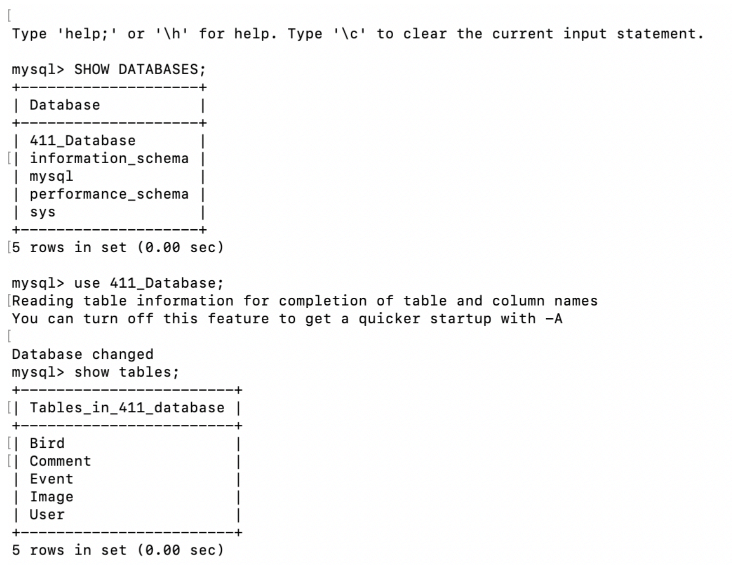
Part 1

Connection:

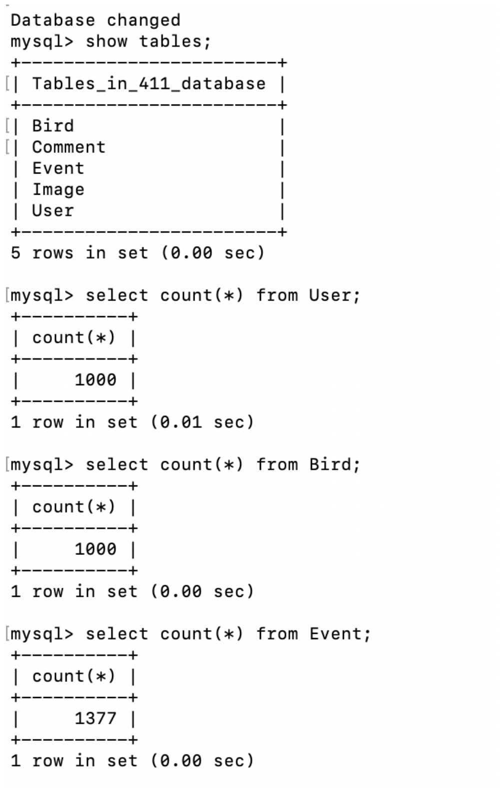


```
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| 411_Database       |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.00 sec)

mysql> use 411_Database;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+------------------------+
| Tables_in_411_database |
+------------------------+
| Bird                   |
| Comment                |
| Event                  |
| Image                  |
| User                   |
+------------------------+
5 rows in set (0.00 sec)
```

DDL commands:

```
CREATE DATABASE 411_Database;
USE 411_Database;
CREATE TABLE User (
  uid INT PRIMARY KEY AUTO_INCREMENT,
  user_name VARCHAR(100) NOT NULL,
  password VARCHAR(255) NOT NULL
);
CREATE TABLE Bird (
  bird_scientific_name VARCHAR(150) PRIMARY KEY,
  bird_common_name VARCHAR(100) NOT NULL,
  bird_description TEXT
);
CREATE TABLE Image (
  image_id INT PRIMARY KEY AUTO_INCREMENT,
  event_id INT NOT NULL,
  bird_scientific_name VARCHAR(150) NOT NULL,
  FOREIGN KEY (bird_scientific_name) REFERENCES Bird(bird_scientific_name)
);
CREATE TABLE Event (
  event_id INT PRIMARY KEY AUTO_INCREMENT,
  user_id INT NOT NULL,
  event_time DATETIME NOT NULL,
  longitude DECIMAL(9,6),
  latitude DECIMAL(9,6),
  Country VARCHAR(100),
  State VARCHAR(100),
  bird_scientific_name VARCHAR(150) NOT NULL,
  FOREIGN KEY (user_id) REFERENCES User(uid),
  FOREIGN KEY (bird_scientific_name) REFERENCES Bird(bird_scientific_name)
);
CREATE TABLE Comment (
  comment_id INT AUTO_INCREMENT,
  event_id INT NOT NULL,
  content TEXT NOT NULL,
  uid INT NOT NULL,
  comment_time DATETIME NOT NULL,
  PRIMARY KEY(comment_id, event_id),
  FOREIGN KEY (event_id) REFERENCES Event(event_id)
);
```

Table Rows:

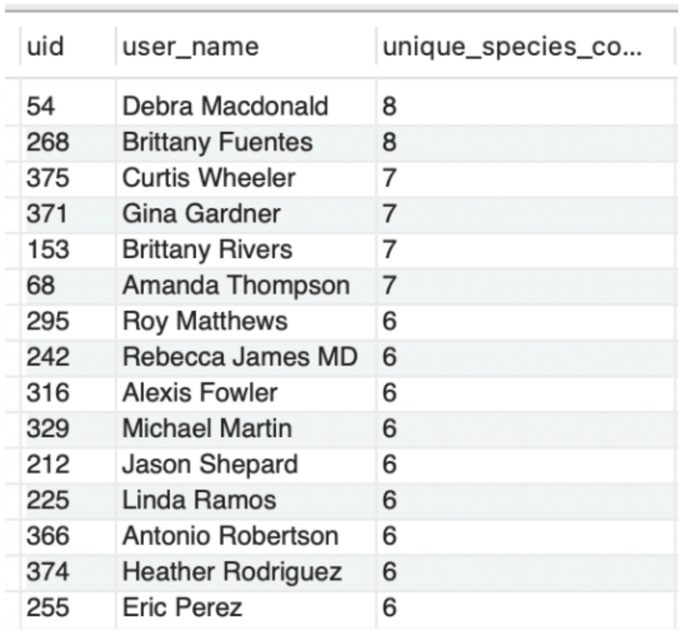


```
Database changed
mysql> show tables;
+------------------------+
| Tables_in_411_database |
+------------------------+
| Bird                   |
| Comment                |
| Event                  |
| Image                  |
| User                   |
+------------------------+
5 rows in set (0.00 sec)

mysql> select count(*) from User;
+----------+
| count(*) |
+----------+
|     1000 |
+----------+
1 row in set (0.01 sec)

mysql> select count(*) from Bird;
+----------+
| count(*) |
+----------+
|     1000 |
+----------+
1 row in set (0.00 sec)

mysql> select count(*) from Event;
+----------+
| count(*) |
+----------+
|     1377 |
+----------+
1 row in set (0.00 sec)
```

Advanced SQL Queries:

1. Query to Find the Top 5 Users Who Have Uploaded the Most Unique Bird Species

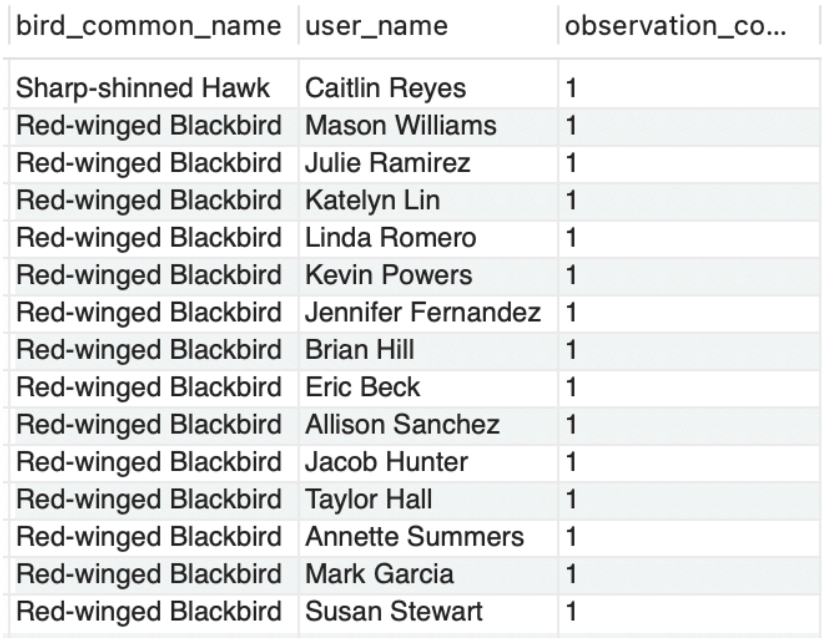
```
SELECT
   U.uid,
   U.user_name,
   COUNT(DISTINCT E.bird_scientific_name) AS unique_species_count
FROM
   User U
JOIN
   Event E ON U.uid = E.user_id
GROUP BY
   U.uid, U.user_name
ORDER BY
   unique_species_count DESC
LIMIT 15;
```



