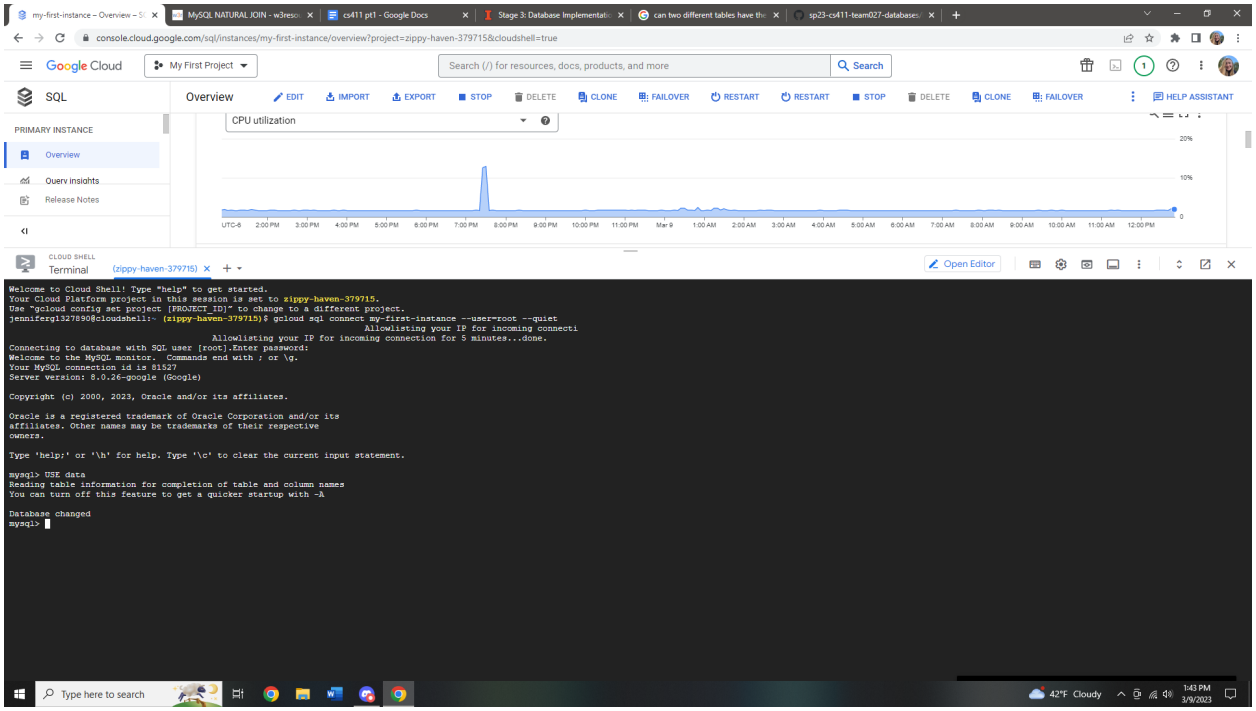
Divya Machineni
Michael Ma
Jennifer Gaw
Kaan Yigit

CONNECTION:



DDL COMMANDS:

```
CREATE TABLE GameData (
      GameID int,
      GameName varchar(255),
      Description varchar(1000),
      Rating int,
      Price REAL,
      Reviews varchar(255),
      PCreq varchar(1000),
      Developer int,
      RequiredAge int,
      Language varchar(255),
      PRIMARY KEY (GameID)
);

CREATE TABLE GenreData (
      GameID int,
      Singleplayer varchar(10),
      Multiplayer varchar(10),
      Coop varchar(10),
      MMO varchar(10),
      InAppPurchase varchar(10),
      VRSupport varchar(10),
      NonGame varchar(10),
      Indie varchar(10),
      Action varchar(10),
      Adventure varchar(10),
      Casual varchar(10),
      Strategy varchar(10),
      RPG varchar(10),
      Simulation varchar(10),
      EarlyAccess varchar(10),
      FreeToPlay varchar(10),
      Sports varchar(10),
      Racing varchar(10),
      MassivelyMultiplayer varchar(10),
      PRIMARY KEY (GameID)
      FOREIGN KEY (GameID) REFERENCES Game(GameID)
```

```
);



CREATE TABLE Region (
     RegionName varchar(255),
     PRIMARY KEY (RegionName)
);

CREATE TABLE PriceRange (
     Range int,
     PriceRangeMin int,
     PriceRangeMax int,
     PRIMARY KEY (Range)
);
```

