

Project Track 1: Stage 3

Ace_DB

1. Database Implementation:

a. Screenshot of connection:

The screenshot shows the Google Cloud console interface. At the top, there's a navigation bar with 'SQL' and 'Instances' tabs. Below this, a table lists the instances. The instance 'pt1-032' is highlighted, showing it's a MySQL 8.0 instance with a public IP address of 104.198.185.40. Below the table, a 'CLOUD SHELL' terminal window is open, displaying several SQL commands and their results. The commands include counting rows in 'Review', 'Upvoted', and 'User' tables, and an attempt to drop a table 'GameFamily' which fails due to a foreign key constraint.

Instance ID	Type	Public IP address	Private IP address	Instance connection name	High availability	Location	Storage used	Labels
pt1-032	MySQL 8.0	104.198.185.40		pt1-032-us-central1-pt1-032	ADD	us-central1-b	1 GB of 10 GB	

```
mysql> SELECT COUNT(*) FROM Review;
+-----+
| COUNT(*) |
+-----+
|      8422 |
+-----+
1 row in set (0.01 sec)

mysql> SELECT COUNT(*) FROM Upvoted ;
+-----+
| COUNT(*) |
+-----+
|      8397 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT COUNT(*) FROM User ;
+-----+
| COUNT(*) |
+-----+
|      4311 |
+-----+
1 row in set (0.00 sec)

mysql> drop table GameFamily;
ERROR 9/30 (HY000): Cannot drop table 'GameFamily' referenced by a foreign key constraint 'FanOf_ibfk_1' on table 'FanOf'.
mysql>
```

b. DDL Commands for tables:

```
CREATE TABLE Review (
  PostID INT NOT NULL,
  UserID INT NOT NULL,
  GameID INT NOT NULL,
  Rating FLOAT,
  Comment VARCHAR(8000),
  PRIMARY KEY (PostID),
  FOREIGN KEY (UserID) REFERENCES User(UserID),
  FOREIGN KEY (GameID) REFERENCES BoardgameProduct(GameID)
);

CREATE TABLE User (
  UserID INT NOT NULL,
  Username VARCHAR(255) NOT NULL,
```

```
Age INT,  
Region VARCHAR(255),  
Level INT,  
Trading INT,  
PRIMARY KEY (UserID)  
);
```

```
CREATE TABLE GameFamily (  
FamilyID INT NOT NULL,  
Familyname VARCHAR(255),  
PartyFriendly VARCHAR(255),  
PRIMARY KEY (FamilyID)  
);
```

```
CREATE TABLE BoardgameProduct (  
GameID INT NOT NULL,  
GameName VARCHAR(255),  
Genre VARCHAR(255),  
Duration INT,  
Trading INT,  
Description VARCHAR(8000),  
Difficulty INT,  
YearPublished INT,  
PRIMARY KEY (GameID)  
);
```

```
CREATE TABLE Contributor (  
ContributorID INT NOT NULL,  
Name VARCHAR(255),  
Popularity INT,  
PRIMARY KEY (ContributorID)  
);
```

```
CREATE TABLE Upvoted(  
PostID INT NOT NULL,  
UserID INT NOT NULL,  
PRIMARY KEY (PostID, UserID),  
FOREIGN KEY (PostID) REFERENCES Review(PostID),  
FOREIGN KEY (UserID) REFERENCES User(UserID)  
);
```

```
CREATE TABLE Owned(  
GameID INT NOT NULL,  
UserID INT NOT NULL,  
PRIMARY KEY (GameID, UserID),  
FOREIGN KEY (GameID) REFERENCES BoardgameProduct(GameID),
```

```
FOREIGN KEY (UserID) REFERENCES User(UserID)
);
```

```
CREATE TABLE FanOf(
  FamilyID INT NOT NULL,
  UserID INT NOT NULL,
  PRIMARY KEY (FamilyID, UserID),
  FOREIGN KEY (FamilyID) REFERENCES GameFamily(FamilyID),
  FOREIGN KEY (UserID) REFERENCES User(UserID)
);
```

```
CREATE TABLE Supports(
  ContributorID INT NOT NULL,
  UserID INT NOT NULL,
  PRIMARY KEY (ContributorID, UserID),
  FOREIGN KEY (ContributorID) REFERENCES Contributor(ContributorID),
  FOREIGN KEY (UserID) REFERENCES User(UserID)
);
```

```
CREATE TABLE BelongsTo(
  FamilyID INT NOT NULL,
  GameID INT NOT NULL,
  PRIMARY KEY (FamilyID, GameID),
  FOREIGN KEY (FamilyID) REFERENCES GameFamily(FamilyID),
  FOREIGN KEY (GameID) REFERENCES BoardgameProduct(GameID)
);
```

```
CREATE TABLE Created(
  ContributorID INT NOT NULL,
  GameID INT NOT NULL,
  Role VARCHAR(50),
  PRIMARY KEY (ContributorID, GameID),
  FOREIGN KEY (ContributorID) REFERENCES Contributor(ContributorID),
  FOREIGN KEY (GameID) REFERENCES BoardgameProduct(GameID)
);
```

