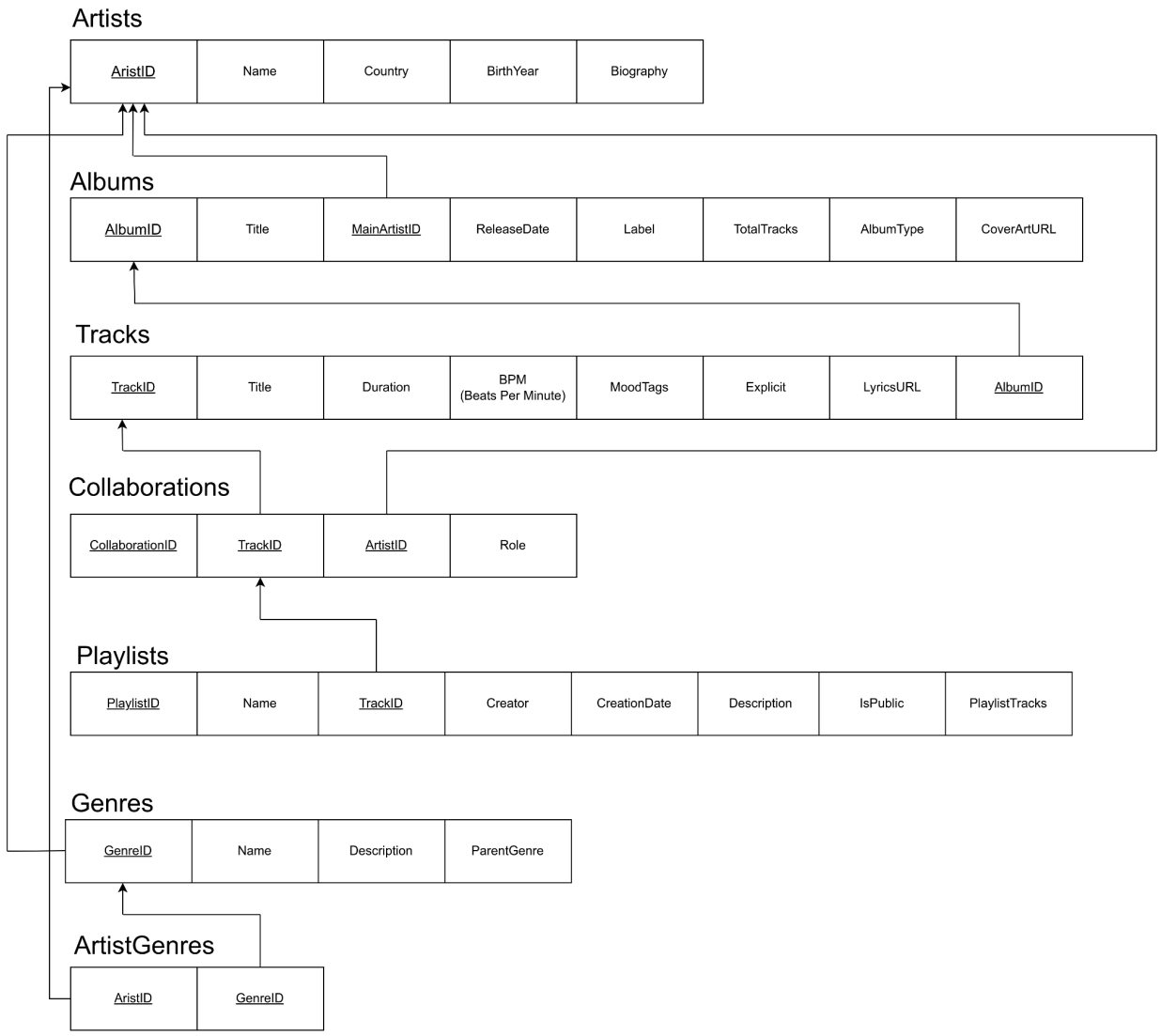
DBT ASSIGNMENT

Name : M C Krishna Kumar

SRN : PES2UG22CS281

Section : E

Relational Schema :



```
mysql> show tables;
+-----------------------------------------+
| Tables_in_DBT25_A1_PES2UG22CS281_Krishna |
+-----------------------------------------+
| Albums                                  |
| ArtistGenres                            |
| Artists                                 |
| Collaborations                          |
| Genres                                  |
| Playlists                               |
| Tracks                                  |
+-----------------------------------------+
7 rows in set (0.00 sec)

mysql>
```

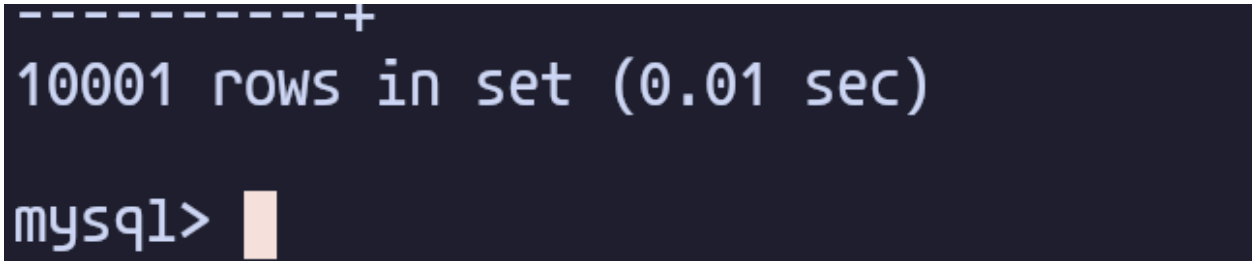
SELECT * FROM Albums;

```
-----+--------------+--------------------
5000 rows in set (0.00 sec)

mysql>
```

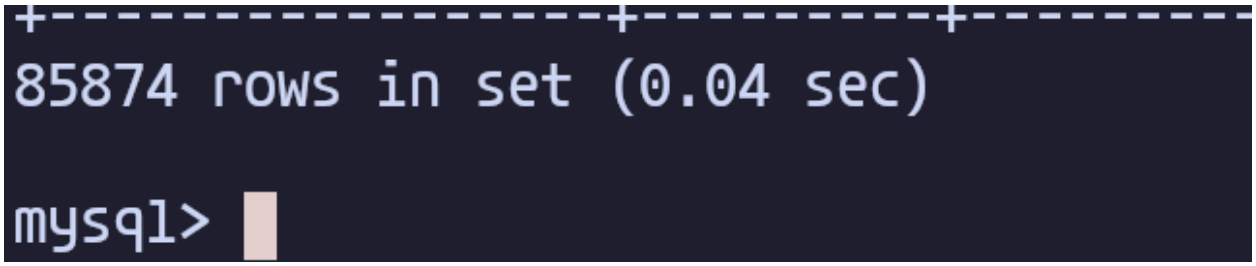
SELECT * FROM ArtistGenres;

```
-----+--------------+--------------------
5000 rows in set (0.00 sec)

mysql>
```

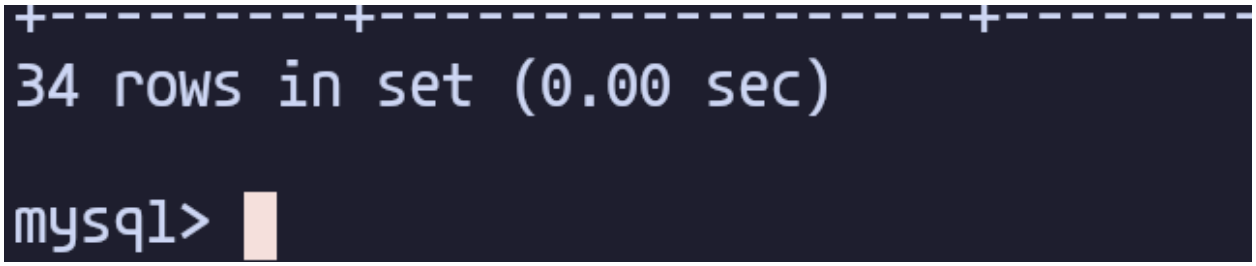
SELECT * FROM Artists;



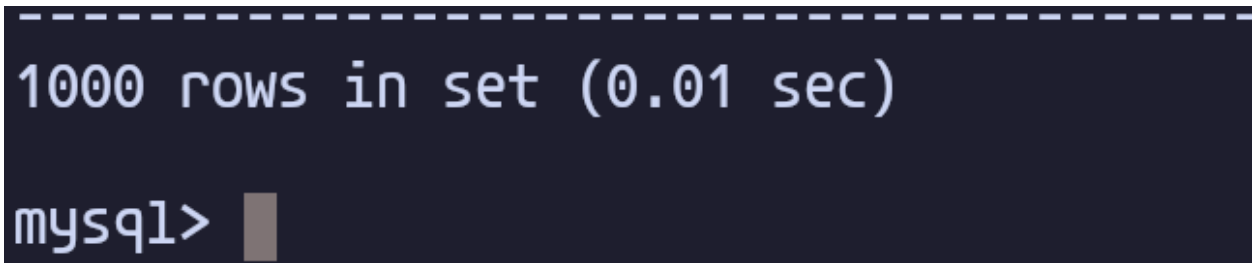
SELECT * FROM Collaborations;



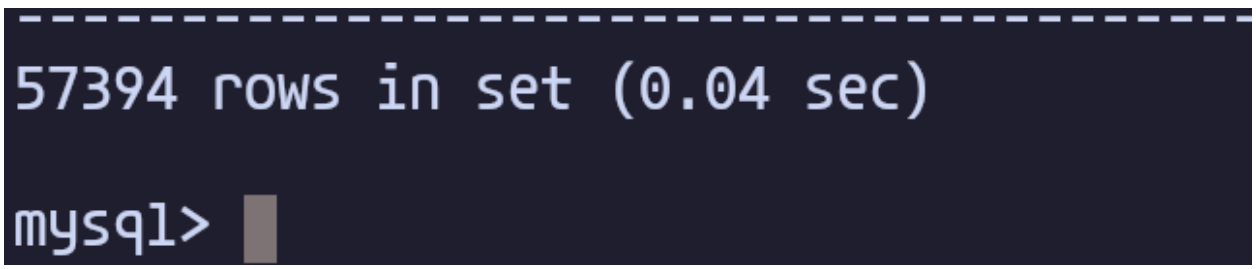
SELECT * FROM Genres;



SELECT * FROM Playlists;



SELECT * FROM Tracks;



Without Indexes

```
mysql> EXPLAIN SELECT * FROM Artists;
+----+-------------+---------+------------+------+---------------+------+---------+------+------+----------+-------+
| id | select_type | table   | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra |
+----+-------------+---------+------------+------+---------------+------+---------+------+------+----------+-------+
|  1 | SIMPLE      | Artists | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 9744 |   100.00 | NULL  |
+----+-------------+---------+------------+------+---------------+------+---------+------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)

mysql>
 mysql -u root -p   nvim main.py
```

```
mysql> EXPLAIN
    -> ANALYZE
    -> SELECT * FROM Artists;
+----------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                      |
+----------------------------------------------------------------------------------------------+
| -> Table scan on Artists  (cost=1063 rows=9744) (actual time=0.0325..7.71 rows=10001 loops=1)
 |
+----------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)

mysql>
```

Here since i am running a SELECT * command with or without indexes does not matter because either way it will perform a full table scan and that is what mysql will prefer.

