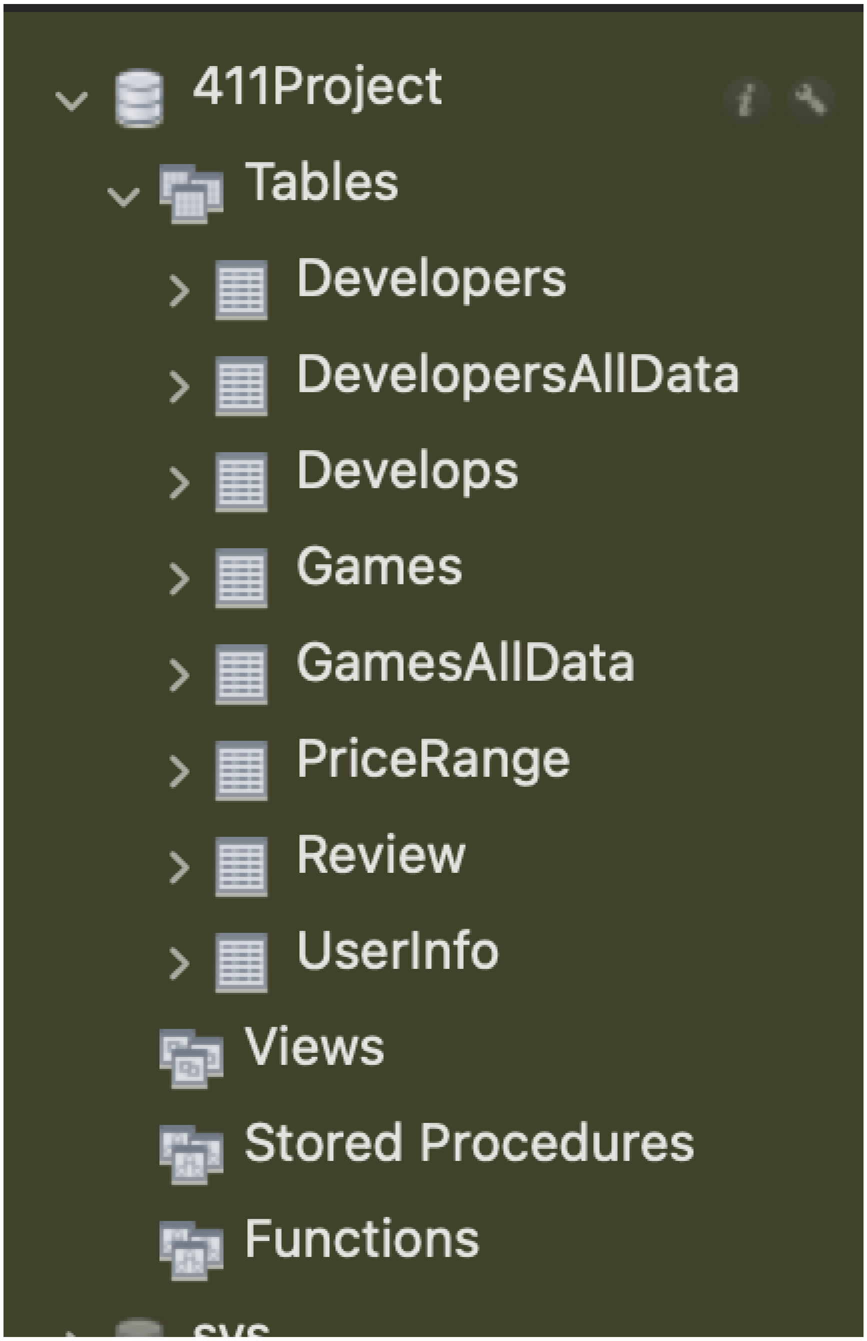
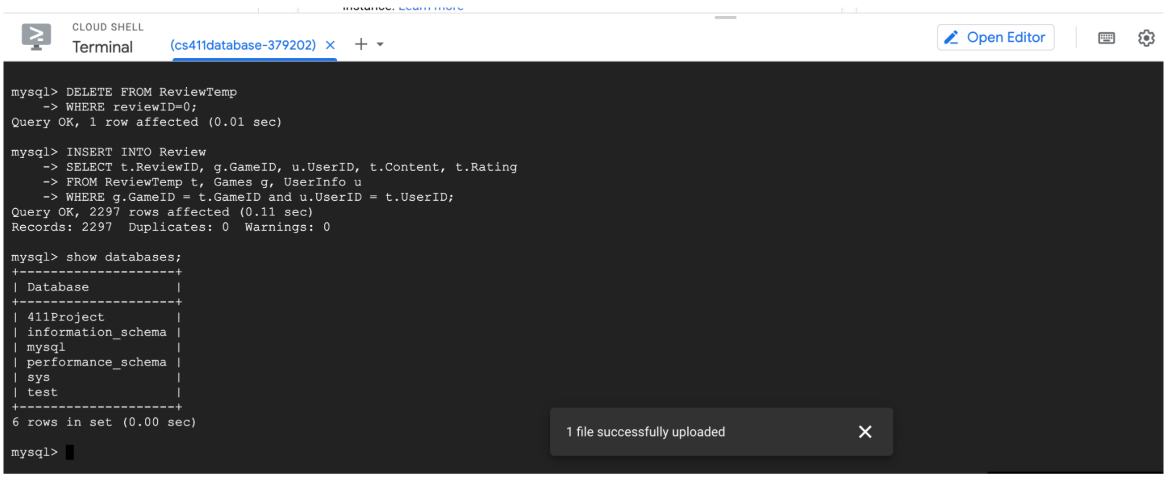
1. Implement at least 4 main tables
   We implemented 6 tables: Developers, Develops, Games, PriceRange, Review, and UserInfo. DevelopersAllData and GamesAllData are helper tables that we created when uploading the csv file to the database, and we will not be using these two tables in our final product, so they can be ignored.



411Project
Tables
Developers
DevelopersAllData
Develops
Games
GamesAllData
PriceRange
Review
UserInfo
Views
Stored Procedures
Functions
sys

Here is a picture of the connection to GCP:

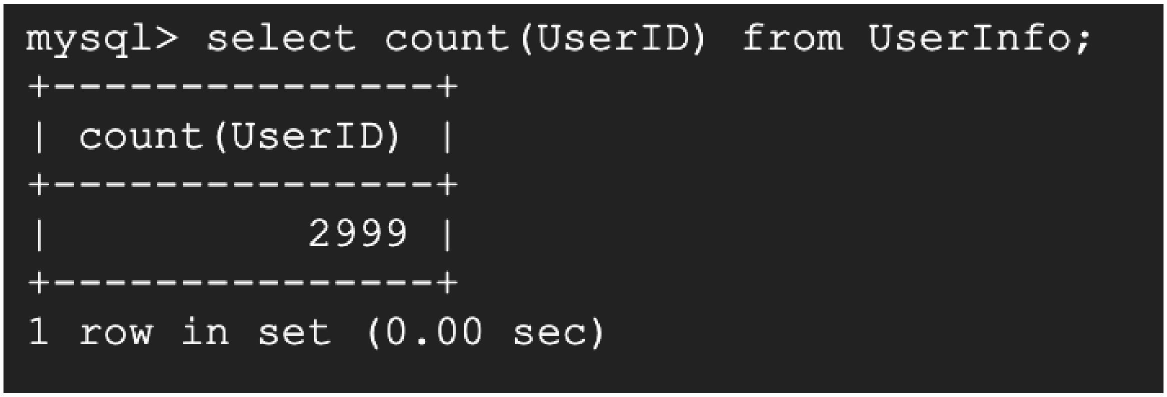


```
mysql> DELETE FROM ReviewTemp
    -> WHERE reviewID=0;
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO Review
    -> SELECT t.ReviewID, g.GameID, u.UserID, t.Content, t.Rating
    -> FROM ReviewTemp t, Games g, UserInfo u
    -> WHERE g.GameID = t.GameID and u.UserID = t.UserID;
Query OK, 2297 rows affected (0.11 sec)
Records: 2297  Duplicates: 0  Warnings: 0

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| 411Project         |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| test               |
+--------------------+
6 rows in set (0.00 sec)

mysql> 
```

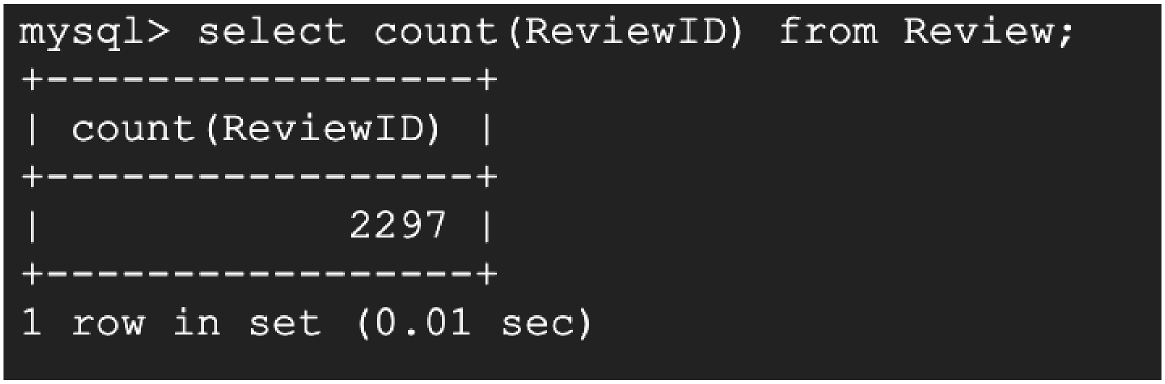
2. In the Database Design markdown or pdf, provide the Data Definition Language (DDL) commands you all used to create each of these tables in the database.

```
CREATE TABLE Games
(
    GameID INT primary key,
    ResponseName   varchar(256),
    ReleaseDate   varchar(100),
    RecommendationCount    INT,
    AchievementCount    int,
    ControllerSupport     varchar(10),
    IsFree    varchar(10),
    FreeVerAvail    varchar(10),
    PurchaseAvail    varchar(10),
    SubscriptionAvail    varchar(10),
    PlatformWindows    varchar(10),
    PlatformLinux    varchar(10),
    PlatformMac    varchar(10),
    CategorySinglePlayer    varchar(10),
    CategoryMultiplayer    varchar(10),
    CategoryCoop    varchar(10),
    CategoryMMO    varchar(10),
    CategoryInAppPurchase    varchar(10),
    CategoryIncludeSrcSDK    varchar(10),
    CategoryIncludeLevelEditor    varchar(10),
    CategoryVRSupport    varchar(10),
    GenreIsNonGame    varchar(10),
    GenreIsIndie    varchar(10),
    GenreIsAction    varchar(10),
    GenreIsAdventure    varchar(10),
    GenreIsCasual    varchar(10),
    GenreIsStrategy    varchar(10),
    GenreIsRPG    varchar(10),
    GenreIsSimulation    varchar(10),
    GenreIsEarlyAccess    varchar(10),
    GenreIsFreeToPlay    varchar(10),
    GenreIsSports    varchar(10),
    GenreIsRacing    varchar(10),
    GenreIsMassivelyMultiplayer    varchar(10),
    PriceFinal    DOUBLE,
    SupportedLanguages    varchar(512)
);


CREATE TABLE Developers
(
    DeveloperID   INT not null Primary Key,
    Developer   varchar(256),
    Country    varchar(100),
    Notes    varchar(300)
);

CREATE TABLE Develops (
    GameID int NOT NULL,
    DeveloperID int NOT NULL,
    PRIMARY KEY (GameID, DeveloperID),
    FOREIGN KEY (DeveloperID) REFERENCES Developers (DeveloperID),
    FOREIGN KEY (GameID) REFERENCES Games (GameID)
);

ALTER TABLE Develops
ADD CONSTRAINT FK_Develops_Games
FOREIGN KEY (GameID)
REFERENCES Games(GameID)
ON DELETE CASCADE;

ALTER TABLE Develops
ADD CONSTRAINT FK_Develops_Developers
FOREIGN KEY (DeveloperID)
REFERENCES Developers(DeveloperID)
ON DELETE CASCADE;

Create Table UserInfo(
        UserID INT not null Primary Key,
        Email VARCHAR(100),
        Username VARCHAR(20),
        Password VARCHAR(40),
        DeviceBackground VARCHAR(10)
);

Create Table Review(
        ReviewID int not null primary key,
        GameID int,
        UserID int,
        Content VARCHAR(1000),
        Rating INT,
        Foreign key (GameID) references Games(GameID),
        Foreign key (UserID) references UserInfo(UserID)
);

Create Table PriceRange(
        Grade int not null primary key,
        LowerPrice INT,
        UpperPrice INT
);
```