c. 1000 rows in tables:

```
mysql> show tables;
+-----+
| Tables_in_group32_seq |
+-----+
| BelongsTo              |
| BoardgameProduct       |
| Contributor            |
| Created               |
| FanOf                  |
| GameFamily             |
| Owned                  |
| Review                 |
| Supports               |
| Upvoted                |
| User                   |
+-----+
11 rows in set (0.00 sec)

mysql> SELECT COUNT(*) FROM BelongsTo;
+-----+
| COUNT(*) |
+-----+
|      5137 |
+-----+
1 row in set (0.01 sec)

mysql> SELECT COUNT(*) FROM BoardgameProduct;
+-----+
| COUNT(*) |
+-----+
|      1063 |
+-----+
1 row in set (0.01 sec)

mysql> SELECT COUNT(*) FROM Contributor;
+-----+
| COUNT(*) |
+-----+
|      1500 |
+-----+
1 row in set (0.00 sec)
```

```
mysql> SELECT COUNT(*) FROM FanOf;
+-----+
| COUNT(*) |
+-----+
|      6983 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT COUNT(*) FROM GameFamily ;
+-----+
| COUNT(*) |
+-----+
|      1535 |
+-----+
1 row in set (0.01 sec)

mysql> SELECT COUNT(*) FROM Owned ;
+-----+
| COUNT(*) |
+-----+
|        252 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT COUNT(*) FROM Review;
+-----+
| COUNT(*) |
+-----+
|      8422 |
+-----+
1 row in set (0.01 sec)

mysql> SELECT COUNT(*) FROM Upvoted ;
+-----+
| COUNT(*) |
+-----+
|      8397 |
+-----+
1 row in set (0.00 sec)
```

2. Advanced SQL Queries

- a. Query 1: This query retrieves users' Users' (level> 3) posts with relatively high numbers (>=10) of upvotes to highlight the most popular posts created by the most popular users. Code below:

```
SELECT Rev.PostID, Rev.Rating, User_.Username
FROM Review Rev NATURAL JOIN Upvoted Uv JOIN User User_ ON Uv.UserID=User_.UserID
WHERE User_.Level >3 AND PostID IN
  (SELECT New_table.PostID
   FROM
     (SELECT Rev1.PostID, COUNT(User1.UserID) AS no_of_up
      FROM Review Rev1 JOIN Upvoted Uv1 JOIN User User1 ON Uv1.UserID=User1.UserID
      GROUP BY Rev1.PostID
      HAVING COUNT(User1.UserID) >=10) AS New_table);
```

```
mysql> SELECT Rev.PostID, Rev.Rating, User_.Username, Rev.Comment
-> FROM Review Rev NATURAL JOIN Upvoted Uv JOIN User User_ ON Uv.UserID=User_.UserID
-> WHERE User_.Level > 3 AND PostID IN
-> (SELECT New_table.PostID
-> FROM
-> (SELECT Rev1.PostID, COUNT(User1.UserID) AS no_of_up
-> FROM Review Rev1 JOIN Upvoted Uv1 JOIN User User1 ON Uv1.UserID=User1.UserID
-> GROUP BY Rev1.PostID
-> HAVING COUNT(User1.UserID) >=10) AS New_table);
+-----+-----+-----+-----+
| PostID | Rating | Username | Comment |
+-----+-----+-----+-----+
| 3981 | 7 | fizzle | Dominant Species lets you replay evolution. If that sounds grand, it is. The game is epic both in a the players strive to become the dominant species on the planet. They do this by placing workers, selecting various actions that allow them he main part of this battle for survival happens on the modular map. This part of the game employs a two-dimensional area control mechanism adapted (or both, of course). The map play is highly interactive with everyone trying to broaden their niche, deliberately making conditions has a ton of moving parts which is the main reason a game takes as long as it does. Analyzing the current state of affairs (and draw er very well. There don't seem to be any dealbreakers, there are mitigation options for pretty much anything, and even the asymmetric sp . At the end of the day it is still a multiplayer conflict game with all the issues that entails but it does also include a completely d | 3934 | 6 | godfeather | This game is more than a toy. While this is a typical Fantasy Flight fare with lots of decks and other um and Warrior Knights. Actually, I think this game in many places will replace Warrior Knights as while it has the same feel is nicer l first time, but the mechanics are pretty straightforward. After you have chosen what to play during this round, most actions are pretty end with Fantasy Flight Games with this and Chaos in the old world. The games are shorter in play time, while still feeling epic (maybe s to be pretty balanced, well at least there isn't a critical flaw with it. [b]Last updated:[/b] September 2010
```

