Changes for Stage 2:

All foreign keys were removed in the UML diagram to reduce unnecessary clutter.

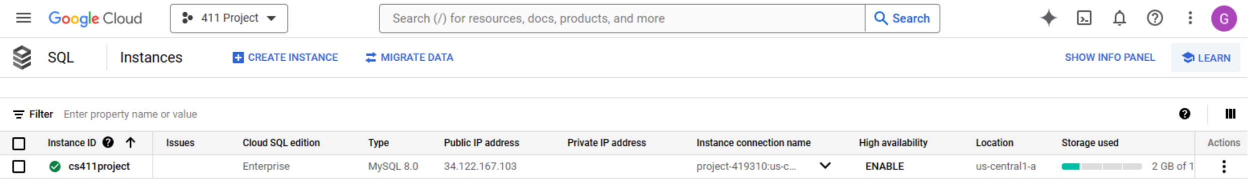
Unnecessary primary keys were removed as uniquely identifying many-many relationships can be done by foreign keys.
Ex. MovieActors can be identified by MovieID and ActorID FKs, so no need for MovieActorID PK

Reduced 8 entities down to 5 after converting DirectorPreferences, ActorPreferences, and WatchHistory into many-to-many relationships. This better reflects the purposes of these tables, which is to link users with directors, actors, and movies respectively.

Stage 3:

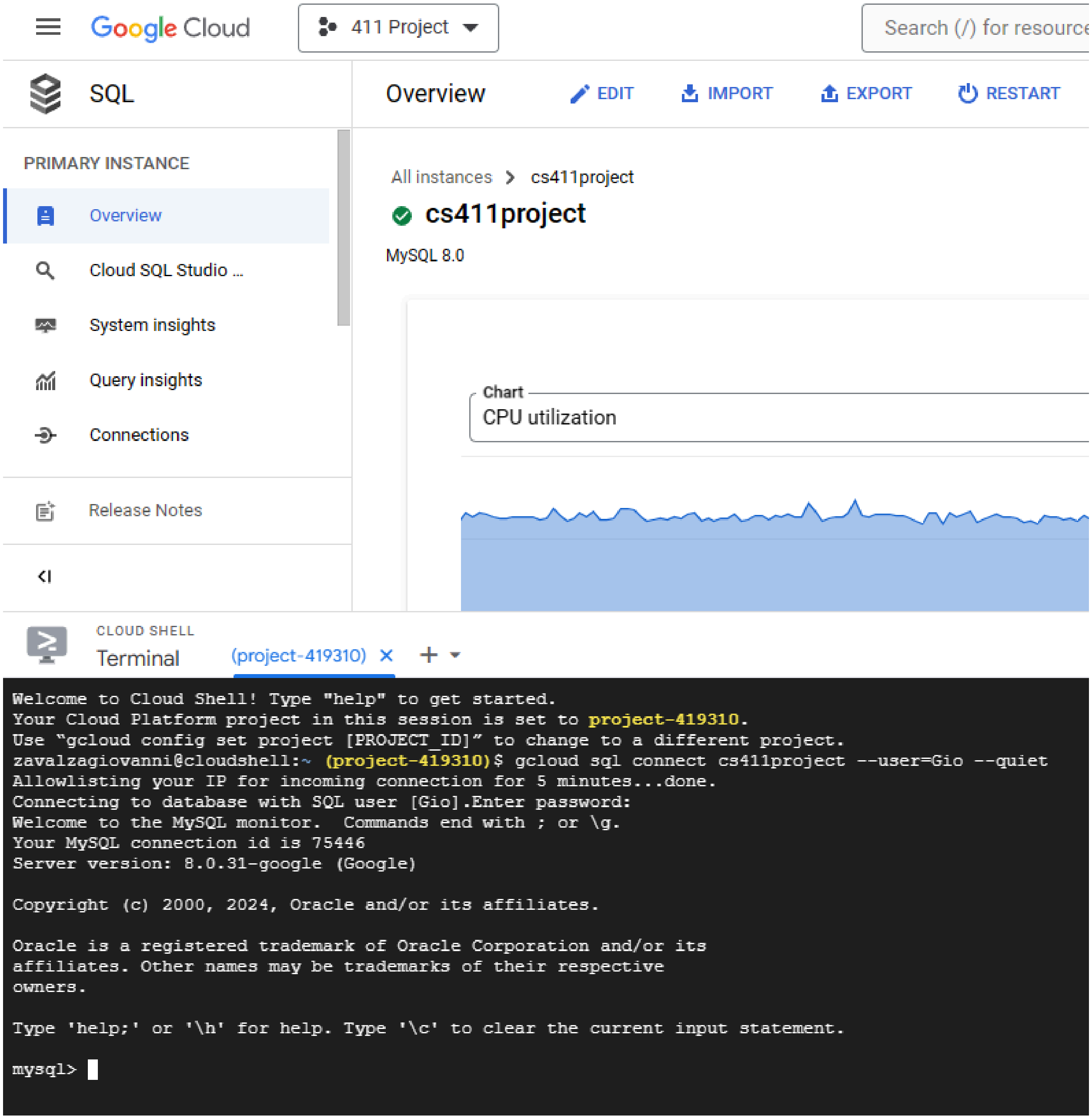
1.1:

GCP implementation:



Instance ID	Issues	Cloud SQL edition	Type	Public IP address	Private IP address	Instance connection name	High availability	Location	Storage used	Actions
cs411project		Enterprise	MySQL 8.0	34.122.167.103		project-419310:us-c...	ENABLE	us-central1-a	2 GB of 1	⋮

Terminal running on Cloud shell:



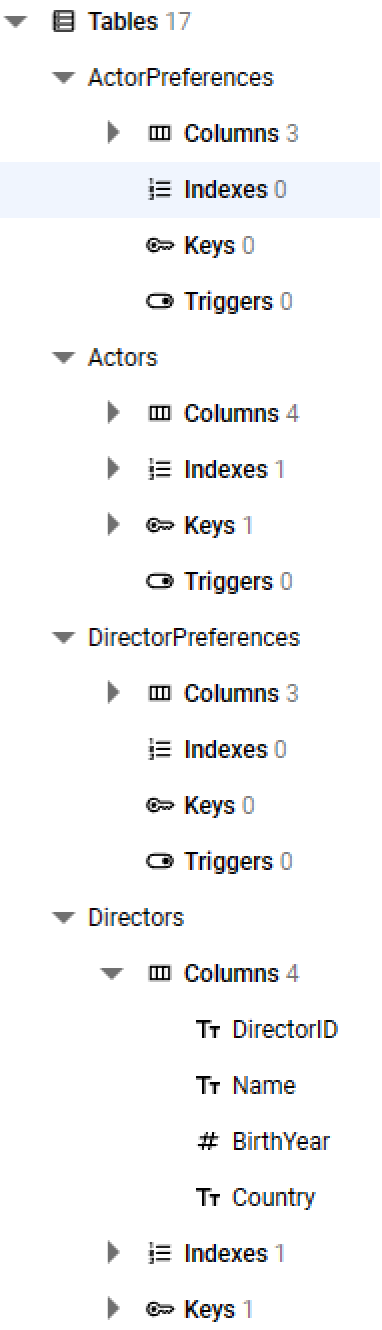
```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to project-419310.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
zavalzagiovanni@cloudshell:~ (project-419310)$ gcloud sql connect cs411project --user=Gio --quiet
Allowlisting your IP for incoming connection for 5 minutes...done.
Connecting to database with SQL user [Gio].Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 75446
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> 
```

1.2:

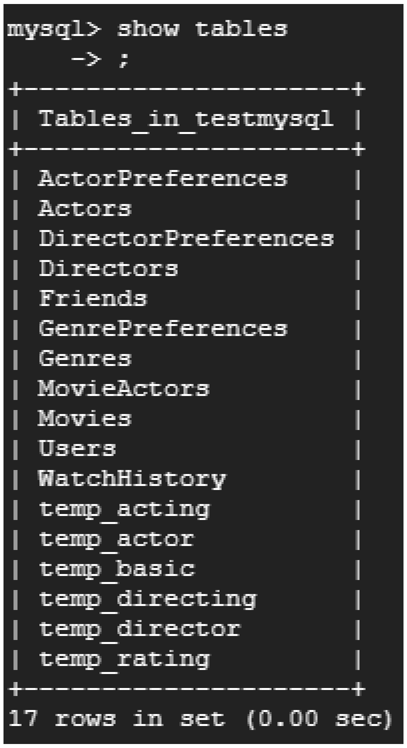
After setting up in GCP, the following was used to create tables for our database:

DDL commands used -
CREATE TABLE Users(

```
    UserID INT PRIMARY KEY,
    Username VARCHAR(40),
    PW VARCHAR(255),
    Email VARCHAR(255)
  );

CREATE TABLE Friends(
    User1ID INT REFERENCES Users(UserID),
    User2ID INT REFERENCES Users(UserID)
  );

CREATE TABLE Actors(
  ActorID VARCHAR(40) PRIMARY KEY,
  Name VARCHAR(255),
  BirthYear YEAR,
  Country VARCHAR(50)
);

CREATE TABLE MovieActors(
  MovieID VARCHAR(40) REFERENCES Movies(MovieID),
  ActorID VARCHAR(40) REFERENCES Actors(ActorID),
  Role VARCHAR(50)
);

CREATE TABLE Movies(
  MovieID VARCHAR(40) PRIMARY KEY,
  DirectorID VARCHAR(40) REFERENCES Directors(DirectorID),
  GenreID INT REFERENCES Genres(GenreID),
  Rating INT,
  Title VARCHAR(100),
  Year YEAR,
  Region VARCHAR(50)
);

CREATE TABLE WatchHistory(
  UserID INT REFERENCES Users(UserID),
  MovieID VARCHAR(40) REFERENCES Movies(MovieID),
  DateWatched VARCHAR(40),
  UserRating INT
);

CREATE TABLE Genres(
  GenreID INT PRIMARY KEY,
  Name VARCHAR(50),
Description VARCHAR(255)
);

CREATE TABLE GenrePreferences(
  UserID INT REFERENCES Users(UserID),
  GenreID INT REFERENCES Actors(ActorID),
  Rating INT
);

CREATE TABLE ActorPreferences(
  UserID INT REFERENCES Users(UserID),
  ActorID VARCHAR(40) REFERENCES Actors(ActorID),
  Rating INT
);

CREATE TABLE Directors(
  DirectorID VARCHAR(40) PRIMARY KEY,
  Name VARCHAR(100),
  BirthYear INT,
  Country VARCHAR(50)
);

CREATE TABLE DirectorPreferences(
  UserID INT REFERENCES Users(UserID),
  DirectorID VARCHAR(40) REFERENCES Directors(DirectorID),
  Rating INT
);
```



- Tables 17
  - ActorPreferences
    - Columns 3
    - Indexes 0
    - Keys 0
    - Triggers 0
  - Actors
    - Columns 4
    - Indexes 1
    - Keys 1
    - Triggers 0
  - DirectorPreferences
    - Columns 3
    - Indexes 0
    - Keys 0
    - Triggers 0
  - Directors
    - Columns 4
      - Tт DirectorID
      - Tт Name
      - # BirthYear
      - Tт Country
    - Indexes 1
    - Keys 1



```
mysql> show tables
    -> ;
+----------------------+
| Tables_in_testmysql |
+----------------------+
| ActorPreferences     |
| Actors               |
| DirectorPreferences |
| Directors            |
| Friends              |
| GenrePreferences     |
| Genres               |
| MovieActors          |
| Movies               |
| Users                |
| WatchHistory         |
| temp_acting          |
| temp_actor           |
| temp_basic           |
| temp_directing       |
| temp_director        |
| temp_rating          |
+----------------------+
17 rows in set (0.00 sec)
```

Here is showing the output for each table and some columns/attributes in the GCP. The temp tables were used to generate and inject data into the permanent tables.

1.3:

For Users, Friends, ActorPreferences, DirectorPreferences, and WatchHistory, python scripts were used to generate 1000 unique users who each had a unique set of friends, preferences, and movies watched. This had to be generated so we can test user-specific queries in step 1.4.

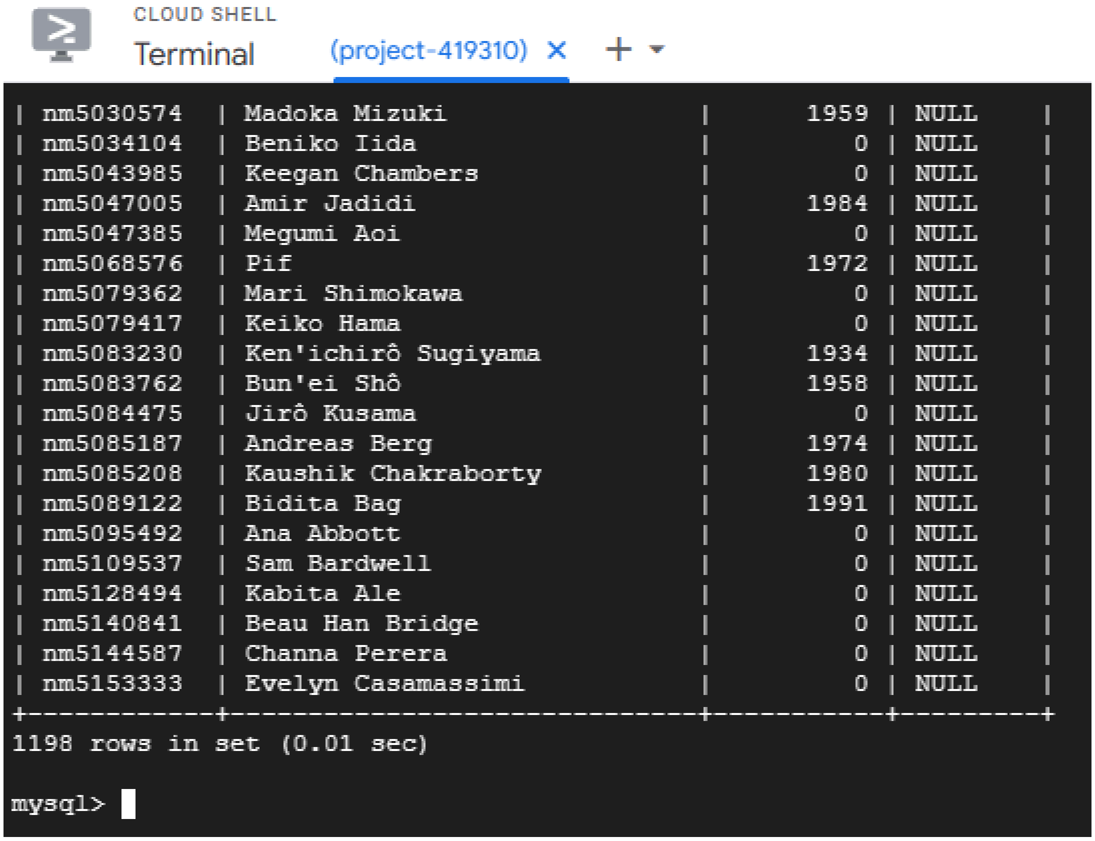
Use linux command to generate CSV table from TSV table using last 1000 movies and all director and acting data.

```
 grep -E "actor|actress" title.principals.tsv | awk 'BEGIN {FS="\t"; OFS=","} {print $1, $2, $3}' > actor.csv
grep -E "director" title.principals.tsv | awk 'BEGIN {FS="\t"; OFS=","} {print $1, $2, $3}' > director.csv
```

Upload CSV into temp_table;
And select only relevant actor and director into formal table.

