1. Database Implementation:

1.1. Setting up the database on GCP:

Here are the connection details:

```
shadowfiend451@cloudshell:~ (buoyant-episode-392516)$ gcloud sql connect cs411-team004-pandas --user=root
Allowlisting your IP for incoming connection for 5 minutes...done.
Connecting to database with SQL user [root].Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2050
Server version: 8.0.31-google (Google)

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

Here is the screenshot of the databases. Our tables are inside the classicmodels database.

```
mysql> show databases
    -> ;
+--------------------+
| Database           |
+--------------------+
| classicmodels      |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.00 sec)
```

Here is the screenshot of our tables within the classicmodels database.

```
mysql> show tables;
+-------------------------+
| Tables_in_classicmodels |
+-------------------------+
| Features                |
| FriendList              |
| Game                    |
| Genre                   |
| LanguagesAndInfo        |
| Platform                |
| User                    |
| WishList                |
+-------------------------+
8 rows in set (0.01 sec)
```

1.2. DDL commands for creating the tables:

(The headers and the footers comes from the sample sql file shown in project workshop 1)

```
/*!40101 SET NAMES utf8 */;
/*!40101 SET SQL_MODE=''*/;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO'
*/;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;
CREATE DATABASE /*!32312 IF NOT EXISTS*/`classicmodels` /*!40100 DEFAULT
CHARACTER
SET latin1 */;
USE `classicmodels`;

DROP TABLE IF EXISTS Game;

CREATE TABLE Game (
ID INT NOT NULL,
NAME VARCHAR(255) NOT NULL,
ReleaseDate VARCHAR(100),
DeveloperCount INT,
DLCcount INT,
MetaCritic INT,
```

```
Recommendation INT,
SteamSpyOwner BIGINT,
SteamSpyPlayer BIGINT,
InitialPrice REAL,
FinalPrice REAL,
PRIMARY KEY (ID)
)ENGINE=InnoDB DEFAULT CHARSET=latin1;


DROP TABLE IF EXISTS Platform;

CREATE TABLE Platform (
PlatformWindows BOOLEAN NOT NULL,
PlatformLinux BOOLEAN NOT NULL,
PlatformMac BOOLEAN NOT NULL,
WindowsMinReqs VARCHAR(500) DEFAULT NULL,
WindowsRecReqs VARCHAR(500) DEFAULT NULL,
LinuxMinReqs VARCHAR(500) DEFAULT NULL,
LinuxRecReqs VARCHAR(500) DEFAULT NULL,
MacMinReqs VARCHAR(500) DEFAULT NULL,
MacRecReqs VARCHAR(500) DEFAULT NULL,
GameID INT NOT NULL,
PRIMARY KEY GameID(GameID),
CONSTRAINT Platform_ibfk_1 FOREIGN KEY (GameID) REFERENCES Game(ID)
)ENGINE=InnoDB DEFAULT CHARSET=latin1;

DROP TABLE IF EXISTS Genre;
CREATE TABLE Genre (
NonGame BOOLEAN,
Indie BOOLEAN,
Action BOOLEAN,
Adventure BOOLEAN,
Casual BOOLEAN,
Strategy BOOLEAN,
RPG BOOLEAN,
Simulation BOOLEAN,
EarlyAccess BOOLEAN,
FreeToPlay BOOLEAN,
           Sports BOOLEAN,
Racing BOOLEAN,
```

```
           MassivelyMultiplayer BOOLEAN,
          GameID INT NOT NULL,
KEY GameID(GameID),
CONSTRAINT Genre_ibfk_1 FOREIGN KEY (GameID) REFERENCES Game(ID),
PRIMARY KEY(GameID)
)ENGINE=InnoDB DEFAULT CHARSET=latin1;

