Team 032
Rithik Morusupalli, Aryaman Joshi, Qiyuan Cheng, Daniel Dilan
Stage 3: Database Implementation and Indexing

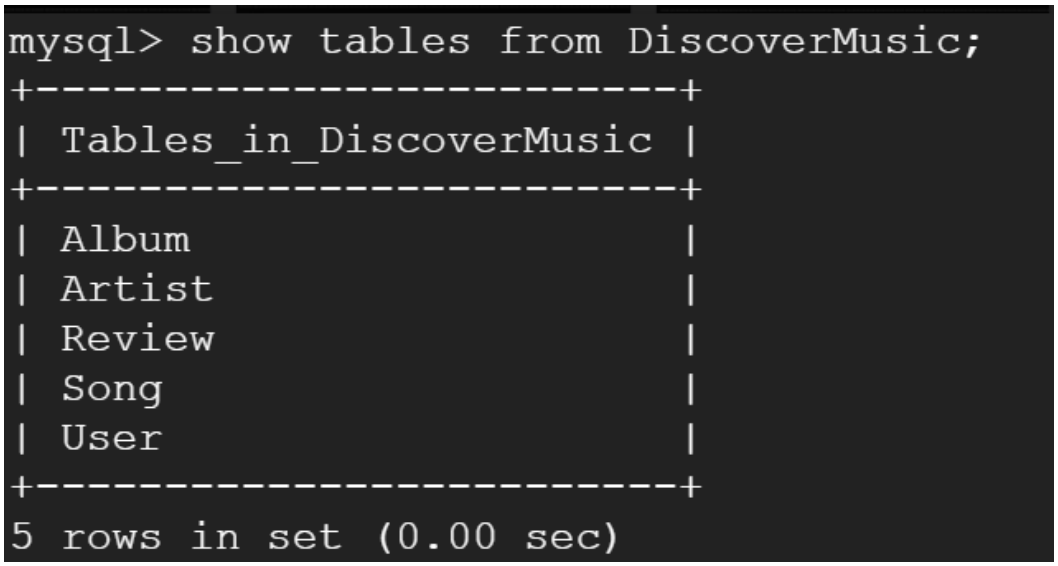
Proof of Connection:

```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to discovermusic-343700.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
gcloud sql connect discovermusic-instance --user=root --quietqi7yuan_cheng@cloudshell:~ (discovermusic-343700)$ gcloud sql connect discovermusic-instance --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 16936
Server version: 8.0.26-google (Google)

Copyright (c) 2000, 2022, 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>
```

Table Data Definition Language Commands:

```
DROP TABLE IF EXISTS `Artist`;
CREATE TABLE `Artist`(
   `artistID` VARCHAR(50) NOT NULL,
   `artistName` VARCHAR(100) DEFAULT NULL,
   PRIMARY KEY(`artistID`)
);
INSERT INTO `Artist`(`artistID`,`artistName`) VALUES

DROP TABLE IF EXISTS `Album`;
CREATE TABLE `Album`(
   `albumID` VARCHAR(50) NOT NULL,
   `albumName` VARCHAR(500) DEFAULT NULL,
   `artistID` VARCHAR(50) NOT NULL,
   `likeCounter` INT,
   PRIMARY KEY(`albumID`),
   FOREIGN KEY (`artistID`) REFERENCES `Artist` (`artistID`)
);
INSERT INTO `Album`(`albumID`,`albumName`,`artistID`,`likeCounter`) VALUES

DROP TABLE IF EXISTS `Song`;
CREATE TABLE `Song` (
   `songID` VARCHAR(50)  NOT NULL,
   `artistID` VARCHAR(50) NOT NULL,
   `songName` VARCHAR(500),
```

```
   `likeCounter` INT,
   `albumID` VARCHAR(30) NOT NULL,
   PRIMARY KEY(`songID`),
   FOREIGN KEY (`artistID`) REFERENCES `Artist` (`artistID`),
   FOREIGN KEY (`albumID`) REFERENCES `Album` (`albumID`)
);
INSERT INTO `Song`(`songID`,`artistID`,`songName`,`likeCounter`,`albumID`) VALUES

DROP TABLE IF EXISTS `User`;
CREATE TABLE `User` (
   `userID` INT(100) NOT NULL,
   `userName` VARCHAR(50),
   `userEmail` VARCHAR(50),
   PRIMARY KEY (`userID`)
);
INSERT INTO `User`(`userID` , `userName`, `userEmail`) VALUES

ROP TABLE IF EXISTS `Review`;
CREATE TABLE `Review`(
  `firstReviewID` VARCHAR(30) NOT NULL,
  `songID` VARCHAR(50),
  `userID` INT(100),
  `reviewTime` DATETIME NOT NULL,
  `reviewContent` VARCHAR (1000),
  PRIMARY KEY(`firstReviewID`),
  FOREIGN KEY (`songID`) REFERENCES `Song` (`songID`),
  FOREIGN KEY (`userID`) REFERENCES `User` (`userID`)
);
INSERT INTO `Review`(`firstReviewID`, `songID`, `userID`,
`reviewContent`,`reviewTime`) VALUES
```



```
mysql> show tables from DiscoverMusic;
+-------------------------+
| Tables_in_DiscoverMusic |
+-------------------------+
| Album                   |
| Artist                  |
| Review                  |
| Song                    |
| User                    |
+-------------------------+
5 rows in set (0.00 sec)
```

