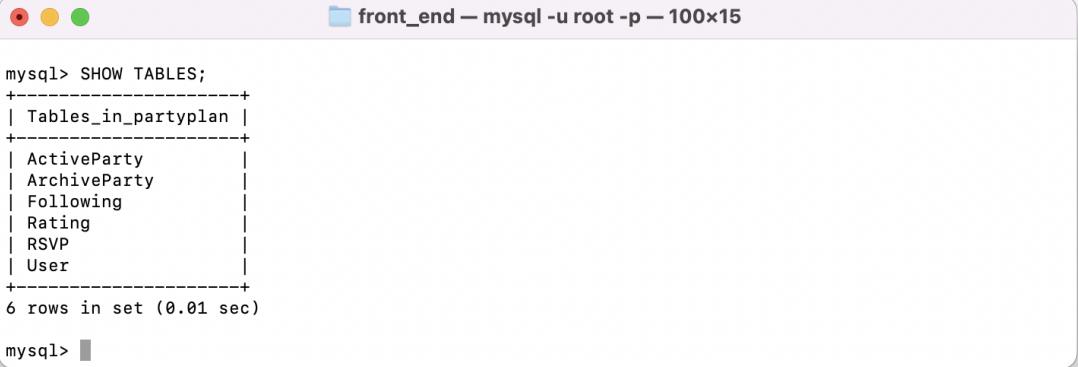


Database Implementation



A screenshot of a terminal window titled "front_end — mysql -u root -p — 100x15". The window displays the following MySQL command and its results:

```
mysql> SHOW TABLES;
+---------------------+
| Tables_in_partyplan |
+-----+
| ActiveParty         |
| ArchiveParty        |
| Following           |
| Rating              |
| RSVP                |
| User                |
+-----+
6 rows in set (0.01 sec)

mysql>
```

Proof of implementing the database tables locally or on GCP, you should provide a screenshot of the connection (i.e. showing your terminal information).

DDL Commands:

```
CREATE TABLE User (
    userID INT UNSIGNED NOT NULL PRIMARY KEY,
    first VARCHAR(63) NOT NULL,
    last VARCHAR(63) NOT NULL,
    email VARCHAR(63) NOT NULL,
    phone_number BIGINT UNSIGNED NOT NULL
);
```

```
CREATE TABLE ActiveParty (
    partyID INT UNSIGNED NOT NULL PRIMARY KEY,
    userID INT UNSIGNED NOT NULL,
    latitude REAL NOT NULL,
    longitude REAL NOT NULL,
    tags JSON,
    capacity INT NOT NULL,
    start_time DATE NOT NULL,
    end_time DATE NOT NULL,
    FOREIGN KEY(userID) REFERENCES User(userID) ON DELETE CASCADE
);
```

```
CREATE TABLE ArchiveParty (
    partyID INT UNSIGNED NOT NULL PRIMARY KEY,
    userID INT UNSIGNED NOT NULL,
    latitude REAL NOT NULL,
    longitude REAL NOT NULL,
    tags JSON,
    capacity INT NOT NULL,
    start_time DATE NOT NULL,
    end_time DATE NOT NULL,
    FOREIGN KEY(userID) REFERENCES User(userID) ON DELETE CASCADE
);

CREATE TABLE Rating (
    ratingID INT UNSIGNED NOT NULL PRIMARY KEY,
    safety INT UNSIGNED NOT NULL,
    fun INT UNSIGNED NOT NULL,
    userID INT UNSIGNED NOT NULL,
    FOREIGN KEY(userID) REFERENCES User(userID) ON DELETE CASCADE
);

CREATE TABLE Following (
    Followingid INT UNSIGNED NOT NULL PRIMARY KEY,
    userID INT UNSIGNED NOT NULL,
    followsID INT UNSIGNED NOT NULL,
    FOREIGN KEY(userID) REFERENCES User(userID) ON DELETE CASCADE,
    FOREIGN KEY(followsID) REFERENCES User(userID) ON DELETE CASCADE
);

CREATE TABLE RSVP (
    RSVPid INT UNSIGNED NOT NULL PRIMARY KEY,
    partyID INT UNSIGNED NOT NULL,
    userID INT UNSIGNED NOT NULL,
    reponse CHAR(11),
    FOREIGN KEY(userID) REFERENCES User(userID) ON DELETE CASCADE,
    FOREIGN KEY(partyID) REFERENCES ActiveParty(partyID) ON DELETE CASCADE
);
```

```
front_end — mysql -u root -p — 105x49
mysql> SELECT COUNT(*) FROM User;
+-----+
| COUNT(*) |
+-----+
| 3000 |
+-----+
1 row in set (0.04 sec)

mysql> SELECT COUNT(*) FROM ActiveParty;
+-----+
| COUNT(*) |
+-----+
| 1000 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT COUNT(*) FROM ArchiveParty;
+-----+
| COUNT(*) |
+-----+
| 1000 |
+-----+
1 row in set (0.01 sec)

mysql> SELECT COUNT(*) FROM Rating;
+-----+
| COUNT(*) |
+-----+
| 1000 |
+-----+
1 row in set (0.01 sec)

mysql> SELECT COUNT(*) FROM Following;
+-----+
| COUNT(*) |
+-----+
| 2000 |
+-----+
1 row in set (0.02 sec)

mysql> SELECT COUNT(*) FROM RSVP;
+-----+
| COUNT(*) |
+-----+
| 1000 |
+-----+
1 row in set (0.00 sec)

mysql>
```

