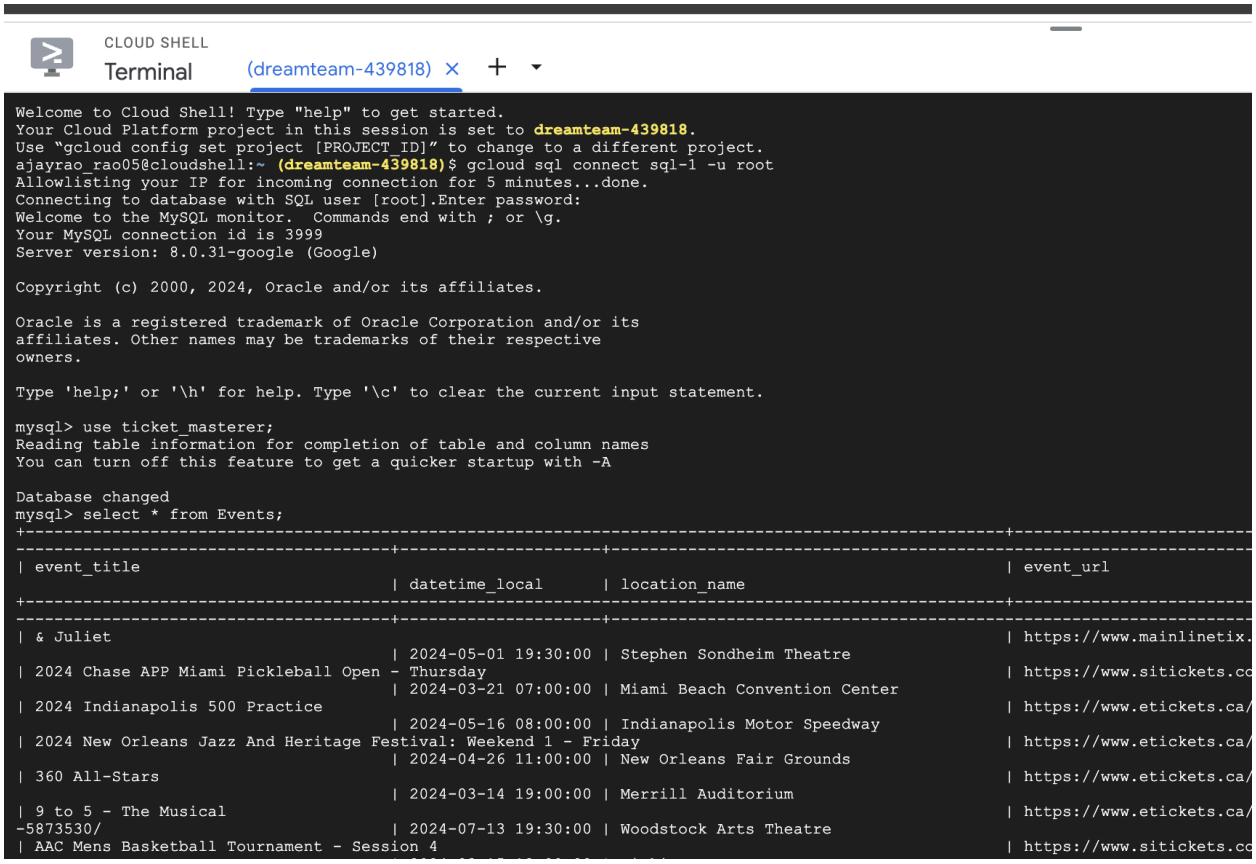
Part 1

GCP Connection:



DDL Commands:

CREATE TABLE Users (username VARCHAR(100) PRIMARY KEY, name VARCHAR(100), password VARCHAR(100), is_admin BOOLEAN);

CREATE TABLE Locations (location_name VARCHAR(250) PRIMARY KEY, address VARCHAR(250), city VARCHAR(100), state VARCHAR(100), country VARCHAR(100), postal_code VARCHAR(10));

CREATE TABLE Promoter (promoter_name VARCHAR(100) PRIMARY KEY, promoter_url VARCHAR(250));

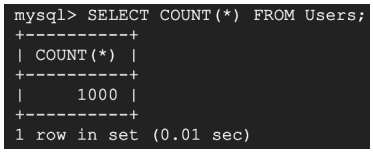
CREATE TABLE Events (event_title VARCHAR(250) PRIMARY KEY, event_url VARCHAR(250), datetime_local DATETIME, location_name VARCHAR(250),

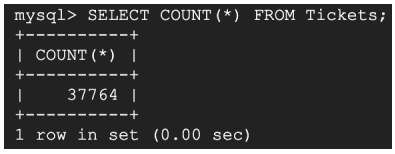
promoter_name VARCHAR(100), username VARCHAR(100), FOREIGN KEY (location_name) REFERENCES Locations(location_name) ON DELETE CASCADE ON UPDATE CASCADE, FOREIGN KEY (promoter_name) REFERENCES Promoter(promoter_name) ON DELETE CASCADE ON UPDATE CASCADE, FOREIGN KEY (username) REFERENCES Users(username) ON DELETE SET NULL ON UPDATE CASCADE);

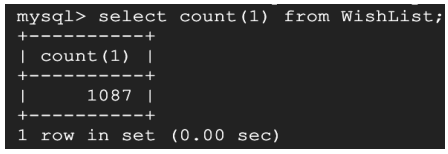
CREATE TABLE Tickets (ticket_id VARCHAR(100) PRIMARY KEY, event_title VARCHAR(250), ticket_price DECIMAL(10,2), total_price DECIMAL(10,2), fee DECIMAL(10,2), full_section VARCHAR(100), section VARCHAR(100), row_num VARCHAR(100), quantity INT, username VARCHAR(100), FOREIGN KEY (event_title) REFERENCES Events(event_title) ON DELETE CASCADE ON UPDATE CASCADE, FOREIGN KEY (username) REFERENCES Users(username) ON DELETE SET NULL ON UPDATE CASCADE));

CREATE TABLE WishList (username VARCHAR(100), event_title VARCHAR(250), wishlist_date DATETIME, PRIMARY KEY (username, event_title), FOREIGN KEY (username) REFERENCES Users(username) ON DELETE CASCADE ON UPDATE CASCADE, FOREIGN KEY (event_title) REFERENCES Events(event_title) ON DELETE CASCADE ON UPDATE CASCADE);

