Department of Computer Science

Databases Group Project
Research Review
COMS20700

Contributors:

James Hamblion, jh12021, 0.333
Alex Parrott, ap12332, 0.333
Will Woodhead, ww13694, 0.333

MSc Computer Science
MS50
May 2014

Table of Contents

Introduction

This database is written in MySQL version 5. Source code files are as follows:

- `createTablesAll.sql` : Creates all database relations.
- `triggers.sql` : Adds all database functionality (procedures and triggers).
- `dummyData.sql` : Fills relations with dummy values for example purposes.
- `dropTablesAll.sql` : Deletes all tables in the correct order to avoid foreign key restrictions.

All source code files are fully commented to explain how functions, procedures and triggers work.

The group was a team of 3. All team members contributed over 30 hours work toward this project. Despite our small group size we feel this is a strong submission.

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 needed to create specific Leaderboards.
`Plays` : A record is inserted every time a user plays any 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).

User Account Features

When user names are created by new users and entered into the `UserPublic` relation, the chosen account user name is checked against a list of obscene or offensive words (**Question 8**). If an offensive word is contained within the user name then the account is locked.

Checks are performed through the use of a trigger linked to the `UserPublic` relation. The trigger will automatically execute before new rows are inserted. It passes the user name to a function in the database to process called `isUserNameRude()`.

`isUserNameRude()`

This function checks the user name against all offensive terms contained within the `RudeWord` table. The function searches for character sequences in all parts of the string i.e. sub-strings checked as well as the whole word regardless of individual character cases (capitals or not).

Parameters:

- The `UserName` of the new account.

Example Usage:
SQL command to enter a new user:

```
INSERT INTO UserPublic
            VALUES('KsHiTer','./avatar.jpg',CURDATE(),'Online',NULL,'I am logged
            in!');
```

The output for the associated row entry inserted into the `UserPublic` relation has the field for the `AccountStatus` set to `Locked` as shown below:

```
+-------------------+---------------+
| username          | AccountStatus |
+-------------------+---------------+
| AlexParrott       | Online        |
| AliceInWonderland | Online        |
| BarackObama       | Online        |
| BobHope           | Online        |
| BradPitt          | Online        |
| DavidCameron      | Online        |
| GeorgeClooney     | Online        |
| JamesHamblion     | Offline       |
| KsHiTer           | Locked        |
| ScarlettJo        | Online        |
| WillWoodhead      | Online        |
+-------------------+---------------+
```

All locked user accounts can be displayed with the SQL query:

```
SELECT * FROM UserPublic WHERE AccountStatus = ‘Locked’;
```

Note also that the `RudeWord` table can have new words added/removed as required by the database administrator without requiring any alterations to the database function `isUserNameRude()`. This can be done with the SQL syntax:

```
INSERT INTO RudeWord
      VALUES ('xxxx');
```

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

Step 1: Create a request

- 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`.

Step 2: Respond to the request

- 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()`

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()`

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.

Parameters:

- 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.

Parameters:

- The `UserName` of the user to lookup friends.

Example usage:

```
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.

Parameters:

- The `UserName` of the user to suggest friends to.

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**.

Parameters:

- The `GameID` (INT) of the game to lookup.

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 |
+---------+--------------------+-------------+
```

Achievement Features

AchievementsForUserGame()

This procedure displays the achievement status for a specific user in a specific game (**Question 13)**. It contains several queries that use the UserToGame, Achievement, and AchievementToUserToGame relations to form a status output of the form: 16 of 80 achievements (95 points).

Parameters:

1. The UserName of the of the user.
2. The GameID of the game.

Example Queries:

```
CALL AchievementsForUserGame('WillWoodhead', 3);
+--------------------------------+
| Your_Achievements              |
+--------------------------------+
| 3 of 5 achievements (50 points) |
+--------------------------------+

CALL AchievementsForUserGame('AlexParrott', 3);
+--------------------------------+
| Your_Achievments               |
+--------------------------------+
| 0 of 5 achievements (0 points) |
+--------------------------------+

CALL AchievementsForUserGame('WillWoodhead', 4);
+--------------------------------+
| Your_Achievements              |
+--------------------------------+
| Error: game not owned by user! |
+--------------------------------+
```

ShowStatusScreen()

This procedure shows a status screen for the requested user. It shows their UserName, UserStatus, the number of games they own, their total number of achievement points and their total number of friends (**Question 14)**.

Parameters:

- The UserName of the of the user to show status.

Example Usage:

```
CALL ShowStatusScreen('AlexParrott');
+------------+----------------+----------------------+-----------------------------------+------------------+
| Username   | Status_Line    | Number_of_Games_Owned | Total_Number_of_Achievement_Points | Number_of_Friends |
+------------+----------------+----------------------+-----------------------------------+------------------+
| AlexParrott | I am logged in! |                    3 |                                40 |                4 |
+------------+----------------+----------------------+-----------------------------------+------------------+
```

ListUserGameAchievements()

This procedure lists the achievements for a particular game that a user has earned and also lists those that they have not earned (if they are not hidden i.e. Achievement relation attribute hiddenFlag = False) (**Question 15)**. Earned achievements are shown first. The procedure also determines whether the earned description (attribute postDescription) or the not earned description (attribute preDescription) should be displayed.

Parameters:

1. The UserName of the of the user to lookup.
2. The GameID of the Game to lookup.

Example Usage:

```
CALL ListUserGameAchievements('WillWoodhead', 3);
```

Title	PointValue	Description	DateGained
Goalie Scorer	20	Scored with your goal keeper	2014-02-12
Always Friendly	20	Crossed for a Friend to score	2013-12-13
Fowler	10	Received 5 red cards in a game	2013-04-26
Penalty guru	50	Win 50 games through penalties	NULL

Note: The 'Post and in' achievement not shown because it is hidden and has not been earned. The 'Fowler' achievement is shown because it has been earned (even though it has the hidden flag set). All example achievements in the database are shown below to support this.

achievementID	gameid	title	hiddenFlag	pointValue	postDescription	preDescription
1	1	I wish I was a policeman!	0	10	You stole 100 police cars	Steal 100 police cars
2	2	Up close and personal	0	60	You killed 80 creatures with a melee weapon	Kill 80 creatures with a melee weapon
3	3	Penalty guru	0	50	Won 50 games through penalties	Win 50 games through penalties
4	3	Fowler	1	10	Received 5 red cards in a game	Get 5 red cards in a game
5	4	Score obsessed	0	30	Achieved a score of 3000000	Achieve a score of 3000000
6	3	Always Friendly	0	20	Crossed for a Friend to score	Cross for a Friend to score
7	3	Goalie Scorer	0	20	Scored with your goal keeper	Score with your goal keeper
8	3	Post and in	1	20	Scored off the post or cross bar in a match	Score off the post or cross bar in a match

CompListGameAchievFriend()

This procedure produces a game and achievement comparison screen between a user and one of their friends (**Question 16)**. All games owned by both the user and his/her friend are shown. If one of them does not own the game then their respective achievement point parts of the output are left blank.

Parameters:

1. The `UserName` of the of the user to lookup.
2. The `UserName` of their friend.

Example Usage:

```
CALL CompListGameAchievFriend('AlexParrott', 'WillWoodhead');
```

Game_Title	Your_Achievement_Points	Achievement_Points_of_WillWoodhead
GTA V	10	0
FIFA 14	0	50
Angry Birds	0	
Crash Bandicoot		0
Bike Runner		0
mission Impossible		0

```
CALL CompListGameAchievFriend('AlexParrott', 'JamesHamblion');
```

Game_Title	Your_Achievement_Points	Achievement_Points_of_JamesHamblion
Angry Birds	30	0
GTA V	10	0
FIFA 14	0	
The Last of Us		0

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 |
...
```

`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.

Parameters:

1. The `UserName` of the of the user to lookup.
2. The `GameID` of the Game to lookup.

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 7th on the leaderboard for GTA V, his best score is 87, 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 |
...
```

`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.

Parameters:

- The `GameID` of the Game leaderboard to lookup.