Because the comment is too long to show the whole structure, we decided not to include the comment in the select clause:

```
mysql> SELECT Rev.PostID, Rev.Rating, User_.Username
-> FROM Review Rev NATURAL JOIN Upvoted Uv JOIN User User_ ON Uv.UserID=User_.UserID
-> WHERE User_.Level > 3 AND PostID IN
-> (SELECT New_table.PostID
-> FROM
-> (SELECT Rev1.PostID, COUNT(User1.UserID) AS no_of_up
-> FROM Review Rev1 JOIN Upvoted Uv1 JOIN User User1 ON Uv1.UserID=User1.UserID
-> GROUP BY Rev1.PostID
-> HAVING COUNT(User1.UserID) >=10) AS New_table);
```

PostID	Rating	Username
3981	7	fizzle
3934	6	godfeather

- b. Query 2: This advanced SQL shows the difference between PartyFriendly MAX and MIN's BoardGameProduct. Since there are many games in each family, we calculated the average of all games in each selected family for comparison.

```
(SELECT Familyname, Avg(g.Difficulty) as AvgDifficulty, Avg(g.Duration) as AvgDuration, Avg(g.Trading) as
AvgTrading, PartyFriendly
FROM GameFamily f Natural JOIN BelongsTo b JOIN BoardgameProduct g on b.GameID = g.GameID
WHERE PartyFriendly = 10
Group By Familyname
)UNION(
SELECT Familyname, Avg(g.Difficulty) as AvgDifficulty, Avg(g.Duration) as AvgDuration, Avg(g.Trading) as
AvgTrading, PartyFriendly
FROM GameFamily f Natural JOIN BelongsTo b JOIN BoardgameProduct g on b.GameID = g.GameID
WHERE PartyFriendly = 1
Group By Familyname
)
```

```
mysql> (SELECT Familyname, Avg(g.Difficulty) as AvgDifficulty, Avg(g.Duration) as AvgDuration, Avg(g.Trading) as AvgTrading, PartyFriendly
-> FROM GameFamily f Natural JOIN BelongsTo b JOIN BoardgameProduct g on b.GameID = g.GameID
-> WHERE PartyFriendly = 10
-> Group By Familyname
-> LIMIT 10
-> )UNION(
-> SELECT Familyname, Avg(g.Difficulty) as AvgDifficulty, Avg(g.Duration) as AvgDuration, Avg(g.Trading) as AvgTrading, PartyFriendly
-> FROM GameFamily f Natural JOIN BelongsTo b JOIN BoardgameProduct g on b.GameID = g.GameID
-> WHERE PartyFriendly = 1
-> Group By Familyname
-> LIMIT 10
-> )
-> ;
```

Familyname	AvgDifficulty	AvgDuration	AvgTrading	PartyFriendly
ln: Researcher / Scientist	3.0000	68.0000	1091.0000	10
lheap	2.0000	66.0000	613.0000	10
l: Give a Clue / Get a Clue	1.2727	35.1818	400.5455	10
lrs	2.7143	106.4286	591.0000	10
lTwo Player Only Games	2.3118	62.3656	331.7097	10
l& Editions: Two-Player Versions of More-Player Games	2.0000	42.1429	339.5714	10
licola	3.3333	100.0000	1269.0000	10
ls: Map (Continental / National scale)	3.1000	121.5000	710.4667	10
ls: Miniatures	2.9143	129.3571	590.2714	10
ling: Kickstarter	2.5814	84.5736	407.2171	10
lrance	3.0588	120.5882	696.8824	1
la Select	1.9556	43.2222	415.9111	1
lames with Solitaire Rules	2.7536	95.6635	461.6682	1
lorses	3.3333	100.0000	1298.3333	1
lapest Shared World	1.0000	25.0000	355.0000	1
lrds	2.3333	60.0000	1210.3333	1
lCastles of Burgundy	3.0000	75.0000	917.0000	1
l for the Galaxy	2.6667	45.0000	1060.3333	1
lkin	2.0000	105.0000	444.0000	1
lan Games (Lookout)	2.0000	45.0000	756.0000	1

20 rows in set (0.01 sec)

3. Indexing

- a. **Advanced Query #1:** The reason we will choose the following two indices (i.e., Level and Rating) is because for one thing, the user level is in Where clause, and it is not a Primary Key in the User table; for another, though the Rating is not a Primary Key in the Review table, Rating is an important attribute of the reviews and the ratings given by the commenters will be presented as important information with the reviews.

