Table of Contents

Introduction

????

Relations Breakdown

To prevent any data anomalies, all relations have been normalised to Boyce-Codd Normal Form (BCNF):

- There are no partial functional dependencies (FDs).
- There are no transitive FDs.
- Every determinant is a unique candidate key.
- Foreign keys are used for all linked relations to ensure referential integrity

`UserPublic`	: Holds information about users which can be viewed by other users.
`UserPrivate`	: Holds all information about users which cannot be viewed by other users.
`Email`	: Holds all user email addresses.
`Friends`	: Holds a list of all user friendships.
`FriendRequest`	: Holds any friend request information.
`Game`	: Holds information about games.
`Genre`	: Holds all the game genre information.
`GameImage`	: Holds information about images to be linked with games.
`UserToGame`	: This relation is used to link Users to Games holding all information about each user's separate instance of a game.
`GameToGenre`	: This relations is used to link Games to Genres.
`Leaderboard`	: Holds all information on user Leaderboards.
`Plays`	: Records the number of times a user plays a game.
`Scores`	: Records all the scores made on any game.
`Achievement`	: Holds all achievement details.
`AchievementToUserToGame`	: Linking relation to achievements, users and games.
`Matches`	: Holds match details.
`MatchToUserToGame`	: Linking relations for matches, users and games.
`MatchRequest`	: Holds any match request information.
`RudeWord`	: Stores a list of offensive words (to prevent obscene names).

Friendship Features

All user friendships are stored in the **`Friends`** relation:

`AccHolder`	**`Friend`**
`UserName` *of the Account Holder*	`UserName` *of their Friend*

The primary key for this relation is multi-valued (`AccHolder`, `Friend`) to ensure that each account holder pairing is not duplicated. Note that all friendships are bidirectional. This means that a matching reverse friendship must exist for all friendship pairs. For example:

`AccHolder`	**`Friend`**
`AlexParrott`	`ScarlettJo`
`ScarlettJo`	`AlexParrott`

This design decision was made primarily to make queries simpler:

- All friendships are stored in a single relation.
- The same simple query can be used to get a complete friend list for all users
  `SELECT * Friends WHERE AccHolder = 'ScarlettJo';`

A matching friendship is automatically generated or deleted by the `ProcessRequest()` procedure (see below). `INSERT` and `UPDATE` triggers in MySQL cannot insert new data into a table that is already being amended. Therefore, if a trigger was used to create matching reverse friendships, a second `Friends` relation would be needed. Also, triggers have the additional drawbacks of being global and vulnerable to hidden consequences. This feature is an implementation of **Question 10**.

Creating and Deleting Friendships

All changes to friendships (creation or deletion) are done via the **`FriendRequest`** relation:

`RequestID`	**`Requester`**	**`Requestee`**	**`Email`**	**`Response`**

Unique Friend Request ID	`UserName` *of the requester*	`UserName` *of the requested user*	`Email` *address of the requested user*	*Flag for status of the request:*
				`- Pending`
				`- Accepted`
				`- Declined`
				`- Completed`

When a FriendRequest is created it is assigned a unique `RequestID` which can be used to create a new friendship (when `Response='Accepted'`) or delete a friendship (`Response='Declined'`) by calling `ProcessRequest()`.

How to create new friendships

<u>Step 1: Create a request</u>

- This is done by calling **`CreateRequest()`**. Potential friends can be looked up using either their `UserName` or their `Email` address. Ensure the delete flag parameter is set to `False`.

<u>Step 2: Respond to the request</u>

- If a user wishes to accept a friend request then the `Response` attribute in the relevant `FriendRequest` must be updated to `'Accepted'`. If they do not want to accept the request, the `Response` attribute can be updated to `'Declined'`. In this case the request will be marked as complete and deleted when **`ProcessRequest()`** is next called.

Step 3: Process the request

- This is done by calling **`ProcessRequest()`**. If the response is set to `'Accepted'` then the friendship will be inserted into the `Friends` relation.