Users, Tickets, and Wishlist have 1,000+ rows:

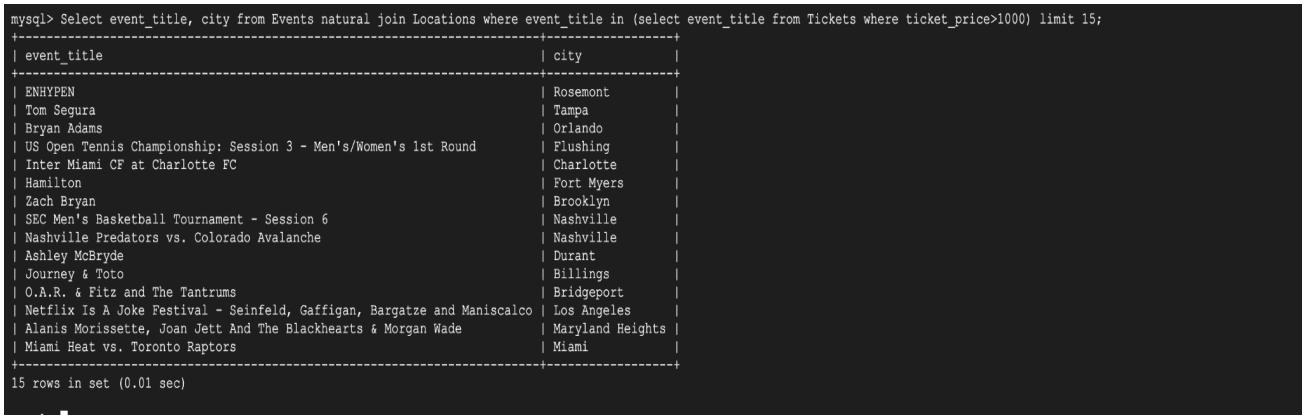






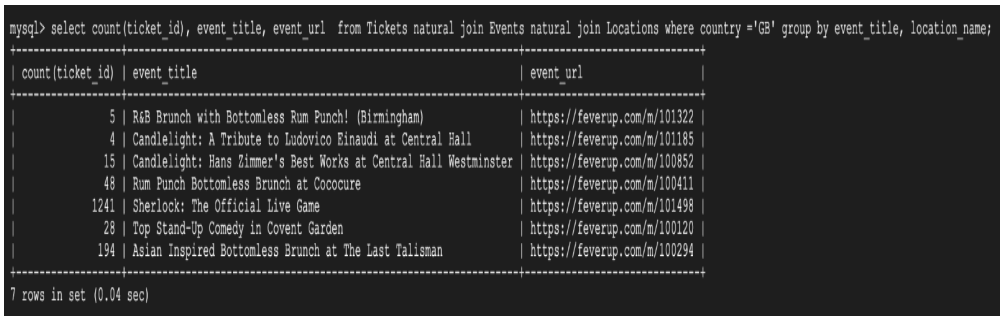
Advanced SQL Queries:

****Events and city with Ticket_price starting from $1000****
Select event_title, city from Events natural join Locations where event_title in (select event_title from Tickets where ticket_price>1000);



```
mysql> Select event_title, city from Events natural join Locations where event_title in (select event_title from Tickets where ticket_price>1000) limit 15;
+-----------------------------------------------------------------------+------------------+
| event_title                                                           | city             |
+-----------------------------------------------------------------------+------------------+
| ENHYPEN                                                               | Rosemont         |
| Tom Segura                                                            | Tampa            |
| Bryan Adams                                                           | Orlando          |
| US Open Tennis Championship: Session 3 - Men's/Women's 1st Round      | Flushing         |
| Inter Miami CF at Charlotte FC                                        | Charlotte        |
| Hamilton                                                              | Fort Myers       |
| Zach Bryan                                                            | Brooklyn         |
| SEC Men's Basketball Tournament - Session 6                           | Nashville        |
| Nashville Predators vs. Colorado Avalanche                            | Nashville        |
| Ashley McBryde                                                        | Durant           |
| Journey & Toto                                                        | Billings         |
| O.A.R. & Fitz and The Tantrums                                        | Bridgeport       |
| Netflix Is A Joke Festival - Seinfeld, Gaffigan, Bargatze and Maniscalco | Los Angeles      |
| Alanis Morissette, Joan Jett And The Blackhearts & Morgan Wade        | Maryland Heights |
| Miami Heat vs. Toronto Raptors                                        | Miami            |
+-----------------------------------------------------------------------+------------------+
15 rows in set (0.01 sec)
```

****Events and number of tickets available for all events in Great Britain****
select count(ticket_id), event_title, event_url from Tickets natural join Events natural join Locations where country ='GB' group by event_title, location_name;



```
mysql> select count(ticket_id), event_title, event_url  from Tickets natural join Events natural join Locations where country ='GB' group by event_title, location_name;
+------------------+-----------------------------------------------------------+---------------------------+
| count(ticket_id) | event_title                                               | event_url                 |
+------------------+-----------------------------------------------------------+---------------------------+
|                5 | R&B Brunch with Bottomless Rum Punch! (Birmingham)        | https://feverup.com/m/101322 |
|                4 | Candlelight: A Tribute to Ludovico Einaudi at Central Hall | https://feverup.com/m/101185 |
|               15 | Candlelight: Hans Zimmer's Best Works at Central Hall Westminster | https://feverup.com/m/100852 |
|               48 | Rum Punch Bottomless Brunch at Cococure                   | https://feverup.com/m/100411 |
|             1241 | Sherlock: The Official Live Game                          | https://feverup.com/m/101498 |
|               28 | Top Stand-Up Comedy in Covent Garden                      | https://feverup.com/m/100120 |
|              194 | Asian Inspired Bottomless Brunch at The Last Talisman     | https://feverup.com/m/100294 |
+------------------+-----------------------------------------------------------+---------------------------+
7 rows in set (0.04 sec)
```