uid	user_name	unique_species_co...
54	Debra Macdonald	8
268	Brittany Fuentes	8
375	Curtis Wheeler	7
371	Gina Gardner	7
153	Brittany Rivers	7
68	Amanda Thompson	7
295	Roy Matthews	6
242	Rebecca James MD	6
316	Alexis Fowler	6
329	Michael Martin	6
212	Jason Shepard	6
225	Linda Ramos	6
366	Antonio Robertson	6
374	Heather Rodriguez	6
255	Eric Perez	6

2. List which user observed each bird most often

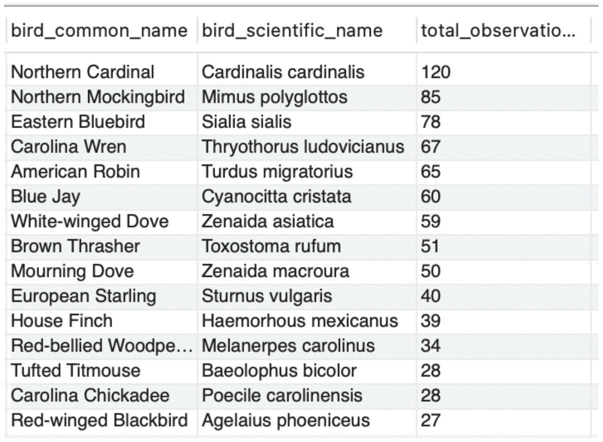
```
SELECT
  b.bird_common_name,
  u.user_name,
  bird_stats.observation_count
FROM (
  SELECT
    e.bird_scientific_name,
    e.user_id,
    COUNT(*) AS observation_count,
    RANK() OVER (PARTITION BY e.bird_scientific_name ORDER BY COUNT(*) DESC) AS rnk
  FROM Event e
  GROUP BY e.bird_scientific_name, e.user_id
) AS bird_stats
JOIN User u ON bird_stats.user_id = u.uid
JOIN Bird b ON bird_stats.bird_scientific_name = b.bird_scientific_name
WHERE bird_stats.rnk = 1
LIMIT 15;
```



bird_common_name	user_name	observation_co...
Sharp-shinned Hawk	Caitlin Reyes	1
Red-winged Blackbird	Mason Williams	1
Red-winged Blackbird	Julie Ramirez	1
Red-winged Blackbird	Katelyn Lin	1
Red-winged Blackbird	Linda Romero	1
Red-winged Blackbird	Kevin Powers	1
Red-winged Blackbird	Jennifer Fernandez	1
Red-winged Blackbird	Brian Hill	1
Red-winged Blackbird	Eric Beck	1
Red-winged Blackbird	Allison Sanchez	1
Red-winged Blackbird	Jacob Hunter	1
Red-winged Blackbird	Taylor Hall	1
Red-winged Blackbird	Annette Summers	1
Red-winged Blackbird	Mark Garcia	1
Red-winged Blackbird	Susan Stewart	1

3. Find the bird with the highest number of observations

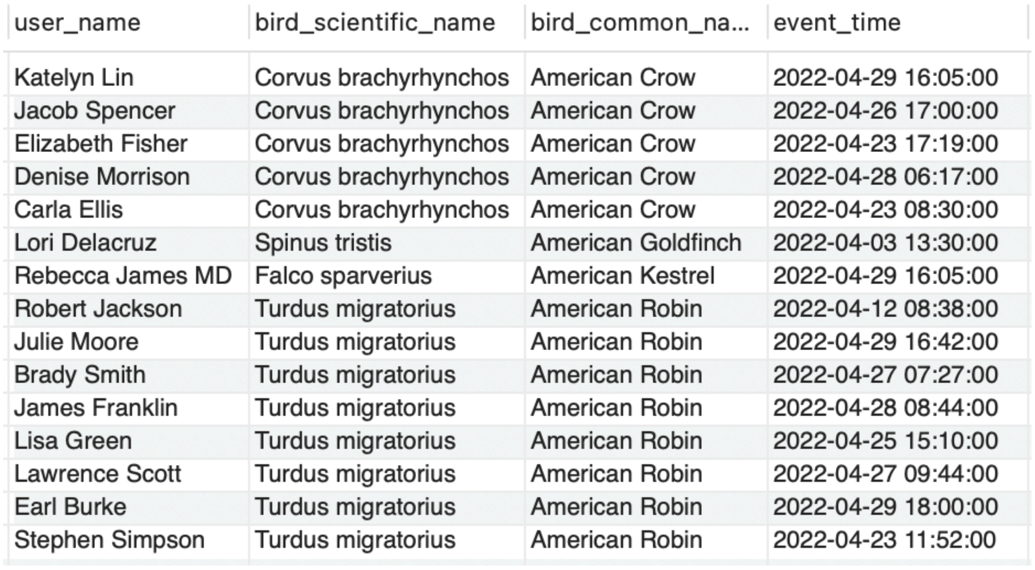
```
SELECT
  b.bird_common_name,
  b.bird_scientific_name,
  COUNT(e.event_id) AS total_observations
FROM
  Bird b
JOIN
  Event e ON b.bird_scientific_name = e.bird_scientific_name
GROUP BY
  b.bird_scientific_name, b.bird_common_name
ORDER BY
  total_observations DESC
LIMIT 15;
```



bird_common_name	bird_scientific_name	total_observatio...
Northern Cardinal	Cardinalis cardinalis	120
Northern Mockingbird	Mimus polyglottos	85
Eastern Bluebird	Sialia sialis	78
Carolina Wren	Thryothorus ludovicianus	67
American Robin	Turdus migratorius	65
Blue Jay	Cyanocitta cristata	60
White-winged Dove	Zenaida asiatica	59
Brown Thrasher	Toxostoma rufum	51
Mourning Dove	Zenaida macroura	50
European Starling	Sturnus vulgaris	40
House Finch	Haemorhous mexicanus	39
Red-bellied Woodpe...	Melanerpes carolinus	34
Tufted Titmouse	Baeolophus bicolor	28
Carolina Chickadee	Poecile carolinensis	28
Red-winged Blackbird	Agelaius phoeniceus	27

4. List the most recent bird observed by the user (one row per user)

```
SELECT
  u.user_name,
  e.bird_scientific_name,
  b.bird_common_name,
  e.event_time
FROM
  User u
JOIN
  Event e ON u.uid = e.user_id
JOIN
  Bird b ON e.bird_scientific_name = b.bird_scientific_name
WHERE
  e.event_time = (
    SELECT MAX(e2.event_time)
    FROM Event e2
    WHERE e2.user_id = u.uid
  )
LIMIT 15;
```

