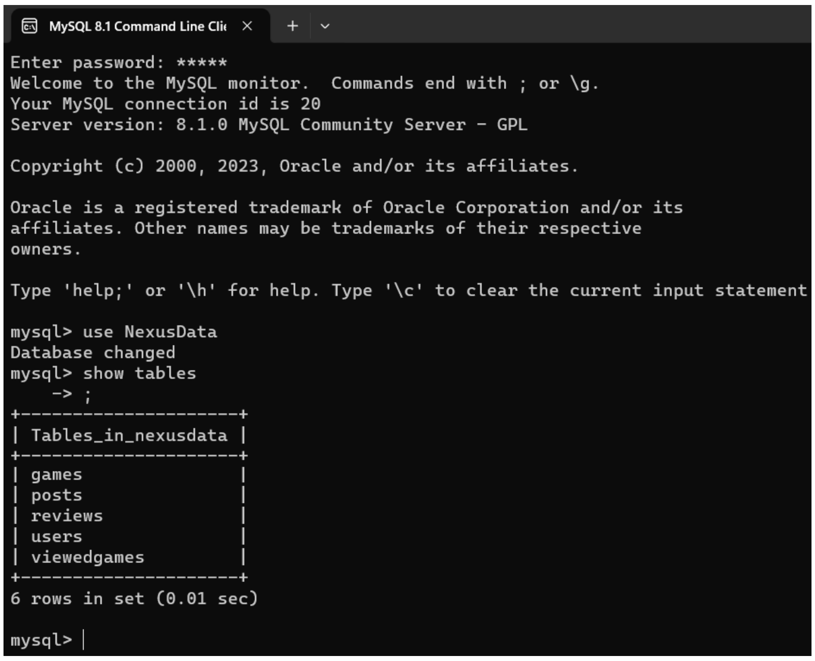
Project Track 1 Stage 3

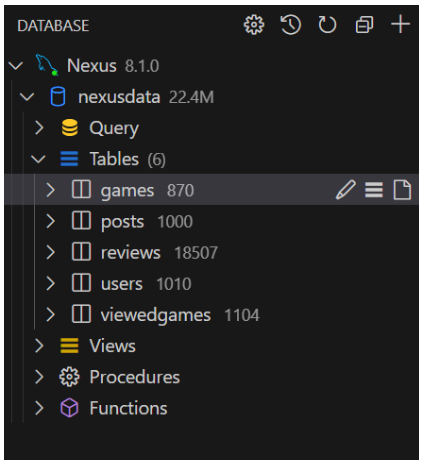
1. Database implementation

a) Screenshots

Connection:

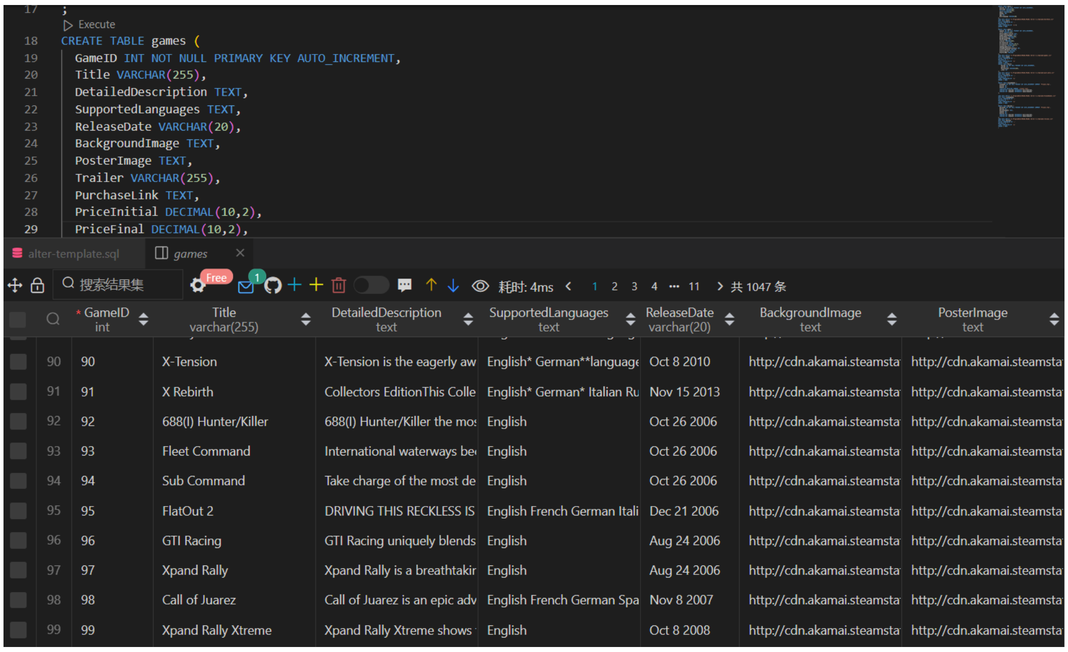


```
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 20
Server version: 8.1.0 MySQL Community Server - GPL

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> use NexusData
Database changed
mysql> show tables
    -> ;
+----------------------+
| Tables_in_nexusdata |
+----------------------+
| games                |
| posts                |
| reviews              |
| users                |
| viewedgames          |
+----------------------+
6 rows in set (0.01 sec)

mysql> |
```



```
DATABASE
Nexus 8.1.0
  nexusdata 22.4M
    Query
    Tables (6)
      games 870
      posts 1000
      reviews 18507
      users 1010
      viewedgames 1104
    Views
    Procedures
    Functions
```

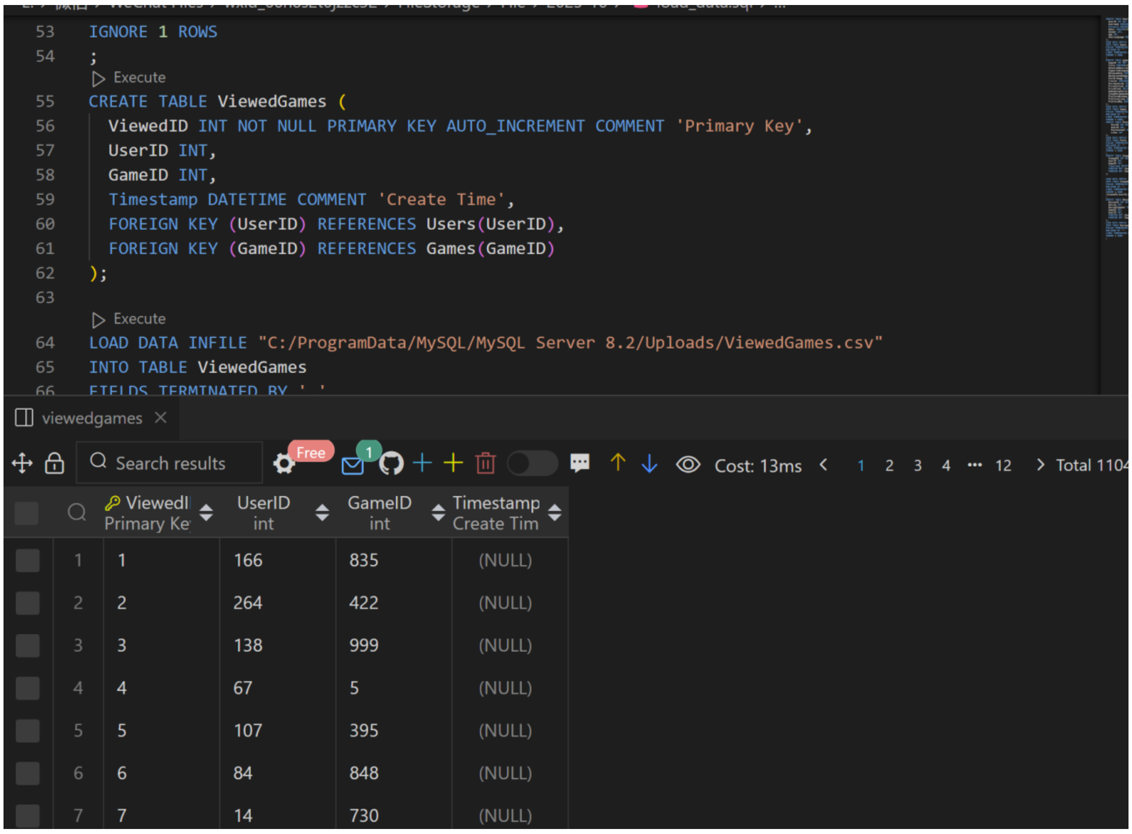
Games:



```sql
;
CREATE TABLE games (
  GameID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
  Title VARCHAR(255),
  DetailedDescription TEXT,
  SupportedLanguages TEXT,
  ReleaseDate VARCHAR(20),
  BackgroundImage TEXT,
  PosterImage TEXT,
  Trailer VARCHAR(255),
  PurchaseLink TEXT,
  PriceInitial DECIMAL(10,2),
  PriceFinal DECIMAL(10,2),
```