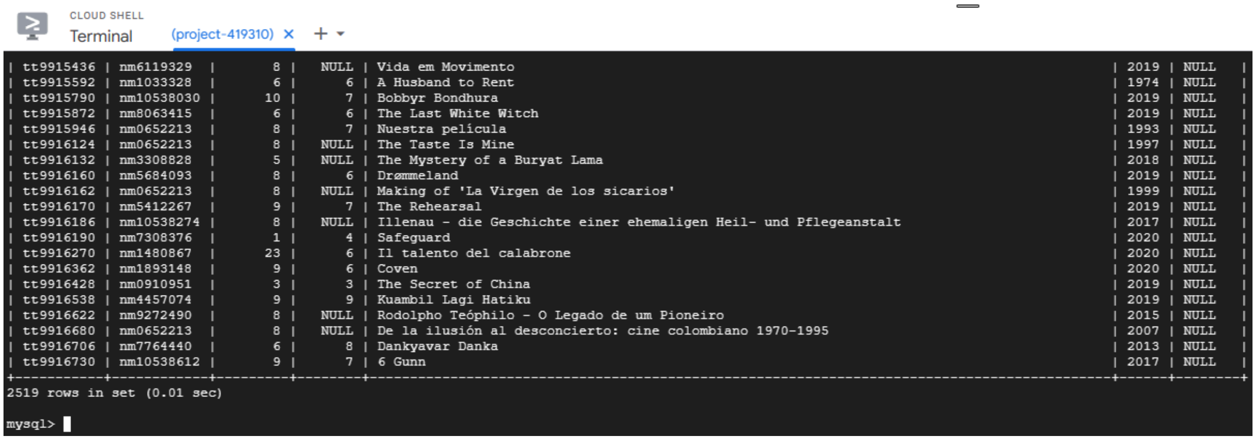
```
INSERT INTO Directors (DirectorID, Name, BirthYear)
SELECT DISTINCT  b.nconst AS DirectorID,    d.Name,    d.BirthYear
FROM temp_basic b LEFT JOIN  temp_director d ON b.nconst = d.nconst;
```

Actors entity with 1198 entries:

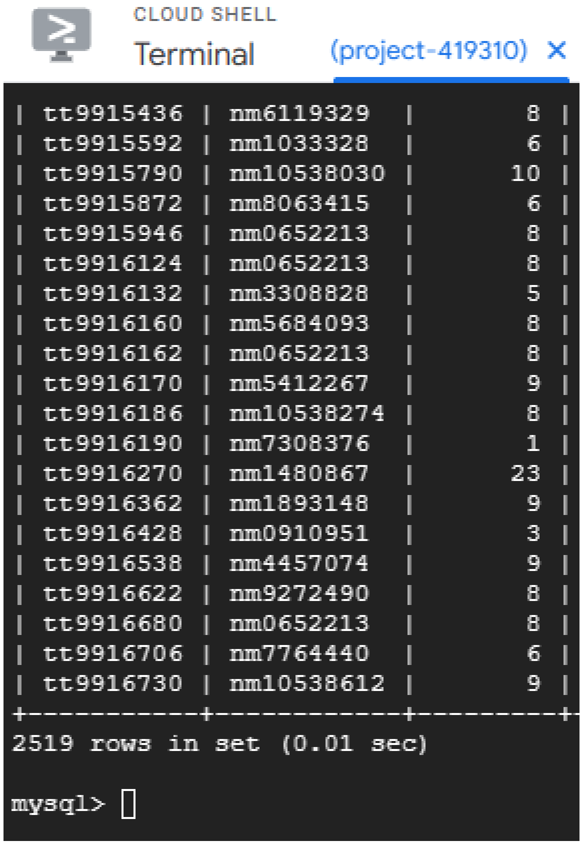


```
| nm5030574  | Madoka Mizuki                |       1959 | NULL     |
| nm5034104  | Beniko Iida                  |          0 | NULL     |
| nm5043985  | Keegan Chambers              |          0 | NULL     |
| nm5047005  | Amir Jadidi                  |       1984 | NULL     |
| nm5047385  | Megumi Aoi                   |          0 | NULL     |
| nm5068576  | Pif                          |       1972 | NULL     |
| nm5079362  | Mari Shimokawa               |          0 | NULL     |
| nm5079417  | Keiko Hama                   |          0 | NULL     |
| nm5083230  | Ken'ichirô Sugiyama          |       1934 | NULL     |
| nm5083762  | Bun'ei Shô                   |       1958 | NULL     |
| nm5084475  | Jirô Kusama                  |          0 | NULL     |
| nm5085187  | Andreas Berg                 |       1974 | NULL     |
| nm5085208  | Kaushik Chakraborty          |       1980 | NULL     |
| nm5089122  | Bidita Bag                   |       1991 | NULL     |
| nm5095492  | Ana Abbott                   |          0 | NULL     |
| nm5109537  | Sam Bardwell                 |          0 | NULL     |
| nm5128494  | Kabita Ale                   |          0 | NULL     |
| nm5140841  | Beau Han Bridge              |          0 | NULL     |
| nm5144587  | Channa Perera                |          0 | NULL     |
| nm5153333  | Evelyn Casamassimi           |          0 | NULL     |
+------------+------------------------------+------------+----------+
1198 rows in set (0.01 sec)

mysql> 
```

Movies entity with 2519 entries:

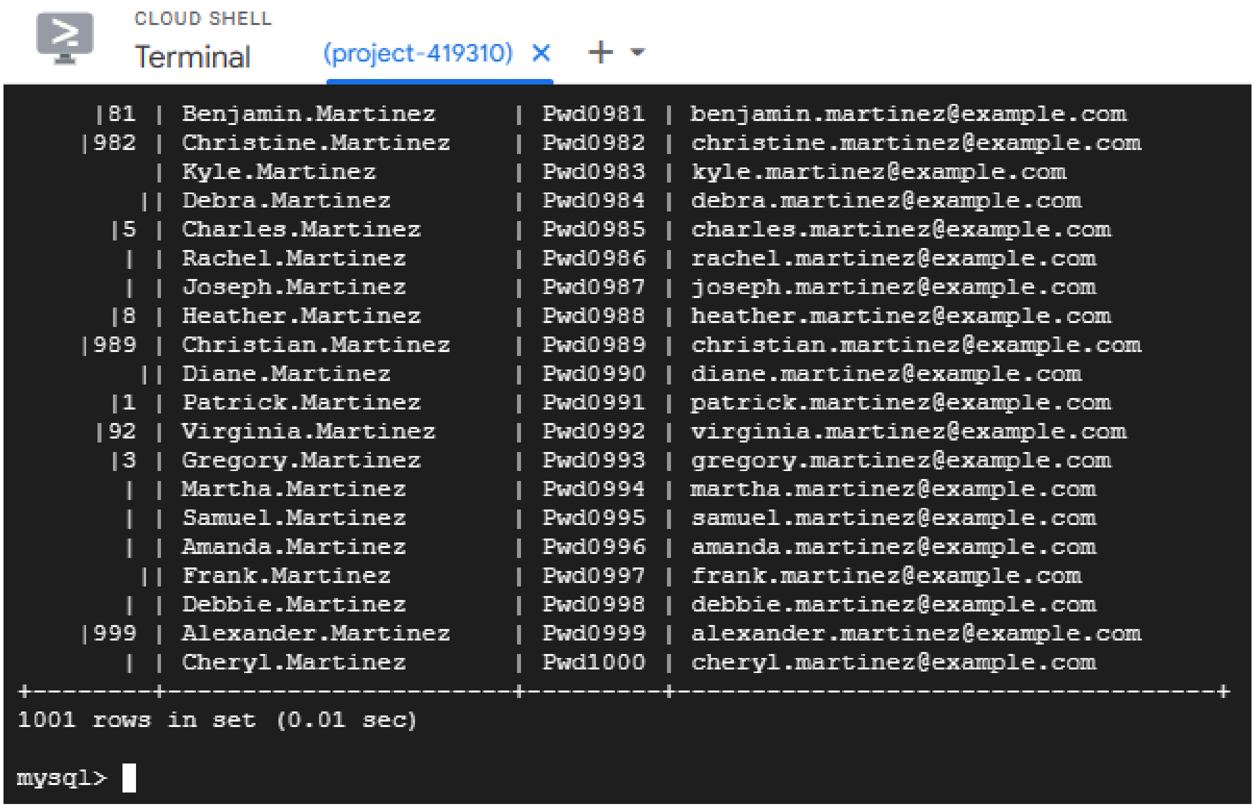


```
| tt9915436 | nm6119329  |        8 |    NULL | Vida em Movimento                                                                         | 2019 | NULL    |
| tt9915592 | nm1033328  |        6 |       6 | A Husband to Rent                                                                         | 1974 | NULL    |
| tt9915790 | nm10538030 |       10 |       7 | Bobbyr Bondhura                                                                           | 2019 | NULL    |
| tt9915872 | nm8063415  |        6 |       6 | The Last White Witch                                                                      | 2019 | NULL    |
| tt9915946 | nm0652213  |        8 |       7 | Nuestra película                                                                          | 1993 | NULL    |
| tt9916124 | nm0652213  |        8 |    NULL | The Taste Is Mine                                                                         | 1997 | NULL    |
| tt9916132 | nm3308828  |        5 |    NULL | The Mystery of a Buryat Lama                                                              | 2018 | NULL    |
| tt9916160 | nm5684093  |        8 |       6 | Drømmeland                                                                                | 2019 | NULL    |
| tt9916162 | nm0652213  |        8 |    NULL | Making of 'La Virgen de los sicarios'                                                     | 1999 | NULL    |
| tt9916170 | nm5412267  |        9 |       7 | The Rehearsal                                                                             | 2019 | NULL    |
| tt9916186 | nm10538274 |        8 |    NULL | Illenau - die Geschichte einer ehemaligen Heil- und Pflegeanstalt                         | 2017 | NULL    |
| tt9916190 | nm7308376  |        1 |       4 | Safeguard                                                                                 | 2020 | NULL    |
| tt9916270 | nm1480867  |       23 |       6 | Il talento del calabrone                                                                  | 2020 | NULL    |
| tt9916362 | nm1893148  |        9 |       6 | Coven                                                                                     | 2020 | NULL    |
| tt9916428 | nm0910951  |        3 |       3 | The Secret of China                                                                       | 2019 | NULL    |
| tt9916538 | nm4457074  |        9 |       9 | Kuambil Lagi Hatiku                                                                       | 2019 | NULL    |
| tt9916622 | nm9272490  |        8 |    NULL | Rodolpho Teóphilo - O Legado de um Pioneiro                                               | 2015 | NULL    |
| tt9916680 | nm0652213  |        8 |    NULL | De la ilusión al desconcierto: cine colombiano 1970-1995                                  | 2007 | NULL    |
| tt9916706 | nm7764440  |        6 |       8 | Dankyavar Danka                                                                           | 2013 | NULL    |
| tt9916730 | nm10538612 |        9 |       7 | 6 Gunn                                                                                    | 2017 | NULL    |
+-----------+------------+----------+---------+-------------------------------------------------------------------------------------------+------+---------+
2519 rows in set (0.01 sec)

mysql> 
```



```
| tt9915436 | nm6119329  |        8 |
| tt9915592 | nm1033328  |        6 |
| tt9915790 | nm10538030 |       10 |
| tt9915872 | nm8063415  |        6 |
| tt9915946 | nm0652213  |        8 |
| tt9916124 | nm0652213  |        8 |
| tt9916132 | nm3308828  |        5 |
| tt9916160 | nm5684093  |        8 |
| tt9916162 | nm0652213  |        8 |
| tt9916170 | nm5412267  |        9 |
| tt9916186 | nm10538274 |        8 |
| tt9916190 | nm7308376  |        1 |
| tt9916270 | nm1480867  |       23 |
| tt9916362 | nm1893148  |        9 |
| tt9916428 | nm0910951  |        3 |
| tt9916538 | nm4457074  |        9 |
| tt9916622 | nm9272490  |        8 |
| tt9916680 | nm0652213  |        8 |
| tt9916706 | nm7764440  |        6 |
| tt9916730 | nm10538612 |        9 |
+-----------+------------+---------+-
2519 rows in set (0.01 sec)

mysql> 
```

Users entity with 1001 entries:

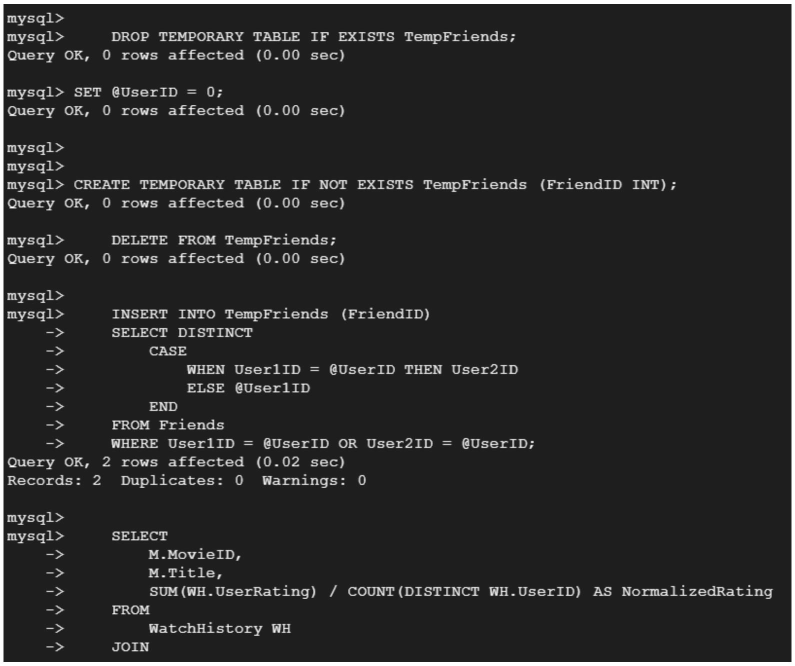


```
     |81 | Benjamin.Martinez      | Pwd0981 | benjamin.martinez@example.com
    |982 | Christine.Martinez     | Pwd0982 | christine.martinez@example.com
         | Kyle.Martinez          | Pwd0983 | kyle.martinez@example.com
        || Debra.Martinez         | Pwd0984 | debra.martinez@example.com
      |5 | Charles.Martinez       | Pwd0985 | charles.martinez@example.com
       | | Rachel.Martinez        | Pwd0986 | rachel.martinez@example.com
       | | Joseph.Martinez        | Pwd0987 | joseph.martinez@example.com
      |8 | Heather.Martinez       | Pwd0988 | heather.martinez@example.com
    |989 | Christian.Martinez     | Pwd0989 | christian.martinez@example.com
        || Diane.Martinez         | Pwd0990 | diane.martinez@example.com
      |1 | Patrick.Martinez       | Pwd0991 | patrick.martinez@example.com
     |92 | Virginia.Martinez      | Pwd0992 | virginia.martinez@example.com
      |3 | Gregory.Martinez       | Pwd0993 | gregory.martinez@example.com
       | | Martha.Martinez        | Pwd0994 | martha.martinez@example.com
       | | Samuel.Martinez        | Pwd0995 | samuel.martinez@example.com
       | | Amanda.Martinez        | Pwd0996 | amanda.martinez@example.com
        || Frank.Martinez         | Pwd0997 | frank.martinez@example.com
       | | Debbie.Martinez        | Pwd0998 | debbie.martinez@example.com
    |999 | Alexander.Martinez     | Pwd0999 | alexander.martinez@example.com
       | | Cheryl.Martinez        | Pwd1000 | cheryl.martinez@example.com
+--------+-----------------------+---------+-------------------------------------+
1001 rows in set (0.01 sec)

mysql> 
```

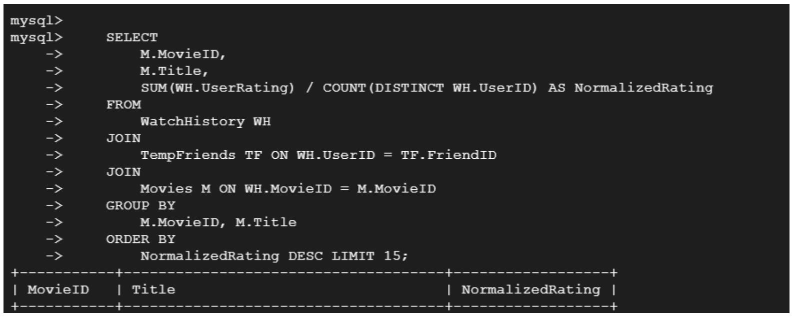
1.4 Advanced Queries:

Advanced Query “GetFriendsRatedMovies”:
This query returns the table of movies with normalized ratings amongst all of a user’s friends. The movies with the highest ratings amongst friends will be the first movies listed in the table. This can be used for a movie recommendation system based on what the user's friends watch. The SQL concepts used are the JOIN of multiple relations and GROUP BY aggregation.

