

# Stage 3: Database Implementation and Indexing

## Data Definition Language (DDL) commands

```
CREATE Table User (
    userId INT,
    userName VARCHAR(200),
    account VARCHAR(50),
    password VARCHAR(50),
    email VARCHAR(50),
    phoneNumber INT,
    PRIMARY KEY(userId)
);
```

```
CREATE Table Product (
    productId INT,
    productName VARCHAR(50),
    storeName,
    price REAL,
    link VARCHAR(20),
    PRIMARY KEY(productId)
);
```

```
CREATE Table Post (
    postId INT,
    userId INT,
    expirationDate Date,
    groupLimit INT,
    paymentMethod VARCHAR(20),
    categoryId INT,
    PRIMARY KEY(postId),
    FOREIGN KEY(userId) REFERENCES User (userId) ON DELETE SET NULL,
    FOREIGN KEY(categoryId) REFERENCES Category (categoryId) ON UPDATE
    CASCADE
);
```

```
CREATE Table Category (
    categoryId INT,
    categoryName VARCHAR(200),
    PRIMARY KEY(categoryId)
);
```

```
CREATE Table Payment (
    paymentId INT,
    userId INT,
    postId INT,
    paymentMethod VARCHAR(20),
    PRIMARY KEY(paymentId),
    FOREIGN KEY(userId) REFERENCES User (userId) ON DELETE CASCADE,
    FOREIGN KEY(postId) REFERENCES Post (postId) ON DELETE CASCADE
);
```

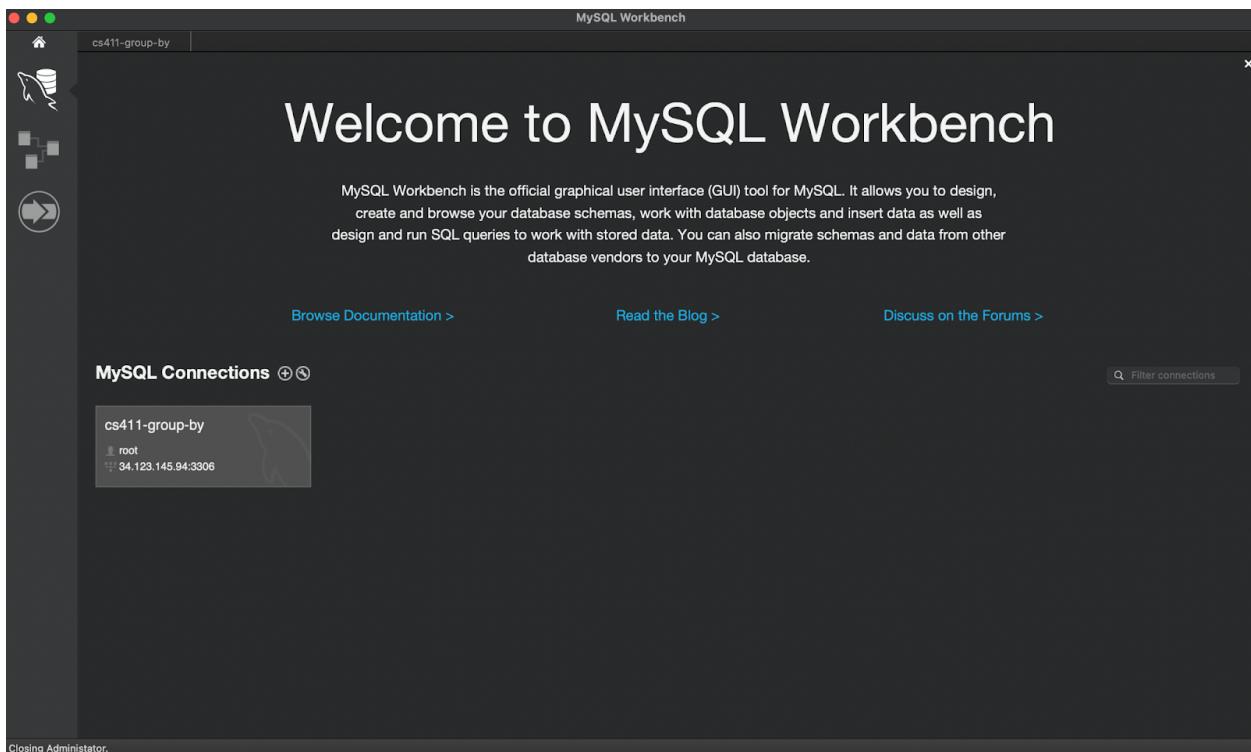
```
CREATE Table UserPost (
    userId INT,
    postId INT,
    PRIMARY KEY(userId, postId),
    FOREIGN KEY(userId) REFERENCES User (userId) ON DELETE CASCADE,
    FOREIGN KEY(postId) REFERENCES Post (postId) ON DELETE CASCADE
);
```

```
CREATE Table PostProduct (
    postId INT,
    productId INT
    PRIMARY KEY(postId, productId)
    FOREIGN KEY(postId) REFERENCES Post (postId) ON DELETE CASCADE,
    FOREIGN KEY(productId) REFERENCES Product (productId) ON DELETE CASCADE
);
```

```
CREATE Table UserProduct (
    userId INT,
    productId INT
    PRIMARY KEY(userId, productId),
    FOREIGN KEY(userId) REFERENCES User (userId) ON DELETE CASCADE,
    FOREIGN KEY(productId) REFERENCES Product (productId) ON DELETE CASCADE
);
```