Row Count for Tables:

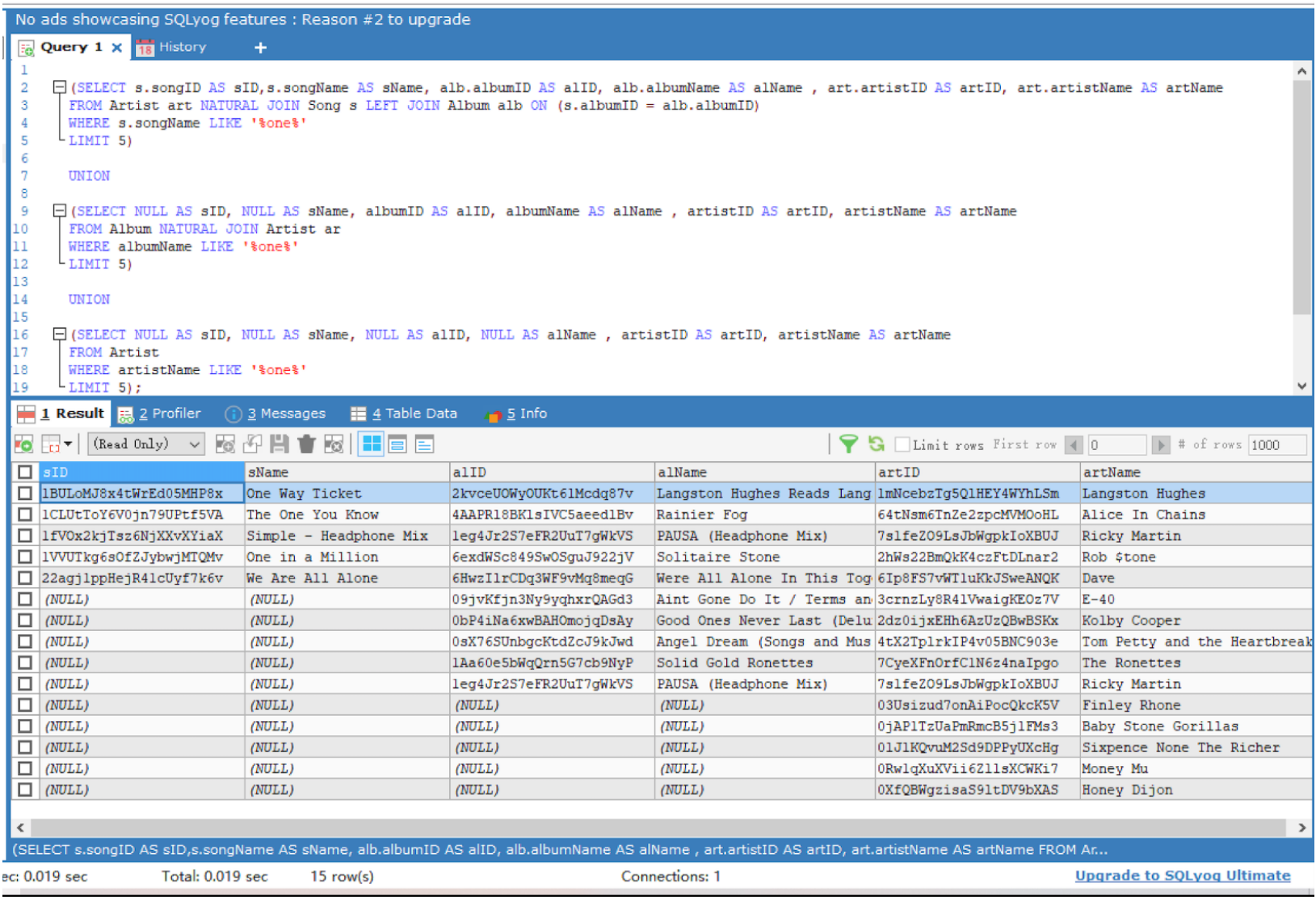
```
mysql> SELECT COUNT(*) FROM Album;
+----------+
| COUNT(*) |
+----------+
|     1407 |
+----------+
1 row in set (0.02 sec)

mysql> SELECT COUNT(*) FROM Artist;
+----------+
| COUNT(*) |
+----------+
|     1925 |
+----------+
1 row in set (0.01 sec)

mysql> SELECT COUNT(*) FROM Song;
+----------+
| COUNT(*) |
+----------+
|     1407 |
+----------+
1 row in set (0.02 sec)
```