Proof for inserting at least 1000 rows in the tables.

Advanced Queries

Query 1:

```
SELECT DISTINCT *
FROM ActiveParty NATURAL JOIN (SELECT u.userID, AVG(safety) as safety, AVG(fun) as fun
                                FROM User u JOIN Rating r ON (u.userID = r.userID)
                                GROUP BY u.userID) as temp
WHERE temp.safety >= 3 AND temp.fun >= 3.5
LIMIT 15;
```

```

front_end — mysql -u root -p — 182x29

mysql> SELECT DISTINCT *
->   FROM ActiveParty NATURAL JOIN (SELECT u.userID, AVG(safety) as safety, AVG(fun) as fun
->     FROM User JOIN Rating r ON (u.userID = r.userID)
->   GROUP BY u.userID) as temp
-> WHERE temp.safety >= 3 AND temp.fun >= 3.5
-> LIMIT 15;
+-----+-----+-----+-----+-----+-----+
| userID | partyID | latitude | longitude | tags | capacity | start_time | end_time | safety | fun |
+-----+-----+-----+-----+-----+-----+
| 294752 | 133996 | 48.82457889886853 | -88.29597727538774 | ["BYOB", "RSO"] | 98 | 2022-10-21 | 2022-10-21 | 3.0000 | 4.0000 |
| 294751 | 351348 | 48.88976477991896 | -88.31552189162905 | ["BYOB", "Costume", "Chill", "RSO", "Open House"] | 167 | 2022-10-21 | 2022-10-21 | 3.0000 | 4.0000 |
| 129316 | 100441 | 48.20277494077739 | -88.21292503406845 | ["BYOB", "Open House"] | 51 | 2022-10-21 | 2022-10-21 | 3.0000 | 4.0000 |
| 998071 | 464684 | 48.138630885373189 | -88.21385023885347 | ["Greek", "Open House", "Costume", "RSO", "BYOB"] | 54 | 2022-10-21 | 2022-10-21 | 3.0000 | 5.0000 |
| 742920 | 942736 | 48.10487685334384 | -88.19853657841443 | [] | 41 | 2022-10-21 | 2022-10-21 | 3.0000 | 4.0000 |
| 985764 | 288348 | 48.1782499954536596 | -88.316642568989532 | ["Costume", "Greek", "BYOB"] | 115 | 2022-10-21 | 2022-10-21 | 5.0000 | 4.0000 |
| 256578 | 323179 | 48.059276962158314 | -88.21243180219981 | ["RSO", "Chill", "BYOB", "Costume"] | 57 | 2022-10-21 | 2022-10-21 | 5.0000 | 5.0000 |
| 617980 | 413689 | 48.130807133979715 | -88.2741187834088 | ["Open House", "Costume", "RSO"] | 63 | 2022-10-21 | 2022-10-21 | 3.0000 | 4.0000 |
| 419387 | 661996 | 48.130807133979715 | -88.2741187834088 | ["Open House", "BYOB", "Costume", "Chill"] | 173 | 2022-10-21 | 2022-10-21 | 4.0000 | 4.0000 |
| 388853 | 334181 | 48.036438612162286 | -88.3682214126464 | ["Chill"] | 79 | 2022-10-21 | 2022-10-21 | 3.0000 | 4.0000 |
| 798843 | 388853 | 48.0838461787861 | -88.3682214126464 | ["Chill"] | 56 | 2022-10-21 | 2022-10-21 | 3.0000 | 4.0000 |
| 133882 | 128108 | 48.0838461787861 | -88.3682214126464 | ["Chill", "RSO"] | 109 | 2022-10-21 | 2022-10-21 | 3.0000 | 4.0000 |
| 172432 | 647562 | 48.09165310165715 | -88.26185051992628 | ["Open House", "RSO", "Greek", "BYOB", "Costume"] | 96 | 2022-10-21 | 2022-10-21 | 4.0000 | 5.0000 |
| 548287 | 568572 | 48.096986249381894 | -88.2984464616382 | ["RSO", "Costume", "Chill"] | 27 | 2022-10-21 | 2022-10-21 | 5.0000 | 4.0000 |
| 133840 | 493631 | 48.01865478132756 | -88.1722235927832 | ["RSO", "BYOB", "Open House"] | 196 | 2022-10-21 | 2022-10-21 | 4.0000 | 4.0000 |
+-----+
15 rows in set (0.00 sec)

mysql> 