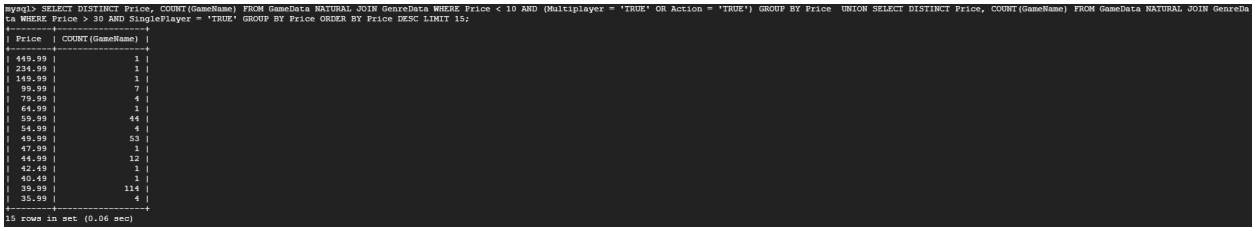
TABLE COUNTS:

```
mysql> SELECT COUNT(*) FROM GameData
    -> ;
+----------+
| COUNT(*) |
+----------+
|    13358 |
+----------+
1 row in set (0.01 sec)

mysql> SELECT COUNT(*) FROM GenreData;
+----------+
| COUNT(*) |
+----------+
|    13358 |
+----------+
1 row in set (0.01 sec)

mysql> SELECT COUNT(*) FROM Region;
+----------+
| COUNT(*) |
+----------+
|     1000 |
+----------+
1 row in set (0.00 sec)

mysql> SELECT COUNT(*) FROM PriceRange;
+----------+
| COUNT(*) |
+----------+
|     1000 |
+----------+
1 row in set (0.00 sec)

mysql>
```

```
SELECT COUNT(*) FROM GameData;
SELECT COUNT(*) FROM GenreData;
SELECT COUNT(*) FROM Region;
SELECT COUNT(*) FROM PriceRange;
```

ADVANCED QUERY 1:

```
SELECT DISTINCT Price, COUNT(GameName)
FROM GameData NATURAL JOIN GenreData
WHERE Price < 10 AND (Multiplayer = 'TRUE' OR Action = 'TRUE')
GROUP BY Price
UNION
SELECT DISTINCT Price, COUNT(GameName)
FROM GameData NATURAL JOIN GenreData
WHERE Price > 30 AND SinglePlayer = 'TRUE'
GROUP BY Price
ORDER BY Price DESC
LIMIT 15;
```



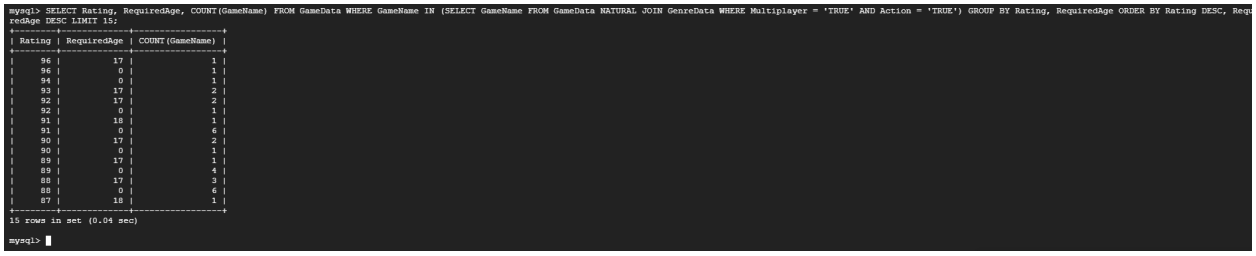
Price	COUNT(GameName)
449.99	1
234.99	1
149.99	1
99.99	7
79.99	4
64.99	1
59.99	44
54.99	4
49.99	53
47.99	1
44.99	12
42.49	1
40.49	1
39.99	114
35.99	4