DROP TABLE IF EXISTS Features;
CREATE TABLE Features (
     ControllerSupport BOOLEAN,
     Free BOOLEAN,
     FreeVersionAvailable BOOLEAN,
     Subscription BOOLEAN,
     SinglePlayer BOOLEAN,
     MultiPlayer BOOLEAN,
     Coop BOOLEAN,
     MMO BOOLEAN,
     InAppPurchases BOOLEAN,
IncludeSrcSDK BOOLEAN,
IncludeLevelEditor BOOLEAN,
VRSupport BOOLEAN,
GameID INT NOT NULL,
                    KEY GameID(GameID),
CONSTRAINT Features_ibfk_1 FOREIGN KEY (GameID) REFERENCES Game(ID),
PRIMARY KEY(GameID)
)ENGINE=InnoDB DEFAULT CHARSET=latin1;

DROP TABLE IF EXISTS User;
CREATE TABLE User (
UserID INT NOT NULL,
Email VARCHAR(100) NOT NULL,
Password VARCHAR(100) NOT NULL,
UserName VARCHAR(100) NOT NULL,
PRIMARY KEY (UserID)
)ENGINE=InnoDB DEFAULT CHARSET=latin1;

DROP TABLE IF EXISTS WishList;
CREATE TABLE WishList (
UserID INT NOT NULL,
GameID INT NOT NULL,
```

```
DateAdded VARCHAR(255) DEFAULT NULL,
PRIMARY KEY (UserID, GameID),
CONSTRAINT WishList_ibfk_1 FOREIGN KEY (UserID) REFERENCES User(UserID),
CONSTRAINT WishList_ibfk_2 FOREIGN KEY (GameID) REFERENCES Game(ID)
)ENGINE=InnoDB DEFAULT CHARSET=latin1;

DROP TABLE IF EXISTS FriendList;
CREATE TABLE FriendList (
FriendID INT NOT NULL,
UserID INT NOT NULL,
DateAdded VARCHAR(255) DEFAULT NULL,
PRIMARY KEY (FriendID),
KEY UserID(UserID),
CONSTRAINT Friends_ibfk_1 FOREIGN KEY (UserID) REFERENCES User(UserID)
)ENGINE=InnoDB DEFAULT CHARSET=latin1;

DROP TABLE IF EXISTS LanguagesAndInfo;
CREATE TABLE LanguagesAndInfo (
Languages VARCHAR(255) NOT NULL,
GameID INT NOT NULL,
AboutText VARCHAR(10000) DEFAULT NULL,
PRIMARY KEY GameID(GameID),
CONSTRAINT Language_ibfk_1 FOREIGN KEY (GameID) REFERENCES Game(ID)
)ENGINE=InnoDB DEFAULT CHARSET=latin1;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;
```

1.3. The content of our tables

We inserted > 10000 rows/entries into our Game, Features, Genre, LanguageAndInfo, and Platform main tables. Here are the screenshots of the Count queries from the 5 main tables:

```
mysql> SELECT COUNT(ID) FROM Game;
+-----------+
| COUNT(ID) |
+-----------+
|     13209 |
+-----------+
1 row in set (0.01 sec)
```

```
mysql> SELECT COUNT(GameID) FROM Genre;
+---------------+
| COUNT(GameID) |
+---------------+
|         13209 |
+---------------+
1 row in set (0.01 sec)
```

```
mysql> SELECT COUNT(GameID) FROM Features;
+---------------+
| COUNT(GameID) |
+---------------+
|         13209 |
+---------------+
1 row in set (0.01 sec)
```

```
mysql> SELECT COUNT(GameID) FROM LanguagesAndInfo;
+---------------+
| COUNT(GameID) |
+---------------+
|         13209 |
+---------------+
1 row in set (0.01 sec)
```