	GameID int	Title varchar(255)	DetailedDescription text	SupportedLanguages text	ReleaseDate varchar(20)	BackgroundImage text	PosterImage text
90	90	X-Tension	X-Tension is the eagerly aw	English* German**language	Oct 8 2010	http://cdn.akamai.steamsta	http://cdn.akamai.steamsta
91	91	X Rebirth	Collectors EditionThis Colle	English* German* Italian Ru	Nov 15 2013	http://cdn.akamai.steamsta	http://cdn.akamai.steamsta
92	92	688(I) Hunter/Killer	688(I) Hunter/Killer the mos	English	Oct 26 2006	http://cdn.akamai.steamsta	http://cdn.akamai.steamsta
93	93	Fleet Command	International waterways be	English	Oct 26 2006	http://cdn.akamai.steamsta	http://cdn.akamai.steamsta
94	94	Sub Command	Take charge of the most de	English	Oct 26 2006	http://cdn.akamai.steamsta	http://cdn.akamai.steamsta
95	95	FlatOut 2	DRIVING THIS RECKLESS IS	English French German Itali	Dec 21 2006	http://cdn.akamai.steamsta	http://cdn.akamai.steamsta
96	96	GTI Racing	GTI Racing uniquely blends	English	Aug 24 2006	http://cdn.akamai.steamsta	http://cdn.akamai.steamsta
97	97	Xpand Rally	Xpand Rally is a breathtakir	English	Aug 24 2006	http://cdn.akamai.steamsta	http://cdn.akamai.steamsta
98	98	Call of Juarez	Call of Juarez is an epic adv	English French German Spa	Nov 8 2007	http://cdn.akamai.steamsta	http://cdn.akamai.steamsta
99	99	Xpand Rally Xtreme	Xpand Rally Xtreme shows	English	Oct 8 2008	http://cdn.akamai.steamsta	http://cdn.akamai.steamsta

ViewedGames:



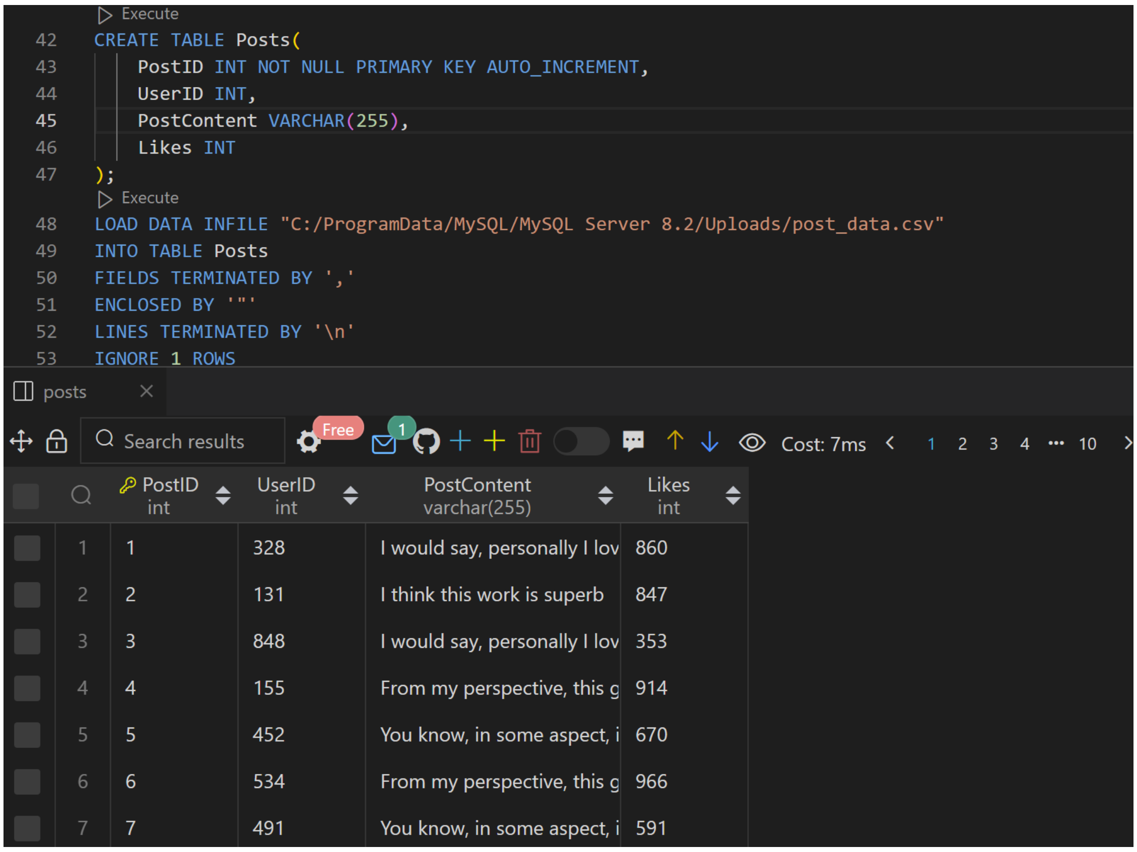
```
IGNORE 1 ROWS
;
▷ Execute
CREATE TABLE ViewedGames (
  ViewedID INT NOT NULL PRIMARY KEY AUTO_INCREMENT COMMENT 'Primary Key',
  UserID INT,
  GameID INT,
  Timestamp DATETIME COMMENT 'Create Time',
  FOREIGN KEY (UserID) REFERENCES Users(UserID),
  FOREIGN KEY (GameID) REFERENCES Games(GameID)
);

▷ Execute
LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server 8.2/Uploads/ViewedGames.csv"
INTO TABLE ViewedGames
FIELDS TERMINATED BY ','
```

viewedgames

Cost: 13ms 1 2 3 4 ... 12 Total 1104

	ViewedI Primary Ke	UserID int	GameID int	Timestamp Create Tim
1	1	166	835	(NULL)
2	2	264	422	(NULL)
3	3	138	999	(NULL)
4	4	67	5	(NULL)
5	5	107	395	(NULL)
6	6	84	848	(NULL)
7	7	14	730	(NULL)

Posts:



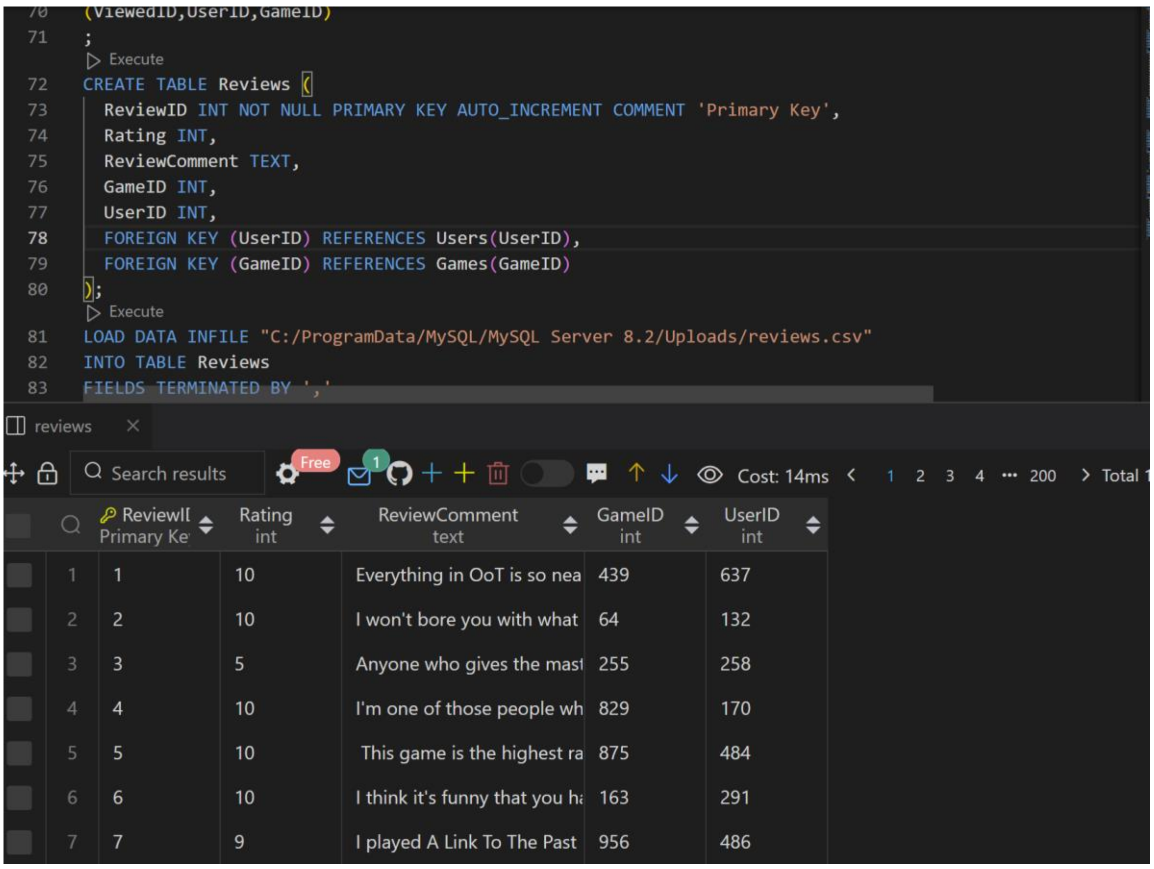
```
▷ Execute
CREATE TABLE Posts(
    PostID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    UserID INT,
    PostContent VARCHAR(255),
    Likes INT
);
▷ Execute
LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server 8.2/Uploads/post_data.csv"
INTO TABLE Posts
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
```