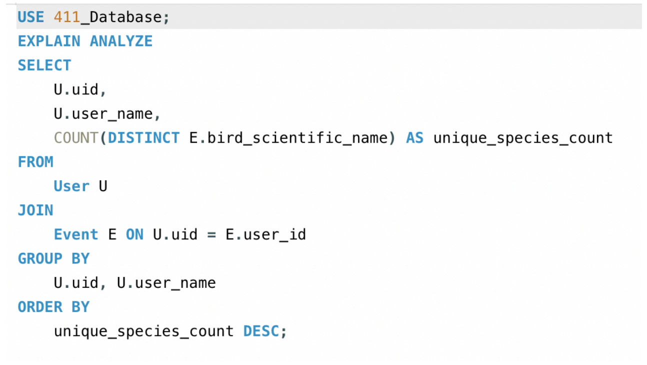


user_name	bird_scientific_name	bird_common_na...	event_time
Katelyn Lin	Corvus brachyrhynchos	American Crow	2022-04-29 16:05:00
Jacob Spencer	Corvus brachyrhynchos	American Crow	2022-04-26 17:00:00
Elizabeth Fisher	Corvus brachyrhynchos	American Crow	2022-04-23 17:19:00
Denise Morrison	Corvus brachyrhynchos	American Crow	2022-04-28 06:17:00
Carla Ellis	Corvus brachyrhynchos	American Crow	2022-04-23 08:30:00
Lori Delacruz	Spinus tristis	American Goldfinch	2022-04-03 13:30:00
Rebecca James MD	Falco sparverius	American Kestrel	2022-04-29 16:05:00
Robert Jackson	Turdus migratorius	American Robin	2022-04-12 08:38:00
Julie Moore	Turdus migratorius	American Robin	2022-04-29 16:42:00
Brady Smith	Turdus migratorius	American Robin	2022-04-27 07:27:00
James Franklin	Turdus migratorius	American Robin	2022-04-28 08:44:00
Lisa Green	Turdus migratorius	American Robin	2022-04-25 15:10:00
Lawrence Scott	Turdus migratorius	American Robin	2022-04-27 09:44:00
Earl Burke	Turdus migratorius	American Robin	2022-04-29 18:00:00
Stephen Simpson	Turdus migratorius	American Robin	2022-04-23 11:52:00

Part 2

Advance Query #1: Query to Find the Top 5 Users Who Have Uploaded the Most Unique Bird Species

EXPLAIN ANALYZE Command



```sql
USE 411_Database;
EXPLAIN ANALYZE
SELECT
    U.uid,
    U.user_name,
    COUNT(DISTINCT E.bird_scientific_name) AS unique_species_count
FROM
    User U
JOIN
    Event E ON U.uid = E.user_id
GROUP BY
    U.uid, U.user_name
ORDER BY
    unique_species_count DESC;
```

result before adding indexing

```
-> Sort: unique_species_count DESC  (actual time=8.36..8.41 rows=467 loops=1)
   -> Stream results  (actual time=6.73..8.06 rows=467 loops=1)
      -> Group aggregate: count(distinct `event`.bird_scientific_name)  (actual time=6.72..7.86 rows=467 loops=1)
         -> Sort: u.uid, u.user_name  (actual time=6.68..6.9 rows=1377 loops=1)
            -> Stream results  (cost=622 rows=1377) (actual time=0.0629..5.6 rows=1377 loops=1)
               -> Nested loop inner join  (cost=622 rows=1377) (actual time=0.0596..4.8 rows=1377 loops=1)
                  -> Table scan on E  (cost=140 rows=1377) (actual time=0.0458..0.854 rows=1377 loops=1)
                  -> Single-row index lookup on U using PRIMARY (uid = E.user_id)  (cost=0.25 rows=1) (actual time=0.00241..0.00248 rows=1 loops=1377)
```

After CREATE INDEX idx_event_user_id ON Event(user_id):

```
-> Sort: unique_species_count DESC  (actual time=9.01..9.05 rows=467 loops=1)
   -> Stream results  (actual time=7.4..8.8 rows=467 loops=1)
      -> Group aggregate: count(distinct `event`.bird_scientific_name)  (actual time=7.39..8.51 rows=467 loops=1)
         -> Sort: u.uid, u.user_name  (actual time=7.38..7.59 rows=1377 loops=1)
            -> Stream results  (cost=622 rows=1377) (actual time=0.131..6.16 rows=1377 loops=1)
               -> Nested loop inner join  (cost=622 rows=1377) (actual time=0.127..5.34 rows=1377 loops=1)
                  -> Table scan on E  (cost=140 rows=1377) (actual time=0.106..1.02 rows=1377 loops=1)
                  -> Single-row index lookup on U using PRIMARY (uid = E.user_id)  (cost=0.25 rows=1) (actual time=0.00271..0.00277 rows=1 loops=1377)
```

After CREATE INDEX idx_event_bird_name ON Event(bird_scientific_name):

```
-> Sort: unique_species_count DESC  (actual time=5.47..5.49 rows=467 loops=1)
   -> Stream results  (actual time=4.64..5.34 rows=467 loops=1)
      -> Group aggregate: count(distinct `event`.bird_scientific_name)  (actual time=4.64..5.24
rows=467 loops=1)
         -> Sort: u.uid, u.user_name  (actual time=4.63..4.74 rows=1377 loops=1)
            -> Stream results  (cost=622 rows=1377) (actual time=0.0868..3.98 rows=1377 loops=1)
               -> Nested loop inner join  (cost=622 rows=1377) (actual time=0.0841..3.43 rows=1377
loops=1)
                  -> Table scan on E  (cost=140 rows=1377) (actual time=0.0654..0.624 rows=1377
loops=1)
                  -> Single-row index lookup on U using PRIMARY (uid = E.user_id)  (cost=0.25 rows=1)
(actual time=0.00173..0.00177 rows=1 loops=1377)
```

After CREATE INDEX idx_event_user_bird ON Event(user_id, bird_scientific_name):

```
-> Sort: unique_species_count DESC  (actual time=3.8..3.87 rows=467 loops=1)
   -> Stream results  (actual time=2.69..3.64 rows=467 loops=1)
      -> Group aggregate: count(distinct `event`.bird_scientific_name)  (actual time=2.69..3.5
rows=467 loops=1)
         -> Sort: u.uid, u.user_name  (actual time=2.68..2.79 rows=1377 loops=1)
            -> Stream results  (cost=622 rows=1377) (actual time=0.0923..2.2 rows=1377 loops=1)
               -> Nested loop inner join  (cost=622 rows=1377) (actual time=0.0891..1.7 rows=1377
loops=1)
                  -> Covering index scan on E using idx_event_user_bird  (cost=140 rows=1377) (actual
time=0.0697..0.468 rows=1377 loops=1)
                  -> Single-row index lookup on U using PRIMARY (uid = E.user_id)  (cost=0.25 rows=1)
(actual time=649e-6..690e-6 rows=1 loops=1377)
```

After CREATE INDEX idx_user_name ON User(user_name):

```
-> Sort: unique_species_count DESC  (actual time=10.1..10.2 rows=467 loops=1)
   -> Stream results  (actual time=8.57..9.8 rows=467 loops=1)
      -> Group aggregate: count(distinct `event`.bird_scientific_name)  (actual time=8.57..9.63
rows=467 loops=1)
         -> Sort: u.uid, u.user_name  (actual time=8.55..8.75 rows=1377 loops=1)
            -> Stream results  (cost=622 rows=1377) (actual time=0.138..7.34 rows=1377 loops=1)
```

```
        -> Nested loop inner join  (cost=622 rows=1377) (actual time=0.131..6.36 rows=1377
loops=1)
            -> Table scan on E  (cost=140 rows=1377) (actual time=0.0973..1.16 rows=1377
loops=1)
            -> Single-row index lookup on U using PRIMARY (uid = E.user_id)  (cost=0.25 rows=1)
(actual time=0.00332..0.00339 rows=1 loops=1377)
```

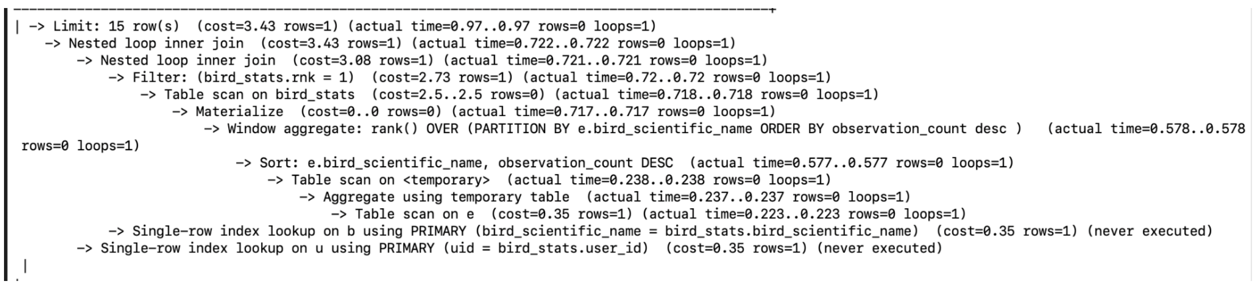
Report:

The final index design chosen for optimizing this query is a composite index on (user_id, bird_scientific_name) in the Event table. This index significantly reduced the query cost in the run time by optimizing both the join between User and Event and the aggregation performed by COUNT(DISTINCT bird_scientific_name). The composite index proved more efficient than individual indexes, avoiding redundancy and improving overall query performance. This design was selected because it minimized execution time, reduced table scans, and effectively handled most query variations without adding unnecessary indexes.