Now Here is an example of SELECT * command with a WHERE clause (without indexes).

```
mysql> SELECT * FROM Artists WHERE Biography = "PES2UG22CS281";
+----------+------+---------+-----------+---------------+
| ArtistID | Name | Country | BirthYear | Biography     |
+----------+------+---------+-----------+---------------+
|    10001 | KK   | India   |      2004 | PES2UG22CS281 |
+----------+------+---------+-----------+---------------+
1 row in set (0.01 sec)

mysql> EXPLAIN ANALYZE SELECT * FROM Artists WHERE Biography = "PES2UG22CS281";
+-----------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------+
| EXPLAIN
                                                              |
+-----------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------+
| -> Filter: (Artists.Biography = 'PES2UG22CS281')  (cost=1063 rows=974) (actual time=3.88..3.88 rows=1 loops=1)
    -> Table scan on Artists  (cost=1063 rows=9744) (actual time=0.0204..3.44 rows=10001 loops=1)
 |
+-----------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------+
1 row in set (0.00 sec)

mysql>
```

Multi-join queries (without indexes)

i)

```sql
-- Query to find all song details where specific Artist collaborated
SELECT t.TrackID, t.title, a.Name AS MainArtist, c.Role
FROM `Tracks` t
JOIN `Albums` al ON t.`AlbumID` = al.`AlbumID`
JOIN `Artists` a ON al.`MainArtistID` = a.`ArtistID`
JOIN `Collaborations` c ON t.`TrackID` = c.`TrackID`
JOIN `Artists` collaber ON c.`ArtistID` = collaber.`ArtistID`
WHERE collaber.`Name` = "Laura Williams";
```

```
mysql> SELECT t.TrackID, t.title, a.Name AS MainArtist, c.Role
    -> FROM `Tracks` t
    -> JOIN `Albums` al ON t.`AlbumID` = al.`AlbumID`
    -> JOIN `Artists` a ON al.`MainArtistID` = a.`ArtistID`
    -> JOIN `Collaborations` c ON t.`TrackID` = c.`TrackID`
    -> JOIN `Artists` collaber ON c.`ArtistID` = collaber.`ArtistID`
    -> WHERE collaber.`Name` = "Laura Williams";
+---------+-----------------------------------------+------------------+------------------+
| TrackID | title                                   | MainArtist       | Role             |
+---------+-----------------------------------------+------------------+------------------+
|    6612 | Grass-roots bi-directional matrices     | Jacob Butler     | Background Vocals |
|   30652 | Persevering web-enabled strategy        | Patricia Smith   | Featured Artist  |
|   30712 | Cross-platform composite infrastructure | Daniel Chambers  | Background Vocals |
|   32704 | Reverse-engineered leadingedge model    | Ryan Short       | Producer         |
|   36675 | Adaptive multi-state open system        | Lisa King        | Featured Artist  |
|   49038 | Proactive global alliance               | Sonia Gallagher  | Featured Artist  |
|   49608 | Focused zero administration extranet    | Stephanie Watkins | Background Vocals |
|   55131 | Networked high-level service-desk       | John Ray         | Featured Artist  |
|   18609 | Profit-focused mobile knowledge user    | Thomas Brown     | Featured Artist  |
|   19108 | Reduced context-sensitive standardization | Noah Allen     | Producer         |
|   22939 | Diverse local architecture              | Bruce Kim        | Background Vocals |
|   24455 | Versatile disintermediate standardization | Samantha Chandler | Featured Artist |
|   28272 | Virtual cohesive moratorium             | Keith Mcguire    | Producer         |
|   34144 | Reactive optimal analyzer               | Erik Bailey      | Background Vocals |
|   34870 | Implemented radical definition          | Daniel Cross     | Featured Artist  |
|   44620 | Sharable fresh-thinking forecast        | Eric Stein       | Songwriter       |
|   45387 | De-engineered upward-trending capacity  | Dean Bennett     | Background Vocals |
|   47053 | Secured upward-trending Internet solution | Wendy Ward     | Songwriter       |
|   48698 | Polarized radical matrix                | Cassandra Walsh  | Songwriter       |
|   56511 | Profound attitude-oriented system engine | Lisa Foster     | Songwriter       |
+---------+-----------------------------------------+------------------+------------------+
```

```
-> Nested loop inner join  (cost=24463 rows=8307) (actual time=0.183..7.24 rows=20 loops=1)
  -> Nested loop inner join  (cost=21556 rows=8307) (actual time=0.174..6.89 rows=20 loops=1)
    -> Nested loop inner join  (cost=18649 rows=8307) (actual time=0.166..6.61 rows=20 loops=1)
      -> Nested loop inner join  (cost=9736 rows=8307) (actual time=0.153..6.43 rows=20 loops=1)
        -> Filter: (collaber.`Name` = 'Laura Williams')  (cost=1063 rows=974) (actual time=0.0856..6.25 rows=2 loops=1)
          -> Table scan on collaber  (cost=1063 rows=9744) (actual time=0.0821..5.01 rows=10001 loops=1)
        -> Filter: (c.TrackID is not null)  (cost=8.05 rows=8.52) (actual time=0.0702..0.0856 rows=10 loops=2)
          -> Index lookup on c using ArtistID (ArtistID=collaber.ArtistID)  (cost=8.05 rows=8.52) (actual time=0.0693..0.0822 rows=10 loops=2)
      -> Filter: (t.AlbumID is not null)  (cost=0.973 rows=1) (actual time=0.00838..0.00859 rows=1 loops=20)
        -> Single-row index lookup on t using PRIMARY (TrackID=c.TrackID)  (cost=0.973 rows=1) (actual time=0.00805..0.00811 rows=1 loops=20)
    -> Filter: (al.MainArtistID is not null)  (cost=0.25 rows=1) (actual time=0.0134..0.0136 rows=1 loops=20)
      -> Single-row index lookup on al using PRIMARY (AlbumID=t.AlbumID)  (cost=0.25 rows=1) (actual time=0.0131..0.0132 rows=1 loops=20)
  -> Single-row index lookup on a using PRIMARY (ArtistID=al.MainArtistID)  (cost=0.25 rows=1) (actual time=0.0171..0.0171 rows=1 loops=20)
```

ii)

```
19    -- Find the Top 10 Most Collaborative Artists
      ▷Run | +Tab | JSON | Select | Ask Copilot
20    SELECT a.ArtistID, a.Name, COUNT(c.CollaborationID) AS NumCollaborations
21    FROM Artists a
22    JOIN Collaborations c ON a.ArtistID = c.ArtistID
23    GROUP BY a.ArtistID, a.Name
24    ORDER BY NumCollaborations DESC
25    LIMIT 10;  54ms
```

```
mysql> SELECT a.ArtistID, a.Name, COUNT(c.CollaborationID) AS NumCollaborations
    -> FROM Artists a
    -> JOIN Collaborations c ON a.ArtistID = c.ArtistID
    -> GROUP BY a.ArtistID, a.Name
    -> ORDER BY NumCollaborations DESC
    -> LIMIT 10;
+----------+----------------------+-------------------+
| ArtistID | Name                 | NumCollaborations |
+----------+----------------------+-------------------+
|     4978 | Antonio Harrison DVM |                20 |
|     3107 | Anthony Arellano     |                19 |
|     2384 | Marcus Yu            |                19 |
|     3476 | Lauren Taylor        |                19 |
|     1801 | Donna Brown          |                19 |
|     6964 | John Martin          |                19 |
|     9523 | Jonathan Johnson     |                19 |
|       30 | Mark Ortiz           |                18 |
|      452 | Julie Evans          |                18 |
|     1665 | Kristina Morrow      |                18 |
+----------+----------------------+-------------------+
10 rows in set (0.07 sec)

mysql>
```

```
-> Limit: 10 row(s)  (actual time=62.5..62.5 rows=10 loops=1)
  -> Sort: NumCollaborations DESC, limit input to 10 row(s) per chunk  (actual time=62.5..62.5 rows=10 loops=1)
      -> Table scan on <temporary>  (actual time=61..62 rows=9994 loops=1)
          -> Aggregate using temporary table  (actual time=61..61 rows=9994 loops=1)
              -> Nested loop inner join  (cost=11825 rows=83065) (actual time=0.102..27.6 rows=85874 loops=1)
                  -> Table scan on a  (cost=1063 rows=9744) (actual time=0.0771..2.21 rows=10001 loops=1)
                  -> Covering index lookup on c using ArtistID (ArtistID=a.ArtistID)  (cost=0.252 rows=8.52) (actual time=0.0014..0.00221 rows=8.59 loops=10001)
```

iii)

```sql
-- Find Albums with the Most Collaborations
SELECT al.AlbumID, al.Title, al.MainArtistID, COUNT(c.CollaborationID) AS MaxCOllabs
FROM `Albums` al
JOIN `Tracks` t ON t.`AlbumID` = al.`AlbumID`
JOIN `Collaborations` c ON c.`TrackID` = t.`TrackID`
GROUP BY al.`AlbumID`, al.`Title`
ORDER BY MaxCollabs DESC
LIMIT 10;
```