3. Insert data into these tables. You should insert at least 1000 rows each in three of the tables. Try to use real data, but if you cannot find a good dataset for a particular table, you may use auto-generated data.
   We have inserted data to all the tables that we have implemented.
   Among these data, UserInfo and UserReviews are auto generated.

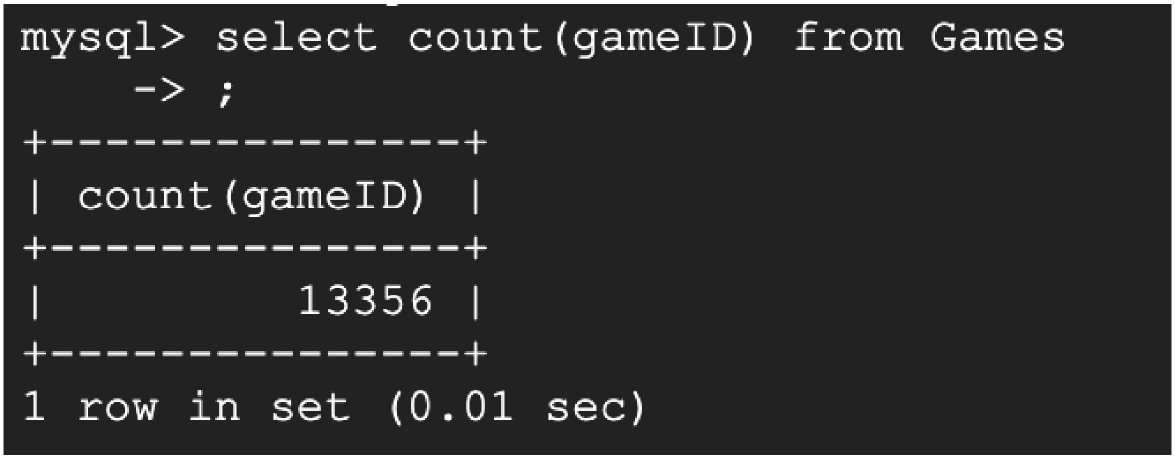


```
mysql> select count(UserID) from UserInfo;
+---------------+
| count(UserID) |
+---------------+
|          2999 |
+---------------+
1 row in set (0.00 sec)
```



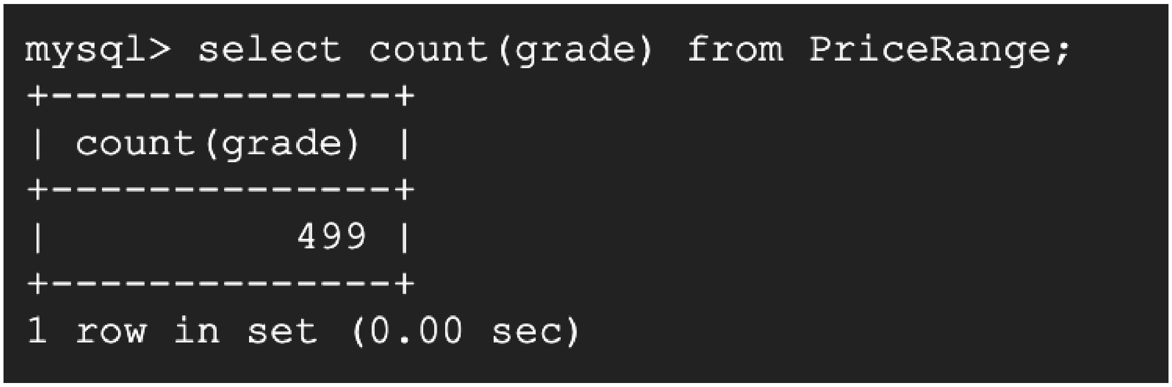
```
mysql> select count(ReviewID) from Review;
+-----------------+
| count(ReviewID) |
+-----------------+
|            2297 |
+-----------------+
1 row in set (0.01 sec)
```

Games table is populated with selected columns of our original dataset.



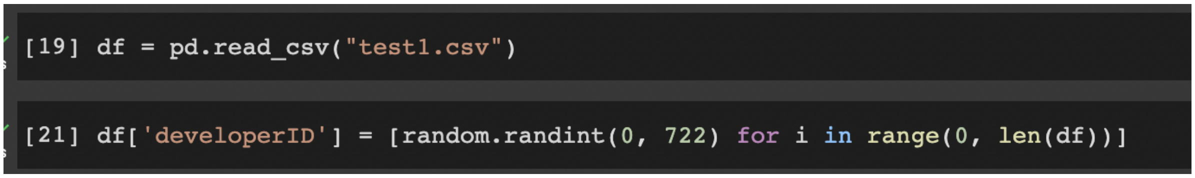
```
mysql> select count(gameID) from Games
    -> ;
+---------------+
| count(gameID) |
+---------------+
|         13356 |
+---------------+
1 row in set (0.01 sec)
```

For Price Range, we divide price ranges into different grades, every 5 dollar is a different grade, up to $2500, which is a little more than the cost of the most expensive game on steam.



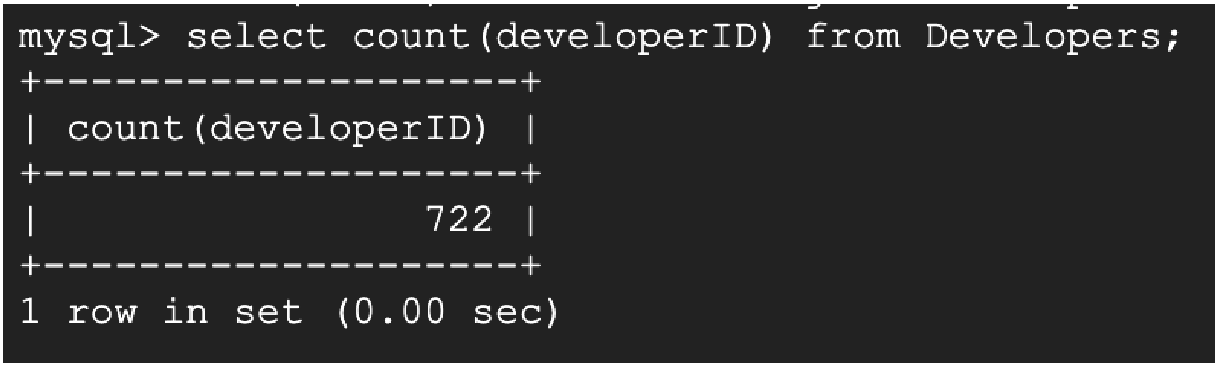
```
mysql> select count(grade) from PriceRange;
+--------------+
| count(grade) |
+--------------+
|          499 |
+--------------+
1 row in set (0.00 sec)
```

For Developers, we could not find any datasets or tools that find game developers for us. Therefore, we pulled a list of game developers (companies) from wikipedia including 700+ different companies. We then randomly assigned each game with a random game developer using the following python code:



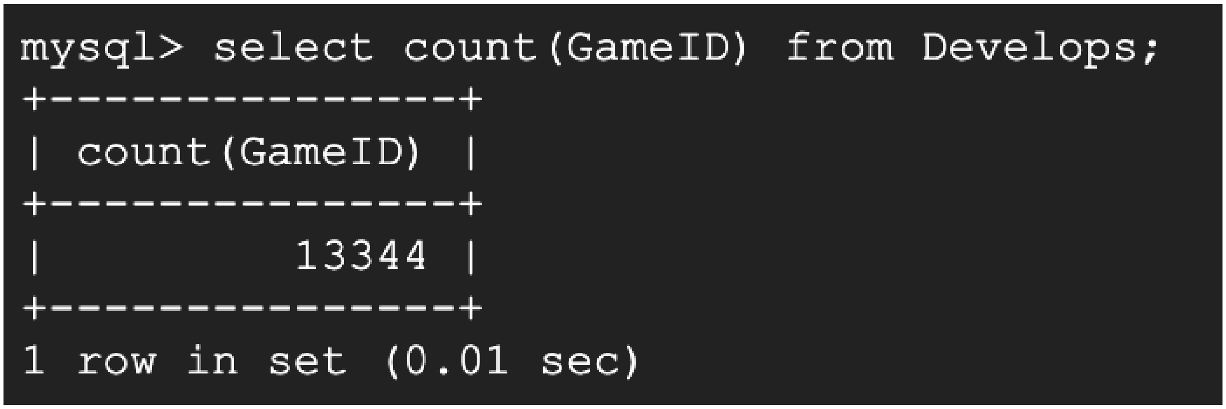
```
[19] df = pd.read_csv("test1.csv")

[21] df['developerID'] = [random.randint(0, 722) for i in range(0, len(df))]
```

We will be clarifying this in our final product to make sure no confusion arises.



```
mysql> select count(developerID) from Developers;
+--------------------+
| count(developerID) |
+--------------------+
|                722 |
+--------------------+
1 row in set (0.00 sec)
```