Advanced Query #2: List which user observed each bird most often

Before implementing: time - 0.97

```
mysql> EXPLAIN ANALYZE
    -> SELECT
    ->     b.bird_common_name,
    ->     u.user_name,
    ->     bird_stats.observation_count
    -> FROM (
    ->     SELECT
    ->         e.bird_scientific_name,
    ->         e.user_id,
    ->         COUNT(*) AS observation_count,
    ->         RANK() OVER (PARTITION BY e.bird_scientific_name ORDER BY COUNT(*) DESC) AS rnk
    ->     FROM Event e
    ->     GROUP BY e.bird_scientific_name, e.user_id
    -> ) AS bird_stats
    -> JOIN User u ON bird_stats.user_id = u.uid
    -> JOIN Bird b ON bird_stats.bird_scientific_name = b.bird_scientific_name
    -> WHERE bird_stats.rnk = 1
    -> LIMIT 15;
```



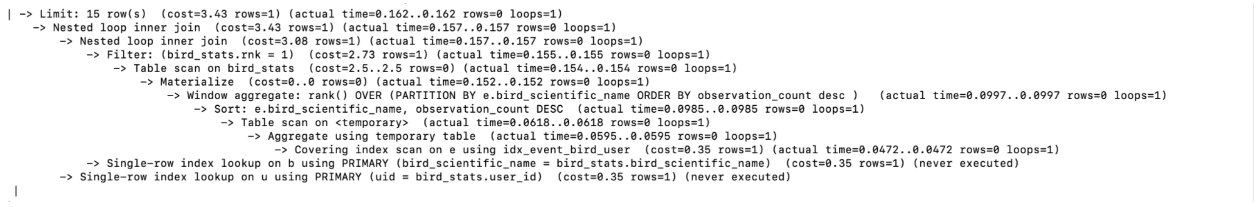
```
-------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (cost=3.43 rows=1) (actual time=0.97..0.97 rows=0 loops=1)
    -> Nested loop inner join  (cost=3.43 rows=1) (actual time=0.722..0.722 rows=0 loops=1)
        -> Nested loop inner join  (cost=3.08 rows=1) (actual time=0.721..0.721 rows=0 loops=1)
            -> Filter: (bird_stats.rnk = 1)  (cost=2.73 rows=1) (actual time=0.72..0.72 rows=0 loops=1)
                -> Table scan on bird_stats  (cost=2.5..2.5 rows=0) (actual time=0.718..0.718 rows=0 loops=1)
                    -> Materialize  (cost=0..0 rows=0) (actual time=0.717..0.717 rows=0 loops=1)
                        -> Window aggregate: rank() OVER (PARTITION BY e.bird_scientific_name ORDER BY observation_count desc )   (actual time=0.578..0.578
 rows=0 loops=1)
                            -> Sort: e.bird_scientific_name, observation_count DESC  (actual time=0.577..0.577 rows=0 loops=1)
                                -> Table scan on <temporary>  (actual time=0.238..0.238 rows=0 loops=1)
                                    -> Aggregate using temporary table  (actual time=0.237..0.237 rows=0 loops=1)
                                        -> Table scan on e  (cost=0.35 rows=1) (actual time=0.223..0.223 rows=0 loops=1)
            -> Single-row index lookup on b using PRIMARY (bird_scientific_name = bird_stats.bird_scientific_name)  (cost=0.35 rows=1) (never executed)
        -> Single-row index lookup on u using PRIMARY (uid = bird_stats.user_id)  (cost=0.35 rows=1) (never executed)
 |
```

Design 1: Composite Index on (bird_scientific_name, user_id)

This index directly supports the GROUP BY and JOIN clauses. By having the leading column as bird_scientific_name (which is used in the PARTITION BY of the window function) followed by user_id, the query can efficiently group and join.

SQL:
CREATE INDEX idx_event_bird_user ON Event(bird_scientific_name, user_id);

Result: **time - 0.162**



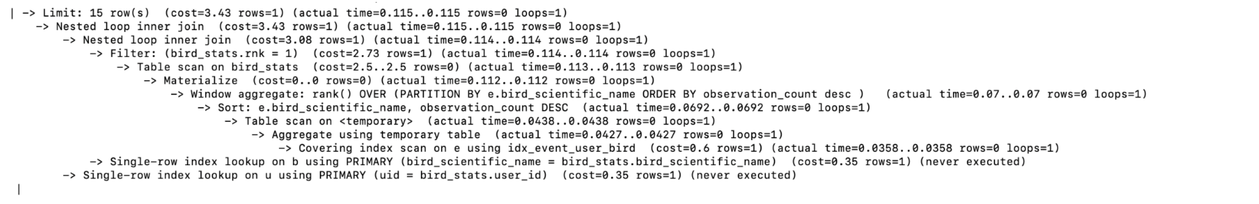
```
| -> Limit: 15 row(s)  (cost=3.43 rows=1) (actual time=0.162..0.162 rows=0 loops=1)
    -> Nested loop inner join  (cost=3.43 rows=1) (actual time=0.157..0.157 rows=0 loops=1)
        -> Nested loop inner join  (cost=3.08 rows=1) (actual time=0.157..0.157 rows=0 loops=1)
            -> Filter: (bird_stats.rnk = 1)  (cost=2.73 rows=1) (actual time=0.155..0.155 rows=0 loops=1)
                -> Table scan on bird_stats  (cost=2.5..2.5 rows=0) (actual time=0.154..0.154 rows=0 loops=1)
                    -> Materialize  (cost=0..0 rows=0) (actual time=0.152..0.152 rows=0 loops=1)
                        -> Window aggregate: rank() OVER (PARTITION BY e.bird_scientific_name ORDER BY observation_count desc )   (actual time=0.0997..0.0997 rows=0 loops=1)
                            -> Sort: e.bird_scientific_name, observation_count DESC  (actual time=0.0985..0.0985 rows=0 loops=1)
                                -> Table scan on <temporary>  (actual time=0.0618..0.0618 rows=0 loops=1)
                                    -> Aggregate using temporary table  (actual time=0.0595..0.0595 rows=0 loops=1)
                                        -> Covering index scan on e using idx_event_bird_user  (cost=0.35 rows=1) (actual time=0.0472..0.0472 rows=0 loops=1)
            -> Single-row index lookup on b using PRIMARY (bird_scientific_name = bird_stats.bird_scientific_name)  (cost=0.35 rows=1) (never executed)
        -> Single-row index lookup on u using PRIMARY (uid = bird_stats.user_id)  (cost=0.35 rows=1) (never executed)
 |
```

Design 2: Composite Index on (user_id, bird_scientific_name)

This alternative index reverses the order. While the GROUP BY still uses both columns, placing user_id first may benefit queries that filter or join on that column more frequently. Comparing the performance here helps determine if column order matters for your workload.