15 rows in set (0.06 sec)

ADVANCED QUERY 2:

```
SELECT Rating, RequiredAge, COUNT(GameName)
```

```
FROM GameData
WHERE GameName IN (SELECT GameName
                   FROM GameData NATURAL JOIN GenreData
                   WHERE Multiplayer = 'TRUE' AND Action = 'TRUE')
GROUP BY Rating, RequiredAge
ORDER BY Rating DESC, RequiredAge DESC
LIMIT 15;
```



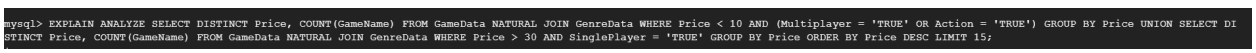
Rating	RequiredAge	COUNT(GameName)
96	17	1
96	0	1
94	0	1
93	17	2
92	17	2
92	0	1
91	18	1
91	0	6
90	17	2
90	0	1
89	17	1
89	0	4
88	17	3
88	0	6
87	18	1

15 rows in set (0.04 sec)

INDEXING:

Advanced Query 1:

Original:



```
| -> Limit: 15 row(s)  (cost=2.50 rows=0) (actual time=0.030..0.031 rows=15 loops=1)
    -> Sort: Price DESC, limit input to 15 row(s) per chunk  (cost=2.50 rows=0) (actual time=0.029..0.030 rows=15 loops=1)
        -> Table scan on <union temporary>  (cost=2.50 rows=0) (actual time=0.000..0.005 rows=99 loops=1)
            -> Union materialize with deduplication  (cost=2.50..2.50 rows=0) (actual time=56.520..56.523 rows=99 loops=1)
                -> Table scan on <temporary>  (actual time=0.001..0.005 rows=80 loops=1)
                    -> Aggregate using temporary table  (actual time=31.509..31.517 rows=80 loops=1)
                        -> Inner hash join (GameData.GameID = GenreData.GameID)  (cost=96587.25 rows=31103) (actual time=13.217..29.979 rows=5241 loops=1)
                            -> Filter: (GameData.Price < 10)  (cost=0.76 rows=368) (actual time=0.025..11.661 rows=10305 loops=1)
                                -> Table scan on GameData  (cost=0.76 rows=11030) (actual time=0.023..10.460 rows=13358 loops=1)
                            -> Hash
                                -> Filter: ((GenreData.Multiplayer = 'TRUE') or (GenreData.`Action` = 'TRUE'))  (cost=1376.25 rows=2538) (actual time=0.038..11.104 rows=7008 loops=1)
                                    -> Table scan on GenreData  (cost=1376.25 rows=13360) (actual time=0.027..8.258 rows=13358 loops=1)
                -> Table scan on <temporary>  (actual time=0.001..0.002 rows=19 loops=1)
                    -> Aggregate using temporary table  (actual time=24.937..24.939 rows=19 loops=1)
                        -> Inner hash join (GameData.GameID = GenreData.GameID)  (cost=51756.30 rows=16370) (actual time=12.636..24.813 rows=288 loops=1)
                            -> Filter: (GameData.Price > 30)  (cost=0.97 rows=368) (actual time=0.150..10.729 rows=396 loops=1)
                                -> Table scan on GameData  (cost=0.97 rows=11030) (actual time=0.020..9.928 rows=13358 loops=1)
                            -> Hash
                                -> Filter: (GenreData.Singleplayer = 'TRUE')  (cost=1376.25 rows=1336) (actual time=0.029..10.362 rows=11687 loops=1)
                                    -> Table scan on GenreData  (cost=1376.25 rows=13360) (actual time=0.018..7.991 rows=13358 loops=1)
 |
```