****Events, location, city, number of users who have wishlisted events for the first quarter of 2025****
select event_title, location_name, city, count(username) from WishList natural join Events natural join Locations where wishlist_date between '2025-01-01' and '2025-03-01' group by event_title, location_name, city;

```
mysql> select event_title, location_name, city, count(username)  from  WishList natural join Events natural join Locations where wishlist_date between '2025-01-01' and '2025-03-01' gr
oup by event_title, location_name, city limit 15;
+---------------------------------------------------+-----------------------------------------------------+-------------+-----------------+
| event_title                                       | location_name                                       | city        | count(username) |
+---------------------------------------------------+-----------------------------------------------------+-------------+-----------------+
| R&B Brunch with Bottomless Rum Punch! (Birmingham) | Bierkeller Birmingham                               | Birmingham  |               1 |
| Annie                                             | Washington Pavilion of Arts & Science               | Sioux Falls |               1 |
| Shane Gillis                                      | The Chicago Theatre                                 | Chicago     |               1 |
| Willis Alan Ramsey & Keith Sykes                  | Germantown Performing Arts Centre                   | Memphis     |               3 |
| Ryan Beatty                                       | The Wiltern                                         | Los Angeles |               1 |
| Augustana                                         | Park West                                           | Chicago     |               1 |
| Chicago Dogs vs. Cleburne Railroaders             | Impact Field                                        | Rosemont    |               1 |
| Knocked Loose                                     | Marathon Music Works                                | Nashville   |               1 |
| Detroit Pistons vs. Toronto Raptors               | Little Caesars Arena                                | Detroit     |               1 |
| Toughest Monster Truck Tour                       | Dow Arena At Dow Event Center                       | Saginaw     |               2 |
| Shrek The Musical                                 | Louisville Palace                                   | Louisville  |               2 |
| Oklahoma Sooners vs. BYU Cougars                  | Lloyd Noble Center                                  | Norman      |               1 |
| Soul Of Motown                                    | Westgate Cabaret At Westgate Las Vegas Resort & Casino | Las Vegas   |               2 |
| Big Something                                     | Madison Theater - Covington                         | Covington   |               2 |
| Tyler Braden                                      | Tally Ho Theater                                    | Leesburg    |               1 |
+---------------------------------------------------+-----------------------------------------------------+-------------+-----------------+
15 rows in set (0.00 sec)
```

****Get all Tickets Details, Event details of all events for the top 5 major cities (top 5 cities by events)****

Select ticket_id, total_price, event_title, city from Tickets natural join Events natural join Locations where city in (Select city from (select city, count(event_title) from Locations natural join Events group by city order by 2 desc limit 5) z
);

```
mysql> Select ticket_id, total_price, event_title, city from Tickets natural join Events natural join Locations where city in ( Select city from (select city, count(event_title) from
Locations natural join Events group by city order by 2 desc limit 5)z ) limit 15;
+-----------+-------------+-------------+-----------+
| ticket_id | total_price | event_title | city      |
+-----------+-------------+-------------+-----------+
| 707771229 |      149.85 | Fantasy     | Las Vegas |
| 707771232 |      139.05 | Fantasy     | Las Vegas |
| 719956804 |      116.10 | Fantasy     | Las Vegas |
| 719956883 |      116.10 | Fantasy     | Las Vegas |
| 719956898 |      116.10 | Fantasy     | Las Vegas |
| 770609924 |      203.85 | Fantasy     | Las Vegas |
| 892230027 |      128.25 | Fantasy     | Las Vegas |
| 892230030 |      152.55 | Fantasy     | Las Vegas |
| 892230031 |      176.85 | Fantasy     | Las Vegas |
| 892249054 |      152.55 | Fantasy     | Las Vegas |
| 892342865 |      176.85 | Fantasy     | Las Vegas |
| 892360244 |      176.85 | Fantasy     | Las Vegas |
| 901075247 |      128.25 | Fantasy     | Las Vegas |
| 905895086 |      144.45 | Fantasy     | Las Vegas |
| 908305901 |      176.85 | Fantasy     | Las Vegas |
+-----------+-------------+-------------+-----------+
15 rows in set (0.00 sec)
```

Part 2

Events and city with Ticket_price starting from $1000

Default Configuration:

```
mysql> explain analyze Select event_title, city from Events natural join Locations where event_title in (select event_title from Tickets where ticket_price>1000);
+----------------------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                                                                |
+----------------------------------------------------------------------------------------------------------------------------------------+
| -> Nested loop inner join  (cost=518110.95 rows=5178782) (actual time=27.326..29.043 rows=48 loops=1)
    -> Nested loop inner join  (cost=190.75 rows=420) (actual time=0.101..1.545 rows=420 loops=1)
        -> Filter: (`Events`.location_name is not null)  (cost=43.75 rows=420) (actual time=0.067..0.267 rows=420 loops=1)
            -> Covering index scan on Events using idx1  (cost=43.75 rows=420) (actual time=0.065..0.232 rows=420 loops=1)
        -> Single-row index lookup on Locations using PRIMARY (location_name=`Events`.location_name)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=420)
    -> Single-row index lookup on <subquery2> using <auto_distinct_key> (event_title=`Events`.event_title)  (actual time=0.065..0.065 rows=0 loops=420)
        -> Materialize with deduplication  (cost=5004.79..5004.79 rows=12330) (actual time=27.122..27.122 rows=48 loops=1)
            -> Filter: (Tickets.event_title is not null)  (cost=3771.75 rows=12330) (actual time=0.148..26.026 rows=2376 loops=1)
                -> Filter: (Tickets.ticket_price > 1000.00)  (cost=3771.75 rows=12330) (actual time=0.148..25.856 rows=2376 loops=1)
                    -> Table scan on Tickets  (cost=3771.75 rows=36995) (actual time=0.093..21.806 rows=37764 loops=1)
 |
+----------------------------------------------------------------------------------------------------------------------------------------
```