## Database implementation

1. The screenshot below shows that we have successfully connected to the GCP via MySQL Workbench.



2. The screenshots below show that we have implemented the database tables.

Main Tables:

- User
- Post
- Product
- Category
- Payment

MySQL Workbench

Administration Schemas Query 1 Administration - Data Export

**SCHEMAS**

Filter objects

**db1**

- Tables
  - Category
  - Payment
  - Post
  - PostProduct
  - Product
  - User
  - UserPost
  - UserProduct
- Views
- Stored Procedures
- Functions

Object Info Session

Action Output

Schema: db1

Time	Action	Response	Duration / Fetch Time
20:56:01	CREATE Table Product ( productid INT, productName VARCHAR(50), storeName VARCHAR(50), price DECIMAL(10,2), PRIMARY KEY(productid) )	Error Code: 1050. Table 'Product' already exists 0 row(s) affected	0.020 sec
20:56:26	CREATE Table Category ( categoryId INT, categoryName VARCHAR(200), PRIMARY KEY(categoryId) )	0 row(s) affected	0.070 sec
20:56:26	CREATE Table Post ( postId INT, userId INT, expirationDate Date, groupLimit INT, paymentMethod VARCHAR(20), categoryId INT, PRIMARY KEY(postId), FOREIGN KEY(userId) REFERENCES User (userId) ON DELETE SET NULL, FOREIGN KEY(categoryId) REFERENCES Category (categoryId) ON UPDATE CASCADE )	0 row(s) affected	0.070 sec
20:56:26	CREATE Table Payment ( paymentid INT, postId INT, userId INT, PRIMARY KEY(paymentid, postId) )	0 row(s) affected	0.077 sec
20:56:26	CREATE Table UserPost ( userId INT, postId INT, PRIMARY KEY(userId, postId), FOREIGN KEY(userId) REFERENCES User (userId), FOREIGN KEY(postId) REFERENCES Post (postId) )	0 row(s) affected	0.064 sec
20:56:26	CREATE Table PostProduct ( postId INT, productid INT, PRIMARY KEY(postId, productid), FOREIGN KEY(postId) REFERENCES Post (postId), FOREIGN KEY(productid) REFERENCES Product (productid) )	0 row(s) affected	0.068 sec
20:56:26	CREATE Table UserProduct ( userId INT, productid INT, PRIMARY KEY(userId, productid), FOREIGN KEY(userId) REFERENCES User (userId), FOREIGN KEY(productid) REFERENCES Product (productid) )	0 row(s) affected	0.061 sec

Closing Administrator.