Two Advanced SQL Queries and Their Returns:

Query 1 - We show the top 15 artists who have the most amount of likes ranked in a descending order.

```
mysql> SELECT art.artistName, SUM(s.likeCounter) AS sumLike
    -> FROM Song s NATURAL JOIN Artist art
    -> GROUP BY artistID
    -> ORDER BY sumLike DESC
    -> LIMIT 15;
+--------------------------+---------+
| artistName               | sumLike |
+--------------------------+---------+
| Dee Watkins              |    9991 |
| Katrina & The Waves      |    9976 |
| Preme                    |    9973 |
| Muriël Bostdorp          |    9969 |
| Becky G                  |    9960 |
| The Young Rascals        |    9954 |
| Mickey Guyton            |    9953 |
| Yung L                   |    9949 |
| Sixpence None The Richer |    9942 |
| The LOX                  |    9933 |
| Mötley Crüe              |    9926 |
| The Seekers              |    9917 |
| BETWEEN FRIENDS          |    9914 |
| Mötley Crüe              |    9926 |
| The Seekers              |    9917 |
| BETWEEN FRIENDS          |    9914 |
| TOBi                     |    9914 |
| Mitchell Tenpenny        |    9912 |
+--------------------------+---------+
15 rows in set (0.01 sec)
```

Query 2 - When the user searches by inputting a string, in this case ‘one’, we will return the relevant songs, albums, and artists along with relevant accompanying information.



Team 032
Rithik Morusupalli, Aryaman Joshi, Qiyuan Cheng, Daniel Dilan
Stage 3: Database Implementation and Indexing

Indexing Analysis:

Query 1

```
mysql> EXPLAIN ANALYZE SELECT art.artistName, SUM(s.likeCounter) AS sumLike
    -> FROM Song s NATURAL JOIN Artist art
    -> GROUP BY artistID
    -> ORDER BY sumLike DESC
    -> LIMIT 15;
+---------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------
-----+
| EXPLAIN                              |
+---------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------
-----+
| -> Limit: 15 row(s)  (actual time=4.789..4.791 rows=15 loops=1)
    -> Sort: sumLike DESC, limit input to 15 row(s) per chunk  (actual time=4.789..4.790 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=0.002..0.100 rows=1407 loops=1)
            -> Aggregate using temporary table  (actual time=4.316..4.500 rows=1407 loops=1)
                -> Nested loop inner join  (cost=629.40 rows=1387) (actual time=0.086..2.580 rows=1407 loops=1)
                    -> Table scan on s  (cost=143.95 rows=1387) (actual time=0.044..0.566 rows=1407loops=1)
                    -> Single-row index lookup on art using PRIMARY (artistID=s.artistID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1407)
 |
+---------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------
-----+
1 row in set (0.01 sec)
```