Index 1:

```
mysql> CREATE INDEX Singleplayer_idx on GenreData(Singleplayer);
Query OK, 0 rows affected (0.15 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Results:

```
| -> Limit: 15 row(s)  (cost=2.50 rows=0) (actual time=0.029..0.031 rows=15 loops=1)
    -> Sort: Price DESC, limit input to 15 row(s) per chunk  (cost=2.50 rows=0) (actual time=0.028..0.029 rows=15 loops=1)
        -> Table scan on <union temporary>  (cost=2.50 rows=0) (actual time=0.000..0.005 rows=99 loops=1)
            -> Union materialize with deduplication  (cost=2.50..2.50 rows=0) (actual time=129.487..129.490 rows=99 loops=1)
                -> Table scan on <temporary>  (actual time=0.001..0.005 rows=80 loops=1)
                    -> Aggregate using temporary table  (actual time=32.317..32.326 rows=80 loops=1)
                        -> Inner hash join (GameData.GameID = GenreData.GameID)  (cost=96587.25 rows=31103) (actual time=13.408..30.835 rows=5241 loops=1)
                            -> Filter: (GameData.Price < 10)  (cost=0.76 rows=368) (actual time=0.028..12.238 rows=10305 loops=1)
                                -> Table scan on GameData  (cost=0.76 rows=11030) (actual time=0.026..11.035 rows=13358 loops=1)
                            -> Hash
                                -> Filter: ((GenreData.Multiplayer = 'TRUE') or (GenreData.`Action` = 'TRUE'))  (cost=1376.25 rows=2538) (actual time=0.033..11.423 rows=7008 loops=1)
                                    -> Table scan on GenreData  (cost=1376.25 rows=13360) (actual time=0.025..8.443 rows=13358 loops=1)
                -> Table scan on <temporary>  (actual time=0.001..0.002 rows=19 loops=1)
                    -> Aggregate using temporary table  (actual time=97.094..97.096 rows=19 loops=1)
                        -> Inner hash join (GameData.GameID = GenreData.GameID)  (cost=248624.51 rows=81851) (actual time=84.648..96.955 rows=288 loops=1)
                            -> Filter: (GameData.Price > 30)  (cost=0.34 rows=368) (actual time=0.208..10.925 rows=396 loops=1)
                                -> Table scan on GameData  (cost=0.34 rows=11030) (actual time=0.024..10.052 rows=13358 loops=1)
                            -> Hash
                                -> Index lookup on GenreData using Singleplayer_idx (Singleplayer='TRUE')  (cost=788.75 rows=6680) (actual time=0.064..81.502 rows=11687 loops=1)
 |
```

Explanation:

When we only index GenreData.Singleplayer, we can see that the cost decreases from 1376.25 to 788.75. This is because the data is stored in Main memory so it can be accessed a lot faster than without indexing Singleplayer.

Index 2:

```
mysql> CREATE INDEX Price ON GameData(price);
```

Results:

```
| -> Limit: 15 row(s)  (cost=2.50 rows=0) (actual time=0.040..0.046 rows=15 loops=1)
    -> Sort: Price DESC, limit input to 15 row(s) per chunk  (cost=2.50 rows=0) (actual time=0.039..0.040 rows=15 loops=1)
        -> Table scan on <union temporary>  (cost=2.50 rows=0) (actual time=0.001..0.006 rows=99 loops=1)
            -> Union materialize with deduplication  (cost=2.50..2.50 rows=0) (actual time=44.712..44.719 rows=99 loops=1)
                -> Table scan on <temporary>  (actual time=0.001..0.005 rows=80 loops=1)
                    -> Aggregate using temporary table  (actual time=31.893..31.902 rows=80 loops=1)
                        -> Inner hash join (GameData.GameID = GenreData.GameID)  (cost=413641.27 rows=204668) (actual time=13.662..30.383 rows=5241 loops=1)
                            -> Filter: (GameData.Price < 10)  (cost=0.40 rows=552) (actual time=0.031..11.635 rows=10305 loops=1)
                                -> Table scan on GameData  (cost=0.40 rows=11030) (actual time=0.028..10.455 rows=13358 loops=1)
                            -> Hash
                                -> Filter: ((GenreData.Multiplayer = 'TRUE') or (GenreData.`Action` = 'TRUE'))  (cost=1376.25 rows=7422) (actual time=0.023..11.718 rows=7008 loops=1)
                                    -> Table scan on GenreData  (cost=1376.25 rows=13360) (actual time=0.017..8.712 rows=13358 loops=1)
                -> Table scan on <temporary>  (actual time=0.001..0.002 rows=19 loops=1)
                    -> Aggregate using temporary table  (actual time=12.730..12.733 rows=19 loops=1)
                        -> Inner hash join (GenreData.GameID = GameData.GameID)  (cost=8656.42 rows=529) (actual time=1.350..12.610 rows=288 loops=1)
                            -> Filter: (GenreData.Singleplayer = 'TRUE')  (cost=8.08 rows=134) (actual time=0.022..10.180 rows=11687 loops=1)
                                -> Table scan on GenreData  (cost=8.08 rows=13360) (actual time=0.011..7.814 rows=13358 loops=1)
                            -> Hash
                                -> Index range scan on GameData using Price, with index condition: (GameData.Price > 30)  (cost=178.46 rows=396) (actual time=0.051..1.077 rows=396 loops=1)
 |
```

Explanation:
When we only index Price, we can see that it affects both parts of the union query.
For the query before the Union:

We can see that the cost relating to GameData.Price decreases from 0.76 to 0.4. This could be because the indexing of the Price would help access the data faster since it is stored in the main memory.

For the query after the Union:

We can see that the cost relating to GameData.Price increased from 0.97 to 178.46. However, the cost relating to GenreData.SinglePlayer decreased from 1376.25 to 8.08. These two variables are linked via this SQL statement: `WHERE Price > 30 AND SinglePlayer = 'TRUE'`. Therefore, the changes in the costs could be due to the Price condition being executed first after indexing it, so this condition would filter out more rows than without the indexing. Thus, for the SinglePlayer condition, there are less rows for it to filter out and that's why the cost lowers for SinglePlayer instead.

Index 3:

```
mysql> CREATE INDEX Singleplayer_idx on GenreData(Singleplayer);
ERROR 1061 (42000): Duplicate key name 'Singleplayer_idx'
mysql> CREATE INDEX Price_idx on GameData(Price);
Query OK, 0 rows affected (0.12 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Results:

```
-> Limit: 15 row(s)  (cost=2.50 rows=0) (actual time=0.039..0.041 rows=15 loops=1)
    -> Sort: Price DESC, limit input to 15 row(s) per chunk  (cost=2.50 rows=0) (actual time=0.038..0.039 rows=15 loops=1)
        -> Table scan on <union temporary>  (cost=2.50 rows=0) (actual time=0.000..0.005 rows=99 loops=1)
            -> Union materialize with deduplication  (cost=2.50..2.50 rows=0) (actual time=44.702..44.705 rows=99 loops=1)
                -> Table scan on <temporary>  (actual time=0.001..0.005 rows=80 loops=1)
                    -> Aggregate using temporary table  (actual time=32.099..32.108 rows=80 loops=1)
                        -> Inner hash join (GameData.GameID = GenreData.GameID)  (cost=142918.50 rows=69996) (actual time=13.938..30.551 rows=5241 loops=1)
                            -> Filter: (GameData.Price < 10)  (cost=0.63 rows=552) (actual time=0.031..11.513 rows=10305 loops=1)
                                -> Table scan on GameData  (cost=0.63 rows=11030) (actual time=0.028..10.329 rows=13358 loops=1)
                            -> Hash
                                -> Filter: ((GenreData.Multiplayer = 'TRUE') or (GenreData.`Action` = 'TRUE'))  (cost=1376.25 rows=2538) (actual time=0.026..12.033 rows=7008 loops=1)
                                    -> Table scan on GenreData  (cost=1376.25 rows=13360) (actual time=0.020..9.036 rows=13358 loops=1)
                -> Table scan on <temporary>  (actual time=0.001..0.002 rows=19 loops=1)
                    -> Aggregate using temporary table  (actual time=12.517..12.520 rows=19 loops=1)
                        -> Inner hash join (GenreData.GameID = GameData.GameID)  (cost=28447.92 rows=13226) (actual time=1.250..12.403 rows=288 loops=1)
                            -> Filter: (GenreData.Singleplayer = 'TRUE')  (cost=4.76 rows=668) (actual time=0.022..10.101 rows=11687 loops=1)
                                -> Table scan on GenreData  (cost=4.76 rows=13360) (actual time=0.012..7.780 rows=13358 loops=1)
                            -> Hash
                                -> Index range scan on GameData using Price_idx, with index condition: (GameData.Price > 30)  (cost=178.46 rows=396) (actual time=0.043..0.979 rows=396 loops=1)
```

Explanation:
When we index both Singleplayer and Price, we can see that the effects from index 2 still holds (explained above in the index 2 section). However, the key difference is that the Singleplayer cost decreased even more than the cost seen in index 2. For reference, with no indexing, the cost was 1376.25. With the conditions listed in index 2, the cost was 8.08. Yet with the conditions listed in index 3, the cost was 4.76. This is because compared to index 2, we have also indexed SinglePlayer so it will be a lot faster to access the data.

Advanced Query 2:

Original:

```
mysql> EXPLAIN ANALYZE SELECT Rating, RequiredAge, COUNT(GameName)
    -> FROM  GameData
    -> WHERE GameName IN (SELECT GameName
    ->     FROM GameData NATURAL JOIN GenreData
    ->     WHERE Multiplayer = 'TRUE' AND Action = 'TRUE')
    -> GROUP BY Rating, RequiredAge
    -> ORDER BY Rating DESC, RequiredAge DESC
    -> LIMIT 15;
```

```
| -> Limit: 15 row(s)  (actual time=44.072..44.073 rows=15 loops=1)
    -> Sort: GameData.Rating DESC, GameData.RequiredAge DESC, limit input to 15 row(s) per chunk  (actual time=44.071..44.072 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=0.001..0.008 rows=130 loops=1)
            -> Aggregate using temporary table  (actual time=44.013..44.028 rows=130 loops=1)
                -> Nested loop inner join  (cost=1785619579.15 rows=17856170924) (actual time=21.891..43.125 rows=2109 loops=1)
                    -> Filter: (GameData.GameName is not null)  (cost=1383.75 rows=11030) (actual time=0.025..11.828 rows=13358 loops=1)
                        -> Table scan on GameData  (cost=1383.75 rows=11030) (actual time=0.024..10.715 rows=13358 loops=1)
                    -> Single-row index lookup on <subquery2> using <auto_distinct_key> (GameName=GameData.GameName)  (actual time=0.000..0.000 rows=0 loops=13358)
                        -> Materialize with deduplication  (cost=1781448.62..1781448.62 rows=1618873) (actual time=29.530..29.711 rows=2091 loops=1)
                            -> Filter: (GameData.GameName is not null)  (cost=1619561.30 rows=1618873) (actual time=7.957..19.981 rows=2130 loops=1)
                                -> Inner hash join (GameData.GameID = GenreData.GameID)  (cost=1619561.30 rows=1618873) (actual time=7.955..19.789 rows=2130 loops=1)
                                    -> Table scan on GameData  (cost=0.36 rows=11030) (actual time=0.019..10.109 rows=13358 loops=1)
                                    -> Hash
                                        -> Filter: (GenreData.`Action` = 'TRUE')  (cost=267.52 rows=1468) (actual time=0.022..7.050 rows=2106 loops=1)
                                            -> Index lookup on GenreData using Multi (Multiplayer='TRUE')  (cost=267.52 rows=3481) (actual time=0.020..6.318 rows=3481 loops=1)
|
```