posts

Cost: 7ms 1 2 3 4 ... 10

	PostID int	UserID int	PostContent varchar(255)	Likes int
1	1	328	I would say, personally I lov	860
2	2	131	I think this work is superb	847
3	3	848	I would say, personally I lov	353
4	4	155	From my perspective, this g	914
5	5	452	You know, in some aspect, i	670
6	6	534	From my perspective, this g	966
7	7	491	You know, in some aspect, i	591

Reviews:



```
(ViewedID,UserID,GameID)
;
CREATE TABLE Reviews (
  ReviewID INT NOT NULL PRIMARY KEY AUTO_INCREMENT COMMENT 'Primary Key',
  Rating INT,
  ReviewComment TEXT,
  GameID INT,
  UserID INT,
  FOREIGN KEY (UserID) REFERENCES Users(UserID),
  FOREIGN KEY (GameID) REFERENCES Games(GameID)
);
LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server 8.2/Uploads/reviews.csv"
INTO TABLE Reviews
FIELDS TERMINATED BY ','
```

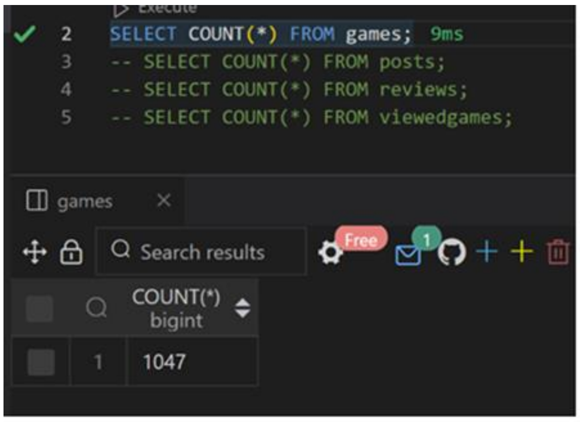
	ReviewII Primary Ke	Rating int	ReviewComment text	GameID int	UserID int
1	1	10	Everything in OoT is so nea	439	637
2	2	10	I won't bore you with what	64	132
3	3	5	Anyone who gives the mast	255	258
4	4	10	I'm one of those people wh	829	170
5	5	10	This game is the highest ra	875	484
6	6	10	I think it's funny that you ha	163	291
7	7	9	I played A Link To The Past	956	486

b) Commands

```
CREATE TABLE games (
  GameID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
  Title VARCHAR(255),
  DetailedDescription TEXT,
  SupportedLanguages TEXT,
  ReleaseDate VARCHAR(20),
  BackgroundImage TEXT,
  PosterImage TEXT,
  Trailer VARCHAR(255),
  PurchaseLink TEXT,
  PriceInitial DECIMAL(10,2),
  PriceFinal DECIMAL(10,2),
  EmbeddingVectors INT,
  SteamRecommendationCount INT,
  PlatformWindows BOOLEAN,
  PlatformLinux BOOLEAN,
  PlatformMac BOOLEAN
);

CREATE TABLE Users (
   UserID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
   Username VARCHAR(20),
   Password VARCHAR(30),
   Email VARCHAR(30),
   Gender INT,
   Age INT,
   UserLanguage VARCHAR(50)
);

CREATE TABLE ViewedGames (
   ViewedID INT NOT NULL PRIMARY KEY AUTO_INCREMENT COMMENT
'Primary Key',
   UserID INT,
   GameID INT,
   Timestamp DATETIME COMMENT 'Create Time',
   FOREIGN KEY (UserID) REFERENCES Users(UserID),
   FOREIGN KEY (GameID) REFERENCES Games(GameID)
);

CREATE TABLE Reviews (
   ReviewID INT NOT NULL PRIMARY KEY AUTO_INCREMENT COMMENT
'Primary Key',
   Rating INT,
   ReviewComment TEXT,
   GameID INT,
   UserID INT,
   FOREIGN KEY (UserID) REFERENCES Users(UserID),
   FOREIGN KEY (GameID) REFERENCES Games(GameID)
);


CREATE TABLE Posts(
      PostID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
      UserID INT,
      PostContent VARCHAR(255),
      Likes INT,
         FOREIGN KEY (UserID) REFERENCES Users(UserID),
         FOREIGN KEY (GameID) REFERENCES Games(GameID)

);
```

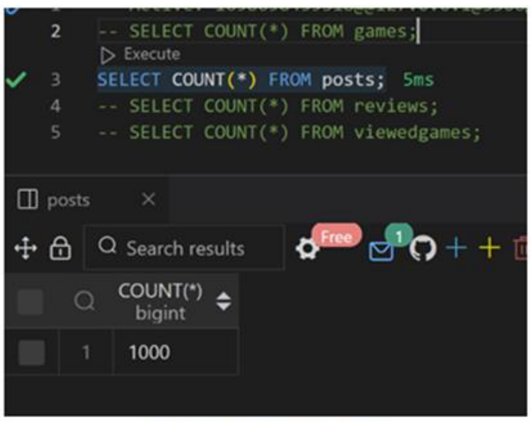
```
LOAD DATA INFILE "C:/ProgramData/MyQSQL/MySQL Server
8.1/Uploads/UserData.csv"
INTO TABLE Users
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;

LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server
8.1/Uploads/games.csv"
INTO TABLE Games
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;

LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server
8.1/Uploads/post_data.csv"
INTO TABLE Posts
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;

LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server
8.1/Uploads/ViewedGames.csv"
INTO TABLE ViewedGames
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;

LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server
8.1/Uploads/reviews.csv"
INTO TABLE Reviews
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
```

c) Count



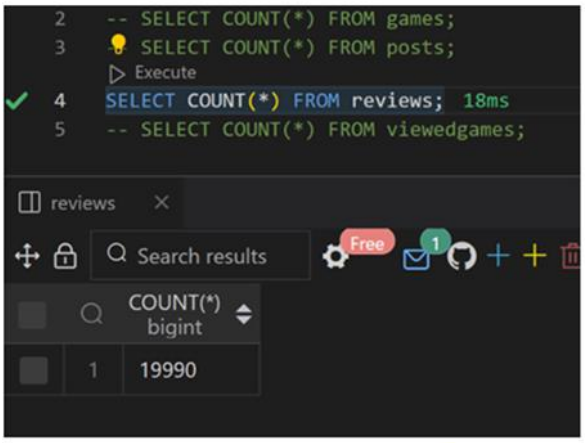
```
SELECT COUNT(*) FROM games;  9ms
-- SELECT COUNT(*) FROM posts;
-- SELECT COUNT(*) FROM reviews;
-- SELECT COUNT(*) FROM viewedgames;
```

```
games
COUNT(*)
bigint
1   1047
```



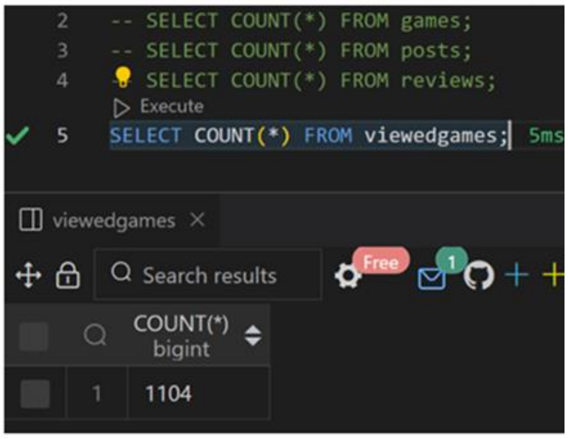
```
-- SELECT COUNT(*) FROM games;
SELECT COUNT(*) FROM posts;  5ms
-- SELECT COUNT(*) FROM reviews;
-- SELECT COUNT(*) FROM viewedgames;
```

```
posts
COUNT(*)
bigint
1   1000
```