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 |
...
```

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.

Appendices

Team Methodology

We initially approached this coursework by discussing our team development methodology together. This methodology included:

1. Constant team communication
- Google Hangouts for remote conference meetings
- GitHub for remote version control
- Email for correspondence
- Scheduled in-person meetings

2. Iterative and incremental development
- build the program bit by bit
- test each addition continually
- revisit to old tests when new alterations are made
- upload to GitHub frequently

3. Continuous testing
- testing was undertaken all though development
- extensive dummy data was created to test the database

4. Agile development
- database design, development, implementation, analysis and testing at the same time

5. Division of Labour
- we divided the tasks into three equal parts. This allowed us to work more efficiently from remote locations.

Because all three members of the team have different schedules and working locations, we chose to use Google Hangups to talk in three way conferences. This allowed us to communicate effectively and hence work efficiently. We used GitHub to facilitate remote version control between team members. GitHub has been a useful tool for remote collaboration.

Methodology for the Database Design

As a team we had very similar views on our approach to the design of this database:

1. The data should be extensively normalised to avoid repetition of data.
- This would ensure that inconsistencies do not creep into the data base

2. The data should be divided into sensible 'object'-like tables
- This allows for a, understandable, logical table
- it also allows for clear queries to be implemented

3. The database should be on MySQL
- all team member had access to a MySQL database

4. Triggers should be kept to a minimum
- Triggers are a way to introduce unwanted errors into a database.

Overall our database design methodology was put in place to ensure an efficient and problem-free implementation of the database. Bugs can quickly manifest themselves into a database especially when three team members are working together on the database. When data is atomic and consistent, the database is more robust.

Project Schedule

In our first meeting together, the team set out an overall schedule and approach for the project. We did not

want to dive straight into implementation because this can cause unwanted design mistakes. We therefore approached the design of the database with great care. Our approach was as follows:

1. Meet as a team to discuss the entire project schedule.

- This allowed us to make initial decisions about how the project was going to be implemented
- We also created a schedule for the project
- We also divided up some of the tasks between the team members.

2. Create a first draft ER diagram to discuss together.

- Careful consideration went into creating our ER diagram.
- we ensured that the data was normalised
- we ensured that the data was understandable

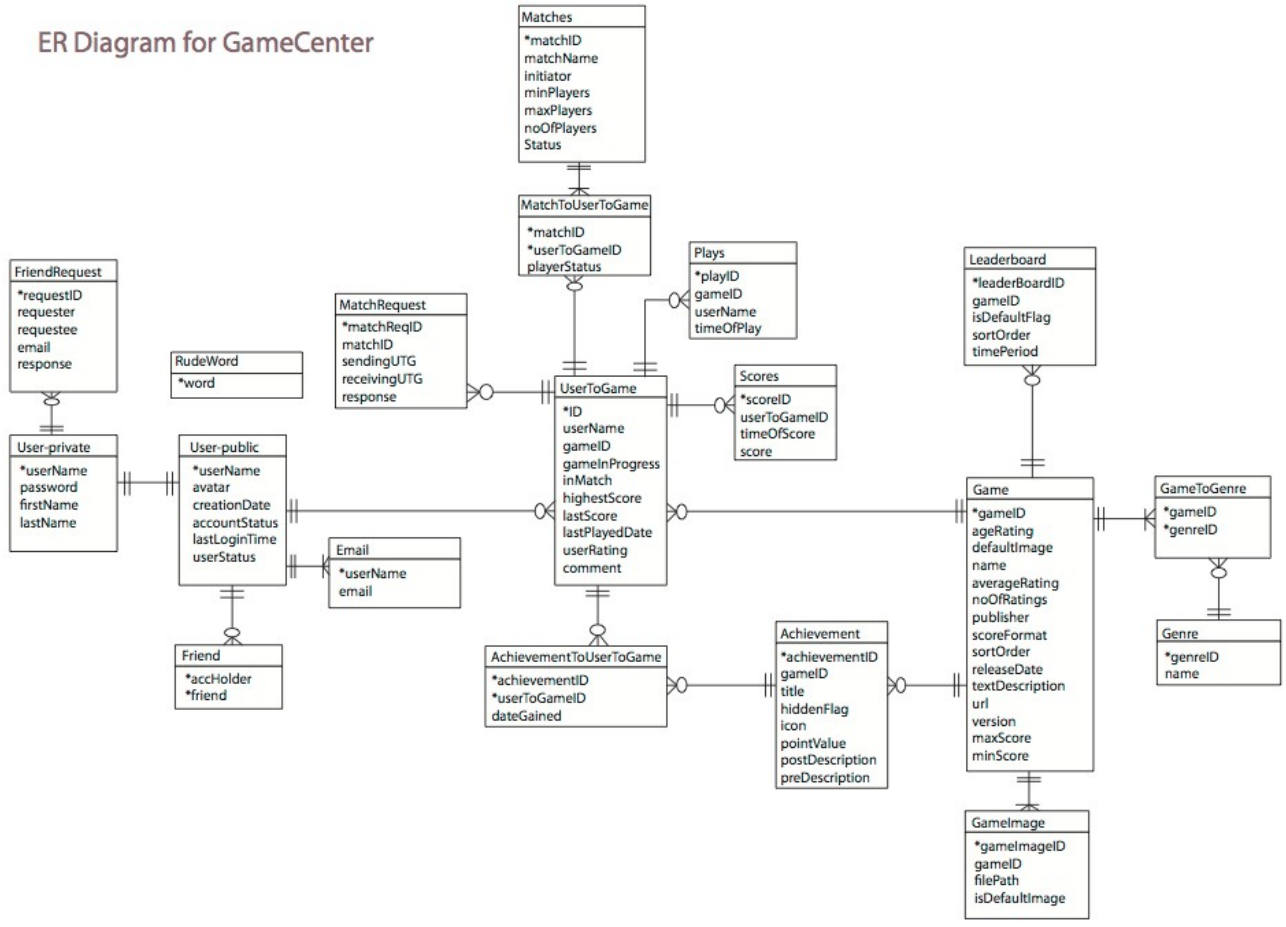
3. Divide up the tasks between the three team members

- we divided up the tasks between the three members to make the schedule more time efficient

4. Implement the database through continuous integration with each other.

- By constant integration, cross programmer bugs were kept to an absolute minimum

ER Diagram



SQL Code

Below is the code used to create all relations and add database functionality. Dummy data and the drop table files are not included here.

createTablesAll.sql

```
/* This file holds all the create statements to generate the database. */

/* User information  relations */
CREATE TABLE UserPublic(
	UserName VARCHAR(20) NOT NULL,
	Avatar VARCHAR(50) NOT NULL,
	CreationDate DATE NOT NULL,
	AccountStatus ENUM('Online','Offline', 'Locked') NOT NULL DEFAULT'Offline',
	LastLogin TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
	UserStatus VARCHAR(100) DEFAULT NULL,

	CONSTRAINT pkUserName
		PRIMARY KEY(UserName)
);

CREATE TABLE UserPrivate(
	UserName VARCHAR(20) NOT NULL,
	Password VARCHAR(20) NOT NULL,
	FirstName VARCHAR(20) NOT NULL,
	LastName VARCHAR(20) NOT NULL,

	CONSTRAINT pkUserName
		PRIMARY KEY(UserName),
	CONSTRAINT fkUserName
		FOREIGN Key(UserName)
		REFERENCES UserPublic(UserName)
);

CREATE TABLE Email(
	UserName VARCHAR(20) NOT NULL,
	Email VARCHAR(30) NOT NULL UNIQUE,

	CONSTRAINT pkUserNameEmail
		PRIMARY KEY(UserName),
	CONSTRAINT fkUserNameEmail
		FOREIGN Key(UserName)
		REFERENCES UserPublic(UserName)
);

/* Game relations */
CREATE TABLE Game(
	GameID INT NOT NULL AUTO_INCREMENT,
	AgeRating ENUM('3','7','12','16','18') NOT NULL,
	DefaultImage VARCHAR(50) NOT NULL,
	Name VARCHAR(30) NOT NULL,
	AverageRating FLOAT DEFAULT NULL,
	NoOfRatings INT DEFAULT NULL,
	Publisher VARCHAR(20) NOT NULL,
	ScoreFormat VARCHAR(20) NOT NULL DEFAULT 'points',
	SortOrder ENUM('asc','desc') NOT NULL DEFAULT 'desc',
	ReleaseDate DATE NOT NULL,
	TextDescription VARCHAR(50)NOT NULL,
	Url VARCHAR(100) DEFAULT NULL,
	Version DECIMAL(4,2) DEFAULT '1.0',
	MaxScore INT DEFAULT NULL,
	MinScore INT DEFAULT NULL,

	CONSTRAINT pkGameID
		PRIMARY KEY(GameID)
);

CREATE TABLE Genre(
	GenreID INT NOT NULL,
	Name VARCHAR(20) NOT NULL,

	CONSTRAINT pkGenreID
		PRIMARY KEY(GenreID)
);

CREATE TABLE GameToGenre(
	GameID INT NOT NULL,
	GenreID INT NOT NULL,

	CONSTRAINT pkIDs
```

```
		PRIMARY KEY(GameID,GenreID),
	CONSTRAINT fkGameID
		FOREIGN Key(GameID)
		REFERENCES Game(GameID),
	CONSTRAINT fkGenreID
		FOREIGN Key(GenreID)
		REFERENCES Genre(GenreID)
);

CREATE TABLE GameImage (
	GameImageID INT NOT NULL,
	GameID INT NOT NULL,
	FilePath VARCHAR(100),
	DefaultImage ENUM('True','False') NOT NULL,

	CONSTRAINT pkGameImgID
		PRIMARY KEY (gameImageID),
	CONSTRAINT fkGameID2
		FOREIGN Key(GameID)
		REFERENCES Game(GameID)
);

/* Linking relation for Users & Games */
CREATE TABLE UserToGame(
	ID INT NOT NULL AUTO_INCREMENT,
	UserName VARCHAR(20) NOT NULL,
	GameID INT NOT NULL,
	GameInProgress ENUM('Yes','No') NOT NULL DEFAULT'No',
	InMatch ENUM('Yes','No') NOT NULL DEFAULT'No',
	HighestScore INT NOT NULL DEFAULT'0',
	LastScore INT DEFAULT '0',
	LastPlayDate DATE DEFAULT NULL,
	UserRating FLOAT NOT NULL DEFAULT'0.0',
	AgeRating ENUM('Unrated','1','2','3','4','5') NOT NULL DEFAULT'Unrated',
	Comments VARCHAR(100) NOT NULL DEFAULT'No comments',

	CONSTRAINT pkID
		PRIMARY KEY(ID),
	CONSTRAINT fk_U2G_UserName
		FOREIGN KEY(UserName)
		REFERENCES UserPublic(UserName),
	CONSTRAINT fk_U2G_GameID
		FOREIGN KEY(GameID)
		REFERENCES Game(GameID)
);

/* Friendship Relations */
CREATE TABLE Friends(
	AccHolder VARCHAR(20) NOT NULL,
	Friend VARCHAR(20) NOT NULL,

	CONSTRAINT pkFriends
		PRIMARY KEY(AccHolder, Friend),
	CONSTRAINT fkUser
		FOREIGN Key(AccHolder)
		REFERENCES UserPublic(UserName),
	CONSTRAINT fkUser2
		FOREIGN Key(Friend)
		REFERENCES UserPublic(UserName)
);

/* Friend request relation with game invites */
CREATE TABLE FriendRequest(
	RequestID INT NOT NULL AUTO_INCREMENT,
	Requester VARCHAR(20) NOT NULL,
	Requestee VARCHAR(20) DEFAULT NULL,
	Email VARCHAR(30) DEFAULT NULL,
	Response ENUM('Pending','Accepted','Declined','Completed') NOT NULL DEFAULT'Pending',

	CONSTRAINT pkFriendReq
		PRIMARY KEY(RequestID),
	CONSTRAINT fkRequester
		FOREIGN Key(Requester)
		REFERENCES UserPrivate(UserName),
	CONSTRAINT fkRequestee
		FOREIGN Key(Requestee)
		REFERENCES UserPrivate(UserName),
	CONSTRAINT fkReqEmail
		FOREIGN Key(Email)
		REFERENCES Email(Email)
);

/* Leaderboards */
CREATE TABLE Leaderboard(
	LeaderboardID INT NOT NULL AUTO_INCREMENT,
	GameID INT NOT NULL,
	SortOrder ENUM('asc','desc') NOT NULL DEFAULT 'desc',
```

```
        TimePeriod ENUM('forever','1_year','1_week','1_day') NOT NULL DEFAULT 'forever',
        IsDefault BOOLEAN NOT NULL DEFAULT 0,

        CONSTRAINT pkLdbdID
                PRIMARY KEY (LeaderboardID),
        CONSTRAINT fk_ldbd_GameID
                FOREIGN KEY(GameID)
                REFERENCES Game(GameID)
);

/* Plays relation records any time a user plays a game */
CREATE TABLE Plays(
        PlayID INT AUTO_INCREMENT,
        GameID INT NOT NULL,
        UserName VARCHAR(20) NOT NULL,
        TimeOfPlay TIMESTAMP,

        CONSTRAINT pkNoOfPlaysID
                PRIMARY KEY(PlayID)
);

/* Scores relation records all of the scores made on any game*/
CREATE TABLE Scores(
        ScoreID INT AUTO_INCREMENT,
        UserToGameID INT NOT NULL,
        Score INT NOT NULL,
        TimeOfScore TIMESTAMP NOT NULL,

        CONSTRAINT pk_scores
                PRIMARY KEY (ScoreID)
);

/* Acheivement relation */
CREATE TABLE Achievement (
        achievementID INT AUTO_INCREMENT,
        gameID INT NOT NULL,
        title VARCHAR(50) NOT NULL,
        hiddenFlag BIT(1) NOT NULL DEFAULT 0, /* achievements shown by default */
        icon INT DEFAULT 0,
        pointValue INT NOT NULL DEFAULT 1,
        postDescription VARCHAR(200),
        preDescription VARCHAR(200),

        CONSTRAINT pkAchievement
                PRIMARY KEY (achievementID),
        CONSTRAINT fkAchievToGame
                FOREIGN KEY (GameID)
                REFERENCES Game (GameID)
);

/* Linking relation for Achievements, Users & Games */
CREATE TABLE AchievementToUserToGame (
        achievementID INT,
        userToGameID INT,
        dateGained DATE NOT NULL,

        CONSTRAINT pkAchievUsrGame
                PRIMARY KEY (userToGameID, achievementID),
        CONSTRAINT fk
                FOREIGN KEY (userToGameID)
                REFERENCES UserToGame (ID)
);

/* Matches relations */
CREATE TABLE Matches (
        MatchID INT AUTO_INCREMENT,
        MatchName VARCHAR(30) NOT NULL,
        Initiator INT NOT NULL,
        MinPlayers INT NOT NULL DEFAULT 2,
        MaxPlayers INT NOT NULL DEFAULT 2,
        NoOfPlayer INT NOT NULL DEFAULT 1,
        Status ENUM('not_started', 'in_play', 'ended') NOT NULL DEFAULT 'not_started',

        CONSTRAINT pkMatch
                PRIMARY KEY (MatchID),
        CONSTRAINT fkmatch1
                FOREIGN KEY (Initiator)
                REFERENCES UserToGame(ID)
);

/* Linking relation for matches, users and games */
CREATE TABLE MatchToUserToGame(
        MatchID INT NOT NULL,
        UserToGameID INT NOT NULL,
        PlayerStatus ENUM('playing', 'paused', 'quit') NOT NULL DEFAULT 'playing',

        CONSTRAINT pkMatchToUserToGame
```

```
		PRIMARY KEY (MatchID, UserToGameID),
	CONSTRAINT fkMTUTG1
		FOREIGN KEY (MatchID)
		REFERENCES Matches(MatchID),
	CONSTRAINT fkmtutg2
		FOREIGN KEY (UserToGameID)
		REFERENCES UserToGame(ID)
);

CREATE TABLE MatchRequest (
	MatchRequestID INT AUTO_INCREMENT,
	SendingUTG INT NOT NULL,
	ReceivingUTG INT NOT NULL,
	MatchID INT NOT NULL,
	Response ENUM('Accepted','Denied','Pending') NOT NULL DEFAULT'Pending',

	CONSTRAINT pkmatchrequest
		PRIMARY KEY (MatchRequestID),
	CONSTRAINT fkmatchrequest
		FOREIGN KEY (SendingUTG)
		REFERENCES UserToGame(ID),
	CONSTRAINT fkmatchrequest2
		FOREIGN KEY (ReceivingUTG)
		REFERENCES UserToGame(ID)
);

/* Relation to hold contain obscene/offensive terms */
CREATE TABLE RudeWord (
	word VARCHAR(50),

	CONSTRAINT pkRudeWord
		PRIMARY KEY (word)
);


SHOW tables;
```

triggers.sql

Some comments have been removed for space purposes.

```
/*
PROCEDURES, FUNCTIONS & TRIGGERS

Included are the relevant procedures and functions for the specified coursework
questions. Where appropriate triggers to activate them are listed afterward.
*/

/*
QUESTION 1
Author: Alex Parrott
*/
DROP PROCEDURE IF EXISTS ListGameOwners;
DELIMITER $$
CREATE PROCEDURE ListGameOwners(IN gameVar INT)
BEGIN
	/* Looks up the Game with the ID provided as parameter */
	SELECT UserPublic.UserName AS Owners
	FROM Game,UserPublic,UserToGame
	WHERE UserPublic.UserName=UserToGame.UserName
	AND Game.GameID=UserToGame.GameID
	AND Game.GameID=gameVar;

END $$
DELIMITER ;

/*
QUESTION 2 & QUESTION 3
Author: Alex Parrott
*/
DROP PROCEDURE IF EXISTS UpdateAverage;
DELIMITER $$
CREATE PROCEDURE UpdateAverage(IN updated INT)
BEGIN
	UPDATE Game
		/* Set new rating count for new rated Game */
		SET NoOfRatings = (
			SELECT COUNT(UserRating)
			FROM UserToGame
			WHERE UserToGame.GameID = Game.GameID
			AND UserToGame.GameID = updated)
		WHERE Game.GameID = updated;

		/* If new rating count is over 10, update the average */
```

```
		IF (SELECT NoOfRatings FROM Game WHERE GameID = updated) >= 10
		THEN BEGIN
			UPDATE Game
				SET AverageRating = (
					SELECT AVG(UserRating)
					FROM UserToGame
					WHERE UserToGame.GameID = Game.GameID
					AND UserToGame.GameID = updated)
				WHERE Game.GameID = updated;
		END; END IF;
END $$
DELIMITER ;

/*
QUESTION 4
Author: Will Woodhead
*/
DROP PROCEDURE IF EXISTS RankLeaderboards;
DELIMITER $$
CREATE PROCEDURE RankLeaderboards(User VARCHAR(30), GID INT)
BEGIN
	SET @rank=0;
	/* @count is the number of users who have registered a score in a particular game */
	SET @count = (
		SELECT COUNT(*)
		FROM Scores
		WHERE UserToGameID IN (
			SELECT ID
			FROM UserToGame
			WHERE GameID = GID)
	);

	/* Check whether the scores are ascending or descending */
	IF ((SELECT SortOrder FROM Game WHERE GameID=GID) = 'asc')
	THEN
		SELECT r AS rank, topXP AS top_x_percent, scor AS BestScore FROM (
			SELECT @rank:=@rank+1 AS r, (@rank/@count)*100 AS topXP ,UserToGameID,Score AS
scor
			FROM Scores
			WHERE UserToGameID IN (
				SELECT ID
				FROM UserToGame
				WHERE GameID = GID)
			ORDER BY Score ASC
		) AS temp WHERE UserToGameID = (
				SELECT ID
				FROM UserToGame
				WHERE Username = User
				AND GameID = GID )
		ORDER BY BestScore ASC LIMIT 1;
	ELSE
		SELECT r AS rank, topXP AS top_x_percent, scor AS BestScore FROM (
			SELECT @rank:=@rank+1 AS r,(@rank/@count)*100 AS topXP, UserToGameID, Score AS
scor
			FROM Scores
			WHERE UserToGameID IN (
				SELECT ID
				FROM UserToGame
				WHERE GameID = GID)
			ORDER BY Score DESC
		) AS temp WHERE UserToGameID = (
				SELECT ID
				FROM UserToGame
				WHERE Username = User
				AND GameID = GID  )
		ORDER BY BestScore DESC LIMIT 1;
	END IF;
END; $$
DELIMITER ;

/*
QUESTION 5
Author: Will Woodhead
*/
DROP PROCEDURE if exists TopTens;
DELIMITER $$
CREATE PROCEDURE TopTens()
BEGIN
	SET @row:=0;
	SET @prev:=null;
	SELECT genre,name,AverageRating
	FROM (
		SELECT Genre.Name AS genre,
		Game.Name AS name,
		AverageRating,
		@row:= IF(@prev = Genre.Name, @row + 1, 1) AS row_number,
		@prev:= Genre.Name
```

```
                FROM Game, Genre, GameToGenre
                WHERE Game.GameID = GameToGenre.GameID
                AND Genre.GenreID = GameToGenre.GenreID
                ORDER BY Genre.Name, AverageRating DESC)
        AS src
        WHERE row_number <= 10
        ORDER BY genre, AverageRating DESC;
END; $$
DELIMITER ;

/*
QUESTION 6
Author: Alex Parrott
*/
DROP FUNCTION IF EXISTS CatchCheaters;
DELIMITER $$
CREATE FUNCTION CatchCheaters(game INT,score INT)
RETURNS INT
BEGIN
        DECLARE checkedScore INT;
        DECLARE minimum INT;
        DECLARE maximum INT;

        /* Initialise the score to return */
        SELECT score
        INTO checkedScore;
        /* Get max and min scores for the Game being updated */
        SELECT MinScore
        INTO minimum
        FROM Game
        WHERE Game.GameID = game;

        SELECT MaxScore
        INTO maximum
        FROM Game
        WHERE Game.GameID = game;

        /*
        If the new score is < min or > max score, set it to the minimum for that Game.
        */
        IF(
                score < minimum
                OR
                score > maximum
        )
        THEN
                SET checkedScore = minimum;
        END IF;

        /* Return the final checked score */
        RETURN checkedScore;
END $$
DELIMITER ;

/*
QUESTION 7
SEE Game_after_insert TRIGGER below.
Author: Will Woodhead
*/

/*
QUESTION 8
Author: James Hamblion
*/
DROP FUNCTION IF EXISTS isUserNameRude;
DELIMITER $$
CREATE FUNCTION isUserNameRude(usrname VARCHAR(50))
RETURNS INT
BEGIN
        DECLARE done INT DEFAULT FALSE;
        DECLARE obscene INT DEFAULT FALSE;
        DECLARE cmpWord VARCHAR(50);
        DECLARE cur CURSOR FOR SELECT word FROM RudeWord;
        DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

        OPEN cur;
        compare_loop: LOOP
                FETCH cur INTO cmpWord;
                /* Build search text */
                SET @searchtxt ='%';
                SET @searchtxt = @searchtxt + cmpWord;
                SET @SearchText = @SearchText + '%';
                /* Handler check */
            IF done THEN
                        LEAVE compare_loop;
            END IF;
                /* Word comparison check (Note STRCMP case insensitive by default) */
```

```
                SET obscene = STRCMP(@usrname, @searchtxt);
                IF obscene
                THEN
                        SET done := TRUE;
                END IF;
        END LOOP;
        CLOSE cur;
        RETURN obscene;
END; $$
DELIMITER ;

/*
QUESTION 9:
Author: Will Woodhead
*/
DROP PROCEDURE IF EXISTS Hotlist;
DELIMITER $$
CREATE PROCEDURE Hotlist()
BEGIN
        CREATE TABLE Hotlist (
                Ranking INT NOT NULL AUTO_INCREMENT,
                GameID INT NOT NULL,
                NOPLastWeek INT,
                CONSTRAINT pkID PRIMARY KEY(ranking)
        );

        INSERT INTO HotList (GameID, NOPLastWeek)
        SELECT GameID, COUNT(GameID) AS count
        FROM Plays
        WHERE Plays.TimeOfPlay > DATE(DATE_SUB(NOW(), INTERVAL 7 DAY))
        GROUP BY GameID
        ORDER BY count DESC;

        SELECT Ranking,Name, NOPLastWeek
        FROM Hotlist, Game
        WHERE Hotlist.GameID = Game.GameID ORDER BY NOPLastWeek DESC limit 10;
        DROP TABLE Hotlist;
END; $$
DELIMITER ;

/*
QUESTION 10
Author: Alex Parrott
*/
/* Procedure creates a Friendship Request - used to create & delete friendships */
DROP PROCEDURE IF EXISTS CreateRequest;
DELIMITER $$
CREATE PROCEDURE CreateRequest(IN User VARCHAR(20),reqFriend VARCHAR(30),deleteFlag INT,emailFlag
INT)
BEGIN
        /* Action friendships to delete */
        IF (deleteFlag)
        THEN
                IF (emailFlag)
                THEN
                        INSERT INTO FriendRequest(Requester,Email,Response)
                        VALUES(User,reqFriend,'Declined');
                ELSE
                        INSERT INTO FriendRequest(Requester,Requestee,Response)
                        VALUES(User,reqFriend,'Declined');
                END IF;
        /* Action new friendship requests */
        ELSE
                IF (emailFlag)
                THEN
                        INSERT INTO FriendRequest(Requester,Email)
                        VALUES(User,reqFriend);
                ELSE
                        INSERT INTO FriendRequest(Requester,Requestee)
                        VALUES(User,reqFriend);
                END IF;
        END IF;
END; $$
DELIMITER ;

/* Procedure processes all FriendRequests */
DROP PROCEDURE IF EXISTS ProcessRequest;
DELIMITER $$
CREATE PROCEDURE ProcessRequest(IN reqID INT)
BEGIN
        DECLARE Friend1 VARCHAR(20);
        DECLARE Friend2 VARCHAR(30);

        /* Assign UserNames to friends */
        SET Friend1 = (
                SELECT Requester
                FROM FriendRequest
```

```
		WHERE RequestID = reqID);
	SET Friend2 = (
		SELECT Requestee
		FROM FriendRequest
		WHERE RequestID = reqID);
	/* If Email is used for request then get the UserName */
	IF Friend2 IS NULL
	THEN
		SET Friend2 = (
			SELECT UserName
			FROM Email
			WHERE Email = (
				SELECT Email
				FROM FriendRequest
				WHERE RequestID = reqID)
		);
	END IF;

	/* Delete any completed requests */
	DELETE FROM FriendRequest
	WHERE Response = 'Completed';

	/* Delete any friendships for unwanted friendships */
	IF (SELECT Response FROM FriendRequest WHERE RequestID = reqID) = 'Declined'
	THEN
		DELETE FROM Friends
		WHERE AccHolder = Friend1
		AND Friend = Friend2;
		DELETE FROM Friends
		WHERE AccHolder = Friend2
		AND Friend = Friend1;
	END IF;
	/* Create new friendship for any accepted friend requests */
	IF (SELECT Response FROM FriendRequest WHERE RequestID = reqID) = 'Accepted'
	THEN
		INSERT INTO Friends(AccHolder,Friend)
		VALUES (Friend1,Friend2);
		INSERT INTO Friends(AccHolder,Friend)
		VALUES (Friend2,Friend1);
	END IF;
	/* Change response status to complete for all actioned requests */
	IF (SELECT Response FROM FriendRequest WHERE RequestID = reqID) <> 'Pending'
	THEN
		UPDATE FriendRequest
		SET Response = 'Completed'
		WHERE RequestID = reqID;
	END IF;
END; $$
DELIMITER ;

/*
QUESTION 11
Author: Will Woodhead
*/
DROP PROCEDURE IF EXISTS GetFriendsLeaderboard;
DELIMITER $$
CREATE PROCEDURE GetFriendsLeaderboard(UserN VARCHAR(30), GID INT)
BEGIN

	SET @ScoreFormat = (SELECT ScoreFormat FROM Game WHERE GameID = GID);

	DROP TABLE IF EXISTS temp;
	CREATE TABLE temp (
		Username VARCHAR(30),
		Score INT ,
		TimeOfScore TIMESTAMP
	);

	INSERT INTO temp
	SELECT Username, Score, TimeOfScore
	FROM Scores, UserToGame
	WHERE Scores.UserToGameID = UserToGame.ID
	AND UserToGame.GameID = GID
	AND Scores.UserToGameID IN (
		SELECT ID
		FROM UserToGame
		WHERE UserName IN (
			SELECT Friend
			FROM Friends
			AS friendtemp
			WHERE AccHolder = UserN
		 UNION SELECT UserN
		)
	);

	IF ((SELECT SortOrder FROM Game WHERE GameID=GID) = 'asc')
	THEN
```

```
                SELECT  Username, Score, CONCAT(' ', @ScoreFormat) AS units
                FROM temp
                ORDER BY Score ASC;
        ELSE
                SELECT  Username, Score, CONCAT(' ', @ScoreFormat) AS units
                FROM temp
                ORDER BY Score DESC;
        END IF;
        DROP TABLE temp;
END; $$
DELIMITER ;

/*
QUESTION 12
Author: Alex Parrott
*/
DROP PROCEDURE IF EXISTS ShowFriends;
DELIMITER $$
CREATE PROCEDURE ShowFriends(IN User VARCHAR(20))
BEGIN
        /* Create a table of all specified user's friends */
        CREATE TABLE AllFriends(
                SELECT Friend
                FROM Friends
                WHERE AccHolder = User
        );
        /* Create a table of last games played by each friend */
        /* (1) Get the date of the last play */
        CREATE TABLE LastDate(
                SELECT UserName,MAX(LastPlayDate) AS LastPlay
                FROM UserToGame
                GROUP BY UserName
                ORDER BY LastPlayDate DESC
        );
        /* (2) Get the unique ID and name of the game played on this date */
        CREATE TABLE LastGame(
                SELECT UserToGame.UserName,Game.GameID,Name
                FROM UserToGame
                JOIN LastDate ON LastDate.UserName = UserToGame.UserName
                JOIN Game ON UserToGame.GameID = Game.GameID
                WHERE LastPlay = LastPlayDate
        );

        /* Display list of all online friends */
        SELECT UserName,AccountStatus
        FROM UserPublic,AllFriends
        WHERE UserPublic.UserName = AllFriends.Friend
        AND AccountStatus = 'Online';

        /* Display list of offline friends with last login and last game played */
        SELECT UserPublic.UserName,AccountStatus,LastLogin,Name AS LastPlayed
        FROM UserPublic,AllFriends,LastGame
        WHERE UserPublic.UserName = AllFriends.Friend
        AND UserPublic.UserName = LastGame.UserName
        AND AccountStatus = 'Offline';

        DROP TABLE AllFriends;
        DROP TABLE LastGame;
        DROP TABLE LastDate;

END; $$
DELIMITER ;

/*
QUESTION 13
Author: James Hamblion
*/
DROP PROCEDURE IF EXISTS AchievementsForUserGame;
DELIMITER $$
CREATE PROCEDURE AchievementsForUserGame(usrname VARCHAR(50),gameident INT)
BEGIN
        DECLARE totalAchiev INT;
        DECLARE userGameid INT;
        DECLARE earntAchiev INT;
        DECLARE pointVal INT;

        /* Get userToGameID for the game and username. If user doesn't own game variable is null. */
        SET userGameid = (
                SELECT ID
                FROM UserToGame utg
                WHERE utg.UserName = usrname
                AND utg.gameid = gameident
        );
        /* Check userGameid not null and continue, else return message 'Error: game not owned by
user!' */
        IF (userGameid IS NOT NULL)
        THEN
```

```
		/* Get total achievement number for game */
		SET totalAchiev = (
			SELECT COUNT(achievementID)
			FROM Achievement a
			WHERE a.gameid = gameident
		);
		/* Get earnt achievements for the user in the specified game */
		SET earntAchiev = (
			SELECT COUNT(achievementID)
			FROM AchievementToUserToGame a
			WHERE a.userToGameID = userGameid
		);
		IF (earntAchiev IS NULL)
		THEN
			SET earntAchiev = 0;
		END IF; /* prevents null output */
		/* Get point value of earnt achievements for the user in the specified game */
		SET pointVal = (
			SELECT SUM(PointValue)
			FROM Achievement a, AchievementToUserToGame b
			WHERE b.userToGameid = userGameid
			AND a.achievementID = b.achievementID
			AND a.gameid = gameident
		);
		IF (pointVal IS NULL)
		THEN
			SET pointVal = 0;
		END IF; /*prevents null output*/
		SELECT CONCAT(earntAchiev,' of ',totalAchiev,' achievements ','(',pointVal,' points',
')') AS 'Your_Achievements';
	ELSE
		SELECT 'Error: game not owned by user!' AS 'Your_Achievements';
	END IF;
END; $$
DELIMITER ;

/*
QUESTION 14
Author: James Hamblion
*/
DROP PROCEDURE IF EXISTS ShowStatusScreen;
DELIMITER $$
CREATE PROCEDURE ShowStatusScreen(usrname VARCHAR(50))
BEGIN
	/* User not found handler (skips query code if no usrname exists in database) */
	IF (
		(SELECT COUNT(UserName)
		FROM UserPublic u
		WHERE u.UserName = usrname)
		!= 0)
	THEN
		/* User status line */
		SET @status = (
			SELECT UserStatus
			FROM UserPublic u
			WHERE u.UserName = usrname
		);
		/* Number of games owned by user */
		SET @numGames = (
			SELECT COUNT(userName)
			FROM UserToGame u
			WHERE u.UserName = usrname
		);
		IF (@numGames IS NULL)
		THEN
			SET @numGames = 0;
		END IF; /* prevents null output */
		/* Total achievement points for user */
		SET @numPoints = (
			SELECT SUM(PointValue)
			FROM Achievement a,AchievementToUserToGame b
			WHERE b.userToGameid IN (
				SELECT ID
				FROM UserToGame u
				WHERE u.UserName = usrname)
			AND a.achievementID = b.achievementID
		);
		IF (@numPoints IS NULL)
		THEN
			SET @numPoints = 0;
		END IF; /* prevents null output */
		/* Number of friends of user */
		SET @numFriends = (
			SELECT COUNT(Friend) FROM Friends f
			WHERE f.AccHolder = usrname
		);
		IF (@numFriends IS NULL)
```

```
		THEN
			SET @numFriends = 0;
		END IF; /* prevents null output */

		/*
		Create status screen table, insert values, print and then drop
		the the Status_Screen table
		*/
		CREATE TABLE Status_Screen (
			Username VARCHAR(20),
			Status_Line VARCHAR(100),
			Number_of_Games_Owned INT,
			Total_Number_of_Achievement_Points INT,
			Number_of_Friends INT
		);
		INSERT INTO Status_Screen
		VALUES (usrname,@status,@numGames,@numPoints,@numFriends);
		SELECT * FROM Status_Screen;
		DROP TABLE Status_Screen;
	END IF;
END; $$
DELIMITER ;

/*
QUESTION 15
Author: James Hamblion
*/
DROP PROCEDURE IF EXISTS ListUserGameAchievements;
DELIMITER $$
CREATE PROCEDURE ListUserGameAchievements(usrname VARCHAR(50),gameident INT)
BEGIN
	DECLARE done INT DEFAULT FALSE;
	/* All var needed to fetch row fields */
	DECLARE achID INT;
	DECLARE gmid INT;
	DECLARE ttl VARCHAR(50);
	DECLARE hidFlag BIT;
	DECLARE icn INT;
	DECLARE pointVal INT;
	DECLARE poDes VARCHAR(200);
	DECLARE preDes VARCHAR(200);
	DECLARE a INT;
	DECLARE u INT;
	DECLARE dgain DATE;
	/* Var to determine post or pre description */
	DECLARE descrpt VARCHAR(200);
	/*The query to give everything needed: */
	DECLARE cur CURSOR FOR
		SELECT * FROM
			(SELECT *
			FROM Achievement x
			WHERE x.gameid = gameident) a
			LEFT OUTER JOIN
			(SELECT *
			FROM AchievementToUserToGame y
			 WHERE y.userToGameid IN
				(SELECT ID
				FROM UserToGame u
				WHERE u.UserName = usrname
				AND u.gameID = gameident)) b
			ON a.achievementID = b.achievementID
		WHERE (b.dateGained IS NULL
		AND a.hiddenFlag = FALSE)
		OR(a.hiddenFlag = FALSE)
		OR(b.dateGained IS NOT NULL AND a.hiddenFlag = TRUE)
		ORDER BY b.dateGained DESC;
	DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

	/* Create result table */
	CREATE TABLE GameAchievementList (
		Title VARCHAR(50),
		PointValue INT,
		Description VARCHAR(200),
		DateGained DATE
	);

	/* Populate the table by looping row by row using a cursor. */
	OPEN cur;
	pop_loop: LOOP
		FETCH cur INTO achID, gmid, ttl, hidFlag, icn, pointVal, poDes, preDes, a, u, dgain;
		IF done
		THEN
			LEAVE pop_loop; /* If user doesn't exist or game not owned procedure ends/does
nothing. */
		END IF;
		/* Check description to use. Earnt achievement = postDescription */
		IF (dgain IS NULL)
```

```
                THEN
                        SET descrpt = preDes;
                ELSE
                        SET descrpt = poDes;
                END IF;

                INSERT INTO GameAchievementList
                VALUES (ttl, pointVal, descrpt, dgain);
        END LOOP;
        CLOSE cur;

        /* Display results and then drop the table as no longer needed. */
        SELECT * FROM GameAchievementList;
        DROP TABLE GameAchievementList;
END; $$
DELIMITER ;

/*
QUESTION 16
Author: James Hamblion
*/
DROP PROCEDURE IF EXISTS CompListGameAchievFriend;
DELIMITER $$
CREATE PROCEDURE CompListGameAchievFriend(usrname VARCHAR(50), frndusrname VARCHAR(50))
BEGIN
        DECLARE ttl VARCHAR(30);
        DECLARE usrA VARCHAR(20);
        DECLARE usrpointsA VARCHAR(20);
        DECLARE usrB VARCHAR(20);
        DECLARE usrpointsB VARCHAR(20);
        DECLARE done INT DEFAULT FALSE;
        /*Cursor for games and achiev of usrname*/
        DECLARE cur CURSOR FOR
        /*Cursor query start*/
                SELECT query1.GameTitle, User_A, User_A_Points, User_B, User_B_Points
                FROM
                                (SELECT ID, User_A, GameTitle, SUM(PointValue) AS User_A_Points
                                 FROM
                                        (SELECT ID, User_A, GameTitle, achievementID
                                         FROM
                                                (SELECT ID, UserName AS User_A, name As GameTitle
                                                FROM UserToGame u, Game g
                                                WHERE u.UserName = usrname AND g.gameid = u.gameid) ug
                                                 LEFT OUTER JOIN
                                                        AchievementToUserToGame atug
                                                 ON ug.ID = atug.userToGameID) x
                                   LEFT OUTER JOIN
                                        Achievement y
                                 ON x.achievementID = y.achievementID
                                 GROUP BY ID
                                 ORDER BY x.achievementID DESC) query1
                        LEFT OUTER JOIN
                                (SELECT ID, User_B, GameTitle, SUM(PointValue) AS User_B_Points
                                FROM
                                        (SELECT ID, User_B, GameTitle, achievementID
                                         FROM
                                                (SELECT ID, UserName AS User_B, name As GameTitle
                                                FROM UserToGame u, Game g
                                                WHERE u.UserName = frndusrname AND g.gameid = u.gameid)
ug
                                                 LEFT OUTER JOIN
                                                        AchievementToUserToGame atug
                                                 ON ug.ID = atug.userToGameID) x
                                   LEFT OUTER JOIN
                                        Achievement y
                                ON x.achievementID = y.achievementID
                                GROUP BY ID
                                ORDER BY x.achievementID DESC) query2
                        ON query1.GameTitle = query2.GameTitle

                UNION

                SELECT query2.GameTitle, User_A, User_A_Points, User_B, User_B_Points
                FROM
                                (SELECT ID, User_A, GameTitle, SUM(PointValue) AS User_A_Points
                                 FROM
                                        (SELECT ID, User_A, GameTitle, achievementID
                                         FROM
                                                (SELECT ID, UserName AS User_A, name As GameTitle
                                                FROM UserToGame u, Game g
                                                WHERE u.UserName = usrname AND g.gameid = u.gameid) ug
                                                 LEFT OUTER JOIN
                                                        AchievementToUserToGame atug
                                                 ON ug.ID = atug.userToGameID) x
                                   LEFT OUTER JOIN
                                        Achievement y
                                 ON x.achievementID = y.achievementID
```

```
                                GROUP BY ID
                                ORDER BY x.achievementID DESC) query1
                        RIGHT OUTER JOIN
                                (SELECT ID, User_B, GameTitle, SUM(PointValue) AS User_B_Points
                                FROM
                                        (SELECT ID, User_B, GameTitle, achievementID
                                         FROM
                                                (SELECT ID, UserName AS User_B, name As GameTitle
                                                FROM UserToGame u, Game g
                                                WHERE u.UserName = frndusrname AND g.gameid = u.gameid)
ug
                                         LEFT OUTER JOIN
                                                AchievementToUserToGame atug
                                         ON ug.ID = atug.userToGameID) x
                                  LEFT OUTER JOIN
                                        Achievement y
                                ON x.achievementID = y.achievementID
                                GROUP BY ID
                                ORDER BY x.achievementID DESC) query2
                        ON query1.GameTitle = query2.GameTitle;
        /*Cursor query end*/
        DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

        /*Build statement for the temporary create comparison table screen
        with procedure input variables in column/attribute names*/
        SET @userAchPoints = CONCAT('CREATE TABLE Compare_List (Game_Title VARCHAR(30), ',
                                        'Your_Achievement_Points VARCHAR(20), ',
                                        'Achievement_Points_of_', frndusrname, ' VARCHAR(20), notOwned
INT)');
        /*Create user/friend game and achievements comparison table*/
        PREPARE stmnt FROM @userAchPoints;
        EXECUTE stmnt;
        DEALLOCATE PREPARE stmnt;
        /*Populate Compare_Screen temporary output table:*/
        OPEN cur;
        pop_loop: LOOP
                FETCH cur INTO ttl, usrA, usrpointsA, usrB, usrpointsB;
                IF done THEN
                        LEAVE pop_loop;
                END IF;
                /*Check if game owned by both users*/
                IF ((usrA IS NOT NULL) AND (usrB IS NOT NULL)) THEN
                        SET @owned = 1; /*Set flag for sort order (1 = owned by both users)*/
                        /*Check if a user has null points for an owned game and set to 0*/
                        IF (usrpointsA IS NULL) THEN
                                SET usrpointsA = '0';
                        ELSEIF (usrpointsB IS NULL) THEN
                                SET usrpointsB = '0';
                        END IF;
                ELSE /*only owned by one user*/
                        SET @owned = 0; /*Set flag for sort order (0 = not owned by one user)*/
                        IF (usrA IS NULL AND usrB IS NOT NULL) THEN /*owned by usrB not usrA*/
                                SET usrpointsA = '';
                                SET usrpointsB = '0';
                        ELSEIF (usrA IS NOT NULL AND usrB IS NULL) THEN /*owned by usrA not usrB*/
                                SET usrpointsA = '0';
                                SET usrpointsB = '';
                        END IF;
                END IF;
                /*Insert results into temporary output table*/
                INSERT INTO Compare_List
                        VALUES (ttl, usrpointsA, usrpointsB, @owned);
        END LOOP;
        CLOSE cur;
        /*Display result query string*/
        SET @resultStr = CONCAT('SELECT Game_Title, Your_Achievement_Points,
Achievement_Points_of_',
                                        frndusrname, ' FROM Compare_List ORDER BY notOwned
DESC');
        PREPARE stmnt FROM @resultStr;
        EXECUTE stmnt;
        DEALLOCATE PREPARE stmnt;
        DROP TABLE Compare_List;
END; $$
DELIMITER ;


/*
QUESTION 17:
Author: Will Woodhead

*/
DROP PROCEDURE IF EXISTS GetLeaderboard;
DELIMITER $$
CREATE PROCEDURE GetLeaderboard(LBID INT)
BEGIN
```

```
	SET @ScoreFormat = (
		SELECT ScoreFormat
		FROM Game
		WHERE GameID = (
			SELECT GameID
			FROM Leaderboard
			WHERE LeaderboardID = LBID)
	);
	SET @GID = (
		SELECT GameID
		FROM Leaderboard
		WHERE LeaderboardID = LBID
	);

	DROP TABLE if exists temp;
	CREATE TABLE temp (
		Username VARCHAR(30),
		Score INT,
		TimeOfScore TIMESTAMP
	);

	IF ((SELECT TimePeriod FROM Leaderboard WHERE LeaderboardID=LBID) = '1_year')
	THEN
		INSERT INTO temp
			SELECT Username, Score, TimeOfScore
			FROM Scores, UserToGame
			WHERE Scores.UserToGameID = UserToGame.ID
			AND UserToGame.GameID = @GID
			AND TimeOfScore > DATE(DATE_SUB(NOW(), INTERVAL 365 DAY)
		);
	ELSEIF ((SELECT TimePeriod FROM Leaderboard WHERE LeaderboardID=LBID) = '1_week')
	THEN
		INSERT INTO temp
			SELECT Username, Score, TimeOfScore
			FROM Scores, UserToGame
			WHERE Scores.UserToGameID = UserToGame.ID
			AND UserToGame.GameID = @GID
			AND TimeOfScore > DATE(DATE_SUB(NOW(), INTERVAL 7 DAY)
		);
	ELSEIF ((SELECT TimePeriod FROM Leaderboard WHERE LeaderboardID=LBID) = '1_day')
	THEN
		INSERT INTO temp
			SELECT Username, Score, TimeOfScore
			FROM Scores, UserToGame
			WHERE Scores.UserToGameID = UserToGame.ID
			AND UserToGame.GameID = @GID
			AND  TimeOfScore > DATE(DATE_SUB(NOW(), INTERVAL 1 DAY)
		);
	ELSE
		INSERT INTO temp
			SELECT Username, Score, TimeOfScore
			FROM Scores, UserToGame
			WHERE Scores.UserToGameID = UserToGame.ID
			AND UserToGame.GameID = @GID;
	END IF;

	IF ((SELECT SortOrder FROM Leaderboard WHERE LeaderboardID=LBID) = 'asc')
	THEN
		SELECT  Username, Score, CONCAT(' ', @ScoreFormat) AS Units, TimeOfScore
		FROM temp
		ORDER BY Score ASC;
	ELSE
		SELECT  Username, Score, CONCAT(' ', @ScoreFormat) AS Units, TimeOfScore
		FROM temp
		ORDER BY Score DESC;
	END IF;

	DROP TABLE temp;
END; $$
DELIMITER ;

/*
QUESTION 18:
Author: Alex Parrott
*/
DROP PROCEDURE IF EXISTS SuggestFriends;
DELIMITER $$
CREATE PROCEDURE SuggestFriends(IN User VARCHAR(20))
BEGIN
	DECLARE toCheck VARCHAR(20);
	DECLARE done INT DEFAULT FALSE;
	DECLARE cur CURSOR FOR
		SELECT UserName FROM SuggestedFriends;
	DECLARE CONTINUE HANDLER FOR
		NOT FOUND SET done = TRUE;

	/* Create a table of all specified user's friends... */
```

```
	CREATE TABLE AllFriends(
		SELECT Friend
		FROM Friends
		WHERE AccHolder = User
	);
	/* ...and one of all of their games */
	CREATE TABLE AllGames(
		SELECT GameID
		FROM UserToGame
		WHERE UserName = User
	);
	/* Create a table to hold non friends to make suggestions */
	CREATE TABLE SuggestedFriends(
		SELECT UserName
		FROM UserPrivate
		WHERE UserName <> User
		AND UserName NOT IN
			(SELECT Friend AS UserName
			FROM AllFriends)
	);
	ALTER TABLE SuggestedFriends
	ADD FriendsInCommon INT;
	ALTER TABLE SuggestedFriends
	ADD GamesInCommon INT;

	/* Loop cycles through all users who are not already friends with provided user */
	OPEN cur;
	getFriends: LOOP

		FETCH cur INTO toCheck;
		IF done = TRUE
		THEN
			LEAVE getFriends;
		END IF;

		/* Create a table of the current user's friends */
		CREATE TABLE CompareFriend(
			SELECT Friend
			FROM Friends
			WHERE AccHolder = toCheck
		);
		/* Count the friends in common with specified user */
		UPDATE SuggestedFriends
		SET FriendsInCommon =(
			SELECT COUNT(CompareFriend.Friend)
			FROM CompareFriend,AllFriends
			WHERE CompareFriend.Friend = AllFriends.Friend
		)
		WHERE UserName = toCheck;

		/* Create a table of the current user's games */
		CREATE TABLE CompareGame(
			SELECT GameID
			FROM UserToGame
			WHERE UserName = toCheck
		);
		/* Count the games in common with specified user */
		UPDATE SuggestedFriends
		SET GamesInCommon =(
			SELECT COUNT(CompareGame.GameID)
			FROM CompareGame,AllGames
			WHERE CompareGame.GameID = AllGames.GameID
		)
		WHERE UserName = toCheck;
		/* Drop tables ready for next iteration of the loop */
		DROP TABLE CompareFriend;
		DROP TABLE CompareGame;


	END LOOP getFriends;
	CLOSE cur;

	/* Display any users with more than one friend or game in common */
	SELECT * FROM SuggestedFriends
	WHERE FriendsInCommon > 1
	OR GamesInCommon > 1;

	DROP TABLE AllFriends;
	DROP TABLE AllGames;
	DROP TABLE SuggestedFriends;
END; $$
DELIMITER ;


/* Question 20
Author: Will Woodhead
*/
```

```
DROP PROCEDURE if exists CreateMatch;
DELIMITER //
CREATE PROCEDURE CreateMatch(UTGID INT, minPlayer INT, maxPlayer INT, matchnm VARCHAR(30))
BEGIN

INSERT INTO Matches (Initiator, MinPlayers, MaxPlayers, MatchName)
VALUES (UTGID, minPlayer, maxPlayer, matchnm);

INSERT INTO MatchToUserToGame (MatchID, UserToGameID)
VALUES (
       (SELECT MatchID FROM Matches WHERE Initiator=UTGID AND MatchName=matchnm),
       UTGID
       );
END; //
DELIMITER ;


DROP PROCEDURE if exists MatchRequesting;
DELIMITER //
CREATE PROCEDURE MatchRequesting(Sending INT, Receiving INT, mID INT)
BEGIN
       INSERT INTO MatchRequest (SendingUTG, ReceivingUTG, MatchID)
       VALUES (Sending, Receiving, mID);
END; //
DELIMITER ;

/* TRIGGERS */

/* Triggers for Game relation  */
DELIMITER $$
CREATE TRIGGER Game_after_insert
AFTER INSERT ON Game
FOR EACH ROW
BEGIN
       /* Create a default leaderboard at the creation of any new game */
       INSERT INTO Leaderboard (GameID, IsDefault, SortOrder)
       VALUES (
              (SELECT GameID
              FROM Game
              WHERE Game.GameID = NEW.GameID), 1, (
                     SELECT SortOrder
                     FROM Game
                     WHERE Game.GameID = NEW.GameID)
       );
       INSERT INTO Leaderboard (GameID, SortOrder, TimePeriod)
              VALUES (
                     NEW.GameID,
                     (SELECT SortOrder FROM Game WHERE GameID = NEW.GameID),
                     '1_week'
              );
              INSERT INTO Leaderboard (GameID, SortOrder, TimePeriod)
              VALUES (
                     NEW.GameID,
                     (SELECT SortOrder FROM Game WHERE GameID = NEW.GameID),
                     '1_day'
              );
END $$
DELIMITER ;

/* TRIGGERS FOR UserToGame RELATION */

/* BEFORE INSERT on UserToGame */
DROP TRIGGER IF EXISTS BeforeInsertUserToGame;
DELIMITER $$
CREATE TRIGGER BeforeInsertUserToGame
BEFORE INSERT ON UserToGame
FOR EACH ROW
BEGIN
       /* QUESTION 6 */
       SET NEW.LastScore = (
              SELECT CatchCheaters(NEW.GameID, NEW.LastScore));
END $$
DELIMITER ;

/* BEFORE UPDATE on UserToGame */
DROP TRIGGER IF EXISTS BeforeUpdateUserToGame;
DELIMITER $$
CREATE TRIGGER BeforeUpdateUserToGame
BEFORE UPDATE ON UserToGame
FOR EACH ROW
BEGIN
       /* QUESTION 6 */
       SET NEW.LastScore = (
              SELECT CatchCheaters(NEW.GameID, NEW.LastScore));

END $$
```

```
DELIMITER ;

/* AFTER INSERT on UserToGame */
DELIMITER $$
CREATE TRIGGER AfterInsertUserToGame
AFTER INSERT ON UserToGame
FOR EACH ROW
BEGIN
	/* QUESTION 2 & QUESTION 3 */
	CALL UpdateAverage(NEW.GameID);
END $$
DELIMITER ;

/* AFTER UPDATE on UserToGame */
DELIMITER $$
CREATE TRIGGER AfterUpdateUserToGame
AFTER UPDATE ON UserToGame
FOR EACH ROW
BEGIN
	/* QUESTION 2 & QUESTION 3 */
	CALL UpdateAverage(NEW.GameID);

	/* QUESTION 9: When a user starts playing a game, this 'play' is logged in the PLays table
	This plays table is queried to find out the Hotlist */
	IF NEW.GameInProgress = 'yes'
	AND OLD.GameInProgress = 'no'
	THEN BEGIN
		INSERT INTO Plays (GameID,UserName,TimeOfPlay)
		VALUES (
			(SELECT GameID
			FROM UserToGame
			WHERE UserToGame.GameID = NEW.GameID
			AND UserToGame.UserName = NEW.UserName),
			(SELECT UserName
			FROM UserToGame
			WHERE UserToGame.GameID = NEW.GameID
			AND UserToGame.UserName = NEW.UserName),
			NOW()
		);
		END;
	END IF;

	/* QUESTION 7: When a user gets a new score in any game, it is recorded in the lastScore
attribute in UserToGame table
	This score is also logged in LeaderboardToUserToGame. THis table holds the record of every
score on every game
	at a certain time. This table can therefore be used to create all of the leaderboards for
any game.*/
	IF NEW.LastScore != OLD.LastScore
	THEN BEGIN
		INSERT INTO Scores (UserToGameID, Score, TimeOfScore)
		VALUES(
			(SELECT ID
			FROM UserToGame
			WHERE UserToGame.ID = NEW.ID),
			(SELECT LastScore
			FROM UserToGame
			WHERE UserToGame.ID = NEW.ID),
			NOW()
		);
		END;
	END IF;
END $$
DELIMITER ;

/* AFTER DELETE on UserToGame */
DELIMITER $$
CREATE TRIGGER AfterDeleteUserToGame
AFTER DELETE ON UserToGame
FOR EACH ROW
BEGIN
	/* QUESTION 2 & QUESTION 3 */
	CALL UpdateAverage(OLD.GameID);

	/* QUESTION 3: If number of ratings drops below 10, set average to NULL */
	IF (SELECT NoOfRatings FROM Game WHERE GameID = OLD.GameID) < 10
	THEN BEGIN
		UPDATE Game
			SET AverageRating = NULL
			WHERE Game.GameID = OLD.GameID;
	END; END IF;
END $$
DELIMITER ;

/* TRIGGERS FOR MatchRequest RELATION */

/* AFTER UPDATE on MatchRequest */
```

```
DELIMITER $$
CREATE TRIGGER matchRequest_after_update
AFTER UPDATE ON MatchRequest
FOR EACH ROW
BEGIN
	/* if response is accepted, then the usertogame is added to the match*/
	SET @num = (SELECT NoOfPlayer FROM Matches WHERE Matches.MatchID = NEW.MatchID);
	IF NEW.Response = 'Accepted'
	THEN BEGIN
		INSERT INTO MatchToUserToGame (MatchID, UserToGameID)
		VALUES(NEW.MatchID, NEW.ReceivingUTG);
		UPDATE Matches
		SET NoOfPlayer = @num + 1
		WHERE MatchID = New.MatchID;
	END; END IF;

END $$
DELIMITER ;

/* AFTER UPDATE on UserTogametomatch */
DELIMITER $$
CREATE TRIGGER matchtousertogame_after_update
AFTER UPDATE ON MatchToUserToGame
FOR EACH ROW
BEGIN

	SET @num = (SELECT NoOfPlayer FROM Matches WHERE Matches.MatchID = NEW.MatchID);
	IF NEW.PlayerStatus = 'Quit'
	THEN BEGIN
		UPDATE Matches
		SET NoOfPlayer =  @num - 1
		WHERE MatchID = NEW.MatchID;
	END; END IF;

END $$
DELIMITER ;

/* Setup the TRIGGER on User-Public table*/
DROP TRIGGER IF EXISTS userNameEntryCheck;
DELIMITER $$
CREATE TRIGGER userNameEntryCheck
BEFORE INSERT ON UserPublic
FOR EACH ROW
BEGIN
	SET @usrname = NEW.userName;
	SET @obscene = isUserNameRude(@usrname);
	IF @obscene THEN
		SET NEW.AccountStatus := 'Locked';
	END IF;
END; $$
DELIMITER ;
```