We can see that if we did not add any additional indices, the query execution time was 22.43 seconds. The system generated indices for both foreign keys (UserID and GameID).

Without any additional indices added manually, there is an index scan on User1 using Username_index. The estimated cost is 435.85 and rows are 4311. The actual time range is 0.037-1.063 and rows are 4311. The query used table scan and the estimated cost is 435.85 for scanning on table User_. For table scanning on the temporary table, the actual time range is 0.002-0.633 and rows are 8422.

After adding Index on Level and Rating respectively, the query execution time is 21.99 and 21.15 seconds. The performance seems to improve a bit for the rating index, but the Level index did not. The index scan on User1 using Level_index, the estimated cost is 438.85 and rows are 4311. The actual time range is 0.067-0.979 and rows are 4311. The overall cost and time of Level index is similar to the one without indices.

Specifically, if we add only Level as our additional index. We can see from the results that the system is using this index and the performance improved a little (Index scan on User1 using Level_index: cost=435.85, rows=4311; actual time upper bound is 0.999, rows=4311, loops=1). This is because the Username index generated automatically by the system has a longer upper bound compared to the Level index there. On the other hand, if we add only Rating as our additional index, the system seems not using the Rating index since there are no results show index scan using Rating_index. The same situation applies to when we used both of the indices.

When we added indices on both Level and Rating, the execution took 21.39 seconds, which is a bit better than the query without any indices. However, the effect is not significant in addition to the scans looking similar. The reason for this might be that when we create an index, the database backend will make some statistics by itself to decide whether it should use the index.

Also, we can see that the mechanism doing some joins used a Nested loop inner join, because it makes use of Index. For example, Username_index here is a primary key.

This is the first advanced sql:

```
mysql> SELECT Rev.PostID, Rev.Rating, User_.Username
-> FROM Review Rev NATURAL JOIN Upvoted Uv JOIN User User_ ON Uv.UserID=User_.UserID
-> WHERE User_.Level >3 AND PostID IN
-> (SELECT New_table.PostID
-> FROM
-> (SELECT Rev1.PostID, COUNT(User1.UserID) AS no_of_up
-> FROM Review Rev1 JOIN Upvoted Uv1 JOIN User User1 ON Uv1.UserID=User1.UserID
-> GROUP BY Rev1.PostID
-> HAVING COUNT(User1.UserID) >=10) AS New_table);
+-----+-----+-----+
| PostID | Rating | Username |
+-----+-----+-----+
| 3981 | 7 | fizzle |
| 3934 | 6 | godfeather |
+-----+-----+-----+
2 rows in set (14.79 sec)
```

This is the indices in it:

```
mysql> show index from Review;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
Review	0	PRIMARY	1	PostID	A	8537	NULL	NULL		BTREE			YES	NULL
Review	1	UserID	1	UserID	A	4310	NULL	NULL	YES	BTREE			YES	NULL
Review	1	GameID	1	GameID	A	1062	NULL	NULL		BTREE			YES	NULL

3 rows in set (0.01 sec)

Without adding any other indices, here is the processing situation:

[illegible]

```

-> Nested loop semijoin (cost=300746.19 rows=0) (actual time=22412.985..22428.082 rows=2 loops=1)
-> Nested loop inner join (cost=2087.78 rows=140) (actual time=7.397..22.487 rows=2 loops=1)
-> Nested loop inner join (cost=1075.43 rows=2799) (actual time=0.068..10.749 rows=5420 loops=1)
-> Filter: (User_`Level` > 3) (cost=435.85 rows=1437) (actual time=0.052..2.882 rows=2845 loops=1)
-> Table scan on User (cost=435.85 rows=4311) (actual time=0.049..2.100 rows=4311 loops=1)

```

```
loops=4311)
```

1

+

1 row in set (22.43 sec)

Then add an index on Level:

```
mysql> CREATE INDEX Level_index on User(Level);
Query OK, 0 rows affected (0.15 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
mysql> show index from User;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
User	0	PRIMARY	1	UserID	A	4311	NULL	NULL		BTREE			YES	NULL
User	1	Username_index	1	Username	A	4308	NULL	NULL		BTREE			YES	NULL
User	1	Level_index	1	Level	A	10	NULL	NULL	YES	BTREE			YES	NULL

3 rows in set (0.01 sec)