Index 1:

```
mysql> CREATE INDEX Multiplayer_idx on GenreData(Multiplayer);
Query OK, 0 rows affected, 1 warning (0.13 sec)
Records: 0  Duplicates: 0  Warnings: 1
```

```
| -> Limit: 15 row(s)  (actual time=42.397..42.399 rows=15 loops=1)
    -> Sort: GameData.Rating DESC, GameData.RequiredAge DESC, limit input to 15 row(s) per chunk  (actual time=42.396..42.397 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=0.001..0.008 rows=130 loops=1)
            -> Aggregate using temporary table  (actual time=42.334..42.348 rows=130 loops=1)
                -> Nested loop inner join  (cost=1785619579.15 rows=17856170924) (actual time=20.585..41.427 rows=2109 loops=1)
                    -> Filter: (GameData.GameName is not null)  (cost=1383.75 rows=11030) (actual time=0.025..11.571 rows=13358 loops=1)
                        -> Table scan on GameData  (cost=1383.75 rows=11030) (actual time=0.024..10.536 rows=13358 loops=1)
                    -> Single-row index lookup on <subquery2> using <auto_distinct_key> (GameName=GameData.GameName)  (actual time=0.000..0.000 rows=0 loops=13358)
                        -> Materialize with deduplication  (cost=1781448.62..1781448.62 rows=1618873) (actual time=28.089..28.277 rows=2091 loops=1)
                            -> Filter: (GameData.GameName is not null)  (cost=1619561.30 rows=1618873) (actual time=7.655..19.153 rows=2130 loops=1)
                                -> Inner hash join (GameData.GameID = GenreData.GameID)  (cost=1619561.30 rows=1618873) (actual time=7.653..18.989 rows=2130 loops=1)
                                    -> Table scan on GameData  (cost=0.36 rows=11030) (actual time=0.019..9.719 rows=13358 loops=1)
                                    -> Hash
                                        -> Filter: (GenreData.`Action` = 'TRUE')  (cost=267.52 rows=1468) (actual time=0.027..6.435 rows=2106 loops=1)
                                            -> Index lookup on GenreData using Multi (Multiplayer='TRUE')  (cost=267.52 rows=3481) (actual time=0.024..5.800 rows=3481 loops=1)
|
```

Explanation:

When we solely index GenreData.Multiplayer, we can see that the actualtime decreases from 0.027.. to 0.024 . This is because the data is stored in Main memory so it can be accessed a lot faster than without indexing Multiplayer.