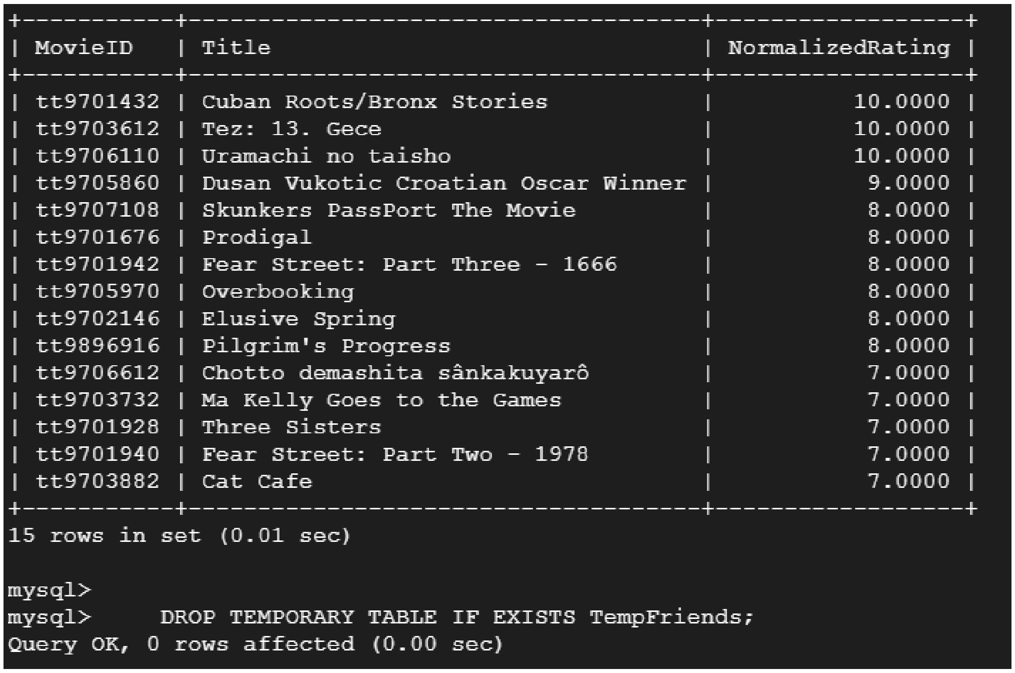
```
SET @UserID = 0;


CREATE TEMPORARY TABLE IF NOT EXISTS TempFriends (FriendID INT);
    DELETE FROM TempFriends;

    INSERT INTO TempFriends (FriendID)
    SELECT DISTINCT
        CASE
            WHEN User1ID = @UserID THEN User2ID
            ELSE @User1ID
        END
    FROM Friends
    WHERE User1ID = @UserID OR User2ID = @UserID;

    SELECT
        M.MovieID,
        M.Title,
        SUM(WH.UserRating) / COUNT(DISTINCT WH.UserID) AS NormalizedRating
    FROM
        WatchHistory WH
    JOIN
        TempFriends TF ON WH.UserID = TF.FriendID
    JOIN
        Movies M ON WH.MovieID = M.MovieID
    GROUP BY
        M.MovieID, M.Title
    ORDER BY
        NormalizedRating DESC;

    DROP TEMPORARY TABLE IF EXISTS TempFriends;
```



```
mysql>
mysql>     DROP TEMPORARY TABLE IF EXISTS TempFriends;
Query OK, 0 rows affected (0.00 sec)

mysql> SET @UserID = 0;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql>
mysql> CREATE TEMPORARY TABLE IF NOT EXISTS TempFriends (FriendID INT);
Query OK, 0 rows affected (0.00 sec)

mysql>     DELETE FROM TempFriends;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql>     INSERT INTO TempFriends (FriendID)
    ->     SELECT DISTINCT
    ->         CASE
    ->             WHEN User1ID = @UserID THEN User2ID
    ->             ELSE @User1ID
    ->         END
    ->     FROM Friends
    ->     WHERE User1ID = @UserID OR User2ID = @UserID;
Query OK, 2 rows affected (0.02 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql>
mysql>     SELECT
    ->         M.MovieID,
    ->         M.Title,
    ->         SUM(WH.UserRating) / COUNT(DISTINCT WH.UserID) AS NormalizedRating
    ->     FROM
    ->         WatchHistory WH
    ->     JOIN
```



```
mysql>
mysql>     SELECT
    ->         M.MovieID,
    ->         M.Title,
    ->         SUM(WH.UserRating) / COUNT(DISTINCT WH.UserID) AS NormalizedRating
    ->     FROM
    ->         WatchHistory WH
    ->     JOIN
    ->         TempFriends TF ON WH.UserID = TF.FriendID
    ->     JOIN
    ->         Movies M ON WH.MovieID = M.MovieID
    ->     GROUP BY
    ->         M.MovieID, M.Title
    ->     ORDER BY
    ->         NormalizedRating DESC LIMIT 15;
+-----------+-------------------------------------+------------------+
| MovieID   | Title                               | NormalizedRating |
+-----------+-------------------------------------+------------------+
```



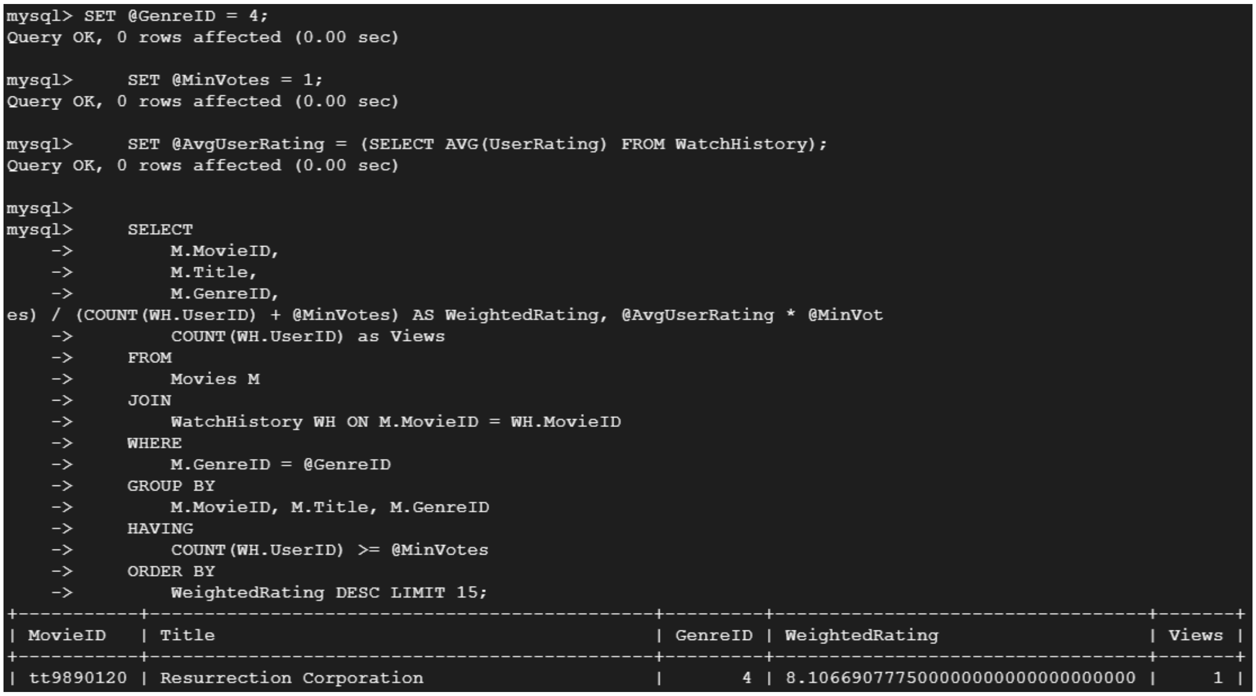
```
+-----------+-------------------------------------+------------------+
| MovieID   | Title                               | NormalizedRating |
+-----------+-------------------------------------+------------------+
| tt9701432 | Cuban Roots/Bronx Stories           |          10.0000 |
| tt9703612 | Tez: 13. Gece                       |          10.0000 |
| tt9706110 | Uramachi no taisho                  |          10.0000 |
| tt9705860 | Dusan Vukotic Croatian Oscar Winner |           9.0000 |
| tt9707108 | Skunkers PassPort The Movie         |           8.0000 |
| tt9701676 | Prodigal                            |           8.0000 |
| tt9701942 | Fear Street: Part Three - 1666      |           8.0000 |
| tt9705970 | Overbooking                         |           8.0000 |
| tt9702146 | Elusive Spring                      |           8.0000 |
| tt9896916 | Pilgrim's Progress                  |           8.0000 |
| tt9706612 | Chotto demashita sânkakuyarô        |           7.0000 |
| tt9703732 | Ma Kelly Goes to the Games          |           7.0000 |
| tt9701928 | Three Sisters                       |           7.0000 |
| tt9701940 | Fear Street: Part Two - 1978        |           7.0000 |
| tt9703882 | Cat Cafe                            |           7.0000 |
+-----------+-------------------------------------+------------------+
15 rows in set (0.01 sec)

mysql>
mysql>     DROP TEMPORARY TABLE IF EXISTS TempFriends;
Query OK, 0 rows affected (0.00 sec)
```

Advanced Query “GetBestMoviesByGenre”:

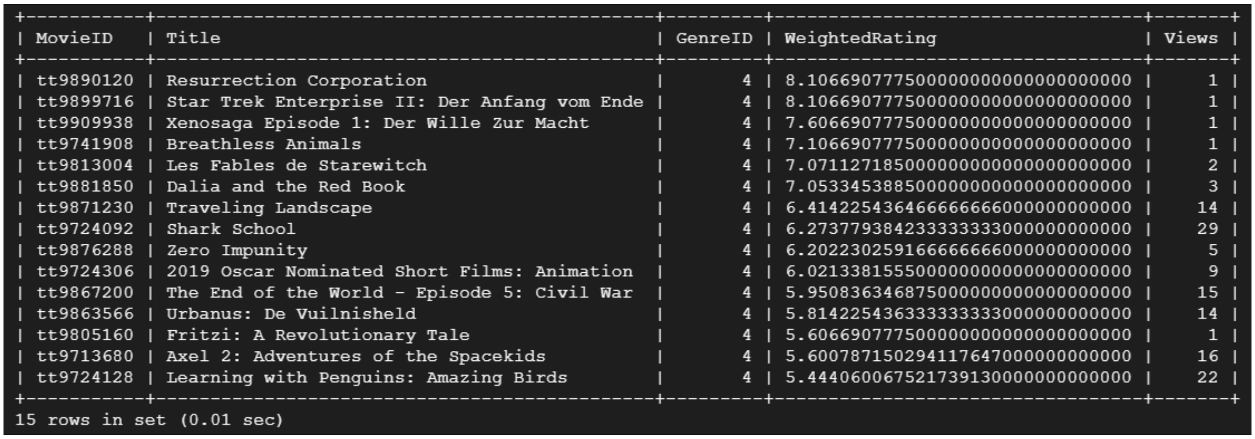
This query returns the highest rated movies within a genre based on a score system that combines the average rating of the movie by all users and the number of times the movie has been viewed. Increasing the minimum number of votes (aka. the number of users that have rated the movie) to filter the final table will produce a list of movies with ratings that are less biased.

The SQL concepts used are the JOIN of multiple relations, SET Operators, GROUP BY aggregation, and subqueries.

```
  SET @GenreID = 4;
  SET @MinVotes = 1;
  SET @AvgUserRating = (SELECT AVG(UserRating) FROM WatchHistory);

  SELECT
    M.MovieID,
    M.Title,
    M.GenreID,
    (AVG(WH.UserRating) * COUNT(WH.UserID) + @AvgUserRating * @MinVotes) /
(COUNT(WH.UserID) + @MinVotes) AS WeightedRating,
    COUNT(WH.UserID) as Views
  FROM
    Movies M
  JOIN
    WatchHistory WH ON M.MovieID = WH.MovieID
  WHERE
    M.GenreID = @GenreID
  GROUP BY
    M.MovieID, M.Title, M.GenreID
  HAVING
    COUNT(WH.UserID) >= @MinVotes
  ORDER BY
    WeightedRating DESC;
```



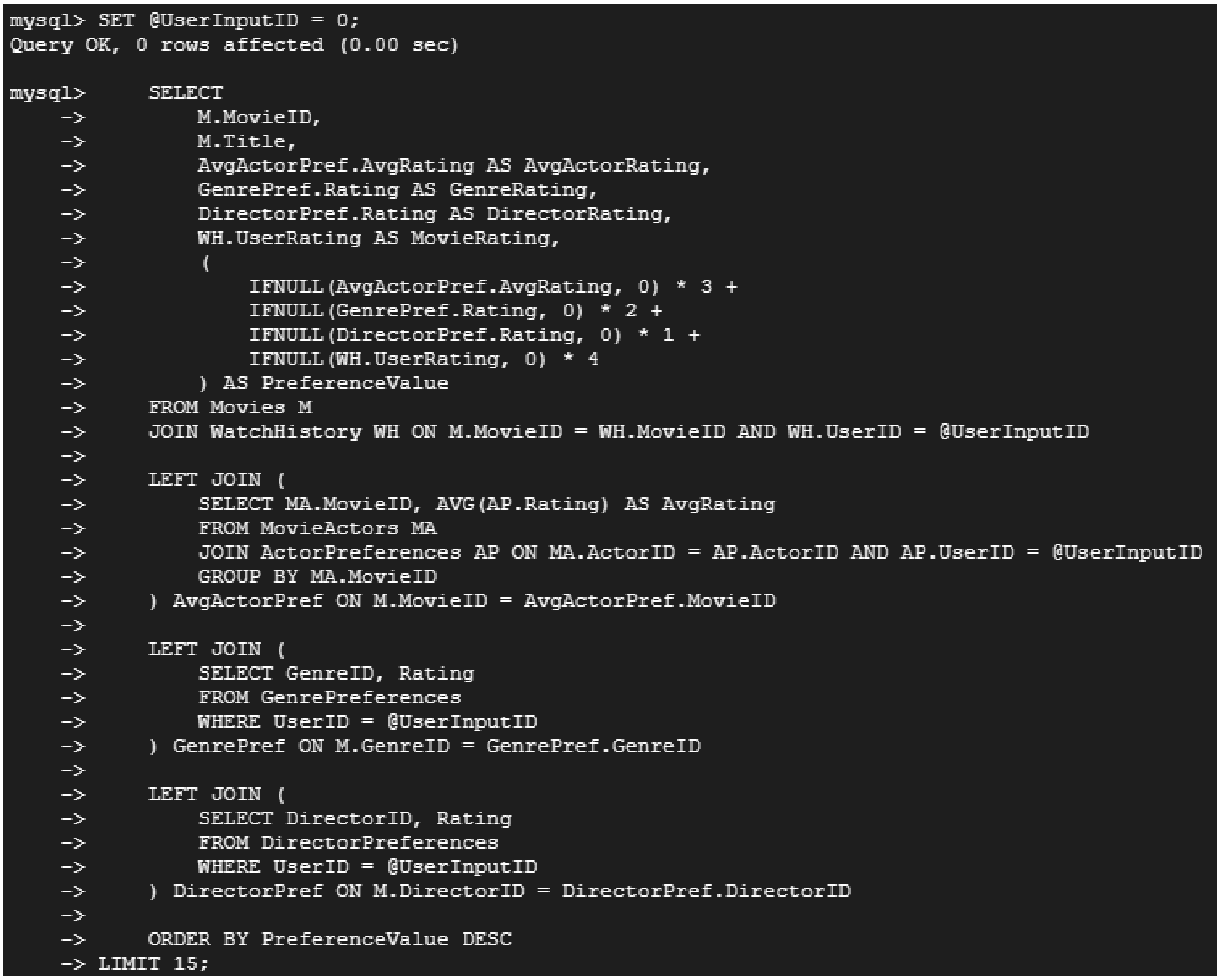
```
mysql> SET @GenreID = 4;
Query OK, 0 rows affected (0.00 sec)

mysql>     SET @MinVotes = 1;
Query OK, 0 rows affected (0.00 sec)

mysql>     SET @AvgUserRating = (SELECT AVG(UserRating) FROM WatchHistory);
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql>     SELECT
    ->         M.MovieID,
    ->         M.Title,
    ->         M.GenreID,
es) / (COUNT(WH.UserID) + @MinVotes) AS WeightedRating, @AvgUserRating * @MinVot
    ->         COUNT(WH.UserID) as Views
    ->     FROM
    ->         Movies M
    ->     JOIN
    ->         WatchHistory WH ON M.MovieID = WH.MovieID
    ->     WHERE
    ->         M.GenreID = @GenreID
    ->     GROUP BY
    ->         M.MovieID, M.Title, M.GenreID
    ->     HAVING
    ->         COUNT(WH.UserID) >= @MinVotes
    ->     ORDER BY
    ->         WeightedRating DESC LIMIT 15;
+-----------+----------------------------------------------+---------+----------------------------------+-------+
| MovieID   | Title                                        | GenreID | WeightedRating                   | Views |
+-----------+----------------------------------------------+---------+----------------------------------+-------+
| tt9890120 | Resurrection Corporation                     |       4 | 8.106690777500000000000000000000 |     1 |
```