SQL:
CREATE INDEX idx_event_user_bird ON Event(user_id, bird_scientific_name);

Result: time - 0.115



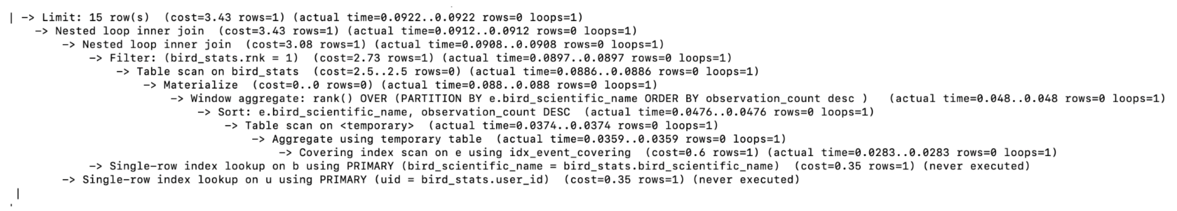
```
| -> Limit: 15 row(s)  (cost=3.43 rows=1) (actual time=0.115..0.115 rows=0 loops=1)
    -> Nested loop inner join  (cost=3.43 rows=1) (actual time=0.115..0.115 rows=0 loops=1)
        -> Nested loop inner join  (cost=3.08 rows=1) (actual time=0.114..0.114 rows=0 loops=1)
            -> Filter: (bird_stats.rnk = 1)  (cost=2.73 rows=1) (actual time=0.114..0.114 rows=0 loops=1)
                -> Table scan on bird_stats  (cost=2.5..2.5 rows=0) (actual time=0.113..0.113 rows=0 loops=1)
                    -> Materialize  (cost=0..0 rows=0) (actual time=0.112..0.112 rows=0 loops=1)
                        -> Window aggregate: rank() OVER (PARTITION BY e.bird_scientific_name ORDER BY observation_count desc )   (actual time=0.07..0.07 rows=0 loops=1)
                            -> Sort: e.bird_scientific_name, observation_count DESC  (actual time=0.0692..0.0692 rows=0 loops=1)
                                -> Table scan on <temporary>  (actual time=0.0438..0.0438 rows=0 loops=1)
                                    -> Aggregate using temporary table  (actual time=0.0427..0.0427 rows=0 loops=1)
                                        -> Covering index scan on e using idx_event_user_bird  (cost=0.6 rows=1) (actual time=0.0358..0.0358 rows=0 loops=1)
            -> Single-row index lookup on b using PRIMARY (bird_scientific_name = bird_stats.bird_scientific_name)  (cost=0.35 rows=1) (never executed)
        -> Single-row index lookup on u using PRIMARY (uid = bird_stats.user_id)  (cost=0.35 rows=1) (never executed)
 |
```

Design 3: Covering Composite Index on (bird_scientific_name, user_id, event_id)

Even though our query only selects bird_scientific_name and user_id from the Event table (with an aggregate count), adding the primary key column (event_id) to the composite index can make it a covering index for the subquery. A covering index means that the query can retrieve all needed columns directly from the index without having to access the full table rows.

SQL:
CREATE INDEX idx_event_covering ON Event(bird_scientific_name, user_id, event_id);

Result: time - 0.0922



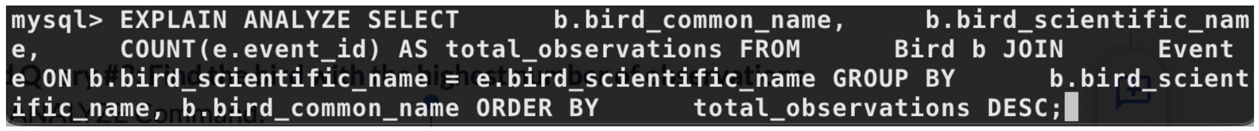
```
| -> Limit: 15 row(s)  (cost=3.43 rows=1) (actual time=0.0922..0.0922 rows=0 loops=1)
    -> Nested loop inner join  (cost=3.43 rows=1) (actual time=0.0912..0.0912 rows=0 loops=1)
        -> Nested loop inner join  (cost=3.08 rows=1) (actual time=0.0908..0.0908 rows=0 loops=1)
            -> Filter: (bird_stats.rnk = 1)  (cost=2.73 rows=1) (actual time=0.0897..0.0897 rows=0 loops=1)
                -> Table scan on bird_stats  (cost=2.5..2.5 rows=0) (actual time=0.0886..0.0886 rows=0 loops=1)
                    -> Materialize  (cost=0..0 rows=0) (actual time=0.088..0.088 rows=0 loops=1)
                        -> Window aggregate: rank() OVER (PARTITION BY e.bird_scientific_name ORDER BY observation_count desc )   (actual time=0.048..0.048 rows=0 loops=1)
                            -> Sort: e.bird_scientific_name, observation_count DESC  (actual time=0.0476..0.0476 rows=0 loops=1)
                                -> Table scan on <temporary>  (actual time=0.0374..0.0374 rows=0 loops=1)
                                    -> Aggregate using temporary table  (actual time=0.0359..0.0359 rows=0 loops=1)
                                        -> Covering index scan on e using idx_event_covering  (cost=0.6 rows=1) (actual time=0.0283..0.0283 rows=0 loops=1)
            -> Single-row index lookup on b using PRIMARY (bird_scientific_name = bird_stats.bird_scientific_name)  (cost=0.35 rows=1) (never executed)
        -> Single-row index lookup on u using PRIMARY (uid = bird_stats.user_id)  (cost=0.35 rows=1) (never executed)
 |
```

Report:
The best index for this query is idx_event_covering (bird_scientific_name, user_id, event_id) because it is a covering index, meaning MySQL can get all needed data directly from the index without accessing the table. This reduces disk I/O and improves query speed.
Compared to other indexes, idx_event_covering provides the fastest execution time because:

- It helps with GROUP BY and JOIN efficiently.
- It avoids extra table lookups (no need to fetch event_id separately).
- It reduces query time significantly (0.0922s, the best performance).

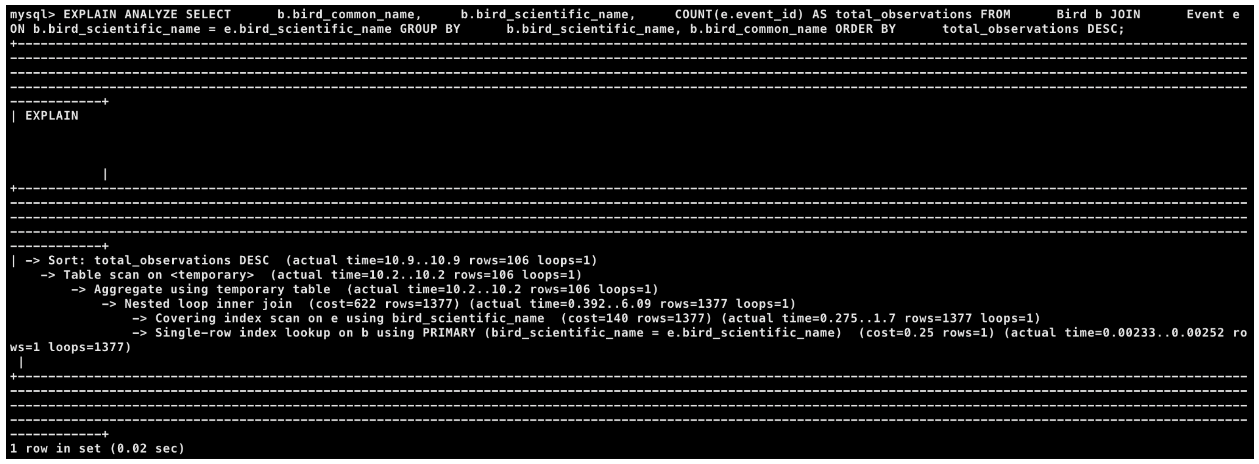
Advanced Query #3: Find the bird with the highest number of observations

EXPLAIN ANALYZE Command:



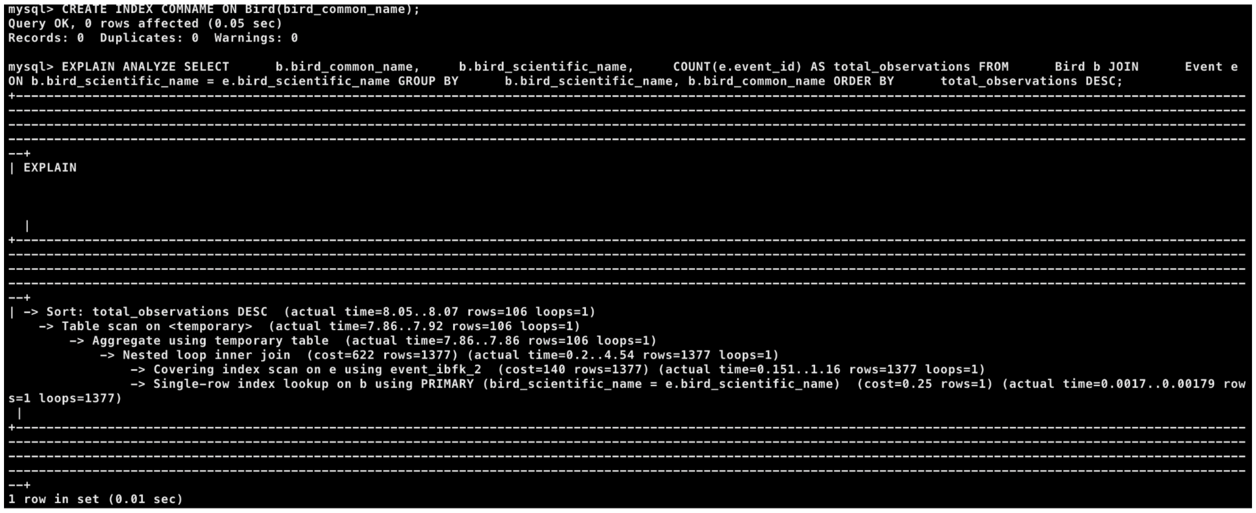
```
mysql> EXPLAIN ANALYZE SELECT      b.bird_common_name,     b.bird_scientific_nam
e,     COUNT(e.event_id) AS total_observations FROM      Bird b JOIN      Event
e ON b.bird_scientific_name = e.bird_scientific_name GROUP BY      b.bird_scient
ific_name, b.bird_common_name ORDER BY      total_observations DESC;
```