3. The screenshots below show that we have 3 tables (User, Post, Payment) that contain at least 1000 rows in each table.

<User table>

MySQL Workbench

Management Schemas Query 1

**SCHEMAS**

Filter objects

**db1**

- Category
- Payment
- Post
- PostProduct
- Product
- User
- UserPost
- Columns
- Indexes
- Foreign Keys
- Triggers
- UserProduct
- Views**
- Stored Procedures
- Functions

Object Info Session

Action Output

Schema: db1

```
select count(userId)
from User;
```

Result Grid

count(userId)
1000

Result 13

Time	Action	Response	Duration / Fetch Time
21:30:59	select count(paymentid) from Payment LIMIT 0, 10000	1 row(s) returned	0.022 sec / 0.000015...
21:37:18	select count(userId) from User LIMIT 0, 5000	1 row(s) returned	0.024 sec / 0.000018...
21:37:36	select count(postId) from Post LIMIT 0, 5000	1 row(s) returned	0.021 sec / 0.000011...
21:41:33	select * from UserPost LIMIT 0, 5000	150 row(s) returned	0.018 sec / 0.000052...
22:45:52	select count(userId) from User LIMIT 0, 5000	1 row(s) returned	0.271 sec / 0.000035...

## <Post table>

The screenshot shows the MySQL Workbench interface with the schema 'CS411\_final project'. In the 'Management' tab, under 'SCHEMAS', the 'Post' table is selected. In the 'Query 1' editor, the following SQL query is run:

```
1 • select count(postId)
  2   from Post;|
```

The results are displayed in the 'Result Grid' tab, showing a single row with the value 1500.

count(postId)
1500

Below the results, the 'Action Output' log shows the following activity:

Action	Time	Response	Duration / Fetch Time
select count(user_id) from User LIMIT 0, 5000	21:37:36	1 row(s) returned	0.024 sec / 0.000018...
select count(postId) from Post LIMIT 0, 5000	21:37:36	1 row(s) returned	0.021 sec / 0.00001...
select * from UserPost LIMIT 0, 5000	21:41:33	150 row(s) returned	0.018 sec / 0.000052...
select count(userId) from User LIMIT 0, 5000	22:45:52	1 row(s) returned	0.271 sec / 0.000035...
select count(postId) from Post LIMIT 0, 5000	22:47:05	1 row(s) returned	0.027 sec / 0.000018...

## <Payment table>

The screenshot shows the MySQL Workbench interface with the schema 'CS411\_final project'. In the 'Management' tab, under 'SCHEMAS', the 'Payment' table is selected. In the 'Query 1' editor, the following SQL query is run:

```
1 • select count(paymentId)
  2   from Payment;|
```

The results are displayed in the 'Result Grid' tab, showing a single row with the value 1027.

count(paymentId)
1027

Below the results, the 'Action Output' log shows the following activity:

Action	Time	Response	Duration / Fetch Time
select count(postId) from Post LIMIT 0, 5000	21:37:36	1 row(s) returned	0.021 sec / 0.000018...
select count(paymentId) from Payment LIMIT 0, 5000	21:41:33	1027 row(s) returned	0.018 sec / 0.000052...
select * from UserPost LIMIT 0, 5000	21:41:33	150 row(s) returned	0.271 sec / 0.000035...
select count(userId) from User LIMIT 0, 5000	22:45:52	1 row(s) returned	0.027 sec / 0.000018...
select count(paymentId) from Payment LIMIT 0, 5000	22:47:05	1 row(s) returned	0.023 sec / 0.000015...

# Advanced Queries

<Query 1> This query returns the userId and the number of posts they have posted.

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** db1
- Tables:** UserPost
- Query 1:**

```
1 • SELECT userId, COUNT(DISTINCT postId)
2   FROM User JOIN UserPost USING (userId)
3   GROUP BY userId LIMIT 15;
```
- Result Grid:** Shows the results of the query:

userId	COUNT(DISTINCT postId)
3	1
4	1
8	1
9	2
12	1
14	2
19	2
24	1
29	1
44	1
47	1
48	1
64	1
71	2
- Action Output:** Shows the execution history:

Time	Action	Response	Duration / Fetch Time
35	21:56:23	SELECT userId, userName, expirationDate, COUNT(postId) FROM User JOIN UserPost USING (userId) GROUP BY userId LIMIT 15;	0.021 sec / 0.000014...
36	21:56:07	SELECT userId, userName, expirationDate, COUNT(postId) FROM User JOIN UserPost USING (userId) GROUP BY userId LIMIT 15;	0.040 sec / 0.00019...
37	21:56:13	SELECT userId, userName, expirationDate, COUNT(postId) FROM User JOIN UserPost USING (userId) GROUP BY userId LIMIT 15;	0.021 sec / 0.000022...
38	21:56:43	SELECT userId, userName, expirationDate, COUNT(postId) FROM User JOIN UserPost USING (userId) GROUP BY userId LIMIT 15;	0.020 sec / 0.000031...
39	21:56:50	SELECT userId, COUNT(DISTINCT postId) FROM UserPost	0.018 sec / 0.000012...

<Query 2> This query returns the userId, user name, expiration date, and the number of posts that have an expiration date no more than 2022-01-01, and the user name contains 'en'.

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** db1
- Tables:** UserPost
- Query 1:**

```
1 • SELECT userId, userName, expirationDate, COUNT(postId)
2   FROM User JOIN Post USING (userId)
3   WHERE expirationDate < ("2022-01-01") AND userName LIKE '%en%'
4   GROUP BY userId, expirationDate LIMIT 15;
```
- Result Grid:** Shows the results of the query:

userId	userName	expirationDate	COUNT(postId)
30	Ariene	2019-12-01	1
55	Damien	2019-09-09	1
106	Karen	2018-04-24	1
109	Brent	2016-01-02	1
119	Darren	2015-10-15	1
119	Darren	2012-11-05	1
119	Darren	2002-08-26	1
119	Darren	2002-07-27	1
120	Darren	2008-09-27	1
126	Bennie775	2004-08-06	1
126	Bennie775	2010-02-03	1
126	Bennie775	2002-12-17	1
126	Bennie775	2014-04-23	1
140	Denai	2015-04-16	1
- Action Output:** Shows the execution history:

Time	Action	Response	Duration / Fetch Time
35	21:56:23	SELECT userId, userName, expirationDate, COUNT(postId) FROM User JOIN Post USING (userId) WHERE expirationDate < ("2022-01-01") AND userName LIKE '%en%' GROUP BY userId, expirationDate LIMIT 15;	0.021 sec / 0.000014...
36	21:56:07	SELECT userId, userName, expirationDate, COUNT(postId) FROM User JOIN Post USING (userId) WHERE expirationDate < ("2022-01-01") AND userName LIKE '%en%' GROUP BY userId, expirationDate LIMIT 15;	0.040 sec / 0.00019...
37	21:56:13	SELECT userId, userName, expirationDate, COUNT(postId) FROM User JOIN Post USING (userId) WHERE expirationDate < ("2022-01-01") AND userName LIKE '%en%' GROUP BY userId, expirationDate LIMIT 15;	0.021 sec / 0.000022...
38	21:56:43	SELECT userId, userName, expirationDate, COUNT(postId) FROM User JOIN Post USING (userId) WHERE expirationDate < ("2022-01-01") AND userName LIKE '%en%' GROUP BY userId, expirationDate LIMIT 15;	0.020 sec / 0.000031...
39	21:56:50	SELECT userId, COUNT(DISTINCT postId) FROM UserPost	0.018 sec / 0.000012...

# Query Indexing Performance

This screenshot shows the initial performance for query 1 before our index design.