1. CREATE INDEX songCounter on Song(likeCounter)

```
mysql> DROP INDEX songCounter ON Song
    -> ;
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> CREATE INDEX songCounter ON Song(likeCounter);
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE SELECT art.artistName, SUM(s.likeCounter) AS sumLike
    -> FROM Song s NATURAL JOIN Artist art
    -> GROUP BY artistID
    -> ORDER BY sumLike DESC
    -> LIMIT 15;
+---------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------
-----+
| EXPLAIN        |
+---------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------
-----+
| -> Limit: 15 row(s)  (actual time=4.826..4.828 rows=15 loops=1)
    -> Sort: sumLike DESC, limit input to 15 row(s) per chunk  (actual time=4.825..4.826 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=0.001..0.098 rows=1407 loops=1)
            -> Aggregate using temporary table  (actual time=4.358..4.541 rows=1407 loops=1)
                -> Nested loop inner join  (cost=629.40 rows=1387) (actual time=0.082..2.628 rows=1407 loops=1)
                    -> Table scan on s  (cost=143.95 rows=1387) (actual time=0.066..0.528 rows=1407 loops=1)
                    -> Single-row index lookup on art using PRIMARY (artistID=s.artistID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1407)
 |
+---------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------
-----+
1 row in set (0.01 sec)
```

We chose to create an index on the like counter of the Song table because we sum them eventually so we thought it may be efficient to have index entries on them. This was not advantageous, and it was even a little slower than the original query. Despite following the same steps, each step takes slightly longer than the original query. We think that the change in indexing did not bring a better effect because despite the end result being the sum of likeCounter, the indexing didn't account for the preliminary steps and made them slower.

2. CREATE INDEX artArtID on Artist(artistID)

```
mysql> CREATE INDEX artArtID ON Artist(artistID);
Query OK, 0 rows affected (0.07 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE SELECT art.artistName, SUM(s.likeCounter) AS sumLike
    -> FROM Song s NATURAL JOIN Artist art
    -> GROUP BY artistID
    -> ORDER BY sumLike DESC
    -> LIMIT 15;
+--------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------
-----+
| EXPLAIN



     |
+--------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------
-----+
| -> Limit: 15 row(s)  (actual time=4.827..4.829 rows=15 loops=1)
    -> Sort: sumLike DESC, limit input to 15 row(s) per chunk  (actual time=4.826..4.827 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=0.001..0.096 rows=1407 loops=1)
            -> Aggregate using temporary table  (actual time=4.372..4.549 rows=1407 loops=1)
                -> Nested loop inner join  (cost=629.40 rows=1387) (actual time=0.106..2.656 rows=1407 loops=1)
                    -> Table scan on s  (cost=143.95 rows=1387) (actual time=0.092..0.585 rows=1407 loops=1)
                    -> Single-row index lookup on art using PRIMARY (artistID=s.artistID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1407)
 |
+--------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------
-----+
1 row in set (0.01 sec)
```

Secondly, we chose to create an index on the artistId. We chose this because we were aggregating by artistID's. Thus we thought this could improve query speed by indexing based on each artist. However, this was also slower than the original query. This was probably not effective because the indexes were not actually helpful to index on since there was no range condition so the index was only adding to the costs of the queries and wasting space and time.

3. CREATE INDEX songSongID on Song(songID)

```
mysql> CREATE INDEX songSongID ON Song(songID)
    -> ;
Query OK, 0 rows affected (0.06 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE SELECT art.artistName, SUM(s.likeCounter) AS sumLike
    -> FROM Song s NATURAL JOIN Artist art
    -> GROUP BY artistID
    -> ORDER BY sumLike DESC
    -> LIMIT 15;
+------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
-----+
| EXPLAIN



     |
+------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
-----+
| -> Limit: 15 row(s)  (actual time=5.027..5.029 rows=15 loops=1)
    -> Sort: sumLike DESC, limit input to 15 row(s) per chunk  (actual time=5.026..5.028 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=0.002..0.098 rows=1407 loops=1)
            -> Aggregate using temporary table  (actual time=4.553..4.745 rows=1407 loops=1)
                -> Nested loop inner join  (cost=629.40 rows=1387) (actual time=0.069..2.771 rows=1407 loops=1)
                    -> Table scan on s  (cost=143.95 rows=1387) (actual time=0.052..0.621 rows=1407 loops=1)
                    -> Single-row index lookup on art using PRIMARY (artistID=s.artistID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1407)
 |
+------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
-----+
1 row in set (0.01 sec)
```