Result Before Adding Indexing



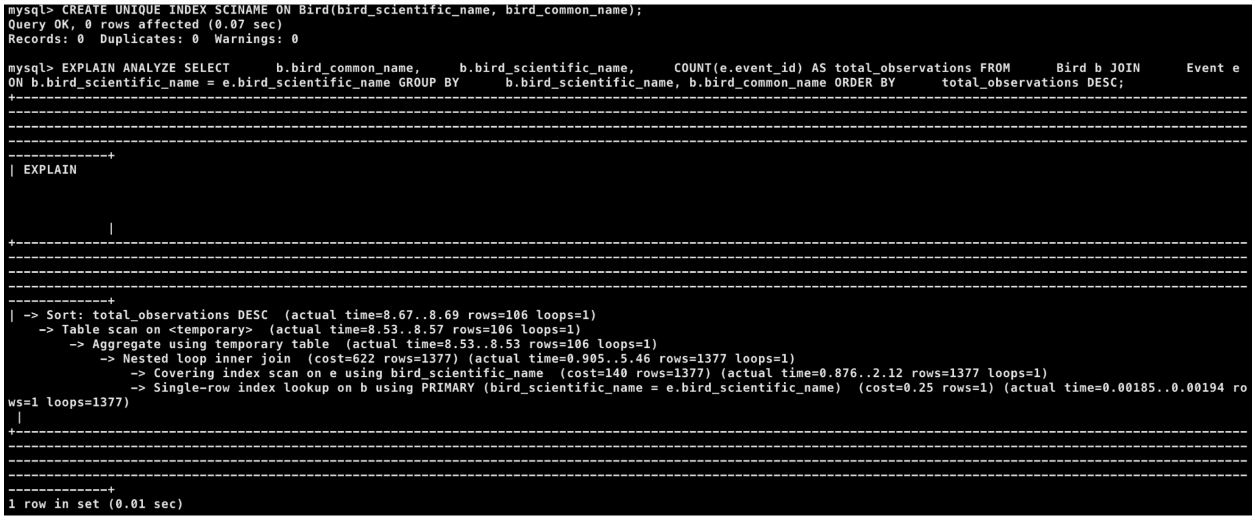
```
mysql> EXPLAIN ANALYZE SELECT      b.bird_common_name,     b.bird_scientific_name,     COUNT(e.event_id) AS total_observations FROM      Bird b JOIN      Event e
ON b.bird_scientific_name = e.bird_scientific_name GROUP BY      b.bird_scientific_name, b.bird_common_name ORDER BY      total_observations DESC;
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
------------+
| EXPLAIN



            |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
------------+
| -> Sort: total_observations DESC  (actual time=10.9..10.9 rows=106 loops=1)
    -> Table scan on <temporary>  (actual time=10.2..10.2 rows=106 loops=1)
        -> Aggregate using temporary table  (actual time=10.2..10.2 rows=106 loops=1)
            -> Nested loop inner join  (cost=622 rows=1377) (actual time=0.392..6.09 rows=1377 loops=1)
                -> Covering index scan on e using bird_scientific_name  (cost=140 rows=1377) (actual time=0.275..1.7 rows=1377 loops=1)
                -> Single-row index lookup on b using PRIMARY (bird_scientific_name = e.bird_scientific_name)  (cost=0.25 rows=1) (actual time=0.00233..0.00252 ro
ws=1 loops=1377)
 |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
------------+
1 row in set (0.02 sec)
```

Result for “CREATE UNIQUE INDEX COMNAME ON Bird(bird_common_name);”



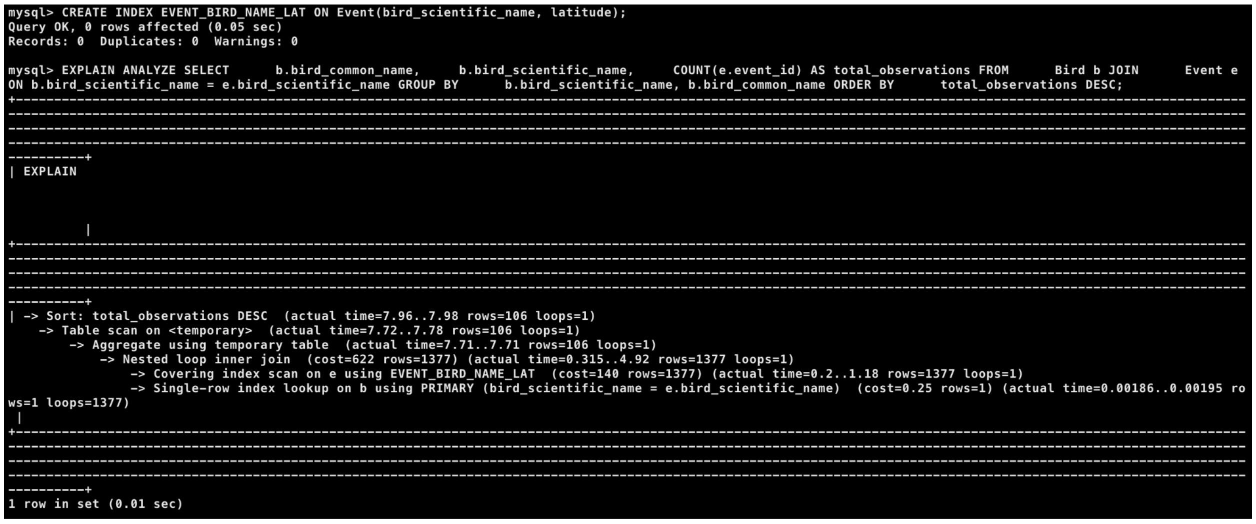
```
mysql> CREATE INDEX COMNAME ON Bird(bird_common_name);
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE SELECT      b.bird_common_name,     b.bird_scientific_name,     COUNT(e.event_id) AS total_observations FROM      Bird b JOIN      Event e
ON b.bird_scientific_name = e.bird_scientific_name GROUP BY      b.bird_scientific_name, b.bird_common_name ORDER BY      total_observations DESC;
+---------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
--+
| EXPLAIN



  |
+---------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
--+
| -> Sort: total_observations DESC  (actual time=8.05..8.07 rows=106 loops=1)
    -> Table scan on <temporary>  (actual time=7.86..7.92 rows=106 loops=1)
        -> Aggregate using temporary table  (actual time=7.86..7.86 rows=106 loops=1)
            -> Nested loop inner join  (cost=622 rows=1377) (actual time=0.2..4.54 rows=1377 loops=1)
                -> Covering index scan on e using event_ibfk_2  (cost=140 rows=1377) (actual time=0.151..1.16 rows=1377 loops=1)
                -> Single-row index lookup on b using PRIMARY (bird_scientific_name = e.bird_scientific_name)  (cost=0.25 rows=1) (actual time=0.0017..0.00179 row
s=1 loops=1377)
 |
+---------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
--+
1 row in set (0.01 sec)
```

Result for "CREATE UNIQUE INDEX SCINAME ON Bird(bird_scientific_name, bird_common_name);"