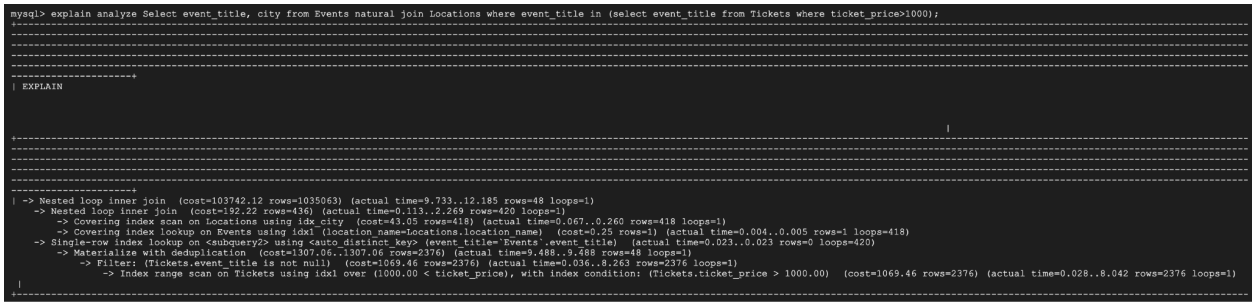
Configuration 1: create index idx1 on Tickets(ticket_price);

```
mysql> explain analyze
    -> Select event_title, city from Events natural join Locations where event_title in (select event_title from Tickets where ticket_price>1000);
+----------------------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                                                                |
+----------------------------------------------------------------------------------------------------------------------------------------+
| -> Nested loop inner join  (cost=100024.75 rows=997920) (actual time=5.171..6.340 rows=48 loops=1)
    -> Nested loop inner join  (cost=190.75 rows=420) (actual time=0.062..1.031 rows=420 loops=1)
        -> Filter: (`Events`.location_name is not null)  (cost=43.75 rows=420) (actual time=0.036..0.179 rows=420 loops=1)
            -> Covering index scan on Events using idx1  (cost=43.75 rows=420) (actual time=0.035..0.150 rows=420 loops=1)
        -> Single-row index lookup on Locations using PRIMARY (location_name=`Events`.location_name)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=420)
    -> Single-row index lookup on <subquery2> using <auto_distinct_key> (event_title=`Events`.event_title)  (actual time=0.013..0.013 rows=0 loops=420)
        -> Materialize with deduplication  (cost=1307.06..1307.06 rows=2376) (actual time=5.052..5.052 rows=48 loops=1)
            -> Filter: (Tickets.event_title is not null)  (cost=1069.46 rows=2376) (actual time=0.018..4.348 rows=2376 loops=1)
                -> Index range scan on Tickets using idx1 over (1000.00 < ticket_price), with index condition: (Tickets.ticket_price > 1000.00)  (cost=1069.46 rows=2376) (actual time=
0.017..4.192 rows=2376 loops=1)
 |
```

Configuration 2: create index idx1 on Locations(city);

```
mysql> explain analyze
    -> Select event_title, city from Events natural join Locations where event_title in (select event_title from Tickets where ticket_price>1000);
+----------------------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN
+----------------------------------------------------------------------------------------------------------------------------------------+
| -> Nested loop inner join  (cost=537389.85 rows=5371541) (actual time=123.621..125.834 rows=48 loops=1)
    -> Nested loop inner join  (cost=192.22 rows=436) (actual time=0.078..2.048 rows=420 loops=1)
        -> Covering index scan on Locations using idx_city  (cost=43.05 rows=418) (actual time=0.053..0.202 rows=418 loops=1)
        -> Covering index lookup on Events using idx1 (location_name=Locations.location_name)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=418)
    -> Single-row index lookup on <subquery2> using <auto_distinct_key> (event_title=`Events`.event_title)  (actual time=0.295..0.295 rows=0 loops=420)
        -> Materialize with deduplication  (cost=5082.38..5082.38 rows=12330) (actual time=123.396..123.396 rows=48 loops=1)
            -> Filter: (Tickets.event_title is not null)  (cost=3849.34 rows=12330) (actual time=1.818..121.811 rows=2376 loops=1)
                -> Filter: (Tickets.ticket_price > 1000.00)  (cost=3849.34 rows=12330) (actual time=1.815..121.522 rows=2376 loops=1)
                    -> Table scan on Tickets  (cost=3849.34 rows=36995) (actual time=1.674..116.593 rows=37764 loops=1)
 |
```

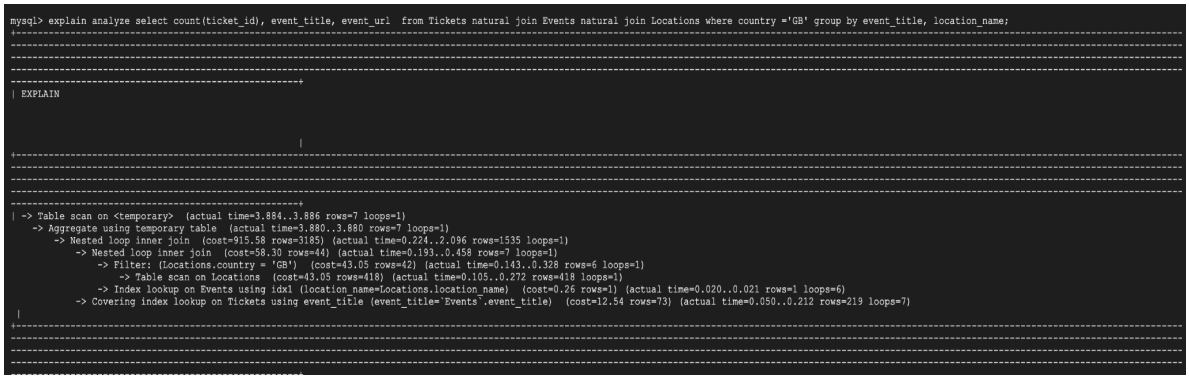
Configuration 3: create index idx1 on Tickets(ticket_price); create index idx1 on Locations(city);



We are using ticket_price with a ‘where’ clause. When an index is created on ticket_price, the database can quickly locate the rows that satisfy this condition. Without an index, the database would need to perform a full table scan, checking every row, which is much slower, especially as the number of rows increases. Thus, the cost for table scan operation (inside subquery) reduced from 3771 to 1069 for a total decrease from 518,110 to 100,024. However, creating an index on Locations(city) increased the cost from 518,110 to 537,389 seemingly because it is strongly associated with event_title, a primary key. As one might assume, combining these keys improves overall cost from the default (from 518,110 to 103,742) but results in a worse cost than with only the Tickets(ticket_price) indexing (from 100,024 to 103,742) because the Locations(city) indexing increases the cost. For this reason, we chose to stick with only the Tickets(ticket_price) indexing configuration (Configuration 1).