Finally, we chose to index by songID because the second step is a table scan, and we hypothesized it would make this step faster. The table scan on s was faster, but the following steps were made slower. Similar to before, despite slight improvement on one step of the query, the index caused the rest to be slower, making the whole query slower than it was initially. None of our indices were faster for this query, this leads us to believe that it is unnecessary to have indices for this query as it is simpler and more efficient without. It is important to note queries that do not need indices as one must find the right balance to achieve optimal results.

Query 2

```
mysql> EXPLAIN ANALYZE(
    -> (SELECT s.songID AS sID,s.songName AS sName, alb.albumID AS alID, alb.albumName AS alName , art.artistID AS artID, art.artistName AS artName
    -> FROM Artist art NATURAL JOIN Song s LEFT JOIN Album alb ON (s.albumID = alb.albumID)
    -> WHERE s.songName LIKE '%one%'
    -> LIMIT 5)
    ->
    -> UNION
    ->
    -> (SELECT NULL AS sID, NULL AS sName, albumID AS alID, albumName AS alName , artistID AS artID, artistName AS artName
    -> FROM Album NATURAL JOIN Artist ar
    -> WHERE albumName LIKE '%one%'
    -> LIMIT 5)
    ->
    -> UNION
    ->
    -> (SELECT NULL AS sID, NULL AS sName, NULL AS alID, NULL AS alName , artistID AS artID, artistName AS artName
    -> FROM Artist
    -> WHERE artistName LIKE '%one%'
    -> LIMIT 5));
+--------------------------------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN
                                                                                                                          |
+--------------------------------------------------------------------------------------------------------------------------------------------------+
```

```
+--------------------------------------------------------------------------------------------------------------------------------------------------+
| -> Table scan on <union temporary>  (cost=0.18..2.69 rows=15) (actual time=0.002..0.004 rows=15 loops=1)
    -> Union materialize with deduplication  (cost=647.16..649.67 rows=15) (actual time=0.759..0.762 rows=15 loops=1)
        -> Limit: 5 row(s)  (cost=251.82 rows=5) (actual time=0.200..0.366 rows=5 loops=1)
            -> Nested loop inner join  (cost=251.82 rows=154) (actual time=0.199..0.364 rows=5 loops=1)
                -> Nested loop left join  (cost=197.88 rows=154) (actual time=0.192..0.345 rows=5 loops=1)
                    -> Filter: (s.songName like '%one%')  (cost=143.95 rows=154) (actual time=0.174..0.314 rows=5 loops=1)
                        -> Table scan on s  (cost=143.95 rows=1387) (actual time=0.056..0.175 rows=369 loops=1)
                    -> Filter: (s.albumID = alb.albumID)  (cost=0.25 rows=1) (actual time=0.006..0.006 rows=1 loops=5)
                        -> Single-row index lookup on alb using PRIMARY (albumID=s.albumID)  (cost=0.25 rows=1) (actual time=0.005..0.005 rows=1 loops=5)
                -> Single-row index lookup on art using PRIMARY (artistID=s.artistID)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=5)
        -> Limit: 5 row(s)  (cost=198.41 rows=5) (actual time=0.086..0.226 rows=5 loops=1)
            -> Nested loop inner join  (cost=198.41 rows=156) (actual time=0.086..0.225 rows=5 loops=1)
                -> Filter: (Album.albumName like '%one%')  (cost=143.70 rows=156) (actual time=0.081..0.212 rows=5 loops=1)
                    -> Table scan on Album  (cost=143.70 rows=1407) (actual time=0.067..0.135 rows=262 loops=1)
                -> Single-row index lookup on ar using PRIMARY (artistID=Album.artistID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=5)
        -> Limit: 5 row(s)  (cost=195.25 rows=5) (actual time=0.035..0.127 rows=5 loops=1)
            -> Filter: (Artist.artistName like '%one%')  (cost=195.25 rows=214) (actual time=0.035..0.126 rows=5 loops=1)
                -> Table scan on Artist  (cost=195.25 rows=1925) (actual time=0.030..0.080 rows=232 loops=1)
 |
+--------------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

1. CREATE INDEX artistIndex on Artist(artistID)