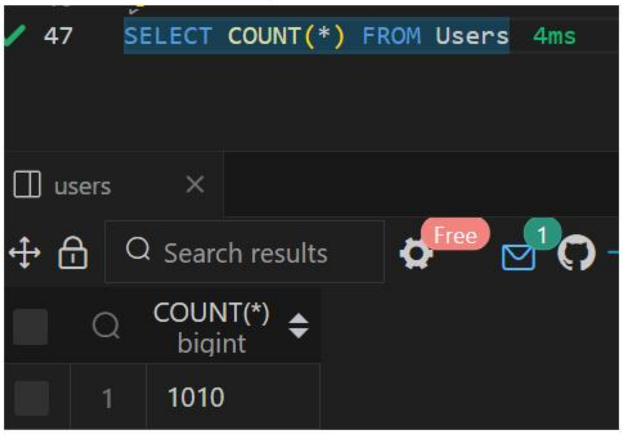
```
-- SELECT COUNT(*) FROM games;
-- SELECT COUNT(*) FROM posts;
SELECT COUNT(*) FROM reviews;  18ms
-- SELECT COUNT(*) FROM viewedgames;
```

```
reviews
COUNT(*)
bigint
1   19990
```



```
-- SELECT COUNT(*) FROM games;
-- SELECT COUNT(*) FROM posts;
-- SELECT COUNT(*) FROM reviews;
SELECT COUNT(*) FROM viewedgames;  5ms
```

```
viewedgames
COUNT(*)
bigint
1   1104
```



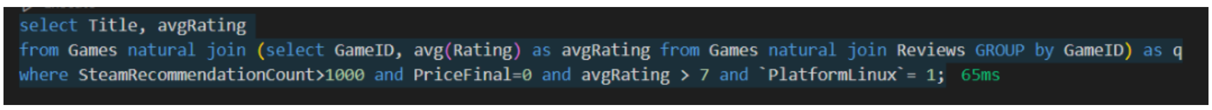
```
SELECT COUNT(*) FROM Users  4ms
```

```
users
COUNT(*)
bigint
1   1010
```

2. Advanced Queries

a) Query 1

Command:

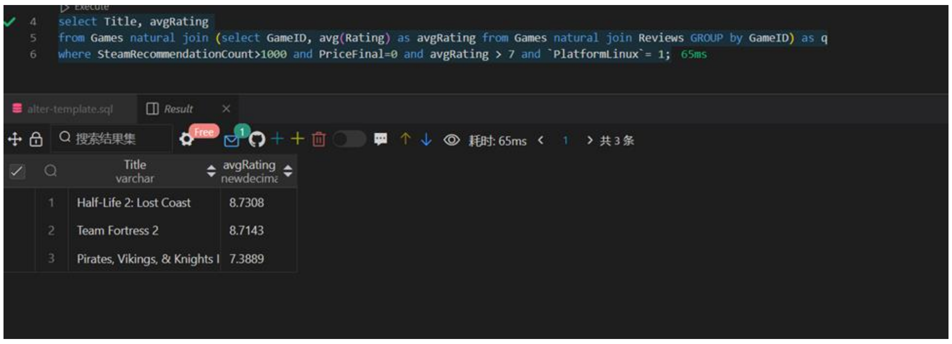


```
select Title, avgRating
from Games natural join (select GameID, avg(Rating) as avgRating from Games natural join Reviews GROUP by GameID) as q
where SteamRecommendationCount>1000 and PriceFinal=0 and avgRating > 7 and `PlatformLinux`= 1;  65ms
```

Functionality:

Get names and ratings of the games which has Steam Recommendation Count greater than 1000, are currently free, has a rating of above 7, and support Linux as a platform. (Used to list games for *Linux Games Collection: free while fun!*)

Result: (There are only three rows in the result)



```
select Title, avgRating
from Games natural join (select GameID, avg(Rating) as avgRating from Games natural join Reviews GROUP by GameID) as q
where SteamRecommendationCount>1000 and PriceFinal=0 and avgRating > 7 and `PlatformLinux`= 1;  65ms
```

耗时: 65ms 共 3 条

	Title varchar	avgRating newdecima
1	Half-Life 2: Lost Coast	8.7308
2	Team Fortress 2	8.7143
3	Pirates, Vikings, & Knights I	7.3889

b) Query 2

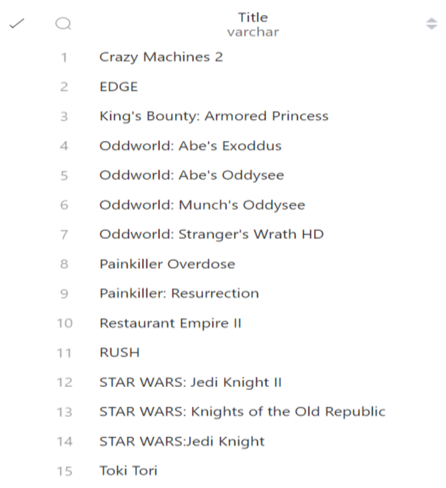
Command:

```
select DISTINCT Title
from (select GameID,Title
    from games where PriceInitial>PriceFinal AND SteamRecommendationCount>=100)AS T
    JOIN viewedgames ON (T.GameID=viewedgames.GameID) Natural join users
WHERE Username="bmfzkkw" ORDER BY Title;
```

Functionality:

Recommend to the user games he has viewed, which have discounts and at least 100 people recommend.

Result:

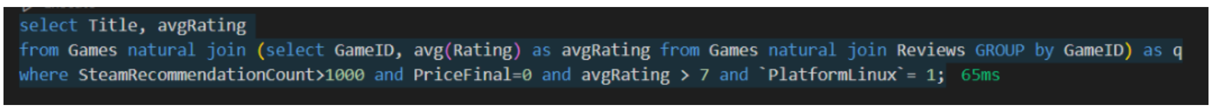


	Title varchar
1	Crazy Machines 2
2	EDGE
3	King's Bounty: Armored Princess
4	Oddworld: Abe's Exoddus
5	Oddworld: Abe's Oddysee
6	Oddworld: Munch's Oddysee
7	Oddworld: Stranger's Wrath HD
8	Painkiller Overdose
9	Painkiller: Resurrection
10	Restaurant Empire II
11	RUSH
12	STAR WARS: Jedi Knight II
13	STAR WARS: Knights of the Old Republic
14	STAR WARS:Jedi Knight
15	Toki Tori

3. Indexing Analysis

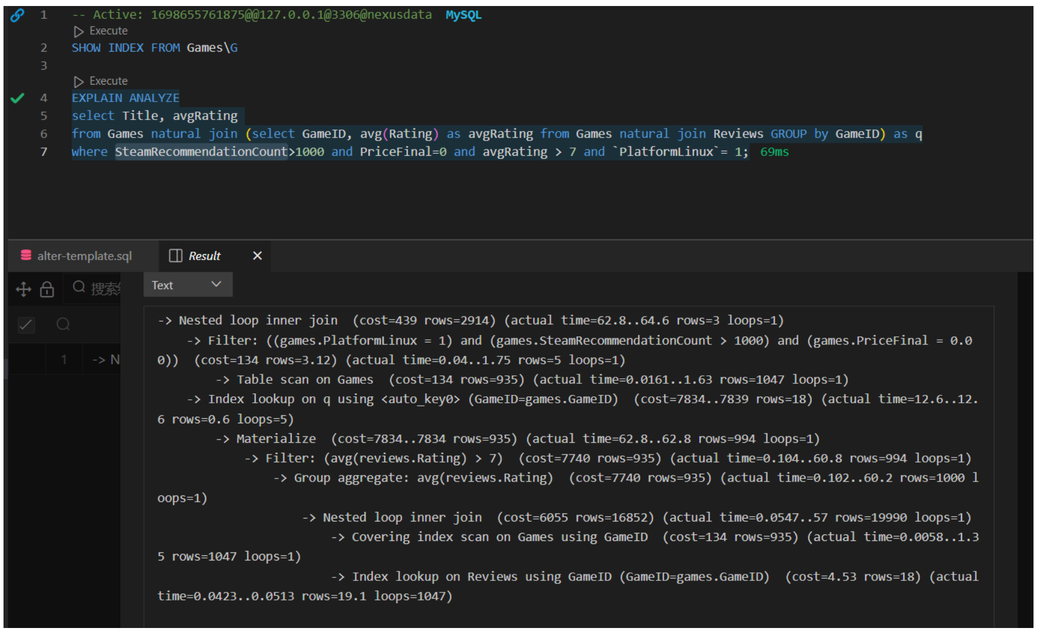
3.1 Query 1

Command:



```
select Title, avgRating
from Games natural join (select GameID, avg(Rating) as avgRating from Games natural join Reviews GROUP by GameID) as q
where SteamRecommendationCount>1000 and PriceFinal=0 and avgRating > 7 and `PlatformLinux`= 1;  65ms
```