****Events and number of tickets available for all events in Great Britain****

Default Configuration:



Configuration 1: create index idx_country on Locations(country);

```
mysql> explain analyze select count(ticket_id), event_title, event_url  from Tickets natural join Events natural join Locations where country ='GB' group by event_title, location_name;
| EXPLAIN
| -> Table scan on <temporary>  (actual time=3.429..3.431 rows=7 loops=1)
    -> Aggregate using temporary table  (actual time=3.424..3.424 rows=7 loops=1)
        -> Nested loop inner join  (cost=126.30 rows=457) (actual time=0.082..1.650 rows=1535 loops=1)
            -> Nested loop inner join  (cost=3.25 rows=6) (actual time=0.041..0.121 rows=7 loops=1)
                -> Covering index lookup on Locations using idx_country (country='GB')  (cost=1.06 rows=6) (actual time=0.014..0.021 rows=6 loops=1)
                -> Index lookup on Events using idx1 (location_name=Locations.location_name)  (cost=0.28 rows=1) (actual time=0.015..0.016 rows=1 loops=6)
            -> Covering index lookup on Tickets using event_title (event_title=`Events`.event_title)  (cost=13.54 rows=73) (actual time=0.044..0.201 rows=219 loops=7)
 |
1 row in set (0.01 sec)
```

Configuration 2: create index idx_url on Events(event_url);

```
mysql> explain analyze
    -> select count(ticket_id), event_title, event_url  from Tickets natural join Events natural join Locations where country ='GB' group by event_title, location_name;
| EXPLAIN
| -> Table scan on <temporary>  (actual time=3.429..3.430 rows=7 loops=1)
    -> Aggregate using temporary table  (actual time=3.427..3.427 rows=7 loops=1)
        -> Nested loop inner join  (cost=915.58 rows=3185) (actual time=0.147..1.739 rows=1535 loops=1)
            -> Nested loop inner join  (cost=58.30 rows=44) (actual time=0.127..0.329 rows=7 loops=1)
                -> Filter: (Locations.country = 'GB')  (cost=43.05 rows=42) (actual time=0.098..0.245 rows=6 loops=1)
                    -> Table scan on Locations  (cost=43.05 rows=418) (actual time=0.074..0.199 rows=418 loops=1)
                -> Index lookup on Events using idx1 (location_name=Locations.location_name)  (cost=0.26 rows=1) (actual time=0.013..0.014 rows=1 loops=6)
            -> Covering index lookup on Tickets using event_title (event_title=`Events`.event_title)  (cost=12.54 rows=73) (actual time=0.039..0.181 rows=219 loops=7)
 |
1 row in set (0.01 sec)
```

Configuration 3: create index idx_url on Events(event_url); create index idx_country on Locations(country);

```
mysql> explain analyze select count(ticket_id), event_title, event_url  from Tickets natural join Events natural join Locations where country ='GB' group by event_title, location_name;
| EXPLAIN
| -> Table scan on <temporary>  (actual time=3.245..3.246 rows=7 loops=1)
    -> Aggregate using temporary table  (actual time=3.242..3.242 rows=7 loops=1)
        -> Nested loop inner join  (cost=126.30 rows=457) (actual time=0.059..1.551 rows=1535 loops=1)
            -> Nested loop inner join  (cost=3.25 rows=6) (actual time=0.040..0.097 rows=7 loops=1)
                -> Covering index lookup on Locations using idx_country (country='GB')  (cost=1.06 rows=6) (actual time=0.014..0.017 rows=6 loops=1)
                -> Index lookup on Events using idx1 (location_name=Locations.location_name)  (cost=0.28 rows=1) (actual time=0.012..0.013 rows=1 loops=6)
            -> Covering index lookup on Tickets using event_title (event_title=`Events`.event_title)  (cost=13.54 rows=73) (actual time=0.042..0.189 rows=219 loops=7)
 |
1 row in set (0.01 sec)
```

One of the query includes a WHERE clause that filters results based on country = 'GB'. If there is an index on the country column, the database can quickly locate rows that match this condition.

Instead of an entire table scan operation on the Locations table, the query will perform index lookup operation, thus reducing the cost of this operation from 43.05 to 1.06 (Config 1). This config reduced the overall query cost from 915.58 to 126.30. When we tried to index the query on Events(event_url), the cost did not see any change, indicating no effect from this column (Config 2). Additionally, we tried to use both the above indexes, the overall cost reduced to 126.30(Config 3 same as Config 1). Thus, the ideal configuration was built using the country as an indexing column.

—---

****Events, location, city, number of users who have wishlisted events for the first quarter of 2025****

Default Configuration:

```
mysql> explain analyze
    -> select event_title, location_name, city, count(username)  from  WishList natural join Events natural join Locations where wishlist_date between '2025-01-01' and '2025-03-01' gr
oup by event_title, location_name, city;
+----------------------------------------------------------------------------------------------------------
------------+
| EXPLAIN
                                                                                      |
+----------------------------------------------------------------------------------------------------------
------------+
| -> Table scan on <temporary>  (actual time=3.421..3.472 rows=190 loops=1)
    -> Aggregate using temporary table  (actual time=3.418..3.418 rows=190 loops=1)
        -> Nested loop inner join  (cost=257.24 rows=159) (actual time=0.133..2.901 rows=246 loops=1)
            -> Nested loop inner join  (cost=201.52 rows=159) (actual time=0.126..2.095 rows=246 loops=1)
                -> Filter: (WishList.wishlist_date between '2025-01-01' and '2025-03-01')  (cost=145.80 rows=159) (actual time=0.096..1.226 rows=246 loops=1)
                    -> Table scan on WishList  (cost=145.80 rows=1433) (actual time=0.086..0.691 rows=1433 loops=1)
                -> Filter: (`Events`.location_name is not null)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=246)
                    -> Single-row index lookup on Events using PRIMARY (event_title=WishList.event_title)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=246)
            -> Single-row index lookup on Locations using PRIMARY (location_name=`Events`.location_name)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=246)
 |
```