```
mysql> DROP INDEX artistIndex on Artist;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> CREATE INDEX albumIndex on Album(albumID)
    -> ;
Query OK, 0 rows affected (0.08 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE(
    -> (SELECT s.songID AS sID,s.songName AS sName, alb.albumID AS alID, alb.albumName AS alName , art.artistID AS artID, art.artistName AS artName
    -> FROM Artist art NATURAL JOIN Song s LEFT JOIN Album alb ON (s.albumID = alb.albumID)
    -> WHERE s.songName LIKE '%one%'
    -> LIMIT 5)
    ->
    -> UNION
    ->
    -> (SELECT NULL AS sID, NULL AS sName, albumID AS alID, albumName AS alName , artistID AS artID, artistName AS artName
    -> FROM Album NATURAL JOIN Artist ar
    -> WHERE albumName LIKE '%one%'
    -> LIMIT 5)
    ->
    -> UNION
    ->
    -> (SELECT NULL AS sID, NULL AS sName, NULL AS alID, NULL AS alName , artistID AS artID, artistName AS artName
    -> FROM Artist
    -> WHERE artistName LIKE '%one%'
    -> LIMIT 5));
+------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN
```

```
------------------------------------------------------------------------------------------------------------------------+
| -> Table scan on <union temporary>  (cost=0.18..2.69 rows=15) (actual time=0.001..0.004 rows=15 loops=1)
    -> Union materialize with deduplication  (cost=647.16..649.67 rows=15) (actual time=0.732..0.735 rows=15 loops=1)
        -> Limit: 5 row(s)  (cost=251.82 rows=5) (actual time=0.237..0.372 rows=5 loops=1)
            -> Nested loop inner join  (cost=251.82 rows=154) (actual time=0.236..0.370 rows=5 loops=1)
                -> Nested loop left join  (cost=197.88 rows=154) (actual time=0.228..0.350 rows=5 loops=1)
                    -> Filter: (s.songName like '%one%')  (cost=143.95 rows=154) (actual time=0.213..0.321 rows=5 loops=1)
                        -> Table scan on s  (cost=143.95 rows=1387) (actual time=0.090..0.213 rows=369 loops=1)
                    -> Filter: (s.albumID = alb.albumID)  (cost=0.25 rows=1) (actual time=0.005..0.005 rows=1 loops=5)
                        -> Single-row index lookup on alb using PRIMARY (albumID=s.albumID)  (cost=0.25 rows=1) (actual time=0.005..0.005 rows=1 loops=5)
                -> Single-row index lookup on art using PRIMARY (artistID=s.artistID)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=5)
        -> Limit: 5 row(s)  (cost=198.41 rows=5) (actual time=0.044..0.191 rows=5 loops=1)
            -> Nested loop inner join  (cost=198.41 rows=156) (actual time=0.043..0.191 rows=5 loops=1)
                -> Filter: (Album.albumName like '%one%')  (cost=143.70 rows=156) (actual time=0.039..0.177 rows=5 loops=1)
                    -> Table scan on Album  (cost=143.70 rows=1407) (actual time=0.026..0.100 rows=262 loops=1)
                -> Single-row index lookup on ar using PRIMARY (artistID=Album.artistID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=5)
        -> Limit: 5 row(s)  (cost=195.25 rows=5) (actual time=0.037..0.128 rows=5 loops=1)
            -> Filter: (Artist.artistName like '%one%')  (cost=195.25 rows=214) (actual time=0.037..0.127 rows=5 loops=1)
                -> Table scan on Artist  (cost=195.25 rows=1925) (actual time=0.032..0.081 rows=232 loops=1)
 |
+------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

Because our query had three subqueries, each index we picked was selected to optimize one of the subqueries. This query was designed to maximize the third subquery. This worked, as each step in the third sub showed slight improvement. In fact the second subquery showed even better improvement, likely due to the natural join. The first subquery was faster and slower on different steps, as some steps included artists and some didn't, but in total was almost the same in speed.