```
mysql> SELECT COUNT(GameID) FROM Platform;
+---------------+
| COUNT(GameID) |
+---------------+
|         13209 |
+---------------+
1 row in set (0.01 sec)
```

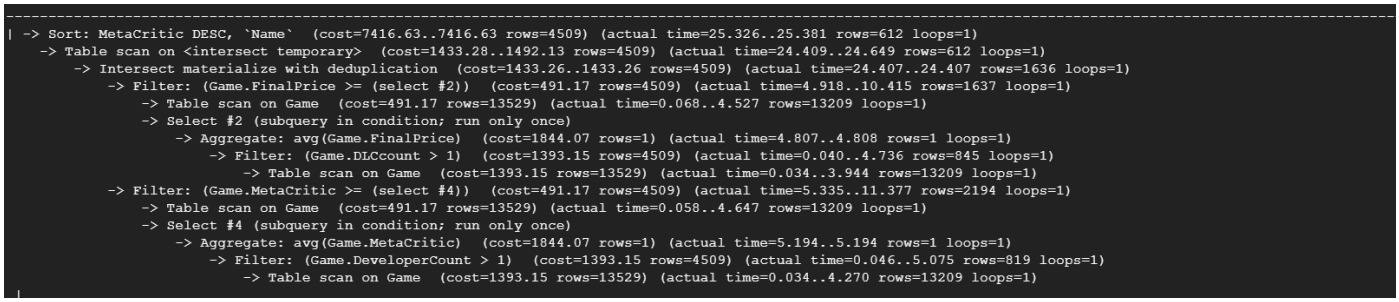
2. Advanced Queries And Indexing Analysis:

First Advanced Query:

```
mysql> SELECT Name, FinalPrice, MetaCritic
    -> FROM Game
    -> WHERE FinalPrice >= (SELECT AVG(FinalPrice)
    ->                      FROM Game
    ->                      WHERE DLCcount > 1)
    ->
    -> INTERSECT
    ->
    -> SELECT Name, FinalPrice, MetaCritic
    -> FROM Game
    -> WHERE MetaCritic >= (SELECT AVG(MetaCritic)
    ->                      FROM Game
    ->                      WHERE DeveloperCount > 1)
    -> ORDER BY MetaCritic DESC, Name
    -> LIMIT 15;
+----------------------------------------+------------+------------+
| Name                                   | FinalPrice | MetaCritic |
+----------------------------------------+------------+------------+
| BioShock                               |      19.99 |         96 |
| Grand Theft Auto V                     |      59.99 |         96 |
| Portal 2                               |      19.99 |         95 |
| Portal 2 - Pre-order                   |      19.99 |         95 |
| BioShock Infinite                      |      29.99 |         94 |
| Divinity: Original Sin Enhanced Edition |      39.99 |         94 |
| Mass Effect 2                          |      19.99 |         94 |
| Sid Meier's Civilization IV            |      19.99 |         94 |
| Sid Meier's Civilization VI            |      59.99 |         94 |
| The Elder Scrolls V: Skyrim            |      19.99 |         94 |
| Company of Heroes                      |      19.99 |         93 |
| The Witcher 3: Wild Hunt               |      39.99 |         93 |
| Call of Duty 4: Modern Warfare         |      19.99 |         92 |
| Out of the Park Baseball 17            |      39.99 |         92 |
| Batman: Arkham Asylum                  |      19.99 |         91 |
+----------------------------------------+------------+------------+
15 rows in set (0.03 sec)
```

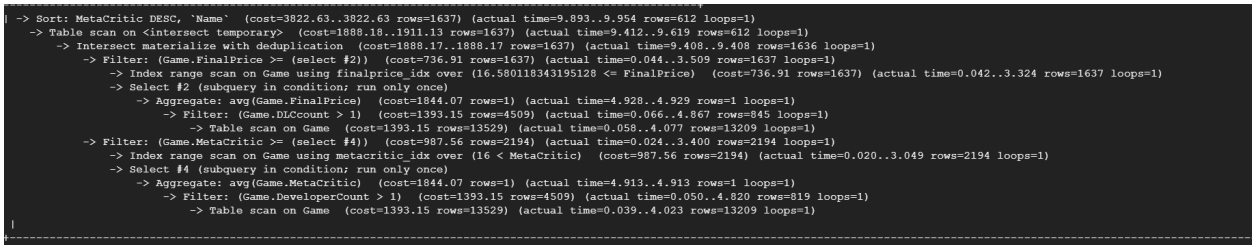
This query displays the name, final price, and Metacritic score of games that are both equally or more expensive than the average final price of games who have more than 1 DLC and have a equal or higher MetaCritic Score than the average MetaCritic score of games developed by more than 1 company. The top 15 rows are displayed.

Running Explain Analyze on Query:



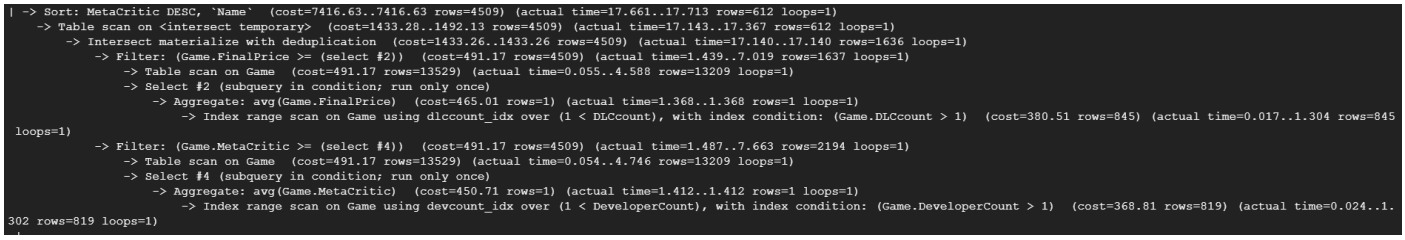
COST: 7416.63

Index Design 1: Index on Final Price -> CREATE INDEX finalprice_idx ON Game(FinalPrice);
Index on MetaCritic -> CREATE INDEX metacritic_idx ON Game(MetaCritic);



COST: 3822.63
DIFF: 7416.63 - 3822.63 = 3594

Index Design 2: Index on DLCCount -> CREATE INDEX dlccount_idx ON Game(DLCcount);
Index on DeveloperCount-> CREATE INDEX devcount_idx ON Game(DeveloperCount);



COST: 7416.63
DIFF: 0

Index Design 3: CREATE INDEX spyowner_idx ON Game(SteamSpyOwner);
CREATE INDEX spyplayer_idx ON Game(SteamSpyPlayer);