Configuration 1: create index idx1 on WishList(wishlisted_date);

```
mysql> explain analyze select event_title, location_name, city, count(username)  from  WishList natural join Events natural join Locations where wishlist_date between '2025-01-01' and
 '2025-03-01' group by event_title, location_name, city;
+----------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------+
| EXPLAIN
                                                                                      |
+----------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------+
| -> Table scan on <temporary>  (actual time=2.141..2.171 rows=190 loops=1)
    -> Aggregate using temporary table  (actual time=2.139..2.139 rows=190 loops=1)
        -> Nested loop inner join  (cost=256.97 rows=246) (actual time=0.054..1.759 rows=246 loops=1)
            -> Nested loop inner join  (cost=170.87 rows=246) (actual time=0.045..1.017 rows=246 loops=1)
                -> Filter: (WishList.wishlist_date between '2025-01-01' and '2025-03-01')  (cost=84.77 rows=246) (actual time=0.021..0.238 rows=246 loops=1)
                    -> Covering index range scan on WishList using idx1 over ('2025-01-01 00:00:00' <= wishlist_date <= '2025-03-01 00:00:00')  (cost=84.77 rows=246) (actual time=0.01
8..0.140 rows=246 loops=1)
                -> Filter: (`Events`.location_name is not null)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=246)
                    -> Single-row index lookup on Events using PRIMARY (event_title=WishList.event_title)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=246)
            -> Single-row index lookup on Locations using PRIMARY (location_name=`Events`.location_name)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=246)
 |
```

Configuration 2: create index idx2 on WishList(event_title, wishlisted_date);

```
mysql> explain analyze
    -> select event_title, location_name, city, count(username) from WishList natural join Events natural join Locations where wishlist_date between '2025-01-01' and '2025-03-01' group by event_title, location_name, city;
+----------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                                                    |
+----------------------------------------------------------------------------------------------------------------------------+
| -> Group aggregate: count(WishList.username)  (cost=273.17 rows=159) (actual time=0.244..2.769 rows=190 loops=1)
    -> Nested loop inner join  (cost=257.24 rows=159) (actual time=0.213..2.597 rows=246 loops=1)
        -> Nested loop inner join  (cost=201.52 rows=159) (actual time=0.201..1.900 rows=246 loops=1)
            -> Filter: (WishList.wishlist_date between '2025-01-01' and '2025-03-01')  (cost=145.80 rows=159) (actual time=0.093..0.974 rows=246 loops=1)
                -> Covering index scan on WishList using idx2  (cost=145.80 rows=1433) (actual time=0.080..0.517 rows=1433 loops=1)
            -> Filter: (`Events`.location_name is not null)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=246)
                -> Single-row index lookup on Events using PRIMARY (event_title=WishList.event_title)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=246)
        -> Single-row index lookup on Locations using PRIMARY (location_name=`Events`.location_name)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=246)
 |
+----------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

Configuration 3: create index idx3 on WishList(username, wishlisted_date);

```
mysql> explain analyze select event_title, location_name, city, count(username) from WishList natural join Events natural join Locations where wishlist_date between '2025-01-01' and '2025-03-01' group by event_title,
 location_name, city;
+----------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                                                    |
+----------------------------------------------------------------------------------------------------------------------------+
| -> Group aggregate: count(WishList.username)  (cost=272.67 rows=159) (actual time=0.614..7.510 rows=190 loops=1)
    -> Nested loop inner join  (cost=256.74 rows=159) (actual time=0.576..7.289 rows=246 loops=1)
        -> Nested loop inner join  (cost=201.02 rows=159) (actual time=0.560..6.425 rows=246 loops=1)
            -> Filter: (WishList.wishlist_date between '2025-01-01' and '2025-03-01')  (cost=145.30 rows=159) (actual time=0.525..5.388 rows=246 loops=1)
                -> Index scan on WishList using event_title  (cost=145.30 rows=1433) (actual time=0.514..4.811 rows=1433 loops=1)
            -> Filter: (`Events`.location_name is not null)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=246)
                -> Single-row index lookup on Events using PRIMARY (event_title=WishList.event_title)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=246)
        -> Single-row index lookup on Locations using PRIMARY (location_name=`Events`.location_name)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=246)
 |
+----------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

We aim to extract the wishlisted events between a specific time frame(quarterly). We extract all the events wishlisted between start date and end date. We also observed that the performance improved given that:

1. There are at least several different values in the date column.
2. Date range used would select less than the most number of rows.

Thus, indexing on wishlisted_date(with appropriate time frame selection) reduced the cost from 145.80 to 84.77 on Wishlist table scan operation (Config 1). The overall cost was reduced from

257.94 to 256.97 (Config 1). In the next config, we used a composite index on (event_title, wishlist_date), this resulted in an extra overhead operation. The cost increased from 257.94 to 273.17 (Config 2). Changing the index to (username, wishlist_date) resulted in the same degradation of performance, from 257.94 to 272.67. Thus, Config 1 was preferred.