We also implemented one relation: develops relation. In this relation, each game has a developer. Here is a count of the number of games in the develops relation (should be equal to the number of games):



```
mysql> select count(GameID) from Develops;
+---------------+
| count(GameID) |
+---------------+
|         13344 |
+---------------+
1 row in set (0.01 sec)
```

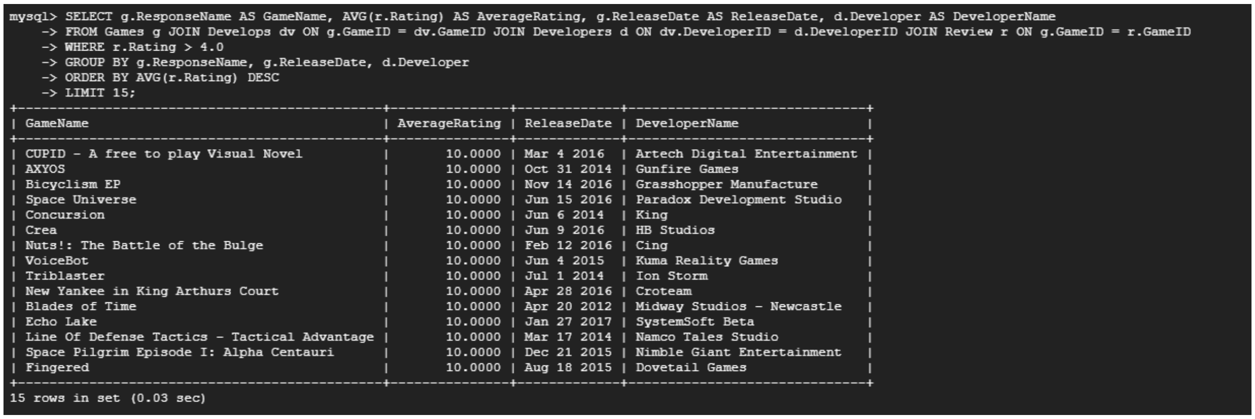
4. As a group, develop two advanced SQL queries related to the project that are different from one another. The two advance queries are expected to be part of your final application. The queries should each involve at least two of the following SQL concepts: Join of multiple relations, Set operations, Aggregation via GROUP BY, Subqueries
5. Execute your advanced SQL queries and provide a screenshot of the top 15 rows of each query result (you can use the LIMIT clause to select the top 15 rows). If your output is less than 15 rows, say that in your output.

Query 1

SELECT g.ResponseName AS GameName, AVG(r.Rating) AS AverageRating, g.ReleaseDate AS ReleaseDate, d.Developer AS DeveloperName
FROM Games g JOIN Develops dv ON g.GameID = dv.GameID JOIN Developers d ON dv.DeveloperID = d.DeveloperID JOIN Review r ON g.GameID = r.GameID
WHERE r.Rating > 4.0
GROUP BY g.ResponseName, g.ReleaseDate, d.Developer

```
ORDER BY AVG(r.Rating) DESC;
```

This query used Join of multiple relations and Aggregation via GROUP BY.
This query gets the list of top-rated games along with their average rating, release date, and the names of the developers who developed them, for all games that have a rating greater than 4.0.
At the end of the query, we added LIMIT 15 to only show the top 15 results.

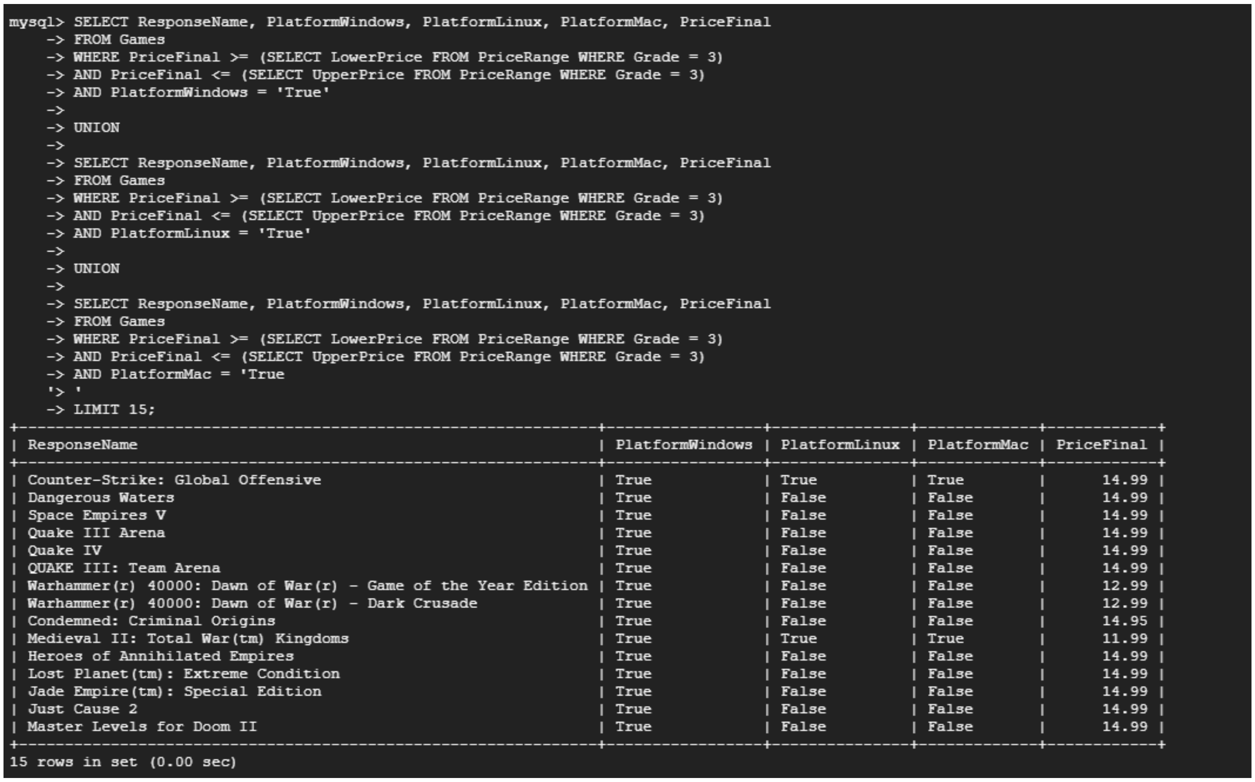


```
mysql> SELECT g.ResponseName AS GameName, AVG(r.Rating) AS AverageRating, g.ReleaseDate AS ReleaseDate, d.Developer AS DeveloperName
    -> FROM Games g JOIN Develops dv ON g.GameID = dv.GameID JOIN Developers d ON dv.DeveloperID = d.DeveloperID JOIN Review r ON g.GameID = r.GameID
    -> WHERE r.Rating > 4.0
    -> GROUP BY g.ResponseName, g.ReleaseDate, d.Developer
    -> ORDER BY AVG(r.Rating) DESC
    -> LIMIT 15;
+----------------------------------------------+---------------+-------------+------------------------------+
| GameName                                     | AverageRating | ReleaseDate | DeveloperName                |
+----------------------------------------------+---------------+-------------+------------------------------+
| CUPID - A free to play Visual Novel          |       10.0000 | Mar 4 2016  | Artech Digital Entertainment |
| AXYOS                                        |       10.0000 | Oct 31 2014 | Gunfire Games                |
| Bicyclism EP                                 |       10.0000 | Nov 14 2016 | Grasshopper Manufacture      |
| Space Universe                               |       10.0000 | Jun 15 2016 | Paradox Development Studio   |
| Concursion                                   |       10.0000 | Jun 6 2014  | King                         |
| Crea                                         |       10.0000 | Jun 9 2016  | HB Studios                   |
| Nuts!: The Battle of the Bulge               |       10.0000 | Feb 12 2016 | Cing                         |
| VoiceBot                                     |       10.0000 | Jun 4 2015  | Kuma Reality Games           |
| Triblaster                                   |       10.0000 | Jul 1 2014  | Ion Storm                    |
| New Yankee in King Arthurs Court             |       10.0000 | Apr 28 2016 | Croteam                      |
| Blades of Time                               |       10.0000 | Apr 20 2012 | Midway Studios - Newcastle   |
| Echo Lake                                    |       10.0000 | Jan 27 2017 | SystemSoft Beta              |
| Line Of Defense Tactics - Tactical Advantage |       10.0000 | Mar 17 2014 | Namco Tales Studio           |
| Space Pilgrim Episode I: Alpha Centauri      |       10.0000 | Dec 21 2015 | Nimble Giant Entertainment   |
| Fingered                                     |       10.0000 | Aug 18 2015 | Dovetail Games               |
+----------------------------------------------+---------------+-------------+------------------------------+
15 rows in set (0.03 sec)
```

Query 2

```
SELECT ResponseName, PlatformWindows, PlatformLinux, PlatformMac, PriceFinal
FROM Games
WHERE PriceFinal >= (SELECT LowerPrice FROM PriceRange WHERE Grade = 3)
AND PriceFinal <= (SELECT UpperPrice FROM PriceRange WHERE Grade = 3)
AND PlatformWindows = 'True'
UNION
SELECT ResponseName, PlatformWindows, PlatformLinux, PlatformMac, PriceFinal
FROM Games
WHERE PriceFinal >= (SELECT LowerPrice FROM PriceRange WHERE Grade = 3)
AND PriceFinal <= (SELECT UpperPrice FROM PriceRange WHERE Grade = 3)
AND PlatformLinux = 'True'
UNION
SELECT ResponseName, PlatformWindows, PlatformLinux, PlatformMac, PriceFinal
FROM Games
WHERE PriceFinal >= (SELECT LowerPrice FROM PriceRange WHERE Grade = 3)
AND PriceFinal <= (SELECT UpperPrice FROM PriceRange WHERE Grade = 3)
AND PlatformMac = 'True';
```

This query uses subqueries set operations.
This query find the game name, platform (Windows, Linux, or Mac), and price, for all games with a price within the specified range (in this case, grade 3) for any of the three platforms.
At the end of the query, we added LIMIT 15 to only show the top 15 results.