Initial Efficiency:



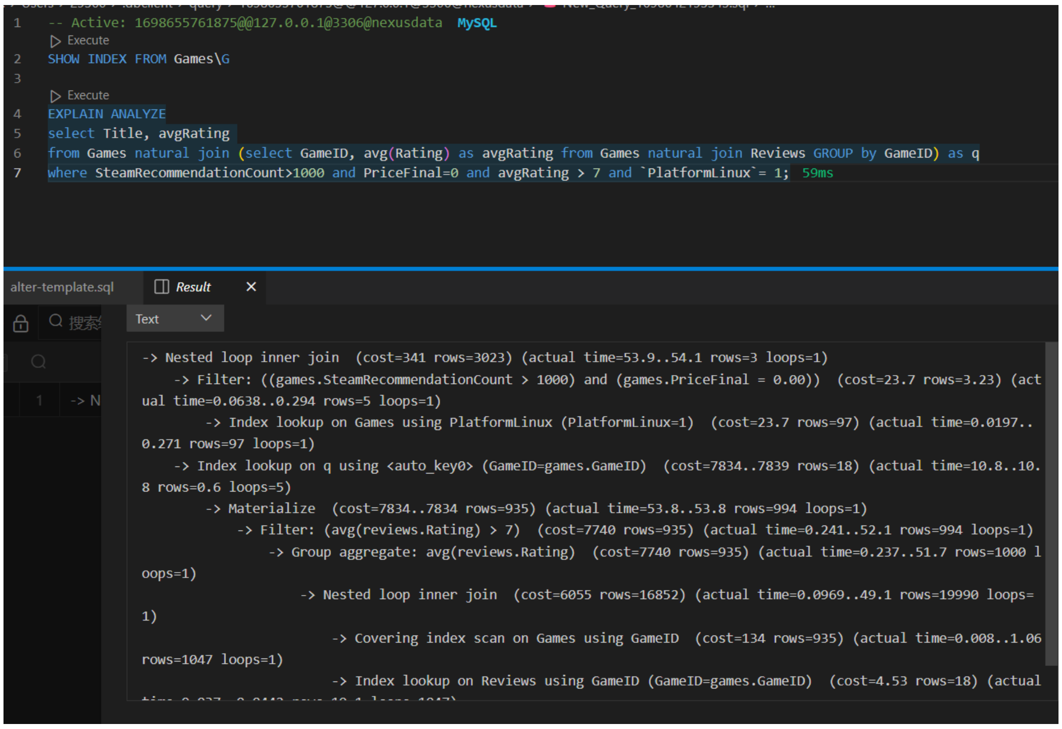
```
-- Active: 1698655761875@@127.0.0.1@3306@nexusdata  MySQL
SHOW INDEX FROM Games\G

EXPLAIN ANALYZE
select Title, avgRating
from Games natural join (select GameID, avg(Rating) as avgRating from Games natural join Reviews GROUP by GameID) as q
where SteamRecommendationCount>1000 and PriceFinal=0 and avgRating > 7 and `PlatformLinux`= 1;  69ms
```

```
-> Nested loop inner join  (cost=439 rows=2914) (actual time=62.8..64.6 rows=3 loops=1)
    -> Filter: ((games.PlatformLinux = 1) and (games.SteamRecommendationCount > 1000) and (games.PriceFinal = 0.0
0))  (cost=134 rows=3.12) (actual time=0.04..1.75 rows=5 loops=1)
        -> Table scan on Games  (cost=134 rows=935) (actual time=0.0161..1.63 rows=1047 loops=1)
    -> Index lookup on q using <auto_key0> (GameID=games.GameID)  (cost=7834..7839 rows=18) (actual time=12.6..12.
6 rows=0.6 loops=5)
        -> Materialize  (cost=7834..7834 rows=935) (actual time=62.8..62.8 rows=994 loops=1)
            -> Filter: (avg(reviews.Rating) > 7)  (cost=7740 rows=935) (actual time=0.104..60.8 rows=994 loops=1)
                -> Group aggregate: avg(reviews.Rating)  (cost=7740 rows=935) (actual time=0.102..60.2 rows=1000 l
oops=1)
                    -> Nested loop inner join  (cost=6055 rows=16852) (actual time=0.0547..57 rows=19990 loops=1)
                        -> Covering index scan on Games using GameID  (cost=134 rows=935) (actual time=0.0058..1.3
5 rows=1047 loops=1)
                        -> Index lookup on Reviews using GameID (GameID=games.GameID)  (cost=4.53 rows=18) (actual
time=0.0423..0.0513 rows=19.1 loops=1047)
```

Cost is 439.

a) **Index1: PlatformLinux**

Screenshot:



```
-- Active: 1698655761875@@127.0.0.1@3306@nexusdata  MySQL
SHOW INDEX FROM Games\G

EXPLAIN ANALYZE
select Title, avgRating
from Games natural join (select GameID, avg(Rating) as avgRating from Games natural join Reviews GROUP by GameID) as q
where SteamRecommendationCount>1000 and PriceFinal=0 and avgRating > 7 and `PlatformLinux`= 1;  59ms
```

```
-> Nested loop inner join  (cost=341 rows=3023) (actual time=53.9..54.1 rows=3 loops=1)
    -> Filter: ((games.SteamRecommendationCount > 1000) and (games.PriceFinal = 0.00))  (cost=23.7 rows=3.23) (act
ual time=0.0638..0.294 rows=5 loops=1)
        -> Index lookup on Games using PlatformLinux (PlatformLinux=1)  (cost=23.7 rows=97) (actual time=0.0197..
0.271 rows=97 loops=1)
    -> Index lookup on q using <auto_key0> (GameID=games.GameID)  (cost=7834..7839 rows=18) (actual time=10.8..10.
8 rows=0.6 loops=5)
        -> Materialize  (cost=7834..7834 rows=935) (actual time=53.8..53.8 rows=994 loops=1)
            -> Filter: (avg(reviews.Rating) > 7)  (cost=7740 rows=935) (actual time=0.241..52.1 rows=994 loops=1)
                -> Group aggregate: avg(reviews.Rating)  (cost=7740 rows=935) (actual time=0.237..51.7 rows=1000 l
oops=1)
                    -> Nested loop inner join  (cost=6055 rows=16852) (actual time=0.0969..49.1 rows=19990 loops=
1)
                        -> Covering index scan on Games using GameID  (cost=134 rows=935) (actual time=0.008..1.06
rows=1047 loops=1)
                        -> Index lookup on Reviews using GameID (GameID=games.GameID)  (cost=4.53 rows=18) (actual
```

Nested Loop inner Cost reduces to 341.

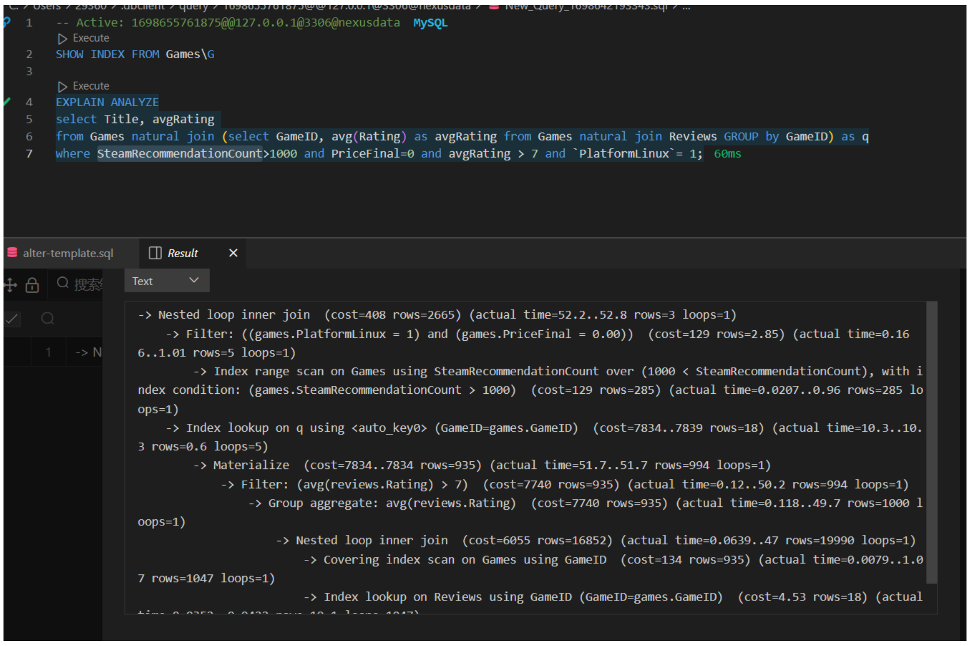
Filter of SteamRecommendationCount and PriceFinal Cost reduces to 23.7 (compared to 134)

