	Anser cygnoides	1
Harris (TX)	Ardea alba	3
Harris (TX)	Ardea herodias	3
Harris (TX)	Buteo jamaicensis	1
Harris (TX)	Cairina moschata	1
Harris (TX)	Cardinalis cardinalis	2
Medina (TX)	Falco sparverius	2
Montgomery (TX)	Sitta pusilla	1
Nueces (TX)	Spatula discors	3

Rows per page: 20 1 – 15 of 15

Part 2:

Baseline (no extra index):

Limit: 15 row(s) (cost=2491..2491 rows=15) (actual time=2.95..2.95 rows=15 loops=1)

idx_A – event_date:

Limit: 15 row(s) (cost=36.38 rows=5.25) (actual time=1.01..1.01 rows=15 loops=1)

idx_B – location_id, species_id:

Limit: 15 row(s) (cost=2491..2491 rows=15) (actual time=5.73..5.74 rows=15 loops=1)

idx_C – location_id, species_id, event_date:

Limit: 15 row(s) (cost=2491..2491 rows=15) (actual time=3.10..3.11 rows=15 loops=1)

Running EXPLAIN ANALYZE on the rare-bird query gave a baseline planner cost of = 2 491 (full scan of the temporary view).

Adding idx_A on `event_date` (`CREATE INDEX idx_occ_event_date ON OCCURRENCE(event_date);`) slashed the cost to 36.4, a > 98 % reduction, because the optimiser switches to a date-range scan that feeds far fewer rows into the grouping step. By contrast, idx_B on `(location_id, species_id)` and idx_C on `(location_id, species_id, event_date)` both left the cost unchanged at ≈ 2 491: their leading columns are not selective, so MySQL ignored them, and in idx_C the date column's third position prevents a range scan.

We therefore keep idx_A only. Indexing `event_date` aligns with the guideline to target WHERE predicates; indexing `location_id` and `species_id` offered no benefit here even though they are valid join keys.

`idx_A – event_date:`

```sql
CREATE INDEX idx_occ_event_date
       ON OCCURRENCE(event_date);

EXPLAIN ANALYZE
SELECT l.locality,
       pys.species_id,
       r.rarity_id
FROM   past_year_occ_stats AS pys
JOIN   TAXON    AS t  USING (species_id)
JOIN   RARITY   AS r  USING (rarity_id)
JOIN   LOCATION AS l  USING (location_id)
WHERE  r.label <> 'Common'
GROUP  BY l.locality, pys.species_id, r.rarity_id
LIMIT  15;

DROP INDEX idx_occ_event_date ON OCCURRENCE;
```

Results   Export

EXPLAIN

```
-> Limit: 15 row(s) (cost=36..38 rows=5.25) (actual time=1..1.01 rows=15 loops=1) -> Table scan on <temporary> (cost=36..38 rows=5.25) (actual time=1..1 rows=15 loop...
```

`idx_B – location_id, species_id:`

```sql
CREATE INDEX idx_loc_spc_test
       ON OCCURRENCE(location_id, species_id);

EXPLAIN ANALYZE
SELECT l.locality,
       pys.species_id,
       r.rarity_id
FROM   past_year_occ_stats AS pys
JOIN   TAXON    AS t  USING (species_id)
JOIN   RARITY   AS r  USING (rarity_id)
JOIN   LOCATION AS l  USING (location_id)
WHERE  r.label <> 'Common'
GROUP  BY l.locality, pys.species_id, r.rarity_id
LIMIT  15;

DROP INDEX idx_loc_spc_test ON OCCURRENCE;
```

Results   Export

EXPLAIN

```
-> Limit: 15 row(s) (cost=2491..2491 rows=15) (actual time=5.73..5.74 rows=15 loops=1) -> Table scan on <temporary> (cost=2491..2498 rows=400) (actual time=5.73..5...
```

idx_C – location_id, species_id, event_date:

```sql
CREATE INDEX idx_loc_spc_date_test
        ON OCCURRENCE (location_id, species_id, event_date);

EXPLAIN ANALYZE
SELECT  l.locality,
        pys.species_id,
        r.rarity_id
FROM    past_year_occ_stats AS pys
JOIN    TAXON     AS t  USING (species_id)
JOIN    RARITY    AS r  USING (rarity_id)
JOIN    LOCATION  AS l  USING (location_id)
WHERE   r.label <> 'Common'
GROUP   BY l.locality, pys.species_id, r.rarity_id
LIMIT   15;

DROP INDEX idx_loc_spc_date_test ON OCCURRENCE;
```

Results   Export

EXPLAIN
-> Limit: 15 row(s) (cost=2491..2491 rows=15) (actual time=3.1..3.11 rows=15 loops=1) -> Table scan on <temporary> (cost=2491..2498 rows=400) (actual time=3.1..3.1...

