**ONLINE MULTIPLAYER GAME DATABASE**

A PROJECT REPORT

*Submitted by*

BL.EN.U4ECE17008 ALAN ARIMPUR RAJU

BL.EN.U4ECE17015 ASWIN A

BL.EN.U4ECE17019 SARAGADAM ROSHNI

BL.EN.U4ECE17045 G ASHWIN

***for the course***

***19CSE377- Foundation of Information Technology***

***Guided and Evaluated by***

***Ms. D. Radha***

***Asst. Prof (SG),***

***Dept. of CSE,***



**AMRITA SCHOOL OF ENGINEERING, BANGALORE**

**AMRITA VISHWA VIDHYAPEETHAM**

**BANGALORE-560 035**

**February-2021**

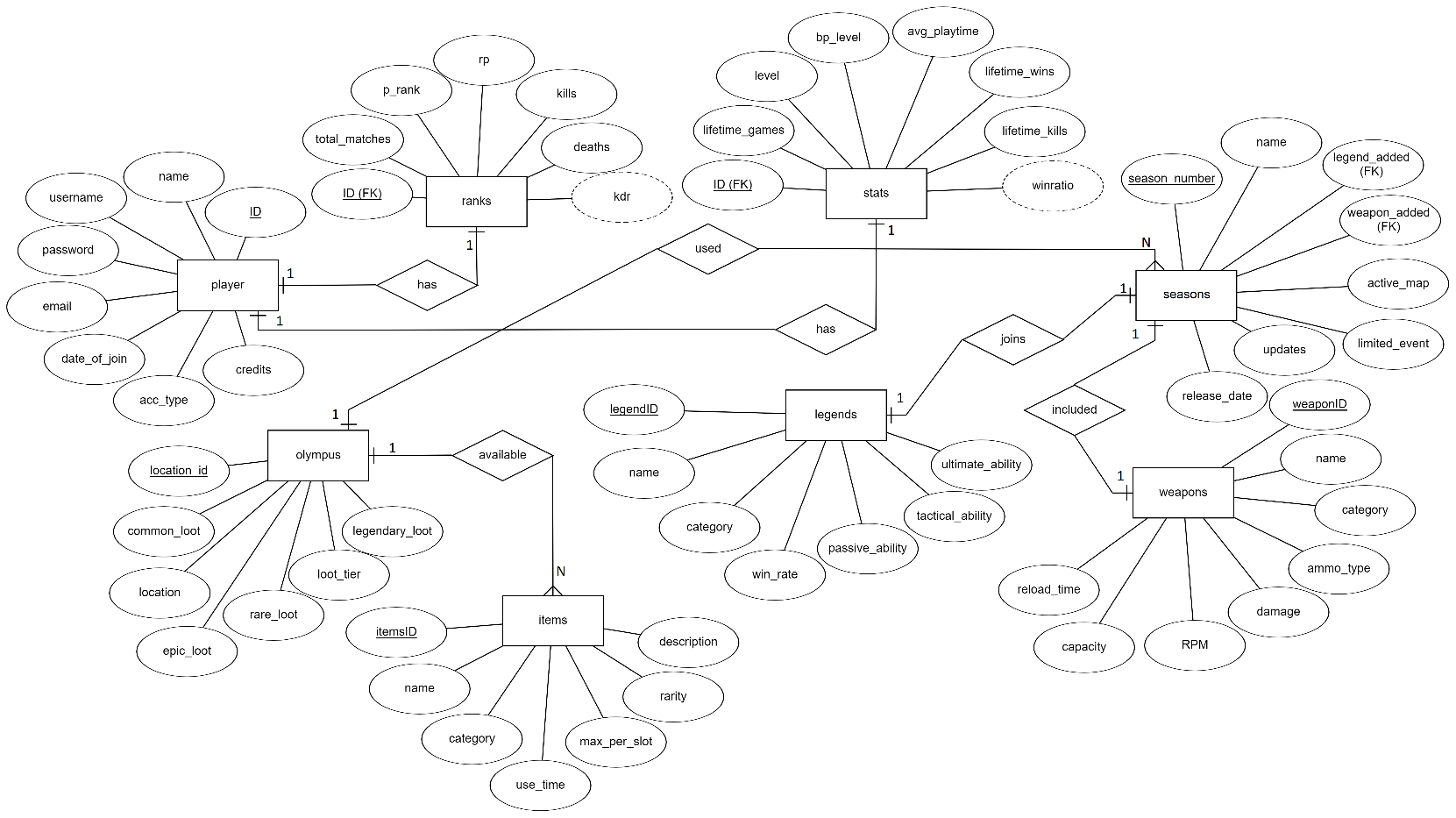
**ONLINE MULTIPLAYER GAME DATABASE**

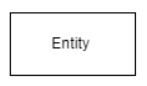
ABSTRACT

A database management system (DBMS) is a software package designed to define, manipulate, retrieve and manage data in a database. A DBMS generally manipulates the data itself, the data format, field names, record structure and file structure. It also defines rules to validate and manipulate this data.

In this project we have designed a database for an online multiplayer game. The game goes by the name as Apex Legends. Typically, game assets have some unique id, these can be the characters, models, players, levels and so on. The use of a database can enable the game to refer the tables within the database to function in a stable manner. It brings flexibility to data management and analysis and allows units to respond calmly in a rapidly changing environment, so as to gain a competitive advantage. This enables us to analyse the data for making the game balanced and better in the future. In the database that we have made, the player data shares relations with the stats and the rank. This will contain all the data that is released out of the player during the course of his gameplay over time. Another part of the database contains the data required for the game to keep track of all the assets that are being used in the game. On observing the ER diagram below, we can see that we have made entities related to weapons, legends, locations, seasons which are have relations within each of them in some way or the other. On programming the database along with the tables and added data into the tables in the form of rows for the respective attributes defined by the table. Once the database was implemented, we executed complex SQL queries that pull out useful information such as analysing the strongest weapons, best players, to find recommended locations or legends for the newer player that join the game.

ER DIAGRAM





Entities ->

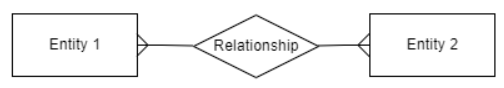


Attributes ->

One-One Relationship ->



One-Many Relationship ->

Many-Many Relationship ->

TABLES AND ATTRIBUTES

The tables used for the online multiplayer game database is displayed below in Figure1. The command used here was “**show tables;”** This command can be used to display all the tables that are present in the database that is named as “**Apex**” as the name of the game is Apex Legends in this scenario.

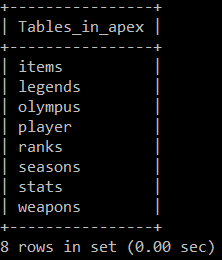


Figure 1: List of every table used in “**Apex**” database

TABLE 1: ITEMS

The “**items**” table describes all the information related to all the items that is present in the game. The description of the attributes in the table is retrieved using the “**desc items;**” command.