2. CREATE INDEX albumIndex on Album(albumID)

```
mysql> DROP INDEX artistIndex on Artist;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> CREATE INDEX albumIndex on Album(albumID)
    -> ;
Query OK, 0 rows affected (0.08 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE(
    -> (SELECT s.songID AS sID,s.songName AS sName, alb.albumID AS alID, alb.albumName AS alName , art.artistID AS artID, art.artistName AS artName
    -> FROM Artist art NATURAL JOIN Song s LEFT JOIN Album alb ON (s.albumID = alb.albumID)
    -> WHERE s.songName LIKE '%one%'
    -> LIMIT 5)
    ->
    -> UNION
    ->
    -> (SELECT NULL AS sID, NULL AS sName, albumID AS alID, albumName AS alName , artistID AS artID, artistName AS artName
    -> FROM Album NATURAL JOIN Artist ar
    -> WHERE albumName LIKE '%one%'
    -> LIMIT 5)
    ->
    -> UNION
    ->
    -> (SELECT NULL AS sID, NULL AS sName, NULL AS alID, NULL AS alName , artistID AS artID, artistName AS artName
    -> FROM Artist
    -> WHERE artistName LIKE '%one%'
    -> LIMIT 5));
+----------------------------------------------------------------------------------------------------------+
| EXPLAIN
```

```
+----------------------------------------------------------------------------------------------------------+
| -> Table scan on <union temporary>  (cost=0.18..2.69 rows=15) (actual time=0.001..0.004 rows=15 loops=1)
    -> Union materialize with deduplication  (cost=647.16..649.67 rows=15) (actual time=0.732..0.735 rows=15 loops=1)
        -> Limit: 5 row(s)  (cost=251.82 rows=5) (actual time=0.237..0.372 rows=5 loops=1)
            -> Nested loop inner join  (cost=251.82 rows=154) (actual time=0.236..0.370 rows=5 loops=1)
                -> Nested loop left join  (cost=197.88 rows=154) (actual time=0.228..0.350 rows=5 loops=1)
                    -> Filter: (s.songName like '%one%')  (cost=143.95 rows=154) (actual time=0.213..0.321 rows=5 loops=1)
                        -> Table scan on s  (cost=143.95 rows=1387) (actual time=0.090..0.213 rows=369 loops=1)
                    -> Filter: (s.albumID = alb.albumID)  (cost=0.25 rows=1) (actual time=0.005..0.005 rows=1 loops=5)
                        -> Single-row index lookup on alb using PRIMARY (albumID=s.albumID)  (cost=0.25 rows=1) (actual time=0.005..0.005 rows=1 loops=5)
                -> Single-row index lookup on art using PRIMARY (artistID=s.artistID)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=5)
        -> Limit: 5 row(s)  (cost=198.41 rows=5) (actual time=0.044..0.191 rows=5 loops=1)
            -> Nested loop inner join  (cost=198.41 rows=156) (actual time=0.043..0.191 rows=5 loops=1)
                -> Filter: (Album.albumName like '%one%')  (cost=143.70 rows=156) (actual time=0.039..0.177 rows=5 loops=1)
                    -> Table scan on Album  (cost=143.70 rows=1407) (actual time=0.026..0.100 rows=262 loops=1)
                -> Single-row index lookup on ar using PRIMARY (artistID=Album.artistID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=5)
        -> Limit: 5 row(s)  (cost=195.25 rows=5) (actual time=0.037..0.128 rows=5 loops=1)
            -> Filter: (Artist.artistName like '%one%')  (cost=195.25 rows=214) (actual time=0.037..0.127 rows=5 loops=1)
                -> Table scan on Artist  (cost=195.25 rows=1925) (actual time=0.032..0.081 rows=232 loops=1)
 |
+----------------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

Next we chose to create indexes on the albumID's. We thought this was efficient to index on because it was an equality condition for the query. This means it allowed the database to filter and go through a small number of rows to return when that subquery was

running. This optimized the query as it ran faster than the original query, from .759 .. .762 to .732 .. .735.

3. CREATE INDEX songIndex on Song(songID)

