Stage 3: Database Implementation and Indexing

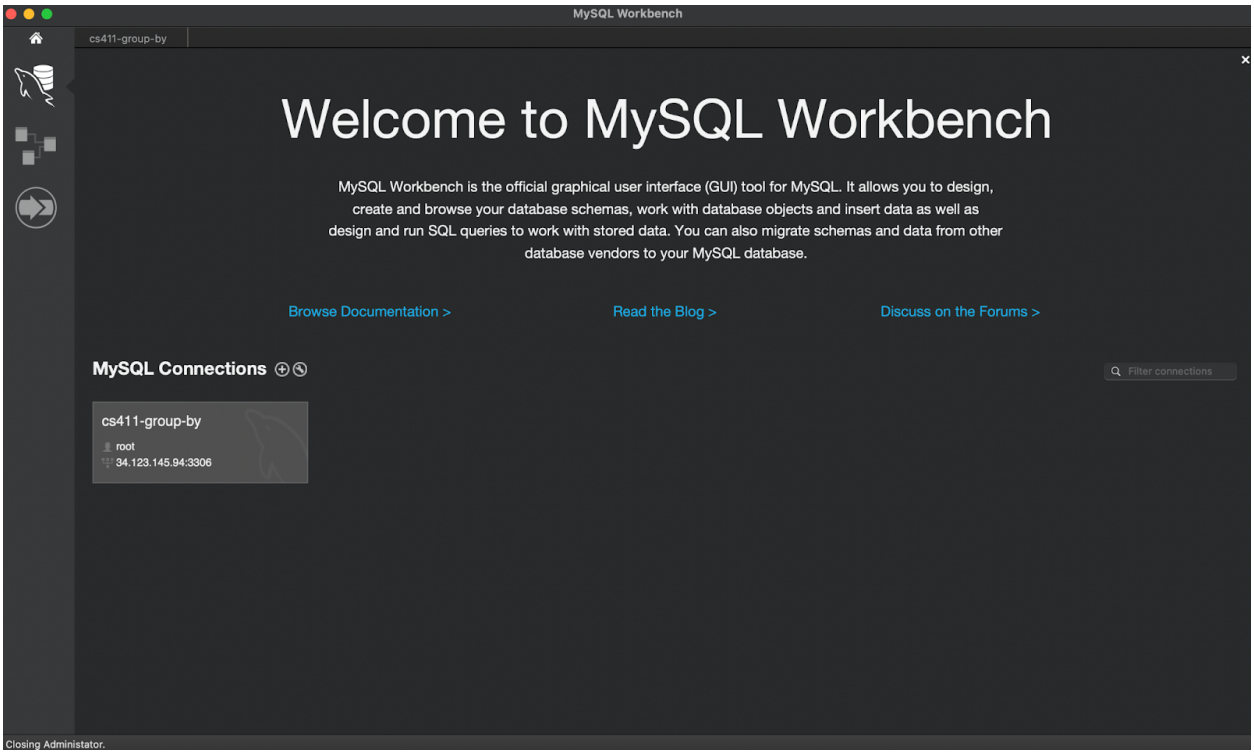
Data Definition Language (DDL) commands

```
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