144ms

```
mysql> SELECT al.AlbumID, al.Title, al.MainArtistID, COUNT(c.CollaborationID) AS MaxCOllabs
    -> FROM `Albums` al
    -> JOIN `Tracks` t ON t.`AlbumID` = al.`AlbumID`
    -> JOIN `Collaborations` c ON c.`TrackID` = t.`TrackID`
    -> GROUP BY al.`AlbumID`, al.`Title`
    -> ORDER BY MaxCollabs DESC
    -> LIMIT 10;
+---------+-----------------------------------------------+--------------+-------------+
| AlbumID | Title                                         | MainArtistID | MaxCOllabs |
+---------+-----------------------------------------------+--------------+-------------+
|    2607 | Streamlined local orchestration               |         9376 |          36 |
|    2595 | Inverse hybrid policy                         |         8945 |          36 |
|    3234 | Digitized 5thgeneration functionalities       |         7896 |          33 |
|    3290 | Ameliorated logistical task-force             |          418 |          33 |
|    3901 | Vision-oriented mission-critical leverage     |         6814 |          33 |
|     206 | Open-source context-sensitive moratorium      |         9622 |          33 |
|    4523 | Ameliorated fault-tolerant workforce          |         4130 |          33 |
|    3474 | Profound responsive approach                  |          742 |          32 |
|    3547 | Fully-configurable fresh-thinking throughput  |         8793 |          32 |
|     639 | Customizable 5thgeneration attitude           |         4934 |          32 |
+---------+-----------------------------------------------+--------------+-------------+
10 rows in set (0.14 sec)

mysql>
```

```
-> Limit: 10 row(s)  (actual time=156..156 rows=10 loops=1)
  -> Sort: MaxCOllabs DESC, limit input to 10 row(s) per chunk  (actual time=156..156 rows=10 loops=1)
    -> Table scan on <temporary>  (actual time=155..156 rows=5000 loops=1)
      -> Aggregate using temporary table  (actual time=155..155 rows=5000 loops=1)
        -> Nested loop inner join  (cost=33570 rows=119176) (actual time=0.129..111 rows=85874 loops=1)
          -> Nested loop inner join  (cost=7458 rows=56708) (actual time=0.116..22 rows=57394 loops=1)
            -> Table scan on al  (cost=524 rows=4997) (actual time=0.0934..1.15 rows=5000 loops=1)
            -> Covering index lookup on t using AlbumID (AlbumID=al.AlbumID)  (cost=0.253 rows=11.3) (actual time=0.00102..0.00375 rows=11.5 loops=5000)
          -> Covering index lookup on c using TrackID (TrackID=t.TrackID)  (cost=0.25 rows=2.1) (actual time=0.00116..0.00142 rows=1.5 loops=57394)
```

Adding Indexes and executing the same queries as above

- Starting with creating Index only on Primary Key

```
mysql> CREATE INDEX pri_artistID ON Artists(ArtistID);
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN SELECT * FROM Artists;
+----+-------------+---------+------------+------+---------------+------+---------+------+------+----------+-------+
| id | select_type | table   | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra |
+----+-------------+---------+------------+------+---------------+------+---------+------+------+----------+-------+
|  1 | SIMPLE      | Artists | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 9744 |   100.00 | NULL  |
+----+-------------+---------+------------+------+---------------+------+---------+------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)

mysql> EXPLAIN ANALYZE SELECT * FROM Artists;
+-----------------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                             |
+-----------------------------------------------------------------------------------------------------+
| -> Table scan on Artists  (cost=1063 rows=9744) (actual time=0.0407..6.46 rows=10001 loops=1)
 |
+-----------------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

As we can see even though we have created an index on the primary key of Artists. It does barely has any effect as mentioned earlier we are doing a full table scan and in SELECT * statements the indexes are not utilized.

Now let us run the same command using a where clause-

```
mysql> EXPLAIN ANALYZE SELECT * FROM Artists WHERE ArtistID=587;
+-----------------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                             |
+-----------------------------------------------------------------------------------------------------+
| -> Rows fetched before execution  (cost=0..0 rows=1) (actual time=148e-6..285e-6 rows=1 loops=1)
 |
+-----------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)

mysql>
```

As we can see using Index the search speed is significantly faster.

Note : When tables are created indexes are created on the primary key by default. So creating an index again on the primary the key does not make sense.

Now let us create an index on Name, here is output with and without index on Artist.

```
mysql> EXPLAIN ANALYZE SELECT * FROM Artists WHERE Name = 'Eminem';
+--------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------+
| EXPLAIN
                                                    |
+--------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------+
| -> Filter: (Artists.`Name` = 'Eminem')  (cost=1063 rows=974) (actual time=10.5..10.5 rows=0 loops=1)
    -> Table scan on Artists  (cost=1063 rows=9744) (actual time=0.0346..9.36 rows=10001 loops=1)
 |
+--------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------+
1 row in set (0.01 sec)

mysql> CREATE INDEX name_indi ON Artists(Name);
Query OK, 0 rows affected (0.06 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE SELECT * FROM Artists WHERE Name = 'Eminem';
+------------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                                                      |
+------------------------------------------------------------------------------------------------------------------------------+
| -> Index lookup on Artists using name_indi (Name='Eminem')  (cost=0.35 rows=1) (actual time=0.0205..0.0205 rows=0 loops=1)
 |
+------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)

mysql> 
```

This example makes it clear that using an index reduces time and cost significantly.

Alternatively we can also use this method to analyze time taken by queries to

```
mysql> SET profiling = 1;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> SELECT * FROM Artists WHERE Name = 'kk';
+----------+------+---------+-----------+--------------+
| ArtistID | Name | Country | BirthYear | Biography    |
+----------+------+---------+-----------+--------------+
|    10001 | KK   | India   |      2004 | PES2UG22CS281 |
+----------+------+---------+-----------+--------------+
1 row in set (0.00 sec)

mysql> SHOW PROFILES;
+----------+------------+-------------------------------------------+
| Query_ID | Duration   | Query                                     |
+----------+------------+-------------------------------------------+
|        1 | 0.00066150 | SELECT * FROM Artists WHERE Name = 'Eminem' |
|        2 | 0.00068450 | SELECT * FROM Artists WHERE Name = 'kk'     |
|        3 | 0.00052025 | SELECT * FROM Artists WHERE Name = 'kk'     |
+----------+------------+-------------------------------------------+
3 rows in set, 1 warning (0.00 sec)

mysql> 
```

Now let us create indexes for other columns which were used in multi-way joins.
i)

```
-- Query to find all song details where specific Artist collaborated
▷Run | Select | Ask Copilot
EXPLAIN ANALYZE SELECT t.TrackID, t.title, a.Name AS MainArtist, c.Role
FROM `Tracks` t
JOIN `Albums` al ON t.`AlbumID` = al.`AlbumID`
JOIN `Artists` a ON al.`MainArtistID` = a.`ArtistID`
JOIN `Collaborations` c ON t.`TrackID` = c.`TrackID`
JOIN `Artists` collaber ON c.`ArtistID` = collaber.`ArtistID`
WHERE collaber.`Name` = "Laura Williams";
```

MainArtistID Albums is Foreign Key.
ArtistID, TrackID in Collaborations is Foreign Key.
AlbumID in Tracks is Foreign Key.

Let us create indexes on these foreign keys

```
43  CREATE INDEX main_artist_Albums ON Albums(MainArtistID);
    ▷Run
44  CREATE INDEX artistID_Collab ON Collaborations(ArtistID);
    ▷Run
45  CREATE INDEX trackID_Collab ON Collaborations(TrackID);
    ▷Run
46  CREATE INDEX albumID_Tracks ON Tracks(AlbumID);
```

```
mysql> SHOW INDEXES FROM Collaborations;
+----------------+------------+----------------+--------------+----------------+-----------+-------------+----------+--------+------+------------+
| Table          | Non_unique | Key_name       | Seq_in_index | Column_name    | Collation | Cardinality | Sub_part | Packed | Null | Index_type |
+----------------+------------+----------------+--------------+----------------+-----------+-------------+----------+--------+------+------------+
| Collaborations |          0 | PRIMARY        |            1 | CollaborationID | A         |       86100 |     NULL |   NULL |      | BTREE      |
| Collaborations |          1 | artistID_Collab |            1 | ArtistID       | A         |       10113 |     NULL |   NULL | YES  | BTREE      |
| Collaborations |          1 | trackID_Collab  |            1 | TrackID        | A         |       43497 |     NULL |   NULL | YES  | BTREE      |
+----------------+------------+----------------+--------------+----------------+-----------+-------------+----------+--------+------+------------+
3 rows in set (0.00 sec)
```

```
mysql> SHOW INDEXES FROM Tracks;
+--------+------------+---------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+
| Table  | Non_unique | Key_name      | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type |
+--------+------------+---------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+
| Tracks |          0 | PRIMARY       |            1 | TrackID     | A         |       56878 |     NULL |   NULL |      | BTREE      |
| Tracks |          1 | albumID_Tracks |            1 | AlbumID     | A         |        5077 |     NULL |   NULL | YES  | BTREE      |
+--------+------------+---------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+
2 rows in set (0.01 sec)
```

```
mysql> SHOW INDEXES FROM Albums;
+--------+------------+--------------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+
| Table  | Non_unique | Key_name           | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type |
+--------+------------+--------------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+
| Albums |          0 | PRIMARY            |            1 | AlbumID     | A         |        4997 |     NULL |   NULL |      | BTREE      |
| Albums |          1 | main_artist_Albums |            1 | MainArtistID | A         |        3918 |     NULL |   NULL | YES  | BTREE      |
+--------+------------+--------------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+
2 rows in set (0.01 sec)
```

Running the Query :

```
mysql> EXPLAIN ANALYZE SELECT t.TrackID, t.title, a.Name AS MainArtist, c.Role
    -> FROM `Tracks` t
    -> JOIN `Albums` al ON t.`AlbumID` = al.`AlbumID`
    -> JOIN `Artists` a ON al.`MainArtistID` = a.`ArtistID`
    -> JOIN `Collaborations` c ON t.`TrackID` = c.`TrackID`
    -> JOIN `Artists` collaber ON c.`ArtistID` = collaber.`ArtistID`
    -> WHERE collaber.`Name` = "Laura Williams";
```