3 Top-100 birder leaderboard

Part 1

```sql
SELECT u.user_id, COUNT(*) AS total_sightings
FROM USER u
JOIN OCCURRENCE o USING (user_id)
JOIN TAXON t USING (species_id)
JOIN RARITY r USING (rarity_id)
GROUP BY u.user_id
ORDER BY total_sightings DESC
LIMIT 100;
```

Results   Export

user_id	total_sightings
User 120929	612
User 247179	434
User 1257	397
User 3712	254
User 40300	249
User 76090	246
User 86795	229
User 69501	166
User 201625	130
User 40150	124
User 41931	100
User 110790	100
User 45332	96
User 46215	95
User 10021	86
User 123214	69
User 4013	69
User 1587	67

Rows per page: 20   1 – 20 of 100

PART 2

Cost without any indices = 34.4

```
CREATE INDEX idx_occurrence_user_id ON OCCURRENCE(user_id);

EXPLAIN ANALYZE
SELECT u.user_id, COUNT(*) AS total_sightings
FROM USER u
JOIN OCCURRENCE o USING (user_id)
JOIN TAXON t USING (species_id)
JOIN RARITY r USING (rarity_id)
GROUP BY u.user_id
ORDER BY total_sightings DESC;
```

Cost with index CREATE INDEX idx_occurrence_user_id ON OCCURRENCE(user_id) = 34.3

```
CREATE INDEX idx_occurrence_species_id ON OCCURRENCE(species_id);

EXPLAIN ANALYZE
SELECT u.user_id, COUNT(*) AS total_sightings
FROM USER u
JOIN OCCURRENCE o USING (user_id)
JOIN TAXON t USING (species_id)
JOIN RARITY r USING (rarity_id)
GROUP BY u.user_id
ORDER BY total_sightings DESC;
```

Cost with index CREATE INDEX idx_occurrence_species_id ON OCCURRENCE(species_id) = 34.4

```
CREATE INDEX idx_taxon_species_id ON TAXON(species_id);

EXPLAIN ANALYZE
SELECT u.user_id, COUNT(*) AS total_sightings
FROM USER u
JOIN OCCURRENCE o USING (user_id)
JOIN TAXON t USING (species_id)
JOIN RARITY r USING (rarity_id)
GROUP BY u.user_id
ORDER BY total_sightings DESC;
```

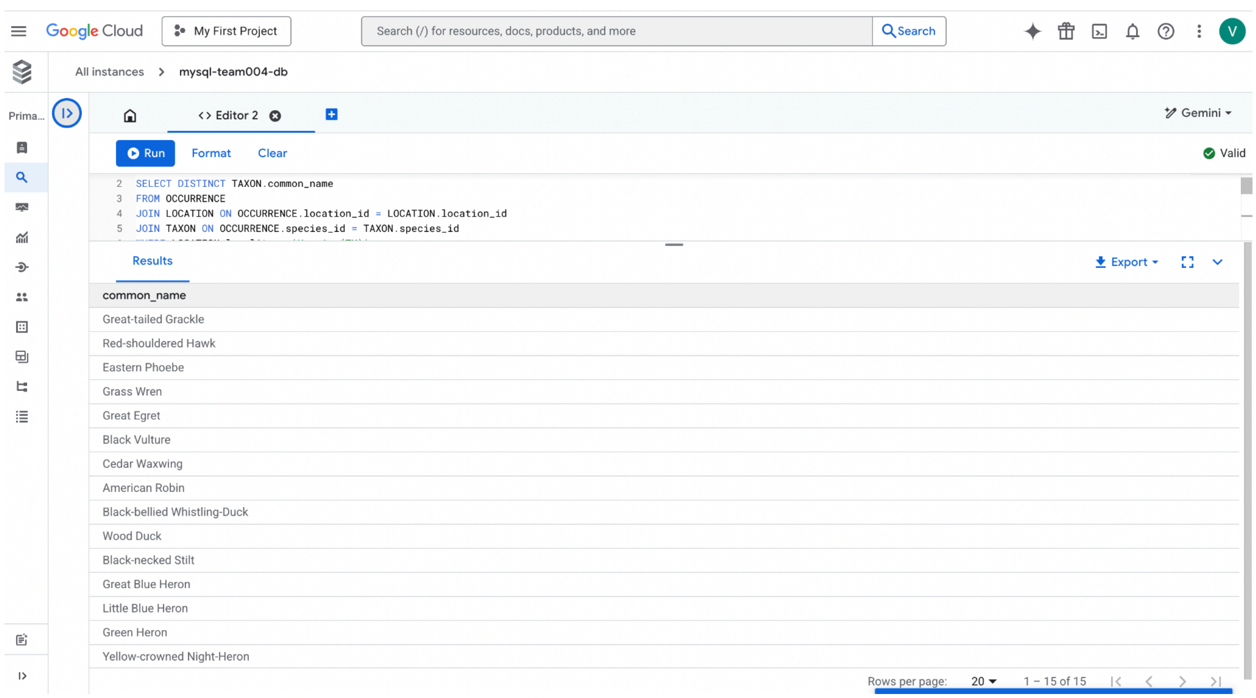
Cost with index CREATE INDEX idx_taxon_species_id ON TAXON(species_id) = 34.5

