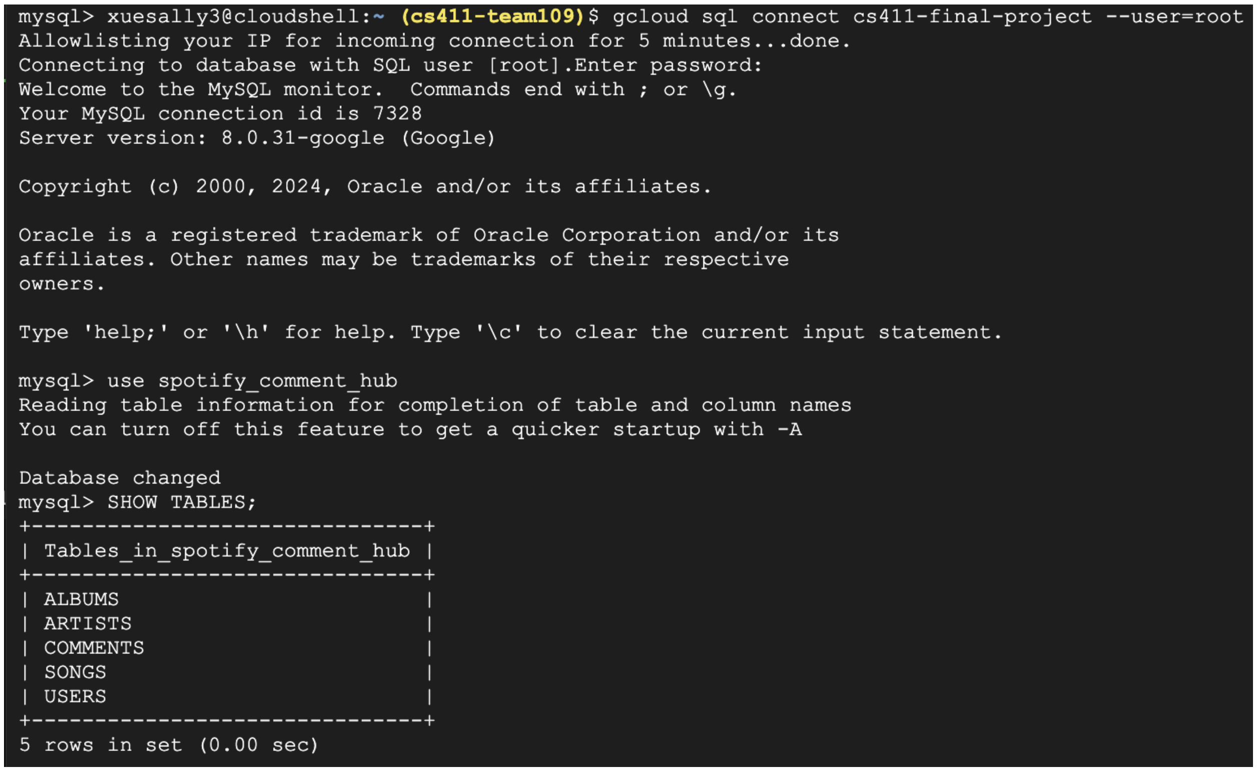
Database Design

Team members: Sally Xue, Pradyumann Singhal, Lily Zhang, Joseph Schanne

GCP Connection Screenshot

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



```
mysql> xuesally3@cloudshell:~ (cs411-team109)$ gcloud sql connect cs411-final-project --user=root
Allowlisting your IP for incoming connection for 5 minutes...done.
Connecting to database with SQL user [root].Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7328
Server version: 8.0.31-google (Google)

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

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

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

Database changed
mysql> SHOW TABLES;
+-------------------------------+
| Tables_in_spotify_comment_hub |
+-------------------------------+
| ALBUMS                        |
| ARTISTS                       |
| COMMENTS                      |
| SONGS                         |
| USERS                         |
+-------------------------------+
5 rows in set (0.00 sec)
```

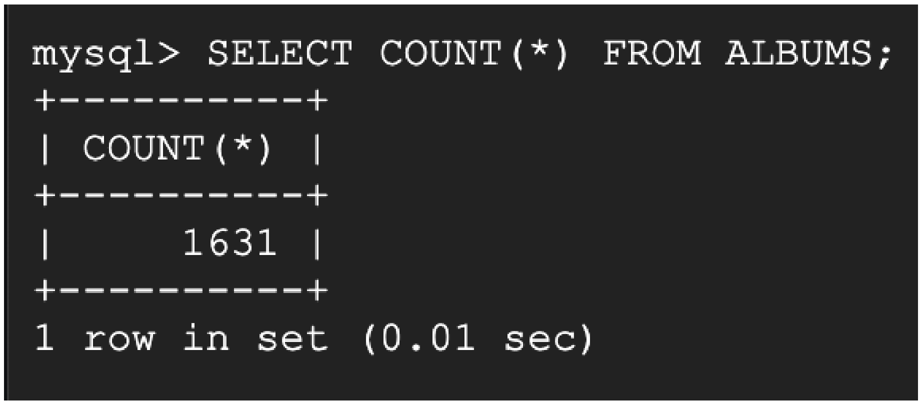
DDL Commands

```
CREATE TABLE ARTISTS (
    ArtistID VARCHAR(255) PRIMARY KEY,
    ArtistName VARCHAR(255)
)
```

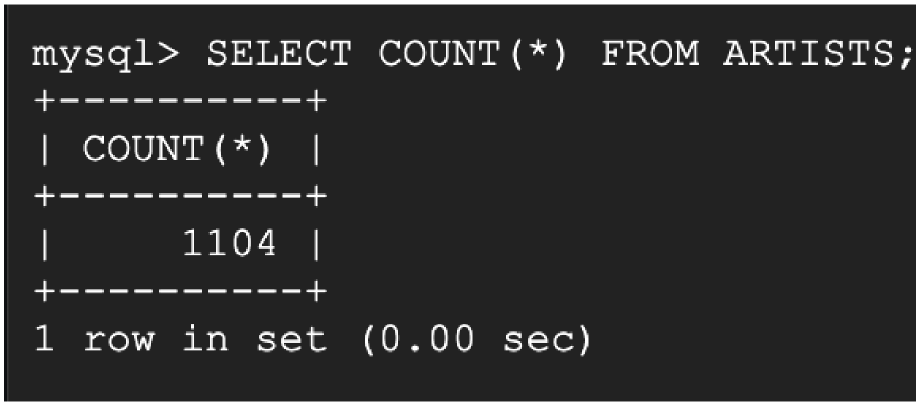
```
CREATE TABLE USERS (
    UserID VARCHAR(255) PRIMARY KEY,
    Username VARCHAR(255),
    Password VARCHAR(255),
    Email VARCHAR(255)
);

CREATE TABLE ALBUMS (
    AlbumID VARCHAR(255) PRIMARY KEY,
    ArtistID VARCHAR(255),
    AlbumName VARCHAR(255),
    Description VARCHAR(255),
    FOREIGN KEY (ArtistID) REFERENCES ARTISTS(ArtistID)
);

CREATE TABLE SONGS (
    SongID VARCHAR(255) PRIMARY KEY,
    SongName VARCHAR(255),
    ArtistID VARCHAR(255),
    AlbumID VARCHAR(255),
    ReleaseDate DATE,
    FOREIGN KEY (ArtistID) REFERENCES ARTISTS(ArtistID),
    FOREIGN KEY (AlbumID) REFERENCES ALBUMS(AlbumID)
);

CREATE TABLE COMMENTS (
    CommentID VARCHAR(255) PRIMARY KEY,
    UserID VARCHAR(255),
    SongID VARCHAR(255),
    CommentInfo VARCHAR(255),
    Rating INTEGER,
    CreatedOn DATE,
    ResponseTo VARCHAR(255),
    FOREIGN KEY (UserID) REFERENCES USERS(UserID),
    FOREIGN KEY (SongID) REFERENCES SONGS(SongID)
);
```

Table Row Count

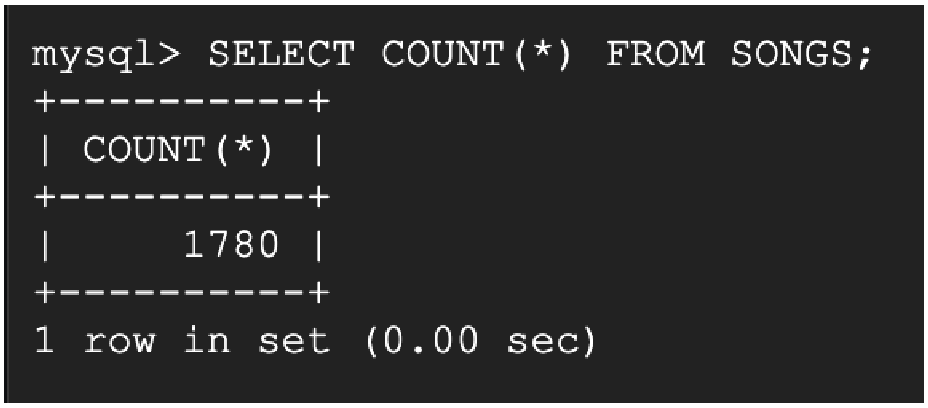
ALBUMS:


```
mysql> SELECT COUNT(*) FROM ALBUMS;
+----------+
| COUNT(*) |
+----------+
|     1631 |
+----------+
1 row in set (0.01 sec)
```