```
| -> Sort: MetaCritic DESC, `Name`  (cost=7416.63..7416.63 rows=4509) (actual time=24.536..24.590 rows=612 loops=1)
    -> Table scan on <intersect temporary>  (cost=1433.28..1492.13 rows=4509) (actual time=24.017..24.234 rows=612 loops=1)
        -> Intersect materialize with deduplication  (cost=1433.26..1433.26 rows=4509) (actual time=24.014..24.014 rows=1636 loops=1)
            -> Filter: (Game.FinalPrice >= (select #2))  (cost=491.17 rows=4509) (actual time=4.962..10.658 rows=1637 loops=1)
                -> Table scan on Game  (cost=491.17 rows=13529) (actual time=0.085..4.685 rows=13209 loops=1)
                -> Select #2 (subquery in condition; run only once)
                    -> Aggregate: avg(Game.FinalPrice)  (cost=1844.07 rows=1) (actual time=4.855..4.855 rows=1 loops=1)
                        -> Filter: (Game.DLCcount > 1)  (cost=1393.15 rows=4509) (actual time=0.040..4.791 rows=845 loops=1)
                            -> Table scan on Game  (cost=1393.15 rows=13529) (actual time=0.034..3.929 rows=13209 loops=1)
            -> Filter: (Game.MetaCritic >= (select #4))  (cost=491.17 rows=4509) (actual time=4.729..10.921 rows=2194 loops=1)
                -> Table scan on Game  (cost=491.17 rows=13529) (actual time=0.052..4.728 rows=13209 loops=1)
                -> Select #4 (subquery in condition; run only once)
                    -> Aggregate: avg(Game.MetaCritic)  (cost=1844.07 rows=1) (actual time=4.651..4.651 rows=1 loops=1)
                        -> Filter: (Game.DeveloperCount > 1)  (cost=1393.15 rows=4509) (actual time=0.046..4.563 rows=819 loops=1)
                            -> Table scan on Game  (cost=1393.15 rows=13529) (actual time=0.034..3.837 rows=13209 loops=1)
|
```

COST: 7416.63
DIFF: 0

We are choosing Index Design 1 which has indices on FinalPrice and MetaCritic score. Since we are checking for each game, if the final price is greater than or equal to the avg price in the subquery. We are also checking for each game, if the metacritic score is greater than or equal to the avg score in the subquery. Therefore, these two attributes are accessed the most in the query so putting an index in both of them can lower the total cost compared to no indices.

Index Design 2 had no change. We think putting indices on DLCcount and DeveloperCount did not make any difference because of the constraints we put in the subqueries (DLCcount > 1 and DeveloperCount > 1). These constraints probably returned a large amount of rows that the system might think a full table scan would be better and therefore, did not change the cost.

Index Design 3 had no change. There is no change because the indices on SteamSpyOwner and SteamSpyPlayer are not included in the query.

Advanced Query 2:

```
mysql> SELECT g.Name, g.MetaCritic
    -> FROM Game g JOIN Features f ON (g.ID = f.GameID)
    -> WHERE f.Free = TRUE AND g.MetaCritic >= (SELECT AVG(MetaCritic)
    ->                                          FROM Game g JOIN Features f ON (g.ID = f.GameID)
    ->                                          WHERE f.Free = TRUE)
    -> ORDER BY MetaCritic DESC, Name
    -> LIMIT 15;
+---------------------------------------------+------------+
| Name                                        | MetaCritic |
+---------------------------------------------+------------+
| Team Fortress 2                             |         92 |
| Dota 2                                      |         90 |
| DOOM 3                                      |         87 |
| HEX: Shards of Fate                         |         87 |
| Path of Exile                               |         86 |
| The Lord of the Rings Online                |         86 |
| Tribes: Ascend                              |         86 |
| Might & Magic: Duel of Champions            |         84 |
| Pinball FX2                                 |         84 |
| PlanetSide 2                                |         84 |
| RIFT                                        |         84 |
| EverQuest(r) II Free-To-Play. Your Way.(tm) |         83 |
| SMITE                                       |         83 |
| Duelyst                                     |         82 |
| Realm of the Mad God                        |         82 |
+---------------------------------------------+------------+
15 rows in set (0.02 sec)
```

This advanced query searches for the name and MetaCritic score of games that are free and have a equal or higher MetraCritic score than the average MetaCritic score of all free games. The top 15 rows are displayed

Running Explain Analyze on Query:

```
| -> Sort: g.MetaCritic DESC, g.`NAME`  (actual time=12.544..12.550 rows=87 loops=1)
    -> Stream results  (cost=1830.96 rows=446) (actual time=6.188..12.490 rows=87 loops=1)
        -> Nested loop inner join  (cost=1830.96 rows=446) (actual time=6.182..12.460 rows=87 loops=1)
            -> Filter: (f.Free = true)  (cost=1362.55 rows=1338) (actual time=0.049..4.743 rows=1039 loops=1)
                -> Table scan on f  (cost=1362.55 rows=13383) (actual time=0.045..3.912 rows=13209 loops=1)
            -> Filter: (g.MetaCritic >= (select #2))  (cost=0.25 rows=0.3) (actual time=0.007..0.007 rows=0 loops=1039)
                -> Single-row index lookup on g using PRIMARY (ID=f.GameID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1039)
                -> Select #2 (subquery in condition; run only once)
                    -> Aggregate: avg(g.MetaCritic)  (cost=1964.79 rows=1) (actual time=6.088..6.089 rows=1 loops=1)
                        -> Nested loop inner join  (cost=1830.96 rows=1338) (actual time=0.026..5.964 rows=1039 loops=1)
                            -> Filter: (f.Free = true)  (cost=1362.55 rows=1338) (actual time=0.024..4.607 rows=1039 loops=1)
                                -> Table scan on f  (cost=1362.55 rows=13383) (actual time=0.021..3.807 rows=13209 loops=1)
                            -> Single-row index lookup on g using PRIMARY (ID=f.GameID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1039)
|
```

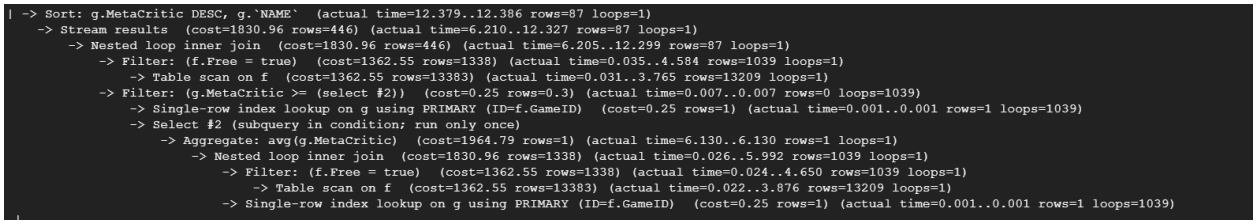
Cost: 1830.96

Index Design 1: CREATE INDEX metacritic_idx ON Game(MetaCritic);
CREATE INDEX free_idx ON Features(Free);