The screenshot shows the MySQL Workbench interface with the schema db1 selected. In the central pane, a query window displays the following SQL code:

```
1 EXPLAIN ANALYZE (
2   SELECT userId, COUNT(DISTINCT postId)
3     FROM User JOIN UserPost USING (userId)
4   GROUP BY userId LIMIT 15)
```

The results pane shows the execution log with the following entries:

Action	Time	Response	Duration / Fetch Time
EXPLAIN ANALYZE (	21:56:43	SELECT userId, userName, expirationDate, COUNT(postId) FROM User JOIN Post USING (userId) WHERE expirationDate < ("2022-01-01") AND userName LIKE '%en%' GROUP BY userId, expirationDate LIMIT 15	0.020 sec / 0.000031...
SELECT userId, COUNT(DISTINCT postId) FROM...	21:56:50	15 row(s) returned	0.018 sec / 0.000012...
EXPLAIN ANALYZE	21:58:05	Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL se...	0.021 sec
EXPLAIN ANALYZE ( SELECT userId, userName, e...	21:58:30	1 row(s) returned	0.022 sec / 0.000013...
EXPLAIN ANALYZE ( SELECT userId, COUNT(DISTI...)	22:01:01	1 row(s) returned	0.020 sec / 0.000015...

This screenshot shows the initial performance for query 2 before our index design.

The screenshot shows the MySQL Workbench interface with the schema db1 selected. In the central pane, a query window displays the following SQL code:

```
1 EXPLAIN ANALYZE (
2   SELECT userId, userName, expirationDate, COUNT(postId)
3     FROM User JOIN Post USING (userId)
4   WHERE expirationDate < ("2022-01-01") AND userName LIKE '%en%'
5   GROUP BY userId, expirationDate LIMIT 15)
```

The results pane shows the execution log with the following entries:

Action	Time	Response	Duration / Fetch Time
EXPLAIN ANALYZE (	21:56:13	SELECT userId, userName, expirationDate, COUNT(postId) FROM User JOIN Post USING (userId) WHERE expirationDate < ("2022-01-01") AND userName LIKE '%en%' GROUP BY userId, expirationDate LIMIT 15	0.021 sec / 0.000022...
SELECT userId, userName, expirationDate, COUNT...	21:56:43	86 row(s) returned	0.020 sec / 0.000031...
SELECT userId, COUNT(DISTINCT postId) FROM...	21:56:50	15 row(s) returned	0.018 sec / 0.000012...
EXPLAIN ANALYZE	21:58:05	Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL se...	0.021 sec
EXPLAIN ANALYZE ( SELECT userId, userName, e...	21:58:30	1 row(s) returned	0.022 sec / 0.000013...

58	22:31:12	EXPLAIN ANALYZE ( SELECT user... 1 row(s) returned	0.021 sec / 0.000008...	Query 1
59	22:34:00	CREATE INDEX indexUserPost O... 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.245 sec	
60	22:34:13	EXPLAIN ANALYZE ( SELECT user... 1 row(s) returned	0.022 sec / 0.000030...	
61	22:35:05	ALTER TABLE UserPost DROP IN... 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.036 sec	
62	22:35:39	CREATE INDEX indexUserON Us... 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.067 sec	
63	22:35:47	EXPLAIN ANALYZE ( SELECT user... 1 row(s) returned	0.022 sec / 0.000018...	
64	22:37:30	ALTER TABLE User DROP INDEX i... 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.046 sec	
65	22:37:45	CREATE INDEX indexUserPost O... 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.073 sec	
66	22:37:52	EXPLAIN ANALYZE ( SELECT user... 1 row(s) returned	0.271 sec / 0.000009...	
67	22:42:10	ALTER TABLE UserPost DROP IN... 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.036 sec	
68	22:42:19	EXPLAIN ANALYZE ( SELECT user... 1 row(s) returned	0.111 sec / 0.000012...	Query 2
69	22:43:28	) Error Code: 1064. You have an error in your SQL syntax; check the manual that correspond...	0.020 sec	
70	22:44:05	CREATE INDEX indexPostId ON P... 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.064 sec	
71	22:44:19	Id -- ON Post (postId); Error Code: 1064. You have an error in your SQL syntax; check the manual that correspond...	0.237 sec	
72	22:44:25	Id -- ON Post (postId); Error Code: 1064. You have an error in your SQL syntax; check the manual that correspond...	0.019 sec	
73	22:45:06	EXPLAIN ANALYZE ( SELECT user... 1 row(s) returned	0.093 sec / 0.000007...	
74	22:47:07	ALTER TABLE Post DROP INDEX i... 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.040 sec	
75	22:47:51	CREATE INDEX indexUserON Us... 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.065 sec	
76	22:48:05	EXPLAIN ANALYZE ( SELECT user... 1 row(s) returned	0.063 sec / 0.00000...	
78	23:09:03	CREATE INDEX indexPost ON Pos... 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.089 sec	
79	23:09:09	EXPLAIN ANALYZE ( SELECT user... 1 row(s) returned	0.021 sec / 0.000011...	
23:14:09		CREATE INDEX indexPost ON Pos... 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.058 sec	
23:14:22		EXPLAIN ANALYZE ( SELECT user... 1 row(s) returned	0.021 sec / 0.000009...	