```

Query 2:

```

SELECT partyID, userID, tags, first, last
FROM ActiveParty NATURAL JOIN User
WHERE TAGS LIKE "%C%" OR FIRST LIKE "%C%" OR LAST LIKE "%C%"

UNION

```

```

SELECT partyID, userID, tags, first, last
FROM ArchiveParty NATURAL JOIN User
WHERE TAGS LIKE "%C%" OR FIRST LIKE "%C%" OR LAST LIKE "%C%"
LIMIT 15;

```

```

front_end — mysql -u root -p — 182x33

mysql> SELECT partyID, userID, tags, first, last
->   FROM ActiveParty NATURAL JOIN User
-> WHERE TAGS LIKE "%C%" OR FIRST LIKE "%C%" OR LAST LIKE "%C%"
->
-> UNION
->
->   SELECT partyID, userID, tags, first, last
->   FROM ArchiveParty NATURAL JOIN User
-> WHERE TAGS LIKE "%C%" OR FIRST LIKE "%C%" OR LAST LIKE "%C%"
-> ORDER BY partyID
-> LIMIT 15;
+-----+-----+-----+-----+-----+
| partyID | userID | tags | first | last |
+-----+-----+-----+-----+-----+
| 181476 | 193216 | ["Open House", "Costume"] | Vere | Latty |
| 100000 | 998468 | ["RSO", "Costume"] | Fionna | Sherrill |
| 106340 | 99369 | [] | Mais | Cul |
| 106393 | 562507 | ["BYOB", "Greek", "Open House", "Costume"] | Ulla | Truwell |
| 186449 | 264839 | ["RSO", "BYOB", "Greek", "Costume"] | Elspeth | Lail |
| 108386 | 573079 | ["RSO", "Chill"] | Avrit | Jannel |
| 109315 | 638739 | ["RSO", "BYOB", "Open House", "Greek"] | Vernice | Karyn |
| 109562 | 435815 | ["Costume", "RSO", "BYOB"] | Katherine | Chemesh |
| 109854 | 445700 | ["Costume", "Open House"] | Collen | Middlesworth |
| 111028 | 718832 | ["Costume", "BYOB", "Greek", "Chill", "Open House"] | Jaymee | Tasia |
| 111343 | 730564 | ["Costume", "Greek", "Chill", "Open House", "RSO"] | Catherine | Reid |
| 111337 | 956408 | ["Chill", "RSO", "Greek", "BYOB", "Open House"] | Mikayla | Property |
| 111587 | 268921 | ["Chill"] | Agace | Camila |
| 111611 | 941297 | [] | Romonda | Fawcett |
| 112825 | 976349 | ["RSO", "BYOB", "Greek", "Open House", "Chill"] | Caritta | Weld |
+-----+
15 rows in set (0.01 sec)

mysql> 

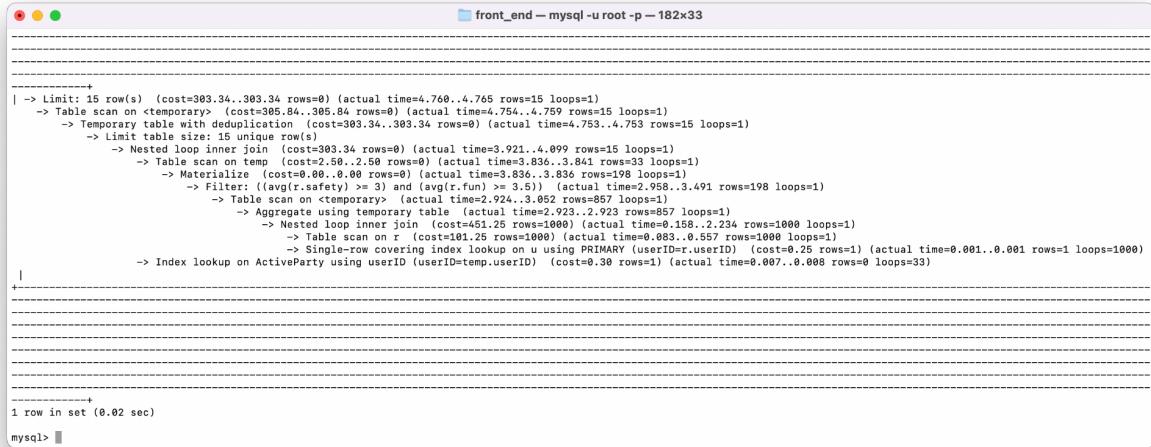
```

Indexing Analysis

For Query 1:

First index

```
CREATE INDEX rating_idx_1
ON Rating (ratingID);
```



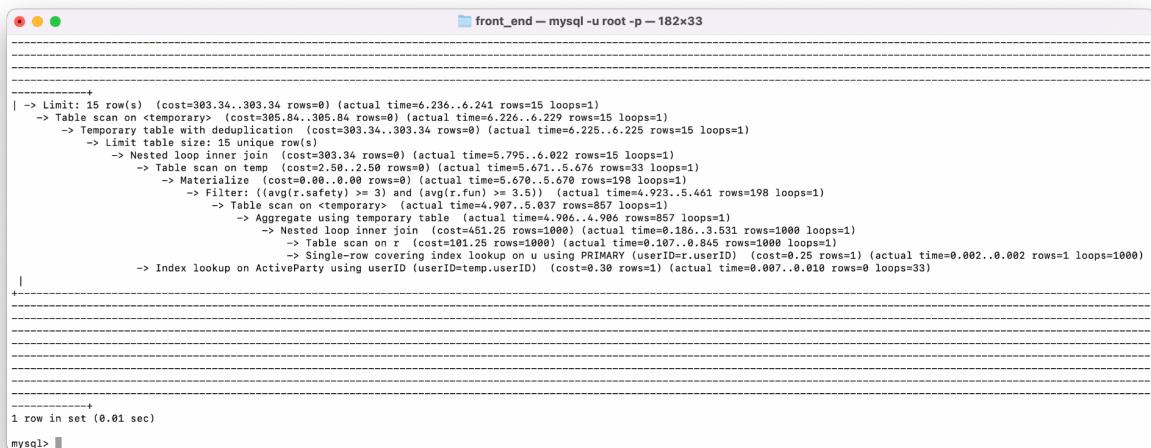
```
+-----+
| -> Limit: 15 row(s) (cost=303.34..303.34 rows=0) (actual time=4.760..4.765 rows=15 loops=1)
  -> Table scan on <temporary> (cost=305.84..305.84 rows=0) (actual time=4.754..4.759 rows=15 loops=1)
    -> Temporary table with deduplication (cost=303.34..303.34 rows=0) (actual time=4.753..4.753 rows=15 loops=1)
      -> Limit table size: 15 unique row(s)
        -> Nested loop inner join (cost=303.34..303.34 rows=0) (actual time=3.921..4.092 rows=15 loops=1)
          -> Table scan on temp (cost=2.58..2.58 rows=0) (actual time=3.834..3.941 rows=33 loops=1)
            -> Materialize (cost=0.00..0.00 rows=0) (actual time=3.836..3.836 rows=198 loops=1)
              -> Filter: ((avg(r.safety) >= 3) and (avg(r.fun) >= 3.5)) (actual time=2.958..3.491 rows=198 loops=1)
                -> Table scan on <temporary> (actual time=2.924..3.052 rows=857 loops=1)
                  -> Aggregate using temporary table (actual time=2.923..2.923 rows=857 loops=1)
                    -> Nested loop inner join (cost=451.25 rows=1000) (actual time=0.158..2.234 rows=1000 loops=1)
                      -> Table scan on r (cost=101.25 rows=1000) (actual time=0.083..0.057 rows=1000 loops=1)
                        -> Single-row covering index lookup on u using PRIMARY (userID=r.userID) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1000)
                          -> Index lookup on ActiveParty using userID (userID=temp.userID) (cost=0.30 rows=1) (actual time=0.007..0.008 rows=0 loops=33)
|
+-----+
1 row in set (0.02 sec)

mysql> |
```

Second index

```
CREATE INDEX rating_idx_2
ON Rating (safety ASC, fun ASC);
```

Reasoning for change: Since we search against safety and fun in the query, it makes sense to create an index for these.



```
+-----+
| -> Limit: 15 row(s) (cost=303.34..303.34 rows=0) (actual time=6.236..6.241 rows=15 loops=1)
  -> Table scan on <temporary> (cost=305.84..305.84 rows=0) (actual time=6.226..6.229 rows=15 loops=1)
    -> Temporary table with deduplication (cost=303.34..303.34 rows=0) (actual time=6.225..6.225 rows=15 loops=1)
      -> Limit table size: 15 unique row(s)
        -> Nested loop inner join (cost=303.34..303.34 rows=0) (actual time=5.795..6.022 rows=15 loops=1)
          -> Table scan on <temporary> (cost=2.58..2.58 rows=0) (actual time=5.671..5.676 rows=33 loops=1)
            -> Materialize (cost=0.00..0.00 rows=0) (actual time=5.679..5.679 rows=198 loops=1)
              -> Filter: ((avg(r.safety) >= 3) and (avg(r.fun) >= 3.5)) (actual time=4.923..5.461 rows=198 loops=1)
                -> Table scan on <temporary> (actual time=4.906..4.906 rows=857 loops=1)
                  -> Aggregate using temporary table (actual time=4.906..4.906 rows=857 loops=1)
                    -> Nested loop inner join (cost=451.25 rows=1000) (actual time=0.186..3.531 rows=1000 loops=1)
                      -> Table scan on r (cost=101.25 rows=1000) (actual time=0.107..0.045 rows=1000 loops=1)
                        -> Single-row covering index lookup on u using PRIMARY (userID=r.userID) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1000)
                          -> Index lookup on ActiveParty using userID (userID=temp.userID) (cost=0.30 rows=1) (actual time=0.007..0.010 rows=0 loops=33)
|
+-----+
1 row in set (0.01 sec)

mysql> |
```