[illegible]

```
-> Nested loop semijoin (cost=595055.48 rows=0) (actual time=21975.831..21987.758 rows=2 loops=1)
-> Nested loop inner join (cost=3706.71 rows=277) (actual time=5.794..17.717 rows=2 loops=1)
-> Nested loop inner join (cost=1702.24 rows=5542) (actual time=0.053..8.775 rows=5420 loops=1)
-> Filter: (User_>'level' > 3) (cost=435.85 rows=2845) (actual time=0.042..2.186 rows=2845 loops=1)
-> Table scan on User_ (cost=435.85 rows=4311) (actual time=0.039..1.545 rows=4311 loops=1)
-> Index lookup on Uv using UserID (UserID=User.UserID) (cost=0.25 rows=2) (actual time=0.001..0.002 rows=2 loops=2845)
-> Filter: (Rev.UserID = User.UserID) (cost=0.26 rows=0) (actual time=0.001..0.001 rows=0 loops=5420)
-> Single-row index lookup on Rev using PRIMARY (PostID=Uv.PostID) (cost=0.26 rows=1) (actual time=0.001..0.001 rows=1 loops=5420)
-> Index lookup on Rev table using cache key(s) (PostID=Uv.PostID) (actual time=0.006..0.006 rows=1 loops=2)
-> Materialize (cost=0.00..0.00 rows=0) (actual time=21970.039..21970.039 rows=8422 loops=1)
-> Filter: (count(User.UserID) >= 10) (actual time=21961.628..21963.830 rows=8422 loops=1)
-> Table scan on <temporary> (actual time=0.002..0.506 rows=8422 loops=1)
-> Aggregate using temporary table (actual time=21961.622..21962.837 rows=8422 loops=1)
-> Inner hash join (no condition) (cost=711199.76 rows=7158518) (actual time=11.569..6038.719 rows=70719536 loops=1)
-> Index scan on Rev1 using GameID (cost=0.14 rows=8537) (actual time=0.030..28.264 rows=8422 loops=1)
-> Hash
-> Nested loop inner join (cost=2354.80 rows=8397) (actual time=0.043..10.623 rows=8397 loops=1)
-> Index scan on User1 using level index (cost=435.85 rows=4311) (actual time=0.036..0.990 rows=4311 loops=1)
-> Index lookup on Uv1 using UserID (UserID=User1.UserID) (cost=0.25 rows=2) (actual time=0.001..0.002 rows=2 loops=4311)
```

+

1 row in set (21.99 sec)

```
| -> Nested loop semijoin (cost=595055.48 rows=0) (actual time=21975.831..21987.758 rows=2 loops=1)
| -> Nested loop inner join (cost=3706.71 rows=277) (actual time=5.794..17.717 rows=2 loops=1)
```

- > Nested loop inner join (cost=1702.24 rows=5542) (actual time=0.053..8.775 rows=5420 loops=1)
 - > Filter: (User_.'Level' > 3) (cost=435.85 rows=2845) (actual time=0.042..2.186 rows=2845 loops=1)
 - > Table scan on User_ (cost=435.85 rows=4311) (actual time=0.039..1.545 rows=4311 loops=1)
 - > Index lookup on Uv using UserID (UserID=User_.'UserID') (cost=0.25 rows=2) (actual time=0.001..0.002 rows=2 loops=2845)
 - > Filter: (Rev.UserID = User_.'UserID') (cost=0.26 rows=0) (actual time=0.001..0.001 rows=0 loops=5420)
 - > Single-row index lookup on Rev using PRIMARY (PostID=Uv.PostID) (cost=0.26 rows=1) (actual time=0.001..0.001 rows=1 loops=5420)
- > Index lookup on New_table using <auto_key0> (PostID=Uv.PostID) (actual time=0.006..0.006 rows=1 loops=2)
 - > Materialize (cost=0.00..0.00 rows=0) (actual time=21970.039..21970.039 rows=8422 loops=1)
 - > Filter: (count(User1.UserID) >= 10) (actual time=21961.628..21963.830 rows=8422 loops=1)
 - > Table scan on <temporary> (actual time=0.002..0.506 rows=8422 loops=1)
 - > Aggregate using temporary table (actual time=21961.622..21962.837 rows=8422 loops=1)
 - > Inner hash join (no condition) (cost=7171199.76 rows=71685188) (actual time=11.569..6038.719 rows=70719536 loops=1)
 - > Index scan on Rev1 using GameID (cost=0.14 rows=8537) (actual time=0.030..28.264 rows=8422 loops=1)
 - > Hash
 - > Nested loop inner join (cost=2354.80 rows=8397) (actual time=0.043..10.623 rows=8397 loops=1)
 - > Index scan on User1 using **Level_index** (cost=435.85 rows=4311) (actual time=0.036..0.999 rows=4311 loops=1)
 - > Index lookup on Uv1 using UserID (UserID=User1.UserID) (cost=0.25 rows=2) (actual time=0.001..0.002 rows=2

loops=4311)

|

+

-----+

1 row in set (21.99 sec)

If we add an index on only Rating:

```
mysql> CREATE INDEX Rating_index ON Review(Rating);
Query OK, 0 rows affected (0.09 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> show index from Review;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible | Expression |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Review | 0 | PRIMARY | 1 | PostID | A | 8537 | NULL | NULL | NULL | BTREE | | | YES | NULL |
| Review | 1 | UserID | 1 | UserID | A | 4310 | NULL | NULL | YES | BTREE | | | YES | NULL |
| Review | 1 | GameID | 1 | GameID | A | 1062 | NULL | NULL | YES | BTREE | | | YES | NULL |
| Review | 1 | Rating_index | 1 | Rating | A | 152 | NULL | NULL | YES | BTREE | | | YES | NULL |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> show index from User;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible | Expression |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| User | 0 | PRIMARY | 1 | UserID | A | 4311 | NULL | NULL | NULL | BTREE | | | YES | NULL |
| User | 1 | Username_index | 1 | Username | A | 4308 | NULL | NULL | NULL | BTREE | | | YES | NULL |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> show index from Review;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible | Expression |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Review | 0 | PRIMARY | 1 | PostID | A | 8537 | NULL | NULL | NULL | BTREE | | | YES | NULL |
| Review | 1 | UserID | 1 | UserID | A | 4310 | NULL | NULL | YES | BTREE | | | YES | NULL |
| Review | 1 | GameID | 1 | GameID | A | 1062 | NULL | NULL | YES | BTREE | | | YES | NULL |
| Review | 1 | Rating_index | 1 | Rating | A | 152 | NULL | NULL | YES | BTREE | | | YES | NULL |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> EXPLAIN ANALYZE SELECT Rev.PostID, Rev.Rating, User_.'Username' FROM Review Rev NATURAL JOIN Upvoted Uv JOIN User User_ ON Uv.UserID=User_.'UserID' WHERE User_.'Level' > 3 AND PostID IN (SELECT New_table.PostID FROM (SELECT Rev1.PostID, COUNT(User1.UserID) AS no_of_up FROM Review Rev1 JOIN Upvoted Uv1 JOIN User User1 ON Uv1.UserID=User1.UserID GROUP BY Rev1.PostID HAVING COUNT(User1.UserID) >=10) AS New_table);
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible | Expression |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Review | 0 | PRIMARY | 1 | PostID | A | 8537 | NULL | NULL | NULL | BTREE | | | YES | NULL |
| Review | 1 | UserID | 1 | UserID | A | 4310 | NULL | NULL | YES | BTREE | | | YES | NULL |
| Review | 1 | GameID | 1 | GameID | A | 1062 | NULL | NULL | YES | BTREE | | | YES | NULL |
| Review | 1 | Rating_index | 1 | Rating | A | 152 | NULL | NULL | YES | BTREE | | | YES | NULL |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> EXPLAIN
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible | Expression |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Review | 0 | PRIMARY | 1 | PostID | A | 8537 | NULL | NULL | NULL | BTREE | | | YES | NULL |
| Review | 1 | UserID | 1 | UserID | A | 4310 | NULL | NULL | YES | BTREE | | | YES | NULL |
| Review | 1 | GameID | 1 | GameID | A | 1062 | NULL | NULL | YES | BTREE | | | YES | NULL |
| Review | 1 | Rating_index | 1 | Rating | A | 152 | NULL | NULL | YES | BTREE | | | YES | NULL |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

--> Nested loop semi join (cost=300746.15 rows=0) (actual time=21124.054..21136.310 rows=2 loops=1)
--> Nested loop inner join (cost=2087.78 rows=140) (actual time=5.698..17.909 rows=2 loops=1)
--> Nested loop inner join (cost=1075.43 rows=2799) (actual time=0.074..8.974 rows=5420 loops=1)
--> Filter: (User_.'Level' > 3) (cost=435.85 rows=1437) (actual time=0.059..2.219 rows=2845 loops=1)
--> Table scan on User_ (cost=435.85 rows=4311) (actual time=0.056..1.582 rows=4311 loops=1)
--> Index lookup on Uv using UserID (UserID=User_.'UserID') (cost=0.25 rows=2) (actual time=0.001..0.002 rows=2 loops=2845)
--> Filter: (Rev.UserID = User_.'UserID') (cost=0.26 rows=0) (actual time=0.001..0.001 rows=0 loops=5420)
--> Single-row index lookup on Rev using PRIMARY (PostID=Uv.PostID) (cost=0.26 rows=1) (actual time=0.001..0.001 rows=1 loops=5420)
--> Index lookup on New_table using <auto_key0> (PostID=Uv.PostID) (actual time=0.005..0.005 rows=1 loops=2)
```


[illegible]

+-----+
+-----+
1 row in set (21.39 sec)

- b. Advanced Query #2: The reason why we choose these three indices (BoardgameProduct.Difficulty, BoardgameProduct.Duration and GameFamily.PartyFriendly) is because both BoardgameProduct.Difficulty and BoardgameProduct.Duration are important information with the game and in this SQL, which help us compare the differences between boardgames in different families. And GameFamily.PartyFriendly, is an important attribute reflecting the popularity of Boardgames belonging to a family.

As we can see from the initial performance, the time taken to implement this SQL is short and there is little room for improvement. There is little difference between PartyFriendly=10 and PartyFriendly=1, except that PartyFriendly=10 takes a little more time than PartyFriendly=1.

After we added Difficulty as the index, we can see that the SQL execution doesn't seem to have changed, and Duration as the index doesn't seem to work in this SQL.

After we added Duration as the index, we can see that the SQL seems to take longer to execute, and Table scan time is shortened when Partyfriendly=10, but increased when Partyfriendly=1. And Duration as the index doesn't seem to work in this SQL.

After adding PartyFriendly as index, we can see that the runtime of the SQL does not change significantly, and table scan time compared with the original, when Partyfriendly=10, time increased, but when Partyfriendly=1, time decreased.

We can see that before adding index on Difficulty, the execution time is 0.01seconds, and add index idx_difficulty, the performance has no change, the execution time is still 0.01 seconds. For the attribute of Difficulty, it is not in the WHERE clause and no tables joining on this attribute. We can see that before and after index, the database did not use this index, and use table scan and look up using the primary key. Specifically, before using index on Difficulty, the cost of table scan on f estimated 155.25, and rows are 1535. The actual time range is 0.051..0.727 and rows are 153. After using the index, the cost and rows are not changed. Also, the actual time is no different. So far, the index we've added has not contributed to performance for the time being. Therefore, we might consider adding some other indices to improve the performance in the future.

```
(SELECT Familyname, Avg(g.Difficulty) as AvgDifficulty, Avg(g.Duration) as AvgDuration, Avg(g.Trading) as AvgTrading, PartyFriendly
FROM GameFamily f Natural JOIN BelongsTo b JOIN BoardgameProduct g on b.GameID = g.GameID
WHERE PartyFriendly = 10
Group By Familyname
)UNION(
SELECT Familyname, Avg(g.Difficulty) as AvgDifficulty, Avg(g.Duration) as AvgDuration, Avg(g.Trading) as AvgTrading, PartyFriendly
```

```

FROM GameFamily f Natural JOIN BelongsTo b JOIN BoardgameProduct g on b.GameID = g.GameID
WHERE PartyFriendly = 1
Group By Familyname
)

```

INITIAL INDEX and EXPLAIN ANALYZE:

```

mysql> SHOW INDEX FROM BoardgameProduct;
+-----+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible | Expression |
+-----+
| BoardgameProduct | 0 | PRIMARY | 1 | GameID | A | 1008 | NULL | NULL | NULL | BTREE | | | YES | NULL |
+-----+
1 row in set (0.00 sec)

mysql> SHOW INDEX FROM GameFamily;
+-----+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible | Expression |
+-----+
| GameFamily | 0 | PRIMARY | 1 | FamilyID | A | 1535 | NULL | NULL | NULL | BTREE | | | YES | NULL |
+-----+
1 row in set (0.01 sec)

mysql> EXPLAIN ANALYZE (SELECT Familyname, Avg(g.Difficulty) as AvgDifficulty, Avg(g.Duration) as AvgDuration, Avg(g.Trading) as AvgTrading, PartyFriendly
ly
-> FROM GameFamily f Natural JOIN BelongsTo b JOIN BoardgameProduct g on b.GameID = g.GameID
-> WHERE PartyFriendly = 10
-> Group By Familyname
-> )UNION(
-> SELECT Familyname, Avg(g.Difficulty) as AvgDifficulty, Avg(g.Duration) as AvgDuration, Avg(g.Trading) as AvgTrading, PartyFriendly
-> FROM GameFamily f Natural JOIN BelongsTo b JOIN BoardgameProduct g on b.GameID = g.GameID
-> WHERE PartyFriendly = 1
-> Group By Familyname
-> )
-> ;
+-----+
| EXPLAIN
+-----+

```


ADD Difficulty index on BoardgameProduct:

```
mysql> CREATE INDEX idx_difficulty ON BoardgameProduct(Difficulty);
Query OK, 0 rows affected (0.07 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

```
mysql> SHOW INDEX FROM BoardgameProduct;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'INDEX FROM BoardgamePro
duct' at line 1
mysql> SHOW INDEX FROM BoardgameProduct;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visib |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| BoardgameProduct | 0 | PRIMARY | 1 | GameID | A | 1008 | NULL | NULL | NULL | BTREE | | | YES | |
| NULL | | | | | | | | | | | | | | |
| BoardgameProduct | 1 | idx_difficulty | 1 | Difficulty | A | 6 | NULL | NULL | YES | BTREE | | | YES |
| NULL | | | | | | | | | | | | | | |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
2 rows in set (0.01 sec)

mysql> EXPLAIN ANALYZE (SELECT Familyname, Avg(g.Difficulty) as AvgDifficulty, Avg(g.Duration) as AvgDuration, Avg(g.Trading) as AvgTrading, PartyFriendly
-> FROM GameFamily f Natural JOIN BelongsTo b JOIN BoardgameProduct g on b.GameID = g.GameID
-> WHERE PartyFriendly = 10
-> Group By Familyname
-> ) UNION(
-> SELECT Familyname, Avg(g.Difficulty) as AvgDifficulty, Avg(g.Duration) as AvgDuration, Avg(g.Trading) as AvgTrading, PartyFriendly
-> FROM GameFamily f Natural JOIN BelongsTo b JOIN BoardgameProduct g on b.GameID = g.GameID
-> WHERE PartyFriendly = 1
-> Group By Familyname
-> )
-> ;
```

```
| EXPLAIN
```

```
+-----+
| -> Table scan on <union temporary> (cost=2.50..2.50 rows=0) (actual time=0.001..0.030 rows=295 loops=1)
-> Union materialize with deduplication (cost=2.50..2.50 rows=0) (actual time=9.092..9.144 rows=295 loops=1)
-> Table scan on <temporary> (actual time=0.002..0.024 rows=142 loops=1)
-> Aggregate using temporary table (actual time=4.688..4.723 rows=142 loops=1)
-> Nested loop inner join (cost=425.12 rows=514) (actual time=0.726..3.668 rows=635 loops=1)
-> Nested loop inner join (cost=245.20 rows=514) (actual time=0.690..2.062 rows=635 loops=1)
-> Filter: (f.PartyFriendly = 10) (cost=155.25 rows=154) (actual time=0.670..1.364 rows=142 loops=1)
```


DROP Difficulty index, ADD Duration index on BoardgameProduct:

```
Query OK, 0 rows affected (0.04 sec)
```

Query OK, 0 rows affected (0.04 sec)

```
Records: 0  Duplicates: 0  Warnings: 0
```

```
Query OK, 0 rows affected (0.05 sec)
```

Query OK, 0 rows affected (0.05 sec)

```
mysql> SHOW INDEX FROM BoardgameProduct;
```

```
-> FROM GameFamily f Natural JOIN BelongsTo b JOIN BoardgameProduct g on b.GameID = g.GameID
```

```
-> Group By Familyname
```

```
-> SELECT Familyname, Avg(g.Difficulty) as AvgDifficulty, Avg(g.Duration) as AvgDuration, Avg
> FROM GameFamily f Natural JOIN BelongsTo b JOIN BoardgameProduct g on b.GameID = g.GameID
```

```
-> WHERE PartyFriendly
-> Group By Familyname
```

$$\rightarrow ;$$

| EXPLAI

EXHIBIT

11/11/2016

11/11/2016

+

DROP Duration index on BoardgameProduct, ADD Familyname index on GameFamily:

```
mysql> DROP INDEX idx_duration ON BoardgameProduct;
Query OK, 0 rows affected (2.54 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
mysql> CREATE INDEX idx_partyfriendly ON GameFamily(PartyFriendly);
Query OK, 0 rows affected (0.09 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
mysql> SHOW INDEX FROM GameFamily;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| GameFamily | 0 | PRIMARY | 1 | FamilyID | A | 1535 | NULL | NULL | NULL | BTREE | | | YES |
| GameFamily | 1 | idx_partyfriendly | 1 | PartyFriendly | A | 10 | NULL | NULL | YES | BTREE | | | YES |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
2 rows in set (0.01 sec)
```

```
mysql> EXPLAIN ANALYZE (SELECT Familyname, Avg(g.Difficulty) as AvgDifficulty, Avg(g.Duration) as AvgDuration, Avg(g.Trading) as AvgTrading, PartyFriendly
-> FROM GameFamily f Natural JOIN BelongsTo b JOIN BoardgameProduct g on b.GameID = g.GameID
-> WHERE PartyFriendly = 10
-> Group By Familyname
-> )UNION(
-> SELECT Familyname, Avg(g.Difficulty) as AvgDifficulty, Avg(g.Duration) as AvgDuration, Avg(g.Trading) as AvgTrading, PartyFriendly
-> FROM GameFamily f Natural JOIN BelongsTo b JOIN BoardgameProduct g on b.GameID = g.GameID
-> WHERE PartyFriendly = 1
-> Group By Familyname
-> )
-> ;
```

```
| EXPLAIN
```