—---

****Get all Tickets Details, Event details of all events for the top 5 major cities (top 5 cities by events)****

Default Configuration:

```
mysql> explain analyze  Select ticket_id, total_price, event_title, city from Tickets natural join Events natural join Locations where city in ( Select city from (select city, count(event_title) from Locations natura
l join Events group by city order by 2 desc limit 5)z );
+------------------------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                                    |
+------------------------------------------------------------------------------------------------------------+
| -> Nested loop inner join  (cost=813.10 rows=0) (actual time=2.652..17.600 rows=6477 loops=1)
    -> Nested loop inner join  (cost=16.84 rows=0) (actual time=2.418..3.072 rows=78 loops=1)
        -> Filter: (Locations.city = `<subquery2>`.city)  (cost=6.28 rows=0) (actual time=2.403..2.676 rows=73 loops=1)
            -> Inner hash join (<hash>(Locations.city)=<hash>(`<subquery2>`.city))  (cost=6.28 rows=0) (actual time=2.400..2.652 rows=73 loops=1)
                -> Table scan on Locations  (cost=5.46 rows=418) (actual time=0.045..0.206 rows=418 loops=1)
                -> Hash
                    -> Table scan on <subquery2>  (cost=5.00..5.00 rows=0) (actual time=2.311..2.312 rows=5 loops=1)
                        -> Materialize with deduplication  (cost=2.50..2.50 rows=0) (actual time=2.310..2.310 rows=5 loops=1)
                            -> Filter: (z.city is not null)  (cost=2.50..2.50 rows=0) (actual time=2.302..2.303 rows=5 loops=1)
                                -> Table scan on z  (cost=2.50..2.50 rows=0) (actual time=2.301..2.302 rows=5 loops=1)
                                    -> Materialize  (cost=0.00..0.00 rows=0) (actual time=2.301..2.301 rows=5 loops=1)
                                        -> Limit: 5 row(s)  (actual time=2.286..2.287 rows=5 loops=1)
                                            -> Sort: count(event_title) DESC, limit input to 5 row(s) per chunk  (actual time=2.285..2.286 rows=5 loops=1)
                                                -> Table scan on <temporary>  (actual time=2.219..2.248 rows=222 loops=1)
                                                    -> Aggregate using temporary table  (actual time=2.216..2.216 rows=222 loops=1)
                                                        -> Nested loop inner join  (cost=190.75 rows=420) (actual time=0.069..1.770 rows=420 loops=1)
                                                            -> Filter: (`Events`.location_name is not null)  (cost=43.75 rows=420) (actual time=0.049..0.267 rows=420 loops=1)
                                                                -> Covering index scan on Events using idx1  (cost=43.75 rows=420) (actual time=0.047..0.225 rows=420 loops=1)
                                                            -> Single-row index lookup on Locations using PRIMARY (location_name=`Events`.location_name)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=420)
        -> Covering index lookup on Events using idx1 (location_name=Locations.location_name)  (cost=0.26 rows=1) (actual time=0.004..0.005 rows=1 loops=73)
    -> Index lookup on Tickets using event_title (event_title=`Events`.event_title)  (cost=18.45 rows=73) (actual time=0.067..0.179 rows=83 loops=78)
 |
+------------------------------------------------------------------------------------------------------------+
```

Configuration 1: create index idx1 on Locations(city);

```
mysql> explain analyze
    -> Select ticket_id, total_price, event_title, city from Tickets natural join Events natural join Locations where city in ( Select city from (select city, count(event_title) from
Locations natural join Events group by city order by 2 desc limit 5)z );
+------------------------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                                    |
+------------------------------------------------------------------------------------------------------------+
| -> Nested loop inner join  (cost=113.09 rows=714) (actual time=1.979..16.824 rows=6477 loops=1)
    -> Nested loop inner join  (cost=5.97 rows=10) (actual time=1.872..2.395 rows=78 loops=1)
        -> Nested loop inner join  (cost=4.52 rows=9) (actual time=1.861..1.970 rows=73 loops=1)
            -> Filter: (`<subquery2>`.city is not null)  (cost=2.00..0.01 rows=5) (actual time=1.821..1.837 rows=5 loops=1)
                -> Table scan on <subquery2>  (cost=5.00..5.00 rows=0) (actual time=1.821..1.832 rows=5 loops=1)
                    -> Materialize with deduplication  (cost=2.50..2.50 rows=0) (actual time=1.820..1.820 rows=5 loops=1)
                        -> Filter: (z.city is not null)  (cost=2.50..2.50 rows=0) (actual time=1.814..1.815 rows=5 loops=1)
                            -> Table scan on z  (cost=2.50..2.50 rows=0) (actual time=1.813..1.814 rows=5 loops=1)
                                -> Materialize  (cost=0.00..0.00 rows=0) (actual time=1.812..1.812 rows=5 loops=1)
                                    -> Limit: 5 row(s)  (actual time=1.804..1.804 rows=5 loops=1)
                                        -> Sort: count(event_title) DESC, limit input to 5 row(s) per chunk  (actual time=1.803..1.803 rows=5 loops=1)
                                            -> Table scan on <temporary>  (actual time=1.738..1.767 rows=222 loops=1)
                                                -> Aggregate using temporary table  (actual time=1.735..1.735 rows=222 loops=1)
                                                    -> Nested loop inner join  (cost=190.75 rows=420) (actual time=0.067..1.397 rows=420 loops=1)
                                                        -> Filter: (`Events`.location_name is not null)  (cost=43.75 rows=420) (actual time=0.047..0.238 rows=420 loops=1)
                                                            -> Covering index scan on Events using idx1  (cost=43.75 rows=420) (actual time=0.046..0.203 rows=420 loops=1)
                                                        -> Single-row index lookup on Locations using PRIMARY (location_name=`Events`.location_name)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=420)
            -> Covering index lookup on Locations using idx2 (city=`<subquery2>`.city)  (cost=3.77 rows=2) (actual time=0.012..0.025 rows=15 loops=5)
        -> Covering index lookup on Events using idx1 (location_name=Locations.location_name)  (cost=0.31 rows=1) (actual time=0.005..0.006 rows=1 loops=73)
    -> Index lookup on Tickets using event_title (event_title=`Events`.event_title)  (cost=22.02 rows=73) (actual time=0.064..0.180 rows=83 loops=78)
 |
+------------------------------------------------------------------------------------------------------------+
```

Configuration 2: create index idx_price on Tickets(total_price);

```
mysql> explain analyze
    -> Select ticket_id, total_price, event_title, city from Tickets natural join Events natural join Locations where city in ( Select city from (select city, count(event_title) from Locations natural join Events gro
up by city order by 2 desc limit 5) z );
+------------------------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                                    |
+------------------------------------------------------------------------------------------------------------+
| -> Nested loop inner join  (cost=813.10 rows=0) (actual time=2.183..20.386 rows=6477 loops=1)
    -> Nested loop inner join  (cost=16.84 rows=0) (actual time=1.894..2.737 rows=78 loops=1)
        -> Filter: (Locations.city = `<subquery2>`.city)  (cost=6.28 rows=0) (actual time=1.877..2.258 rows=73 loops=1)
            -> Inner hash join (<hash>(Locations.city)=<hash>(`<subquery2>`.city))  (cost=6.28 rows=0) (actual time=1.874..2.223 rows=73 loops=1)
                -> Table scan on Locations  (cost=5.46 rows=418) (actual time=0.046..0.269 rows=418 loops=1)
                -> Hash
                    -> Table scan on <subquery2>  (cost=5.00..5.00 rows=0) (actual time=1.793..1.793 rows=5 loops=1)
                        -> Materialize with deduplication  (cost=2.50..2.50 rows=0) (actual time=1.792..1.792 rows=5 loops=1)
                            -> Filter: (z.city is not null)  (cost=2.50..2.50 rows=0) (actual time=1.781..1.782 rows=5 loops=1)
                                -> Table scan on z  (cost=2.50..2.50 rows=0) (actual time=1.779..1.780 rows=5 loops=1)
                                    -> Materialize  (cost=0.00..0.00 rows=0) (actual time=1.779..1.779 rows=5 loops=1)
                                        -> Limit: 5 row(s)  (actual time=1.767..1.768 rows=5 loops=1)
                                            -> Sort: count(event_title) DESC, limit input to 5 row(s) per chunk  (actual time=1.767..1.767 rows=5 loops=1)
                                                -> Table scan on <temporary>  (actual time=1.696..1.724 rows=222 loops=1)
                                                    -> Aggregate using temporary table  (actual time=1.693..1.693 rows=222 loops=1)
                                                        -> Nested loop inner join  (cost=190.75 rows=420) (actual time=0.117..1.384 rows=420 loops=1)
                                                            -> Filter: (`Events`.location_name is not null)  (cost=43.75 rows=420) (actual time=0.088..0.259 rows=420 loops=1)
                                                                -> Covering index scan on Events using idx1  (cost=43.75 rows=420) (actual time=0.086..0.224 rows=420 loops=1)
                                                            -> Single-row index lookup on Locations using PRIMARY (location_name=`Events`.location_name)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=420)
        -> Covering index lookup on Events using idx1 (location_name=Locations.location_name)  (cost=0.26 rows=1) (actual time=0.005..0.006 rows=1 loops=73)
    -> Index lookup on Tickets using event_title (event_title=`Events`.event_title)  (cost=18.45 rows=73) (actual time=0.086..0.221 rows=83 loops=78)
 |
+------------------------------------------------------------------------------------------------------------+
```

Configuration 3: create index idx1 on Locations(city); create index idx_price on Tickets(total_price);

```
mysql> explain analyze Select ticket_id, total_price, event_title, city from Tickets natural join Events natural join Locations where city in ( Select city from (select city, count(event_title) from Locations natural
 join Events group by city order by 2 desc limit 5) z );
+------------------------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                                    |
+------------------------------------------------------------------------------------------------------------+
| -> Nested loop inner join  (cost=113.09 rows=714) (actual time=1.854..18.445 rows=6477 loops=1)
    -> Nested loop inner join  (cost=5.97 rows=10) (actual time=1.712..2.227 rows=78 loops=1)
        -> Nested loop inner join  (cost=4.52 rows=9) (actual time=1.694..1.806 rows=73 loops=1)
            -> Filter: (`<subquery2>`.city is not null)  (cost=2.00..0.01 rows=5) (actual time=1.679..1.699 rows=5 loops=1)
                -> Table scan on <subquery2>  (cost=5.00..5.00 rows=0) (actual time=1.678..1.692 rows=5 loops=1)
                    -> Materialize with deduplication  (cost=2.50..2.50 rows=0) (actual time=1.678..1.678 rows=5 loops=1)
                        -> Filter: (z.city is not null)  (cost=2.50..2.50 rows=0) (actual time=1.671..1.672 rows=5 loops=1)
                            -> Table scan on z  (cost=2.50..2.50 rows=0) (actual time=1.671..1.672 rows=5 loops=1)
                                -> Materialize  (cost=0.00..0.00 rows=0) (actual time=1.670..1.670 rows=5 loops=1)
                                    -> Limit: 5 row(s)  (actual time=1.662..1.663 rows=5 loops=1)
                                        -> Sort: count(event_title) DESC, limit input to 5 row(s) per chunk  (actual time=1.661..1.661 rows=5 loops=1)
                                            -> Table scan on <temporary>  (actual time=1.592..1.619 rows=222 loops=1)
                                                -> Aggregate using temporary table  (actual time=1.590..1.590 rows=222 loops=1)
                                                    -> Nested loop inner join  (cost=190.75 rows=420) (actual time=0.065..1.289 rows=420 loops=1)
                                                        -> Filter: (`Events`.location_name is not null)  (cost=43.75 rows=420) (actual time=0.045..0.207 rows=420 loops=1)
                                                            -> Covering index scan on Events using idx1  (cost=43.75 rows=420) (actual time=0.044..0.171 rows=420 loops=1)
                                                        -> Single-row index lookup on Locations using PRIMARY (location_name=`Events`.location_name)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=420)
            -> Covering index lookup on Locations using idx1 (city=`<subquery2>`.city)  (cost=3.77 rows=2) (actual time=0.008..0.020 rows=15 loops=5)
        -> Covering index lookup on Events using idx1 (location_name=Locations.location_name)  (cost=0.31 rows=1) (actual time=0.005..0.006 rows=1 loops=73)
    -> Index lookup on Tickets using event_title (event_title=`Events`.event_title)  (cost=22.02 rows=73) (actual time=0.080..0.202 rows=83 loops=78)
 |
+------------------------------------------------------------------------------------------------------------+
```

We aim to extract the top 5 cities by events and display tickets associated with these cities. Thus, we add an index to the city column, which is used with a ‘where’ clause. We observe that the number of operations for the query is reduced after indexing. Before indexing, the query scanned the entire table Location and performed an inner hash join with the subquery. After indexing, index lookup was performed initially which resulted in reduction of cost of overall query from 813.10 to 113.09 (Config 1). With only Tickets(total_price) as index, the overall cost remained constant (Config 2). When both the previous indexes are considered, the cost is reduced to 113.09 (Config 3 same as Config 1). Thus we went ahead with Config 1, since indexing on total_price did not affect the query performance.

Final Index Design:

- create index idx1 on Locations(city);
- create index idx1 on WishList(wishlisted_date);
- create index idx_country on Locations(country);
- create index idx1 on Tickets(ticket_price);