Third index

```
CREATE INDEX rating_idx_3
ON Rating (safety DESC, fun DESC);
```

Reasoning for change: we are searching for users with only above a certain safety/fun threshold, so it makes sense to make them descending.

```
front_end — mysql -u root -p — 182x33

+-----+
| --> Limit: 15 row(s) (cost=303.34..303.34 rows=0) (actual time=5.964..5.970 rows=15 loops=1)
|   --> Table scan on <temporary> (cost=305.84..305.84 rows=0) (actual time=5.953..5.957 rows=15 loops=1)
|     --> Temporary table with deduplication (cost=303.34..303.34 rows=0) (actual time=5.952..5.952 rows=15 loops=1)
|       --> Limit table size: 15 unique row(s)
|         --> Nested loop inner join (cost=303.34..303.34 rows=0) (actual time=5.497..5.915 rows=15 loops=1)
|           --> Table scan on temp (cost=2.08..2.50 rows=0) (actual time=5.315..5.374 rows=33 loops=1)
|             --> Materialize (cost=0.00..8.00 rows=0) (actual time=5.368..5.368 rows=198 loops=1)
|               --> Filter: ((avg(r.safety) >= 3) and (avg(r.fun) >= 3.5)) (actual time=4.677..5.286 rows=198 loops=1)
|                 --> Table scan on <temporary> (actual time=4.661..4.790 rows=857 loops=1)
|                   --> Aggregate using temporary table (actual time=4.659..4.659 rows=857 loops=1)
|                     --> Nested loop inner join (cost=451.25 rows=1000) (actual time=0.198..3.513 rows=1000 loops=1)
|                       --> Table scan on r (cost=101.25 rows=1000) (actual time=0.108..0.861 rows=1000 loops=1)
|                         --> Single-row covering index lookup on u using PRIMARY (userID=r.userID) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1000)
|                           --> Index lookup on ActiveParty using userID (userID=temp.userID) (cost=0.30 rows=1) (actual time=0.012..0.013 rows=0 loops=33)
|
|-----+
1 row in set (0.01 sec)

mysql>
```

For Query 1, we decided to go with rating_idx_3. Although the performance difference was negligible, with more rows the minute difference in time might scale to something significant. We think that the difference was so small because there were so few rows and our query wasn't very complicated so SQL optimizes it anyways.

For Query 2:

First index

```
CREATE INDEX user_idx_1
ON User (userID);
```

```
front_end — mysql -u root -p — 182x33

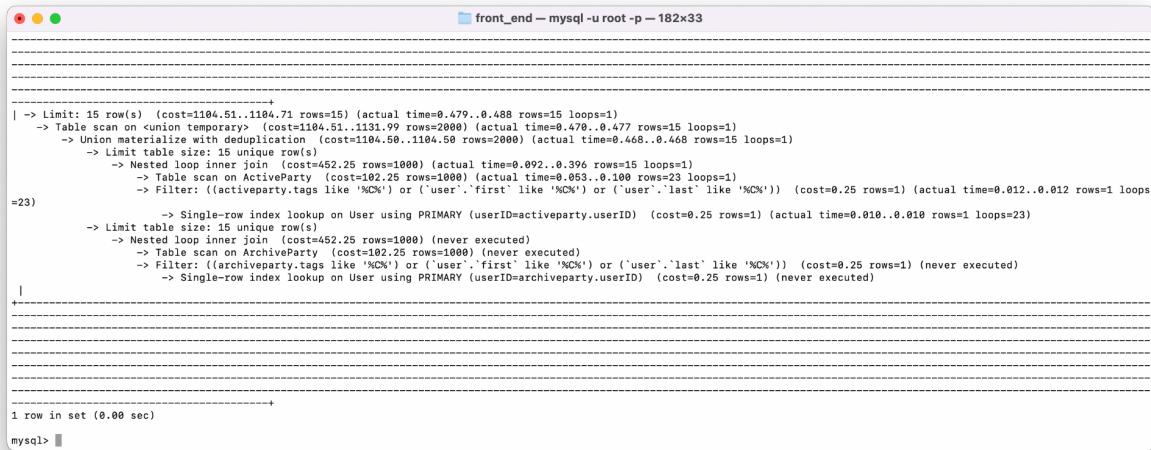
+-----+
| --> Limit: 15 row(s) (cost=1184.51..1184.71 rows=15) (actual time=9.459..0.468 rows=15 loops=1)
|   --> Table scan on <temporary> (cost=1184.51..1184.99 rows=2000) (actual time=0.456..0.463 rows=15 loops=1)
|     --> Union materialized with deduplication (cost=1184.50..1184.50 rows=2000) (actual time=0.454..0.454 rows=15 loops=1)
|       --> Limit table size: 15 unique row(s)
|         --> Nested loop inner join (cost=452.25 rows=1000) (actual time=0.138..0.363 rows=15 loops=1)
|           --> Table scan on ActiveParty (cost=102.25 rows=1000) (actual time=0.085..0.134 rows=23 loops=1)
|             --> Filter: ((activeparty.tags like '%CK%' or ('user'.`first` like '%CK%') or ('user'.`last` like '%CK%')) (cost=0.25 rows=1) (actual time=0.009..0.009 rows=1 loops=23)
|               --> Single-row index lookup on User using PRIMARY (userID=activeparty.userID) (cost=0.25 rows=1) (actual time=0.006..0.006 rows=1 loops=23)
|                 --> Limit table size: 15 unique row(s)
|                   --> Nested loop inner join (cost=452.25 rows=1000) (never executed)
|                     --> Table scan on ArchiveParty (cost=102.25 rows=1000) (never executed)
|                       --> Filter: ((archiveparty.tags like '%CK%' or ('user'.`first` like '%CK%') or ('user'.`last` like '%CK%')) (cost=0.25 rows=1) (never executed)
|                         --> Single-row index lookup on User using PRIMARY (userID=archiveparty.userID) (cost=0.25 rows=1) (never executed)
|
|-----+
1 row in set (0.01 sec)

mysql>
```

Second index