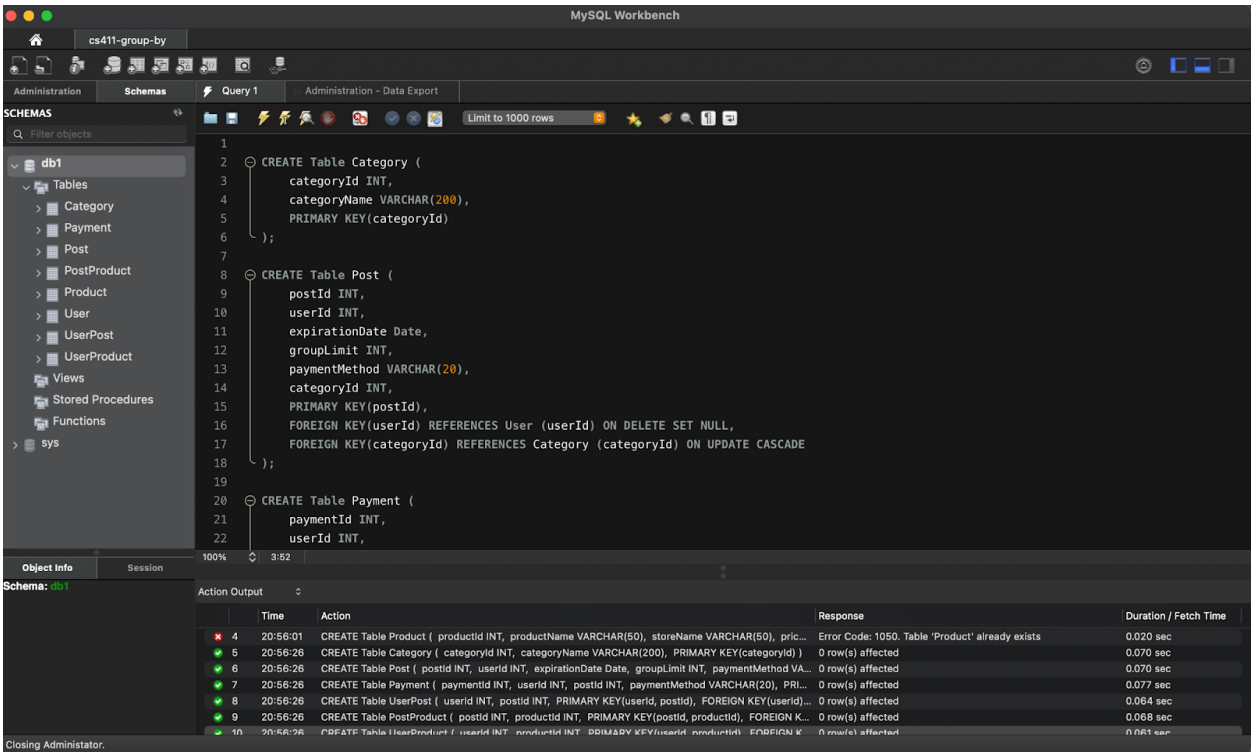
1. The screenshot below shows that we have successfully connected to the GCP via MySQL Workbench.