```
mysql> DROP INDEX albumIndex on Album;
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> CREATE INDEX songIndex on Song(songID);
Query OK, 0 rows affected, 1 warning (0.08 sec)
Records: 0  Duplicates: 0  Warnings: 1

mysql> EXPLAIN ANALYZE(
    -> (SELECT s.songID AS sID,s.songName AS sName, alb.albumID AS alID, alb.albumName AS alName , art.artistID AS artID, art.artistName AS artName
    -> FROM Artist art NATURAL JOIN Song s LEFT JOIN Album alb ON (s.albumID = alb.albumID)
    -> WHERE s.songName LIKE '%one%'
    -> LIMIT 5)
    ->
    -> UNION
    ->
    -> (SELECT NULL AS sID, NULL AS sName, albumID AS alID, albumName AS alName , artistID AS artID, artistName AS artName
    -> FROM Album NATURAL JOIN Artist ar
    -> WHERE albumName LIKE '%one%'
    -> LIMIT 5)
    ->
    -> UNION
    ->
    -> (SELECT NULL AS sID, NULL AS sName, NULL AS alID, NULL AS alName , artistID AS artID, artistName AS artName
    -> FROM Artist
    -> WHERE artistName LIKE '%one%'
    -> LIMIT 5));
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN
```

```
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| -> Table scan on <union temporary>  (cost=0.18..2.69 rows=15) (actual time=0.002..0.004 rows=15 loops=1)
    -> Union materialize with deduplication  (cost=647.16..649.67 rows=15) (actual time=0.700..0.703 rows=15 loops=1)
        -> Limit: 5 row(s)  (cost=251.82 rows=5) (actual time=0.202..0.338 rows=5 loops=1)
            -> Nested loop inner join  (cost=251.82 rows=154) (actual time=0.201..0.336 rows=5 loops=1)
                -> Nested loop left join  (cost=197.88 rows=154) (actual time=0.194..0.316 rows=5 loops=1)
                    -> Filter: (s.songName like '%one%')  (cost=143.95 rows=154) (actual time=0.175..0.282 rows=5 loops=1)
                        -> Table scan on s  (cost=143.95 rows=1387) (actual time=0.051..0.178 rows=369 loops=1)
                    -> Filter: (s.albumID = alb.albumID)  (cost=0.25 rows=1) (actual time=0.006..0.006 rows=1 loops=5)
                        -> Single-row index lookup on alb using PRIMARY (albumID=s.albumID)  (cost=0.25 rows=1) (actual time=0.006..0.006 rows=1 loops=5)
                -> Single-row index lookup on art using PRIMARY (artistID=s.artistID)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=5)
        -> Limit: 5 row(s)  (cost=198.41 rows=5) (actual time=0.047..0.187 rows=5 loops=1)
            -> Nested loop inner join  (cost=198.41 rows=156) (actual time=0.047..0.187 rows=5 loops=1)
                -> Filter: (Album.albumName like '%one%')  (cost=143.70 rows=156) (actual time=0.042..0.173 rows=5 loops=1)
                    -> Table scan on Album  (cost=143.70 rows=1407) (actual time=0.028..0.095 rows=262 loops=1)
                -> Single-row index lookup on ar using PRIMARY (artistID=Album.artistID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=5)
        -> Limit: 5 row(s)  (cost=195.25 rows=5) (actual time=0.036..0.132 rows=5 loops=1)
            -> Filter: (Artist.artistName like '%one%')  (cost=195.25 rows=214) (actual time=0.036..0.131 rows=5 loops=1)
                -> Table scan on Artist  (cost=195.25 rows=1925) (actual time=0.031..0.082 rows=232 loops=1)
 |
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

Finally, we chose to create indexes on the songID. This was the last subquery we have chosen to focus on. This was not only faster than the original query, but the fastest out of

all the indexes we created as actual time around .700 .. .703. By adding an index on the songID, it was able to quickly get the rows of the table for the most complicated query, which may explain why it reduced the overall time the most. Since all of these indexes were faster, we may try to implement a combination of these indexes so that we have multiple optimized subqueries and increased performance overall. If we were to move forward using only one, it would be the songIndex since it had the largest improvement in performance.