This includes data like the name of the item that are present in the game on observing, we can see that the key is set to unique which means that there can be only one item in the game with a particular name and repeated names are not allowed according to this table. There is also a other fields like category; that tells us what type of item it is , use\_time; the amount of time it takes to use an item, max \_per\_slot; every player in the game can carry a bag with a certain number of slots in them so this fields tells that amount of space an item takes when in a bag, rarity defines the ease of finding the particular item in the map and a description field is added so that anyone else that is working with the database can understand or have an idea what an item does when used or carried. Also note that each legend comes with a “**itemsID**” which is set to primary key so that it can be referenced by other tables.

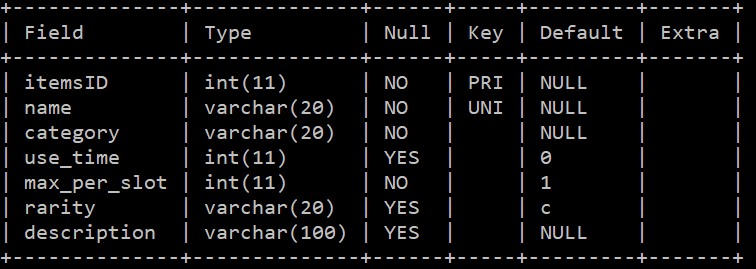


Figure 2: Description of “items” table

TABLE 2: LEGENDS

The next table is the “**legends**” table that tells contains all sorts of data that are related to the legends in the game. These legends are the playable characters that the players can select when entering into a game. Each legends are completely independent of each other; which contains names with the keys unique and not null to avoid any later issues with the database when adding new legends. The other information’s are related to the category; talks about how the legends can be mainly used, win rate attribute is added as a means of making a legend balance so that there is a balance between; to prevent overpowered chanteclers in the game. The rest of the attributes are related to description of the abilities that a legend has. In this game each legend has strictly 3 abilities that goes with names passive, tactical and ultimate. Using the “**desc legends;**” command we can see the table description.

Also note that each legend comes with a “**legendID**” which is set to primary key so that it can be referenced by other tables;

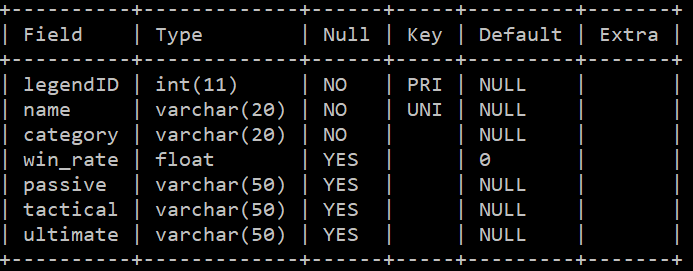


Figure 3: Description of “**legends**” table

TABLE 3: OLYMPUS

As every game has a map to play on, we also made a table with all the map data. Here we named the table “**olympus**” as this is the name of the map we have given and we only plan to use a single map during the whole run of the game hence a single table for map. Using the command “**desc olympus;**” we can see there are 7 attributes. Starting with the “**location\_id**” which gives the other tables an opportunity to refer this data as it is set to primary key, “location” that contains the name of landmarks that are present in the map, the rest of the attributes tells the statistics of the number of loot and its type based on the location. These set of data can be analyzed to make the map more balanced for the players; to prevent players from going to a specific location all the time.

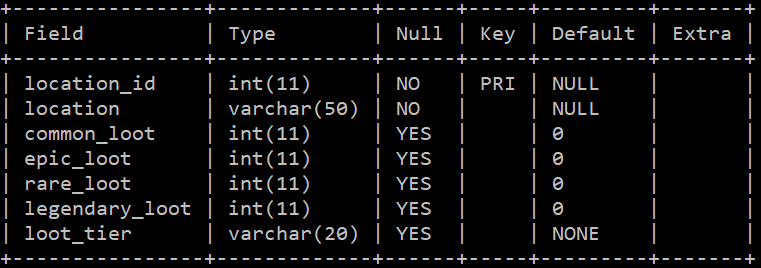


Figure 4: Description of “**olympus**” table

TABLE 4: PLAYER

The next table is the “**player**” table, the description output is achieved by using the “**desc player;**” command. Here, every player is given a unique ID that is sent to primary key, the real “name” of the player, the “username” that the player decides to use; this field is also set to unique as multiple same user names can cause unwanted data retrieving uses later in the future as the database grows bigger when more players join. This table also contains other information like the “**email**”, “**date\_of\_join**”, “**acc\_type**” which tells if the player is a FREE or PREMIUM player and the “credits” that represent the in-game currency that a player has.

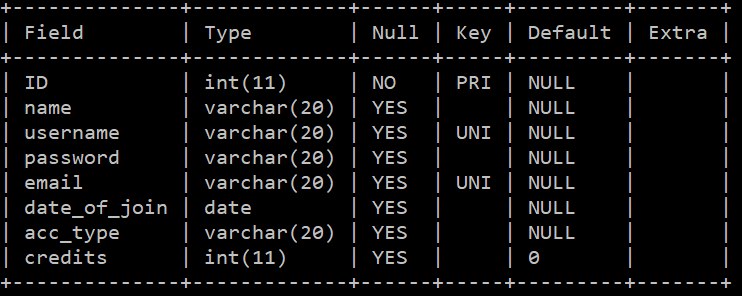


Figure 5: Description of “**player**” table

TABLE 5: RANKS

This table contains the rank related data on the players that play ranked in the game. The table is called “**ranks**” and the table attribute data is seen by using the “**desc ranks;**” this specific table is intended on the more official ranked gameplay which is not considered as a casual game by the players. The purpose of this table is to see the which players are eligible for rewards. In this case a player that reaches of rank PREADATOR gets the maximum rewards as it is the hardest and the highest achievable rank in the game. Also, not that here the player\_ID is both a primary key and a foreign key which enables us to refer this tables as well as this table refereeing another.

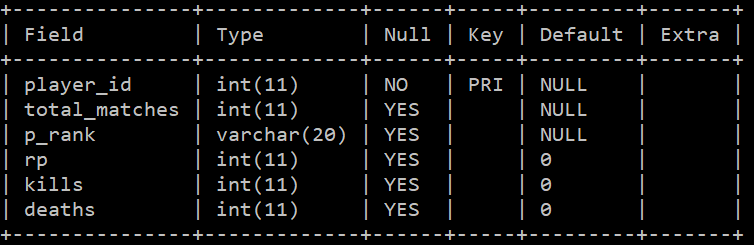


Figure 6: Description of “**rank**” table

TABLE 6: SEASONS

The purpose of this table “**seasons**” is to have all the data related to the season. The table description is achieved using “**desc seasons;**” here the season number is the primary key and this can be used for the other table to refer. Also, if we observe we can see that the legend added and the weapons added is set as “**MUL**” this means that those particular attributes are foreign keys that refer to the table “legends” and “**weapons**”. It also contains data like the number of updates that was pushed during each season to have an idea on the performance of the staff every season. Here, ever season is like a refresher with new game mores along with legends and weapons so that the players will feel refreshed to keep playing the game.