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

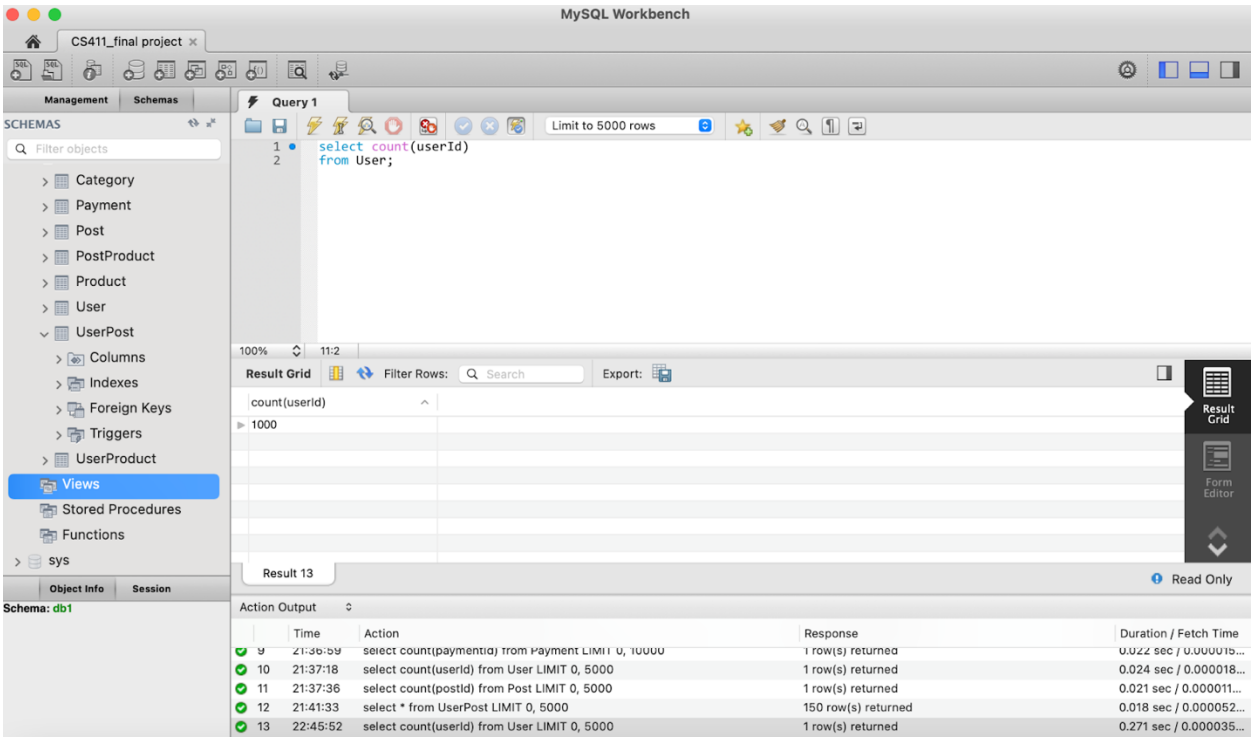
   Main Tables:

- User
- Post
- Product
- Category
- Payment

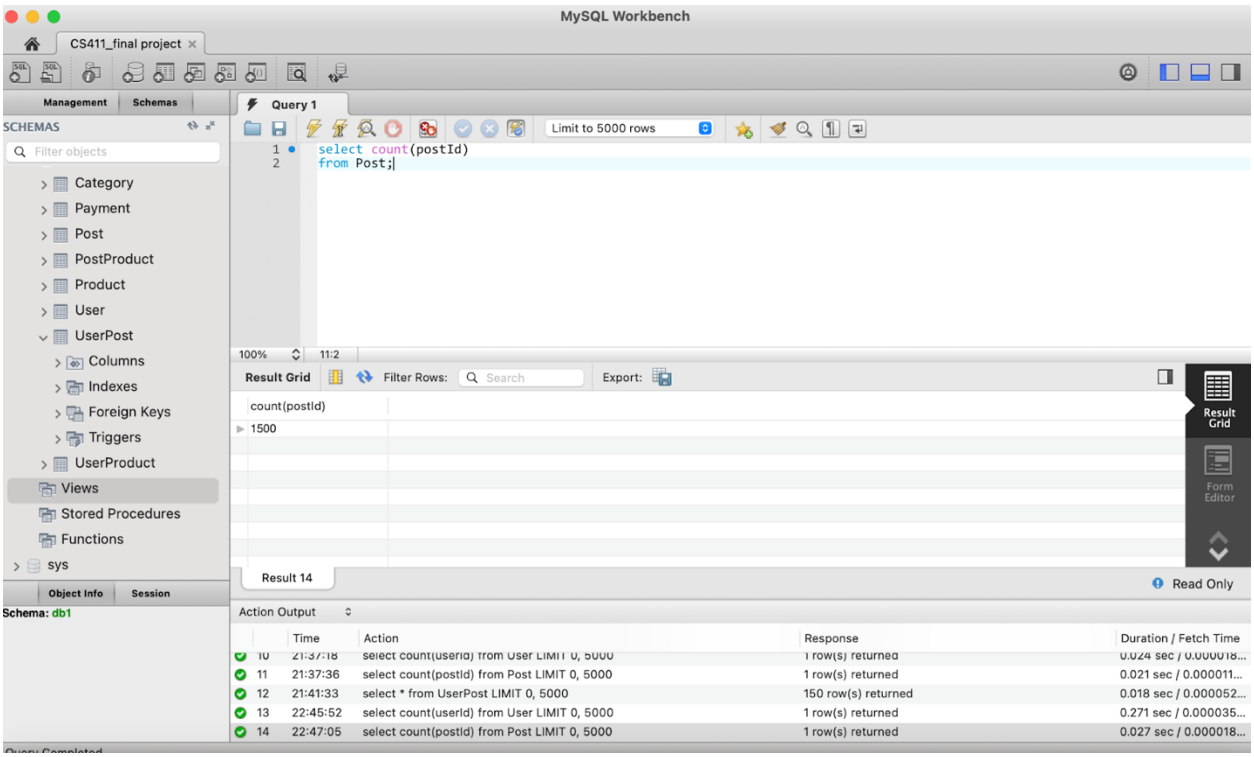