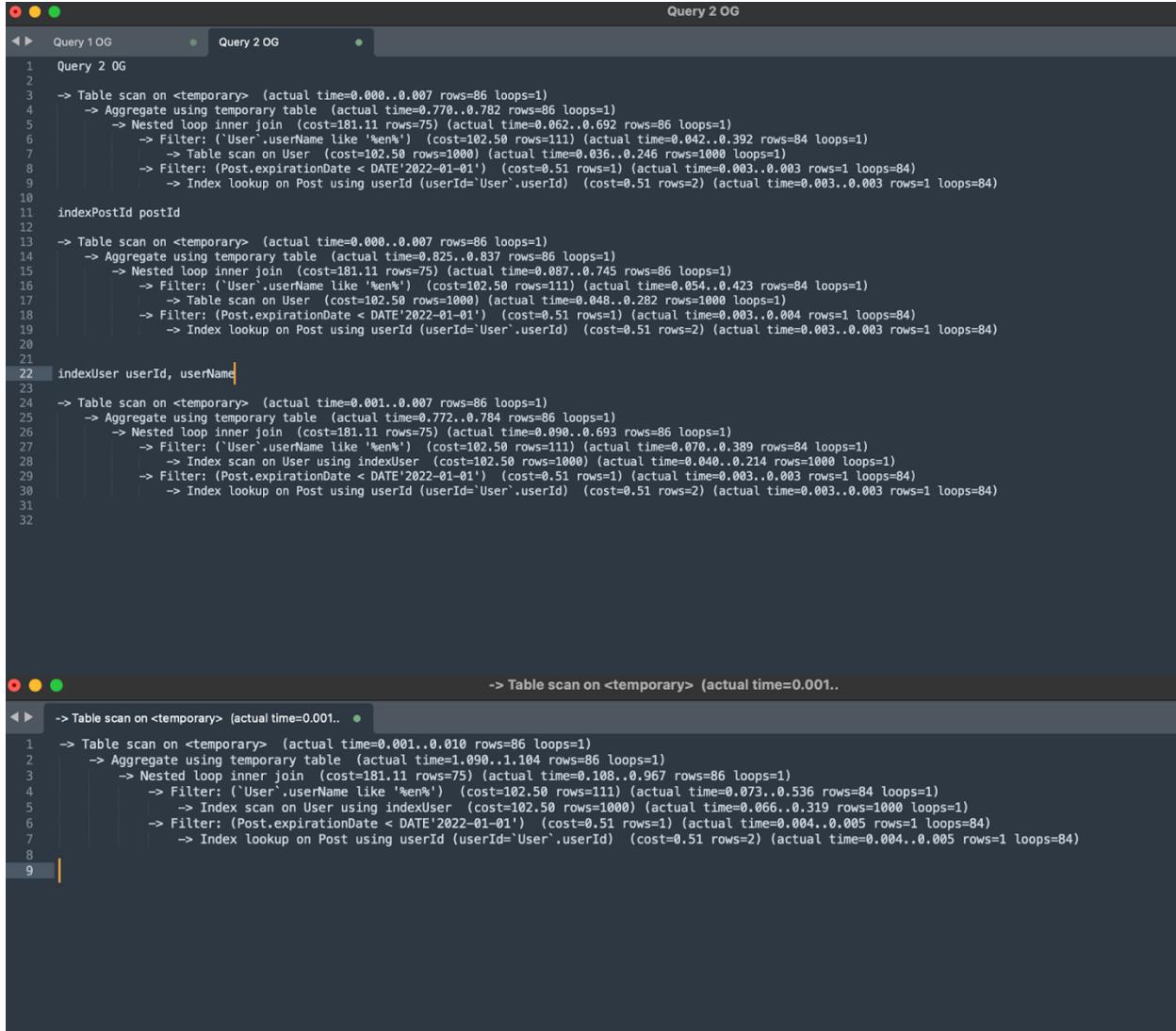
<Query 1 indexing>

```
Query 1 OG          Query 1 OG
1  Query 1 OG
2
3  -> Group aggregate: count(distinct UserPost.postId) (actual time=1.561..1.722 rows=632 loops=1)
4    -> Sort: <temporary>.userId (actual time=1.557..1.598 rows=1000 loops=1)
5      -> Table scan on <temporary> (actual time=0.001..0.037 rows=1000 loops=1)
6        -> Temporary table (actual time=1.287..1.386 rows=1000 loops=1)
7          -> Nested loop inner join (cost=450.75 rows=1000) (actual time=0.848..1.173 rows=1000 loops=1)
8            -> Index scan on UserPost using postId (cost=100.75 rows=1000) (actual time=0.037..0.208 rows=1000 loops=1)
9              -> Single-row index lookup on User using PRIMARY (userId=UserPost.userId) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1000)
10
11 indexUser userId
12
13 -> Group aggregate: count(distinct UserPost.postId) (actual time=1.449..1.623 rows=632 loops=1)
14   -> Sort: <temporary>.userId (actual time=1.445..1.491 rows=1000 loops=1)
15     -> Table scan on <temporary> (actual time=0.000..0.037 rows=1000 loops=1)
16       -> Temporary table (actual time=1.190..1.289 rows=1000 loops=1)
17         -> Nested loop inner join (cost=450.75 rows=1000) (actual time=0.825..1.086 rows=1000 loops=1)
18           -> Index scan on UserPost using postId (cost=100.75 rows=1000) (actual time=0.018..0.182 rows=1000 loops=1)
19             -> Single-row index lookup on User using PRIMARY (userId=UserPost.userId) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1000)