```
-> Nested loop inner join  (cost=37.1 rows=17) (actual time=0.12..0.58 rows=20 loops=1)
 -> Nested loop inner join  (cost=19.1 rows=17) (actual time=0.11..0.459 rows=20 loops=1)
    -> Nested loop inner join  (cost=13.2 rows=17) (actual time=0.102..0.346 rows=20 loops=1)
       -> Nested loop inner join  (cost=7.23 rows=17) (actual time=0.0884..0.19 rows=20 loops=1)
          -> Covering index lookup on collaber using name_indi (Name='Laura Williams')  (cost=1.27 rows=2) (actual time=0.0263..0.0284 rows=2 loops=1)
          -> Filter: (c.TrackID is not null)  (cost=2.55 rows=8.51) (actual time=0.0644..0.0787 rows=10 loops=2)
             -> Index lookup on c using artistID_Collab (ArtistID=collaber.ArtistID)  (cost=2.55 rows=8.51) (actual time=0.0638..0.076 rows=10 loops=2)
       -> Filter: (t.AlbumID is not null)  (cost=0.256 rows=1) (actual time=0.00721..0.0074 rows=1 loops=20)
          -> Single-row index lookup on t using PRIMARY (TrackID=c.TrackID)  (cost=0.256 rows=1) (actual time=0.00689..0.00694 rows=1 loops=20)
    -> Filter: (al.MainArtistID is not null)  (cost=0.256 rows=1) (actual time=0.00508..0.00525 rows=1 loops=20)
       -> Single-row index lookup on al using PRIMARY (AlbumID=t.AlbumID)  (cost=0.256 rows=1) (actual time=0.00481..0.00486 rows=1 loops=20)
 -> Single-row index lookup on a using PRIMARY (ArtistID=al.MainArtistID)  (cost=0.961 rows=1) (actual time=0.00553..0.00559 rows=1 loops=20)
```

ii)

```
-- Find Albums with the Most Collaborations
▷Run | +Tab | JSON | Select | Ask Copilot
SELECT al.AlbumID, al.Title, al.MainArtistID, COUNT(c.CollaborationID) AS MaxCOllabs
FROM `Albums` al
JOIN `Tracks` t ON t.`AlbumID` = al.`AlbumID`
JOIN `Collaborations` c ON c.`TrackID` = t.`TrackID`
GROUP BY al.`AlbumID`, al.`Title`
ORDER BY MaxCollabs DESC
LIMIT 10;
```

Since we have already created the necessary indexes for this query, let us run it and verify the speed-ups and cost.

```
mysql> EXPLAIN ANALYZE SELECT al.AlbumID, al.Title, al.MainArtistID, COUNT(c.CollaborationID) AS MaxCOllabs
    -> FROM `Albums` al
    -> JOIN `Tracks` t ON t.`AlbumID` = al.`AlbumID`
    -> JOIN `Collaborations` c ON c.`TrackID` = t.`TrackID`
    -> GROUP BY al.`AlbumID`, al.`Title`
    -> ORDER BY MaxCollabs DESC
    -> LIMIT 10;
```

```
-> Limit: 10 row(s)  (actual time=219..219 rows=10 loops=1)
 -> Sort: MaxCOllabs DESC, limit input to 10 row(s) per chunk  (actual time=219..219 rows=10 loops=1)
    -> Table scan on <temporary>  (actual time=217..218 rows=5000 loops=1)
       -> Aggregate using temporary table  (actual time=217..217 rows=5000 loops=1)
          -> Nested loop inner join  (cost=68952 rows=86100) (actual time=1.3..146 rows=85874 loops=1)
             -> Nested loop inner join  (cost=38817 rows=86100) (actual time=1.29..116 rows=85874 loops=1)
                -> Filter: (c.TrackID is not null)  (cost=8682 rows=86100) (actual time=1.27..25.3 rows=85874 loops=1)
                   -> Covering index scan on c using trackID_Collab  (cost=8682 rows=86100) (actual time=1.27..19.4 rows=85874 loops=1)
                -> Filter: (t.AlbumID is not null)  (cost=0.25 rows=1) (actual time=856e-6..927e-6 rows=1 loops=85874)
                   -> Single-row index lookup on t using PRIMARY (TrackID=c.TrackID)  (cost=0.25 rows=1) (actual time=745e-6..768e-6 rows=1 loops=85874)
             -> Single-row index lookup on al using PRIMARY (AlbumID=t.AlbumID)  (cost=0.25 rows=1) (actual time=180e-6..202e-6 rows=1 loops=85874)
```

Conclusion: By adding indexes on all Foreign Keys involved in the Joins we see a significant improvement in performance.

Now let us try changing the order of joins and check whether it benefits us or not. Let us consider the query we took initially-

```
-- Query to find all song details where specific Artist collaborated
▷Run | Select | Ask Copilot
EXPLAIN ANALYZE SELECT t.TrackID, t.title, a.Name AS MainArtist, c.Role
FROM `Tracks` t
JOIN `Albums` al ON t.`AlbumID` = al.`AlbumID`
JOIN `Artists` a ON al.`MainArtistID` = a.`ArtistID`
JOIN `Collaborations` c ON t.`TrackID` = c.`TrackID`
JOIN `Artists` collaber ON c.`ArtistID` = collaber.`ArtistID`
WHERE collaber.`Name` = "Laura Williams";
```

```
12 • EXPLAIN ANALYZE
13   SELECT t.TrackID, t.Title, a.Name AS MainArtist, c.Role
14   FROM Tracks t
15   JOIN Albums al ON t.AlbumID = al.AlbumID
16   JOIN Artists a ON al.MainArtistID = a.ArtistID
17   JOIN Collaborations c ON t.TrackID = c.TrackID
18   JOIN Artists collaber ON c.ArtistID = collaber.ArtistID
19   WHERE collaber.Name = "Laura Williams";
20
```

Query Statistics

Timing (as measured at client side):
Execution time: 0:00:0.00207305

Timing (as measured by the server):
Execution time: 0:00:0.00193405
Table lock wait time: 0:00:0.00000400

Errors:
Had Errors: NO
Warnings: 0

Rows Processed:
Rows affected: 0
Rows sent to client: 1
Rows examined: 82

Joins per Type:
Full table scans (Select_scan): 0
Joins using table scans (Select_full_join): 0
Joins using range search (Select_full_range_join): 0
Joins with range checks (Select_range_check): 0
Joins using range (Select_range): 0

Sorting:
Sorted rows (Sort_rows): 0
Sort merge passes (Sort_merge_passes): 0
Sorts with ranges (Sort_range): 0
Sorts with table scans (Sort_scan): 0

Index Usage:
At least one Index was used

i) Here since our filter is first applied to get Artists.Name so let us fetch the Artists first.

```
53   EXPLAIN ANALYZE
54   SELECT t.TrackID, t.Title, a.Name AS MainArtist, c.Role
55   FROM `Artists` collaber
56   JOIN `Collaborations` c ON collaber.ArtistID = c.ArtistID
57   JOIN `Tracks` t ON c.TrackID = t.TrackID
58   JOIN `Albums` al ON t.AlbumID = al.AlbumID
59   JOIN `Artists` a ON al.MainArtistID = a.ArtistID
60   WHERE collaber.`Name` = "Laura Williams";
```

```
-> Nested loop inner join  (cost=58.5 rows=17) (actual time=0.0404..0.134 rows=20 loops=1)
  -> Nested loop inner join  (cost=40.6 rows=17) (actual time=0.0374..0.108 rows=20 loops=1)
      -> Nested loop inner join  (cost=28.9 rows=17) (actual time=0.0351..0.082 rows=20 loops=1)
          -> Nested loop inner join  (cost=19 rows=17) (actual time=0.031..0.0497 rows=20 loops=1)
              -> Covering index lookup on collaber using name_indi (Name='Laura Williams')  (cost=1.27 rows=2) (actual time=0.0117..0.0123 rows=2 loops=1)
              -> Filter: (c.TrackID is not null)  (cost=8.46 rows=8.51) (actual time=0.0144..0.0178 rows=10 loops=2)
                  -> Index lookup on c using artistID_Collab (ArtistID=collaber.ArtistID)  (cost=8.46 rows=8.51) (actual time=0.0141..0.017 rows=10 loops=2
          -> Filter: (t.AlbumID is not null)  (cost=0.487 rows=1) (actual time=0.00146..0.00151 rows=1 loops=20)
              -> Single-row index lookup on t using PRIMARY (TrackID=c.TrackID)  (cost=0.487 rows=1) (actual time=0.00137..0.00138 rows=1 loops=20)
      -> Filter: (al.MainArtistID is not null)  (cost=0.589 rows=1) (actual time=0.00117..0.00121 rows=1 loops=20)
          -> Single-row index lookup on al using PRIMARY (AlbumID=t.AlbumID)  (cost=0.589 rows=1) (actual time=0.0011..0.00111 rows=1 loops=20)
  -> Single-row index lookup on a using PRIMARY (ArtistID=al.MainArtistID)  (cost=0.961 rows=1) (actual time=0.00116..0.00117 rows=1 loops=20)
```

Here we can notice that the time of execution has reduced.

Alternate way to verify using workbench-

Initial Query :

```
10 • EXPLAIN ANALYZE
11   SELECT t.TrackID, t.Title, a.Name AS MainArtist, c.Role
12   FROM Tracks t
13   JOIN Albums al ON t.AlbumID = al.AlbumID
14   JOIN Artists a ON al.MainArtistID = a.ArtistID
15   JOIN Collaborations c ON t.TrackID = c.TrackID
16   JOIN Artists collaber ON c.ArtistID = collaber.ArtistID
17   WHERE collaber.Name = "Laura Williams";
```

Query Statistics

Timing (as measured at client side):
Execution time: 0:00:0.00175118

Timing (as measured by the server):
Execution time: 0:00:0.00169882
Table lock wait time: 0:00:0.00000300

Errors:
Had Errors: NO
Warnings: 0

Rows Processed:
Rows affected: 0
Rows sent to client: 1
Rows examined: 82

Temporary Tables:
Temporary disk tables created: 0
Temporary tables created: 0

Joins per Type:
Full table scans (Select_scan): 0
Joins using table scans (Select_full_join): 0
Joins using range search (Select_full_range_join): 0
Joins with range checks (Select_range_check): 0
Joins using range (Select_range): 0

Sorting:
Sorted rows (Sort_rows): 0
Sort merge passes (Sort_merge_passes): 0
Sorts with ranges (Sort_range): 0
Sorts with table scans (Sort_scan): 0

Index Usage:
At least one Index was used

Other Info:
Event Id: 85
Thread Id: 51

Optimized Version :