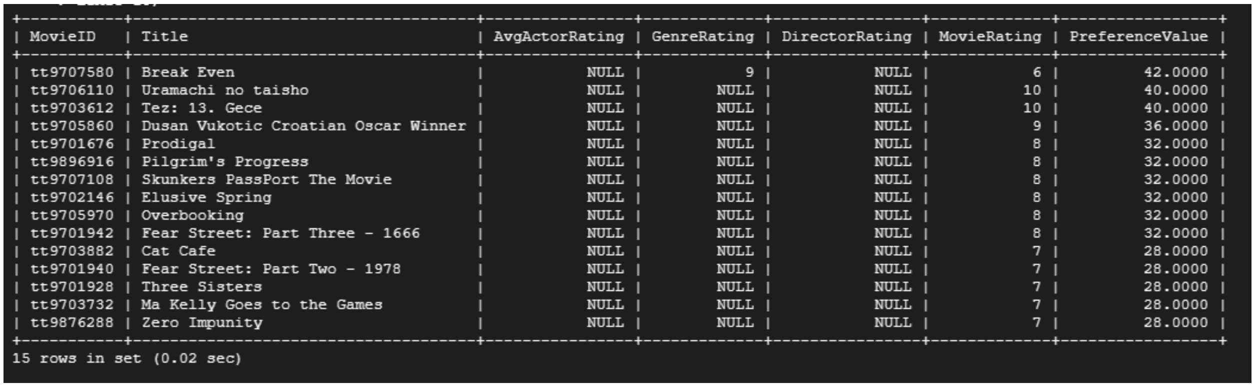
```
+-----------+----------------------------------------------+---------+----------------------------------+-------+
| MovieID   | Title                                        | GenreID | WeightedRating                   | Views |
+-----------+----------------------------------------------+---------+----------------------------------+-------+
| tt9890120 | Resurrection Corporation                     |       4 | 8.106690777500000000000000000000 |     1 |
| tt9899716 | Star Trek Enterprise II: Der Anfang vom Ende |       4 | 8.106690777500000000000000000000 |     1 |
| tt9909938 | Xenosaga Episode 1: Der Wille Zur Macht      |       4 | 7.606690777500000000000000000000 |     1 |
| tt9741908 | Breathless Animals                           |       4 | 7.106690777500000000000000000000 |     1 |
| tt9813004 | Les Fables de Starewitch                     |       4 | 7.071127185000000000000000000000 |     2 |
| tt9881850 | Dalia and the Red Book                       |       4 | 7.053345388500000000000000000000 |     3 |
| tt9871230 | Traveling Landscape                          |       4 | 6.414225436466666666000000000000 |    14 |
| tt9724092 | Shark School                                 |       4 | 6.273779384233333333000000000000 |    29 |
| tt9876288 | Zero Impunity                                |       4 | 6.202230259166666666000000000000 |     5 |
| tt9724306 | 2019 Oscar Nominated Short Films: Animation  |       4 | 6.021338155500000000000000000000 |     9 |
| tt9867200 | The End of the World - Episode 5: Civil War  |       4 | 5.950836346875000000000000000000 |    15 |
| tt9863566 | Urbanus: De Vuilnisheld                      |       4 | 5.814225436333333333000000000000 |    14 |
| tt9805160 | Fritzi: A Revolutionary Tale                 |       4 | 5.606690777500000000000000000000 |     1 |
| tt9713680 | Axel 2: Adventures of the Spacekids          |       4 | 5.600787150294117647000000000000 |    16 |
| tt9724128 | Learning with Penguins: Amazing Birds        |       4 | 5.444060067521739130000000000000 |    22 |
+-----------+----------------------------------------------+---------+----------------------------------+-------+
15 rows in set (0.01 sec)
```

Advanced Query “RankUserMovies”:
This query returns a user’s movies ranked in order of how much they favor this movie. This is based on a score system that combines the user’s rating of the movie itself, the user’s rating of the genre that the movie belongs to, the user’s rating for the director that produced the movie, and the user’s average rating for all of the actors in the movie. Movies with higher scores will be listed first in the table.
The SQL concepts used are the JOIN of multiple relations, GROUP BY aggregation, and subqueries.

```
SET @UserInputID = 0;

SELECT
   M.MovieID,
   M.Title,
   AvgActorPref.AvgRating AS AvgActorRating,
   GenrePref.Rating AS GenreRating,
   DirectorPref.Rating AS DirectorRating,
   WH.UserRating AS MovieRating,
   (
      IFNULL(AvgActorPref.AvgRating, 0) * 3 +
      IFNULL(GenrePref.Rating, 0) * 2 +
      IFNULL(DirectorPref.Rating, 0) * 1 +
      IFNULL(WH.UserRating, 0) * 4
   ) AS PreferenceValue
FROM Movies M
JOIN WatchHistory WH ON M.MovieID = WH.MovieID AND WH.UserID = @UserInputID
LEFT JOIN (
   SELECT MA.MovieID, AVG(AP.Rating) AS AvgRating
   FROM MovieActors MA
   JOIN ActorPreferences AP ON MA.ActorID = AP.ActorID AND AP.UserID = @UserInputID
   GROUP BY MA.MovieID
) AvgActorPref ON M.MovieID = AvgActorPref.MovieID
LEFT JOIN (
   SELECT GenreID, Rating
   FROM GenrePreferences
   WHERE UserID = @UserInputID
) GenrePref ON M.GenreID = GenrePref.GenreID
LEFT JOIN (
   SELECT DirectorID, Rating
   FROM DirectorPreferences
   WHERE UserID = @UserInputID
) DirectorPref ON M.DirectorID = DirectorPref.DirectorID
ORDER BY PreferenceValue DESC LIMIT 15;
```



```
mysql> SET @UserInputID = 0;
Query OK, 0 rows affected (0.00 sec)

mysql>     SELECT
    ->         M.MovieID,
    ->         M.Title,
    ->         AvgActorPref.AvgRating AS AvgActorRating,
    ->         GenrePref.Rating AS GenreRating,
    ->         DirectorPref.Rating AS DirectorRating,
    ->         WH.UserRating AS MovieRating,
    ->         (
    ->             IFNULL(AvgActorPref.AvgRating, 0) * 3 +
    ->             IFNULL(GenrePref.Rating, 0) * 2 +
    ->             IFNULL(DirectorPref.Rating, 0) * 1 +
    ->             IFNULL(WH.UserRating, 0) * 4
    ->         ) AS PreferenceValue
    ->     FROM Movies M
    ->     JOIN WatchHistory WH ON M.MovieID = WH.MovieID AND WH.UserID = @UserInputID
    ->
    ->     LEFT JOIN (
    ->         SELECT MA.MovieID, AVG(AP.Rating) AS AvgRating
    ->         FROM MovieActors MA
    ->         JOIN ActorPreferences AP ON MA.ActorID = AP.ActorID AND AP.UserID = @UserInputID
    ->         GROUP BY MA.MovieID
    ->     ) AvgActorPref ON M.MovieID = AvgActorPref.MovieID
    ->
    ->     LEFT JOIN (
    ->         SELECT GenreID, Rating
    ->         FROM GenrePreferences
    ->         WHERE UserID = @UserInputID
    ->     ) GenrePref ON M.GenreID = GenrePref.GenreID
    ->
    ->     LEFT JOIN (
    ->         SELECT DirectorID, Rating
    ->         FROM DirectorPreferences
    ->         WHERE UserID = @UserInputID
    ->     ) DirectorPref ON M.DirectorID = DirectorPref.DirectorID
    ->
    ->     ORDER BY PreferenceValue DESC
    -> LIMIT 15;
```



```
+-----------+-------------------------------------+----------------+-------------+----------------+-------------+-----------------+
| MovieID   | Title                               | AvgActorRating | GenreRating | DirectorRating | MovieRating | PreferenceValue |
+-----------+-------------------------------------+----------------+-------------+----------------+-------------+-----------------+
| tt9707580 | Break Even                          |           NULL |           9 |           NULL |           6 |         42.0000 |
| tt9706110 | Uramachi no taisho                  |           NULL |        NULL |           NULL |          10 |         40.0000 |
| tt9703612 | Tez: 13. Gece                       |           NULL |        NULL |           NULL |          10 |         40.0000 |
| tt9705860 | Dusan Vukotic Croatian Oscar Winner |           NULL |        NULL |           NULL |           9 |         36.0000 |
| tt9701676 | Prodigal                            |           NULL |        NULL |           NULL |           8 |         32.0000 |
| tt9896916 | Pilgrim's Progress                  |           NULL |        NULL |           NULL |           8 |         32.0000 |
| tt9707108 | Skunkers PassPort The Movie         |           NULL |        NULL |           NULL |           8 |         32.0000 |
| tt9702146 | Elusive Spring                      |           NULL |        NULL |           NULL |           8 |         32.0000 |
| tt9705970 | Overbooking                         |           NULL |        NULL |           NULL |           8 |         32.0000 |
| tt9701942 | Fear Street: Part Three - 1666      |           NULL |        NULL |           NULL |           8 |         32.0000 |
| tt9703882 | Cat Cafe                            |           NULL |        NULL |           NULL |           7 |         28.0000 |
| tt9701940 | Fear Street: Part Two - 1978        |           NULL |        NULL |           NULL |           7 |         28.0000 |
| tt9701928 | Three Sisters                       |           NULL |        NULL |           NULL |           7 |         28.0000 |
| tt9703732 | Ma Kelly Goes to the Games          |           NULL |        NULL |           NULL |           7 |         28.0000 |
| tt9876288 | Zero Impunity                       |           NULL |        NULL |           NULL |           7 |         28.0000 |
+-----------+-------------------------------------+----------------+-------------+----------------+-------------+-----------------+
15 rows in set (0.02 sec)
```

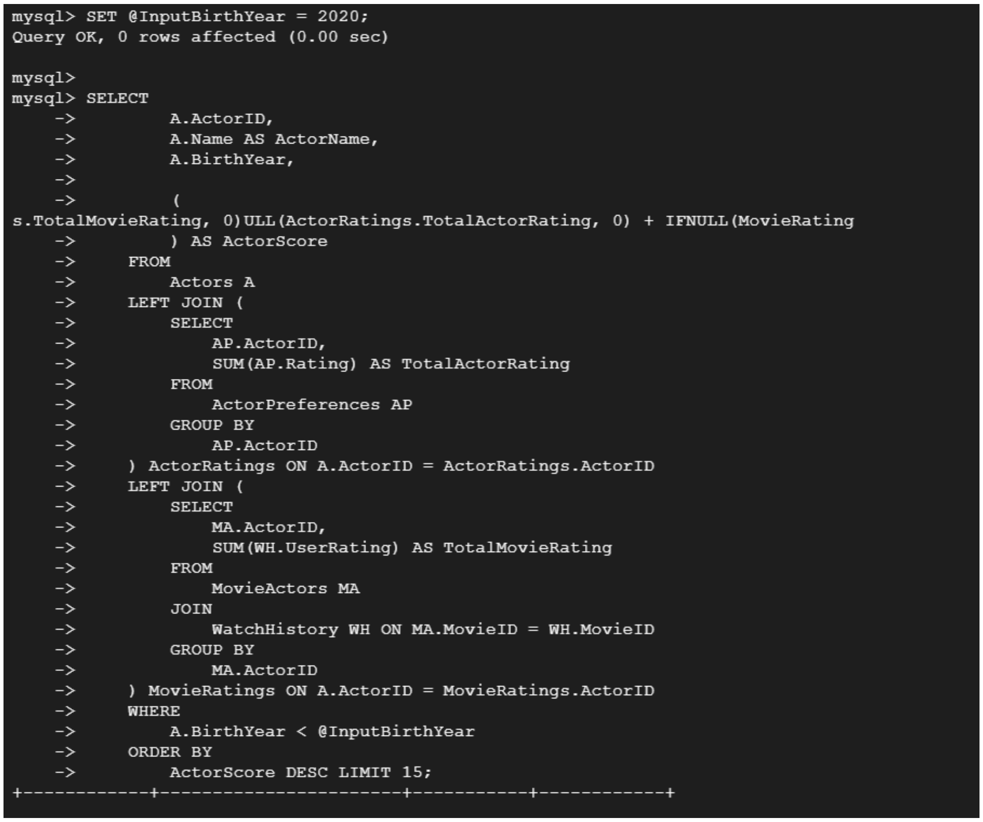
Because of the massive amounts of movie, actors, and genre elements, it was very unlikely for there to be corresponding ratings by User elements for the capacity of User elements that we could create at the time.

Advanced Query “GlobalRankActors”

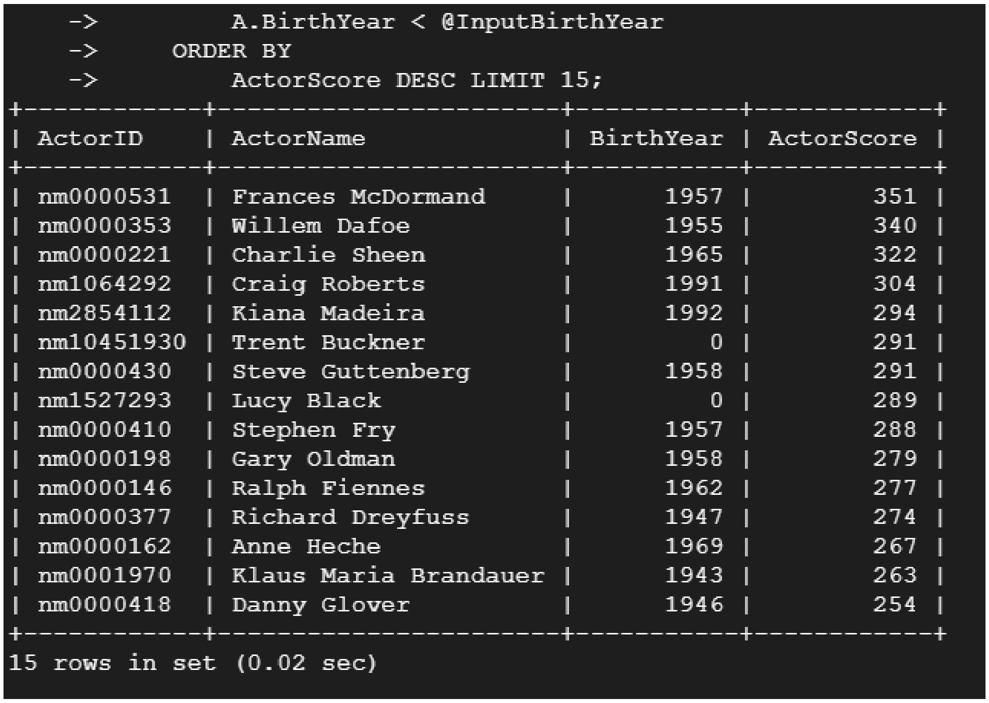
This query returns a ranked order of all the actors born before a certain year, based on a score system that combines the total ratings amongst all users that have rated the actor and the total user ratings of all movies that the actor has played in. Actors with higher scores will be displayed higher in the table.

The SQL concepts used are the JOIN of multiple relations, GROUP BY aggregation, and subqueries.

```
SET @InputBirthYear = 2020;

SELECT
        A.ActorID,
        A.Name AS ActorName,
        A.BirthYear,

        (
            IFNULL(ActorRatings.TotalActorRating, 0) + IFNULL(MovieRatings.TotalMovieRating, 0)
        ) AS ActorScore
    FROM
        Actors A
    LEFT JOIN (
        SELECT
            AP.ActorID,
            SUM(AP.Rating) AS TotalActorRating
        FROM
            ActorPreferences AP
        GROUP BY
            AP.ActorID
    ) ActorRatings ON A.ActorID = ActorRatings.ActorID
    LEFT JOIN (
        SELECT
            MA.ActorID,
            SUM(WH.UserRating) AS TotalMovieRating
        FROM
            MovieActors MA
        JOIN
            WatchHistory WH ON MA.MovieID = WH.MovieID
        GROUP BY
            MA.ActorID
    ) MovieRatings ON A.ActorID = MovieRatings.ActorID
    WHERE
        A.BirthYear < @InputBirthYear
    ORDER BY
        ActorScore DESC;
```



```
mysql> SET @InputBirthYear = 2020;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> SELECT
    ->         A.ActorID,
    ->         A.Name AS ActorName,
    ->         A.BirthYear,
    ->
    ->         (
s.TotalMovieRating, 0)ULL(ActorRatings.TotalActorRating, 0) + IFNULL(MovieRating
    ->         ) AS ActorScore
    ->     FROM
    ->         Actors A
    ->     LEFT JOIN (
    ->         SELECT
    ->             AP.ActorID,
    ->             SUM(AP.Rating) AS TotalActorRating
    ->         FROM
    ->             ActorPreferences AP
    ->         GROUP BY
    ->             AP.ActorID
    ->     ) ActorRatings ON A.ActorID = ActorRatings.ActorID
    ->     LEFT JOIN (
    ->         SELECT
    ->             MA.ActorID,
    ->             SUM(WH.UserRating) AS TotalMovieRating
    ->         FROM
    ->             MovieActors MA
    ->         JOIN
    ->             WatchHistory WH ON MA.MovieID = WH.MovieID
    ->         GROUP BY
    ->             MA.ActorID
    ->     ) MovieRatings ON A.ActorID = MovieRatings.ActorID
    ->     WHERE
    ->         A.BirthYear < @InputBirthYear
    ->     ORDER BY
    ->         ActorScore DESC LIMIT 15;
+------------+----------------------+-----------+------------+
```



```
    ->         A.BirthYear < @InputBirthYear
    ->     ORDER BY
    ->         ActorScore DESC LIMIT 15;
```

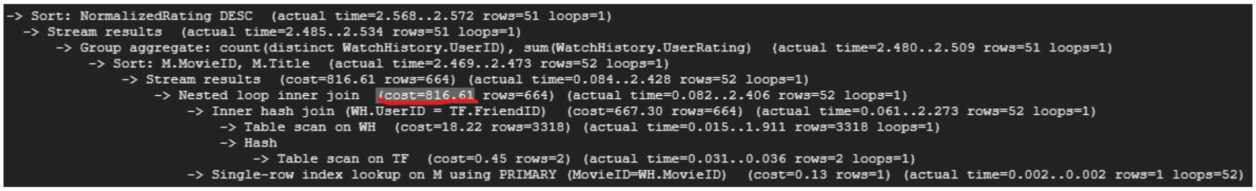
```
+------------+-----------------------+-----------+------------+
| ActorID    | ActorName             | BirthYear | ActorScore |
+------------+-----------------------+-----------+------------+
| nm0000531  | Frances McDormand     |      1957 |        351 |
| nm0000353  | Willem Dafoe          |      1955 |        340 |
| nm0000221  | Charlie Sheen         |      1965 |        322 |
| nm1064292  | Craig Roberts         |      1991 |        304 |
| nm2854112  | Kiana Madeira         |      1992 |        294 |
| nm10451930 | Trent Buckner         |         0 |        291 |
| nm0000430  | Steve Guttenberg      |      1958 |        291 |
| nm1527293  | Lucy Black            |         0 |        289 |
| nm0000410  | Stephen Fry           |      1957 |        288 |
| nm0000198  | Gary Oldman           |      1958 |        279 |
| nm0000146  | Ralph Fiennes         |      1962 |        277 |
| nm0000377  | Richard Dreyfuss      |      1947 |        274 |
| nm0000162  | Anne Heche            |      1969 |        267 |
| nm0001970  | Klaus Maria Brandauer |      1943 |        263 |
| nm0000418  | Danny Glover          |      1946 |        254 |
+------------+-----------------------+-----------+------------+
15 rows in set (0.02 sec)
```