20
21 indexUserPost userId, postId
22
23 -> Group aggregate: count(distinct UserPost.postId) (actual time=1.525..1.687 rows=632 loops=1)
24   -> Sort: <temporary>.userId (actual time=1.521..1.563 rows=1000 loops=1)
25     -> Table scan on <temporary> (actual time=0.001..0.062 rows=1000 loops=1)
26       -> Temporary table (actual time=1.238..1.362 rows=1000 loops=1)
27         -> Nested loop inner join (cost=450.75 rows=1000) (actual time=0.845..1.139 rows=1000 loops=1)
28           -> Index scan on UserPost using postId (cost=100.75 rows=1000) (actual time=0.034..0.197 rows=1000 loops=1)
29             -> Single-row index lookup on User using PRIMARY (userId=UserPost.userId) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1000)
30
31
```

```
-> Group aggregate: count(distinct UserPost.postId)
1  -> Group aggregate: count(distinct UserPost.postId) (actual time=1.480..1.681 rows=632 loops=1)
2    -> Sort: <temporary>.userId (actual time=1.477..1.520 rows=1000 loops=1)
3      -> Table scan on <temporary> (actual time=0.001..0.037 rows=1000 loops=1)
4        -> Temporary table (actual time=1.221..1.319 rows=1000 loops=1)
5          -> Nested loop inner join (cost=450.75 rows=1000) (actual time=0.829..1.115 rows=1000 loops=1)
6            -> Index scan on UserPost using postId (cost=100.75 rows=1000) (actual time=0.022..0.188 rows=1000 loops=1)
7              -> Single-row index lookup on User using PRIMARY (userId=UserPost.userId) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1000)
8
9
```