```
19 • EXPLAIN ANALYZE
20   SELECT t.TrackID, t.Title, a.Name AS MainArtist, c.Role
21   FROM `Artists` collaber
22   JOIN `Collaborations` c ON collaber.ArtistID = c.ArtistID
23   JOIN `Tracks` t ON c.TrackID = t.TrackID
24   JOIN `Albums` al ON t.AlbumID = al.AlbumID
25   JOIN `Artists` a ON al.MainArtistID = a.ArtistID
26   WHERE collaber.`Name` = "Laura Williams";
```

Query Statistics

Timing (as measured at client side):
Execution time: 0:00:0.00158191

Timing (as measured by the server):
Execution time: 0:00:0.00145662
Table lock wait time: 0:00:0.00000200

Errors:
Had Errors: NO
Warnings: 0

Rows Processed:
Rows affected: 0
Rows sent to client: 1
Rows examined: 82

Temporary Tables:
Temporary disk tables created: 0
Temporary tables created: 0

Joins per Type:
Full table scans (Select_scan): 0
Joins using table scans (Select_full_join): 0
Joins using range search (Select_full_range_join): 0
Joins with range checks (Select_range_check): 0
Joins using range (Select_range): 0

Sorting:
Sorted rows (Sort_rows): 0
Sort merge passes (Sort_merge_passes): 0
Sorts with ranges (Sort_range): 0
Sorts with table scans (Sort_scan): 0

Index Usage:
At least one Index was used

Other Info:
Event Id: 89
Thread Id: 51

As we can see that in the Optimized version the execution time has reduced.

ii) Using a subquery within a query and cost analysis

```sql
-- Using a Subquery to Pre-Filter Data for i)
EXPLAIN ANALYZE
SELECT t.TrackID, t.Title, a.Name AS MainArtist, c.Role
FROM (
    SELECT c.TrackID
    FROM `Collaborations` c
    JOIN `Artists` collaber ON collaber.`ArtistID` = c.`ArtistID`
    WHERE collaber.`Name` = "Laura Williams"
) preFilter
JOIN `Tracks` t ON preFilter.`TrackID` = t.`TrackID`
JOIN `Albums` al ON t.`AlbumID` = al.`AlbumID`
JOIN `Artists` a ON al.`MainArtistID` = a.`ArtistID`
JOIN `Collaborations` c on t.`TrackID` = c.`TrackID`;
```

```
-> Nested loop inner join  (cost=93.7 rows=33.7) (actual time=1.31..3.81 rows=51 loops=1)
  -> Nested loop inner join  (cost=58.5 rows=17) (actual time=0.0442..0.204 rows=20 loops=1)
     -> Nested loop inner join  (cost=40.6 rows=17) (actual time=0.0419..0.165 rows=20 loops=1)
        -> Nested loop inner join  (cost=28.9 rows=17) (actual time=0.0397..0.129 rows=20 loops=1)
           -> Nested loop inner join  (cost=19 rows=17) (actual time=0.0342..0.0747 rows=20 loops=1)
              -> Covering index lookup on collaber using name_indi (Name='Laura Williams')  (cost=1.27 rows=2) (actual time=0.0112..0.0143 rows=2 loops=1)
              -> Filter: (c.TrackID is not null)  (cost=8.46 rows=8.51) (actual time=0.0234..0.029 rows=10 loops=2)
                 -> Index lookup on c using artistID_Collab (ArtistID=collaber.ArtistID)  (cost=8.46 rows=8.51) (actual time=0.0232..0.0279 rows=10 loops=2)
           -> Filter: (t.AlbumID is not null)  (cost=0.487 rows=1) (actual time=0.0025..0.00257 rows=1 loops=20)
              -> Single-row index lookup on t using PRIMARY (TrackID=c.TrackID)  (cost=0.487 rows=1) (actual time=0.00239..0.0024 rows=1 loops=20)
        -> Filter: (al.MainArtistID is not null)  (cost=0.589 rows=1) (actual time=0.00162..0.00168 rows=1 loops=20)
           -> Single-row index lookup on al using PRIMARY (AlbumID=t.AlbumID)  (cost=0.589 rows=1) (actual time=0.00153..0.00154 rows=1 loops=20)
     -> Single-row index lookup on a using PRIMARY (ArtistID=al.MainArtistID)  (cost=0.961 rows=1) (actual time=0.00175..0.00178 rows=1 loops=20)
  -> Index lookup on c using trackID_Collab (TrackID=c.TrackID)  (cost=1.88 rows=1.98) (actual time=0.179..0.18 rows=2.55 loops=20)
```

```sql
EXPLAIN ANALYZE
SELECT t.TrackID, t.Title, a.Name AS MainArtist, c.Role
FROM (
    SELECT c.TrackID
    FROM `Collaborations` c
    JOIN `Artists` collaber ON collaber.`ArtistID` = c.`ArtistID`
    WHERE collaber.`Name` = "Laura Williams"
) preFilter
JOIN `Tracks` t ON preFilter.`TrackID` = t.`TrackID`
JOIN `Albums` al ON t.`AlbumID` = al.`AlbumID`
JOIN `Artists` a ON al.`MainArtistID` = a.`ArtistID`
JOIN `Collaborations` c on t.`TrackID` = c.`TrackID`;
```

```
Query Statistics

Timing (as measured at client side):
Execution time: 0:00:0.00208306

Timing (as measured by the server):
Execution time: 0:00:0.00196223
Table lock wait time: 0:00:0.00000400

Errors:
Had Errors: NO
Warnings: 0

Rows Processed:
Rows affected: 0
Rows sent to client: 1
Rows examined: 133

Joins per Type:
Full table scans (Select_scan): 0
Joins using table scans (Select_full_join): 0
Joins using range search (Select_full_range_join): 0
Joins with range checks (Select_range_check): 0
Joins using range (Select_range): 0

Sorting:
Sorted rows (Sort_rows): 0
Sort merge passes (Sort_merge_passes): 0
Sorts with ranges (Sort_range): 0
Sorts with table scans (Sort_scan): 0

Index Usage:
At least one Index was used
```

iii) Using Left Join

```
77   -- Outer Join
     ▷Run | Select | Ask Copilot
78   EXPLAIN ANALYZE
79   SELECT t.TrackID, t.Title, a.Name AS MainArtist, c.Role
80   FROM `Tracks` t
81   LEFT JOIN `Collaborations` c ON t.`TrackID` = c.`TrackID`
82   LEFT JOIN Artists collaber ON c.`ArtistID` = collaber.`ArtistID` AND collaber.`Name` = "Laura Williams"
83   JOIN Albums al ON t.AlbumID = al.AlbumID
84   JOIN Artists a ON al.`MainArtistID` = a.`ArtistID`;
```

```
-> Nested loop left join  (cost=99468 rows=110813) (actual time=0.0732..248 rows=100251 loops=1)
  -> Nested loop left join  (cost=60683 rows=110813) (actual time=0.0694..181 rows=100251 loops=1)
      -> Nested loop inner join  (cost=21899 rows=55982) (actual time=0.063..54.1 rows=57394 loops=1)
          -> Nested loop inner join  (cost=2305 rows=4997) (actual time=0.0482..3.8 rows=5000 loops=1)
              -> Filter: (al.MainArtistID is not null)  (cost=556 rows=4997) (actual time=0.0411..1.08 rows=5000 loops=1)
                  -> Covering index scan on al using main_artist_Albums  (cost=556 rows=4997) (actual time=0.0405..0.828 rows=5000 loops=1)
              -> Single-row index lookup on a using PRIMARY (ArtistID=al.MainArtistID)  (cost=0.25 rows=1) (actual time=453e-6..466e-6 rows=1 loops=5000)
          -> Index lookup on t using albumID_Tracks (AlbumID=al.AlbumID)  (cost=2.8 rows=11.2) (actual time=0.00294..0.00969 rows=11.5 loops=5000)
      -> Index lookup on c using trackID_Collab (TrackID=t.TrackID)  (cost=0.495 rows=1.98) (actual time=0.00185..0.00211 rows=1.5 loops=57394)
  -> Filter: (collaber.`Name` = 'Laura Williams')  (cost=0.25 rows=1) (actual time=597e-6..598e-6 rows=199e-6 loops=100251)
      -> Single-row index lookup on collaber using PRIMARY (ArtistID=c.ArtistID)  (cost=0.25 rows=1) (actual time=479e-6..490e-6 rows=0.857 loops=100251)
```

```
50 • EXPLAIN ANALYZE
51   SELECT t.TrackID, t.Title, a.Name AS MainArtist, c.Role
52   FROM Tracks t
53   LEFT JOIN Collaborations c ON t.TrackID = c.TrackID
54   LEFT JOIN Artists collaber ON c.ArtistID = collaber.ArtistID AND collaber.Name = "Laura Williams"
55   JOIN Albums al ON t.AlbumID = al.AlbumID
56   JOIN Artists a ON al.MainArtistID = a.ArtistID;
57
```

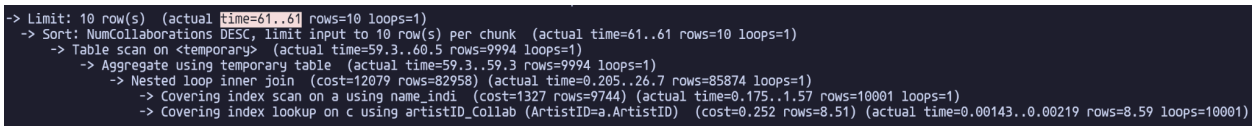
```
Query Statistics

Timing (as measured at client side):
Execution time: 0:00:0.25411391

Timing (as measured by the server):
Execution time: 0:00:0.25400799
Table lock wait time: 0:00:0.00000300

Errors:
Had Errors: NO
Warnings: 0

Rows Processed:
Rows affected: 0
Rows sent to client: 1
Rows examined: 239142

Joins per Type:
Full table scans (Select_scan): 0
Joins using table scans (Select_full_join): 0
Joins using range search (Select_full_range_join): 0
Joins with range checks (Select_range_check): 0
Joins using range (Select_range): 0

Sorting:
Sorted rows (Sort_rows): 0
Sort merge passes (Sort_merge_passes): 0
Sorts with ranges (Sort_range): 0
Sorts with table scans (Sort_scan): 0

Index Usage:
At least one Index was used
```

Since Left Join returns all rows from the left table even if no corresponding value is found in the right table it takes more time than the previous queries.

Consider this simple query

```
20   EXPLAIN ANALYZE SELECT a.ArtistID, a.Name, COUNT(c.CollaborationID) AS NumCollaborations
21   FROM Artists a
22   JOIN Collaborations c ON a.ArtistID = c.ArtistID
23   GROUP BY a.ArtistID, a.Name
24   ORDER BY NumCollaborations DESC
25   LIMIT 10;       You, 16 hours ago • cleanUP
```

Without any optimizations the performance would be-



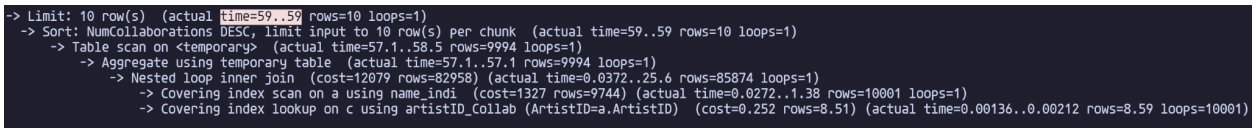
Optimization-

Push the Aggregate COUNT down to ORDER BY Clause, this helps because as the Artists relation has a lot of tuples it is not efficient to count the total number of collaborations initially and then sort them in descending order as this will cause overhead.

Instead we will focus on ordering while it calculates the COUNT which will be faster

Optimized Query-

```
94  EXPLAIN ANALYZE SELECT a.ArtistID, a.Name, COUNT(c.CollaborationID) AS NumCollaborations
95  FROM Artists a
96  JOIN Collaborations c ON a.ArtistID = c.ArtistID
97  GROUP BY a.ArtistID, a.Name
98  ORDER BY COUNT(c.CollaborationID) DESC
99  LIMIT 10;
```



As we can see the time has decreased, the decrease is not that much because i have created indexes priorly on all foreign keys used in the query.