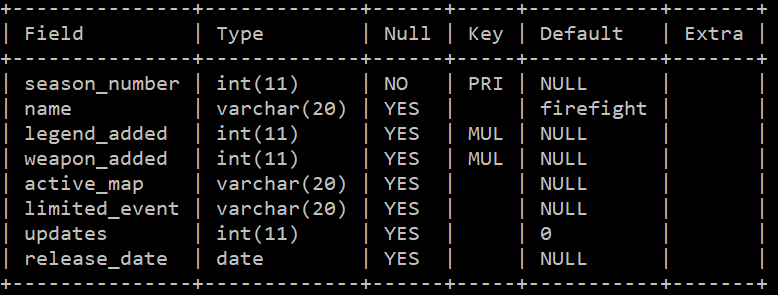


Figure 7: Description of “**seasons**” table

TABLE 7: STATS

This table is called “**stats**” which contains all the player in-game information. The table description is known using “**desc stats;**” command and not to be confused with the above table that is related to ranks. Here, the table is related to the casual gaming information of the players. This type of data is for the players to evaluate their statistics. We also use this table to signify if a player is banned or not, this will be shown in the queries section of the report. Here the player\_id is a primary key as well as a foreign key so that it can be used as references to the other table as well as refer to the player table to retrieve player data.

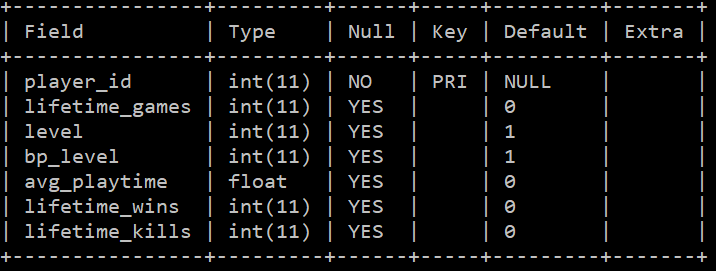


Figure 8: Description of “**stats**” table

TABLE 8: WEAPONS

This table is called the “**weapons**” table that contains the information of all the data related to the weapons in the game. It contains information like “**weaponID**” which is a primary key that can be used by other tables as a reference, “**name**”, “**category**” which tells what class gun is it to suit the players playstyle, ”**ammo\_type**” and the rest of the attributes are related to statistics of the particular gun as a means of balancing the weapon meta in the game; to prevent any sort of unfair gameplay. The description of the table can be seen by using the “**desc weapons;**” command.

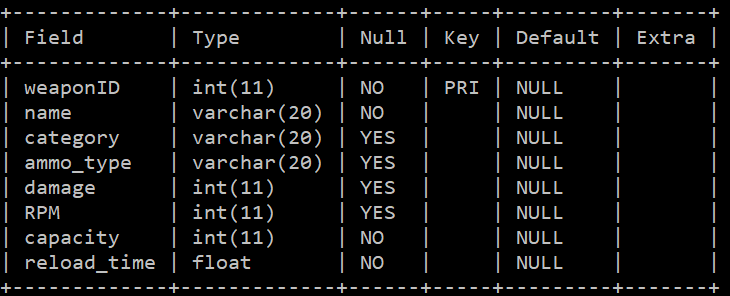


Figure 9: Description of “**weapons**” table

ROWS OF ALL TABLES

TABLE 1: ITEMS

SELECT \* FROM items;

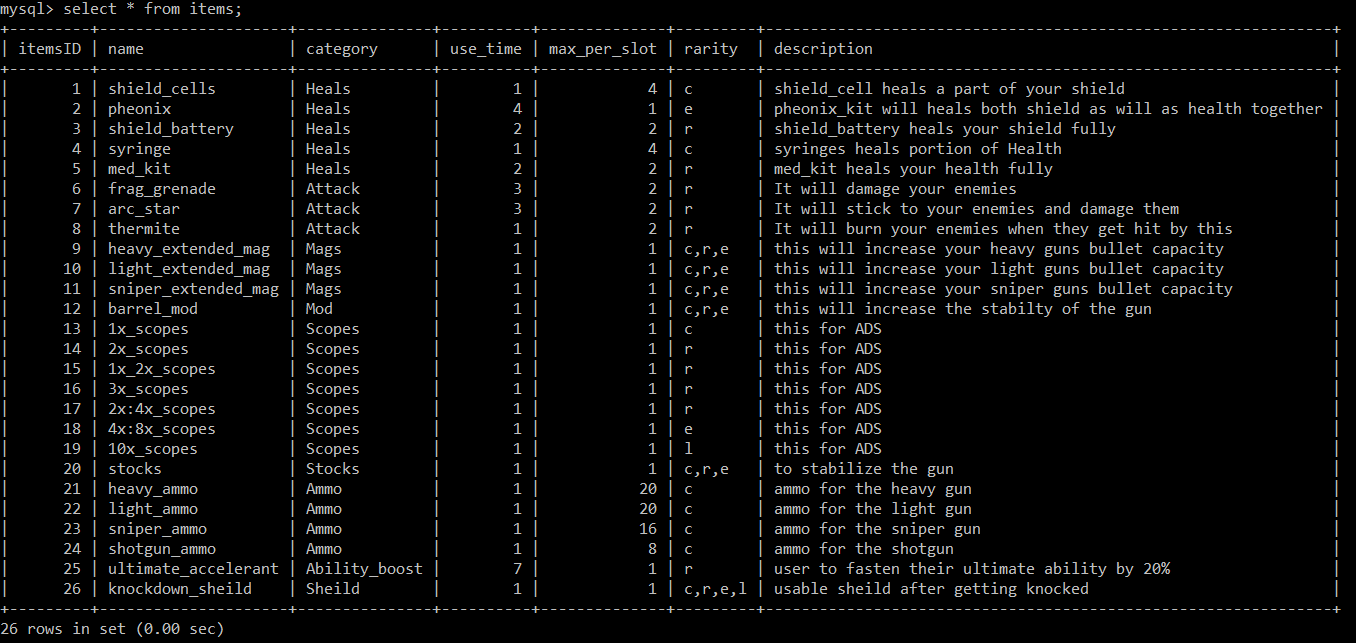


Figure 10: Rows in “**items**” table

TABLE 2: LEGENDS

SELECT \* FROM legends;



Figure 11: Rows in “**legends**” table

TABLE 3: WEAPONS

SELECT \* FROM weapons;