```
-> Sort: g.MetaCritic DESC, g.`NAME`  (actual time=1.831..1.837 rows=87 loops=1)
  -> Stream results  (cost=469.18 rows=168) (actual time=0.051..1.786 rows=87 loops=1)
      -> Nested loop inner join  (cost=469.18 rows=168) (actual time=0.049..1.762 rows=87 loops=1)
          -> Covering index lookup on f using free_idx (Free=true)  (cost=105.53 rows=1039) (actual time=0.035..0.275 rows=1039 loops=1)
          -> Filter: (g.MetaCritic >= (select #2))  (cost=0.25 rows=0.2) (actual time=0.001..0.001 rows=0 loops=1039)
              -> Single-row index lookup on g using PRIMARY (ID=f.GameID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1039)
              -> Select #2 (subquery in condition; run only once)
                  -> Aggregate: avg(g.MetaCritic)  (cost=573.08 rows=1) (actual time=1.736..1.736 rows=1 loops=1)
                      -> Nested loop inner join  (cost=469.18 rows=1039) (actual time=0.045..1.628 rows=1039 loops=1)
                          -> Covering index lookup on f using free_idx (Free=true)  (cost=105.53 rows=1039) (actual time=0.034..0.269 rows=1039 loops=1)
                          -> Single-row index lookup on g using PRIMARY (ID=f.GameID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1039)
|
```

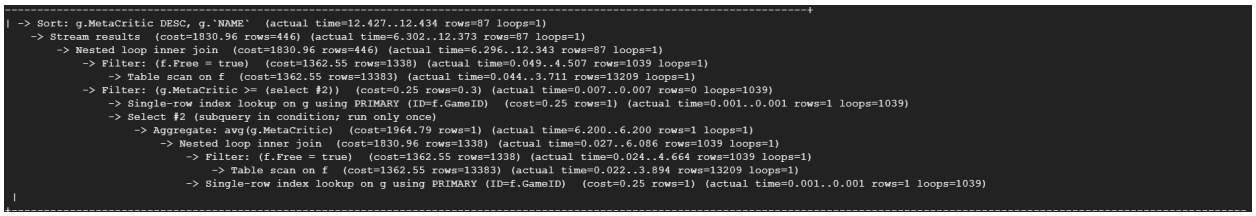
Cost: 469.18
DIFF: 1830.96 - 469.18 = 1361.78

Index Design 2: CREATE INDEX name_idx ON Game(Name);



COST: 1830.96
DIFF: 0

Index Design 3: CREATE INDEX coop_idx ON Features(Coop);
CREATE INDEX mmo_idx ON Features(MMO);



COST: 1830.96
DIFF: 0

We chose Index Design 1 which has indices on MetaCritic and Free attributes. Putting an index within these attributes decreases the cost significantly because for each game we are checking if the game is free and if the MetaCritic score is greater than the average metacritic score of free games. Therefore, since these attributes are accessed the most in the query, it would make sense to put indices here.

Index Design 2 had no change, because we are just selecting the Names from the relation that is already calculated, so we are only accessing each name for each game once. It is not used in the filtering conditions.

Index Design 3 had no change because the attributes Coop and MMO are not used in the query and therefore, not accessed by the query so putting indices here would not improve performance.