Explanation:

We add this index since Platform Linux is a restriction in Where clause. the database can quickly identify the relevant rows, reducing the number of disk reads and improving query performance and less rows are filtered in index lookup process (linuxplatform = 1), thus reducing the second filter's cost as well.

b) **Index2: SteamRecommendationCount**

Screenshot:



```
-- Active: 1698655761875@@127.0.0.1@3306@nexusdata  MySQL
SHOW INDEX FROM Games\G

EXPLAIN ANALYZE
select Title, avgRating
from Games natural join (select GameID, avg(Rating) as avgRating from Games natural join Reviews GROUP by GameID) as q
where SteamRecommendationCount>1000 and PriceFinal=0 and avgRating > 7 and `PlatformLinux`= 1;  60ms
```

```
-> Nested loop inner join  (cost=408 rows=2665) (actual time=52.2..52.8 rows=3 loops=1)
    -> Filter: ((games.PlatformLinux = 1) and (games.PriceFinal = 0.00))  (cost=129 rows=2.85) (actual time=0.16
6..1.01 rows=5 loops=1)
        -> Index range scan on Games using SteamRecommendationCount over (1000 < SteamRecommendationCount), with i
ndex condition: (games.SteamRecommendationCount > 1000)  (cost=129 rows=285) (actual time=0.0207..0.96 rows=285 lo
ops=1)
    -> Index lookup on q using <auto_key0> (GameID=games.GameID)  (cost=7834..7839 rows=18) (actual time=10.3..10.
3 rows=0.6 loops=5)
        -> Materialize  (cost=7834..7834 rows=935) (actual time=51.7..51.7 rows=994 loops=1)
            -> Filter: (avg(reviews.Rating) > 7)  (cost=7740 rows=935) (actual time=0.12..50.2 rows=994 loops=1)
                -> Group aggregate: avg(reviews.Rating)  (cost=7740 rows=935) (actual time=0.118..49.7 rows=1000 l
oops=1)
                    -> Nested loop inner join  (cost=6055 rows=16852) (actual time=0.0639..47 rows=19990 loops=1)
                        -> Covering index scan on Games using GameID  (cost=134 rows=935) (actual time=0.0079..1.0
7 rows=1047 loops=1)
                        -> Index lookup on Reviews using GameID (GameID=games.GameID)  (cost=4.53 rows=18) (actual
```

Nested loop inner Cost reduces to 408.

Filter of SteamRecommendationCount and PriceFinal Cost reduces to 129 (compared to 134)

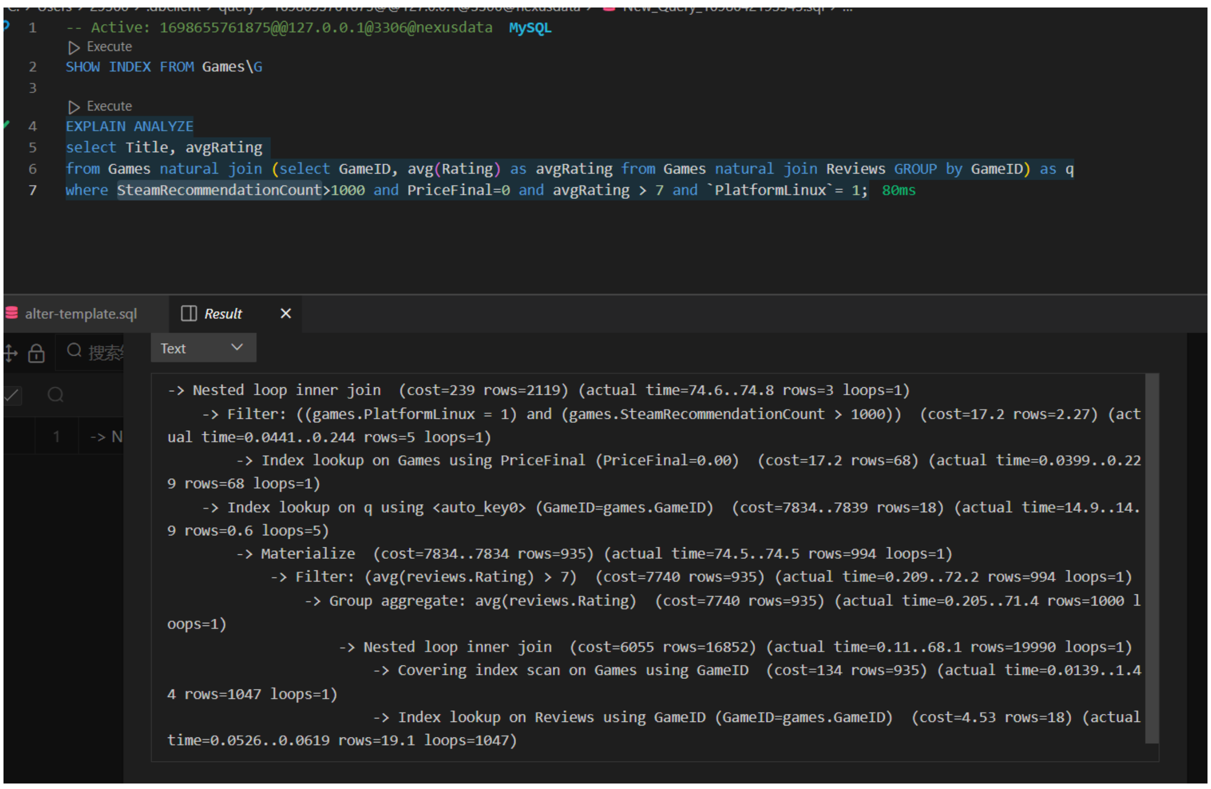
Index Range scan on games using SteamRecommendationCount is created (from 0 to 129)

Explanation:

We add this index since SteamRecommendationCount is a restriction in Where clause. There is no improvement in Cost may because most of the SteamRecommendationCount is larger than 1000 and no much help to location.

c) **Index3: PriceFinal**

Screenshot:



```
-- Active: 1698655761875@@127.0.0.1@3306@nexusdata  MySQL
▷ Execute
SHOW INDEX FROM Games\G

▷ Execute
EXPLAIN ANALYZE
select Title, avgRating
from Games natural join (select GameID, avg(Rating) as avgRating from Games natural join Reviews GROUP by GameID) as q
where SteamRecommendationCount>1000 and PriceFinal=0 and avgRating > 7 and `PlatformLinux`= 1;  80ms
```

```
-> Nested loop inner join  (cost=239 rows=2119) (actual time=74.6..74.8 rows=3 loops=1)
    -> Filter: ((games.PlatformLinux = 1) and (games.SteamRecommendationCount > 1000))  (cost=17.2 rows=2.27) (act
ual time=0.0441..0.244 rows=5 loops=1)
        -> Index lookup on Games using PriceFinal (PriceFinal=0.00)  (cost=17.2 rows=68) (actual time=0.0399..0.22
9 rows=68 loops=1)
    -> Index lookup on q using <auto_key0> (GameID=games.GameID)  (cost=7834..7839 rows=18) (actual time=14.9..14.
9 rows=0.6 loops=5)
        -> Materialize  (cost=7834..7834 rows=935) (actual time=74.5..74.5 rows=994 loops=1)
            -> Filter: (avg(reviews.Rating) > 7)  (cost=7740 rows=935) (actual time=0.209..72.2 rows=994 loops=1)
                -> Group aggregate: avg(reviews.Rating)  (cost=7740 rows=935) (actual time=0.205..71.4 rows=1000 l
oops=1)
                    -> Nested loop inner join  (cost=6055 rows=16852) (actual time=0.11..68.1 rows=19990 loops=1)
                        -> Covering index scan on Games using GameID  (cost=134 rows=935) (actual time=0.0139..1.4
4 rows=1047 loops=1)
                        -> Index lookup on Reviews using GameID (GameID=games.GameID)  (cost=4.53 rows=18) (actual
time=0.0526..0.0619 rows=19.1 loops=1047)
```

Nested loop inner Cost reduces to 239.