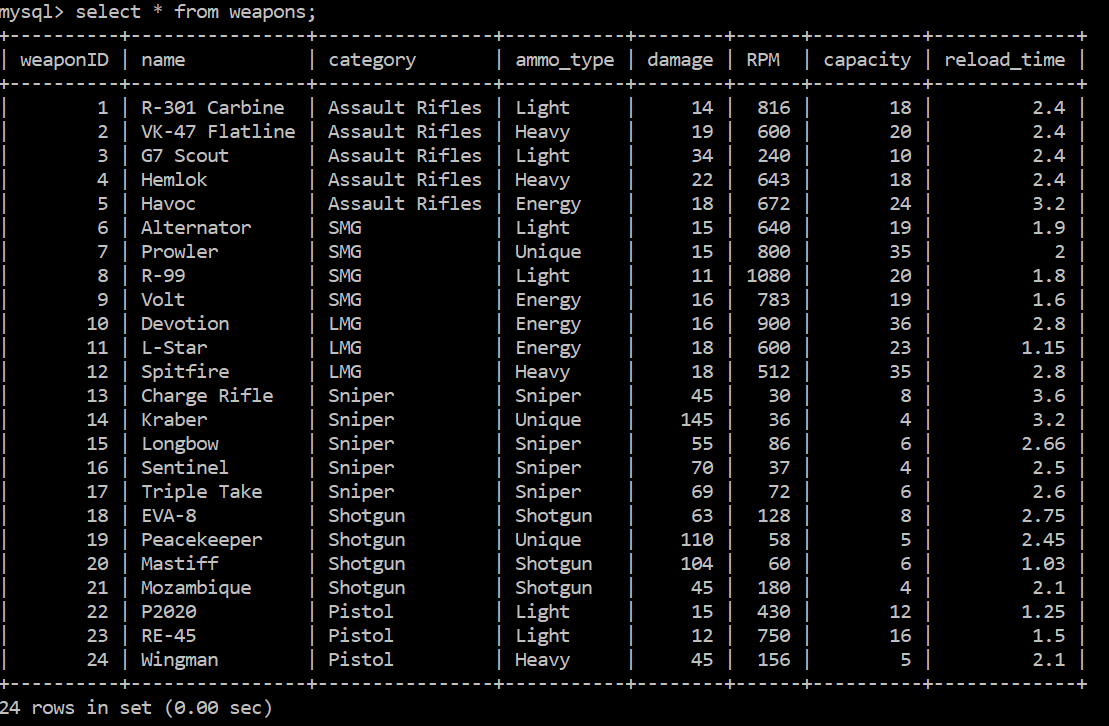


Figure 12: Rows in “**weapons**” table

TABLE 4: SEASONS

SELECT \* FROM seasons;

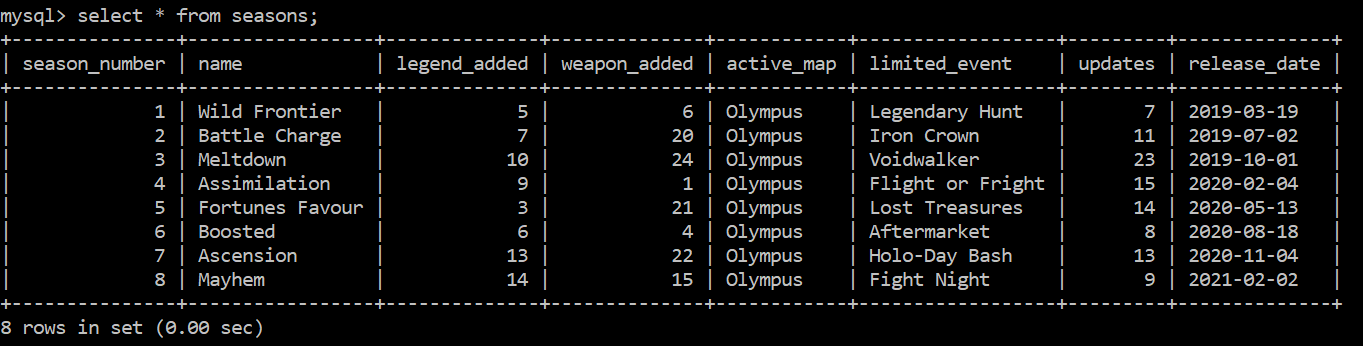


Figure 13: Rows in “**seasons**” table

TABLE 5: STATS

SELECT \* FROM stats;

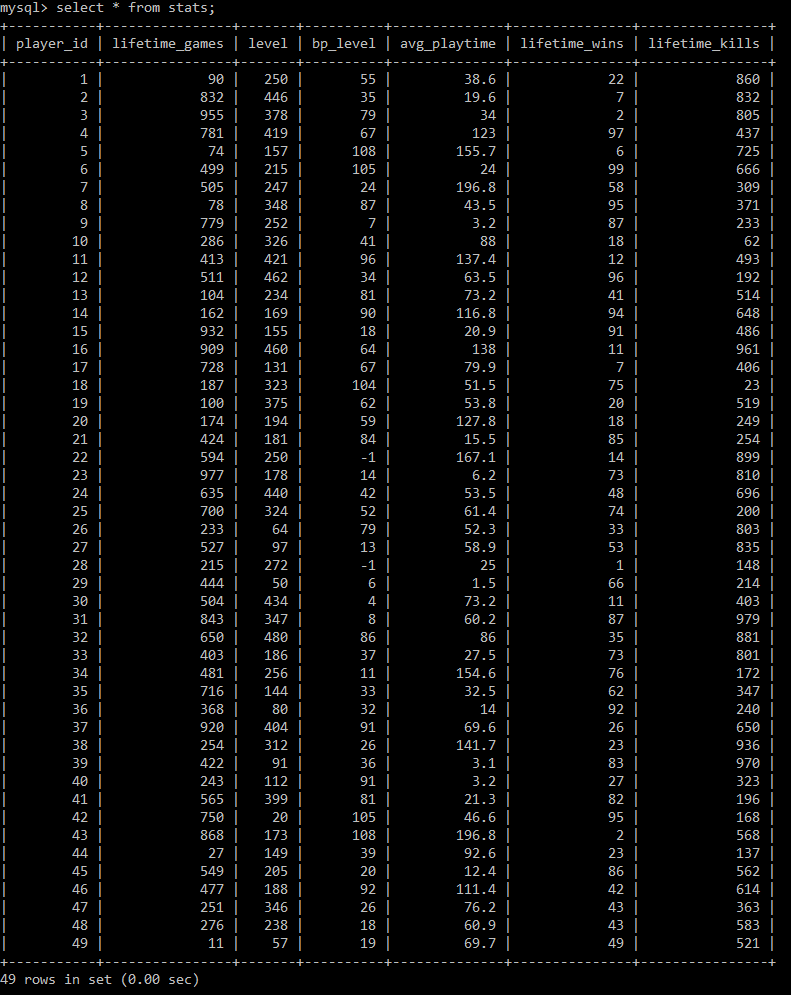


Figure 14: Rows in “**stats**” table

TABLE 6: PLAYER

SELECT \* FROM player;