How to delete friendships

Step 1: Create a request

- This is done by calling **`CreateRequest()`**. Friends can be looked up using either their `UserName` or their `Email` address. Ensure the delete flag parameter is set to `True`.

Step 2: Process the request

- This is done by calling **`ProcessRequest()`**. This will delete the friendship from the `Friends` relation. The request will then be automatically deleted.

`CreateRequest()`

Description:
This procedure creates a new request in the `FriendRequest` relation using a provided `UserName` to look up requested user to create or delete a friendship.

Parameters:

1. The `UserName` of the of the user making the request.
2. The `UserName` or `Email` of the user to request/delete friendship.
3. Delete flag: TRUE = request new friend; FALSE = request friendship deletion
4. Email flag: TRUE = lookup user with `Email` attribute; FALSE = lookup user with `UserName` attribute

Example Usage:
Request new friendship with `UserName` lookup:
```
CALL CreateRequest('AlexParrott','WillWoodhead',FALSE,FALSE);
```

Request friendship deletion with `UserName` lookup:
```
CALL CreateRequest('AlexParrott','WillWoodhead',TRUE,FALSE);
```

Request new friendship with `Email` lookup:
```
CALL CreateRequest('AlexParrott','Will@Woodhead.com',FALSE,TRUE);
```

Request friendship deletion with `Email` lookup:
```
CALL CreateRequest('AlexParrott','Will@Woodhead.com',TRUE,TRUE);
```

`ProcessRequest()`

Description:
This procedure processes a specified request in the `FriendRequest` relation according to the request response status:

• `Response = 'Accepted'`	: Creates a new friendship pair and matching reverse friendship in the `Friends` relation.
• `Response = 'Declined'`	: Deletes the friendship pair and the matching reverse friendship in the `Friends` relation.
• `Response = 'Pending'`	: No action.
• `Response = 'Completed'`	: Deletes entry from `FriendRequest` relation.

Parameter:

- The `RequestID` (INT) of the `FriendRequest` to action.

Example Usage:
```
CALL ProcessRequest(14);
```

`ShowFriends()`

The `ShowFriends()` procedure lists all of a specified user's friends as requested in **Question 12**. All online friends are shown in one table and then all offline friends are shown including their last logon time and the name of the last game they were playing. The user to lookup is specified by passing the `UserName` as a parameter when calling this procedure.

Example query:
```
mysql> CALL ShowFriends('AlexParrott');
+--------------+---------------+
| UserName     | AccountStatus |
+--------------+---------------+
| DavidCameron | Online        |
| ScarlettJo   | Online        |
| WillWoodhead | Online        |
+--------------+---------------+

+---------------+---------------+---------------------+-------------+
| UserName      | AccountStatus | LastLogin           | LastPlayed  |
+---------------+---------------+---------------------+-------------+
| JamesHamblion | Offline       | 2014-05-05 15:02:37 | Angry Birds |
+---------------+---------------+---------------------+-------------+
```

`SuggestFriends()`

The `SuggestFriends()` procedure creates a list of suggested friends for a specified user (**Question 18**). This works by compiling a list of any users who are not already friends with the user and have 2 or more friends *or* owned games in common with the user. The number of friends and games in common are also displayed in the final list. The user to lookup is specified by passing the `UserName` as a parameter when calling this procedure.

Example query:
```
mysql> CALL SuggestFriends('DavidCameron');
+---------------+-----------------+---------------+
| UserName      | FriendsInCommon | GamesInCommon |
+---------------+-----------------+---------------+
| BobHope       |               0 |             2 |
| JamesHamblion |               1 |             3 |
| ScarlettJo    |               2 |             3 |
+---------------+-----------------+---------------+
```

Games Features

`ListGameOwners()`

The `ListGameOwners()` procedure creates a list of all the users who own a specified game. The game to lookup is specified by passing the `GameID` as a parameter when calling this procedure. This feature relates to **Question 1**.

Example query:
```
mysql> CALL ListGameOwners(1);
+-------------------+
| Owners            |
+-------------------+
| AlexParrott       |
| JamesHamblion     |
| WillWoodhead      |
| ScarlettJo        |
| AliceInWonderland |
| BobHope           |
| BarackObama       |
| DavidCameron      |
| GeorgeClooney     |
| BradPitt          |
+-------------------+
```

`UpdateAverage()`

The `UpdateAverage()` procedure updates the average user rating of a specified game (stored in the `UserToGame` relation as the `AverageRating` attribute). Average ratings only apply when a game has 10 or more user ratings. If a game has less than 10 ratings, the average is set to `NULL`. This procedure implements this feature by manipulating the `AverageRating` and `NoOfRatings` attributes in the Game relation and relates to **Question 2** and **Question 3**.

Note, this procedure is called automatically by the following triggers:

`AfterInsertUserToGame`	: Triggers after any insert to the `UserToGame` relation.
`AfterUpdateUserToGame`	: Triggers after any update to the `UserToGame` relation.
`AfterDeleteUserToGame`	: Triggers after any delete from the `UserToGame` relation.

Therefore, **anytime a user rates a game the average rating is automatically updated in the database**.

Example query:
```
mysql> SELECT Name, AverageRating From Game;
+--------------------+---------------+
| Name               | AverageRating |
+--------------------+---------------+
| GTA V              |          5.68 |
| The Last of Us     |          NULL |
| FIFA 14            |          NULL |
| Angry Birds        |          NULL |
| mission Impossible |          NULL |
| James Bond         |          NULL |
```

This table shows that *only* GTA V has been rated by more than 10 users.

`CatchCheaters()`

The `CatchCheaters()` function is setup to prevent cheaters who fix their scores (as requested by **Question 6**). A maximum score (`MaxScore`) and minimum score (`MinScore`) attributes are stored in the `Game` relation. If these are not set to `NULL` then this function checks a provided score against the played game's max and min scores. If the score provided is legal then it is returned to the function caller unchanged. However, if the provided score is an illegal value then the minimum score for than game is returned.

Note, this function is called automatically by the following triggers:

`BeforeInsertUserToGame`	: Triggers before any insert to the `UserToGame` relation.

`BeforeUpdateUserToGame` : Triggers before any update to the `UserToGame` relation.

Both of these triggers set the the `LastScore` in the `UserToGame` relation. Therefore, **anytime a user's last score is added or updated, it is automatically checked against the legal values.**

Example query:

Here is the score range for the game 'Angry Birds' (GameID 4):

```
mysql> select Name,MaxScore,Minscore from Game WHERE GameID=4;
+-------------+----------+----------+
| Name        | MaxScore | Minscore |
+-------------+----------+----------+
| Angry Birds |     1000 |        1 |
+-------------+----------+----------+
```

Here are the last people to play this game and their scores:

```
mysql> select ID,UserName,LastScore from UserToGame where GameID=4;
+----+---------------+-----------+
| ID | UserName      | LastScore |
+----+---------------+-----------+
|  1 | AlexParrott   |        28 |
|  5 | JamesHamblion |        53 |
| 33 | DavidCameron  |        41 |
+----+---------------+-----------+
```

Now let's set AlexParrott's last score to an illegal value (over 1000)...

```
mysql> UPDATE UserToGame SET LastScore=1007 WHERE ID=1;
```

...then have a look at these scores again:

```
mysql> select ID,UserName,LastScore from UserToGame where GameID=4;
+----+---------------+-----------+
| ID | UserName      | LastScore |
+----+---------------+-----------+
|  1 | AlexParrott   |         1 |
|  5 | JamesHamblion |        53 |
| 33 | DavidCameron  |        41 |
+----+---------------+-----------+
```

As is highlighted, this score has been automatically set to the game's minimum score (1).

TopTens()

This procedure gets the top ten rated games in each genre (**Question 5**). This will list the top ten rated games in descending order for each genre. If there are less than ten games, only the existing games will be listed.

Example query:

```
mysql> CALL TopTens();
+-------------+--------------------+---------------+
| genre       | name               | AverageRating |
+-------------+--------------------+---------------+
| Adventure   | GTA V              |          9.87 |
| Adventure   | The Last of Us     |          5.43 |
| Adventure   | mission Impossible |          3.46 |
| Adventure   | 2048               |          3.44 |
| Adventure   | Black ops          |          2.45 |
| Adventure   | bin throw          |          2.34 |
| Adventure   | flick men          |          2.12 |
| Horror      | Crash Bandicoot    |          6.51 |
| Horror      | carrot peel        |          6.43 |
| Horror      | skyroads           |          6.12 |
| Horror      | The Last of Us     |          5.43 |
| Mutliplayer | GTA V              |          9.87 |
| Mutliplayer | FIFA 14            |          8.65 |
| Mutliplayer | Angry Birds        |          6.74 |
| Mutliplayer | James Bond         |          5.67 |
| Mutliplayer | COD4               |          4.34 |
| Mutliplayer | mash up            |          3.23 |
| Sport       | The Last of Us     |          5.43 |
| Sport       | FIFA 14            |          4.35 |
| Sport       | Bike Runner        |          3.65 |
+-------------+--------------------+---------------+
```

hotlist()

This procedures gets a Hotlist of the most played games – relates to **Question 9**. If users are interested in knowing which games have been played the most, they can simply get the hotlist. This is a dynamic realtime view of which games are being played the most.

Example query:

```
mysql> CALL hotlist();
+---------+--------------------+-------------+
| Ranking | Name               | NOPLastWeek |
+---------+--------------------+-------------+
|       1 | GTA V              |          23 |
|       2 | mission Impossible |          12 |
|       3 | Angry Birds        |          10 |
|       4 | The Last of Us     |           8 |
|       8 | Crash Bandicoot    |           7 |
|       7 | bin throw          |           7 |
|       6 | COD4               |           7 |
|       5 | FIFA 14            |           7 |
|       9 | James Bond         |           6 |
|      10 | mash up            |           5 |
+---------+--------------------+-------------+
```

Leaderboard Features

Daily and weekly leaderboards

When a new game is inserted into the game table, daily and weekly leaderboards are generated automatically alongside a normal default leaderboard for the game. These allow users to see the scores have been each week and day for every game (**Question 7**).

Note, this feature occurs automatically. It is called by the following trigger:
`Game_After_Insert` : Triggers after any insert to the `Game` relation.

The table below shows all of the leaderboards on the Game Centre. Each game has a default leaderboard, and then any number of other leaderboards that are not default. This leaderboard table is essentially a set of criteria to inform how to query the `Scores` table to get the desired leaderboard. This means that the leaderboards are always up to date.

```
+---------------+--------+-----------+------------+-----------+
| LeaderboardID | GameID | SortOrder | TimePeriod | IsDefault |
+---------------+--------+-----------+------------+-----------+
|             1 |      1 | desc      | forever    |         1 |
|             2 |      1 | desc      | 1_week     |         0 |
|             3 |      1 | desc      | 1_day      |         0 |
|             4 |      2 | desc      | forever    |         1 |
|             5 |      2 | desc      | 1_week     |         0 |
|             6 |      2 | desc      | 1_day      |         0 |
|             7 |      3 | desc      | forever    |         1 |
|             8 |      3 | desc      | 1_week     |         0 |
|             9 |      3 | desc      | 1_day      |         0 |
|            10 |      4 | desc      | forever    |         1 |
|            11 |      4 | desc      | 1_week     |         0 |
|            12 |      4 | desc      | 1_day      |         0 |
|            13 |      5 | desc      | forever    |         1 |
|            14 |      5 | desc      | 1_week     |         0 |
|            15 |      5 | desc      | 1_day      |         0 |
|            16 |      6 | desc      | forever    |         1 |
|            17 |      6 | desc      | 1_week     |         0 |
|            18 |      6 | desc      | 1_day      |         0 |
|            19 |      7 | desc      | forever    |         1 |
|            20 |      7 | desc      | 1_week     |         0 |
|            21 |      7 | desc      | 1_day      |         0 |
|            22 |      8 | desc      | forever    |         1 |
|            23 |      8 | desc      | 1_week     |         0 |
|            24 |      8 | desc      | 1_day      |         0 |
|            25 |      9 | desc      | forever    |         1 |
|            26 |      9 | desc      | 1_week     |         0 |
|            27 |      9 | desc      | 1_day      |         0 |
|            28 |     10 | desc      | forever    |         1 |
|            29 |     10 | desc      | 1_week     |         0 |
|            30 |     10 | desc      | 1_day      |         0 |
|            31 |     11 | desc      | forever    |         1 |
|            32 |     11 | desc      | 1_week     |         0 |
|            33 |     11 | desc      | 1_day      |         0 |
|            34 |     12 | desc      | forever    |         1 |
|            35 |     12 | desc      | 1_week     |         0 |
|            36 |     12 | desc      | 1_day      |         0 |
|            37 |     13 | desc      | forever    |         1 |
|            38 |     13 | desc      | 1_week     |         0 |
|            39 |     13 | desc      | 1_day      |         0 |
|            40 |     14 | desc      | forever    |         1 |
|            41 |     14 | desc      | 1_week     |         0 |
|            42 |     14 | desc      | 1_day      |         0 |
|            43 |     15 | desc      | forever    |         1 |
|            44 |     15 | desc      | 1_week     |         0 |
|            45 |     15 | desc      | 1_day      |         0 |
|            46 |     16 | desc      | forever    |         1 |
|            47 |     16 | desc      | 1_week     |         0 |
|            48 |     16 | desc      | 1_day      |         0 |
+---------------+--------+-----------+------------+-----------+
```

`RankLeaderboards()`

This feature shows how a user is doing on a game's leaderboard – relates to **Question 4**. When given a user and a game, the procedure displays the user's best score, the rank on the entire leaderboard of that game, and

a suggestion of what percentile their score is in compared with everybody who has ever played that game.
Example query:
The following query looks up the user '`BobHope`' and the game '`GTA V`' (`GameID=1`).

```
mysql> CALL RankLeaderboards('BobHope', 1);
+------+---------------+-----------+
| rank | top_x_percent | BestScore |
+------+---------------+-----------+
|    7 |       17.0732 |        87 |
+------+---------------+-----------+
```

This shows that `BobHope` ranked 6th on the leaderboard for GTA V, his best score is 91, and he is in the top 17 % of scores for this game.

`GetFriendsLeaderboard()`

This feature enables users to see `Leaderboards` with only their friends on it (**Question 11**).

Example query:
This query lists friends of `WillWoodhead` who have also registered high scores on GTA V (`GameID=1`).

```
mysql> CALL GetFriendsLeaderboard('WillWoodhead', 1);
+--------------+-------+---------+
| Username     | Score | units   |
+--------------+-------+---------+
| ScarlettJo   |    98 |  points |
| WillWoodhead |    95 |  points |
| AlexParrott  |    93 |  points |
| ScarlettJo   |    89 |  points |
| WillWoodhead |    79 |  points |
| AlexParrott  |    75 |  points |
| DavidCameron |    73 |  points |
| WillWoodhead |    73 |  points |
| DavidCameron |    70 |  points |
| ScarlettJo   |    55 |  points |
| ScarlettJo   |    52 |  points |
| DavidCameron |    14 |  points |
| DavidCameron |    14 |  points |
| AlexParrott  |    12 |  points |
| WillWoodhead |     9 |  points |
+--------------+-------+---------+
```

`GetLeaderboard()`

This procedure creates a Leaderboard with sort orders and score formats (**Question 17**). This feature allows leaderboards to be used for games with ascending sort orders and descending sort orders. It also allows for points to have a format, e.g. money, miles, coins etc.

Example query:
```
mysql> CALL GetLeaderboard(1);
+-------------------+-------+---------+---------------------+
| Username          | Score | Units   | TimeOfScore         |
+-------------------+-------+---------+---------------------+
| BradPitt          |    98 |  points | 2014-05-06 11:08:51 |
| ScarlettJo        |    98 |  points | 2014-05-06 11:08:51 |
| JamesHamblion     |    96 |  points | 2014-05-06 11:08:51 |
| BobHope           |    95 |  points | 2014-05-06 11:08:51 |
| WillWoodhead      |    95 |  points | 2014-05-06 11:08:51 |
| JamesHamblion     |    95 |  points | 2014-05-06 11:08:51 |
| BobHope           |    94 |  points | 2014-05-06 11:08:51 |
| AlexParrott       |    93 |  points | 2014-05-06 11:08:51 |
| GeorgeClooney     |    91 |  points | 2014-05-06 11:08:51 |
| JamesHamblion     |    91 |  points | 2014-05-06 11:08:51 |
| ScarlettJo        |    89 |  points | 2014-05-06 11:08:51 |
| BradPitt          |    84 |  points | 2014-05-06 11:08:51 |
| JamesHamblion     |    83 |  points | 2014-05-06 11:08:51 |
| BarackObama       |    81 |  points | 2014-05-06 11:08:51 |
| WillWoodhead      |    79 |  points | 2014-05-06 11:08:51 |
| JamesHamblion     |    78 |  points | 2014-05-06 11:08:51 |
| BobHope           |    77 |  points | 2014-05-06 11:08:51 |
| GeorgeClooney     |    75 |  points | 2014-05-06 11:08:51 |
| AlexParrott       |    75 |  points | 2014-05-06 11:08:51 |
| WillWoodhead      |    73 |  points | 2014-05-06 11:08:51 |
| DavidCameron      |    73 |  points | 2014-05-06 11:08:51 |
| GeorgeClooney     |    73 |  points | 2014-05-06 11:08:51 |
| BradPitt          |    71 |  points | 2014-05-06 11:08:51 |
| DavidCameron      |    70 |  points | 2014-05-06 11:08:51 |
| AliceInWonderland |    66 |  points | 2014-05-06 11:08:51 |
| BarackObama       |    63 |  points | 2014-05-06 11:08:51 |
| ScarlettJo        |    55 |  points | 2014-05-06 11:08:51 |
| ScarlettJo        |    52 |  points | 2014-05-06 11:08:51 |
| BradPitt          |    48 |  points | 2014-05-06 11:08:51 |
| JamesHamblion     |    42 |  points | 2014-05-06 11:08:51 |
| BarackObama       |    29 |  points | 2014-05-06 11:08:51 |
| AliceInWonderland |    18 |  points | 2014-05-06 11:08:51 |
| DavidCameron      |    14 |  points | 2014-05-06 11:08:51 |
| DavidCameron      |    14 |  points | 2014-05-06 11:08:51 |
| AlexParrott       |    12 |  points | 2014-05-06 11:08:51 |
| BradPitt          |    10 |  points | 2014-05-06 11:08:51 |
| WillWoodhead      |     9 |  points | 2014-05-06 11:08:51 |
| BarackObama       |     7 |  points | 2014-05-06 11:08:51 |
| BarackObama       |     3 |  points | 2014-05-06 11:08:51 |
| BarackObama       |     3 |  points | 2014-05-06 11:08:51 |
| JamesHamblion     |     3 |  points | 2014-05-06 11:08:51 |
+-------------------+-------+---------+---------------------+
```

This query calls the leaderboard with ID 1. The criteria in the Leaderboard table will inform how this table is arranged. If the order is Ascending it will be displayed so. It will also display the units for the score. This result shows the sort order as descending with the units in points.

Match Features

As an extra feature (**Question 20**), matches can be created between users on a certain game. This database allows users to start matches with each other in groups. The process of this arrangement is as follows:
1. One user creates a match.
2. This user then invites people to join the match
3. This invitation is a match request which is initially pending.
4. When an invitee says yes to the match request, they are added to the match.
5. If they say no, the match request is terminated.
6. When the match is over, the match itself is terminated.

Example usage:
```
/* a user creates a match */
CALL CreateMatch (3, 2, 4, "Family round robin");
SELECT * FROM Matches;

/*request other users who play the game to join the game*/
CALL MatchRequesting(3, 6, 1);
CALL MatchRequesting(3, 7, 1);
CALL MatchRequesting(3, 12, 1);
SELECT * FROM MatchRequest;

/* the other users accept the request*/
UPDATE MatchRequest
SET Response = 'Accepted'
WHERE MatchRequestID = 1;

UPDATE MatchRequest
SET Response = 'Accepted'
WHERE MatchRequestID = 2;

UPDATE MatchRequest
SET Response = 'Accepted'
WHERE MatchRequestID = 3;

SELECT * FROM Matches;
SELECT * FROM MatchToUserToGame;

/* one of the players quits the game*/
UPDATE MatchToUserToGame
SET PlayerStatus = 'Quit'
WHERE MatchID = 1 AND UserToGameID = 6;

SELECT * FROM Matches;
```

This output the following:
```
mysql> SELECT * FROM Matches;
+---------+--------------------+-----------+------------+------------+------------+-------------+
| MatchID | MatchName          | Initiator | MinPlayers | MaxPlayers | NoOfPlayer | Status      |
+---------+--------------------+-----------+------------+------------+------------+-------------+
|       1 | Family round robin |         3 |          2 |          4 |          1 | not_started |
+---------+--------------------+-----------+------------+------------+------------+-------------+
```

In this table it is possible to see that a match has been created.
```
mysql> SELECT * FROM MatchRequest;
+----------------+------------+--------------+---------+----------+
| MatchRequestID | SendingUTG | ReceivingUTG | MatchID | Response |
+----------------+------------+--------------+---------+----------+
|              1 |          3 |            6 |       1 | Pending  |
|              2 |          3 |            7 |       1 | Pending  |
|              3 |          3 |           12 |       1 | Pending  |
+----------------+------------+--------------+---------+----------+
```

In this table the friend requests are still pending.
```
mysql> SELECT * FROM Matches;
+---------+--------------------+-----------+------------+------------+------------+-------------+
| MatchID | MatchName          | Initiator | MinPlayers | MaxPlayers | NoOfPlayer | Status      |
+---------+--------------------+-----------+------------+------------+------------+-------------+
|       1 | Family round robin |         3 |          2 |          4 |          4 | not_started |
+---------+--------------------+-----------+------------+------------+------------+-------------+
```

In this table the friend requests have been accepted so one can see that `NoOfPlayer` attribute has risen from 1 in the top table to 4 in this table. This is achieved through triggers.

```
mysql> SELECT * FROM MatchToUserToGame;
+---------+--------------+--------------+
| MatchID | UserToGameID | PlayerStatus |
+---------+--------------+--------------+
|       1 |            3 | playing      |
|       1 |            6 | playing      |
|       1 |            7 | playing      |
|       1 |           12 | playing      |
+---------+--------------+--------------+
```

This table shows all the UserToGameIDs playing in the match.

```
mysql> SELECT * FROM Matches;
+---------+-------------------+-----------+------------+------------+------------+-------------+
| MatchID | MatchName         | Initiator | MinPlayers | MaxPlayers | NoOfPlayer | Status      |
+---------+-------------------+-----------+------------+------------+------------+-------------+
|       1 | Family round robin |         3 |          2 |          4 |          3 | not_started |
+---------+-------------------+-----------+------------+------------+------------+-------------+
```

Once a players quits, the `NoOfPlayer` attribute goes down to 3.

This process is achieved by using a number of triggers and procedures all shown in the code. This will allow users to create multiplayer matches with each other, invite other to play, and play together. This process serves as only the starting point for this topic. Matches could become more complex and more automated if more time were available.