Filter of platformlinux and SteamRecommendationCount reduces to 17.2 compared to 134

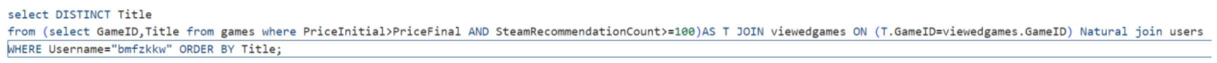
Index lookup on Games using pricefinal is created (from 0 to 17.2)

Explanation:

Faster Data Retrieval: When an index is added to Games, it creates a separate data structure that organizes the values of Username in a specific order. This allows the database engine to locate and retrieve the required data more efficiently. Especially, PriceFinal is used in a WHERE clause, the database can quickly identify the relevant rows, reducing the number of disks reads and improving query performance. Index lookup before Filter of platformlinux and SteamRecommendationCount help to filter less rows and reduce the following costs.

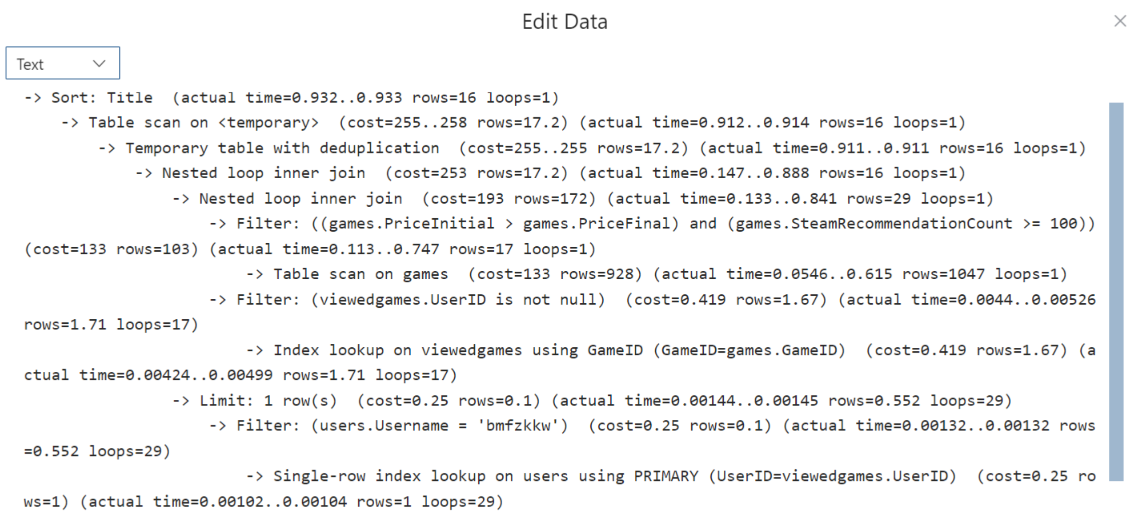
3.2 Query 2

Command:



```
select DISTINCT Title
from (select GameID,Title from games where PriceInitial>PriceFinal AND SteamRecommendationCount>=100)AS T JOIN viewedgames ON (T.GameID=viewedgames.GameID) Natural join users
WHERE Username="bmfzkkw" ORDER BY Title;
```

Initial Efficiency:



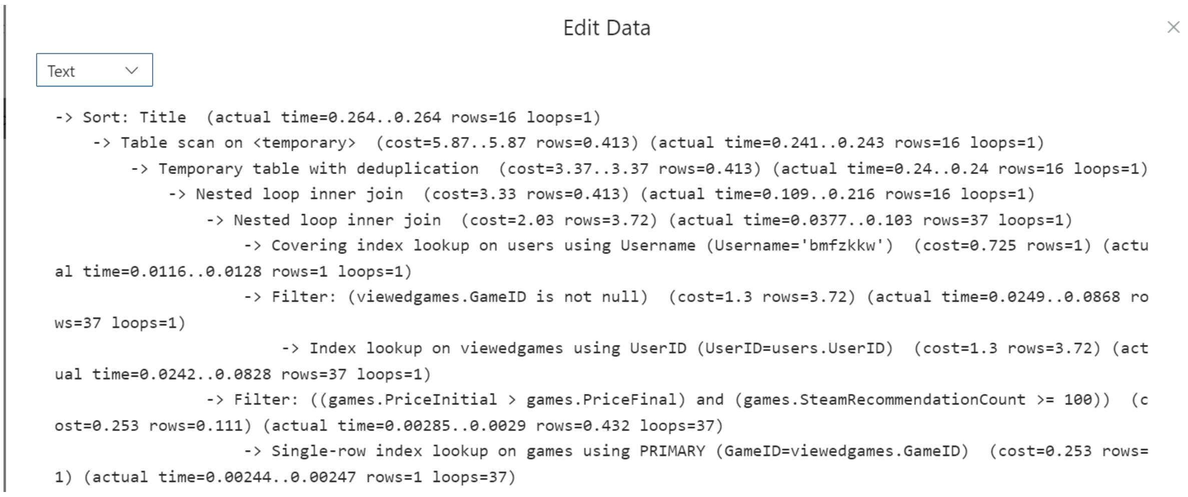
Edit Data

Text

```
-> Sort: Title  (actual time=0.932..0.933 rows=16 loops=1)
    -> Table scan on <temporary>  (cost=255..258 rows=17.2) (actual time=0.912..0.914 rows=16 loops=1)
        -> Temporary table with deduplication  (cost=255..255 rows=17.2) (actual time=0.911..0.911 rows=16 loops=1)
            -> Nested loop inner join  (cost=253 rows=17.2) (actual time=0.147..0.888 rows=16 loops=1)
                -> Nested loop inner join  (cost=193 rows=172) (actual time=0.133..0.841 rows=29 loops=1)
                    -> Filter: ((games.PriceInitial > games.PriceFinal) and (games.SteamRecommendationCount >= 100))
(cost=133 rows=103) (actual time=0.113..0.747 rows=17 loops=1)
                        -> Table scan on games  (cost=133 rows=928) (actual time=0.0546..0.615 rows=1047 loops=1)
                    -> Filter: (viewedgames.UserID is not null)  (cost=0.419 rows=1.67) (actual time=0.0044..0.00526
rows=1.71 loops=17)
                        -> Index lookup on viewedgames using GameID (GameID=games.GameID)  (cost=0.419 rows=1.67) (a
ctual time=0.00424..0.00499 rows=1.71 loops=17)
                -> Limit: 1 row(s)  (cost=0.25 rows=0.1) (actual time=0.00144..0.00145 rows=0.552 loops=29)
                    -> Filter: (users.Username = 'bmfzkkw')  (cost=0.25 rows=0.1) (actual time=0.00132..0.00132 rows
=0.552 loops=29)
                        -> Single-row index lookup on users using PRIMARY (UserID=viewedgames.UserID)  (cost=0.25 ro
ws=1) (actual time=0.00102..0.00104 rows=1 loops=29)
```

a) Index1 (select Username as index):

Screenshot:



Edit Data

Text

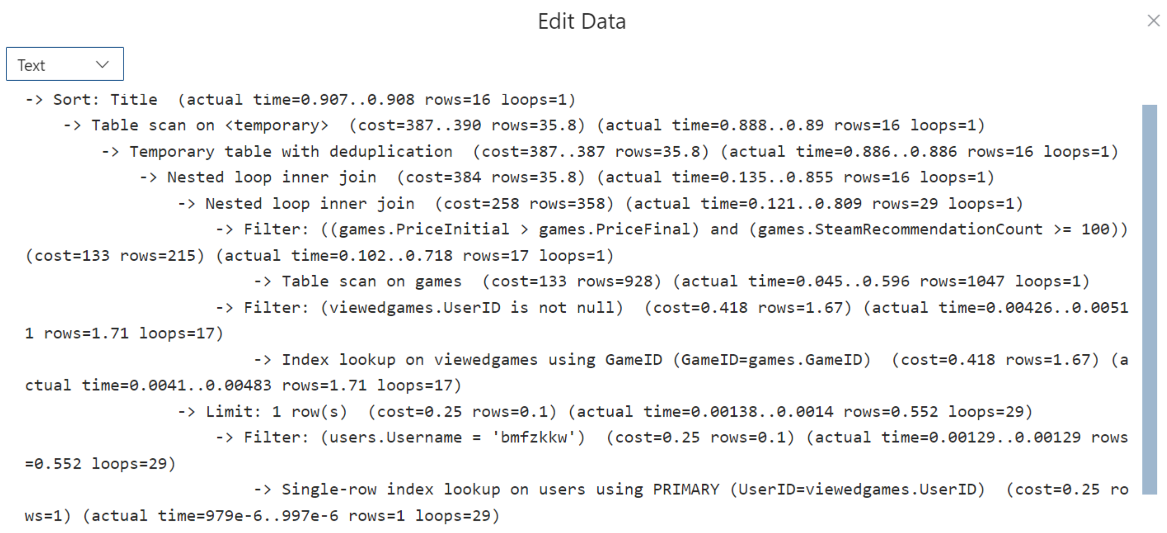
```
-> Sort: Title  (actual time=0.264..0.264 rows=16 loops=1)
    -> Table scan on <temporary>  (cost=5.87..5.87 rows=0.413) (actual time=0.241..0.243 rows=16 loops=1)
        -> Temporary table with deduplication  (cost=3.37..3.37 rows=0.413) (actual time=0.24..0.24 rows=16 loops=1)
            -> Nested loop inner join  (cost=3.33 rows=0.413) (actual time=0.109..0.216 rows=16 loops=1)
                -> Nested loop inner join  (cost=2.03 rows=3.72) (actual time=0.0377..0.103 rows=37 loops=1)
                    -> Covering index lookup on users using Username (Username='bmfzkkw')  (cost=0.725 rows=1) (actu
al time=0.0116..0.0128 rows=1 loops=1)
                    -> Filter: (viewedgames.GameID is not null)  (cost=1.3 rows=3.72) (actual time=0.0249..0.0868 ro
ws=37 loops=1)
                        -> Index lookup on viewedgames using UserID (UserID=users.UserID)  (cost=1.3 rows=3.72) (act
ual time=0.0242..0.0828 rows=37 loops=1)
                -> Filter: ((games.PriceInitial > games.PriceFinal) and (games.SteamRecommendationCount >= 100))  (c
ost=0.253 rows=0.111) (actual time=0.00285..0.0029 rows=0.432 loops=37)
                    -> Single-row index lookup on games using PRIMARY (GameID=viewedgames.GameID)  (cost=0.253 rows=
1) (actual time=0.00244..0.00247 rows=1 loops=37)
```