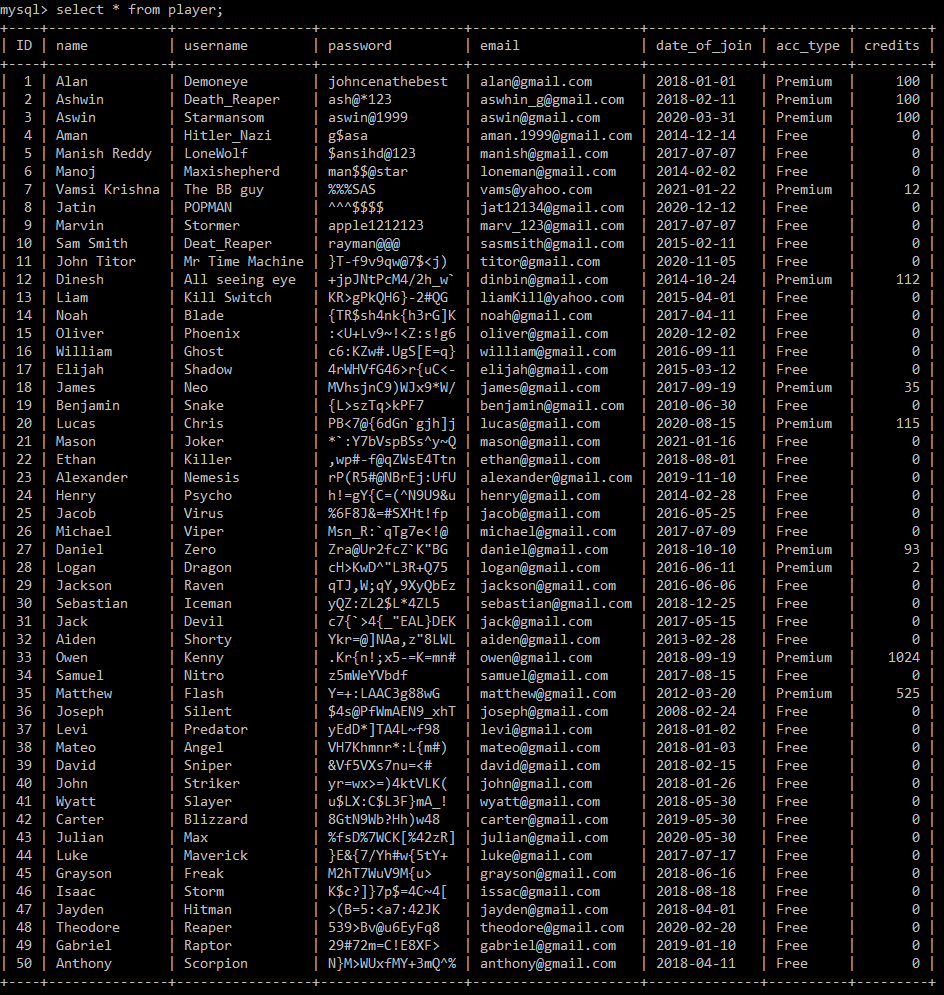


Figure 15: Rows in “**player**” table

TABLE 7: RANKS

SELECT \* FROM ranks;

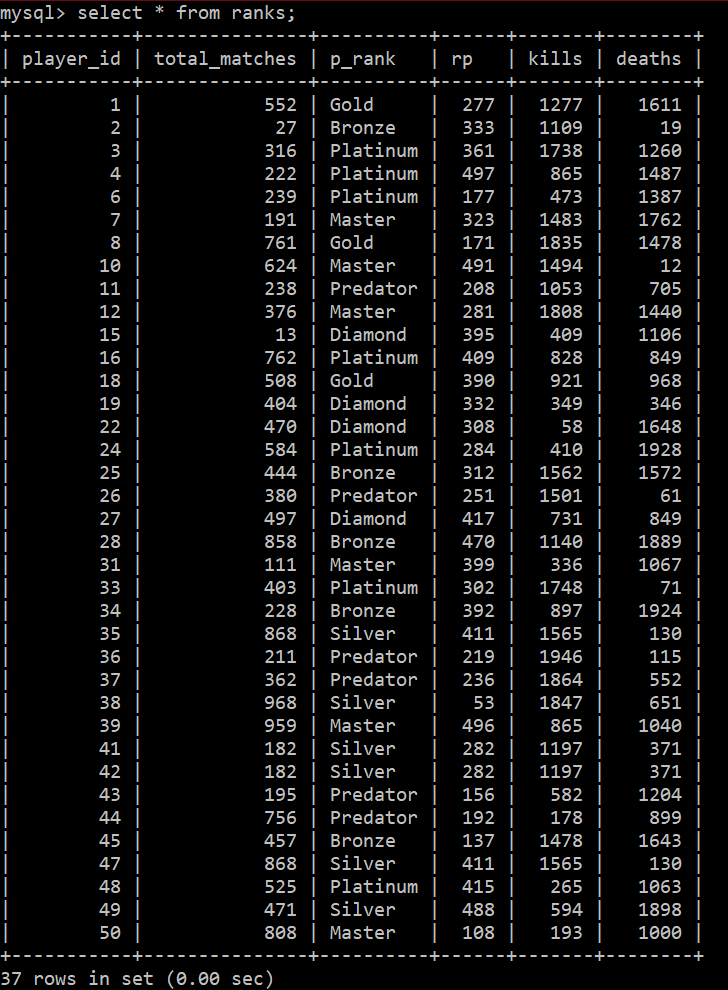


Figure 16: Rows in “**ranks**” table

TABLE 8: OLYMPUS

SELECT \* FROM olympus;