```
CREATE INDEX user_idx_2
ON User (first DESC, last DESC);
```

Reasoning for change: since we search against first and last names, it makes sense to create an index for these.



```
+--> Limit: 15 row(s) (cost=1184.51..1184.71 rows=15) (actual time=0.479..0.488 rows=15 loops=1)
      --> Table scan on <union temporary> (cost=1184.51..1131.99 rows=2000) (actual time=0.478..0.477 rows=15 loops=1)
          -- Union materialize with deduplication (cost=1184.50..1184.50 rows=2000) (actual time=0.468..0.468 rows=15 loops=1)
              --> Limit table size: 15 unique row(s)
                  --> Nested loop inner join (cost=452.25 rows=1000) (actual time=0.092..0.396 rows=15 loops=1)
                      --> Table scan on ActiveParty (cost=102.25 rows=1000) (actual time=0.053..0.100 rows=23 loops=1)
                          --> Filter: ((activeparty.tags like '%CN%') or ('user'.`first` like '%CN%') or ('user'.`last` like '%CN%')) (cost=0.25 rows=1) (actual time=0.012..0.012 rows=1 loops=23)
                              --> Single-row index lookup on User using PRIMARY (userID=activeparty.userID) (cost=0.25 rows=1) (actual time=0.010..0.010 rows=1 loops=23)
                      --> Nested loop inner join (cost=452.25 rows=1000) (never executed)
                          --> Table scan on ArchiveParty (cost=102.25 rows=1000) (never executed)
                              --> Filter: ((archiveparty.tags like '%CN%') or ('user'.`first` like '%CN%') or ('user'.`last` like '%CN%')) (cost=0.25 rows=1) (never executed)
                                  --> Single-row index lookup on User using PRIMARY (userID=archiveparty.userID) (cost=0.25 rows=1) (never executed)
|
```