2.1 Initial Costs:

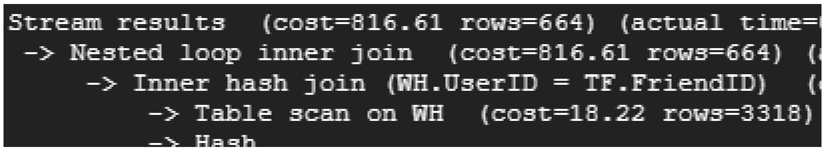
GetFriendsRatedMovies:

NO INDEXES

Initial Nested Loop Inner Join Cost: 816.61



```
-> Sort: NormalizedRating DESC  (actual time=2.568..2.572 rows=51 loops=1)
  -> Stream results  (actual time=2.485..2.534 rows=51 loops=1)
      -> Group aggregate: count(distinct WatchHistory.UserID), sum(WatchHistory.UserRating)  (actual time=2.480..2.509 rows=51 loops=1)
          -> Sort: M.MovieID, M.Title  (actual time=2.469..2.473 rows=52 loops=1)
              -> Stream results  (cost=816.61 rows=664) (actual time=0.084..2.428 rows=52 loops=1)
                  -> Nested loop inner join  (cost=816.61 rows=664) (actual time=0.082..2.406 rows=52 loops=1)
                      -> Inner hash join (WH.UserID = TF.FriendID)  (cost=667.30 rows=664) (actual time=0.061..2.273 rows=52 loops=1)
                          -> Table scan on WH  (cost=18.22 rows=3318) (actual time=0.015..1.911 rows=3318 loops=1)
                          -> Hash
                              -> Table scan on TF  (cost=0.45 rows=2) (actual time=0.031..0.036 rows=2 loops=1)
                      -> Single-row index lookup on M using PRIMARY (MovieID=WH.MovieID)  (cost=0.13 rows=1) (actual time=0.002..0.002 rows=1 loops=52)
```

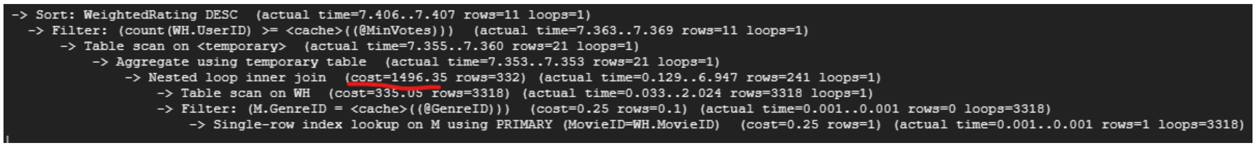


```
Stream results  (cost=816.61 rows=664) (actual time=
 -> Nested loop inner join  (cost=816.61 rows=664) (
     -> Inner hash join (WH.UserID = TF.FriendID)  (
         -> Table scan on WH  (cost=18.22 rows=3318)
         -> Hash
```

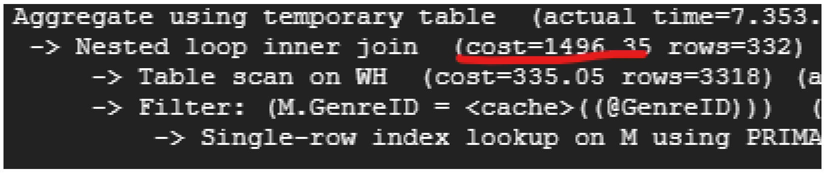
GetBestMoviesByGenre:

NO INDEXES

Initial Nested Loop Inner Join Cost: 1496.35



```
-> Sort: WeightedRating DESC  (actual time=7.406..7.407 rows=11 loops=1)
  -> Filter: (count(WH.UserID) >= <cache>((@MinVotes)))  (actual time=7.363..7.369 rows=11 loops=1)
      -> Table scan on <temporary>  (actual time=7.355..7.360 rows=21 loops=1)
          -> Aggregate using temporary table  (actual time=7.353..7.353 rows=21 loops=1)
              -> Nested loop inner join  (cost=1496.35 rows=332) (actual time=0.129..6.947 rows=241 loops=1)
                  -> Table scan on WH  (cost=335.05 rows=3318) (actual time=0.033..2.024 rows=3318 loops=1)
                  -> Filter: (M.GenreID = <cache>((@GenreID)))  (cost=0.25 rows=0.1) (actual time=0.001..0.001 rows=0 loops=3318)
                      -> Single-row index lookup on M using PRIMARY (MovieID=WH.MovieID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=3318)
```

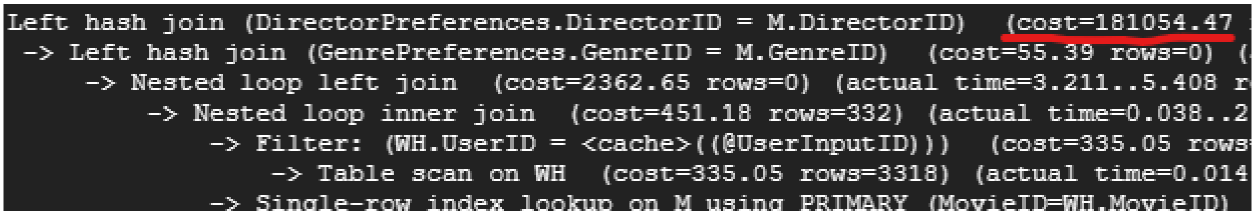


```
Aggregate using temporary table  (actual time=7.353.
 -> Nested loop inner join  (cost=1496.35 rows=332)
     -> Table scan on WH  (cost=335.05 rows=3318) (a
     -> Filter: (M.GenreID = <cache>((@GenreID)))  (
         -> Single-row index lookup on M using PRIMA
```

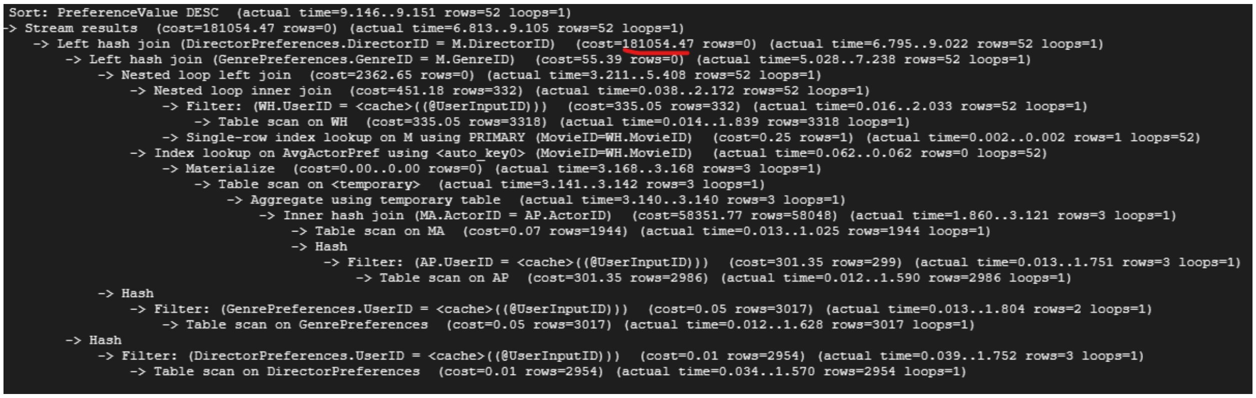
RankUserMovies:

NO INDEXES

Initial Left Hash Join Cost: 181054.47



```
Left hash join (DirectorPreferences.DirectorID = M.DirectorID)  (cost=181054.47
 -> Left hash join (GenrePreferences.GenreID = M.GenreID)  (cost=55.39 rows=0) (
     -> Nested loop left join  (cost=2362.65 rows=0) (actual time=3.211..5.408 r
         -> Nested loop inner join  (cost=451.18 rows=332) (actual time=0.038..2
             -> Filter: (WH.UserID = <cache>((@UserInputID)))  (cost=335.05 rows
                 -> Table scan on WH  (cost=335.05 rows=3318) (actual time=0.014
             -> Single-row index lookup on M using PRIMARY (MovieID=WH.MovieID)
```

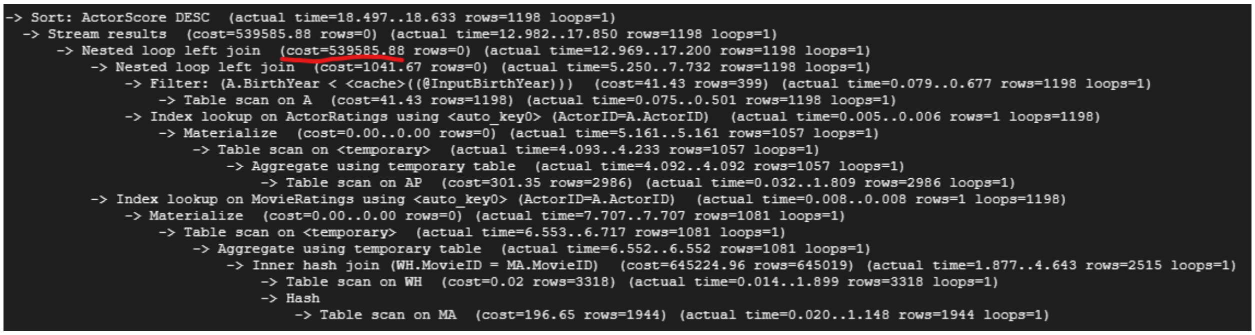


```
Sort: PreferenceValue DESC  (actual time=9.146..9.151 rows=52 loops=1)
-> Stream results  (cost=181054.47 rows=0) (actual time=6.813..9.105 rows=52 loops=1)
    -> Left hash join (DirectorPreferences.DirectorID = M.DirectorID)  (cost=181054.47 rows=0) (actual time=6.795..9.022 rows=52 loops=1)
        -> Left hash join (GenrePreferences.GenreID = M.GenreID)  (cost=55.39 rows=0) (actual time=5.028..7.238 rows=52 loops=1)
            -> Nested loop left join  (cost=2362.65 rows=0) (actual time=3.211..5.408 rows=52 loops=1)
                -> Nested loop inner join  (cost=451.18 rows=332) (actual time=0.038..2.172 rows=52 loops=1)
                    -> Filter: (WH.UserID = <cache>((@UserInputID)))  (cost=335.05 rows=332) (actual time=0.016..2.033 rows=52 loops=1)
                        -> Table scan on WH  (cost=335.05 rows=3318) (actual time=0.014..1.839 rows=3318 loops=1)
                    -> Single-row index lookup on M using PRIMARY (MovieID=WH.MovieID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=52)
                -> Index lookup on AvgActorPref using <auto_key0> (MovieID=WH.MovieID)  (actual time=0.062..0.062 rows=0 loops=52)
                    -> Materialize  (cost=0.00..0.00 rows=0) (actual time=3.168..3.168 rows=3 loops=1)
                        -> Table scan on <temporary>  (actual time=3.141..3.142 rows=3 loops=1)
                            -> Aggregate using temporary table  (actual time=3.140..3.140 rows=3 loops=1)
                                -> Inner hash join (MA.ActorID = AP.ActorID)  (cost=58351.77 rows=58048) (actual time=1.860..3.121 rows=3 loops=1)
                                    -> Table scan on MA  (cost=0.07 rows=1944) (actual time=0.013..1.025 rows=1944 loops=1)
                                    -> Hash
                                        -> Filter: (AP.UserID = <cache>((@UserInputID)))  (cost=301.35 rows=299) (actual time=0.013..1.751 rows=3 loops=1)
                                            -> Table scan on AP  (cost=301.35 rows=2986) (actual time=0.012..1.590 rows=2986 loops=1)
            -> Hash
                -> Filter: (GenrePreferences.UserID = <cache>((@UserInputID)))  (cost=0.05 rows=3017) (actual time=0.013..1.804 rows=2 loops=1)
                    -> Table scan on GenrePreferences  (cost=0.05 rows=3017) (actual time=0.012..1.628 rows=3017 loops=1)
        -> Hash
            -> Filter: (DirectorPreferences.UserID = <cache>((@UserInputID)))  (cost=0.01 rows=2954) (actual time=0.039..1.752 rows=3 loops=1)
                -> Table scan on DirectorPreferences  (cost=0.01 rows=2954) (actual time=0.034..1.570 rows=2954 loops=1)
```

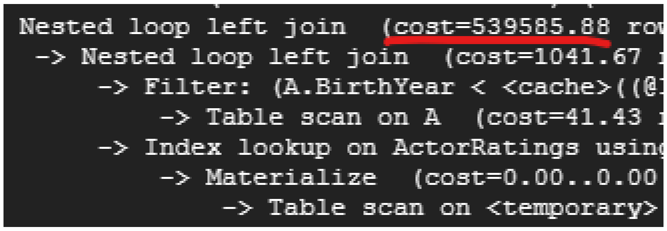
GlobalRankActors:

NO INDEXES

Initial Nested Loop Left Join Cost: 539585.88



```
-> Sort: ActorScore DESC  (actual time=18.497..18.633 rows=1198 loops=1)
  -> Stream results  (cost=539585.88 rows=0) (actual time=12.982..17.850 rows=1198 loops=1)
      -> Nested loop left join  (cost=539585.88 rows=0) (actual time=12.969..17.200 rows=1198 loops=1)
          -> Nested loop left join  (cost=1041.67 rows=0) (actual time=5.250..7.732 rows=1198 loops=1)
              -> Filter: (A.BirthYear < <cache>((@InputBirthYear)))  (cost=41.43 rows=399) (actual time=0.079..0.677 rows=1198 loops=1)
                  -> Table scan on A  (cost=41.43 rows=1198) (actual time=0.075..0.501 rows=1198 loops=1)
              -> Index lookup on ActorRatings using <auto_key0> (ActorID=A.ActorID)  (actual time=0.005..0.006 rows=1 loops=1198)
                  -> Materialize  (cost=0.00..0.00 rows=0) (actual time=5.161..5.161 rows=1057 loops=1)
                      -> Table scan on <temporary>  (actual time=4.093..4.233 rows=1057 loops=1)
                          -> Aggregate using temporary table  (actual time=4.092..4.092 rows=1057 loops=1)
                              -> Table scan on AP  (cost=301.35 rows=2986) (actual time=0.032..1.809 rows=2986 loops=1)
          -> Index lookup on MovieRatings using <auto_key0> (ActorID=A.ActorID)  (actual time=0.008..0.008 rows=1 loops=1198)
              -> Materialize  (cost=0.00..0.00 rows=0) (actual time=7.707..7.707 rows=1081 loops=1)
                  -> Table scan on <temporary>  (actual time=6.553..6.717 rows=1081 loops=1)
                      -> Aggregate using temporary table  (actual time=6.552..6.552 rows=1081 loops=1)
                          -> Inner hash join (WH.MovieID = MA.MovieID)  (cost=645224.96 rows=645019) (actual time=1.877..4.643 rows=2515 loops=1)
                              -> Table scan on WH  (cost=0.02 rows=3318) (actual time=0.014..1.899 rows=3318 loops=1)
                              -> Hash
                                  -> Table scan on MA  (cost=196.65 rows=1944) (actual time=0.020..1.148 rows=1944 loops=1)
```



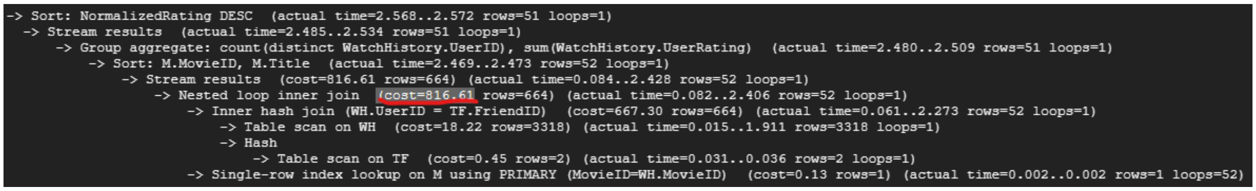
```
Nested loop left join  (cost=539585.88 row
 -> Nested loop left join  (cost=1041.67 r
     -> Filter: (A.BirthYear < <cache>((@I
         -> Table scan on A  (cost=41.43 r
     -> Index lookup on ActorRatings using
         -> Materialize  (cost=0.00..0.00
             -> Table scan on <temporary>
```