3. The screenshots below show that we have 3 tables (User, Post, Payment) that contain at least 1000 rows in each table.

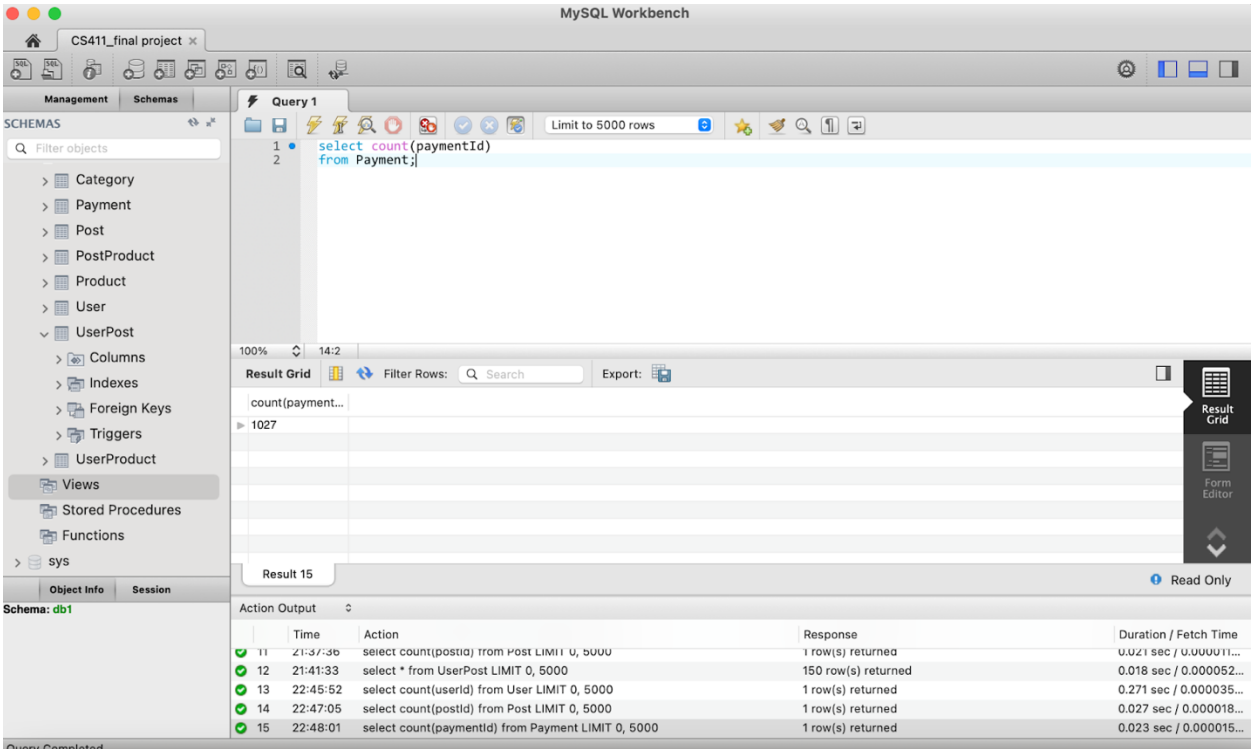
<User table>



<Post table>

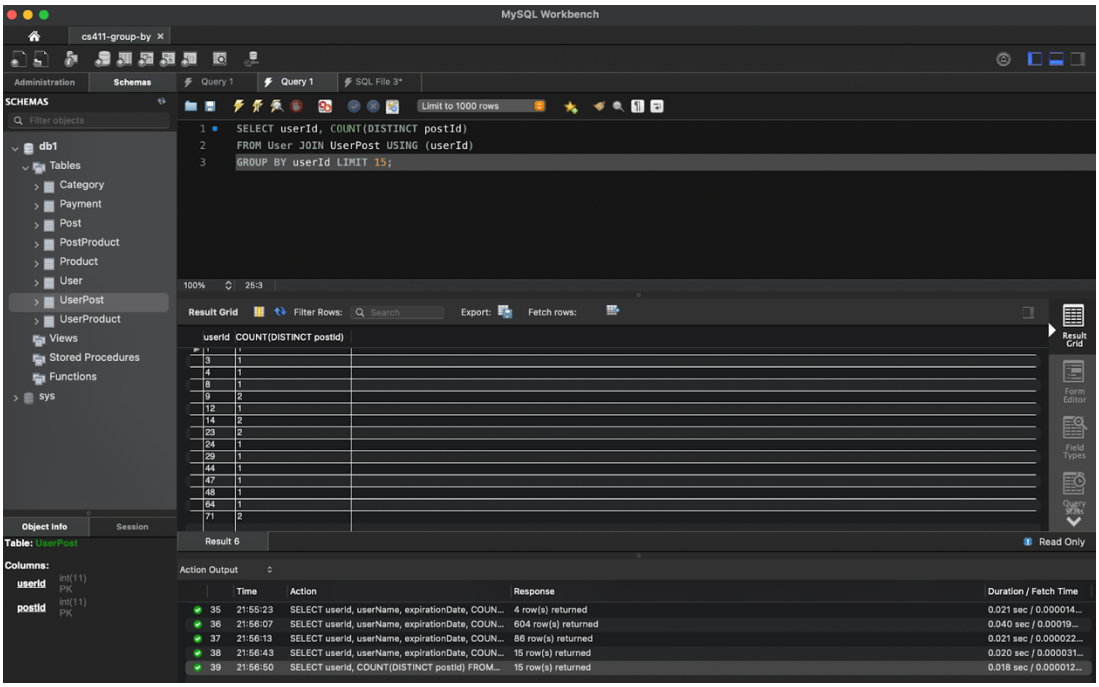


<Payment table>

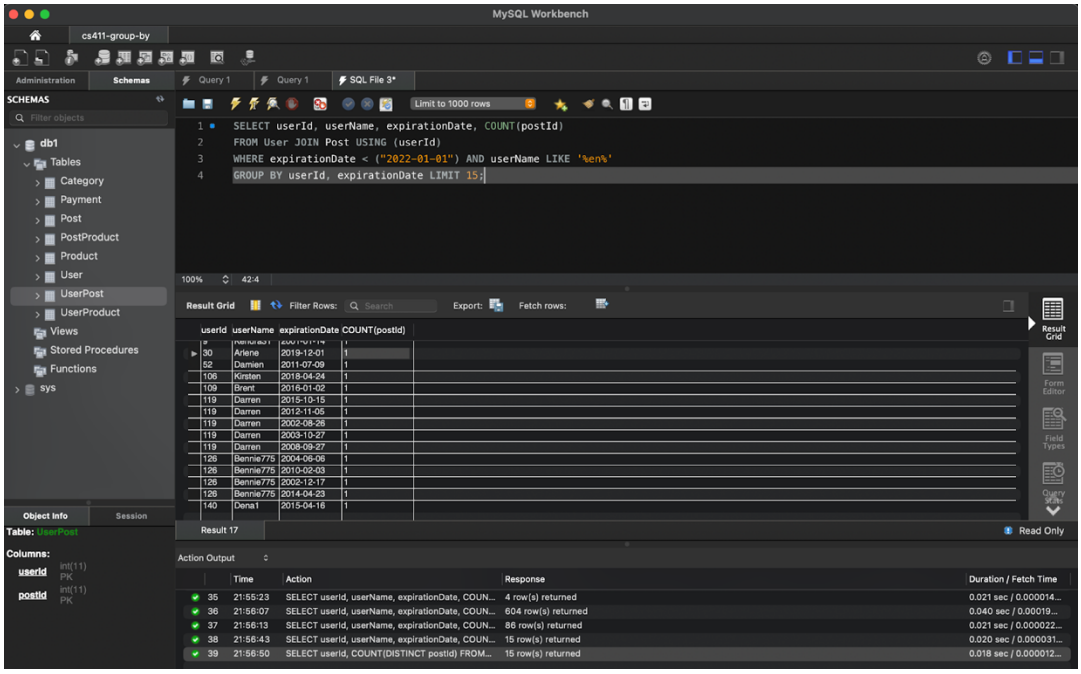