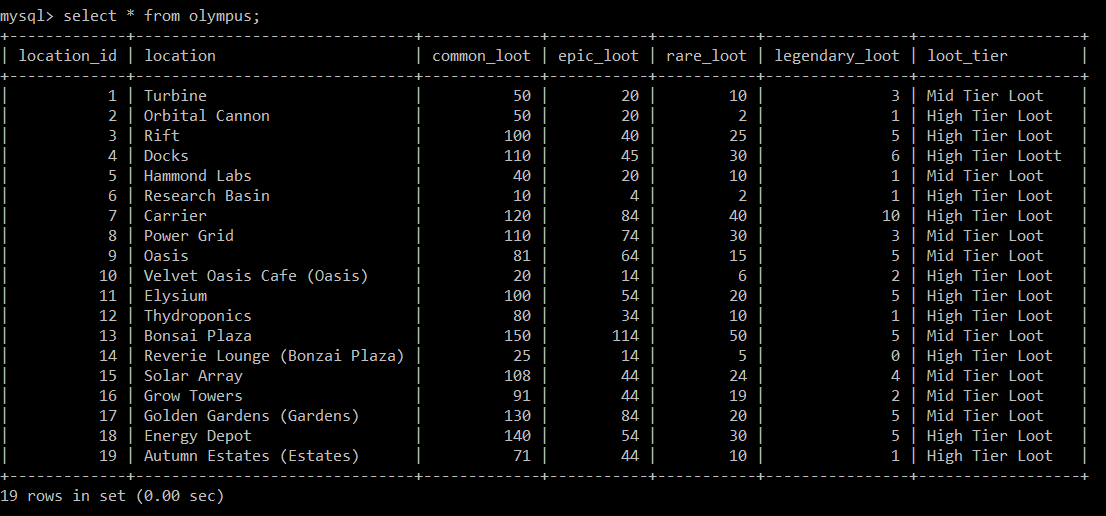


Figure 17: Rows in “**olympus**” table

COMPLEX QUERIES WITH DESCRIPTION AND THEIR O/P

QUERY 1 (TOP 5 PLAYERS):

Command used:

SELECT player.ID,player.name,player.username,stats.lifetime\_wins,stats.lifetime\_wins/stats.lifetime\_games AS win\_ratio FROM player INNER JOIN stats ON stats.player\_id=player.ID ORDER BY win\_ratio DESC LIMIT 5;

This command tells us the top 5 players of the game based on their kill to win ratio. To do this we implement the above command that uses simple division and join operations to refer the player table and display the required output. The win ratio is the division between the lifetime wins and the lifetime matches. The purpose of this list is to find the people that are eligible to compete in the E-sports tournaments.

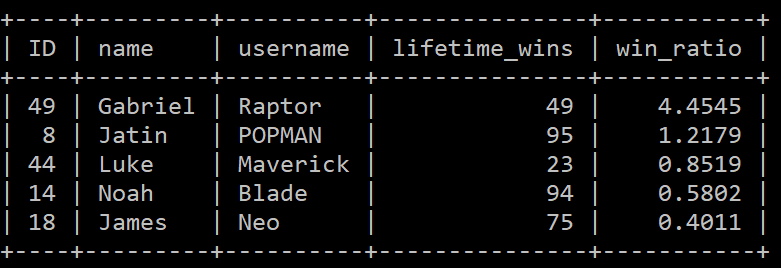


Figure 18: Output of Query 1

QUERY 2 (RECOMMENDED LEGEND ID’S):

Command used:

SELECT legendID,name,category,passive,tactical,ultimate FROM legends WHERE win\_rate = (SELECT max(win\_rate) FROM legends AS f WHERE f.category = legends.category);

This command is used to suggest the recommended legends that a player can use. This is done by implementing the above command that uses the select command with a where clause that checks the highest winrate for a particular category; used a subquery select command to get max(win\_rate) which is used a s condition data for the main select query. The output can be useful for newer players that would need a form of guidance to decide which legend they wish to play.

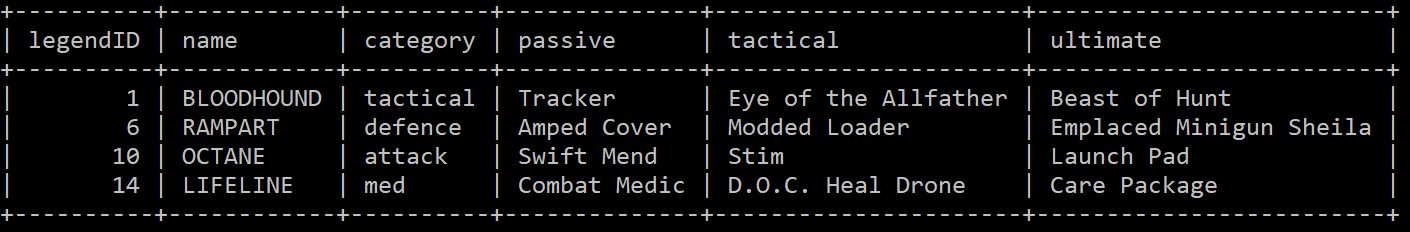


Figure 19: Output of Query 2

QUERY 3 (PLAYERS KILL/DEATH RATIO):

Command used:

SELECT player.ID,player.name, ranks.kills/ranks.deaths AS KDR FROM ranks INNER JOIN player ON player.ID = ranks.player\_id ORDER BY KDR DESC;

This table output is a statistic that shows the kill to death ratio that is represented by KDR, calculated by the division between the kills and deaths from the ranked table. The name that is displayed below is from the inner join command used, there the kdr from ranks table and the name from the player table is joined together with reference to the ID giving us the corresponding output. The purpose of this table is to give players the details of the KDR as it is an aspect most players check. Also note that here the KDR is a derived attribute.

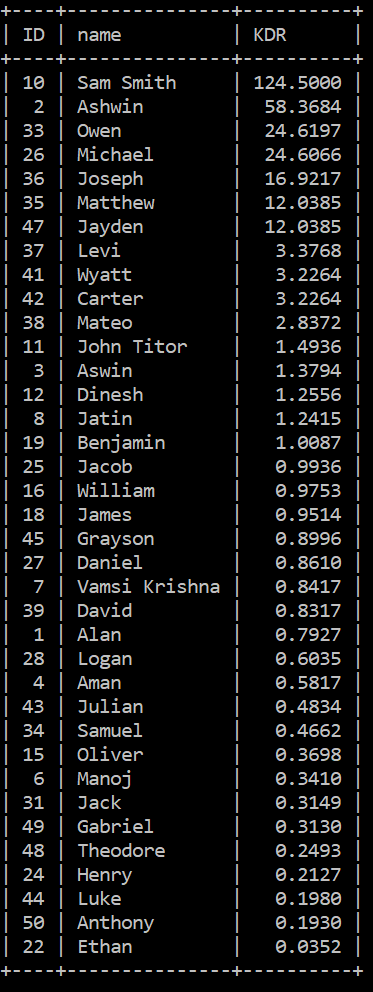


Figure 20: Output of Query 3

QUERY 4 (MAP STATE):

Command Used:

SELECT loot\_tier,count(loot\_tier) AS num\_of\_locations,SUM(common\_loot) AS total\_common,SUM(rare\_loot) as total\_rare,SUM(epic\_loot) as total\_epic,SUM(legendary\_loot) AS total\_legendary FROM olympus GROUP BY loot\_tier;

In the rare case of adding a new map is required, we can use this table for fixing or balancing the map meta to prevent any form of exploited unfair gameplay. Hence having the data of the loot items present based on the rarity and the tier can help shape better maps. This is done by executing the above-mentioned command; it uses the SUM command to get the sum of particular fields with respect to the tier by using the GROUP BY statement.