```sql
-- Find Albums with the Most Collaborations
EXPLAIN ANALYZE SELECT al.AlbumID, al.Title, al.MainArtistID, COUNT(c.CollaborationID) AS MaxCollabs
FROM `Albums` al
JOIN `Tracks` t ON t.`AlbumID` = al.`AlbumID`
JOIN `Collaborations` c ON c.`TrackID` = t.`TrackID`
GROUP BY al.`AlbumID`, al.`Title`
ORDER BY MaxCollabs DESC
LIMIT 10;
```

Without Optimization-

```
-> Limit: 10 row(s)  (actual time=141..141 rows=10 loops=1)
  -> Sort: MaxCollabs DESC, limit input to 10 row(s) per chunk  (actual time=141..141 rows=10 loops=1)
      -> Table scan on <temporary>  (actual time=140..141 rows=5000 loops=1)
          -> Aggregate using temporary table  (actual time=140..140 rows=5000 loops=1)
              -> Nested loop inner join  (cost=32477 rows=110813) (actual time=0.0648..99.1 rows=85874 loops=1)
                  -> Nested loop inner join  (cost=7385 rows=55982) (actual time=0.059..20.9 rows=57394 loops=1)
                      -> Table scan on al  (cost=524 rows=4997) (actual time=0.0484..1.01 rows=5000 loops=1)
                      -> Covering index lookup on t using albumID_Tracks (AlbumID=al.AlbumID)  (cost=0.253 rows=11.2) (actual time=0.00103..0.00359 rows=11.5 loops=5000)
                  -> Covering index lookup on c using trackID_Collab (TrackID=t.TrackID)  (cost=0.25 rows=1.98) (actual time=0.00103..0.00126 rows=1.5 loops=57394)
```

Using sub-query optimization-

```sql
EXPLAIN ANALYZE
SELECT al.AlbumID, al.Title, al.MainArtistID, MaxCollabs
FROM Albums al
JOIN (
    SELECT t.AlbumID, COUNT(c.CollaborationID) AS MaxCollabs
    FROM Tracks t
    JOIN Collaborations c ON c.TrackID = t.TrackID
    GROUP BY t.AlbumID
) AS collabCounts ON al.AlbumID = collabCounts.AlbumID
ORDER BY MaxCollabs DESC
LIMIT 10;
```

```
-> Limit: 10 row(s)  (cost=79542 rows=10) (actual time=88.8..88.8 rows=10 loops=1)
  -> Nested loop inner join  (cost=79542 rows=5077) (actual time=88.8..88.8 rows=10 loops=1)
      -> Sort: collabCounts.MaxCollabs DESC  (cost=50888..50888 rows=5077) (actual time=88.8..88.8 rows=10 loops=1)
          -> Filter: (collabCounts.AlbumID is not null)  (cost=43557..44130 rows=5077) (actual time=87.7..88.1 rows=5000 loops=1)
              -> Table scan on collabCounts  (cost=43557..43623 rows=5077) (actual time=87.7..87.9 rows=5000 loops=1)
                  -> Materialize  (cost=43557..43557 rows=5077) (actual time=87.7..87.7 rows=5000 loops=1)
                      -> Group aggregate: count(c.CollaborationID)  (cost=43049 rows=5077) (actual time=0.0568..87.3 rows=5000 loops=1)
                          -> Nested loop inner join  (cost=31790 rows=112587) (actual time=0.032..83.4 rows=85874 loops=1)
                              -> Covering index scan on t using albumID_Tracks  (cost=6297 rows=56878) (actual time=0.022..8.67 rows=57394 loops=1)
                              -> Covering index lookup on c using trackID_Collab (TrackID=t.TrackID)  (cost=0.25 rows=1.98) (actual time=990e-6..0.0012 rows=1.5 loops=57394)
      -> Single-row index lookup on al using PRIMARY (AlbumID=collabCounts.AlbumID)  (cost=0.25 rows=1) (actual time=0.00364..0.00365 rows=1 loops=10)
```

Why does this work ?

In the first query we are joining all albums with tracks => data size is large, instead we can compute only the required collaboration count then join with albums later. Processing only relevant albums with collaborations which reduces data size and prevents sorting of a large dataset.

Hand-Drawn Diagrams :

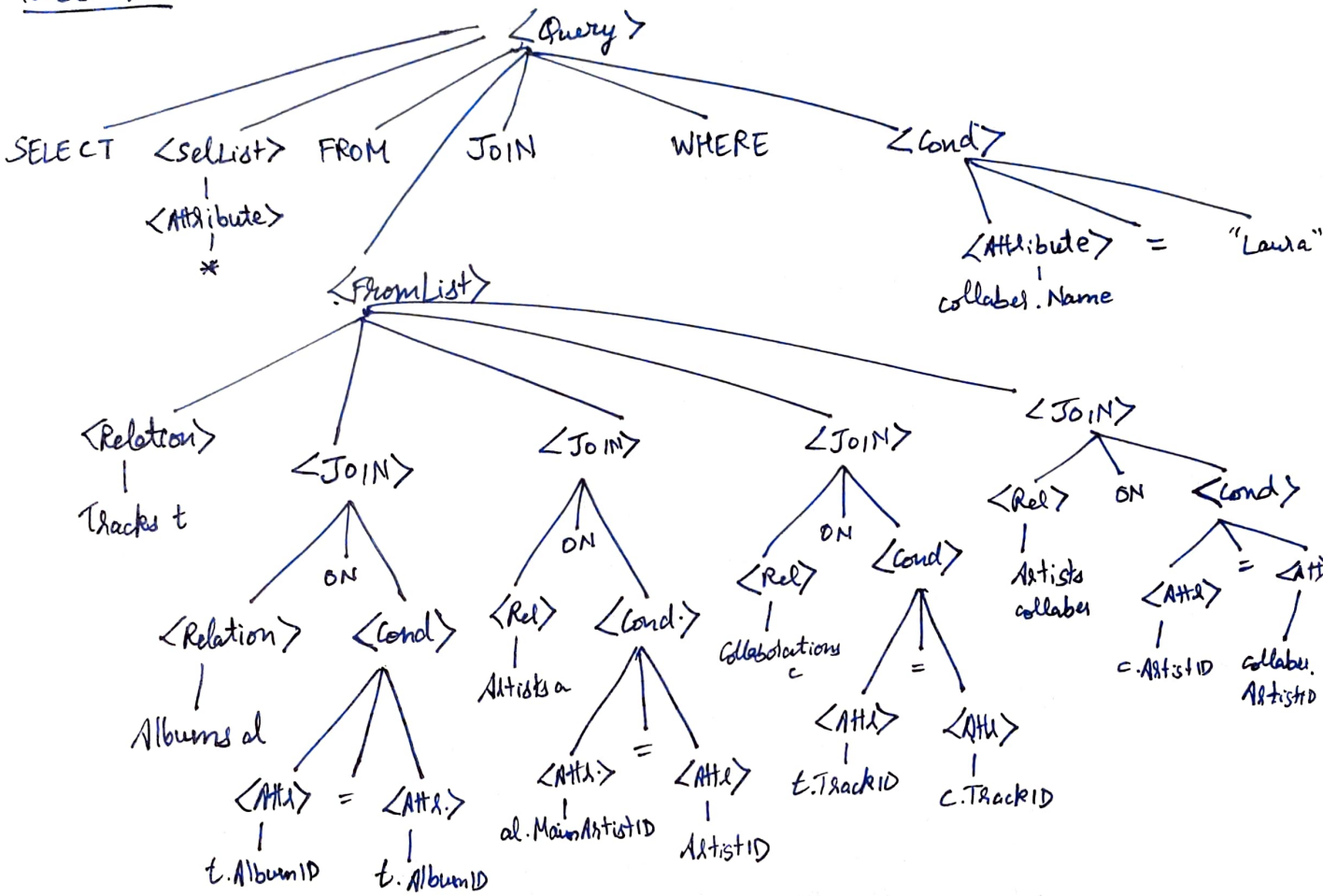
M·C· Krishna Kumar ①
PES2UG22CS281
E Section

9.

Query:

```
SELECT *
FROM Tracks t
JOIN Albums al ON t.AlbumID = al.AlbumID
JOIN Artists a ON al.MainArtistID = a.ArtistID
JOIN Collaborations c ON t.TrackID = c.TrackID
JOIN Artists collabor ON c.ArtistID = collabor.ArtistID
WHERE collabor.Name = "Laura";
```

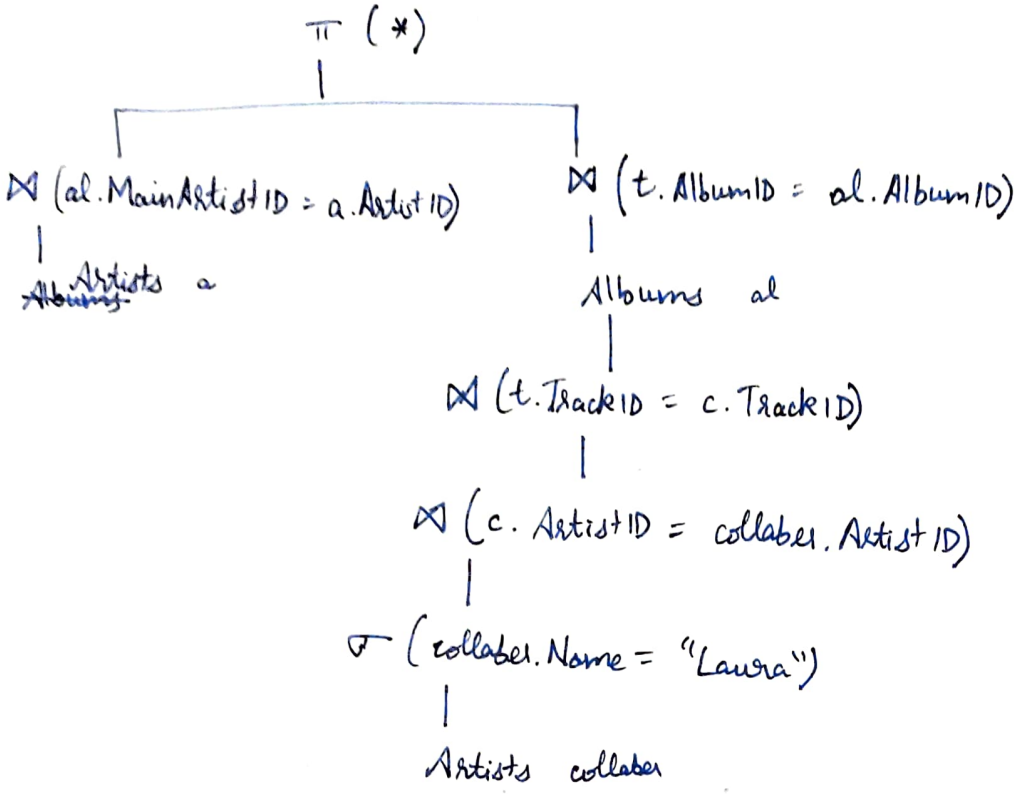
Parse Tree:



Relational Algebra:

$$\Pi_{*}\Big(\Big(\Big(\Big(\sigma_{collabor.Name = \text{"Laura"}}(Artists) \bowtie_{collabor.ArtistID = c.ArtistID} Collaborations\Big) \bowtie_{c.TrackID = t.TrackID} Tracks\Big) \bowtie_{t.AlbumID = al.AlbumID} Albums\Big) \bowtie_{al.MainArtistID = a.ArtistID} Artists\Big)\Big)$$

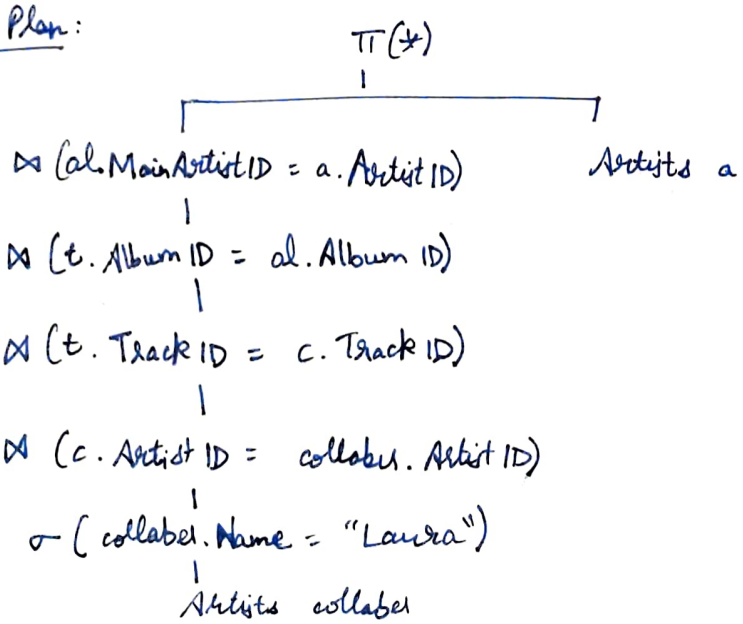
Query Plan Tree:



→ Optimization: Join Artists table first so that early filtering can happen which reduces number of rows to be processed in the subsequent joins.

```
Query:  SELECT *
        FROM   Artists    collabel
        JOIN   Collabdrations c   ON   collabel.ArtistID = c.ArtistID
        JOIN   Tracks     t   ON   c.TrackID = t.TrackID
        JOIN   Albums     al  ON   t.AlbumID = al.AlbumID
        JOIN   Artists    a   ON   al.MainArtistID = a.ArtistID
        WHERE  collabel.Name = "Laura";
```

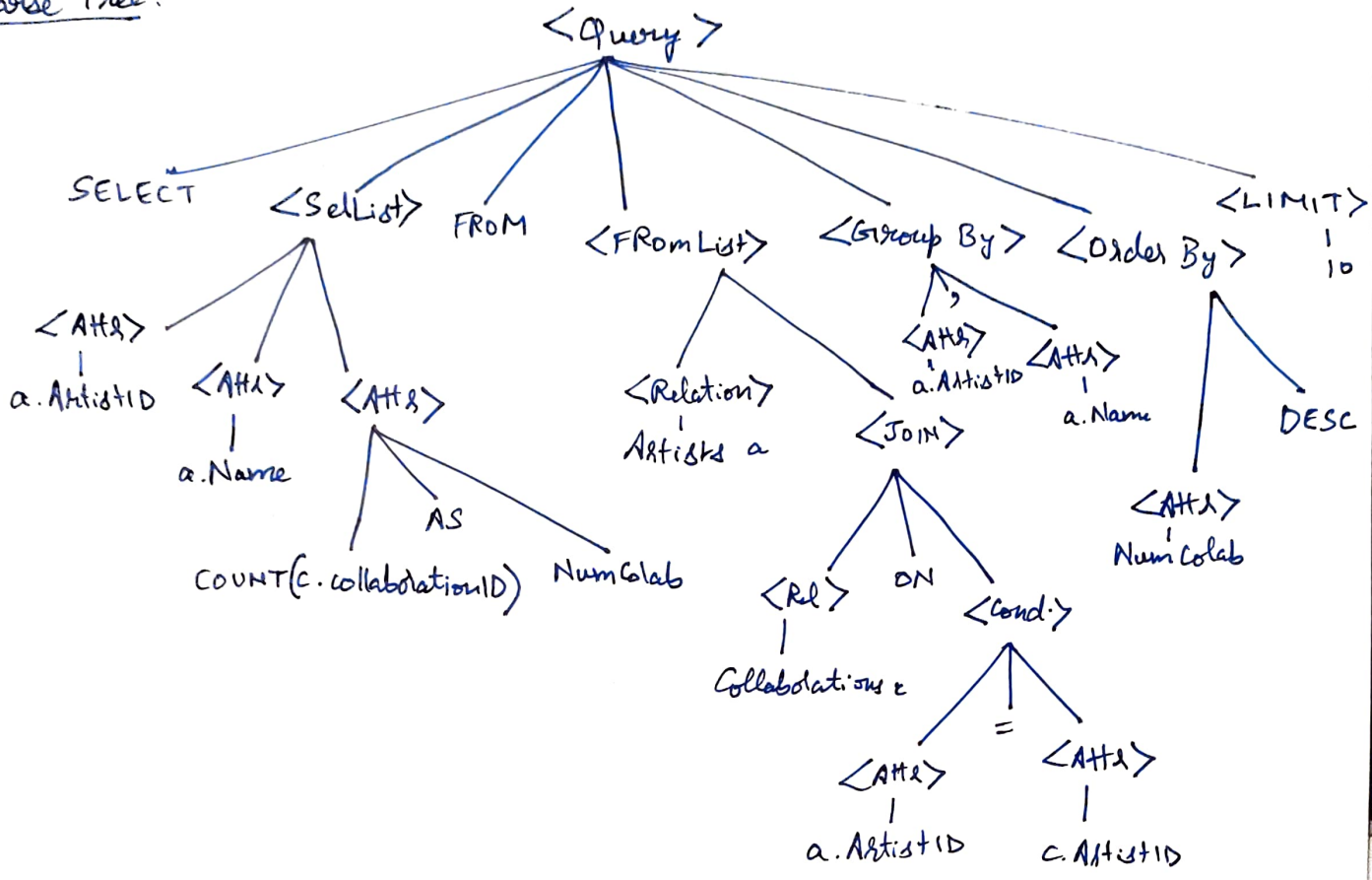
Query Plan:



92. Query: (3)