Advanced Queries

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

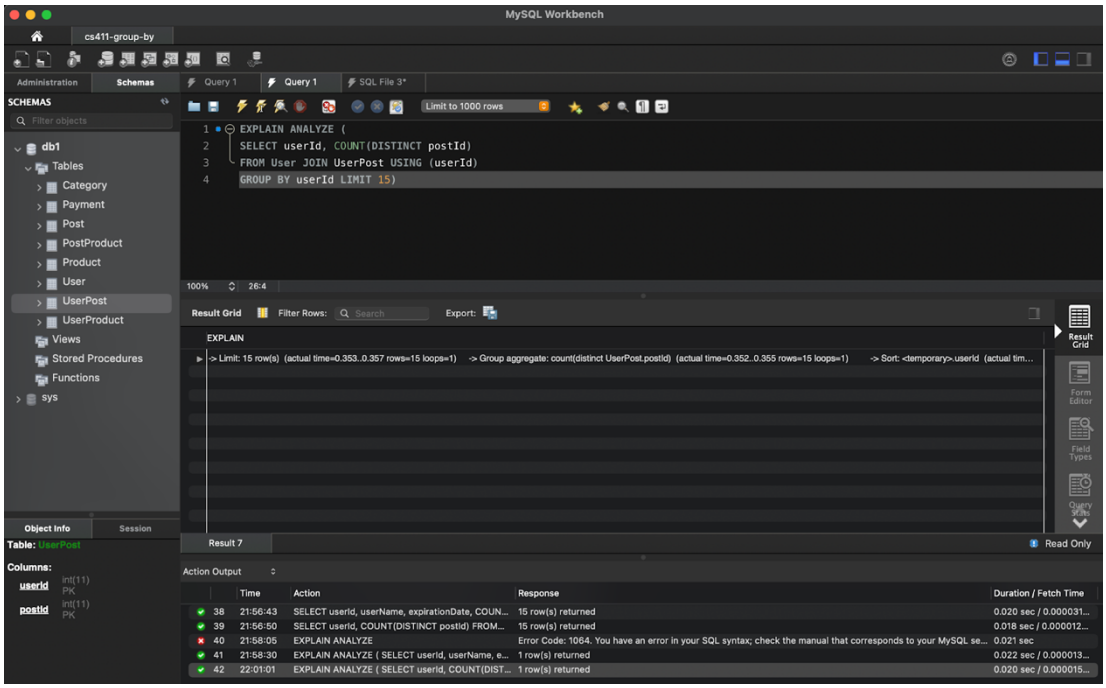


<Query 2> This query returns the userId, user name, expiration date, and the number of posts that have an expiration date no more than 2022-01-01, and the user name contains ‘en’.