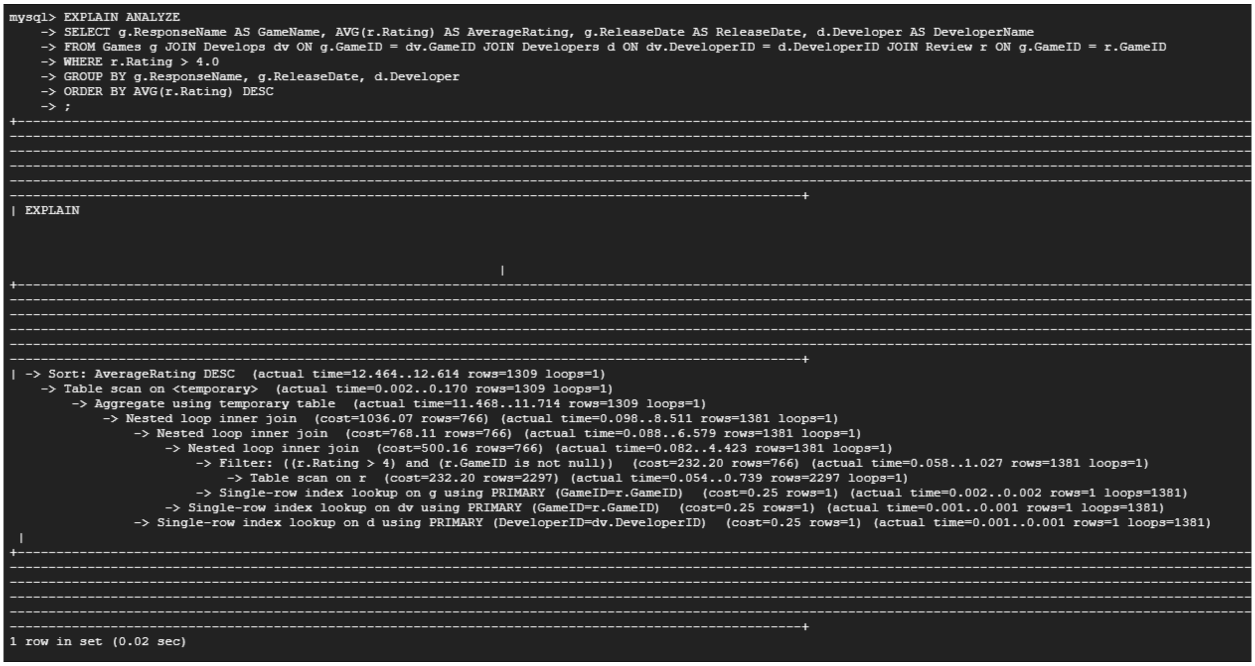


```
mysql> SELECT ResponseName, PlatformWindows, PlatformLinux, PlatformMac, PriceFinal
    -> FROM Games
    -> WHERE PriceFinal >= (SELECT LowerPrice FROM PriceRange WHERE Grade = 3)
    -> AND PriceFinal <= (SELECT UpperPrice FROM PriceRange WHERE Grade = 3)
    -> AND PlatformWindows = 'True'
    ->
    -> UNION
    ->
    -> SELECT ResponseName, PlatformWindows, PlatformLinux, PlatformMac, PriceFinal
    -> FROM Games
    -> WHERE PriceFinal >= (SELECT LowerPrice FROM PriceRange WHERE Grade = 3)
    -> AND PriceFinal <= (SELECT UpperPrice FROM PriceRange WHERE Grade = 3)
    -> AND PlatformLinux = 'True'
    ->
    -> UNION
    ->
    -> SELECT ResponseName, PlatformWindows, PlatformLinux, PlatformMac, PriceFinal
    -> FROM Games
    -> WHERE PriceFinal >= (SELECT LowerPrice FROM PriceRange WHERE Grade = 3)
    -> AND PriceFinal <= (SELECT UpperPrice FROM PriceRange WHERE Grade = 3)
    -> AND PlatformMac = 'True
    '> '
    -> LIMIT 15;
+---------------------------------------------------------------+-----------------+---------------+-------------+------------+
| ResponseName                                                  | PlatformWindows | PlatformLinux | PlatformMac | PriceFinal |
+---------------------------------------------------------------+-----------------+---------------+-------------+------------+
| Counter-Strike: Global Offensive                              | True            | True          | True        |      14.99 |
| Dangerous Waters                                              | True            | False         | False       |      14.99 |
| Space Empires V                                               | True            | False         | False       |      14.99 |
| Quake III Arena                                               | True            | False         | False       |      14.99 |
| Quake IV                                                      | True            | False         | False       |      14.99 |
| QUAKE III: Team Arena                                         | True            | False         | False       |      14.99 |
| Warhammer(r) 40000: Dawn of War(r) - Game of the Year Edition | True            | False         | False       |      12.99 |
| Warhammer(r) 40000: Dawn of War(r) - Dark Crusade             | True            | False         | False       |      12.99 |
| Condemned: Criminal Origins                                   | True            | False         | False       |      14.95 |
| Medieval II: Total War(tm) Kingdoms                           | True            | True          | True        |      11.99 |
| Heroes of Annihilated Empires                                 | True            | False         | False       |      14.99 |
| Lost Planet(tm): Extreme Condition                            | True            | False         | False       |      14.99 |
| Jade Empire(tm): Special Edition                              | True            | False         | False       |      14.99 |
| Just Cause 2                                                  | True            | False         | False       |      14.99 |
| Master Levels for Doom II                                     | True            | False         | False       |      14.99 |
+---------------------------------------------------------------+-----------------+---------------+-------------+------------+
15 rows in set (0.00 sec)
```

6. Indexing: As a team, for each advanced query:
   Use the EXPLAIN ANALYZE command to measure your advanced query performance before adding indexes.
   Explore adding different indices to different attributes on the advanced query. For each indexing design you try, use the EXPLAIN ANALYZE command to measure the query performance after adding the indices.
   Report on the index design you all select and explain why you chose it, referencing the analysis you performed in (b).
   Note that if you did not find any difference in your results, report that as well. Explain why you think this change in indexing did not bring a better effect to your query.

Query 1

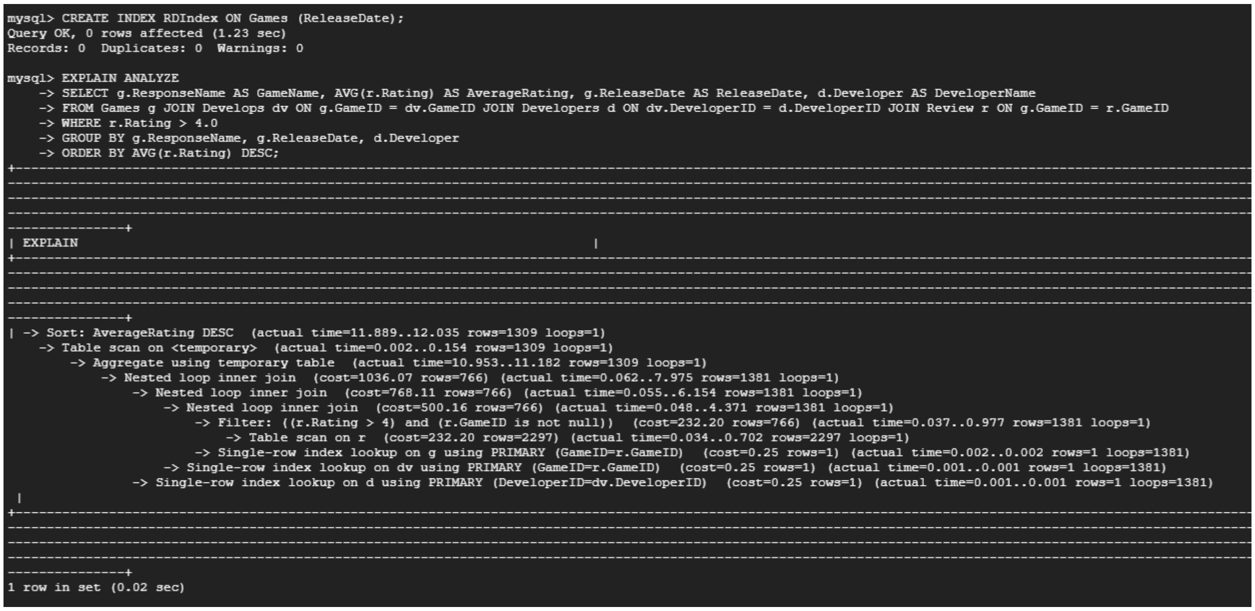
Without Index:



```
mysql> EXPLAIN ANALYZE
    -> SELECT g.ResponseName AS GameName, AVG(r.Rating) AS AverageRating, g.ReleaseDate AS ReleaseDate, d.Developer AS DeveloperName
    -> FROM Games g JOIN Develops dv ON g.GameID = dv.GameID JOIN Developers d ON dv.DeveloperID = d.DeveloperID JOIN Review r ON g.GameID = r.GameID
    -> WHERE r.Rating > 4.0
    -> GROUP BY g.ResponseName, g.ReleaseDate, d.Developer
    -> ORDER BY AVG(r.Rating) DESC
    -> ;
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------+
| EXPLAIN



                                                                 |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------+
| -> Sort: AverageRating DESC  (actual time=12.464..12.614 rows=1309 loops=1)
    -> Table scan on <temporary>  (actual time=0.002..0.170 rows=1309 loops=1)
        -> Aggregate using temporary table  (actual time=11.468..11.714 rows=1309 loops=1)
            -> Nested loop inner join  (cost=1036.07 rows=766) (actual time=0.098..8.511 rows=1381 loops=1)
                -> Nested loop inner join  (cost=768.11 rows=766) (actual time=0.088..6.579 rows=1381 loops=1)
                    -> Nested loop inner join  (cost=500.16 rows=766) (actual time=0.082..4.423 rows=1381 loops=1)
                        -> Filter: ((r.Rating > 4) and (r.GameID is not null))  (cost=232.20 rows=766) (actual time=0.058..1.027 rows=1381 loops=1)
                            -> Table scan on r  (cost=232.20 rows=2297) (actual time=0.054..0.739 rows=2297 loops=1)
                        -> Single-row index lookup on g using PRIMARY (GameID=r.GameID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1381)
                    -> Single-row index lookup on dv using PRIMARY (GameID=r.GameID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1381)
                -> Single-row index lookup on d using PRIMARY (DeveloperID=dv.DeveloperID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1381)
 |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------+
1 row in set (0.02 sec)
```

Index 1:

CREATE INDEX RDIndex ON Games (ReleaseDate);



```
mysql> CREATE INDEX RDIndex ON Games (ReleaseDate);
Query OK, 0 rows affected (1.23 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE
    -> SELECT g.ResponseName AS GameName, AVG(r.Rating) AS AverageRating, g.ReleaseDate AS ReleaseDate, d.Developer AS DeveloperName
    -> FROM Games g JOIN Develops dv ON g.GameID = dv.GameID JOIN Developers d ON dv.DeveloperID = d.DeveloperID JOIN Review r ON g.GameID = r.GameID
    -> WHERE r.Rating > 4.0
    -> GROUP BY g.ResponseName, g.ReleaseDate, d.Developer
    -> ORDER BY AVG(r.Rating) DESC;
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------+
| EXPLAIN                                                                  |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------+
| -> Sort: AverageRating DESC  (actual time=11.889..12.035 rows=1309 loops=1)
    -> Table scan on <temporary>  (actual time=0.002..0.154 rows=1309 loops=1)
        -> Aggregate using temporary table  (actual time=10.953..11.182 rows=1309 loops=1)
            -> Nested loop inner join  (cost=1036.07 rows=766) (actual time=0.062..7.975 rows=1381 loops=1)
                -> Nested loop inner join  (cost=768.11 rows=766) (actual time=0.055..6.154 rows=1381 loops=1)
                    -> Nested loop inner join  (cost=500.16 rows=766) (actual time=0.048..4.371 rows=1381 loops=1)
                        -> Filter: ((r.Rating > 4) and (r.GameID is not null))  (cost=232.20 rows=766) (actual time=0.037..0.977 rows=1381 loops=1)
                            -> Table scan on r  (cost=232.20 rows=2297) (actual time=0.034..0.702 rows=2297 loops=1)
                        -> Single-row index lookup on g using PRIMARY (GameID=r.GameID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1381)
                    -> Single-row index lookup on dv using PRIMARY (GameID=r.GameID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1381)
                -> Single-row index lookup on d using PRIMARY (DeveloperID=dv.DeveloperID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1381)
 |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------+
1 row in set (0.02 sec)
```