ARTISTS:


```
mysql> SELECT COUNT(*) FROM ARTISTS;
+----------+
| COUNT(*) |
+----------+
|     1104 |
+----------+
1 row in set (0.00 sec)
```

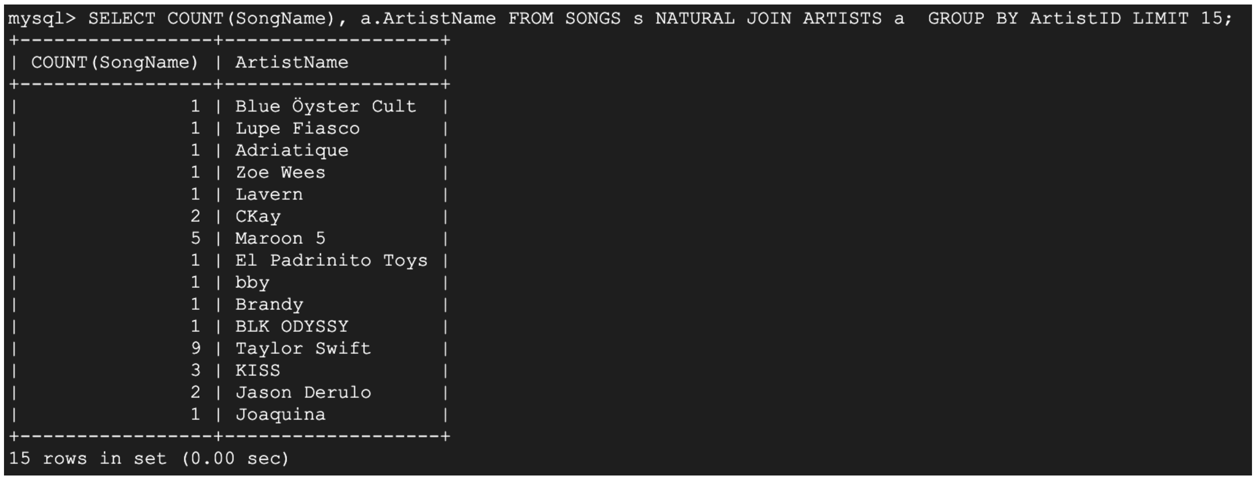
SONGS:


```
mysql> SELECT COUNT(*) FROM SONGS;
+----------+
| COUNT(*) |
+----------+
|     1780 |
+----------+
1 row in set (0.00 sec)
```

Advanced Queries with Screenshots

- **Song Count per Artist**

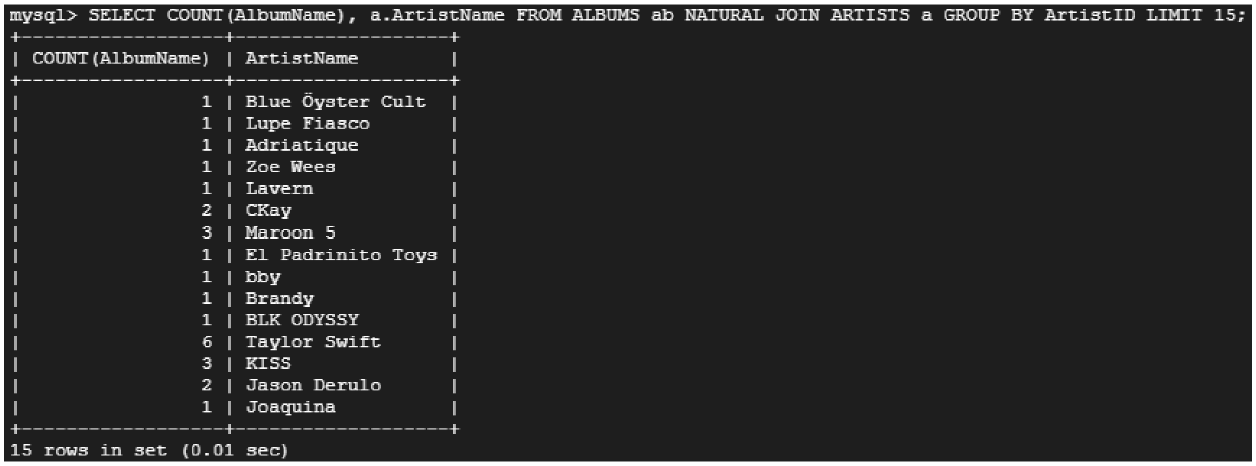
  SELECT COUNT(SongName), a.ArtistName FROM SONGS s NATURAL JOIN ARTISTS a GROUP BY ArtistID LIMIT 15;



```
mysql> SELECT COUNT(SongName), a.ArtistName FROM SONGS s NATURAL JOIN ARTISTS a  GROUP BY ArtistID LIMIT 15;
+-----------------+-------------------+
| COUNT(SongName) | ArtistName        |
+-----------------+-------------------+
|               1 | Blue Öyster Cult  |
|               1 | Lupe Fiasco       |
|               1 | Adriatique        |
|               1 | Zoe Wees          |
|               1 | Lavern            |
|               2 | CKay              |
|               5 | Maroon 5          |
|               1 | El Padrinito Toys |
|               1 | bby               |
|               1 | Brandy            |
|               1 | BLK ODYSSY        |
|               9 | Taylor Swift      |
|               3 | KISS              |
|               2 | Jason Derulo      |
|               1 | Joaquina          |
+-----------------+-------------------+
15 rows in set (0.00 sec)
```

- **Album Count per Artist**

  SELECT COUNT(AlbumName), a.ArtistName FROM ALBUMS ab NATURAL JOIN ARTISTS a GROUP BY ArtistID;

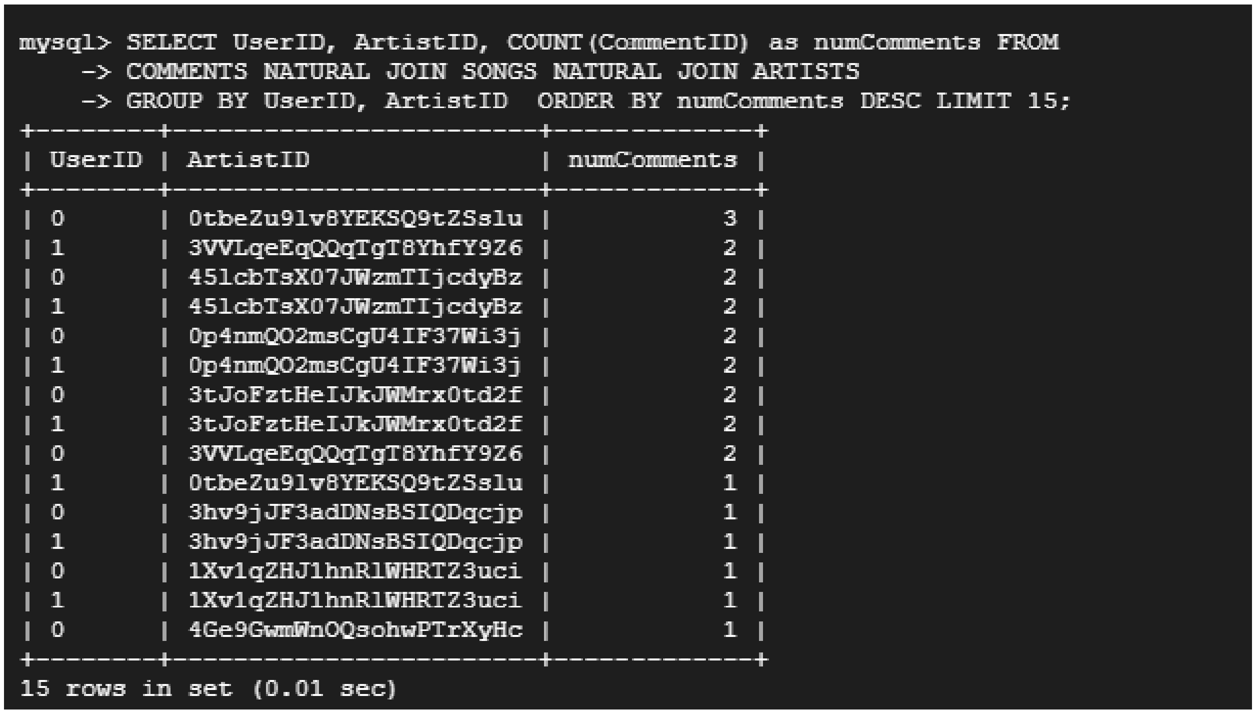


```
mysql> SELECT COUNT(AlbumName), a.ArtistName FROM ALBUMS ab NATURAL JOIN ARTISTS a GROUP BY ArtistID LIMIT 15;
+------------------+-------------------+
| COUNT(AlbumName) | ArtistName        |
+------------------+-------------------+
|                1 | Blue Öyster Cult  |
|                1 | Lupe Fiasco       |
|                1 | Adriatique        |
|                1 | Zoe Wees          |
|                1 | Lavern            |
|                2 | CKay              |
|                3 | Maroon 5          |
|                1 | El Padrinito Toys |
|                1 | bby               |
|                1 | Brandy            |
|                1 | BLK ODYSSY        |
|                6 | Taylor Swift      |
|                3 | KISS              |
|                2 | Jason Derulo      |
|                1 | Joaquina          |
+------------------+-------------------+
15 rows in set (0.01 sec)
```