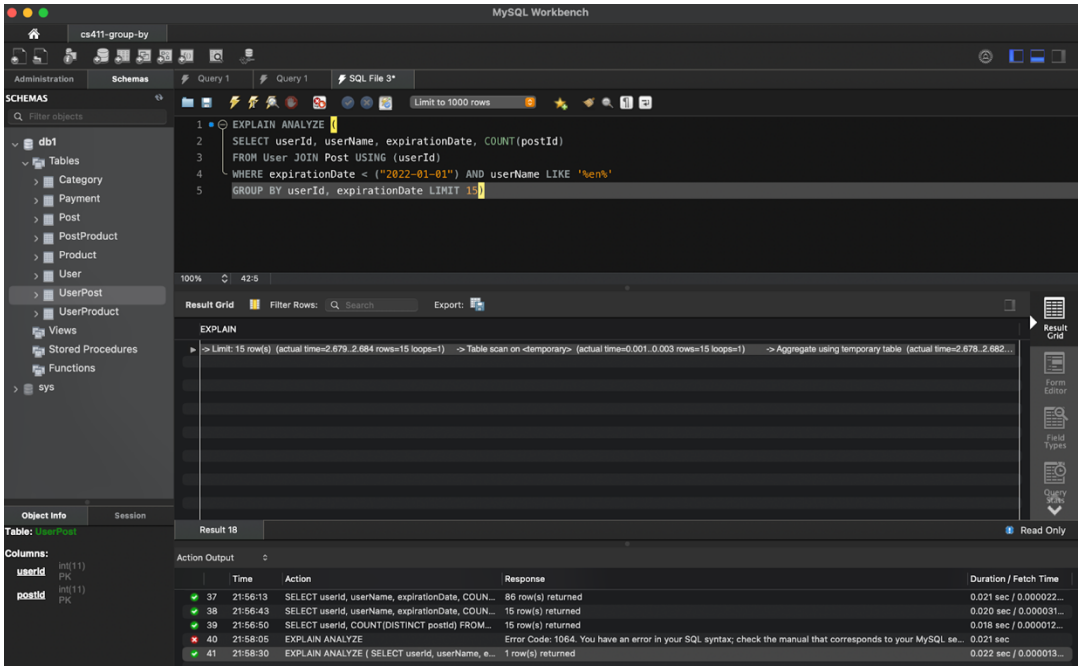


Query Indexing Performance

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



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

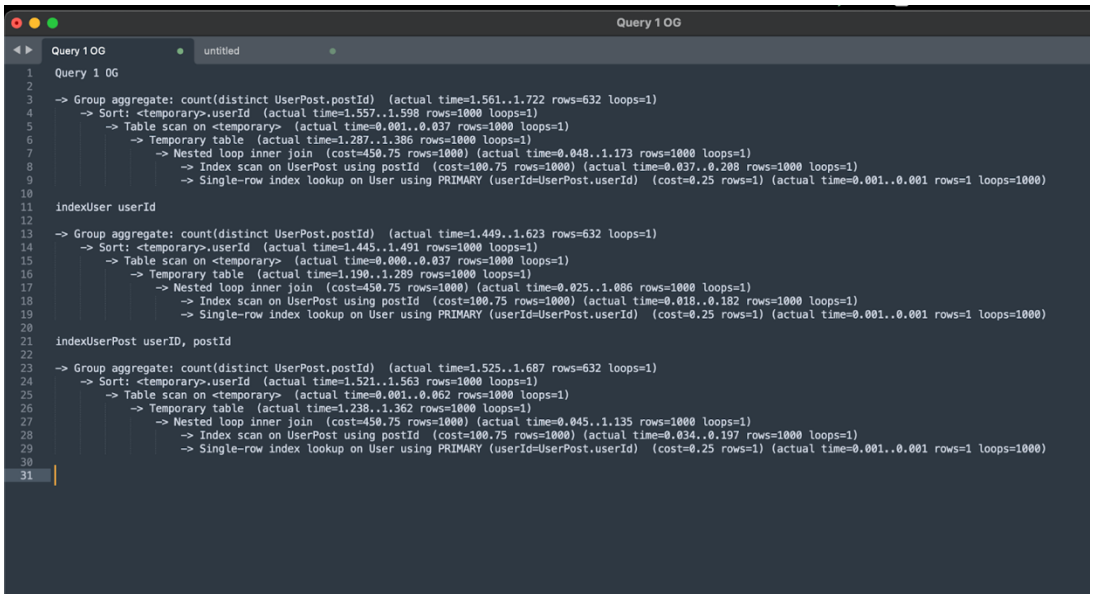


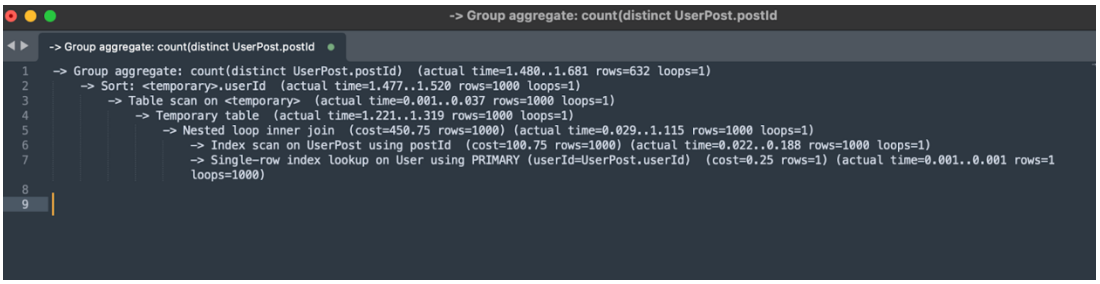
58	22:31:12	EXPLAIN ANALYZE (SELECT user...	1 row(s) returned	0.021 sec / 0.000008...
59	22:34:00	CREATE INDEX indexUserPost O...	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.245 sec
60	22:34:13	EXPLAIN ANALYZE (SELECT user...	1 row(s) returned	0.022 sec / 0.000030...