```
SELECT  a.ArtistID, a.Name, COUNT(c.CollaborationID) AS NumColab
FROM    Artists a
JOIN    Collaborations c  ON  a.ArtistID = c.ArtistID
GROUP BY  a.ArtistID, a.Name
ORDER  BY  NumColab  DESC
LIMIT  10;
```

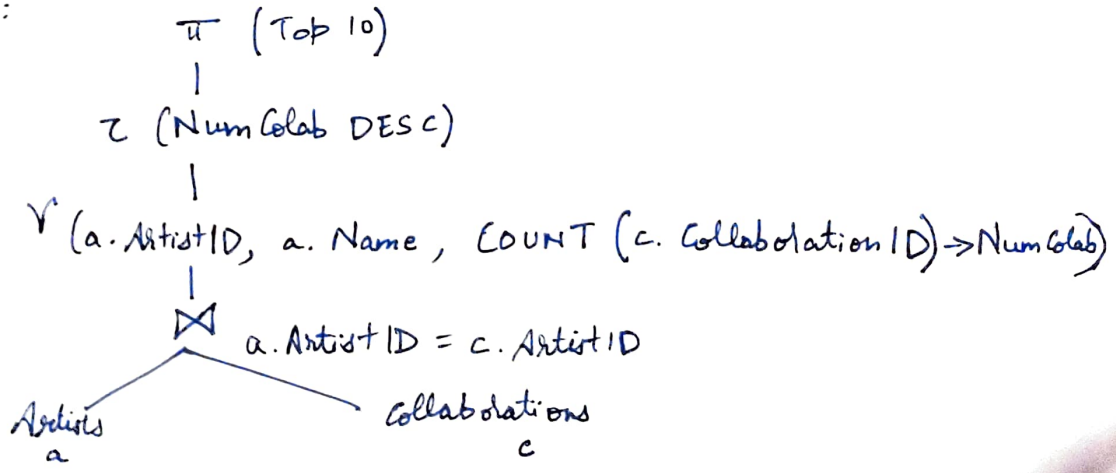
Parse Tree:



Relational Algebra:

$$\tau_{[NumColab\ DESC[10]]}\Big(\gamma_{(a.ArtistID,\ a.Name,\ COUNT(c.CollaborationID)\rightarrow NumColab)}\big(Artists\ a \bowtie_{a.ArtistID = c.ArtistID} Collaborations\ c\big)\Big)$$

Query Plan Tree:

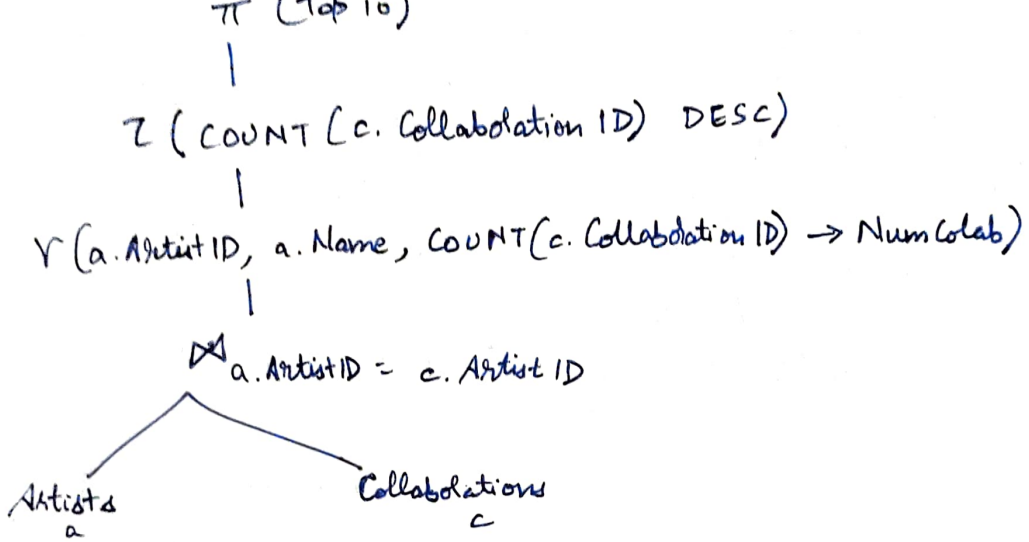


→ Optimization : Pushing the aggregate function COUNT to ORDER By Clause so that lesser number of tuples are selected for sorting

Query :

```
SELECT a.ArtistID, a.Name, COUNT(c.CollaborationID)
FROM Artists a                                    AS NumColab
JOIN Collaborations c ON a.ArtistID = c.ArtistID
GROUP BY a.ArtistID, a.Name
ORDER BY COUNT(c.CollaborationID) DESC
LIMIT 10;
```

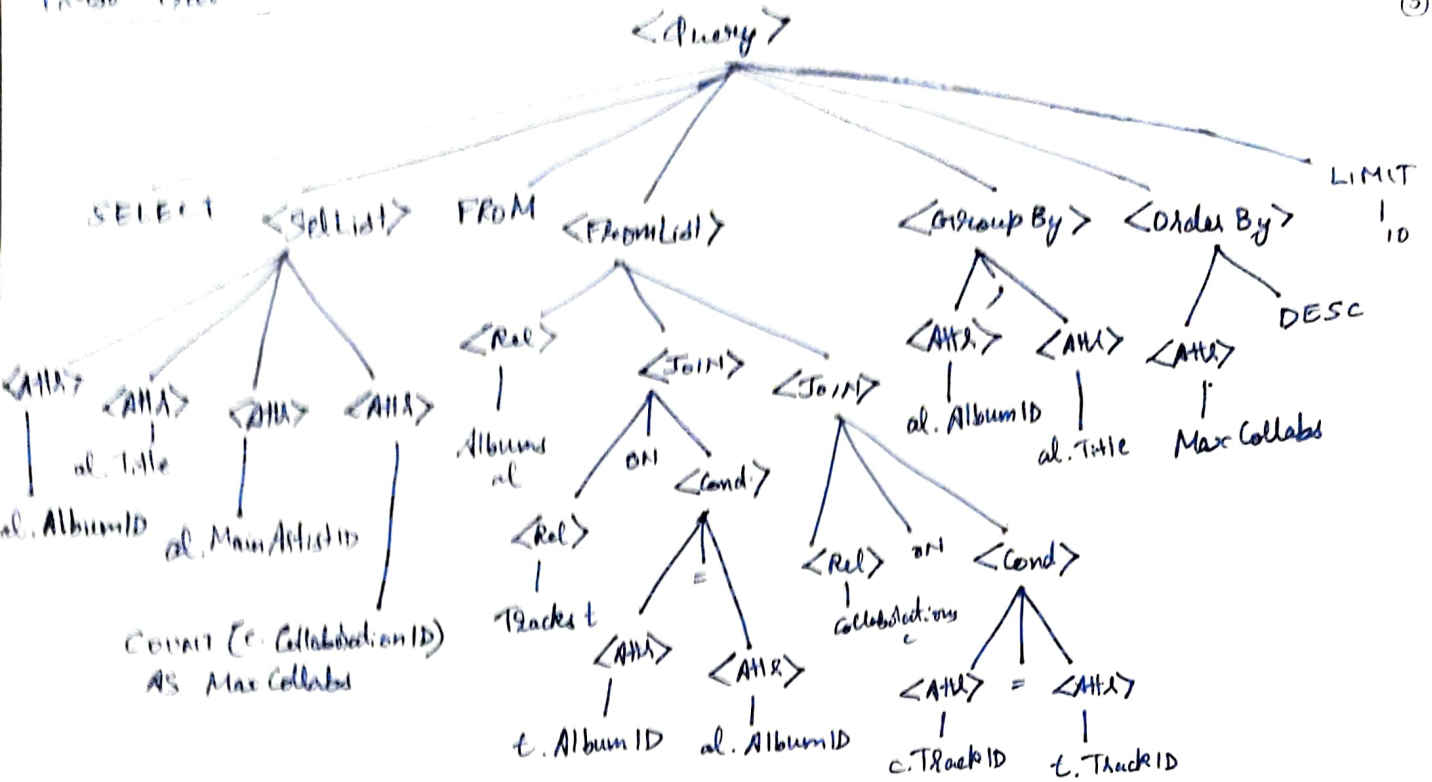
Query Plan :



Q3.

```
SELECT al.AlbumID, al.Title, al.MainArtistID, COUNT(c.CollaborationID)
                                                AS MaxCollabs
FROM Albums al
JOIN Tracks t ON t.AlbumID = al.AlbumID
JOIN Collaborations c ON c.TrackID = t.TrackID
GROUP BY al.AlbumID, al.Title
ORDER BY MaxCollabs DESC
LIMIT 10;
```

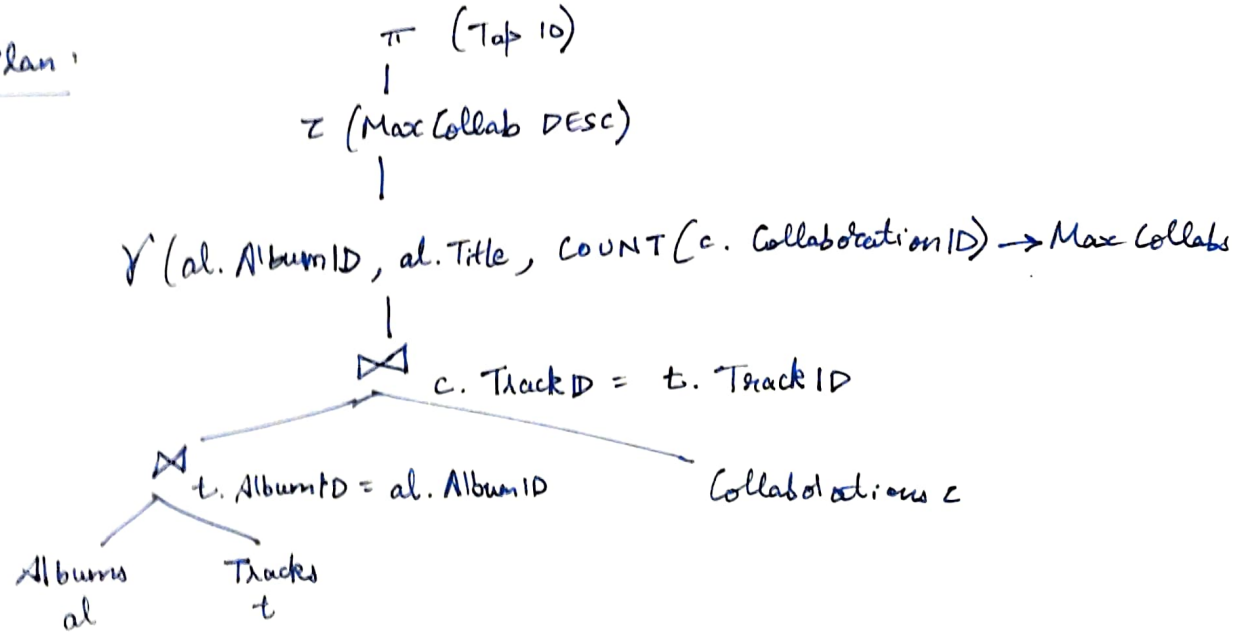
Parse Tree:



Relation Algebra:

$$\pi_{\text{Top 10}}\left(\tau_{\text{MaxCollabs DESC}}\left(\gamma_{\text{AlbumID, Title, MainArtistID, COUNT(CollaborationID)} \rightarrow \text{MaxCollabs}}\left(\left(\text{Albums} \bowtie_{\text{Albums.AlbumID} = \text{Tracks.AlbumID}} \text{Tracks}\right) \bowtie_{\text{Tracks.TrackID} = \text{Collaborations.TrackID}} \text{Collaborations}\right)\right)\right)$$

Query Plan:



→ Optimization: Using a Derived Table
Instead of joining all albums and track first we shall find the collaboration counts first and then join

Query:

```
SELECT al.AlbumID, al.Title, al.MainArtistID, MaxCollabs
FROM Albums al
JOIN (
    SELECT t.AlbumID, COUNT(c.CollaborationID) AS MaxCollabs
    FROM Tracks t
    JOIN Collaborations c ON c.TrackID = t.TrackID
    GROUP BY t.AlbumID
) AS CollabCounts ON al.AlbumID = collabCounts.AlbumID
ORDER BY MaxCollabs DESC
LIMIT 10;
```

Query Tree: