Stage 3: Database Implementation and Indexing (22%)

1.Implemented the database tables locally or on GCP

```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to cs411-frenchtoast.
Use “gcloud config set project [PROJECT_ID]” to change to a different project.
jairajpal13@cloudshell:~ (cs411-frenchtoast)$ gcloud sql connect cs411-database --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 38560
Server version: 8.0.26-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.

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

mysql> show tables;
ERROR 1046 (3D000): No database selected
mysql> USE main;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+------------------+
| Tables_in_main   |
+------------------+
| Games            |
| Genre            |
| RecommendedGames |
| UserInputGameList |
| UserLogin        |
+------------------+
5 rows in set (0.00 sec)

mysql>
```

2. DDL commands for your tables

2.1 Games (Main table)

```
CREATE TABLE Games (

    GameId INT NOT NULL PRIMARY KEY,

    GameName VARCHAR(255),

    ReleaseDate VARCHAR(30),

    RequiredAge INT,

    Metacritic INT,

    RecommendationCount INT,

    ControllerSupport BOOLEAN NOT NULL,

    IsFree BOOLEAN NOT NULL,

    PurchaseAvail BOOLEAN NOT NULL,

    SubscriptionAvail BOOLEAN NOT NULL,

    PlatformWindows BOOLEAN NOT NULL,

    PlatformLinux BOOLEAN NOT NULL,

    PlatformMac BOOLEAN NOT NULL,

    PCRequirements BOOLEAN NOT NULL,

    LinuxRequirements BOOLEAN NOT NULL,

    MacRequirements BOOLEAN NOT NULL,

    SinglePlayer BOOLEAN NOT NULL,

    Multiplayer BOOLEAN NOT NULL,

    Coop BOOLEAN NOT NULL,

    MMO BOOLEAN NOT NULL,

    InAppPurchase BOOLEAN NOT NULL,

    IncludeLevelEditor BOOLEAN NOT NULL,
```

```
    VRSupport BOOLEAN NOT NULL,
    Currency VARCHAR(50),
    Price REAL,
    NumberOfOwners INT,
    EstimatePlayers INT,
    GameDescription VARCHAR(2000),
    HeaderImage VARCHAR(2000),
    Reviews VARCHAR(2000)
);
```

2.2 RecommendedGames (Main table)

```
CREATE TABLE RecommendedGames (
    GameId INT NOT NULL PRIMARY KEY,
    GameName VARCHAR(255),
    Metacritic INT,
    RecommendationCount INT,
    Price REAL,
    GameDescription VARCHAR(2000),
    HeaderImage VARCHAR(2000),
    SimilarityScore REAL
);
```

2.3 UserLogin (Auxiliary table)

```
CREATE TABLE UserLogin (
    UserID INT NOT NULL PRIMARY KEY,
    Email VARCHAR(255) NOT NULL,
    Password VARCHAR(255) NOT NULL
);
```

2.4 UserInputGameList (Main table)

```
CREATE TABLE UserInputGameList (
  ListID INT NOT NULL PRIMARY KEY,
  UserID INT NOT NULL,
  GameID INT NOT NULL
);
```

2.5 Genre (Main table)

```
CREATE TABLE Genre (
  GameID INT NOT NULL PRIMARY KEY,
  Indie BOOLEAN  NOT NULL,
  Action BOOLEAN NOT NULL,
  Adventure BOOLEAN NOT NULL,
  Casual BOOLEAN NOT NULL,
  Strategy BOOLEAN NOT NULL,
  RPG BOOLEAN NOT NULL,
  Simulation BOOLEAN NOT NULL,
  Sports BOOLEAN NOT NULL,
  Racing BOOLEAN NOT NULL
);
```

3. Greater Than 1000 Entries in 4 Tables

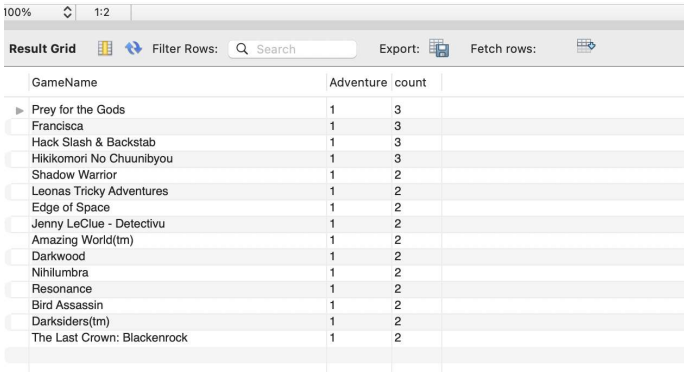
```
mysql> SELECT COUNT(*)
    -> FROM Games;
+----------+
| COUNT(*) |
+----------+
|    13209 |
+----------+
1 row in set (0.01 sec)

mysql> SELECT COUNT(*)
    -> FROM RecommendedGames;
+----------+
| COUNT(*) |
+----------+
|     1833 |
+----------+
1 row in set (0.00 sec)

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

mysql> SELECT COUNT(*)
    -> FROM UserInputGameList;
+----------+
| COUNT(*) |
+----------+
|     2001 |
+----------+
1 row in set (0.01 sec)
```

4. Two Advanced Queries

Advanced Query One

4.1.1 Advanced Query 1: Find the adventure games that users have inputted into their list as well as adventure games from all available games. (top 15 rows of the advanced query results).

```
1  USE main;
2
3  SELECT GameName, Adventure, count(GameId) as count
4  FROM UserInputGameList
5      NATURAL JOIN Games
6      NATURAL JOIN Genre
7  WHERE Adventure = 1
8  GROUP BY GameId
9  ORDER BY count DESC
10 LIMIT 15;
11
12
```

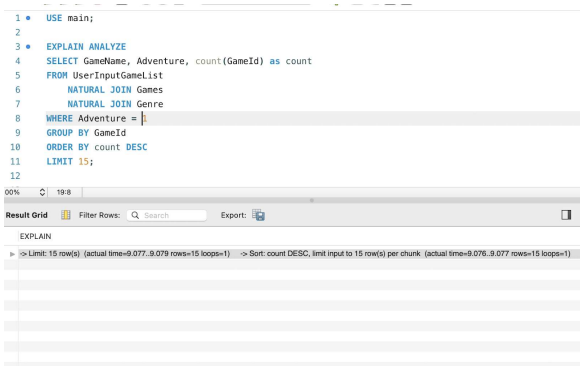


100% 1:2

Result Grid Filter Rows: Search Export: Fetch rows:

GameName	Adventure	count
Prey for the Gods	1	3
Francisca	1	3
Hack Slash & Backstab	1	3
Hikikomori No Chuunibyou	1	3
Shadow Warrior	1	2
Leonas Tricky Adventures	1	2
Edge of Space	1	2
Jenny LeClue - Detectivu	1	2
Amazing World(tm)	1	2
Darkwood	1	2
Nihilumbra	1	2
Resonance	1	2
Bird Assassin	1	2
Darksiders(tm)	1	2
The Last Crown: Blackenrock	1	2

4.1.2 Advanced Query 1: Before Indexing w/ Explain Analyze



```
1  -> Limit: 15 row(s)  (actual time=9.077..9.079 rows=15 loops=1)
2      -> Sort: count DESC, limit input to 15 row(s) per chunk  (actual
   time=9.076..9.077 rows=15 loops=1)
3          -> Table scan on <temporary>  (actual time=0.001..0.076 rows=601
   loops=1)
4              -> Aggregate using temporary table  (actual time=8.881..8.990
   rows=601 loops=1)
5                  -> Nested loop inner join  (cost=1602.30 rows=200) (actual
   time=0.109..8.209 rows=656 loops=1)
6                      -> Nested loop inner join  (cost=901.95 rows=2001)
   (actual time=0.055..5.296 rows=2001 loops=1)
7                          -> Table scan on UserInputGameList  (cost=201.60
   rows=2001) (actual time=0.039..0.557 rows=2001 loops=1)
8                          -> Single-row index lookup on Games using PRIMARY
   (GameId=UserInputGameList.GameID)  (cost=0.25 rows=1) (actual
   time=0.002..0.002 rows=1 loops=2001)
9                      -> Filter: (Genre.Adventure = 1)  (cost=0.25 rows=0)
   (actual time=0.001..0.001 rows=0 loops=2001)
10                         -> Single-row index lookup on Genre using PRIMARY
   (GameID=UserInputGameList.GameID)  (cost=0.25 rows=1) (actual
   time=0.001..0.001 rows=1 loops=2001)
11
```

4.1.3 Advanced Query 1: Index 1 w/Explain Analyze

```
USE main;

CREATE INDEX idx ON Games (GameName, ReleaseDate, Price);

EXPLAIN ANALYZE
SELECT GameName, Adventure, count(GameId) as count
FROM UserInputGameList
    NATURAL JOIN Games
    NATURAL JOIN Genre
WHERE Adventure = 1
GROUP BY GameId
ORDER BY count DESC
LIMIT 15;
```

Edit Data for EXPLAIN (VARCHAR)

Binary | Text

```
-> Limit: 15 row(s)  (actual time=9.142..9.144 rows=15 loops=1)
    -> Sort: count DESC, limit input to 15 row(s) per chunk  (actual time=9.142..9.143 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=0.002..0.063 rows=601 loops=1)
            -> Aggregate using temporary table  (actual time=8.891..8.987 rows=601 loops=1)
                -> Nested loop inner join  (cost=1602.30 rows=200) (actual time=0.183..8.258 rows=656 loops=1)
                    -> Nested loop inner join  (cost=901.95 rows=2001) (actual time=0.147..5.288 rows=2001 loops=1)
                        -> Table scan on UserInputGameList  (cost=201.60 rows=2001) (actual time=0.128..0.609 rows=2001
loops=1)
                        -> Single-row index lookup on Games using PRIMARY (GameId=UserInputGameList.GameID)  (cost=0.25
rows=1) (actual time=0.002..0.002 rows=1 loops=2001)
                    -> Filter: (Genre.Adventure = 1)  (cost=0.25 rows=0) (actual time=0.001..0.001 rows=0 loops=2001)
                        -> Single-row index lookup on Genre using PRIMARY (GameID=UserInputGameList.GameID)  (cost=0.25
rows=1) (actual time=0.001..0.001 rows=1 loops=2001)
```

Justification:

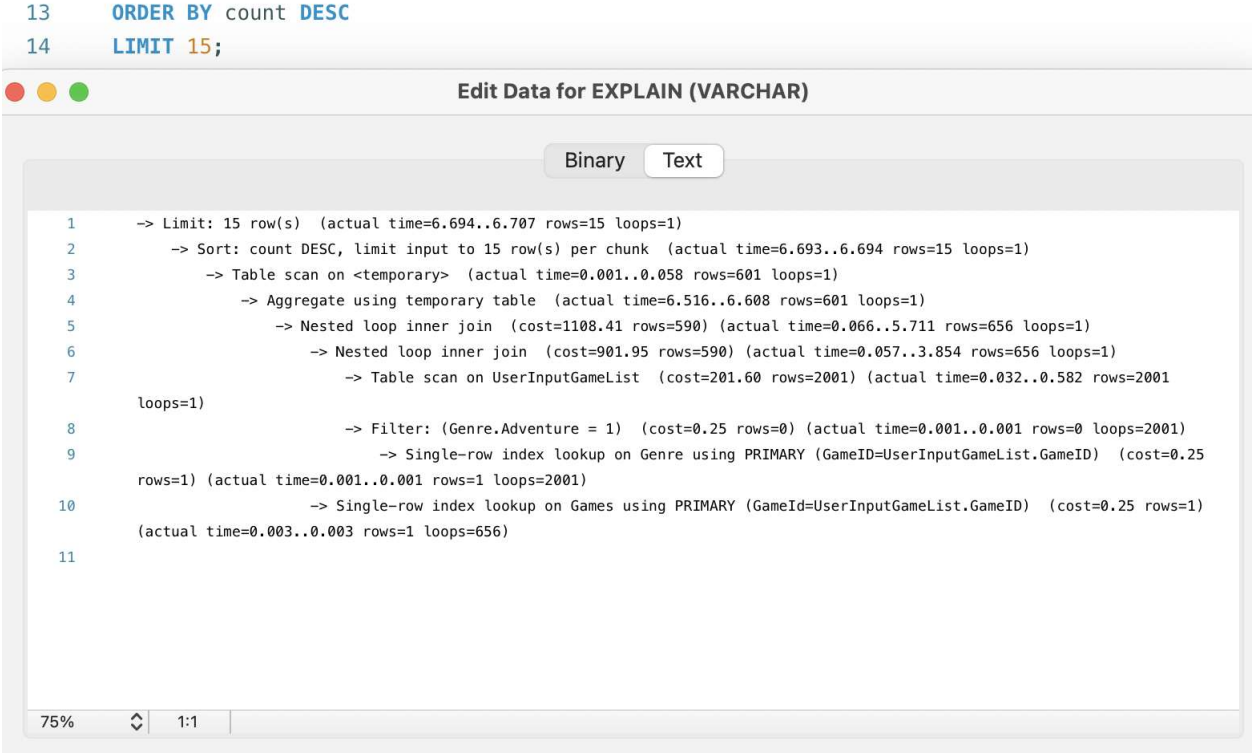
We chose these indexes because we will access GameName frequently

Indexing #	Init time/s	Read time/s
Before indexing	9.077	9.079
1	9.142	9.144

However, there is no improvement on the init time and read time, so indexing on GameName is probably not that helpful for our design.

4.1.4 Advanced Query 1: Index 2 w/ Explain Analyze

```
1  USE main;
2
3  CREATE INDEX idx_1 ON Games (GameId, GameName);
4  CREATE INDEX idx_2 ON Genre (Adventure);
5
6  EXPLAIN ANALYZE
7  SELECT GameName, Adventure, count(GameId) as count
8  FROM UserInputGameList
9      NATURAL JOIN Games
10     NATURAL JOIN Genre
11 WHERE Adventure = 1
12 GROUP BY GameId
13 ORDER BY count DESC
14 LIMIT 15;
```



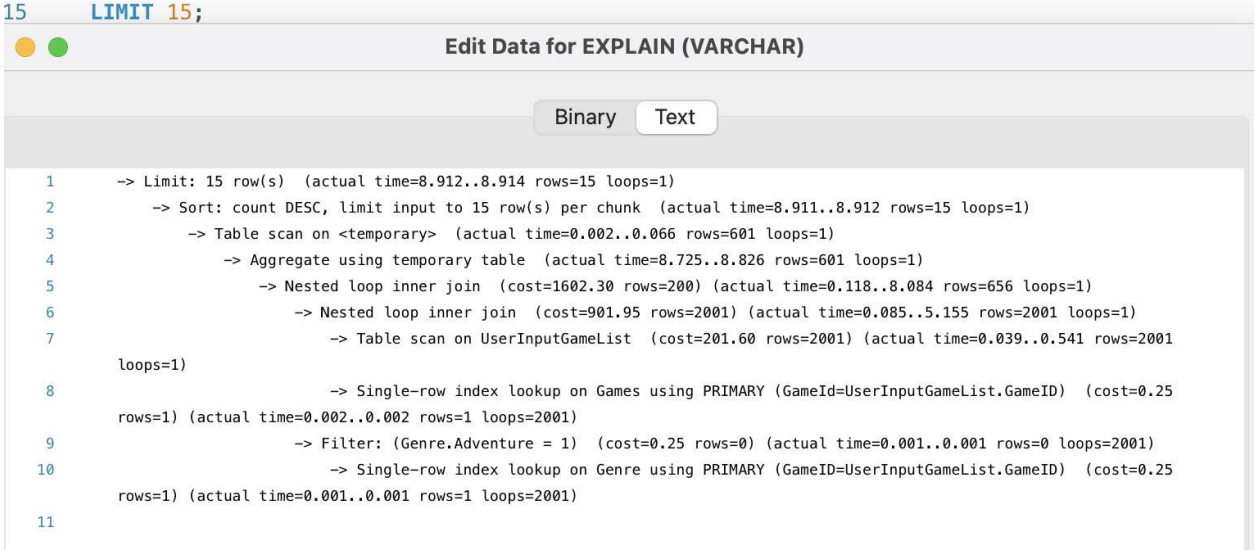
Justification:
We chose these indexes because we will access GameName, GameId and Adventure frequently.

Indexing #	Init time/s	Read time/s
Before indexing	9.077	9.079
2	6.694	6.707

There is considerable improvement in the init time and read time after implementing the indexing. We can see that the first cost of nested inner join drops from 1602 to 1108, which corresponds the join between table UserInputGameList and table Games.

4.1.5 Advanced Query 1: Index 3 w/ Explain Analyze

```sql
USE main;

CREATE INDEX idx_1 ON Games (GameName, NumberOfOwners, EstimatePlayers);
CREATE INDEX idx_2 ON Genre (Indie, Casual, Strategy);


EXPLAIN ANALYZE
SELECT GameName, Adventure, count(GameId) as count
FROM UserInputGameList
    NATURAL JOIN Games
    NATURAL JOIN Genre
WHERE Adventure = 1
GROUP BY GameId
ORDER BY count DESC
LIMIT 15;
```



Justification:
We chose these indexes because these are the attributes that will be accessed frequently.

Indexing #	Init time/s	Read time/s
Before indexing	9.077	9.079
3	8.912	8.914

There is slight improvement in the init time and read time after implementing the indexing.

4.1.6 Advanced Query 1: Indexing Analysis

Indexing #	Init time/s	Read time/s	Improved?
Before indexing	9.077	9.079	
1	9.142	9.144	Not significant
2	6.694	6.707	Significant
3	8.912	8.914	Not significant

Indexing#2 brings significant improvement for our advanced query 1 while the other two do not. That is because the (GameId, GameName) tuple helps to reduce the accessing time.

Advanced Query 2

4.2.1 Advanced Query 1: Find the games where more than 5000 people recommend the game and more than 500000 people own the game. (top 15 rows of the advanced query results).

```
1  USE main;
2
3  (SELECT GameName, RecommendationCount, NumberOfOwners
4  FROM UserInputGameList
5      NATURAL JOIN Games
6      NATURAL JOIN Genre
7  WHERE RecommendationCount > 5000
8  ORDER BY RecommendationCount ASC)
9
10 UNION
11
12 (SELECT GameName, RecommendationCount, NumberOfOwners
13 FROM UserInputGameList
14     NATURAL JOIN Games
15     NATURAL JOIN Genre
16 WHERE NumberOfOwners > 500000
17 ORDER BY NumberOfOwners DESC)
18 LIMIT 15;
```

20:14

Result Grid Filter Rows: Search Export:

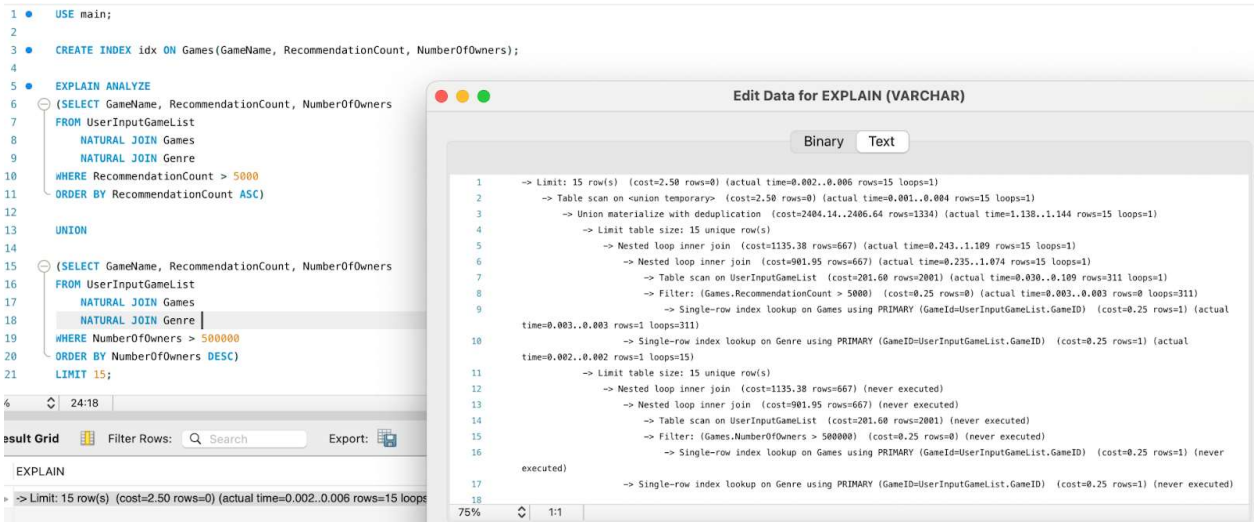
GameName	RecommendationCo...	NumberOfOwne...
Sid Meiers Civilization(r) V	85750	9711421
RAGE	8176	1325035
Resident Evil(tm) 5/ Biohazard 5(r)	5145	839535
Train Simulator 2016	8701	937819
Trine Enchanted Edition	6891	1533260
Call of Duty(r): Modern Warfare(r) 3	19250	3398675
Anno 2070(tm)	7284	865541
Aliens: Colonial Marines Collection	6035	434385
Darksiders(tm)	7445	2506769
Warhammer 40000: Dawn of War II: Retribution	5503	1886399
Amnesia: The Dark Descent	14322	3235825
Terraria	139819	8165607
Batman: Arkham City - Game of the Year Edition	16418	3396164
Tomb Raider	55844	4453968
Awesomenauts	17240	1689652

4.2.2 Advanced Query 2: Before Indexing w/ Explain Analyze

```sql
USE main;

EXPLAIN ANALYZE
(SELECT GameName, RecommendationCount, NumberOfOwners
FROM UserInputGameList
    NATURAL JOIN Games
    NATURAL JOIN Genre
WHERE RecommendationCount > 5000
ORDER BY RecommendationCount ASC)

UNION

(SELECT GameName, RecommendationCount, NumberOfOwners
FROM UserInputGameList
    NATURAL JOIN Games
    NATURAL JOIN Genre
WHERE NumberOfOwners > 500000
ORDER BY NumberOfOwners DESC)
LIMIT 15;
```

```
-> Limit: 15 row(s)  (cost=2.50 rows=0) (actual time=0.002..0.004 rows=15 loops=1)
    -> Table scan on <union temporary>  (cost=2.50 rows=0) (actual time=0.001..0.002 rows=15 loops=1)
        -> Union materialize with deduplication  (cost=2404.14..2406.64 rows=1334) (actual time=0.450..0.453 rows=15 loops=1)
            -> Limit table size: 15 unique row(s)
                -> Nested loop inner join  (cost=1135.38 rows=667) (actual time=0.066..0.424 rows=18 loops=1)
                    -> Nested loop inner join  (cost=901.95 rows=667) (actual time=0.061..0.396 rows=18 loops=1)
                        -> Index scan on UserInputGameList using idx_userinputgamelist_gameid  (cost=201.60 rows=2001) (actual time=0.031..0.053 rows=131 loops=1)
                        -> Filter: (Games.RecommendationCount > 5000)  (cost=0.25 rows=0) (actual time=0.002..0.003 rows=0 loops=131)
                            -> Single-row index lookup on Games using PRIMARY (GameId=UserInputGameList.GameID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=131)
                    -> Single-row index lookup on Genre using PRIMARY (GameID=UserInputGameList.GameID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=18)
            -> Limit table size: 15 unique row(s)
                -> Nested loop inner join  (cost=1135.38 rows=667) (never executed)
                    -> Nested loop inner join  (cost=901.95 rows=667) (never executed)
                        -> Index scan on UserInputGameList using idx_userinputgamelist_gameid  (cost=201.60 rows=2001) (never executed)
                        -> Filter: (Games.NumberOfOwners > 500000)  (cost=0.25 rows=0) (never executed)
                            -> Single-row index lookup on Games using PRIMARY (GameId=UserInputGameList.GameID)  (cost=0.25 rows=1) (never executed)
                    -> Single-row index lookup on Genre using PRIMARY (GameID=UserInputGameList.GameID)  (cost=0.25 rows=1) (never executed)
```

4.2.3 Advanced Query 2: Index 1 w/ Explain Analyze



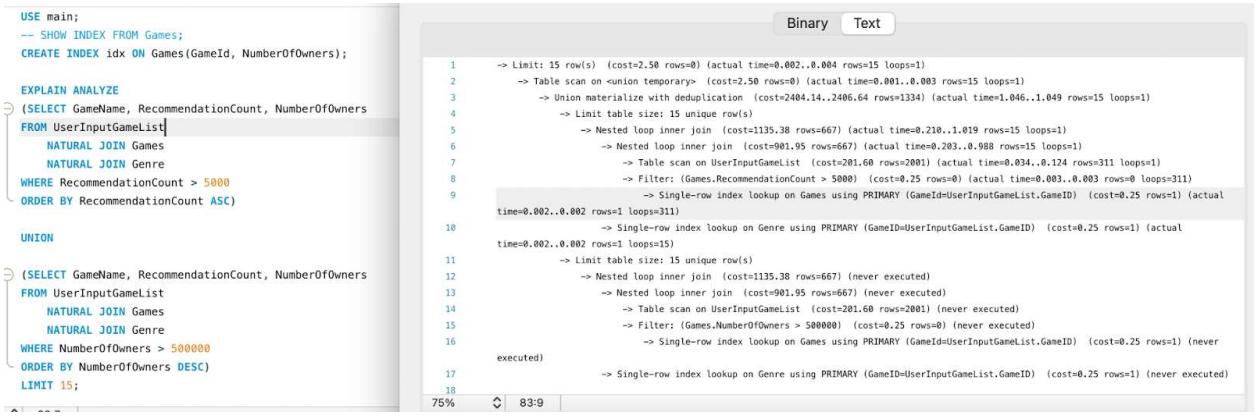
Justification:

We chose these indexes because these are the attributes that will be accessed frequently.

Indexing #	Init time/s	Read time/s
Before indexing	0.002	0.004
1	0.002	0.006

There is no improvement in the init time and read time after implementing the indexing.

4.2.4 Advanced Query 2: Index 2 w/ Explain Analyze



```
USE main;
-- SHOW INDEX FROM Games;
CREATE INDEX idx ON Games(GameId, NumberOfOwners);

EXPLAIN ANALYZE
(SELECT GameName, RecommendationCount, NumberOfOwners
FROM UserInputGameList
    NATURAL JOIN Games
    NATURAL JOIN Genre
WHERE RecommendationCount > 5000
ORDER BY RecommendationCount ASC)

UNION

(SELECT GameName, RecommendationCount, NumberOfOwners
FROM UserInputGameList
    NATURAL JOIN Games
    NATURAL JOIN Genre
WHERE NumberOfOwners > 500000
ORDER BY NumberOfOwners DESC)
LIMIT 15;
```

Binary | Text

```
1  -> Limit: 15 row(s)  (cost=2.50 rows=0) (actual time=0.002..0.004 rows=15 loops=1)
2      -> Table scan on <union temporary>  (cost=2.50 rows=0) (actual time=0.001..0.003 rows=15 loops=1)
3          -> Union materialize with deduplication  (cost=2404.14..2406.64 rows=1334) (actual time=1.046..1.049 rows=15 loops=1)
4              -> Limit table size: 15 unique row(s)
5                  -> Nested loop inner join  (cost=1135.38 rows=667) (actual time=0.210..1.019 rows=15 loops=1)
6                      -> Nested loop inner join  (cost=901.95 rows=667) (actual time=0.203..0.988 rows=15 loops=1)
7                          -> Table scan on UserInputGameList  (cost=201.60 rows=2001) (actual time=0.034..0.124 rows=311 loops=1)
8                          -> Filter: (Games.RecommendationCount > 5000)  (cost=0.25 rows=0) (actual time=0.003..0.003 rows=0 loops=311)
9                              -> Single-row index lookup on Games using PRIMARY (GameId=UserInputGameList.GameID)  (cost=0.25 rows=1) (actual
   time=0.002..0.002 rows=1 loops=311)
10                     -> Single-row index lookup on Genre using PRIMARY (GameID=UserInputGameList.GameID)  (cost=0.25 rows=1) (actual
   time=0.002..0.002 rows=1 loops=15)
11             -> Limit table size: 15 unique row(s)
12                 -> Nested loop inner join  (cost=1135.38 rows=667) (never executed)
13                     -> Nested loop inner join  (cost=901.95 rows=667) (never executed)
14                         -> Table scan on UserInputGameList  (cost=201.60 rows=2001) (never executed)
15                         -> Filter: (Games.NumberOfOwners > 500000)  (cost=0.25 rows=0) (never executed)
16                             -> Single-row index lookup on Games using PRIMARY (GameId=UserInputGameList.GameID)  (cost=0.25 rows=1) (never
   executed)
17                     -> Single-row index lookup on Genre using PRIMARY (GameID=UserInputGameList.GameID)  (cost=0.25 rows=1) (never executed)
18
```

75% 83:9

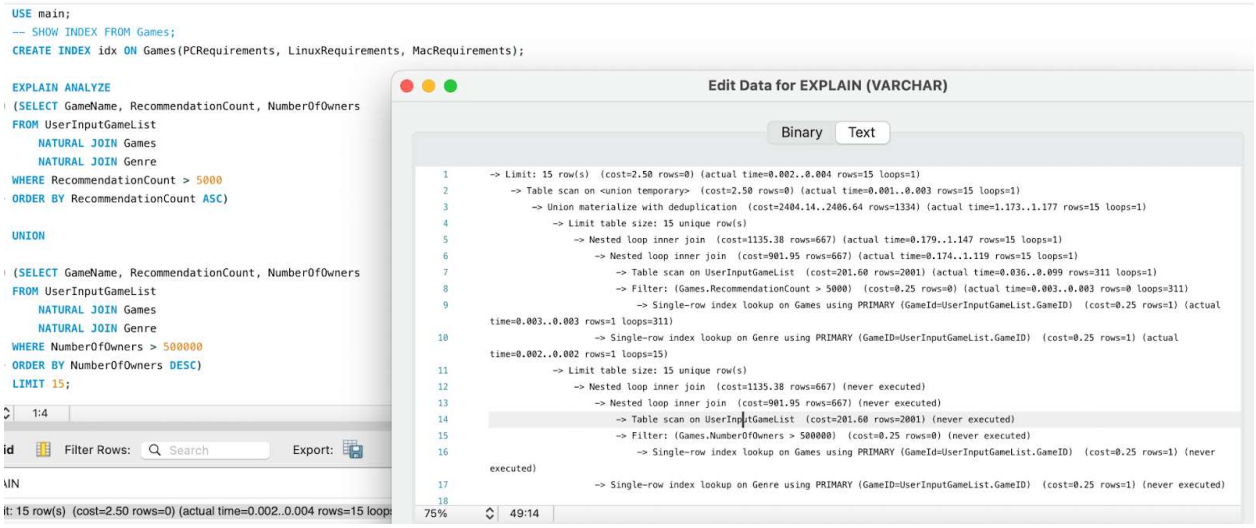
Justification:

We chose these indexes because these are the attributes that will be accessed frequently.

Indexing #	Init time/s	Read time/s
Before indexing	0.002	0.004
2	0.002	0.004

There is no improvement in the init time and read time after implementing the indexing.

4.2.5 Advanced Query 2: Index 3 w/ Explain Analyze



Justification:

We chose these indexes because these are the attributes that may have relationship with the query.

Indexing #	Init time/s	Read time/s
Before indexing	0.002	0.004
1	0.002	0.004

There is no improvement in the init time and read time after implementing the indexing.

4.2.6 Advanced Query 2: Indexing Analysis

Indexing #	Init time/s	Read time/s
Before indexing	0.002	0.004
1	0.002	0.004
2	0.002	0.006
3	0.002	0.004

All three indexing does not have considerableimprovement on the run-time, this is probably because the whole process already takes linear time before indexing.