- We can not find any column for the best indexing in this case.
- The original query was doing the index scan before we created the index.
- As the pictures above, the queries are doing all the same actions with or without index. Therefore, we did not see any difference in the results.

## <Query 2 indexing>



```

1 Query 2 OG
2
3 -> Table scan on <temporary> (actual time=0.000..0.007 rows=86 loops=1)
4   -> Aggregate using temporary table (actual time=0.770..0.782 rows=86 loops=1)
5     -> Nested loop inner join (cost=181.11 rows=75) (actual time=0.062..0.692 rows=86 loops=1)
6       -> Filter: ('User'.userName like '%en%') (cost=102.50 rows=11) (actual time=0.042..0.392 rows=84 loops=1)
7         -> Table scan on User (cost=102.50 rows=1000) (actual time=0.036..0.246 rows=1000 loops=1)
8       -> Filter: (Post.expirationDate < DATE'2022-01-01') (cost=0.51 rows=1) (actual time=0.003..0.003 rows=1 loops=84)
9         -> Index lookup on Post using userId (userId='User'.userId) (cost=0.51 rows=2) (actual time=0.003..0.003 rows=1 loops=84)
10
11 indexPostId postId
12
13 -> Table scan on <temporary> (actual time=0.000..0.007 rows=86 loops=1)
14   -> Aggregate using temporary table (actual time=0.825..0.837 rows=86 loops=1)
15     -> Nested loop inner join (cost=181.11 rows=75) (actual time=0.087..0.745 rows=86 loops=1)
16       -> Filter: ('User'.userName like '%en%') (cost=102.50 rows=11) (actual time=0.054..0.423 rows=84 loops=1)
17         -> Table scan on User (cost=102.50 rows=1000) (actual time=0.048..0.282 rows=1000 loops=1)
18       -> Filter: (Post.expirationDate < DATE'2022-01-01') (cost=0.51 rows=1) (actual time=0.003..0.004 rows=1 loops=84)
19         -> Index lookup on Post using userId (userId='User'.userId) (cost=0.51 rows=2) (actual time=0.003..0.003 rows=1 loops=84)
20
21
22 indexUser userId, userName
23
24 -> Table scan on <temporary> (actual time=0.001..0.007 rows=86 loops=1)
25   -> Aggregate using temporary table (actual time=0.772..0.784 rows=86 loops=1)
26     -> Nested loop inner join (cost=181.11 rows=75) (actual time=0.098..0.693 rows=86 loops=1)
27       -> Filter: ('User'.userName like '%en%') (cost=102.50 rows=11) (actual time=0.070..0.389 rows=84 loops=1)
28         -> Index scan on User using indexUser (cost=102.50 rows=1000) (actual time=0.040..0.214 rows=1000 loops=1)
29       -> Filter: (Post.expirationDate < DATE'2022-01-01') (cost=0.51 rows=1) (actual time=0.003..0.003 rows=1 loops=84)
30         -> Index lookup on Post using userId (userId='User'.userId) (cost=0.51 rows=2) (actual time=0.003..0.003 rows=1 loops=84)
31
32

```

-> Table scan on <temporary> (actual time=0.001..

```

-> Table scan on <temporary> (actual time=0.001..0.010 rows=86 loops=1)
1   -> Table scan on <temporary> (actual time=0.001..0.010 rows=86 loops=1)
2     -> Aggregate using temporary table (actual time=1.090..1.104 rows=86 loops=1)
3       -> Nested loop inner join (cost=181.11 rows=75) (actual time=0.108..0.967 rows=86 loops=1)
4         -> Filter: ('User'.userName like '%en%') (cost=102.50 rows=11) (actual time=0.073..0.536 rows=84 loops=1)
5           -> Index scan on User using indexUser (cost=102.50 rows=1000) (actual time=0.066..0.319 rows=1000 loops=1)
6         -> Filter: (Post.expirationDate < DATE'2022-01-01') (cost=0.51 rows=1) (actual time=0.004..0.005 rows=1 loops=84)
7           -> Index lookup on Post using userId (userId='User'.userId) (cost=0.51 rows=2) (actual time=0.004..0.005 rows=1 loops=84)
8
9

```

```

CREATE INDEX indexUser
ON User (userId, userName);

```

```

CREATE INDEX indexPost
ON Post (postId);

```

- We find out that creating 2 indexes on table User columns (userId, userName) and table Post column (postId) will increase 4~5x efficiency for our query.