Index 2:

```
mysql>
mysql> Create Index Rating on GameData(rating)
    -> ;
Query OK, 0 rows affected (0.10 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
| -> Limit: 15 row(s)  (actual time=41.880..41.881 rows=15 loops=1)
    -> Sort: GameData.Rating DESC, GameData.RequiredAge DESC, limit input to 15 row(s) per chunk  (actual time=41.879..41.879 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=0.001..0.008 rows=130 loops=1)
            -> Aggregate using temporary table  (actual time=41.822..41.835 rows=130 loops=1)
                -> Nested loop inner join  (cost=1785619579.15 rows=17856170924) (actual time=20.302..40.966 rows=2109 loops=1)
                    -> Filter: (GameData.GameName is not null)  (cost=1383.75 rows=11030) (actual time=0.024..11.564 rows=13358 loops=1)
                        -> Table scan on GameData  (cost=1383.75 rows=11030) (actual time=0.022..10.534 rows=13358 loops=1)
                    -> Single-row index lookup on <subquery2> using <auto_distinct_key> (GameName=GameData.GameName)  (actual time=0.000..0.000 rows=0 loops=13358)
                        -> Materialize with deduplication  (cost=1781448.62..1781448.62 rows=1618873) (actual time=27.585..27.764 rows=2091 loops=1)
                            -> Filter: (GameData.GameName is not null)  (cost=1619561.30 rows=1618873) (actual time=7.023..18.818 rows=2130 loops=1)
                                -> Inner hash join (GameData.GameID = GenreData.GameID)  (cost=1619561.30 rows=1618873) (actual time=7.020..18.646 rows=2130 loops=1)
                                    -> Table scan on GameData  (cost=0.36 rows=11030) (actual time=0.018..9.962 rows=13358 loops=1)
                                    -> Hash
                                        -> Filter: (GenreData.`Action` = 'TRUE')  (cost=267.52 rows=1468) (actual time=0.020..6.439 rows=2106 loops=1)
                                            -> Index lookup on GenreData using Multi (Multiplayer='TRUE')  (cost=267.52 rows=3481) (actual time=0.019..5.806 rows=3481 loops=1)
|
```

Explanation:

Indexing by GameData.Rating has improved the performance of the query by reducing the amount of data that needs to be processed, limiting the number of rows that need to be scanned, and improving the efficiency of the joins. Index lookup time on action has decreased from 0.022 to 0.020.

Index 3:

```
mysql> CREATE INDEX Action_id on  GenreData(Action);
```

```
| -> Limit: 15 row(s)  (actual time=49.361..49.363 rows=15 loops=1)
    -> Sort: GameData.Rating DESC, GameData.RequiredAge DESC, limit input to 15 row(s) per chunk  (actual time=49.360..49.361 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=0.001..0.008 rows=130 loops=1)
            -> Aggregate using temporary table  (actual time=49.303..49.318 rows=130 loops=1)
                -> Nested loop inner join  (cost=1785619579.15 rows=17856170924) (actual time=27.177..48.430 rows=2109 loops=1)
                    -> Filter: (GameData.GameName is not null)  (cost=1383.75 rows=11030) (actual time=0.062..11.713 rows=13358 loops=1)
                        -> Table scan on GameData  (cost=1383.75 rows=11030) (actual time=0.028..10.594 rows=13358 loops=1)
                    -> Single-row index lookup on <subquery2> using <auto_distinct_key> (GameName=GameData.GameName)  (actual time=0.000..0.000 rows=0 loops=13358)
                        -> Materialize with deduplication  (cost=1781448.62..1781448.62 rows=1618873) (actual time=34.970..35.148 rows=2091 loops=1)
                            -> Filter: (GameData.GameName is not null)  (cost=1619561.30 rows=1618873) (actual time=9.423..25.577 rows=2130 loops=1)
                                -> Inner hash join (GameData.GameID = GenreData.GameID)  (cost=1619561.30 rows=1618873) (actual time=9.421..25.395 rows=2130 loops=1)
                                    -> Table scan on GameData  (cost=0.36 rows=11030) (actual time=0.017..14.212 rows=13358 loops=1)
                                    -> Hash
                                        -> Filter: (GenreData.`Action` = 'TRUE')  (cost=267.52 rows=1468) (actual time=0.037..8.746 rows=2106 loops=1)
                                            -> Index lookup on GenreData using Multi (Multiplayer='TRUE')  (cost=267.52 rows=3481) (actual time=0.034..8.068 rows=3481 loops=1)
 |
```

Explanation:

Indexing by GenreData.Action, has enabled us to reduce the actual time of the table scan on GameData from 0.024 to 0.017. However it does have some downsides, unlike the previous queries it does not decrease the lookup time on Multiplayer, yet increase it.