Explanation:

For steps Table scan on <temporary>, temporary table with deduplication, and nested loop innerjoin, the cost of index user name is greatly smaller than the cost of original one, which is because When an index is added to Username, it creates a separate data structure that organizes the values of Username in a specific order. This allows the database engine to locate and retrieve the required data more efficiently. Especially, Username is used in a WHERE clause, the database can quickly identify the relevant rows, reducing the number of disk reads and improving query performance.

b) **Index2 (select SteamRecommendationCount as index):**

Screenshot:



Edit Data

Text

```
-> Sort: Title  (actual time=0.907..0.908 rows=16 loops=1)
    -> Table scan on <temporary>  (cost=387..390 rows=35.8) (actual time=0.888..0.89 rows=16 loops=1)
        -> Temporary table with deduplication  (cost=387..387 rows=35.8) (actual time=0.886..0.886 rows=16 loops=1)
            -> Nested loop inner join  (cost=384 rows=35.8) (actual time=0.135..0.855 rows=16 loops=1)
                -> Nested loop inner join  (cost=258 rows=358) (actual time=0.121..0.809 rows=29 loops=1)
                    -> Filter: ((games.PriceInitial > games.PriceFinal) and (games.SteamRecommendationCount >= 100))
(cost=133 rows=215) (actual time=0.102..0.718 rows=17 loops=1)
                        -> Table scan on games  (cost=133 rows=928) (actual time=0.045..0.596 rows=1047 loops=1)
                    -> Filter: (viewedgames.UserID is not null)  (cost=0.418 rows=1.67) (actual time=0.00426..0.0051
1 rows=1.71 loops=17)
                        -> Index lookup on viewedgames using GameID (GameID=games.GameID)  (cost=0.418 rows=1.67) (a
ctual time=0.0041..0.00483 rows=1.71 loops=17)
                -> Limit: 1 row(s)  (cost=0.25 rows=0.1) (actual time=0.00138..0.0014 rows=0.552 loops=29)
                    -> Filter: (users.Username = 'bmfzkkw')  (cost=0.25 rows=0.1) (actual time=0.00129..0.00129 rows
=0.552 loops=29)
                        -> Single-row index lookup on users using PRIMARY (UserID=viewedgames.UserID)  (cost=0.25 ro
ws=1) (actual time=979e-6..997e-6 rows=1 loops=29)
```

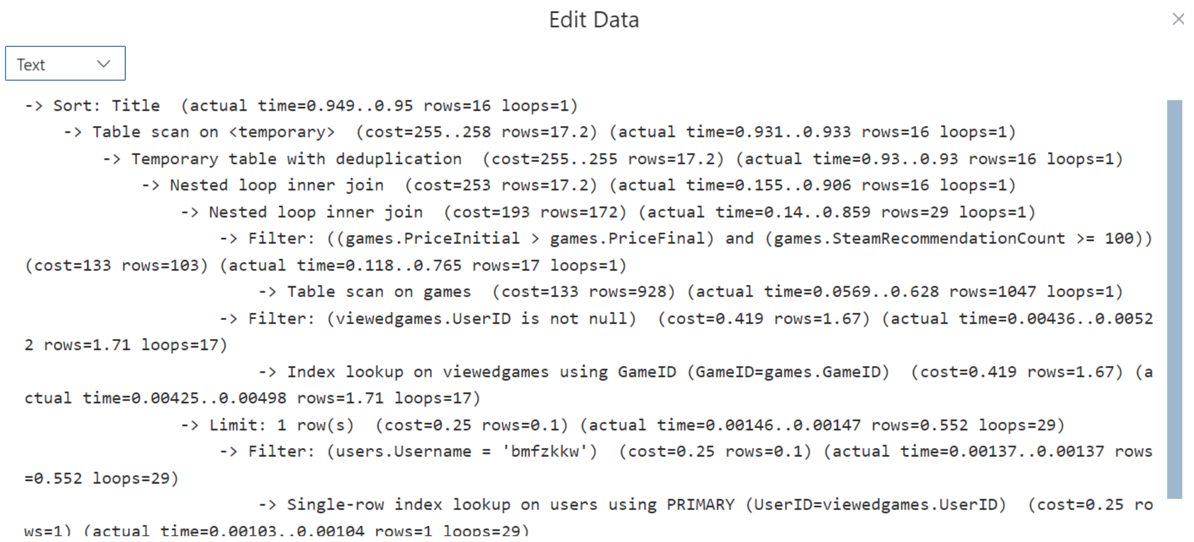
Explanation:

For steps Table scan on <temporary>, temporary table with deduplication, and nested loop innerjoin, the cost of index SteamRecommendationCount is even a little bit bigger than the original method, which means this index can't improve the performance of query. That is because an index is added to SteamRecommendationCount that is not used in the WHERE clause, it may not have a significant impact on the query performance. This is because the index is not directly involved in the filtering or selection of rows.

c) **Index3 (select title as index:**

Screenshot:

Cost is 253



Edit Data

Text

```
-> Sort: Title  (actual time=0.949..0.95 rows=16 loops=1)
    -> Table scan on <temporary>  (cost=255..258 rows=17.2) (actual time=0.931..0.933 rows=16 loops=1)
        -> Temporary table with deduplication  (cost=255..255 rows=17.2) (actual time=0.93..0.93 rows=16 loops=1)
            -> Nested loop inner join  (cost=253 rows=17.2) (actual time=0.155..0.906 rows=16 loops=1)
                -> Nested loop inner join  (cost=193 rows=172) (actual time=0.14..0.859 rows=29 loops=1)
                    -> Filter: ((games.PriceInitial > games.PriceFinal) and (games.SteamRecommendationCount >= 100))
(cost=133 rows=103) (actual time=0.118..0.765 rows=17 loops=1)
                        -> Table scan on games  (cost=133 rows=928) (actual time=0.0569..0.628 rows=1047 loops=1)
                    -> Filter: (viewedgames.UserID is not null)  (cost=0.419 rows=1.67) (actual time=0.00436..0.0052
2 rows=1.71 loops=17)
                        -> Index lookup on viewedgames using GameID (GameID=games.GameID)  (cost=0.419 rows=1.67) (a
ctual time=0.00425..0.00498 rows=1.71 loops=17)
                -> Limit: 1 row(s)  (cost=0.25 rows=0.1) (actual time=0.00146..0.00147 rows=0.552 loops=29)
                    -> Filter: (users.Username = 'bmfzkkw')  (cost=0.25 rows=0.1) (actual time=0.00137..0.00137 rows
=0.552 loops=29)
                        -> Single-row index lookup on users using PRIMARY (UserID=viewedgames.UserID)  (cost=0.25 ro
ws=1) (actual time=0.00103..0.00104 rows=1 loops=29)
```

Explanation:

No cost is changed here. Since Title isn't in WHERE Clause, it doesn't influence the time of query at all. When the index is not utilized in the WHERE clause, the database may need to perform a full table scan or access a significant portion of the data, regardless of the presence of an index.