61	22:35:05	ALTER TABLE UserPost DROP IN...	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.036 sec
62	22:35:39	CREATE INDEX indexUser ON 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...
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 indexUser ON 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 indexUser ON Us...	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

Query 2

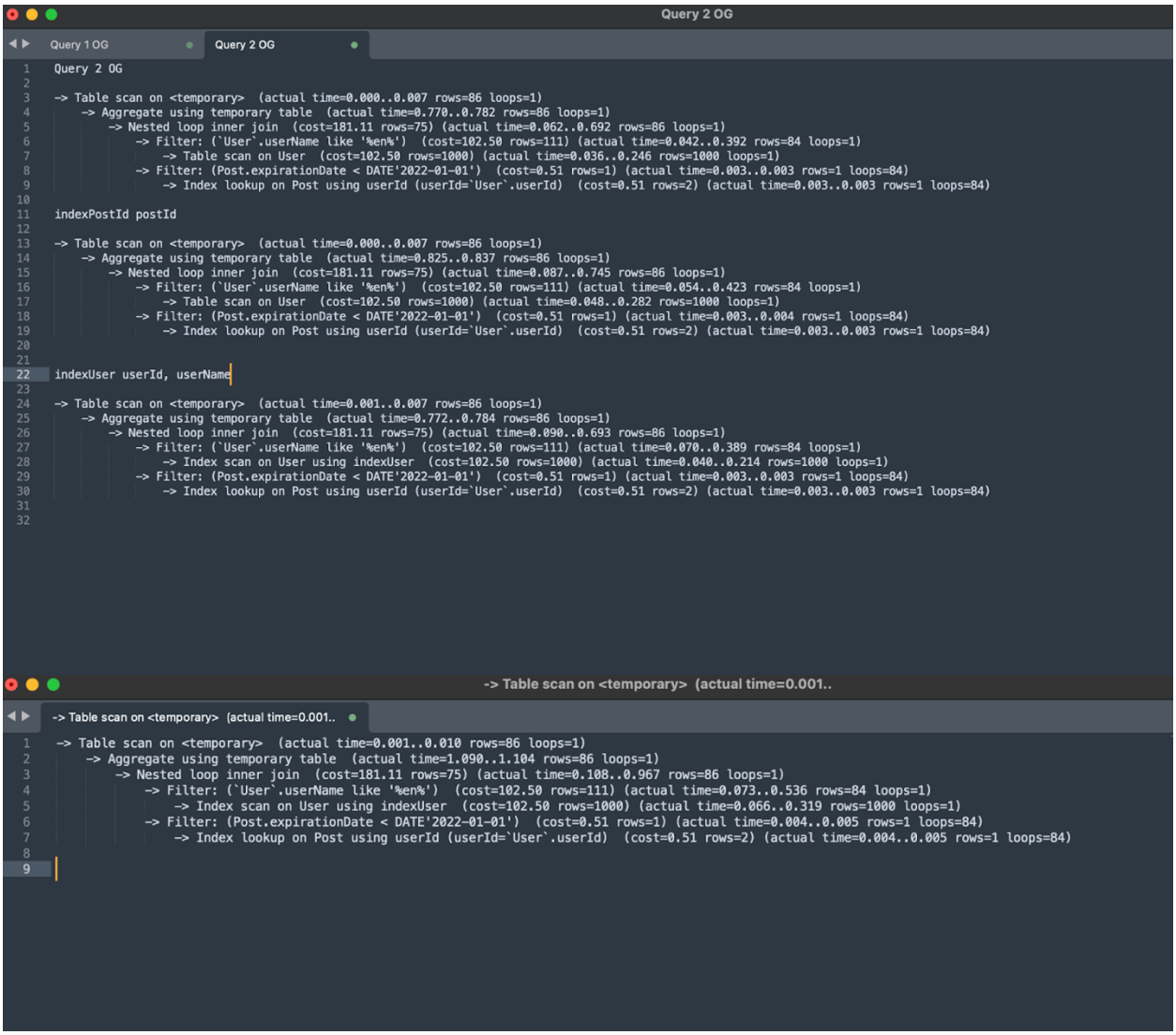
<Query 1 indexing>





- 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>




```
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.