2.2 Explore Tradeoffs:

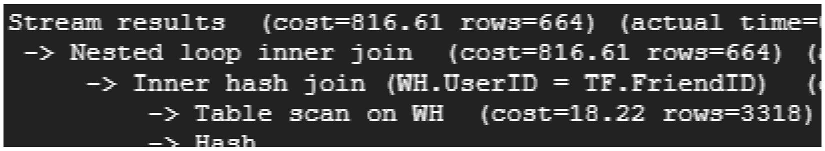
GetFriendsRatedMovies:

Indexing on Friends.UserID1 and Friends.UserID2:
CREATE INDEX friends_user1id_idx ON Friends(User1ID);
CREATE INDEX friends_user1id_idx ON Friends(User1ID);

Initial Cost: 816.61
New Cost: 816.61



```
-> Sort: NormalizedRating DESC  (actual time=2.568..2.572 rows=51 loops=1)
  -> Stream results  (actual time=2.485..2.534 rows=51 loops=1)
      -> Group aggregate: count(distinct WatchHistory.UserID), sum(WatchHistory.UserRating)  (actual time=2.480..2.509 rows=51 loops=1)
          -> Sort: M.MovieID, M.Title  (actual time=2.469..2.473 rows=52 loops=1)
              -> Stream results  (cost=816.61 rows=664) (actual time=0.084..2.428 rows=52 loops=1)
                  -> Nested loop inner join  (cost=816.61 rows=664) (actual time=0.082..2.406 rows=52 loops=1)
                      -> Inner hash join (WH.UserID = TF.FriendID)  (cost=667.30 rows=664) (actual time=0.061..2.273 rows=52 loops=1)
                          -> Table scan on WH  (cost=18.22 rows=3318) (actual time=0.015..1.911 rows=3318 loops=1)
                          -> Hash
                              -> Table scan on TF  (cost=0.45 rows=2) (actual time=0.031..0.036 rows=2 loops=1)
                      -> Single-row index lookup on M using PRIMARY (MovieID=WH.MovieID)  (cost=0.13 rows=1) (actual time=0.002..0.002 rows=1 loops=52)
```

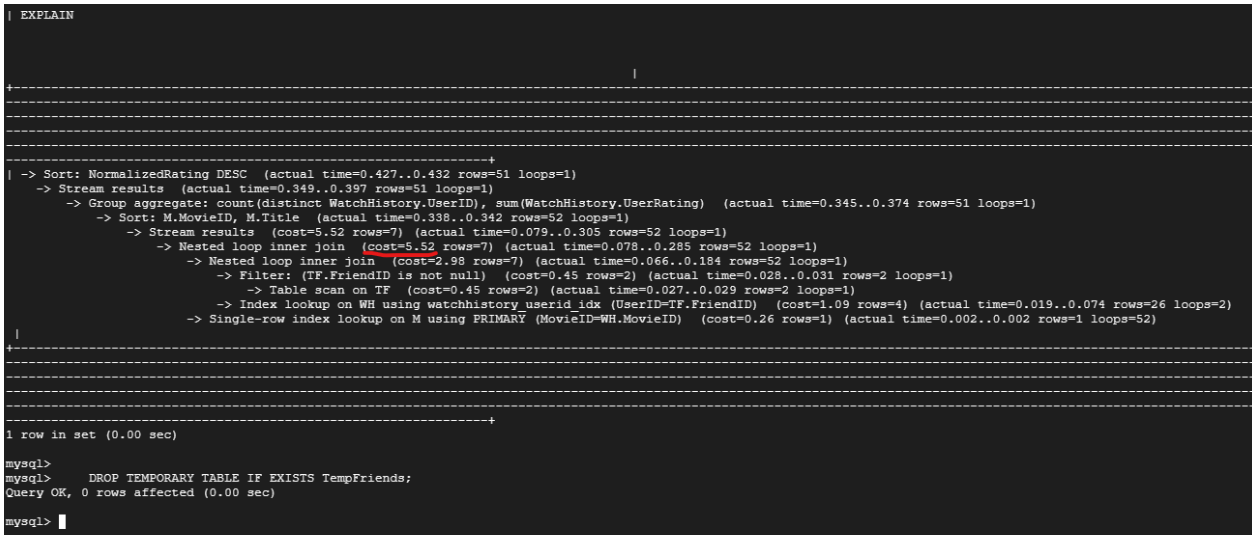


```
Stream results  (cost=816.61 rows=664) (actual time=
 -> Nested loop inner join  (cost=816.61 rows=664) (
     -> Inner hash join (WH.UserID = TF.FriendID)  (
         -> Table scan on WH  (cost=18.22 rows=3318)
         -> Hash
```

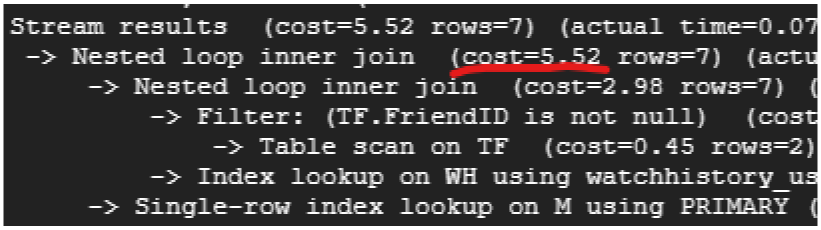
Since we do not have many attributes, we can not do both "Have 3 indexing designs" and "Do not index primary keys, no indexing attributes not selected". So, we were forced to use foreign keys not outputted. It makes sense that the cost did not change here.

Indexing on WatchHistory.UserID indexing:
CREATE INDEX watchhistory_userid_idx ON WatchHistory(UserID);

Initial Cost: 816.61
New Cost: 5.52



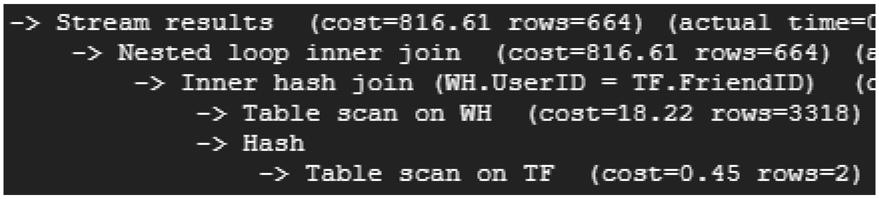
```
| EXPLAIN



                                                                                          |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------+
| -> Sort: NormalizedRating DESC  (actual time=0.427..0.432 rows=51 loops=1)
    -> Stream results  (actual time=0.349..0.397 rows=51 loops=1)
        -> Group aggregate: count(distinct WatchHistory.UserID), sum(WatchHistory.UserRating)  (actual time=0.345..0.374 rows=51 loops=1)
            -> Sort: M.MovieID, M.Title  (actual time=0.338..0.342 rows=52 loops=1)
                -> Stream results  (cost=5.52 rows=7) (actual time=0.079..0.305 rows=52 loops=1)
                    -> Nested loop inner join  (cost=5.52 rows=7) (actual time=0.078..0.285 rows=52 loops=1)
                        -> Nested loop inner join  (cost=2.98 rows=7) (actual time=0.066..0.184 rows=52 loops=1)
                            -> Filter: (TF.FriendID is not null)  (cost=0.45 rows=2) (actual time=0.028..0.031 rows=2 loops=1)
                                -> Table scan on TF  (cost=0.45 rows=2) (actual time=0.027..0.029 rows=2 loops=1)
                            -> Index lookup on WH using watchhistory_userid_idx (UserID=TF.FriendID)  (cost=1.09 rows=4) (actual time=0.019..0.074 rows=26 loops=2)
                        -> Single-row index lookup on M using PRIMARY (MovieID=WH.MovieID)  (cost=0.26 rows=1) (actual time=0.002..0.002 rows=1 loops=52)
 |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------+
1 row in set (0.00 sec)

mysql>
mysql>     DROP TEMPORARY TABLE IF EXISTS TempFriends;
Query OK, 0 rows affected (0.00 sec)

mysql> █
```



```
Stream results  (cost=5.52 rows=7) (actual time=0.07
 -> Nested loop inner join  (cost=5.52 rows=7) (actu
     -> Nested loop inner join  (cost=2.98 rows=7) (
         -> Filter: (TF.FriendID is not null)  (cost
             -> Table scan on TF  (cost=0.45 rows=2)
         -> Index lookup on WH using watchhistory_us
     -> Single-row index lookup on M using PRIMARY (
```

Indexing on WatchHistory.MovieID:
CREATE INDEX watchhistory_movieid_idx ON WatchHistory(MovieID);

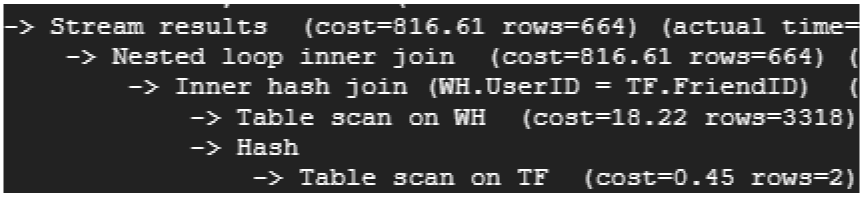
Initial Cost: 816.61
New Cost: 816.61



```
-> Stream results  (cost=816.61 rows=664) (actual time=0
    -> Nested loop inner join  (cost=816.61 rows=664) (a
        -> Inner hash join (WH.UserID = TF.FriendID)  (c
            -> Table scan on WH  (cost=18.22 rows=3318)
            -> Hash
                -> Table scan on TF  (cost=0.45 rows=2)
```

Indexing on Movies.Title:
CREATE INDEX movies_title_idx ON Movies(Title);

Initial Cost: 816.61
New Cost: 816.61



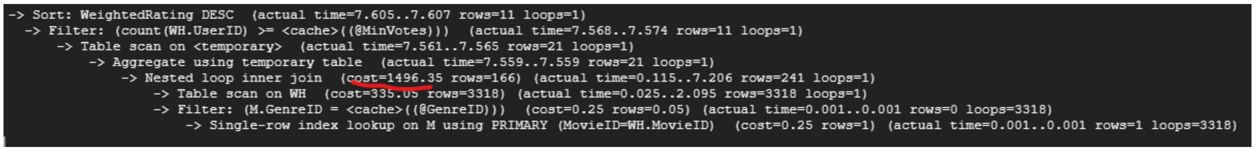
```
-> Stream results  (cost=816.61 rows=664) (actual time=
    -> Nested loop inner join  (cost=816.61 rows=664) (
        -> Inner hash join (WH.UserID = TF.FriendID)  (
            -> Table scan on WH  (cost=18.22 rows=3318)
            -> Hash
                -> Table scan on TF  (cost=0.45 rows=2)
```

Conclusions:
The indexing on WatchHistory.UserID drastically reduced the performance cost from 816.61 to 5.52 so this index variation will be incorporated into our design for this query. This is most likely because the advanced query performs cost-intensive operations (subquery, joining TempFriends table and Movies) centered around the UserID attribute of WatchHistory. The other indexing variations did not reduce the cost all, most likely because the advanced query only pulls data but does not perform operations on these table attributes.

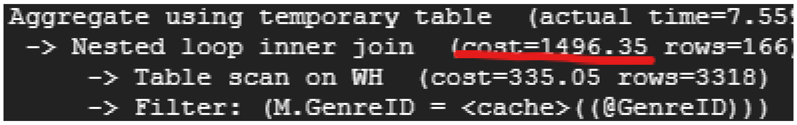
GetBestMoviesByGenre:

Indexing on Movies.GenreID:
CREATE INDEX movies_genreid_idx ON Movies(GenreID);

Initial Cost: 1496.35
New Cost: 1496.35



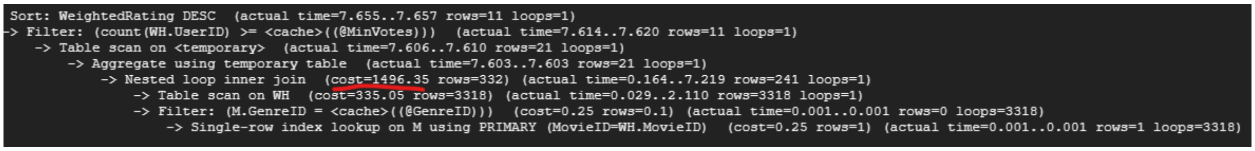
```
-> Sort: WeightedRating DESC  (actual time=7.605..7.607 rows=11 loops=1)
  -> Filter: (count(WH.UserID) >= <cache>((@MinVotes)))  (actual time=7.568..7.574 rows=11 loops=1)
      -> Table scan on <temporary>  (actual time=7.561..7.565 rows=21 loops=1)
          -> Aggregate using temporary table  (actual time=7.559..7.559 rows=21 loops=1)
              -> Nested loop inner join  (cost=1496.35 rows=166) (actual time=0.115..7.206 rows=241 loops=1)
                  -> Table scan on WH  (cost=335.05 rows=3318) (actual time=0.025..2.095 rows=3318 loops=1)
                  -> Filter: (M.GenreID = <cache>((@GenreID)))  (cost=0.25 rows=0.05) (actual time=0.001..0.001 rows=0 loops=3318)
                      -> Single-row index lookup on M using PRIMARY (MovieID=WH.MovieID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=3318)
```



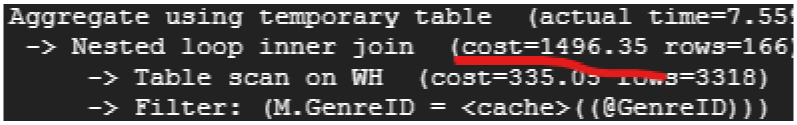
```
Aggregate using temporary table  (actual time=7.55
 -> Nested loop inner join  (cost=1496.35 rows=166
     -> Table scan on WH  (cost=335.05 rows=3318)
     -> Filter: (M.GenreID = <cache>((@GenreID)))
```

Indexing on WatchHistory.UserRating:
CREATE INDEX watchhistory_userrating_idx ON WatchHistory(UserRating);

Initial Cost: 1496.35
New Cost: 1496.35



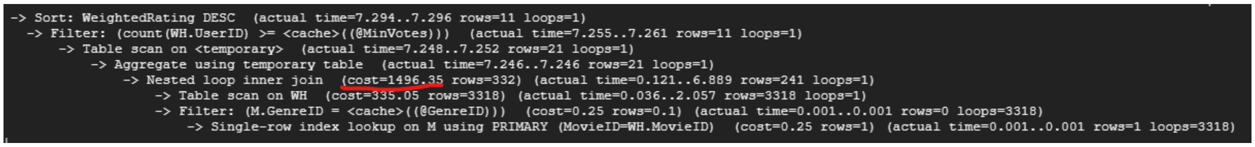
```
Sort: WeightedRating DESC  (actual time=7.655..7.657 rows=11 loops=1)
-> Filter: (count(WH.UserID) >= <cache>((@MinVotes)))  (actual time=7.614..7.620 rows=11 loops=1)
    -> Table scan on <temporary>  (actual time=7.606..7.610 rows=21 loops=1)
        -> Aggregate using temporary table  (actual time=7.603..7.603 rows=21 loops=1)
            -> Nested loop inner join  (cost=1496.35 rows=332) (actual time=0.164..7.219 rows=241 loops=1)
                -> Table scan on WH  (cost=335.05 rows=3318) (actual time=0.029..2.110 rows=3318 loops=1)
                -> Filter: (M.GenreID = <cache>((@GenreID)))  (cost=0.25 rows=0.1) (actual time=0.001..0.001 rows=0 loops=3318)
                    -> Single-row index lookup on M using PRIMARY (MovieID=WH.MovieID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=3318)
```



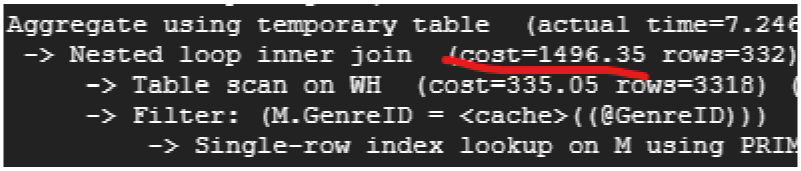
```
Aggregate using temporary table  (actual time=7.55
 -> Nested loop inner join  (cost=1496.35 rows=166
     -> Table scan on WH  (cost=335.05 rows=3318)
     -> Filter: (M.GenreID = <cache>((@GenreID)))
```

Indexing on Movies.Title:
CREATE INDEX movies_title_idx ON Movies(Title);