- **Give User ID, ArtistID, Count of Comments**

  SELECT UserID, ArtistID, COUNT(CommentID) as numComments FROM COMMENTS NATURAL JOIN SONGS NATURAL JOIN ARTISTS

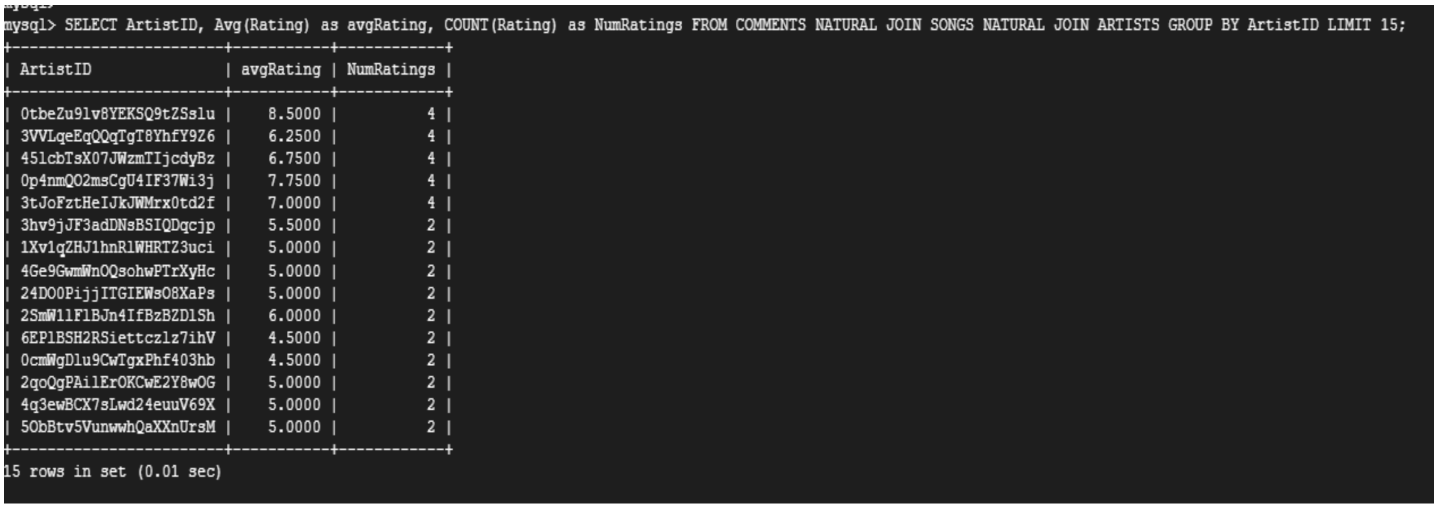
GROUP BY UserID, ArtistID ORDER BY numComments DESC LIMIT 15;



```
mysql> SELECT UserID, ArtistID, COUNT(CommentID) as numComments FROM
    -> COMMENTS NATURAL JOIN SONGS NATURAL JOIN ARTISTS
    -> GROUP BY UserID, ArtistID  ORDER BY numComments DESC LIMIT 15;
+--------+------------------------+-------------+
| UserID | ArtistID               | numComments |
+--------+------------------------+-------------+
| 0      | 0tbeZu9lv8YEKSQ9tZSslu |           3 |
| 1      | 3VVLqeEqQQqTgT8YhfY9Z6 |           2 |
| 0      | 45lcbTsX07JWzmTIjcdyBz |           2 |
| 1      | 45lcbTsX07JWzmTIjcdyBz |           2 |
| 0      | 0p4nmQO2msCgU4IF37Wi3j |           2 |
| 1      | 0p4nmQO2msCgU4IF37Wi3j |           2 |
| 0      | 3tJoFztHeIJkJWMrx0td2f |           2 |
| 1      | 3tJoFztHeIJkJWMrx0td2f |           2 |
| 0      | 3VVLqeEqQQqTgT8YhfY9Z6 |           2 |
| 1      | 0tbeZu9lv8YEKSQ9tZSslu |           1 |
| 0      | 3hv9jJF3adDNsBSIQDqcjp |           1 |
| 1      | 3hv9jJF3adDNsBSIQDqcjp |           1 |
| 0      | 1Xv1qZHJ1hnRlWHRTZ3uci |           1 |
| 1      | 1Xv1qZHJ1hnRlWHRTZ3uci |           1 |
| 0      | 4Ge9GwmWnOQsohwPTrXyHc |           1 |
+--------+------------------------+-------------+
15 rows in set (0.01 sec)
```

- **Get Avg Ratings of all artists based on their songs**

  SELECT ArtistID, Avg(Rating) as avgRating, COUNT(Rating) as NumRatings FROM COMMENTS NATURAL JOIN SONGS NATURAL JOIN ARTISTS GROUP BY ArtistID LIMIT 15;



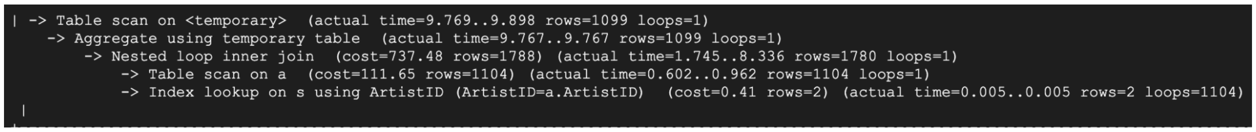
```
mysql> SELECT ArtistID, Avg(Rating) as avgRating, COUNT(Rating) as NumRatings FROM COMMENTS NATURAL JOIN SONGS NATURAL JOIN ARTISTS GROUP BY ArtistID LIMIT 15;
+------------------------+-----------+------------+
| ArtistID               | avgRating | NumRatings |
+------------------------+-----------+------------+
| 0tbeZu9lv8YEKSQ9tZSslu |    8.5000 |          4 |
| 3VVLqeEqQQqTgT8YhfY9Z6 |    6.2500 |          4 |
| 45lcbTsX07JWzmTIjcdyBz |    6.7500 |          4 |
| 0p4nmQO2msCgU4IF37Wi3j |    7.7500 |          4 |
| 3tJoFztHeIJkJWMrx0td2f |    7.0000 |          4 |
| 3hv9jJF3adDNsBSIQDqcjp |    5.5000 |          2 |
| 1Xv1qZHJ1hnRlWHRTZ3uci |    5.0000 |          2 |
| 4Ge9GwmWnOQsohwPTrXyHc |    5.0000 |          2 |
| 24DO0PijjITGIEWsO8XaPs |    5.0000 |          2 |
| 2SmW1lFlBJn4IfBzBZDlSh |    6.0000 |          2 |
| 6EPlBSH2RSiettczlz7ihV |    4.5000 |          2 |
| 0cmWgDlu9CwTgxPhf403hb |    4.5000 |          2 |
| 2qoQgPAilErOKCwE2Y8wOG |    5.0000 |          2 |
| 4q3ewBCX7sLwd24euuV69X |    5.0000 |          2 |
| 5ObBtv5VunwwhQaXXnUrsM |    5.0000 |          2 |
+------------------------+-----------+------------+
15 rows in set (0.01 sec)
```

Indexing Analysis

- **Song Count per Artist**

  SELECT COUNT(SongName), a.ArtistName, s.AlbumID FROM SONGS s NATURAL JOIN ARTISTS a GROUP BY ArtistID;

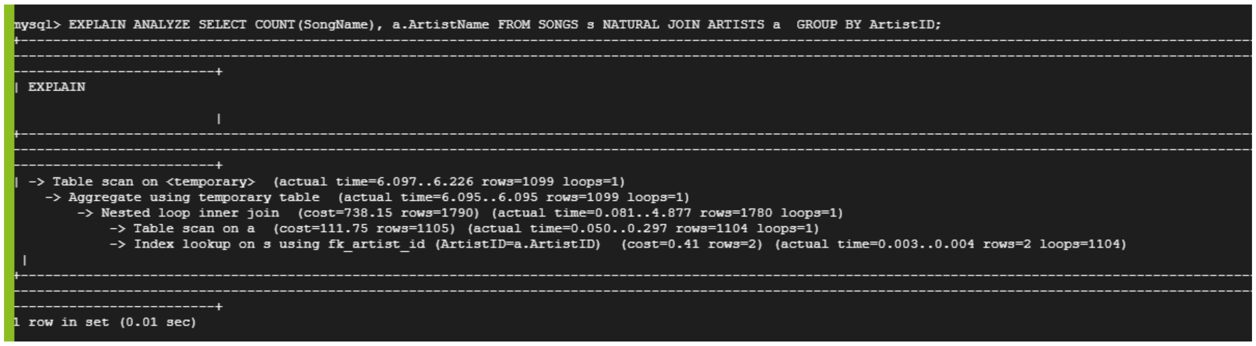
Default index:



```
| -> Table scan on <temporary>  (actual time=9.769..9.898 rows=1099 loops=1)
    -> Aggregate using temporary table  (actual time=9.767..9.767 rows=1099 loops=1)
        -> Nested loop inner join  (cost=737.48 rows=1788) (actual time=1.745..8.336 rows=1780 loops=1)
            -> Table scan on a  (cost=111.65 rows=1104) (actual time=0.602..0.962 rows=1104 loops=1)
            -> Index lookup on s using ArtistID (ArtistID=a.ArtistID)  (cost=0.41 rows=2) (actual time=0.005..0.005 rows=2 loops=1104)
 |
```

Analysis: The costliest part of this query is the nested loop join (row 3), which is the NATURAL JOIN part in the SQL query. Specifically the index lookup on s using `ArtistID`, where each row from a requires an index lookup on s, increasing the join cost to 737.48.

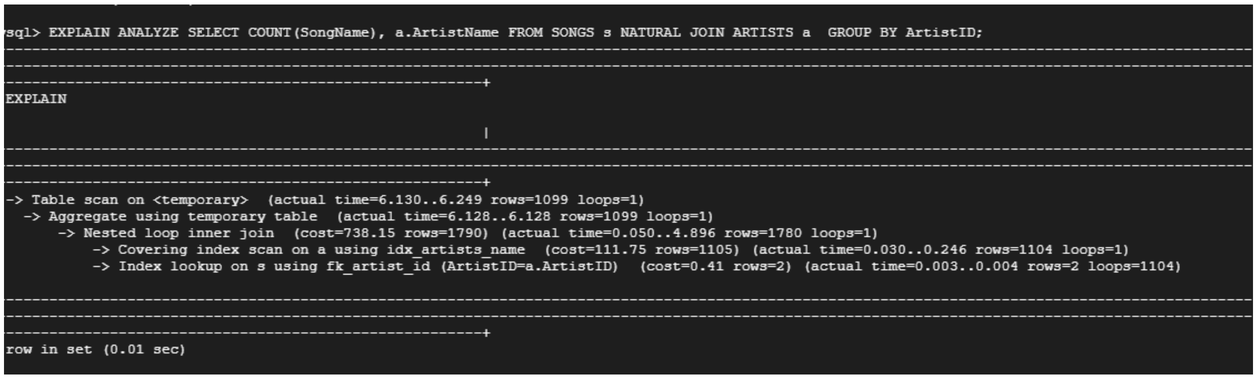
Indexing 1: CREATE INDEX idx_songs_artist ON SONGS (ArtistID);



```
mysql> EXPLAIN ANALYZE SELECT COUNT(SongName), a.ArtistName FROM SONGS s NATURAL JOIN ARTISTS a  GROUP BY ArtistID;
+--------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------
-------------------------+
| EXPLAIN

                         |
+--------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------
-------------------------+
| -> Table scan on <temporary>  (actual time=6.097..6.226 rows=1099 loops=1)
    -> Aggregate using temporary table  (actual time=6.095..6.095 rows=1099 loops=1)
        -> Nested loop inner join  (cost=738.15 rows=1790) (actual time=0.081..4.877 rows=1780 loops=1)
            -> Table scan on a  (cost=111.75 rows=1105) (actual time=0.050..0.297 rows=1104 loops=1)
            -> Index lookup on s using fk_artist_id (ArtistID=a.ArtistID)  (cost=0.41 rows=2) (actual time=0.003..0.004 rows=2 loops=1104)
 |
+--------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------
-------------------------+
1 row in set (0.01 sec)
```

The index `idx_songs_artist` on the `ArtistID` column in the `SONGS` table does not optimize the cost in this query due to the nature of the NATURAL JOIN operation. While indexing `ArtistID` can improve the lookup speed for rows in the `SONGS` table when matched with the `ARTISTS` table, it still performs numerous index lookups for each row in the `ARTISTS` table, which does not optimize the cost.

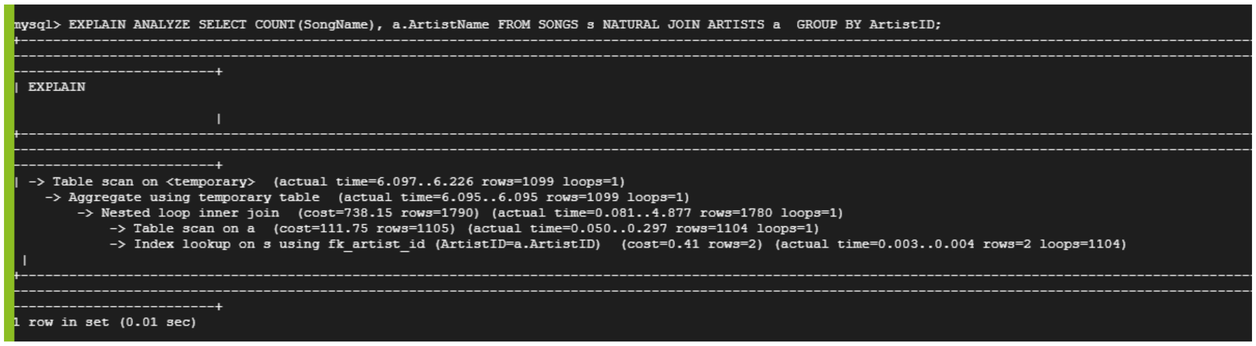
Indexing 2: CREATE INDEX idx_artists_name ON ARTISTS(ArtistName);



```
sql> EXPLAIN ANALYZE SELECT COUNT(SongName), a.ArtistName FROM SONGS s NATURAL JOIN ARTISTS a  GROUP BY ArtistID;
------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------+
EXPLAIN

                                                       |
------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------+
-> Table scan on <temporary>  (actual time=6.130..6.249 rows=1099 loops=1)
  -> Aggregate using temporary table  (actual time=6.128..6.128 rows=1099 loops=1)
      -> Nested loop inner join  (cost=738.15 rows=1790) (actual time=0.050..4.896 rows=1780 loops=1)
          -> Covering index scan on a using idx_artists_name  (cost=111.75 rows=1105) (actual time=0.030..0.246 rows=1104 loops=1)
          -> Index lookup on s using fk_artist_id (ArtistID=a.ArtistID)  (cost=0.41 rows=2) (actual time=0.003..0.004 rows=2 loops=1104)

------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------+
row in set (0.01 sec)
```

The index `idx_artists_name` does not optimize the query because it is not utilized in the join or grouping operations. In this query, the join condition is based solely on the `ArtistID`, which is not affected by the `ArtistName` index. Consequently, the database engine will still need to perform a full scan of the `ARTISTS` table to find matching rows for the join, leading to the cost not changing.

Indexing 3: CREATE INDEX idx_songs_songname ON SONGS (SongName);



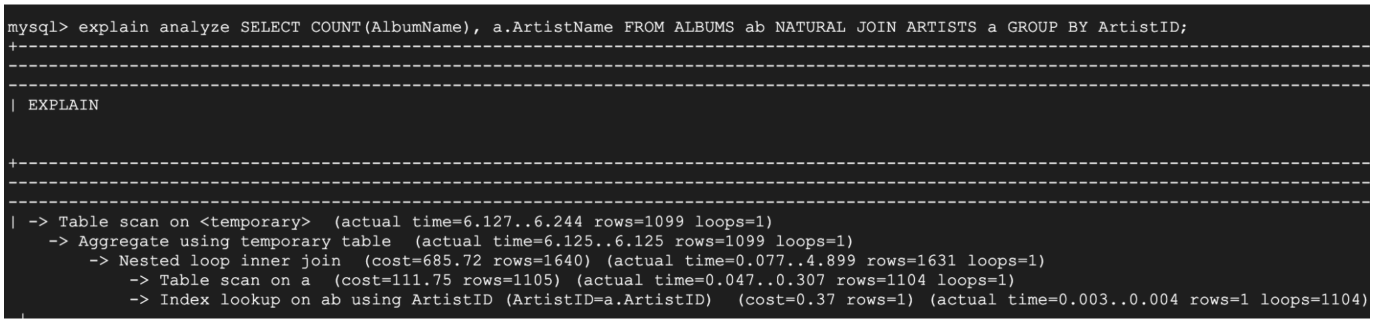
```
mysql> EXPLAIN ANALYZE SELECT COUNT(SongName), a.ArtistName FROM SONGS s NATURAL JOIN ARTISTS a  GROUP BY ArtistID;
+-----------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------+
| EXPLAIN

                         |
+-----------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------+
| -> Table scan on <temporary>  (actual time=6.097..6.226 rows=1099 loops=1)
    -> Aggregate using temporary table  (actual time=6.095..6.095 rows=1099 loops=1)
        -> Nested loop inner join  (cost=738.15 rows=1790) (actual time=0.081..4.877 rows=1780 loops=1)
            -> Table scan on a  (cost=111.75 rows=1105) (actual time=0.050..0.297 rows=1104 loops=1)
            -> Index lookup on s using fk_artist_id (ArtistID=a.ArtistID)  (cost=0.41 rows=2) (actual time=0.003..0.004 rows=2 loops=1104)
 |
+-----------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------+
1 row in set (0.01 sec)
```

Similar to indexing 2, the `SongName` column in the `SONGS` table does not optimize the query because it is not involved in the join or the aggregation process of the query.

- **Album Count per Artist**

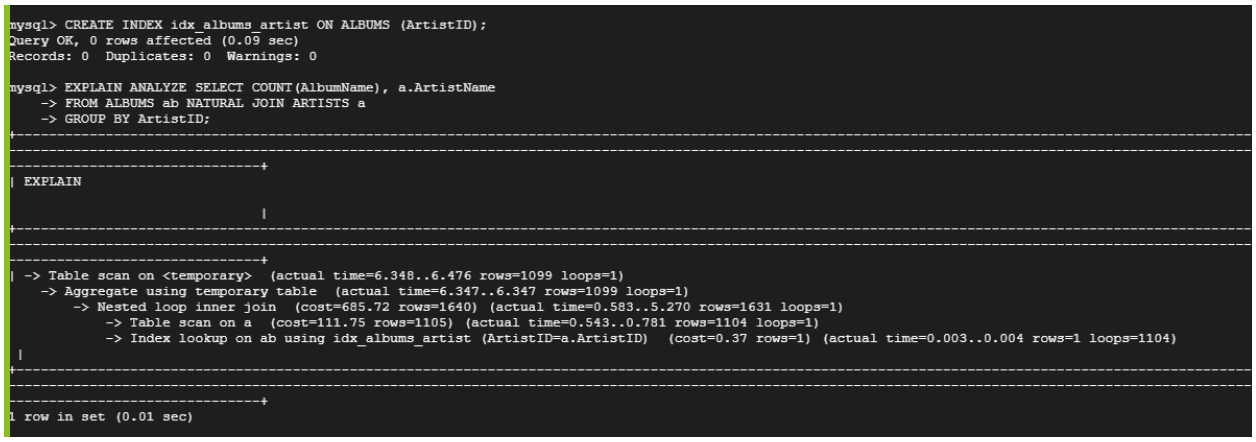
  SELECT COUNT(AlbumName), a.ArtistName
  FROM ALBUMS ab NATURAL JOIN ARTISTS a
  GROUP BY ArtistID;

Default index:



```
mysql> explain analyze SELECT COUNT(AlbumName), a.ArtistName FROM ALBUMS ab NATURAL JOIN ARTISTS a GROUP BY ArtistID;
+-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
| EXPLAIN


+-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
| -> Table scan on <temporary>  (actual time=6.127..6.244 rows=1099 loops=1)
    -> Aggregate using temporary table  (actual time=6.125..6.125 rows=1099 loops=1)
        -> Nested loop inner join  (cost=685.72 rows=1640) (actual time=0.077..4.899 rows=1631 loops=1)
            -> Table scan on a  (cost=111.75 rows=1105) (actual time=0.047..0.307 rows=1104 loops=1)
            -> Index lookup on ab using ArtistID (ArtistID=a.ArtistID)  (cost=0.37 rows=1) (actual time=0.003..0.004 rows=1 loops=1104)
```

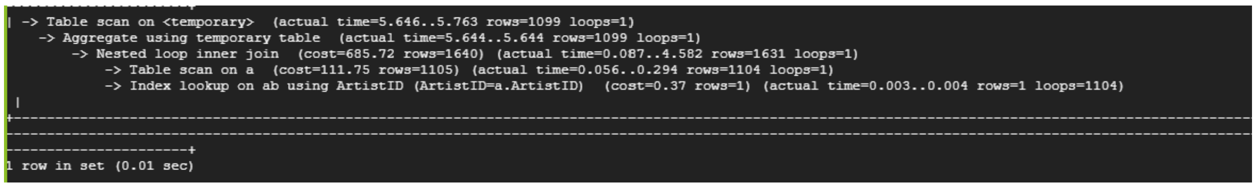
Indexing 1: CREATE INDEX idx_albums_artist ON ALBUMS (ArtistID);



```
mysql> CREATE INDEX idx_albums_artist ON ALBUMS (ArtistID);
Query OK, 0 rows affected (0.09 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE SELECT COUNT(AlbumName), a.ArtistName
    -> FROM ALBUMS ab NATURAL JOIN ARTISTS a
    -> GROUP BY ArtistID;
+-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
-------------------------------+
| EXPLAIN

                               |
+-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
-------------------------------+
| -> Table scan on <temporary>  (actual time=6.348..6.476 rows=1099 loops=1)
    -> Aggregate using temporary table  (actual time=6.347..6.347 rows=1099 loops=1)
        -> Nested loop inner join  (cost=685.72 rows=1640) (actual time=0.583..5.270 rows=1631 loops=1)
            -> Table scan on a  (cost=111.75 rows=1105) (actual time=0.543..0.781 rows=1104 loops=1)
            -> Index lookup on ab using idx_albums_artist (ArtistID=a.ArtistID)  (cost=0.37 rows=1) (actual time=0.003..0.004 rows=1 loops=1104)
 |
+-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
-------------------------------+
1 row in set (0.01 sec)
```

The index `idx_albums_artist` on the `ArtistID` column in the `ALBUMS` table does not optimize the cost in this query due to the nature of the NATURAL JOIN operation. While indexing `ArtistID` can improve the lookup speed for rows in the `ALBUMS` table when matched with the `ARTISTS` table, it still performs numerous index lookups for each row in the `ARTISTS` table, which does not optimize the cost.

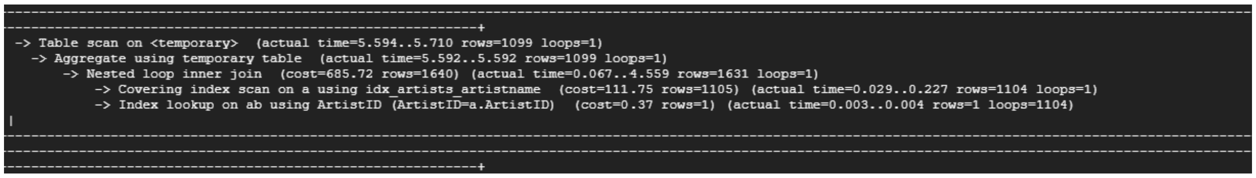
Indexing 2: CREATE INDEX idx_albums_name ON ALBUMS (AlbumName);



```
-----------------------+
| -> Table scan on <temporary>  (actual time=5.646..5.763 rows=1099 loops=1)
    -> Aggregate using temporary table  (actual time=5.644..5.644 rows=1099 loops=1)
        -> Nested loop inner join  (cost=685.72 rows=1640) (actual time=0.087..4.582 rows=1631 loops=1)
            -> Table scan on a  (cost=111.75 rows=1105) (actual time=0.056..0.294 rows=1104 loops=1)
            -> Index lookup on ab using ArtistID (ArtistID=a.ArtistID)  (cost=0.37 rows=1) (actual time=0.003..0.004 rows=1 loops=1104)
 |
+-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
-----------------------+
1 row in set (0.01 sec)
```

The index `idx_albums_name` does not optimize the query because it is not utilized in the join or grouping operations. In this query, the join condition is based solely on the `ArtistID`, which is not affected by the `AlbumName` index. Consequently, the database engine will still need to perform a full scan of the `ARTISTS` table to find matching rows for the join, leading to the cost not changing.

Indexing 3: CREATE INDEX idx_artists_artistname ON ARTISTS (ArtistName);



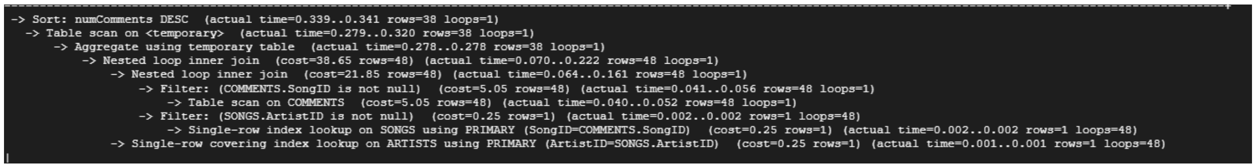
```
--------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------+
 -> Table scan on <temporary>  (actual time=5.594..5.710 rows=1099 loops=1)
   -> Aggregate using temporary table  (actual time=5.592..5.592 rows=1099 loops=1)
       -> Nested loop inner join  (cost=685.72 rows=1640) (actual time=0.067..4.559 rows=1631 loops=1)
           -> Covering index scan on a using idx_artists_artistname  (cost=111.75 rows=1105) (actual time=0.029..0.227 rows=1104 loops=1)
           -> Index lookup on ab using ArtistID (ArtistID=a.ArtistID)  (cost=0.37 rows=1) (actual time=0.003..0.004 rows=1 loops=1104)
|
--------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------+
```

Similar to indexing 2, the `ArtistName` column in the `ARTISTS` table does not optimize the query because it is not involved in the join or the aggregation process of the query.

- **Give User ID, ArtistID, Count of Comments**

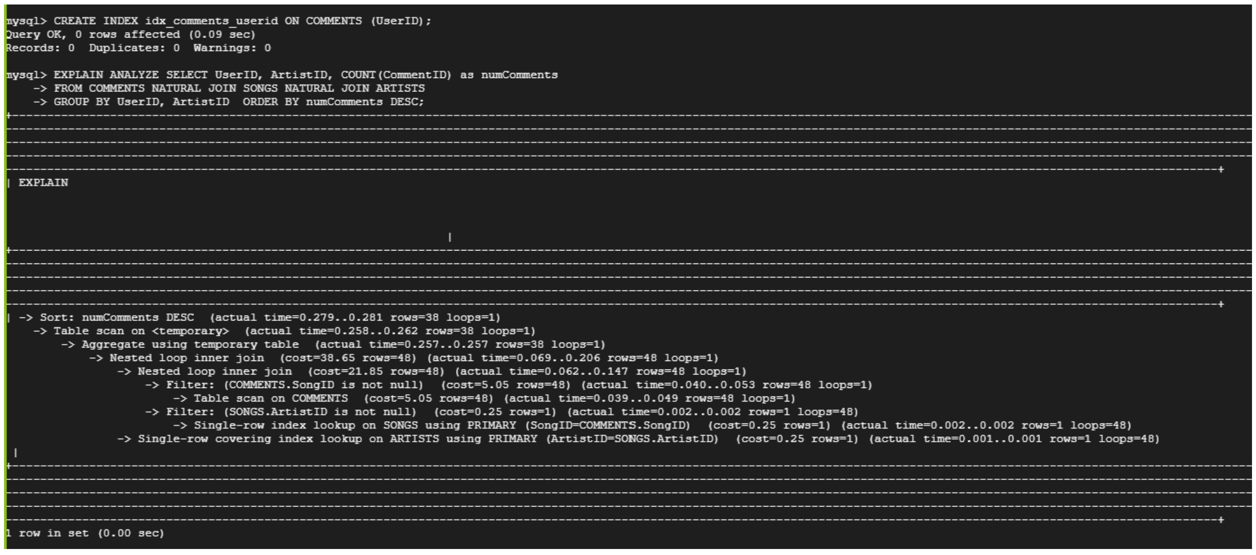
  SELECT UserID, ArtistID, COUNT(CommentID) as numComments
  FROM COMMENTS NATURAL JOIN SONGS NATURAL JOIN ARTISTS
  GROUP BY UserID, ArtistID ORDER BY numComments DESC;

Default Index:



```
----------------------------------------------------------------------------------------------------------------------------------------------+
-> Sort: numComments DESC  (actual time=0.339..0.341 rows=38 loops=1)
  -> Table scan on <temporary>  (actual time=0.279..0.320 rows=38 loops=1)
      -> Aggregate using temporary table  (actual time=0.278..0.278 rows=38 loops=1)
          -> Nested loop inner join  (cost=38.65 rows=48) (actual time=0.070..0.222 rows=48 loops=1)
              -> Nested loop inner join  (cost=21.85 rows=48) (actual time=0.064..0.161 rows=48 loops=1)
                  -> Filter: (COMMENTS.SongID is not null)  (cost=5.05 rows=48) (actual time=0.041..0.056 rows=48 loops=1)
                      -> Table scan on COMMENTS  (cost=5.05 rows=48) (actual time=0.040..0.052 rows=48 loops=1)
                  -> Filter: (SONGS.ArtistID is not null)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=48)
                      -> Single-row index lookup on SONGS using PRIMARY (SongID=COMMENTS.SongID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=48)
              -> Single-row covering index lookup on ARTISTS using PRIMARY (ArtistID=SONGS.ArtistID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=48)
|
```

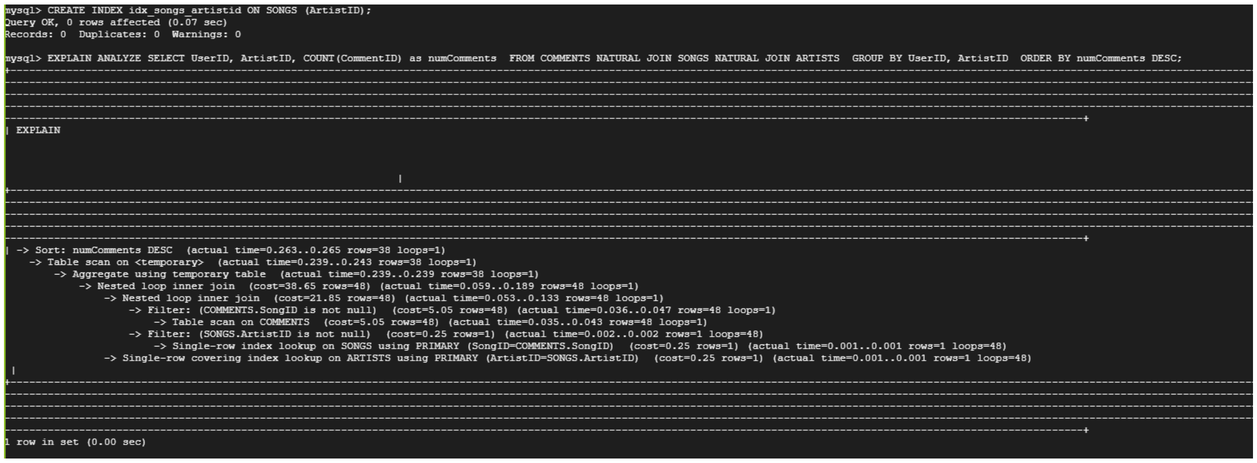
Indexing 1: CREATE INDEX idx_comments_userid ON COMMENTS (UserID);



```
mysql> CREATE INDEX idx_comments_userid ON COMMENTS (UserID);
Query OK, 0 rows affected (0.09 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE SELECT UserID, ArtistID, COUNT(CommentID) as numComments
    -> FROM COMMENTS NATURAL JOIN SONGS NATURAL JOIN ARTISTS
    -> GROUP BY UserID, ArtistID  ORDER BY numComments DESC;
+-------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN



                                                                |
+-------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------+
| -> Sort: numComments DESC  (actual time=0.279..0.281 rows=38 loops=1)
    -> Table scan on <temporary>  (actual time=0.258..0.262 rows=38 loops=1)
        -> Aggregate using temporary table  (actual time=0.257..0.257 rows=38 loops=1)
            -> Nested loop inner join  (cost=38.65 rows=48) (actual time=0.069..0.206 rows=48 loops=1)
                -> Nested loop inner join  (cost=21.85 rows=48) (actual time=0.062..0.147 rows=48 loops=1)
                    -> Filter: (COMMENTS.SongID is not null)  (cost=5.05 rows=48) (actual time=0.040..0.053 rows=48 loops=1)
                        -> Table scan on COMMENTS  (cost=5.05 rows=48) (actual time=0.039..0.049 rows=48 loops=1)
                    -> Filter: (SONGS.ArtistID is not null)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=48)
                        -> Single-row index lookup on SONGS using PRIMARY (SongID=COMMENTS.SongID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=48)
                -> Single-row covering index lookup on ARTISTS using PRIMARY (ArtistID=SONGS.ArtistID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=48)
 |
+-------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

This is mainly due to the NATURAL JOIN nature, similar to previous Indexing 1s for other queries.

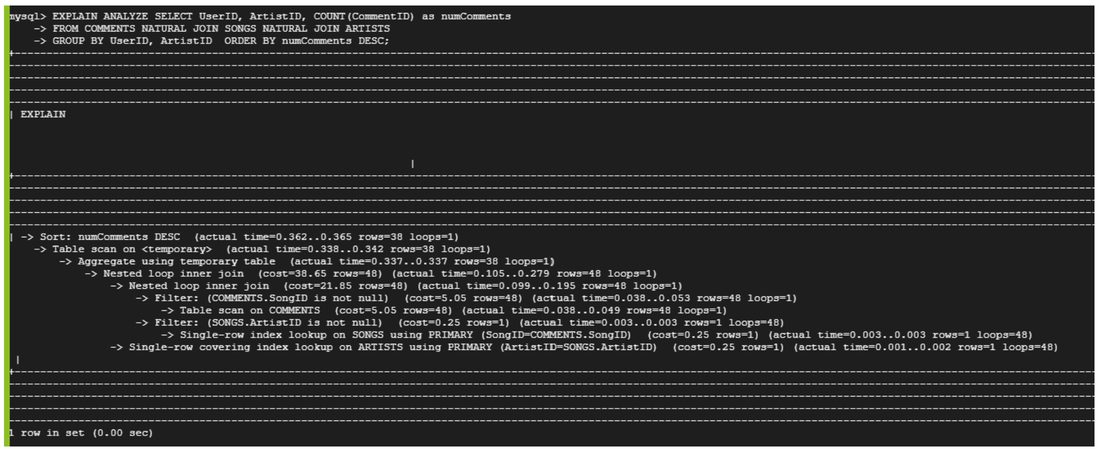
Indexing 2: CREATE INDEX idx_songs_artistid ON SONGS (ArtistID);



```
mysql> CREATE INDEX idx_songs_artistid ON SONGS (ArtistID);
Query OK, 0 rows affected (0.07 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE SELECT UserID, ArtistID, COUNT(CommentID) as numComments  FROM COMMENTS NATURAL JOIN SONGS NATURAL JOIN ARTISTS  GROUP BY UserID, ArtistID  ORDER BY numComments DESC;
+--------------------------------------------------------------------------------------------------------------------------------
| EXPLAIN
                                                       |
+--------------------------------------------------------------------------------------------------------------------------------
| -> Sort: numComments DESC  (actual time=0.263..0.265 rows=38 loops=1)
    -> Table scan on <temporary>  (actual time=0.239..0.243 rows=38 loops=1)
        -> Aggregate using temporary table  (actual time=0.239..0.239 rows=38 loops=1)
            -> Nested loop inner join  (cost=38.65 rows=48) (actual time=0.059..0.189 rows=48 loops=1)
                -> Nested loop inner join  (cost=21.85 rows=48) (actual time=0.053..0.133 rows=48 loops=1)
                    -> Filter: (COMMENTS.SongID is not null)  (cost=5.05 rows=48) (actual time=0.036..0.047 rows=48 loops=1)
                        -> Table scan on COMMENTS  (cost=5.05 rows=48) (actual time=0.035..0.043 rows=48 loops=1)
                    -> Filter: (SONGS.ArtistID is not null)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=48)
                        -> Single-row index lookup on SONGS using PRIMARY (SongID=COMMENTS.SongID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=48)
                -> Single-row covering index lookup on ARTISTS using PRIMARY (ArtistID=SONGS.ArtistID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=48)
 |
+--------------------------------------------------------------------------------------------------------------------------------
1 row in set (0.00 sec)
```

Indexing 3: CREATE INDEX idx_comments_songid ON COMMENTS (SongID);

(We use this because although SongID is not explicit in the query, it is used in the natural join of Song and Comments table)

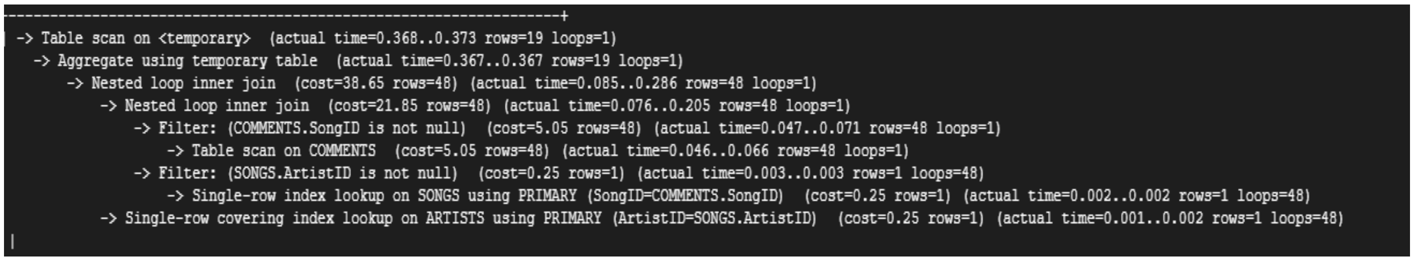


```
mysql> EXPLAIN ANALYZE SELECT UserID, ArtistID, COUNT(CommentID) as numComments
    -> FROM COMMENTS NATURAL JOIN SONGS NATURAL JOIN ARTISTS
    -> GROUP BY UserID, ArtistID  ORDER BY numComments DESC;
+--------------------------------------------------------------------------------------------------------------------------------
| EXPLAIN
                                                       |
+--------------------------------------------------------------------------------------------------------------------------------
| -> Sort: numComments DESC  (actual time=0.362..0.365 rows=38 loops=1)
    -> Table scan on <temporary>  (actual time=0.338..0.342 rows=38 loops=1)
        -> Aggregate using temporary table  (actual time=0.337..0.337 rows=38 loops=1)
            -> Nested loop inner join  (cost=38.65 rows=48) (actual time=0.105..0.279 rows=48 loops=1)
                -> Nested loop inner join  (cost=21.85 rows=48) (actual time=0.099..0.195 rows=48 loops=1)
                    -> Filter: (COMMENTS.SongID is not null)  (cost=5.05 rows=48) (actual time=0.038..0.053 rows=48 loops=1)
                        -> Table scan on COMMENTS  (cost=5.05 rows=48) (actual time=0.038..0.049 rows=48 loops=1)
                    -> Filter: (SONGS.ArtistID is not null)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=48)
                        -> Single-row index lookup on SONGS using PRIMARY (SongID=COMMENTS.SongID)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=48)
                -> Single-row covering index lookup on ARTISTS using PRIMARY (ArtistID=SONGS.ArtistID)  (cost=0.25 rows=1) (actual time=0.001..0.002 rows=1 loops=48)
 |
+--------------------------------------------------------------------------------------------------------------------------------
1 row in set (0.00 sec)
```

Analysis: INNER NESTED JOIN is the costliest here costing 21.85 out of 38.65. To improve the situation, We tried putting Different Indexes based on foreign key values to improve join efficiency however as seen in the images, we notice that no performance increase happens

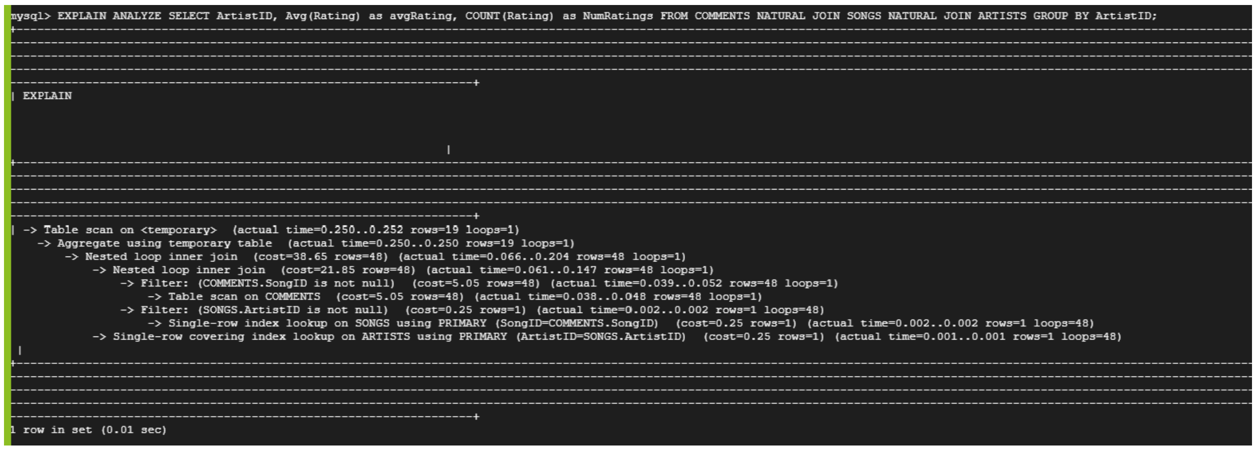
- **Get Avg Ratings of all artists based on their songs**
  SELECT ArtistID, Avg(Rating) as avgRating, COUNT(Rating) as NumRatings FROM COMMENTS NATURAL JOIN SONGS NATURAL JOIN ARTISTS GROUP BY ArtistID;

Default Index:



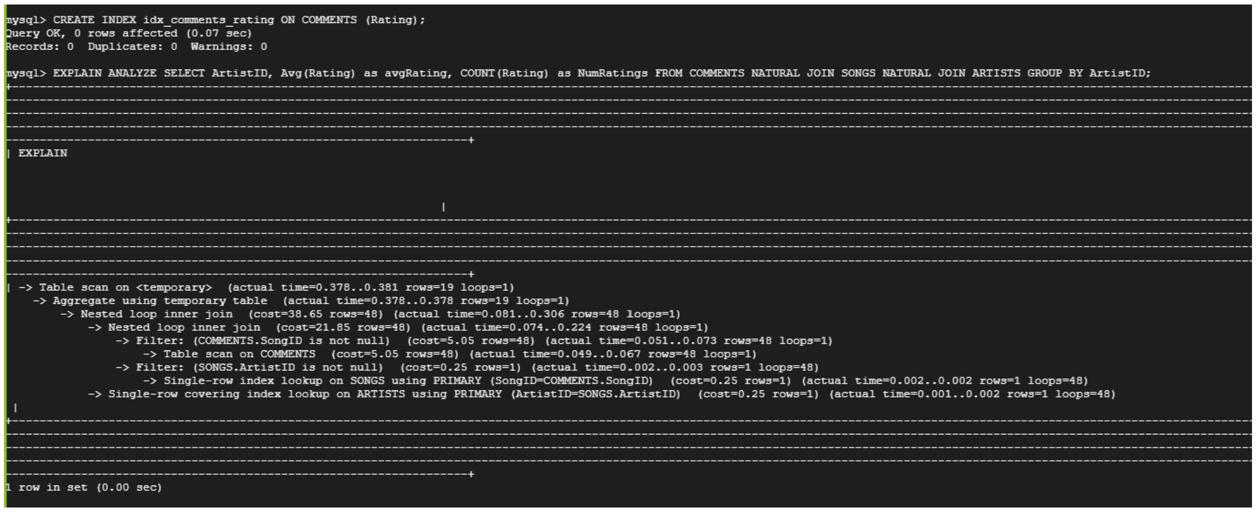
```
------------------------------------------------------------------+
| -> Table scan on <temporary>  (actual time=0.368..0.373 rows=19 loops=1)
    -> Aggregate using temporary table  (actual time=0.367..0.367 rows=19 loops=1)
        -> Nested loop inner join  (cost=38.65 rows=48) (actual time=0.085..0.286 rows=48 loops=1)
            -> Nested loop inner join  (cost=21.85 rows=48) (actual time=0.076..0.205 rows=48 loops=1)
                -> Filter: (COMMENTS.SongID is not null)  (cost=5.05 rows=48) (actual time=0.047..0.071 rows=48 loops=1)
                    -> Table scan on COMMENTS  (cost=5.05 rows=48) (actual time=0.046..0.066 rows=48 loops=1)
                -> Filter: (SONGS.ArtistID is not null)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=48)
                    -> Single-row index lookup on SONGS using PRIMARY (SongID=COMMENTS.SongID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=48)
            -> Single-row covering index lookup on ARTISTS using PRIMARY (ArtistID=SONGS.ArtistID)  (cost=0.25 rows=1) (actual time=0.001..0.002 rows=1 loops=48)
 |
```

Indexing 1: CREATE INDEX idx_songs_artistid ON SONGS (ArtistID);