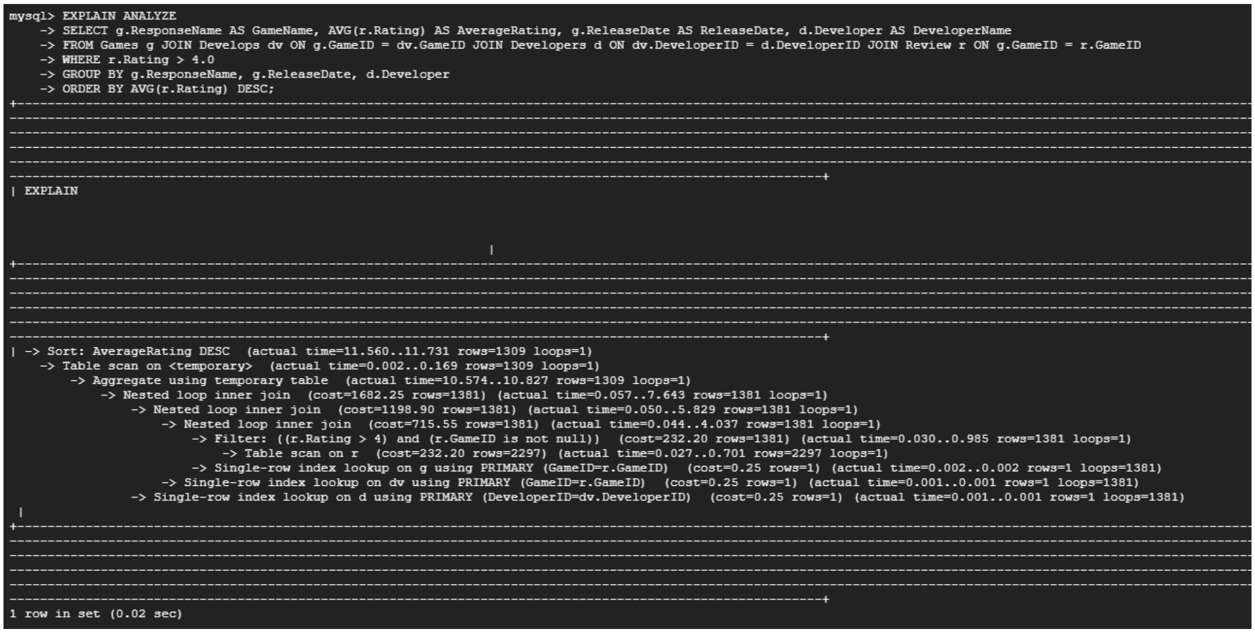
ALTER TABLE Games
DROP INDEX RDIndex;

This line of sql command is to drop this index so it does not affect our later testing.

We decided to test Index 1 because the release date is a commonly used way of selecting a game and is more likely to be used as a filter. We thought it would be a good divider of the different games. Index 1 did improve the performance because the it reduced the number of lookups.

Index 2:

CREATE INDEX RTIndex ON Review (Rating);

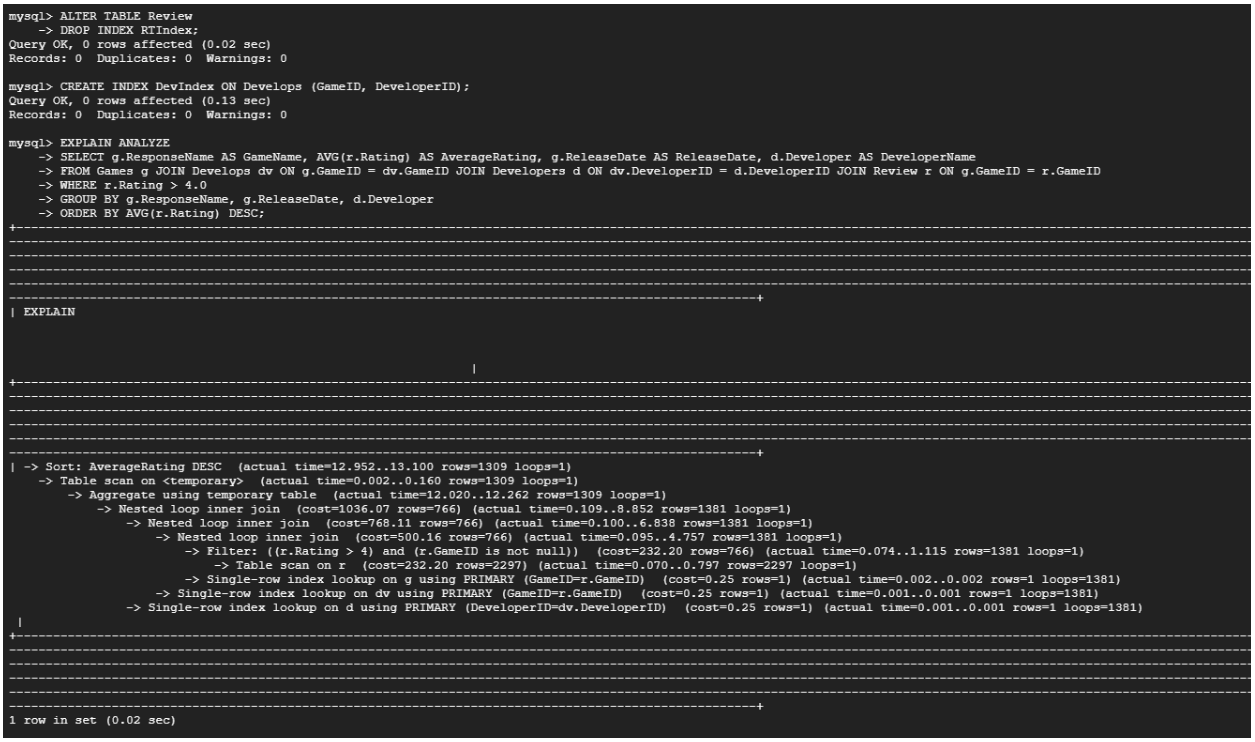


```
mysql> EXPLAIN ANALYZE
    -> SELECT g.ResponseName AS GameName, AVG(r.Rating) AS AverageRating, g.ReleaseDate AS ReleaseDate, d.Developer AS DeveloperName
    -> FROM Games g JOIN Develops dv ON g.GameID = dv.GameID JOIN Developers d ON dv.DeveloperID = d.DeveloperID JOIN Review r ON g.GameID = r.GameID
    -> WHERE r.Rating > 4.0
    -> GROUP BY g.ResponseName, g.ReleaseDate, d.Developer
    -> ORDER BY AVG(r.Rating) DESC;
+---------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------+
| EXPLAIN



                                                                |
+---------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------+
| -> Sort: AverageRating DESC  (actual time=11.560..11.731 rows=1309 loops=1)
    -> Table scan on <temporary>  (actual time=0.002..0.169 rows=1309 loops=1)
        -> Aggregate using temporary table  (actual time=10.574..10.827 rows=1309 loops=1)
            -> Nested loop inner join  (cost=1682.25 rows=1381) (actual time=0.057..7.643 rows=1381 loops=1)
                -> Nested loop inner join  (cost=1198.90 rows=1381) (actual time=0.050..5.829 rows=1381 loops=1)
                    -> Nested loop inner join  (cost=715.55 rows=1381) (actual time=0.044..4.037 rows=1381 loops=1)
                        -> Filter: ((r.Rating > 4) and (r.GameID is not null))  (cost=232.20 rows=1381) (actual time=0.030..0.985 rows=1381 loops=1)
                            -> Table scan on r  (cost=232.20 rows=2297) (actual time=0.027..0.701 rows=2297 loops=1)
                        -> Single-row index lookup on g using PRIMARY (GameID=r.GameID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1381)
                    -> Single-row index lookup on dv using PRIMARY (GameID=r.GameID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1381)
                -> Single-row index lookup on d using PRIMARY (DeveloperID=dv.DeveloperID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1381)
 |
+---------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------+
1 row in set (0.02 sec)
```

ALTER TABLE Review
DROP INDEX RTIndex;

We decided to test index 2 because most people look at the reviews of a game to determine if they would want it or not and it is a useful way to sort to the well rated games. Index 2 improved performance because it has a high number of matches per tables so it is a good lookup index to divide the data.

Index 3:
CREATE INDEX DevIndex ON Develops (GameID, DeveloperID);