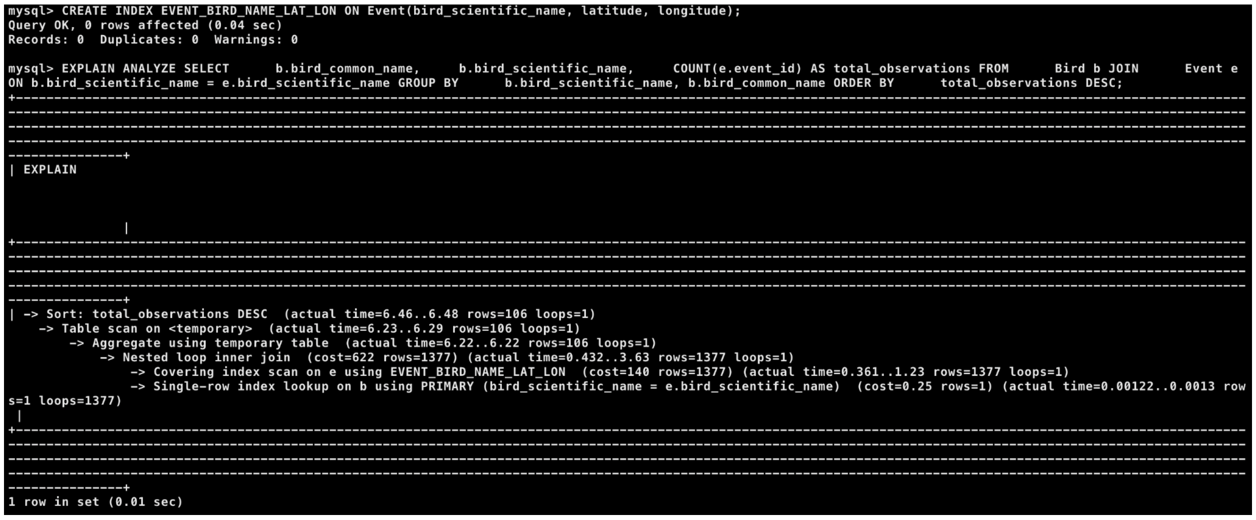
```
mysql> CREATE UNIQUE INDEX SCINAME ON Bird(bird_scientific_name, bird_common_name);
Query OK, 0 rows affected (0.07 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE SELECT      b.bird_common_name,     b.bird_scientific_name,     COUNT(e.event_id) AS total_observations FROM      Bird b JOIN      Event e
ON b.bird_scientific_name = e.bird_scientific_name GROUP BY      b.bird_scientific_name, b.bird_common_name ORDER BY      total_observations DESC;
+---------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------+
| EXPLAIN



             |
+---------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------+
| -> Sort: total_observations DESC  (actual time=8.67..8.69 rows=106 loops=1)
    -> Table scan on <temporary>  (actual time=8.53..8.57 rows=106 loops=1)
        -> Aggregate using temporary table  (actual time=8.53..8.53 rows=106 loops=1)
            -> Nested loop inner join  (cost=622 rows=1377) (actual time=0.905..5.46 rows=1377 loops=1)
                -> Covering index scan on e using bird_scientific_name  (cost=140 rows=1377) (actual time=0.876..2.12 rows=1377 loops=1)
                -> Single-row index lookup on b using PRIMARY (bird_scientific_name = e.bird_scientific_name)  (cost=0.25 rows=1) (actual time=0.00185..0.00194 ro
ws=1 loops=1377)
 |
+---------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------+
1 row in set (0.01 sec)
```

Result for "CREATE INDEX EVENT_BIRD_NAME_LAT ON Event(bird_scientific_name, latitude);"



```
mysql> CREATE INDEX EVENT_BIRD_NAME_LAT ON Event(bird_scientific_name, latitude);
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE SELECT      b.bird_common_name,     b.bird_scientific_name,     COUNT(e.event_id) AS total_observations FROM      Bird b JOIN      Event e
ON b.bird_scientific_name = e.bird_scientific_name GROUP BY      b.bird_scientific_name, b.bird_common_name ORDER BY      total_observations DESC;
+---------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------+
| EXPLAIN



          |
+---------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------+
| -> Sort: total_observations DESC  (actual time=7.96..7.98 rows=106 loops=1)
    -> Table scan on <temporary>  (actual time=7.72..7.78 rows=106 loops=1)
        -> Aggregate using temporary table  (actual time=7.71..7.71 rows=106 loops=1)
            -> Nested loop inner join  (cost=622 rows=1377) (actual time=0.315..4.92 rows=1377 loops=1)
                -> Covering index scan on e using EVENT_BIRD_NAME_LAT  (cost=140 rows=1377) (actual time=0.2..1.18 rows=1377 loops=1)
                -> Single-row index lookup on b using PRIMARY (bird_scientific_name = e.bird_scientific_name)  (cost=0.25 rows=1) (actual time=0.00186..0.00195 ro
ws=1 loops=1377)
 |
+---------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------+
1 row in set (0.01 sec)
```

Result for "CREATE INDEX EVENT_BIRD_NAME_LAT_LON ON Event(bird_scientific_name, latitude, longitude);"

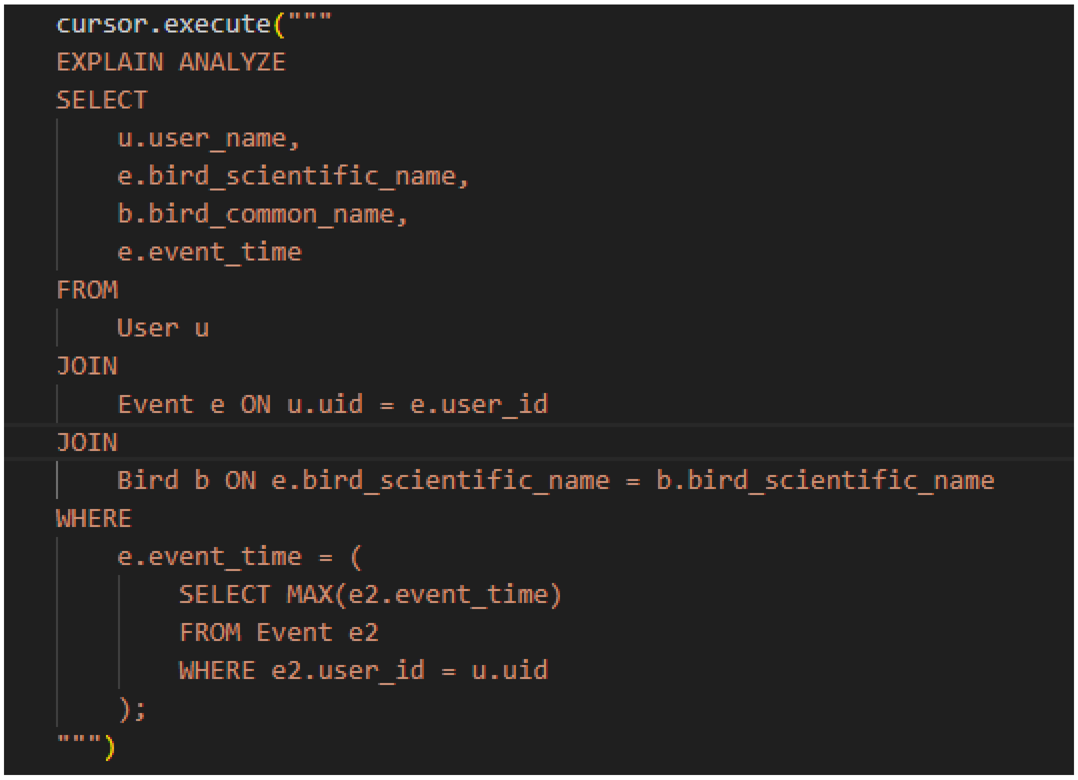


```
mysql> CREATE INDEX EVENT_BIRD_NAME_LAT_LON ON Event(bird_scientific_name, latitude, longitude);
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE SELECT      b.bird_common_name,     b.bird_scientific_name,     COUNT(e.event_id) AS total_observations FROM      Bird b JOIN      Event e
ON b.bird_scientific_name = e.bird_scientific_name GROUP BY      b.bird_scientific_name, b.bird_common_name ORDER BY      total_observations DESC;
+---------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------+
| EXPLAIN



               |
+---------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------+
| -> Sort: total_observations DESC  (actual time=6.46..6.48 rows=106 loops=1)
    -> Table scan on <temporary>  (actual time=6.23..6.29 rows=106 loops=1)
        -> Aggregate using temporary table  (actual time=6.22..6.22 rows=106 loops=1)
            -> Nested loop inner join  (cost=622 rows=1377) (actual time=0.432..3.63 rows=1377 loops=1)
                -> Covering index scan on e using EVENT_BIRD_NAME_LAT_LON  (cost=140 rows=1377) (actual time=0.361..1.23 rows=1377 loops=1)
                -> Single-row index lookup on b using PRIMARY (bird_scientific_name = e.bird_scientific_name)  (cost=0.25 rows=1) (actual time=0.00122..0.0013 row
s=1 loops=1377)
 |
+---------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------+
1 row in set (0.01 sec)
```

Report

The final index design chosen for optimizing this query is a composite index on (bird_scientific_name, latitude, longitude) in the Event table. This index reduce the runtime from 10.9 to 6.5. It allows the database to quickly access the rows for both records in Event and Bird table by matching the criteria on bird scientific name, longitude, and latitude, so it can identify the bird with the highest number of observations faster. The design was selected because it has shorter execution time than indices on bird table due to its limit number of columns and the index on event table with only the longitude.

Advanced Query #4: List the most recent bird observed by the user (one row per user)
EXPLAIN ANALYZE command



```
cursor.execute("""
EXPLAIN ANALYZE
SELECT
    u.user_name,
    e.bird_scientific_name,
    b.bird_common_name,
    e.event_time
FROM
    User u
JOIN
    Event e ON u.uid = e.user_id
JOIN
    Bird b ON e.bird_scientific_name = b.bird_scientific_name
WHERE
    e.event_time = (
        SELECT MAX(e2.event_time)
        FROM Event e2
        WHERE e2.user_id = u.uid
    );
""")
```

Result before adding index:
Result: ('-> Nested loop inner join (cost=1104 rows=1377) (actual time=0.193..201 rows=474 loops=1)\n -> Nested loop inner join (cost=622 rows=1377) (actual time=0.0324..0.626 rows=1377 loops=1)\n -> Table scan on e (cost=140 rows=1377) (actual time=0.0244..0.222 rows=1377 loops=1)\n -> Single-row index lookup on b using PRIMARY (bird_scientific_name=e.bird_scientific_name) (cost=0.25 rows=1) (actual time=190e-6..207e-6 rows=1 loops=1377)\n -> Filter: (e.event_time = (select #2)) (cost=0.25 rows=1) (actual time=0.145..0.145 rows=0.344 loops=1377)\n -> Single-row

index lookup on u using PRIMARY (uid=e.user_id) (cost=0.25 rows=1) (actual time=485e-6..502e-6 rows=1 loops=1377)\n -> Select #2 (subquery in condition; dependent)\n -> Aggregate: max(e2.event_time) (cost=30 rows=1) (actual time=0.144..0.144 rows=1 loops=1377)\n -> Filter: (e2.user_id = u.uid) (cost=16.3 rows=138) (actual time=0.0419..0.144 rows=3.76 loops=1377)\n -> Table scan on e2 (cost=16.3 rows=1377) (actual time=0.014..0.111 rows=1377 loops=1377)\n',)

Result after adding CREATE INDEX idx_bird_name ON Bird(bird_scientific_name):
Result: ('-> Nested loop inner join (cost=1104 rows=1377) (actual time=0.184..203 rows=474 loops=1)\n -> Nested loop inner join (cost=622 rows=1377) (actual time=0.0261..0.621 rows=1377 loops=1)\n -> Table scan on e (cost=140 rows=1377) (actual time=0.0191..0.209 rows=1377 loops=1)\n -> Single-row index lookup on b using PRIMARY (bird_scientific_name=e.bird_scientific_name) (cost=0.25 rows=1) (actual time=197e-6..213e-6 rows=1 loops=1377)\n -> Filter: (e.event_time = (select #2)) (cost=0.25 rows=1) (actual time=0.147..0.147 rows=0.344 loops=1377)\n -> Single-row index lookup on u using PRIMARY (uid=e.user_id) (cost=0.25 rows=1) (actual time=485e-6..503e-6 rows=1 loops=1377)\n -> Select #2 (subquery in condition; dependent)\n -> Aggregate: max(e2.event_time) (cost=30 rows=1) (actual time=0.146..0.146 rows=1 loops=1377)\n -> Filter: (e2.user_id = u.uid) (cost=16.3 rows=138) (actual time=0.0424..0.145 rows=3.76 loops=1377)\n -> Table scan on e2 (cost=16.3 rows=1377) (actual time=0.0142..0.112 rows=1377 loops=1377)\n',)

Result after adding CREATE INDEX idx_event_user_time ON Event(user_id, event_time):
Result: ('-> Nested loop inner join (cost=812 rows=1015) (actual time=0.0858..5.22 rows=474 loops=1)\n -> Nested loop inner join (cost=456 rows=1015) (actual time=0.0801..4.88 rows=474 loops=1)\n -> Table scan on u (cost=101 rows=1000) (actual time=0.0439..0.172 rows=1000 loops=1)\n -> Filter: (e.event_time = (select #2)) (cost=0.254 rows=1.01) (actual time=0.00448..0.00463 rows=0.474 loops=1000)\n -> Index lookup on e using idx_event_user_time (user_id=u.uid, event_time=(select #2)) (cost=0.254 rows=1.01) (actual time=0.00341..0.00352 rows=0.474 loops=1000)\n -> Select #2 (subquery in condition; dependent)\n -> Aggregate: max(e2.event_time) (cost=1.22 rows=1) (actual time=0.00156..0.00158 rows=1 loops=1941)\n -> Covering index lookup on e2 using idx_event_user_time (user_id=u.uid) (cost=0.922 rows=2.95) (actual time=0.00104..0.00138 rows=2.14 loops=1941)\n -> Single-row index lookup on b using PRIMARY (bird_scientific_name=e.bird_scientific_name) (cost=0.25 rows=1) (actual time=607e-6..625e-6 rows=1 loops=474)\n',)

Result after adding CREATE INDEX idx_event_bird ON Event(bird_scientific_name):

Result: ('-> Nested loop inner join (cost=1104 rows=1377) (actual time=0.182..200 rows=474 loops=1)\n -> Nested loop inner join (cost=622 rows=1377) (actual time=0.0243..0.607 rows=1377 loops=1)\n -> Table scan on e (cost=140 rows=1377) (actual time=0.0171..0.199 rows=1377 loops=1)\n -> Single-row index lookup on b using PRIMARY (bird_scientific_name=e.bird_scientific_name) (cost=0.25 rows=1) (actual time=193e-6..209e-6 rows=1 loops=1377)\n -> Filter: (e.event_time = (select #2)) (cost=0.25 rows=1) (actual time=0.144..0.144 rows=0.344 loops=1377)\n -> Single-row index lookup on u using PRIMARY (uid=e.user_id) (cost=0.25 rows=1) (actual time=454e-6..471e-6 rows=1 loops=1377)\n -> Select #2 (subquery in condition; dependent)\n -> Aggregate: max(e2.event_time) (cost=30 rows=1) (actual time=0.143..0.143 rows=1 loops=1377)\n -> Filter: (e2.user_id = u.uid) (cost=16.3 rows=138) (actual time=0.0417..0.143 rows=3.76 loops=1377)\n -> Table scan on e2 (cost=16.3 rows=1377) (actual time=0.0139..0.11 rows=1377 loops=1377)\n',)

Report:
The final index design chosen for optimizing this query is a composite index on `(user_id, event_time)` in the `Event` table. This index led to a substantial reduction in query cost, dropping from 1104 to 812, by efficiently supporting both the main query's filtering condition and the subquery's aggregation (`MAX(event_time)`). It allowed the optimizer to use index lookups and covering index access patterns, significantly reducing the need for full table scans.

Other indexes, such as those on `bird_scientific_name` in the `Event` or `Bird` tables, had no measurable effect on query performance. This is likely because the join with the `Bird` table already benefits from an existing primary key, and the core bottleneck of the query lies in the user-specific filtering and aggregation.