Initial Cost: 1496.35
New Cost: 1496.35



```
-> Sort: WeightedRating DESC  (actual time=7.294..7.296 rows=11 loops=1)
  -> Filter: (count(WH.UserID) >= <cache>((@MinVotes)))  (actual time=7.255..7.261 rows=11 loops=1)
      -> Table scan on <temporary>  (actual time=7.248..7.252 rows=21 loops=1)
          -> Aggregate using temporary table  (actual time=7.246..7.246 rows=21 loops=1)
              -> Nested loop inner join  (cost=1496.35 rows=332) (actual time=0.121..6.889 rows=241 loops=1)
                  -> Table scan on WH  (cost=335.05 rows=3318) (actual time=0.036..2.057 rows=3318 loops=1)
                  -> Filter: (M.GenreID = <cache>((@GenreID)))  (cost=0.25 rows=0.1) (actual time=0.001..0.001 rows=0 loops=3318)
                      -> Single-row index lookup on M using PRIMARY (MovieID=WH.MovieID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=3318)
```



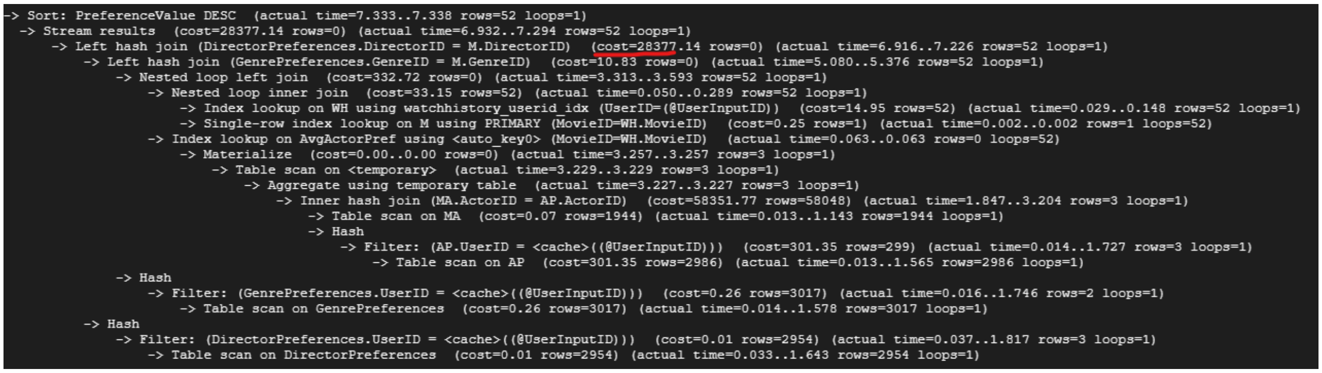
```
Aggregate using temporary table  (actual time=7.246
 -> Nested loop inner join  (cost=1496.35 rows=332)
     -> Table scan on WH  (cost=335.05 rows=3318) (
     -> Filter: (M.GenreID = <cache>((@GenreID)))
         -> Single-row index lookup on M using PRIM
```

Conclusions:
None of these indexing variations improved or even changed the cost of the advanced query, so none of them have a reason to be used in the database design. This is most likely because indexing does not improve the performance of the operations performed on the indexed tables compared to operating with a full table scan.

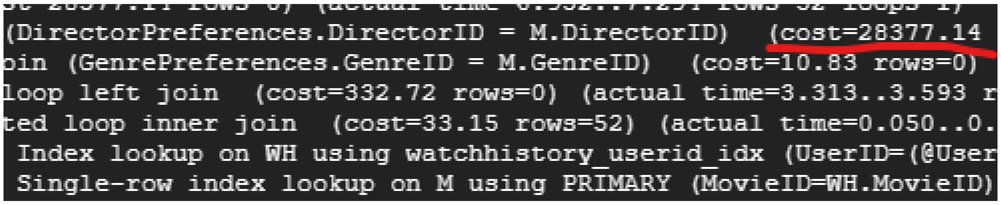
RankUserMovies:

Indexing on WatchHistory.UserID:
CREATE INDEX watchhistory_userid_idx ON WatchHistory(UserID);

Initial Cost: 181054.47
New Cost: 28377.14



```
-> Sort: PreferenceValue DESC  (actual time=7.333..7.338 rows=52 loops=1)
  -> Stream results  (cost=28377.14 rows=0) (actual time=6.932..7.294 rows=52 loops=1)
      -> Left hash join (DirectorPreferences.DirectorID = M.DirectorID)  (cost=28377.14 rows=0) (actual time=6.916..7.226 rows=52 loops=1)
          -> Left hash join (GenrePreferences.GenreID = M.GenreID)  (cost=10.83 rows=0) (actual time=5.080..5.376 rows=52 loops=1)
              -> Nested loop left join  (cost=332.72 rows=0) (actual time=3.313..3.593 rows=52 loops=1)
                  -> Nested loop inner join  (cost=33.15 rows=52) (actual time=0.050..0.289 rows=52 loops=1)
                      -> Index lookup on WH using watchhistory_userid_idx (UserID=(@UserInputID))  (cost=14.95 rows=52) (actual time=0.029..0.148 rows=52 loops=1)
                      -> Single-row index lookup on M using PRIMARY (MovieID=WH.MovieID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=52)
                  -> Index lookup on AvgActorPref using <auto_key0> (MovieID=WH.MovieID)  (actual time=0.063..0.063 rows=0 loops=52)
                      -> Materialize  (cost=0.00..0.00 rows=0) (actual time=3.257..3.257 rows=3 loops=1)
                          -> Table scan on <temporary>  (actual time=3.229..3.229 rows=3 loops=1)
                              -> Aggregate using temporary table  (actual time=3.227..3.227 rows=3 loops=1)
                                  -> Inner hash join (MA.ActorID = AP.ActorID)  (cost=58351.77 rows=58048) (actual time=1.847..3.204 rows=3 loops=1)
                                      -> Table scan on MA  (cost=0.07 rows=1944) (actual time=0.013..1.143 rows=1944 loops=1)
                                      -> Hash
                                          -> Filter: (AP.UserID = <cache>((@UserInputID)))  (cost=301.35 rows=299) (actual time=0.014..1.727 rows=3 loops=1)
                                              -> Table scan on AP  (cost=301.35 rows=2986) (actual time=0.013..1.565 rows=2986 loops=1)
              -> Hash
                  -> Filter: (GenrePreferences.UserID = <cache>((@UserInputID)))  (cost=0.26 rows=3017) (actual time=0.016..1.746 rows=2 loops=1)
                      -> Table scan on GenrePreferences  (cost=0.26 rows=3017) (actual time=0.014..1.578 rows=3017 loops=1)
          -> Hash
              -> Filter: (DirectorPreferences.UserID = <cache>((@UserInputID)))  (cost=0.01 rows=2954) (actual time=0.037..1.817 rows=3 loops=1)
                  -> Table scan on DirectorPreferences  (cost=0.01 rows=2954) (actual time=0.033..1.643 rows=2954 loops=1)
```

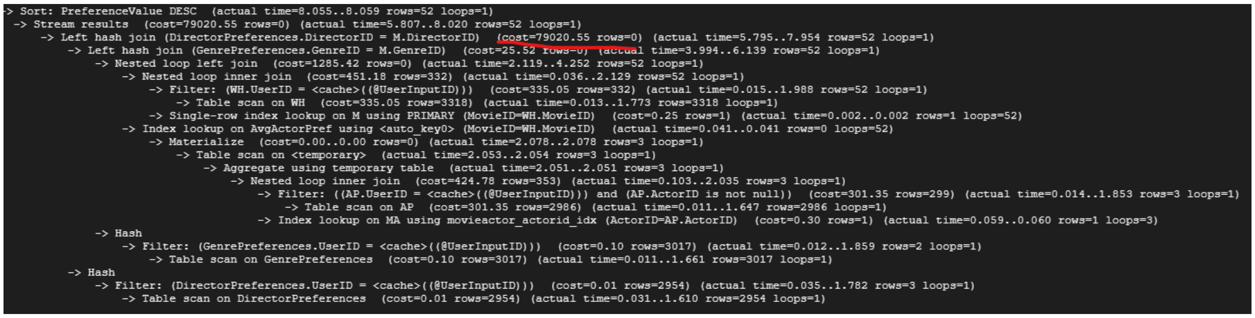


```
(DirectorPreferences.DirectorID = M.DirectorID)  (cost=28377.14
oin (GenrePreferences.GenreID = M.GenreID)  (cost=10.83 rows=0)
loop left join  (cost=332.72 rows=0) (actual time=3.313..3.593 r
ted loop inner join  (cost=33.15 rows=52) (actual time=0.050..0.
 Index lookup on WH using watchhistory_userid_idx (UserID=(@User
 Single-row index lookup on M using PRIMARY (MovieID=WH.MovieID)
```

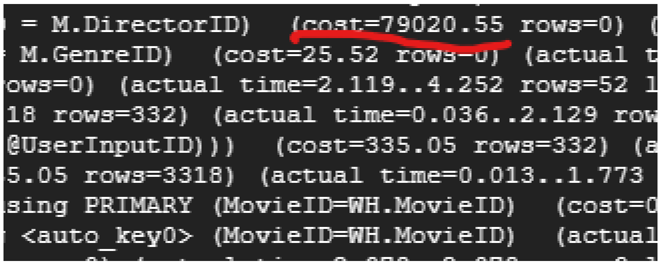
Using the same attribute to index the first advanced query, we get a significant reduction in cost. It was worth considering because it is one of the selected attributes and is not a primary key.

Indexing on WatchHistory.MovieID:
CREATE INDEX watchhistory_movieid_idx ON WatchHistory(MovieID);

Initial Cost: 181054.47
New Cost: 79020.55



```
-> Sort: PreferenceValue DESC  (actual time=8.055..8.059 rows=52 loops=1)
 -> Stream results  (cost=79020.55 rows=0) (actual time=5.807..8.020 rows=52 loops=1)
     -> Left hash join (DirectorPreferences.DirectorID = M.DirectorID)  (cost=79020.55 rows=0) (actual time=5.795..7.954 rows=52 loops=1)
         -> Left hash join (GenrePreferences.GenreID = M.GenreID)  (cost=25.52 rows=0) (actual time=3.994..6.139 rows=52 loops=1)
             -> Nested loop left join  (cost=1285.42 rows=0) (actual time=2.119..4.252 rows=52 loops=1)
                 -> Nested loop inner join  (cost=451.18 rows=332) (actual time=0.036..2.129 rows=52 loops=1)
                     -> Filter: (WH.UserID = <cache>((@UserInputID)))  (cost=335.05 rows=332) (actual time=0.015..1.988 rows=52 loops=1)
                         -> Table scan on WH  (cost=335.05 rows=3318) (actual time=0.013..1.773 rows=3318 loops=1)
                     -> Single-row index lookup on M using PRIMARY (MovieID=WH.MovieID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=52)
                 -> Index lookup on AvgActorPref using <auto_key0> (MovieID=WH.MovieID)  (actual time=0.041..0.041 rows=0 loops=52)
                     -> Materialize  (cost=0.00..0.00 rows=0) (actual time=2.078..2.078 rows=3 loops=1)
                         -> Table scan on <temporary>  (actual time=2.053..2.054 rows=3 loops=1)
                             -> Aggregate using temporary table  (actual time=2.051..2.051 rows=3 loops=1)
                                 -> Nested loop inner join  (cost=424.78 rows=353) (actual time=0.103..2.035 rows=3 loops=1)
                                     -> Filter: ((AP.UserID = <cache>((@UserInputID))) and (AP.ActorID is not null))  (cost=301.35 rows=299) (actual time=0.014..1.853 rows=3 loops=1)
                                         -> Table scan on AP  (cost=301.35 rows=2986) (actual time=0.011..1.647 rows=2986 loops=1)
                                     -> Index lookup on MA using movieactor_actorid_idx (ActorID=AP.ActorID)  (cost=0.30 rows=1) (actual time=0.059..0.060 rows=1 loops=3)
             -> Hash
                 -> Filter: (GenrePreferences.UserID = <cache>((@UserInputID)))  (cost=0.10 rows=3017) (actual time=0.012..1.859 rows=2 loops=1)
                     -> Table scan on GenrePreferences  (cost=0.10 rows=3017) (actual time=0.011..1.661 rows=3017 loops=1)
         -> Hash
             -> Filter: (DirectorPreferences.UserID = <cache>((@UserInputID)))  (cost=0.01 rows=2954) (actual time=0.035..1.782 rows=3 loops=1)
                 -> Table scan on DirectorPreferences  (cost=0.01 rows=2954) (actual time=0.031..1.610 rows=2954 loops=1)
```

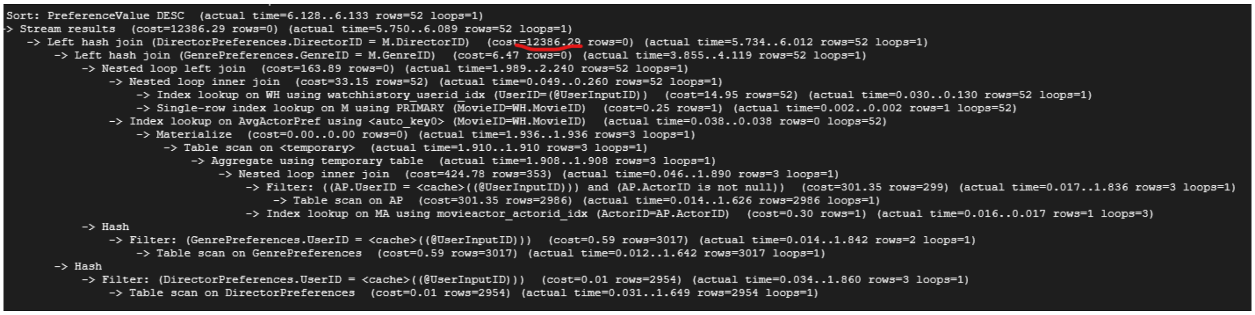


```
 = M.DirectorID)  (cost=79020.55 rows=0)  (
 M.GenreID)  (cost=25.52 rows=0)  (actual t
ows=0) (actual time=2.119..4.252 rows=52 l
18 rows=332) (actual time=0.036..2.129 row
(@UserInputID)))  (cost=335.05 rows=332) (a
5.05 rows=3318) (actual time=0.013..1.773
sing PRIMARY (MovieID=WH.MovieID)  (cost=0
 <auto_key0> (MovieID=WH.MovieID)  (actual
```

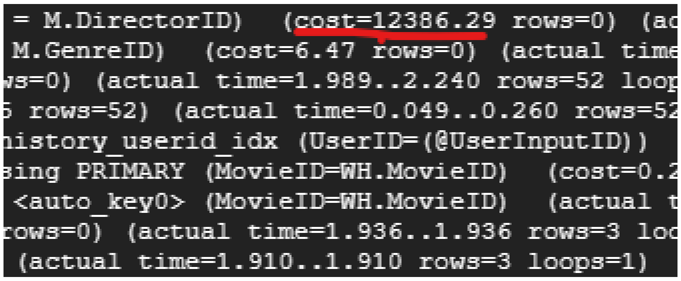
Looking at this line here:
JOIN ActorPreferences AP ON MA.ActorID = AP.ActorID AND AP.UserID = @UserInputID
I thought that MoviesActors(MA) can reduce the cost if we index beforehand so that the MA relationship can connect with the ActorPreferences(AP) relationship more efficiently.