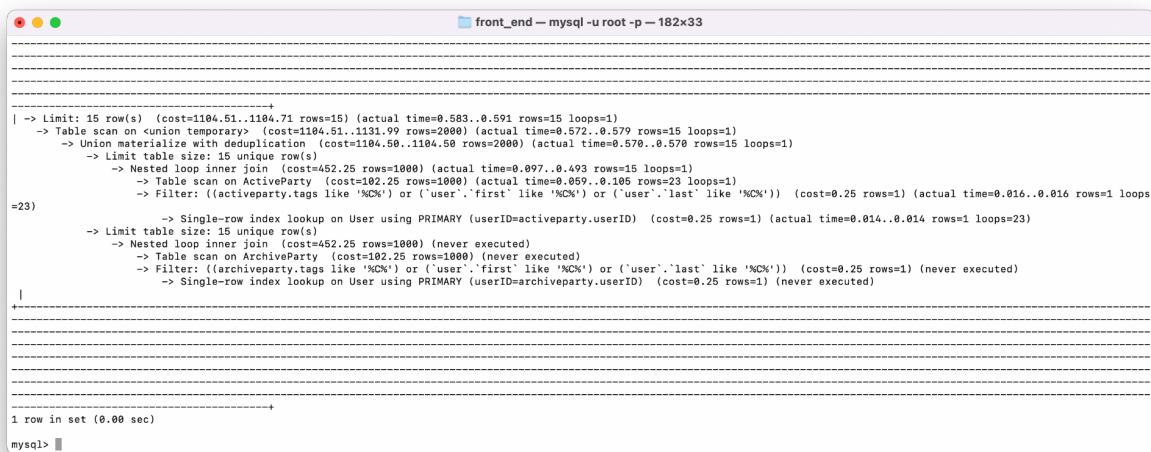
1 row in set (0.00 sec)

mysql>

Third index

```
CREATE INDEX user_idx_3
ON User (first ASC, last ASC);
```

Reasoning for change: people are more likely to have names near the beginning of the alphabet, so it makes sense to create the order alphabetically.



```
+--> Limit: 15 row(s) (cost=1184.51..1184.71 rows=15) (actual time=0.583..0.591 rows=15 loops=1)
      --> Table scan on <union temporary> (cost=1184.51..1131.99 rows=2000) (actual time=0.572..0.579 rows=15 loops=1)
          -- Union materialize with deduplication (cost=1184.50..1184.50 rows=2000) (actual time=0.570..0.570 rows=15 loops=1)
              --> Limit table size: 15 unique row(s)
                  --> Nested loop inner join (cost=452.25 rows=1000) (actual time=0.097..0.493 rows=15 loops=1)
                      --> Table scan on ActiveParty (cost=102.25 rows=1000) (actual time=0.059..0.105 rows=23 loops=1)
                          --> Filter: ((activeparty.tags like '%CN%') or ('user'.`first` like '%CN%') or ('user'.`last` like '%CN%')) (cost=0.25 rows=1) (actual time=0.016..0.016 rows=1 loops=23)
                              --> Single-row index lookup on User using PRIMARY (userID=activeparty.userID) (cost=0.25 rows=1) (actual time=0.014..0.014 rows=1 loops=23)
                      --> Nested loop inner join (cost=452.25 rows=1000) (never executed)
                          --> Table scan on ArchiveParty (cost=102.25 rows=1000) (never executed)
                              --> Filter: ((archiveparty.tags like '%CN%') or ('user'.`first` like '%CN%') or ('user'.`last` like '%CN%')) (cost=0.25 rows=1) (never executed)
                                  --> Single-row index lookup on User using PRIMARY (userID=archiveparty.userID) (cost=0.25 rows=1) (never executed)
|
```

1 row in set (0.00 sec)

mysql>

For Query 2, we decided to go with user_idx_3. The performance of each index was quite similar, but we think that adding more rows would increase the difference significantly. The difference was probably so small because there were so few rows and our query wasn't very complicated so SQL optimizes it anyways.