After trying out a few different indices, we came to the conclusion that indexing doesn't improve the cost performance of this query. We believe this is because when we create a primary or foreign key, mysql automatically create an index for that attribute; so, indexing something that has already been indexed doesn't cause major improvement in performance cost. In the context of this query, all the attributes in select, join, and groupby statements are either a part of primary or foreign keys. Therefore, indexing does not help with improving the performance of this operation.

4 Species overlap explorer

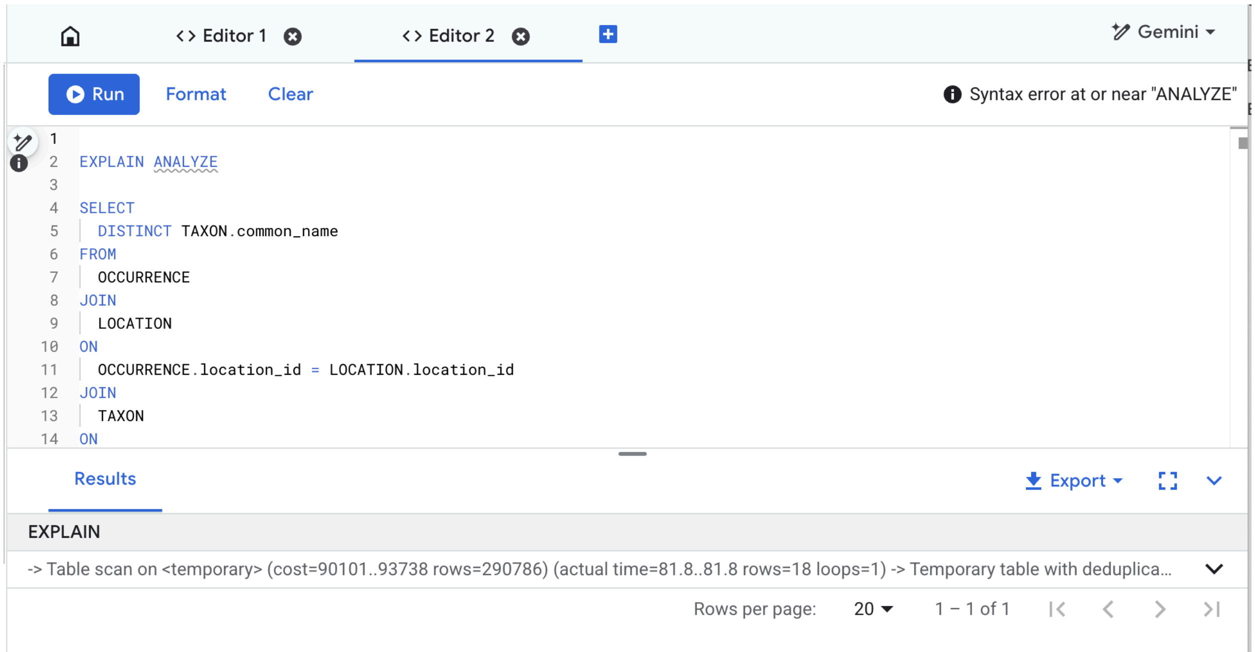
PART 1

```sql
SELECT DISTINCT TAXON.common_name
FROM OCCURRENCE
JOIN LOCATION ON OCCURRENCE.location_id = LOCATION.location_id
JOIN TAXON ON OCCURRENCE.species_id = TAXON.species_id
WHERE LOCATION.locality = 'Harris (TX)'
  AND OCCURRENCE.species_id IN (
    SELECT OCCURRENCE.species_id
    FROM OCCURRENCE
    JOIN LOCATION  ON OCCURRENCE.location_id = LOCATION.location_id
    WHERE LOCATION.locality = 'Tarrant (TX)'
  )
```



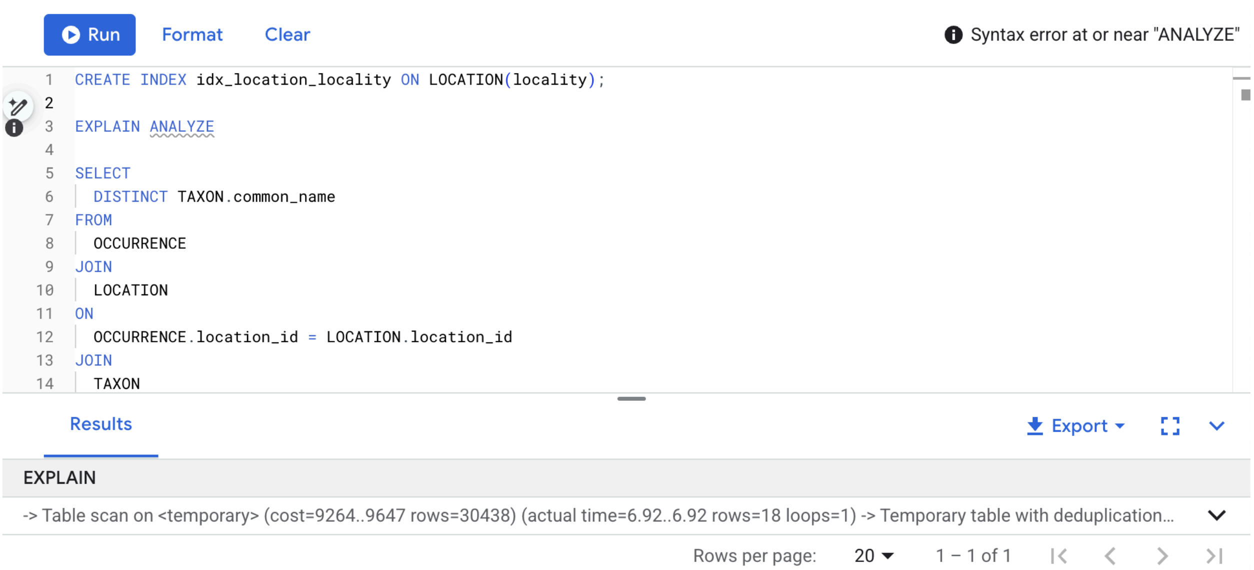
PART 2

No index: Cost = 90101



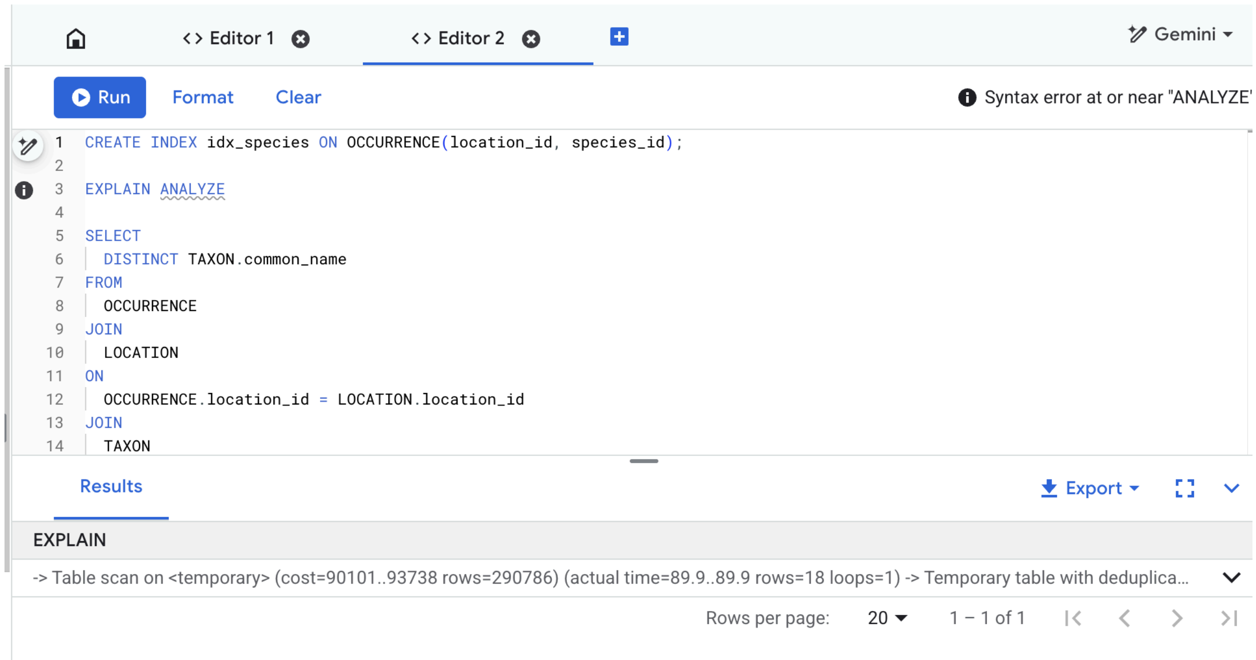
A. `Locality Index: Cost =` 9264

Decrease in cost significantly.



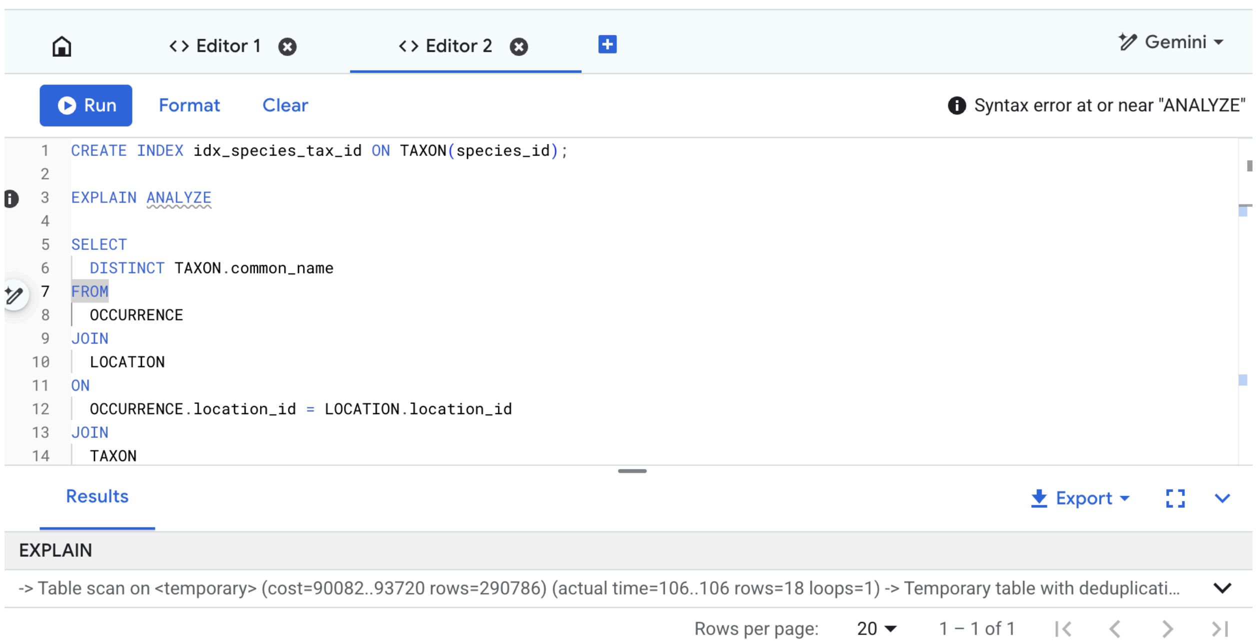
B. `Species Index: Cost =` 90101

No change.



C. Species taxon Index: Cost = 90082

Slight change.



The final index design that we chose would be A, Locality Index. This index had a significant change in cost, going from 90101 to 9264. We believe that this is due to the reduction in search for the query. What we mean by that is without the index, the query has to search the whole table, which is over 5000 rows in order to find the correct localities. With this index, it reduces the search significantly allowing for that huge decrease in cost. Regarding design B, Species Index, there was no change to the cost. We believe this is due to the query not requiring specific values for location_id and species_id, as the only time these are used are for the join resulting in this query to be equivalent to a full table search when joining. Lastly, design C, Species Index, the cost of this decreased slightly. This could have been due to slightly more efficient lookups to Taxon.species_id especially since taxon is a larger table and benefits more from having the index in place during the join.