Indexing on WatchHistory.UserID and MovieActors.ActorID:
CREATE INDEX watchhistory_userid_idx ON WatchHistory(UserID);
CREATE INDEX movieactors_actorid_idx ON MovieActors(ActorID);

Initial Cost: 181054.47
WatchHistory.UserID Cost: 28377.14
New Cost: 12386.29



```
Sort: PreferenceValue DESC  (actual time=6.128..6.133 rows=52 loops=1)
> Stream results  (cost=12386.29 rows=0) (actual time=5.750..6.089 rows=52 loops=1)
   -> Left hash join (DirectorPreferences.DirectorID = M.DirectorID)  (cost=12386.29 rows=0) (actual time=5.734..6.012 rows=52 loops=1)
       -> Left hash join (GenrePreferences.GenreID = M.GenreID)  (cost=6.47 rows=0) (actual time=3.855..4.119 rows=52 loops=1)
           -> Nested loop left join  (cost=163.89 rows=0) (actual time=1.989..2.240 rows=52 loops=1)
               -> Nested loop inner join  (cost=33.15 rows=52) (actual time=0.049..0.260 rows=52 loops=1)
                   -> Index lookup on WH using watchhistory_userid_idx (UserID=(@UserInputID))  (cost=14.95 rows=52) (actual time=0.030..0.130 rows=52 loops=1)
                   -> Single-row index lookup on M using PRIMARY (MovieID=WH.MovieID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=52)
               -> Index lookup on AvgActorPref using <auto_key0> (MovieID=WH.MovieID)  (actual time=0.038..0.038 rows=0 loops=52)
                   -> Materialize  (cost=0.00..0.00 rows=0) (actual time=1.936..1.936 rows=3 loops=1)
                       -> Table scan on <temporary>  (actual time=1.910..1.910 rows=3 loops=1)
                           -> Aggregate using temporary table  (actual time=1.908..1.908 rows=3 loops=1)
                               -> Nested loop inner join  (cost=424.78 rows=353) (actual time=0.046..1.890 rows=3 loops=1)
                                   -> Filter: ((AP.UserID = <cache>((@UserInputID))) and (AP.ActorID is not null))  (cost=301.35 rows=299) (actual time=0.017..1.836 rows=3 loops=1)
                                       -> Table scan on AP  (cost=301.35 rows=2986) (actual time=0.014..1.626 rows=2986 loops=1)
                                   -> Index lookup on MA using movieactor_actorid_idx (ActorID=AP.ActorID)  (cost=0.30 rows=1) (actual time=0.016..0.017 rows=1 loops=3)
           -> Hash
               -> Filter: (GenrePreferences.UserID = <cache>((@UserInputID)))  (cost=0.59 rows=3017) (actual time=0.014..1.842 rows=2 loops=1)
                   -> Table scan on GenrePreferences  (cost=0.59 rows=3017) (actual time=0.012..1.642 rows=3017 loops=1)
       -> Hash
           -> Filter: (DirectorPreferences.UserID = <cache>((@UserInputID)))  (cost=0.01 rows=2954) (actual time=0.034..1.860 rows=3 loops=1)
               -> Table scan on DirectorPreferences  (cost=0.01 rows=2954) (actual time=0.031..1.649 rows=2954 loops=1)
```



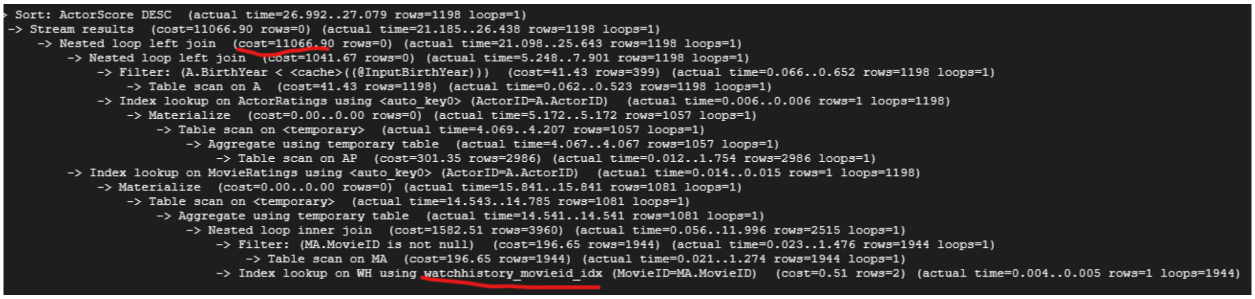
```
 = M.DirectorID)  (cost=12386.29 rows=0)  (ac
 M.GenreID)  (cost=6.47 rows=0)  (actual time
ws=0) (actual time=1.989..2.240 rows=52 loop
5 rows=52) (actual time=0.049..0.260 rows=52
history_userid_idx (UserID=(@UserInputID))
sing PRIMARY (MovieID=WH.MovieID)  (cost=0.2
 <auto_key0> (MovieID=WH.MovieID)  (actual t
rows=0) (actual time=1.936..1.936 rows=3 loo
 (actual time=1.910..1.910 rows=3 loops=1)
```

Conclusion:
2 indexing variations were found that reduces the cost. For the 3rd indexing variation, combining two indexes that affected the cost-intensive operations of the advanced queries reduced the speed even further. Therefore, the final index variation will be integrated into our database design.

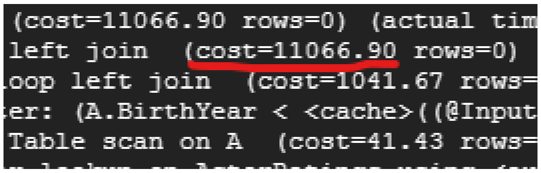
GlobalRankActors:

Indexing on WatchHistory.MovieID:
CREATE INDEX watchhistory_movieid_idx ON WatchHistory(MovieID);

Initial Cost: 539585.88
New Cost: 11066.90



```
> Sort: ActorScore DESC  (actual time=26.992..27.079 rows=1198 loops=1)
 -> Stream results  (cost=11066.90 rows=0) (actual time=21.185..26.438 rows=1198 loops=1)
     -> Nested loop left join  (cost=11066.90 rows=0) (actual time=21.098..25.643 rows=1198 loops=1)
         -> Nested loop left join  (cost=1041.67 rows=0) (actual time=5.248..7.901 rows=1198 loops=1)
             -> Filter: (A.BirthYear < <cache>((@InputBirthYear)))  (cost=41.43 rows=399) (actual time=0.066..0.652 rows=1198 loops=1)
                 -> Table scan on A  (cost=41.43 rows=1198) (actual time=0.062..0.523 rows=1198 loops=1)
             -> Index lookup on ActorRatings using <auto_key0> (ActorID=A.ActorID)  (actual time=0.006..0.006 rows=1 loops=1198)
                 -> Materialize  (cost=0.00..0.00 rows=0) (actual time=5.172..5.172 rows=1057 loops=1)
                     -> Table scan on <temporary>  (actual time=4.069..4.207 rows=1057 loops=1)
                         -> Aggregate using temporary table  (actual time=4.067..4.067 rows=1057 loops=1)
                             -> Table scan on AP  (cost=301.35 rows=2986) (actual time=0.012..1.754 rows=2986 loops=1)
         -> Index lookup on MovieRatings using <auto_key0> (ActorID=A.ActorID)  (actual time=0.014..0.015 rows=1 loops=1198)
             -> Materialize  (cost=0.00..0.00 rows=0) (actual time=15.841..15.841 rows=1081 loops=1)
                 -> Table scan on <temporary>  (actual time=14.543..14.785 rows=1081 loops=1)
                     -> Aggregate using temporary table  (actual time=14.541..14.541 rows=1081 loops=1)
                         -> Nested loop inner join  (cost=1582.51 rows=3960) (actual time=0.056..11.996 rows=2515 loops=1)
                             -> Filter: (MA.MovieID is not null)  (cost=196.65 rows=1944) (actual time=0.023..1.476 rows=1944 loops=1)
                                 -> Table scan on MA  (cost=196.65 rows=1944) (actual time=0.021..1.274 rows=1944 loops=1)
                             -> Index lookup on WH using watchhistory_movieid_idx (MovieID=MA.MovieID)  (cost=0.51 rows=2) (actual time=0.004..0.005 rows=1 loops=1944)
```

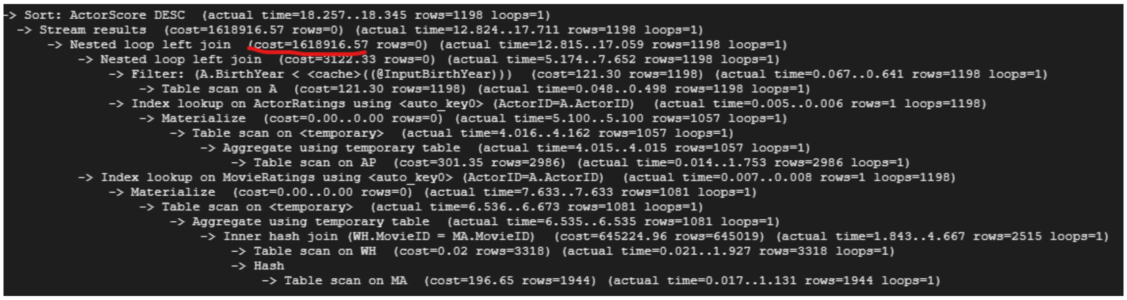


```
 (cost=11066.90 rows=0) (actual tim
 left join  (cost=11066.90 rows=0)
.oop left join  (cost=1041.67 rows=
er: (A.BirthYear < <cache>((@Input
 Table scan on A  (cost=41.43 rows=
```

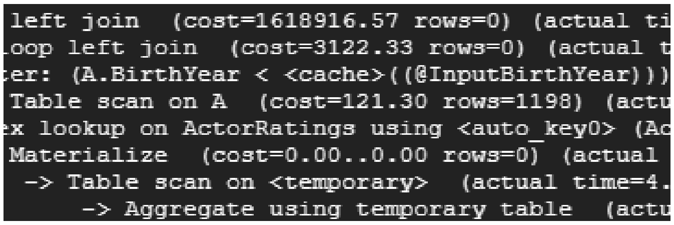
This index was left from an attempt to reduce costs on a different query. There was initial confusion for cost reduction until the last line revealed there was still an index that was used by the query. It makes sense why this one works, it is on a join line contained within a LEFT JOIN subquery. Indexing this will make it a lot easier to fetch values for the subquery.

Indexing on Actors.BirthYear:
CREATE INDEX actors_birthyear_idx ON Actors(BirthYear);

Initial Cost: 539585.88
New Cost: 1618916.57



```
-> Sort: ActorScore DESC  (actual time=18.257..18.345 rows=1198 loops=1)
 -> Stream results  (cost=1618916.57 rows=0) (actual time=12.824..17.711 rows=1198 loops=1)
     -> Nested loop left join  (cost=1618916.57 rows=0) (actual time=12.815..17.059 rows=1198 loops=1)
         -> Nested loop left join  (cost=3122.33 rows=0) (actual time=5.174..7.652 rows=1198 loops=1)
             -> Filter: (A.BirthYear < <cache>((@InputBirthYear)))  (cost=121.30 rows=1198) (actual time=0.067..0.641 rows=1198 loops=1)
                 -> Table scan on A  (cost=121.30 rows=1198) (actual time=0.048..0.498 rows=1198 loops=1)
             -> Index lookup on ActorRatings using <auto_key0> (ActorID=A.ActorID)  (actual time=0.005..0.006 rows=1 loops=1198)
                 -> Materialize  (cost=0.00..0.00 rows=0) (actual time=5.100..5.100 rows=1057 loops=1)
                     -> Table scan on <temporary>  (actual time=4.016..4.162 rows=1057 loops=1)
                         -> Aggregate using temporary table  (actual time=4.015..4.015 rows=1057 loops=1)
                             -> Table scan on AP  (cost=301.35 rows=2986) (actual time=0.014..1.753 rows=2986 loops=1)
         -> Index lookup on MovieRatings using <auto_key0> (ActorID=A.ActorID)  (actual time=0.007..0.008 rows=1 loops=1198)
             -> Materialize  (cost=0.00..0.00 rows=0) (actual time=7.633..7.633 rows=1081 loops=1)
                 -> Table scan on <temporary>  (actual time=6.536..6.673 rows=1081 loops=1)
                     -> Aggregate using temporary table  (actual time=6.535..6.535 rows=1081 loops=1)
                         -> Inner hash join (WH.MovieID = MA.MovieID)  (cost=645224.96 rows=645019) (actual time=1.843..4.667 rows=2515 loops=1)
                             -> Table scan on WH  (cost=0.02 rows=3318) (actual time=0.021..1.927 rows=3318 loops=1)
                             -> Hash
                                 -> Table scan on MA  (cost=196.65 rows=1944) (actual time=0.017..1.131 rows=1944 loops=1)
```

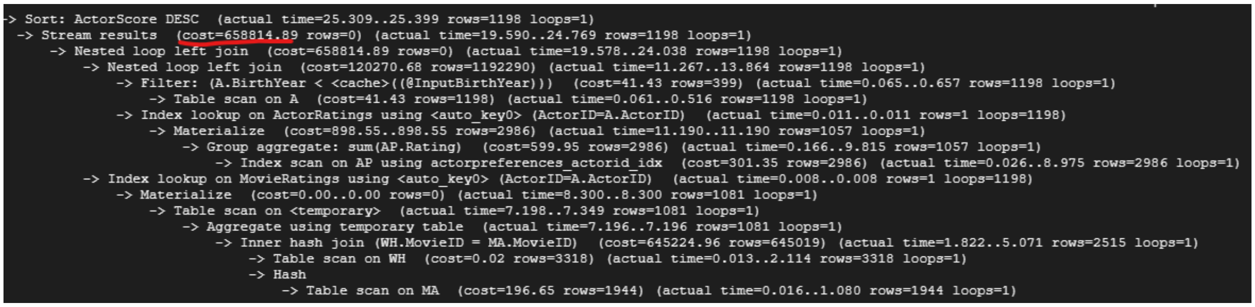


```
 left join  (cost=1618916.57 rows=0)  (actual ti
.oop left join  (cost=3122.33 rows=0)  (actual t
:er: (A.BirthYear < <cache>((@InputBirthYear)))
 Table scan on A  (cost=121.30 rows=1198)  (actu
:x lookup on ActorRatings using <auto_key0> (Ac
 Materialize  (cost=0.00..0.00 rows=0)  (actual
  -> Table scan on <temporary>  (actual time=4.
      -> Aggregate using temporary table  (actu
```

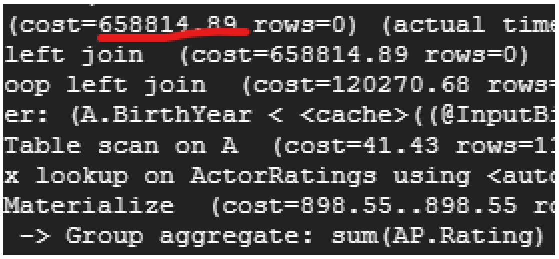
Trying to index a cached attribute seems to backfire. It would be wiser to better consider interfering with the cache even if the attribute is in a WHERE clause.

Indexing on ActorPreferences.ActorID:
CREATE INDEX actorpreferences_actorid_idx ON ActorPreferences(ActorID);

Initial Cost: 539585.88
New Cost: 658814.89

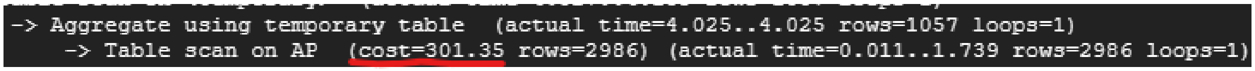


```
-> Sort: ActorScore DESC  (actual time=25.309..25.399 rows=1198 loops=1)
 -> Stream results  (cost=658814.89 rows=0) (actual time=19.590..24.769 rows=1198 loops=1)
     -> Nested loop left join  (cost=658814.89 rows=0) (actual time=19.578..24.038 rows=1198 loops=1)
         -> Nested loop left join  (cost=120270.68 rows=1192290) (actual time=11.267..13.864 rows=1198 loops=1)
             -> Filter: (A.BirthYear < <cache>((@InputBirthYear)))  (cost=41.43 rows=399) (actual time=0.065..0.657 rows=1198 loops=1)
                 -> Table scan on A  (cost=41.43 rows=1198) (actual time=0.061..0.516 rows=1198 loops=1)
             -> Index lookup on ActorRatings using <auto_key0> (ActorID=A.ActorID)  (actual time=0.011..0.011 rows=1 loops=1198)
                 -> Materialize  (cost=898.55..898.55 rows=2986) (actual time=11.190..11.190 rows=1057 loops=1)
                     -> Group aggregate: sum(AP.Rating)  (cost=599.95 rows=2986) (actual time=0.166..9.815 rows=1057 loops=1)
                         -> Index scan on AP using actorpreferences_actorid_idx  (cost=301.35 rows=2986) (actual time=0.026..8.975 rows=2986 loops=1)
         -> Index lookup on MovieRatings using <auto_key0> (ActorID=A.ActorID)  (actual time=0.008..0.008 rows=1 loops=1198)
             -> Materialize  (cost=0.00..0.00 rows=0) (actual time=8.300..8.300 rows=1081 loops=1)
                 -> Table scan on <temporary>  (actual time=7.198..7.349 rows=1081 loops=1)
                     -> Aggregate using temporary table  (actual time=7.196..7.196 rows=1081 loops=1)
                         -> Inner hash join (WH.MovieID = MA.MovieID)  (cost=645224.96 rows=645019) (actual time=1.822..5.071 rows=2515 loops=1)
                             -> Table scan on WH  (cost=0.02 rows=3318) (actual time=0.013..2.114 rows=3318 loops=1)
                             -> Hash
                                 -> Table scan on MA  (cost=196.65 rows=1944) (actual time=0.016..1.080 rows=1944 loops=1)
```



```
(cost=658814.89 rows=0)  (actual time
left join  (cost=658814.89 rows=0)
oop left join  (cost=120270.68 rows=
er: (A.BirthYear < <cache>((@InputBi
Table scan on A  (cost=41.43 rows=11
x lookup on ActorRatings using <autc
Materialize  (cost=898.55..898.55 rc
 -> Group aggregate: sum(AP.Rating)
```

Analyzing the runtime of the initial subquery with no indexes shows that there is a bottle neck on the AP table:



```
-> Aggregate using temporary table  (actual time=4.025..4.025 rows=1057 loops=1)
    -> Table scan on AP  (cost=301.35 rows=2986) (actual time=0.011..1.739 rows=2986 loops=1)
```

Trying to reduce the cost at this point also backfired. It makes sense as its initial table scan is several magnitudes cheaper than the first successful index's table cost. So, the tradeoff for the overhead of a new index does not outweigh the reduction of that cost using the indexed attribute since there is only so much cost to reduce to begin with.

Conclusions:

Indexing on WatchHistory.MovieID drastically reduced the baseline cost of the advanced query from 539585.88 to 11066.90, so this will be integrated into our query design. This most likely means that a critical point of efficiency involves joining between WatchHistory and Movies on MovieID, where there are a lot of data values involved. The other variations increased the cost drastically instead, meaning that indexing was actually detrimental compared to operating with full table scans.