```
mysql> EXPLAIN ANALYZE SELECT ArtistID, Avg(Rating) as avgRating, COUNT(Rating) as NumRatings FROM COMMENTS NATURAL JOIN SONGS NATURAL JOIN ARTISTS GROUP BY ArtistID;
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------+
| EXPLAIN



                                                                  |
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------+
| -> Table scan on <temporary>  (actual time=0.250..0.252 rows=19 loops=1)
    -> Aggregate using temporary table  (actual time=0.250..0.250 rows=19 loops=1)
        -> Nested loop inner join  (cost=38.65 rows=48) (actual time=0.066..0.204 rows=48 loops=1)
            -> Nested loop inner join  (cost=21.85 rows=48) (actual time=0.061..0.147 rows=48 loops=1)
                -> Filter: (COMMENTS.SongID is not null)  (cost=5.05 rows=48) (actual time=0.039..0.052 rows=48 loops=1)
                    -> Table scan on COMMENTS  (cost=5.05 rows=48) (actual time=0.038..0.048 rows=48 loops=1)
                -> Filter: (SONGS.ArtistID is not null)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=48)
                    -> Single-row index lookup on SONGS using PRIMARY (SongID=COMMENTS.SongID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=48)
            -> Single-row covering index lookup on ARTISTS using PRIMARY (ArtistID=SONGS.ArtistID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=48)
 |
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------+
1 row in set (0.01 sec)
```

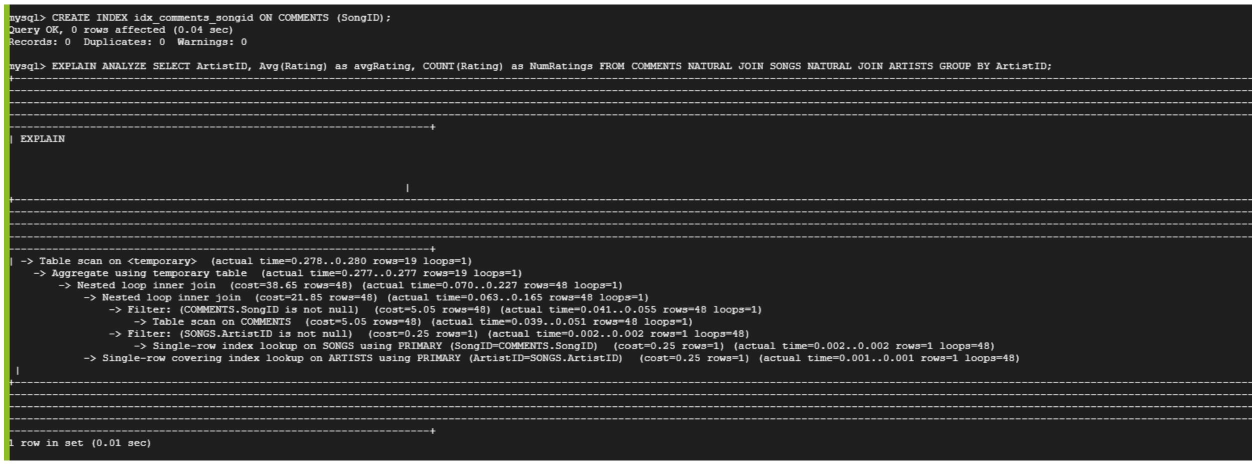
Indexing 2: CREATE INDEX idx_comments_rating ON COMMENTS (Rating);



```
mysql> CREATE INDEX idx_comments_rating ON COMMENTS (Rating);
Query OK, 0 rows affected (0.07 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE SELECT ArtistID, Avg(Rating) as avgRating, COUNT(Rating) as NumRatings FROM COMMENTS NATURAL JOIN SONGS NATURAL JOIN ARTISTS GROUP BY ArtistID;
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------+
| EXPLAIN



                                                                  |
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------+
| -> Table scan on <temporary>  (actual time=0.378..0.381 rows=19 loops=1)
    -> Aggregate using temporary table  (actual time=0.378..0.378 rows=19 loops=1)
        -> Nested loop inner join  (cost=38.65 rows=48) (actual time=0.081..0.306 rows=48 loops=1)
            -> Nested loop inner join  (cost=21.85 rows=48) (actual time=0.074..0.224 rows=48 loops=1)
                -> Filter: (COMMENTS.SongID is not null)  (cost=5.05 rows=48) (actual time=0.051..0.073 rows=48 loops=1)
                    -> Table scan on COMMENTS  (cost=5.05 rows=48) (actual time=0.049..0.067 rows=48 loops=1)
                -> Filter: (SONGS.ArtistID is not null)  (cost=0.25 rows=1) (actual time=0.002..0.003 rows=1 loops=48)
                    -> Single-row index lookup on SONGS using PRIMARY (SongID=COMMENTS.SongID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=48)
            -> Single-row covering index lookup on ARTISTS using PRIMARY (ArtistID=SONGS.ArtistID)  (cost=0.25 rows=1) (actual time=0.001..0.002 rows=1 loops=48)
 |
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------+
1 row in set (0.00 sec)
```

Indexing 3: CREATE INDEX idx_comments_songid ON COMMENTS (SongID);

(We use this because although SongID is not explicit in the query, it is used in the natural join of Song and Comments table)



```
mysql> CREATE INDEX idx_comments_songid ON COMMENTS (SongID);
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE SELECT ArtistID, Avg(Rating) as avgRating, COUNT(Rating) as NumRatings FROM COMMENTS NATURAL JOIN SONGS NATURAL JOIN ARTISTS GROUP BY ArtistID;
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------+
| EXPLAIN



                                                              |
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------+
| -> Table scan on <temporary>  (actual time=0.278..0.280 rows=19 loops=1)
    -> Aggregate using temporary table  (actual time=0.277..0.277 rows=19 loops=1)
        -> Nested loop inner join  (cost=38.65 rows=48) (actual time=0.070..0.227 rows=48 loops=1)
            -> Nested loop inner join  (cost=21.85 rows=48) (actual time=0.063..0.165 rows=48 loops=1)
                -> Filter: (COMMENTS.SongID is not null)  (cost=5.05 rows=48) (actual time=0.041..0.055 rows=48 loops=1)
                    -> Table scan on COMMENTS  (cost=5.05 rows=48) (actual time=0.039..0.051 rows=48 loops=1)
                -> Filter: (SONGS.ArtistID is not null)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=48)
                    -> Single-row index lookup on SONGS using PRIMARY (SongID=COMMENTS.SongID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=48)
            -> Single-row covering index lookup on ARTISTS using PRIMARY (ArtistID=SONGS.ArtistID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=48)
 |
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------+
1 row in set (0.01 sec)
```

Analysis: INNER NESTED JOIN is the costliest here as well costing 21.85 out of 38.65. We tried adding Indexes to SONGS (ArtistID) and COMMENTS (SongID) as they are used to join tables together but that didn't work well as there was no performance boost. We also tried adding indexes to the Rating Column of Table COMMENTS but that doesn't show any improvement either, most probably since that is not used for Joining of tables and even if it made any improvement, it is not as good as default probably because of the small size of this table.