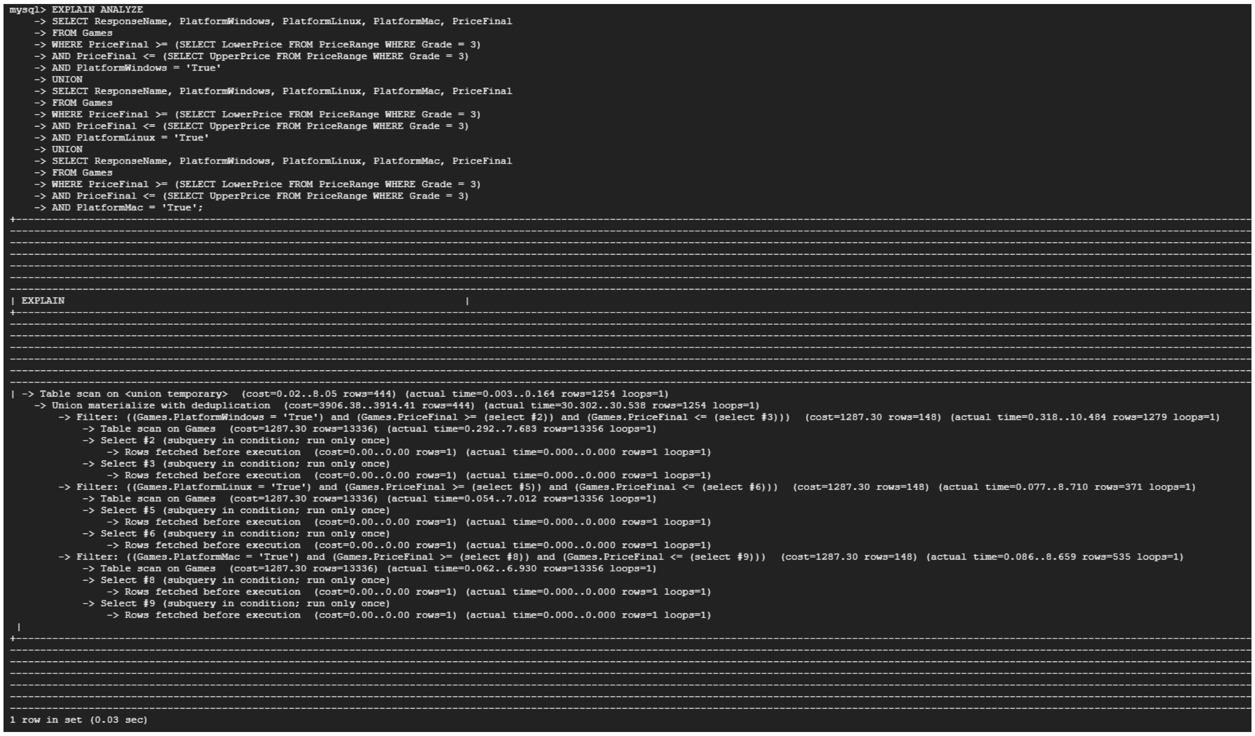
```
mysql> ALTER TABLE Review
    -> DROP INDEX RTIndex;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> CREATE INDEX DevIndex ON Develops (GameID, DeveloperID);
Query OK, 0 rows affected (0.13 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE
    -> SELECT g.ResponseName AS GameName, AVG(r.Rating) AS AverageRating, g.ReleaseDate AS ReleaseDate, d.Developer AS DeveloperName
    -> FROM Games g JOIN Develops dv ON g.GameID = dv.GameID JOIN Developers d ON dv.DeveloperID = d.DeveloperID JOIN Review r ON g.GameID = r.GameID
    -> WHERE r.Rating > 4.0
    -> GROUP BY g.ResponseName, g.ReleaseDate, d.Developer
    -> ORDER BY AVG(r.Rating) DESC;
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------+
| EXPLAIN



                                                               |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------+
| -> Sort: AverageRating DESC  (actual time=12.952..13.100 rows=1309 loops=1)
    -> Table scan on <temporary>  (actual time=0.002..0.160 rows=1309 loops=1)
        -> Aggregate using temporary table  (actual time=12.020..12.262 rows=1309 loops=1)
            -> Nested loop inner join  (cost=1036.07 rows=766) (actual time=0.109..8.852 rows=1381 loops=1)
                -> Nested loop inner join  (cost=768.11 rows=766) (actual time=0.100..6.838 rows=1381 loops=1)
                    -> Nested loop inner join  (cost=500.16 rows=766) (actual time=0.095..4.757 rows=1381 loops=1)
                        -> Filter: ((r.Rating > 4) and (r.GameID is not null))  (cost=232.20 rows=766) (actual time=0.074..1.115 rows=1381 loops=1)
                            -> Table scan on r  (cost=232.20 rows=2297) (actual time=0.070..0.797 rows=2297 loops=1)
                        -> Single-row index lookup on g using PRIMARY (GameID=r.GameID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1381)
                    -> Single-row index lookup on dv using PRIMARY (GameID=r.GameID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1381)
                -> Single-row index lookup on d using PRIMARY (DeveloperID=dv.DeveloperID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1381)
 |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------+
1 row in set (0.02 sec)
```

ALTER TABLE Develops
DROP INDEX DevIndex;

We saw that index 3 made an improvement and we decided to test it because it would not interfere much with the other queries and still have a good amount of results per table.

It seems that Index 2 is the fastest. The total actual time for index 2 is 10.574 - 11.731, which is the fastest among the four. Additionally, index 2’s query plan estimates that there are 1381 rows that satisfy the join conditions between the Review, Games, and Develops tables, which is the most among all other plans. The cost for index 2’s query plan is higher is explainable because there are more matching rows between tables in index 2’s query plan. Therefore, we will be using index plan 2.

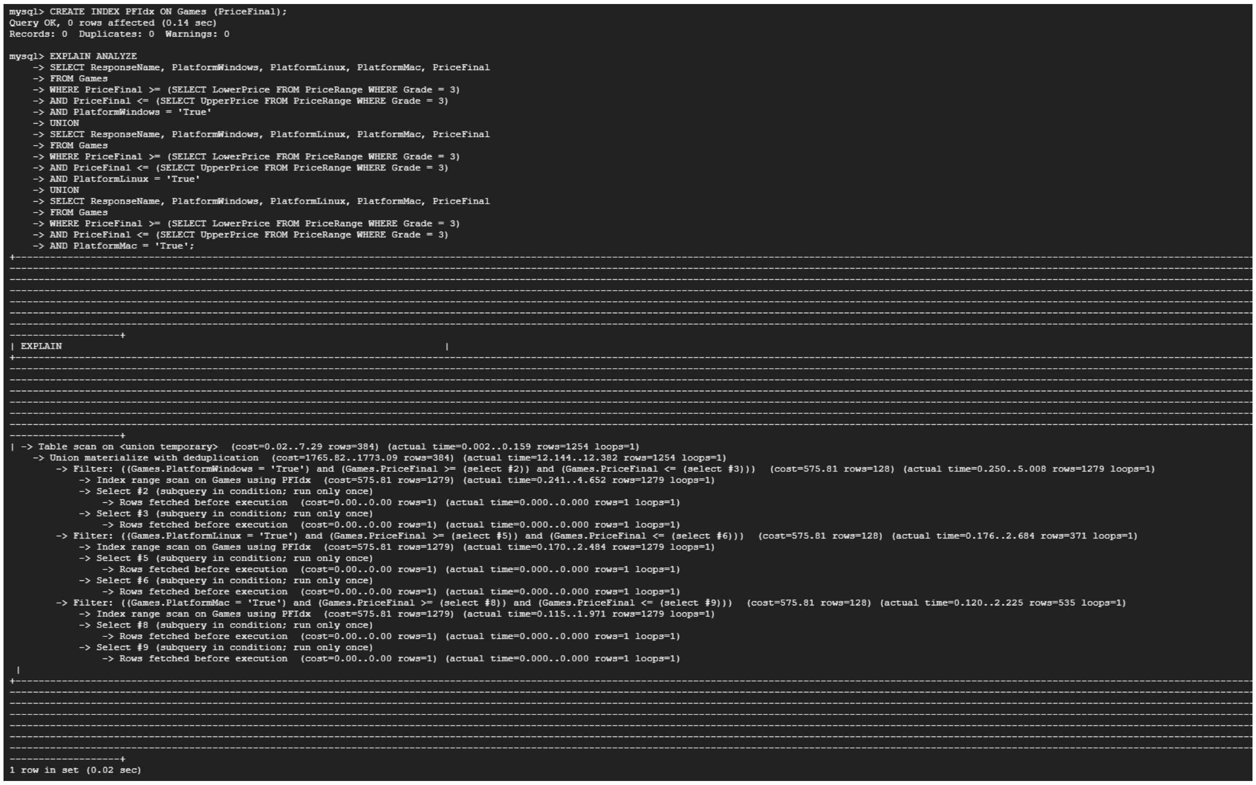
Query 2
Without Index:



```
mysql> EXPLAIN ANALYZE
    -> SELECT ResponseName, PlatformWindows, PlatformLinux, PlatformMac, PriceFinal
    -> FROM Games
    -> WHERE PriceFinal >= (SELECT LowerPrice FROM PriceRange WHERE Grade = 3)
    -> AND PriceFinal <= (SELECT UpperPrice FROM PriceRange WHERE Grade = 3)
    -> AND PlatformWindows = 'True'
    -> UNION
    -> SELECT ResponseName, PlatformWindows, PlatformLinux, PlatformMac, PriceFinal
    -> FROM Games
    -> WHERE PriceFinal >= (SELECT LowerPrice FROM PriceRange WHERE Grade = 3)
    -> AND PriceFinal <= (SELECT UpperPrice FROM PriceRange WHERE Grade = 3)
    -> AND PlatformLinux = 'True'
    -> UNION
    -> SELECT ResponseName, PlatformWindows, PlatformLinux, PlatformMac, PriceFinal
    -> FROM Games
    -> WHERE PriceFinal >= (SELECT LowerPrice FROM PriceRange WHERE Grade = 3)
    -> AND PriceFinal <= (SELECT UpperPrice FROM PriceRange WHERE Grade = 3)
    -> AND PlatformMac = 'True';
+---------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
| EXPLAIN                                                                 |
+---------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
| -> Table scan on <union temporary>  (cost=0.02..8.05 rows=444) (actual time=0.003..0.164 rows=1254 loops=1)
    -> Union materialize with deduplication  (cost=3906.38..3914.41 rows=444) (actual time=30.302..30.538 rows=1254 loops=1)
        -> Filter: ((Games.PlatformWindows = 'True') and (Games.PriceFinal >= (select #2)) and (Games.PriceFinal <= (select #3)))  (cost=1287.30 rows=148) (actual time=0.318..10.484 rows=1279 loops=1)
            -> Table scan on Games  (cost=1287.30 rows=13336) (actual time=0.292..7.683 rows=13356 loops=1)
            -> Select #2 (subquery in condition; run only once)
                -> Rows fetched before execution  (cost=0.00..0.00 rows=1) (actual time=0.000..0.000 rows=1 loops=1)
            -> Select #3 (subquery in condition; run only once)
                -> Rows fetched before execution  (cost=0.00..0.00 rows=1) (actual time=0.000..0.000 rows=1 loops=1)
        -> Filter: ((Games.PlatformLinux = 'True') and (Games.PriceFinal >= (select #5)) and (Games.PriceFinal <= (select #6)))  (cost=1287.30 rows=148) (actual time=0.077..8.710 rows=371 loops=1)
            -> Table scan on Games  (cost=1287.30 rows=13336) (actual time=0.054..7.012 rows=13356 loops=1)
            -> Select #5 (subquery in condition; run only once)
                -> Rows fetched before execution  (cost=0.00..0.00 rows=1) (actual time=0.000..0.000 rows=1 loops=1)
            -> Select #6 (subquery in condition; run only once)
                -> Rows fetched before execution  (cost=0.00..0.00 rows=1) (actual time=0.000..0.000 rows=1 loops=1)
        -> Filter: ((Games.PlatformMac = 'True') and (Games.PriceFinal >= (select #8)) and (Games.PriceFinal <= (select #9)))  (cost=1287.30 rows=148) (actual time=0.086..8.659 rows=535 loops=1)
            -> Table scan on Games  (cost=1287.30 rows=13336) (actual time=0.062..6.930 rows=13356 loops=1)
            -> Select #8 (subquery in condition; run only once)
                -> Rows fetched before execution  (cost=0.00..0.00 rows=1) (actual time=0.000..0.000 rows=1 loops=1)
            -> Select #9 (subquery in condition; run only once)
                -> Rows fetched before execution  (cost=0.00..0.00 rows=1) (actual time=0.000..0.000 rows=1 loops=1)
 |
+---------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
1 row in set (0.03 sec)
```