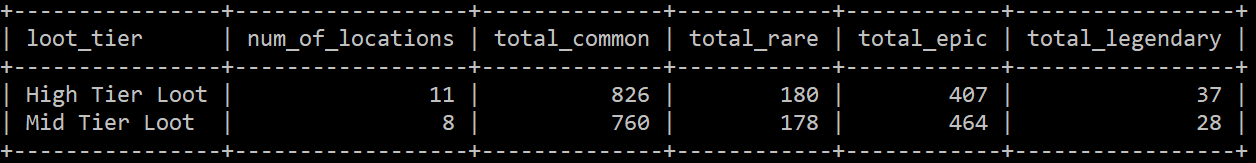


Figure 21: Output of Query 4

QUERY 5 (EQUIPMENT DATA):

Command Used:

SELECT itemsID,name,category,rarity,IF(right(rarity,1)='c','common',IF(right(rarity,1)='r','rare',IF(right(rarity,1)='e','epic',IF(right(rarity,1)='l','legendary','unknown item')))) AS max\_rarity FROM items;

We know that the likely hood of a single person taking care of the database is very unlikely. Therefore, we must take into consideration that in the case of another person is taking care of the database, he must know what represents what as it is not necessary that he plays the game itself. In the normal display of this table there are short forms used as data along an attribute. Therefore, the above command is used so that the person can understand what it represents. The used commands use the nested-if method to display the required data.

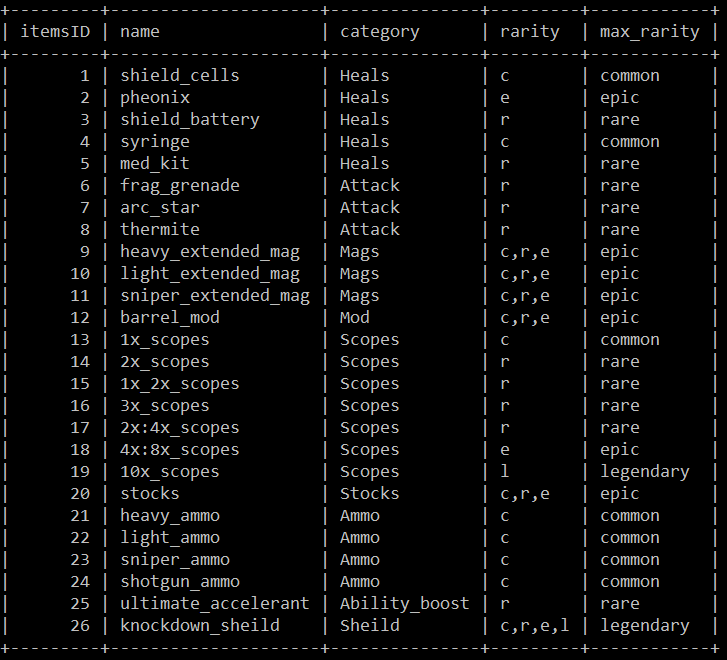


Figure 22: Output of Query 5

QUERY 6 (SEASON DETAILS):

Command Details:

SELECT seasons.season\_number,seasons.name AS season\_name,legends.name AS added\_legend,weapons.name AS added\_gun FROM seasons INNER JOIN legends ON legend\_added=legends.legendID INNER JOIN weapons ON weapon\_added=weapons.weaponID;

Here the seasons table has been normalized, which would lead to certain fields being form of IDs which are reference keys; the data details are stored in a dedicated table. Normalization is done to maintain integrity in the database. Therefore, when using the select statement directly, we will only see numbers, this will not make sense for the person evaluating the table. Hence, we will use the above statement to get outputs which has details in the form of readable text. This command makes use of join command for 2 separate tables that don’t depend on each other.

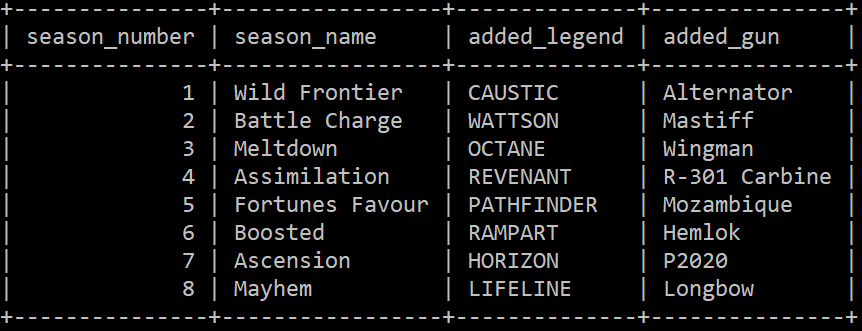


Figure 23: Output of Query 6

QUERY 7 (LEGEND META):

Command Used:

SELECT category,AVG(damage),AVG(RPM),AVG(capacity),AVG(reload\_time) FROM weapons GROUP BY category;

In games, especially the first-person shooter type, we need to make sure that the weapon meta is balanced, so that there are no guns that is better than the other by a huge margin. Otherwise, all players will use that one particular gun only which would ruin the experience and the very purpose of having other guns in the game. The way we plan to fix this is by looking at the averages of all the data of the guns based on the weapon type (category). This way we can inspect by comparing the values with the particular gun of that category. The used command here uses the AVG operation command such that it groups by the category of the weapon.

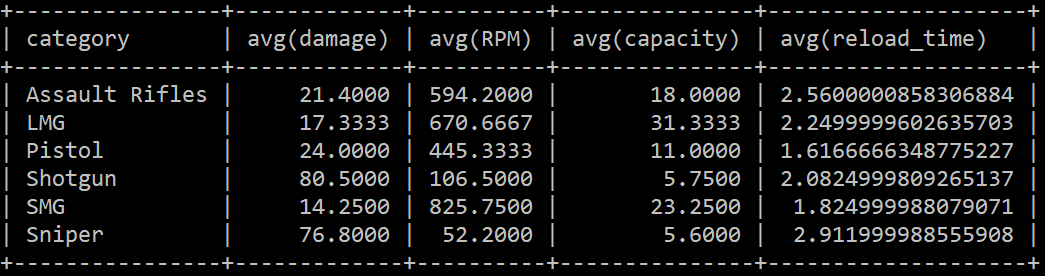


Figure 24: Output of Query 7

QUERY 8 (ID STATUS):

Command Used:

SELECT player.ID,player.name,player.username,IF(bp\_level<0,'!\_\_\_BANNED\_\_\_!','ACTIVE') AS STATUS FROM player INNER JOIN stats ON stats.player\_id=player.ID;

This is a command to view the players that were banned from the game by either cheating or any form of toxic activity. The way we implement this is by making the battle pass of cheaters as “-1”, therefore any player which has a battle pass as “-1” will be tagged banned and unable to continue the game. The way we check the players that is banned is by using the above command. The command uses the if statement along with join command that makes the player table refer the stats table so that we can see the player as a name and not as an ID which is represented as a number.

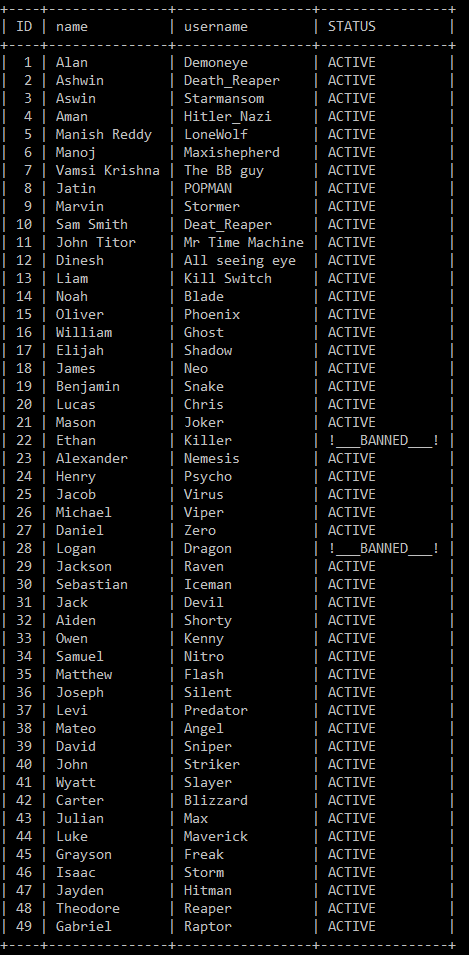


Figure 25: Output of Query 8

QUERY 9 (BEST LOCATION):

Command Used:

SELECT location,common\_loot+epic\_loot+rare\_loot+legendary\_loot AS total\_loot FROM olympus ORDER BY total\_loot DESC LIMIT 1;

This is a command that can be used as a reference for the players to see which particular location in the map has the greatest number of loots in the game. Such a location must exist to make it is a hot drop; a place where many people will land for the reason of killing other enemies at the immediate start of the game. The command used above uses an expression that adds multiple fields and orders them from high to low and then display only the first row of the output of that data. Hence the command outputs the location with the highest loot number.

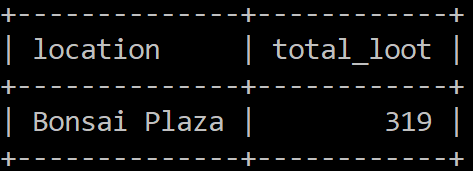


Figure 26: Output of Query 9

QUERY 10 (REWARDS AWARDED):

Command Used:

SELECT player.ID,player.name,player.username,ranks.p\_rank,CASE WHEN ranks.p\_rank='Master' THEN "2,000 CREDITS AWARDED" WHEN ranks.p\_rank='Predator' THEN "FREE LIFETIME MEMBERSHIP" ELSE "" END AS REWARDS FROM player INNER JOIN ranks ON ranks.player\_id=player.ID WHERE ranks.p\_rank='Predator' OR ranks.p\_rank='Master';

Players are the ones that breath life into the game, therefore we must reward our players so that their interest in the game will not be lost. The rewards are only for the player that have the rank MASTER and PREDATOR as these are the most difficult ranks to get in the game. The above command can be used to filter and see which players are eligible for rewards, by using the case statements and join commands so that we can see the result as readable text and not ID numbers. The players with rank PREDATOR will be rewarded Free Lifetime Membership which makes all in game purchases free and for the player with the rank MASTER will be rewarded 2000 in-game credits.

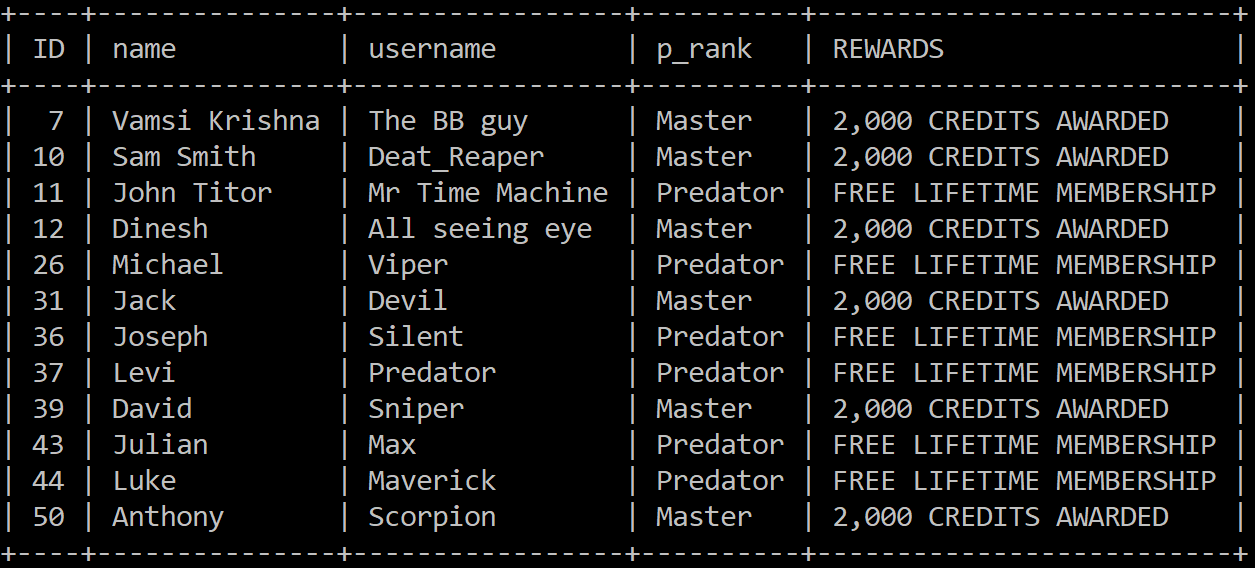


Figure 27: Output of Query 10

REFERENCES

# Complex SELECT Query <https://www.sqlshack.com/> [Last accessed: 23th January 2021]

# JOIN Clause <https://dev.mysql.com/doc/refman/8.0/en/join.html> [Last accessed: 23th January 2021]

# GROUP BY Modifiers <https://dev.mysql.com/doc/refman/8.0/en/group-by-modifiers.html> [Last accessed: 23th January 2021]

# First/Least/Max Row per Group in SQL [https://www.xaprb.com/blog /2006/12/07/how-to-select-the-firstleastmax-row-per-group-in-sql/](https://www.xaprb.com/blog%20/2006/12/07/how-to-select-the-firstleastmax-row-per-group-in-sql/) [Last accessed: 24th January]