Index Plan 1:

CREATE INDEX PFIdx ON Games (PriceFinal);

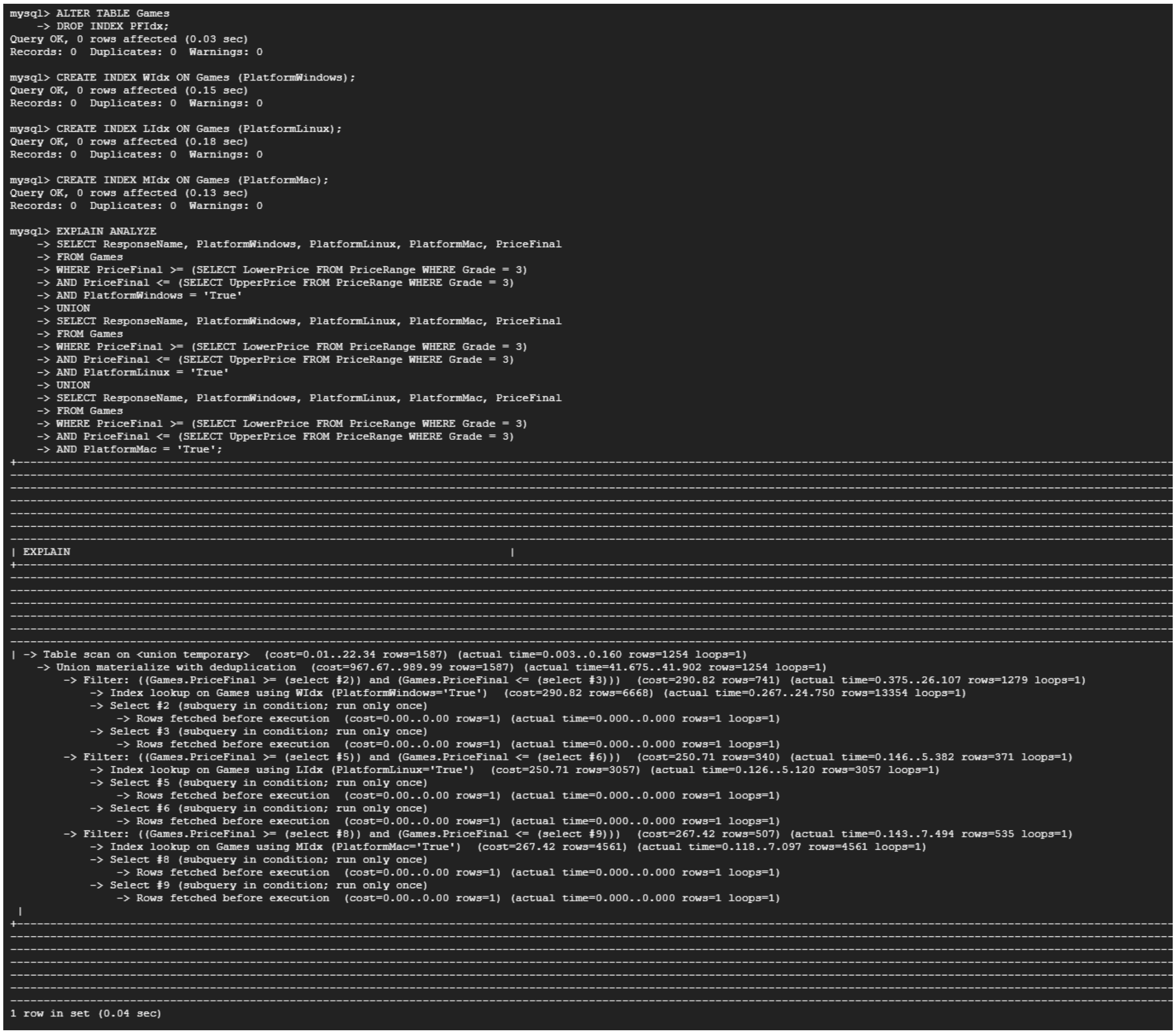


```
mysql> CREATE INDEX PFIdx ON Games (PriceFinal);
Query OK, 0 rows affected (0.14 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE
    -> SELECT ResponseName, PlatformWindows, PlatformLinux, PlatformMac, PriceFinal
    -> FROM Games
    -> WHERE PriceFinal >= (SELECT LowerPrice FROM PriceRange WHERE Grade = 3)
    -> AND PriceFinal <= (SELECT UpperPrice FROM PriceRange WHERE Grade = 3)
    -> AND PlatformWindows = 'True'
    -> UNION
    -> SELECT ResponseName, PlatformWindows, PlatformLinux, PlatformMac, PriceFinal
    -> FROM Games
    -> WHERE PriceFinal >= (SELECT LowerPrice FROM PriceRange WHERE Grade = 3)
    -> AND PriceFinal <= (SELECT UpperPrice FROM PriceRange WHERE Grade = 3)
    -> AND PlatformLinux = 'True'
    -> UNION
    -> SELECT ResponseName, PlatformWindows, PlatformLinux, PlatformMac, PriceFinal
    -> FROM Games
    -> WHERE PriceFinal >= (SELECT LowerPrice FROM PriceRange WHERE Grade = 3)
    -> AND PriceFinal <= (SELECT UpperPrice FROM PriceRange WHERE Grade = 3)
    -> AND PlatformMac = 'True';
+---------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
-------------------+
| EXPLAIN                                                               |
+---------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
-------------------+
| -> Table scan on <union temporary>  (cost=0.02..7.29 rows=384) (actual time=0.002..0.159 rows=1254 loops=1)
    -> Union materialize with deduplication  (cost=1765.82..1773.09 rows=384) (actual time=12.144..12.382 rows=1254 loops=1)
        -> Filter: ((Games.PlatformWindows = 'True') and (Games.PriceFinal >= (select #2)) and (Games.PriceFinal <= (select #3)))  (cost=575.81 rows=128) (actual time=0.250..5.008 rows=1279 loops=1)
            -> Index range scan on Games using PFIdx  (cost=575.81 rows=1279) (actual time=0.241..4.652 rows=1279 loops=1)
            -> Select #2 (subquery in condition; run only once)
                -> Rows fetched before execution  (cost=0.00..0.00 rows=1) (actual time=0.000..0.000 rows=1 loops=1)
            -> Select #3 (subquery in condition; run only once)
                -> Rows fetched before execution  (cost=0.00..0.00 rows=1) (actual time=0.000..0.000 rows=1 loops=1)
        -> Filter: ((Games.PlatformLinux = 'True') and (Games.PriceFinal >= (select #5)) and (Games.PriceFinal <= (select #6)))  (cost=575.81 rows=128) (actual time=0.176..2.684 rows=371 loops=1)
            -> Index range scan on Games using PFIdx  (cost=575.81 rows=1279) (actual time=0.170..2.484 rows=1279 loops=1)
            -> Select #5 (subquery in condition; run only once)
                -> Rows fetched before execution  (cost=0.00..0.00 rows=1) (actual time=0.000..0.000 rows=1 loops=1)
            -> Select #6 (subquery in condition; run only once)
                -> Rows fetched before execution  (cost=0.00..0.00 rows=1) (actual time=0.000..0.000 rows=1 loops=1)
        -> Filter: ((Games.PlatformMac = 'True') and (Games.PriceFinal >= (select #8)) and (Games.PriceFinal <= (select #9)))  (cost=575.81 rows=128) (actual time=0.120..2.225 rows=535 loops=1)
            -> Index range scan on Games using PFIdx  (cost=575.81 rows=1279) (actual time=0.115..1.971 rows=1279 loops=1)
            -> Select #8 (subquery in condition; run only once)
                -> Rows fetched before execution  (cost=0.00..0.00 rows=1) (actual time=0.000..0.000 rows=1 loops=1)
            -> Select #9 (subquery in condition; run only once)
                -> Rows fetched before execution  (cost=0.00..0.00 rows=1) (actual time=0.000..0.000 rows=1 loops=1)
 |
+---------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
-------------------+
1 row in set (0.02 sec)
```

ALTER TABLE Games
DROP INDEX PFIdx;

We test to use index 1 because it is a budget-based index that could be useful in diving up the searches. We thought there will be many results per table because the final price is a large range.

Index Plan 2:
CREATE INDEX WIdx ON Games (PlatformWindows);
CREATE INDEX LIdx ON Games (PlatformLinux);
CREATE INDEX MIdx ON Games (PlatformMac);

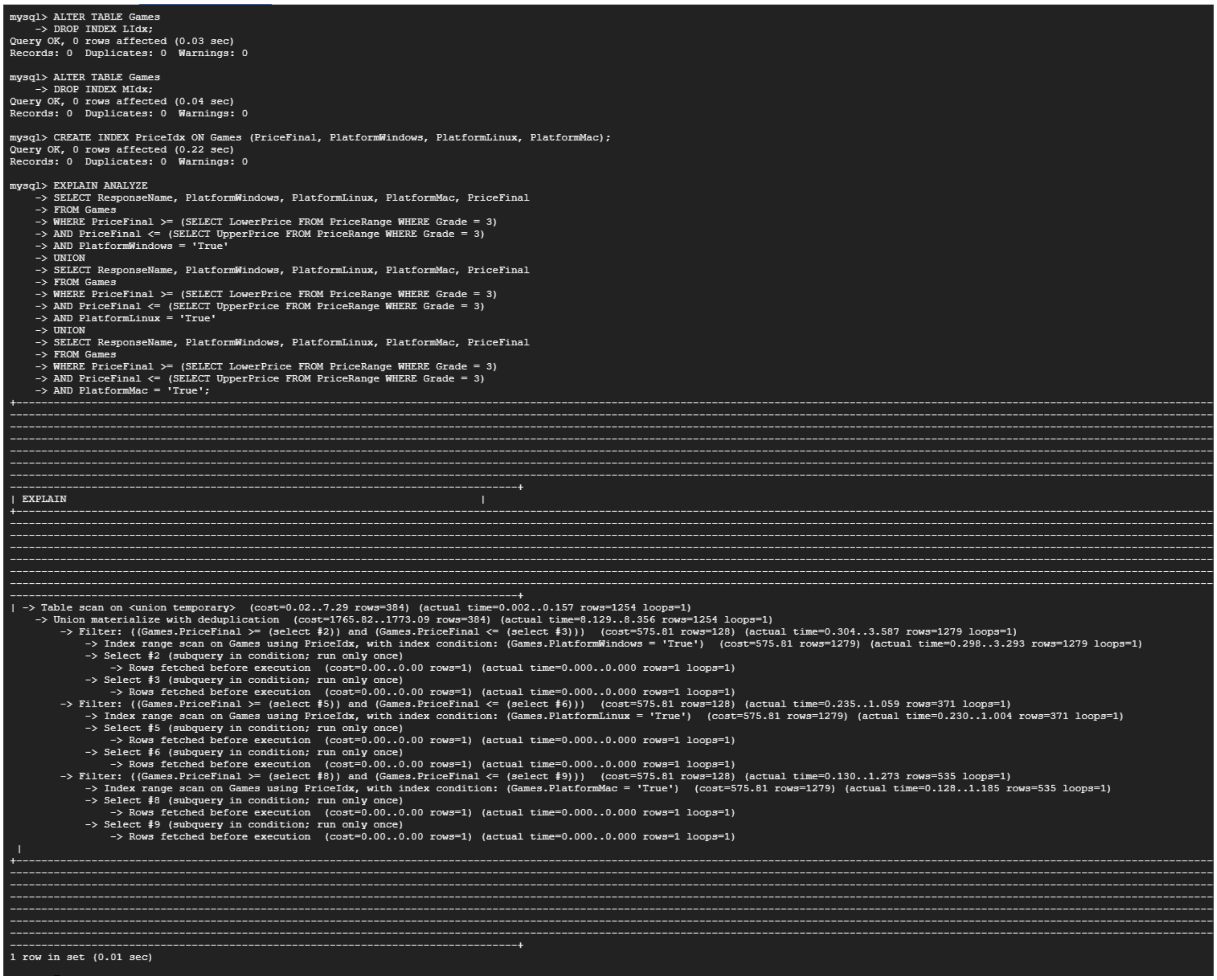


```
mysql> ALTER TABLE Games
    -> DROP INDEX PFIdx;
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> CREATE INDEX WIdx ON Games (PlatformWindows);
Query OK, 0 rows affected (0.15 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> CREATE INDEX LIdx ON Games (PlatformLinux);
Query OK, 0 rows affected (0.18 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> CREATE INDEX MIdx ON Games (PlatformMac);
Query OK, 0 rows affected (0.13 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE
    -> SELECT ResponseName, PlatformWindows, PlatformLinux, PlatformMac, PriceFinal
    -> FROM Games
    -> WHERE PriceFinal >= (SELECT LowerPrice FROM PriceRange WHERE Grade = 3)
    -> AND PriceFinal <= (SELECT UpperPrice FROM PriceRange WHERE Grade = 3)
    -> AND PlatformWindows = 'True'
    -> UNION
    -> SELECT ResponseName, PlatformWindows, PlatformLinux, PlatformMac, PriceFinal
    -> FROM Games
    -> WHERE PriceFinal >= (SELECT LowerPrice FROM PriceRange WHERE Grade = 3)
    -> AND PriceFinal <= (SELECT UpperPrice FROM PriceRange WHERE Grade = 3)
    -> AND PlatformLinux = 'True'
    -> UNION
    -> SELECT ResponseName, PlatformWindows, PlatformLinux, PlatformMac, PriceFinal
    -> FROM Games
    -> WHERE PriceFinal >= (SELECT LowerPrice FROM PriceRange WHERE Grade = 3)
    -> AND PriceFinal <= (SELECT UpperPrice FROM PriceRange WHERE Grade = 3)
    -> AND PlatformMac = 'True';
+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
| EXPLAIN                                                                        |
+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
| -> Table scan on <union temporary>  (cost=0.01..22.34 rows=1587) (actual time=0.003..0.160 rows=1254 loops=1)
    -> Union materialize with deduplication  (cost=967.67..989.99 rows=1587) (actual time=41.675..41.902 rows=1254 loops=1)
        -> Filter: ((Games.PriceFinal >= (select #2)) and (Games.PriceFinal <= (select #3)))  (cost=290.82 rows=741) (actual time=0.375..26.107 rows=1279 loops=1)
            -> Index lookup on Games using WIdx (PlatformWindows='True')  (cost=290.82 rows=6668) (actual time=0.267..24.750 rows=13354 loops=1)
            -> Select #2 (subquery in condition; run only once)
                -> Rows fetched before execution  (cost=0.00..0.00 rows=1) (actual time=0.000..0.000 rows=1 loops=1)
            -> Select #3 (subquery in condition; run only once)
                -> Rows fetched before execution  (cost=0.00..0.00 rows=1) (actual time=0.000..0.000 rows=1 loops=1)
        -> Filter: ((Games.PriceFinal >= (select #5)) and (Games.PriceFinal <= (select #6)))  (cost=250.71 rows=340) (actual time=0.146..5.382 rows=371 loops=1)
            -> Index lookup on Games using LIdx (PlatformLinux='True')  (cost=250.71 rows=3057) (actual time=0.126..5.120 rows=3057 loops=1)
            -> Select #5 (subquery in condition; run only once)
                -> Rows fetched before execution  (cost=0.00..0.00 rows=1) (actual time=0.000..0.000 rows=1 loops=1)
            -> Select #6 (subquery in condition; run only once)
                -> Rows fetched before execution  (cost=0.00..0.00 rows=1) (actual time=0.000..0.000 rows=1 loops=1)
        -> Filter: ((Games.PriceFinal >= (select #8)) and (Games.PriceFinal <= (select #9)))  (cost=267.42 rows=507) (actual time=0.143..7.494 rows=535 loops=1)
            -> Index lookup on Games using MIdx (PlatformMac='True')  (cost=267.42 rows=4561) (actual time=0.118..7.097 rows=4561 loops=1)
            -> Select #8 (subquery in condition; run only once)
                -> Rows fetched before execution  (cost=0.00..0.00 rows=1) (actual time=0.000..0.000 rows=1 loops=1)
            -> Select #9 (subquery in condition; run only once)
                -> Rows fetched before execution  (cost=0.00..0.00 rows=1) (actual time=0.000..0.000 rows=1 loops=1)
 |
+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
1 row in set (0.04 sec)
```

ALTER TABLE Games
DROP INDEX WIdx;
ALTER TABLE Games
DROP INDEX LIdx;
ALTER TABLE Games
DROP INDEX MIdx;

We tested Index 2 because it will evenly divide the results into 3 categories based on the device type without interfering too much with the other queries.

Index Plan 3:
CREATE INDEX PriceIdx ON Games (PriceFinal, PlatformWindows, PlatformLinux, PlatformMac);



```
mysql> ALTER TABLE Games
    -> DROP INDEX LIdx;
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE Games
    -> DROP INDEX MIdx;
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> CREATE INDEX PriceIdx ON Games (PriceFinal, PlatformWindows, PlatformLinux, PlatformMac);
Query OK, 0 rows affected (0.22 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE
    -> SELECT ResponseName, PlatformWindows, PlatformLinux, PlatformMac, PriceFinal
    -> FROM Games
    -> WHERE PriceFinal >= (SELECT LowerPrice FROM PriceRange WHERE Grade = 3)
    -> AND PriceFinal <= (SELECT UpperPrice FROM PriceRange WHERE Grade = 3)
    -> AND PlatformWindows = 'True'
    -> UNION
    -> SELECT ResponseName, PlatformWindows, PlatformLinux, PlatformMac, PriceFinal
    -> FROM Games
    -> WHERE PriceFinal >= (SELECT LowerPrice FROM PriceRange WHERE Grade = 3)
    -> AND PriceFinal <= (SELECT UpperPrice FROM PriceRange WHERE Grade = 3)
    -> AND PlatformLinux = 'True'
    -> UNION
    -> SELECT ResponseName, PlatformWindows, PlatformLinux, PlatformMac, PriceFinal
    -> FROM Games
    -> WHERE PriceFinal >= (SELECT LowerPrice FROM PriceRange WHERE Grade = 3)
    -> AND PriceFinal <= (SELECT UpperPrice FROM PriceRange WHERE Grade = 3)
    -> AND PlatformMac = 'True';
+------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------+
| EXPLAIN                                                                       |
+------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------+
| -> Table scan on <union temporary>  (cost=0.02..7.29 rows=384) (actual time=0.002..0.157 rows=1254 loops=1)
    -> Union materialize with deduplication  (cost=1765.82..1773.09 rows=384) (actual time=8.129..8.356 rows=1254 loops=1)
        -> Filter: ((Games.PriceFinal >= (select #2)) and (Games.PriceFinal <= (select #3)))  (cost=575.81 rows=128) (actual time=0.304..3.587 rows=1279 loops=1)
            -> Index range scan on Games using PriceIdx, with index condition: (Games.PlatformWindows = 'True')  (cost=575.81 rows=1279) (actual time=0.298..3.293 rows=1279 loops=1)
            -> Select #2 (subquery in condition; run only once)
                -> Rows fetched before execution  (cost=0.00..0.00 rows=1) (actual time=0.000..0.000 rows=1 loops=1)
            -> Select #3 (subquery in condition; run only once)
                -> Rows fetched before execution  (cost=0.00..0.00 rows=1) (actual time=0.000..0.000 rows=1 loops=1)
        -> Filter: ((Games.PriceFinal >= (select #5)) and (Games.PriceFinal <= (select #6)))  (cost=575.81 rows=128) (actual time=0.235..1.059 rows=371 loops=1)
            -> Index range scan on Games using PriceIdx, with index condition: (Games.PlatformLinux = 'True')  (cost=575.81 rows=1279) (actual time=0.230..1.004 rows=371 loops=1)
            -> Select #5 (subquery in condition; run only once)
                -> Rows fetched before execution  (cost=0.00..0.00 rows=1) (actual time=0.000..0.000 rows=1 loops=1)
            -> Select #6 (subquery in condition; run only once)
                -> Rows fetched before execution  (cost=0.00..0.00 rows=1) (actual time=0.000..0.000 rows=1 loops=1)
        -> Filter: ((Games.PriceFinal >= (select #8)) and (Games.PriceFinal <= (select #9)))  (cost=575.81 rows=128) (actual time=0.130..1.273 rows=535 loops=1)
            -> Index range scan on Games using PriceIdx, with index condition: (Games.PlatformMac = 'True')  (cost=575.81 rows=1279) (actual time=0.128..1.185 rows=535 loops=1)
            -> Select #8 (subquery in condition; run only once)
                -> Rows fetched before execution  (cost=0.00..0.00 rows=1) (actual time=0.000..0.000 rows=1 loops=1)
            -> Select #9 (subquery in condition; run only once)
                -> Rows fetched before execution  (cost=0.00..0.00 rows=1) (actual time=0.000..0.000 rows=1 loops=1)
 |
+------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

ALTER TABLE Games
DROP INDEX PriceIdx;

It seems that Index plan 3 is the fastest. Index plan 3 was effective because including both the price and the platform type was the best way to have the most results in each table. The total actual time for index 3 is 8.129-8.356, which is the fastest among the four. Additionally, index 3's query plan estimates that there are 1279 rows that satisfy the conditions for each query. The cost for index 3's query plan is higher is explainable because there are more matching rows between tables in index 3's query plan. Therefore, we will be using index plan 3.