# Shooter Game Database

CS4416 DATABASE SYSTEMS PROJECT

Dylan King | 17197813
Louise Madden | 17198232
Brian Malone | 17198178
Szymon Sztyrmer | 17200296

## Table of Contents

## Work Breakdown

SQL FILES

schema.sql: Dylan

views.sql: Szymon

triggers.sql: Louise and Szymon

REPORT

Database Description: Brian

Entity Relationship Diagram: All (Especially Louise)

Example tables: Dylan

Functional Dependencies: Dylan

Proof of Third Normal Form: Szymon

Justification of Views: Brian

Examples of Views: Brian

Indexes: Dylan and Brian

Triggers: Louise and Brian

PERCENTAGE OF WORK BY EACH MEMBER

Dylan: 25%

Louise: 25%

Szymon: 25%

Brian: 25%

## Database Description

Our database represents a FPS (First Person Shooter). We accomplish this by storing user accounts and information spanning from that account. Once a player logs in to the game, they require access to their loadouts and their statistics. Hence, they can view in-game performance as well as adjust their loadout accordingly. From their loadouts they can choose their primary weapon, secondary weapon and a perk. Effectively this database stores the needed information to run the game containing a small or a large player base.

## ENTITY-RELATIONSHIP DIAGRAM

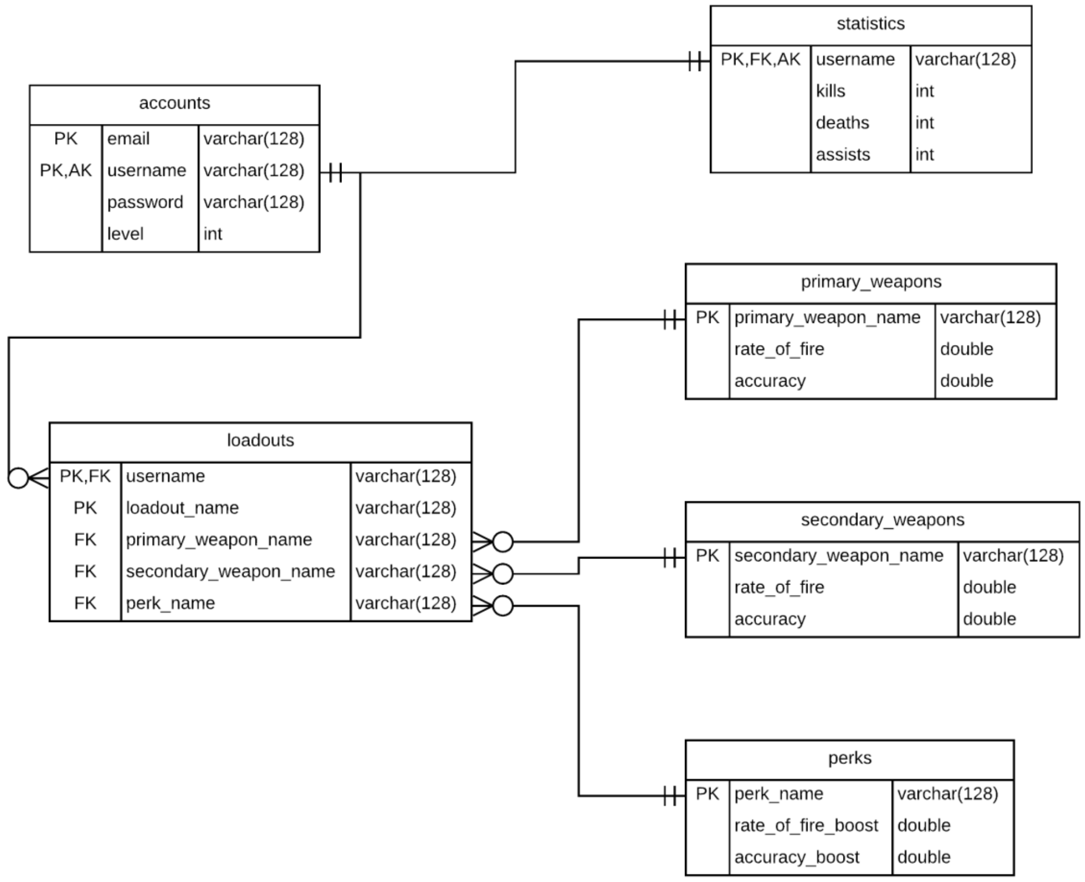

statistics
PK,FK,AK username varchar(128)
kills int
deaths int
assists int
accounts
PK email varchar(128)
PK,AK username varchar(128)
password varchar(128)
level int
primary_weapons
PK primary_weapon_name varchar(128)
rate_of_fire double
accuracy double
loadouts
PK,FK username varchar(128)
PK loadout_name varchar(128)
FK primary_weapon_name varchar(128)
FK secondary_weapon_name varchar(128)
FK perk_name varchar(128)
secondary_weapons
PK secondary_weapon_name varchar(128)
rate_of_fire double
accuracy double
perks
PK perk_name varchar(128)
rate_of_fire_boost double
accuracy_boost double

*Figure 1: Entity Relationship Diagram*

## EXAMPLE TABLES

Primary keys are listed in blue.

### Accounts Table

| **email** | **username** | **password** | **level** |
|---|---|---|---|
| brian@beep.com | bmalone100 | iambrian | 8 |
| dylan@notgmail.com | DKing1543 | password12 | 1 |
| louise@notoutlook.com | lmadden518 | pword456 | 10 |
| nik@ul.com | nikolanikolov | agoodpassword | 6 |
| szymon@notyahoo.com | SoulsIsGood | password12 | 5 |

Statistics Table

| username | kills | deaths | assists |
|---|---|---|---|
| bmalone100 | 16 | 7 | 6 |
| DKing1543 | 5 | 7 | 2 |
| lmadden518 | 7 | 15 | 9 |
| nikolanikolov | 24 | 1 | 19 |
| SoulsIsGood | 36 | 20 | 40 |

Primary Weapons Table

| primary_weapon_name | rate_of_fire | accuracy |
|---|---|---|
| AK-47 | 20 | 0.6 |
| AR-15 | 22 | 0.71 |
| AW50 | 1 | 0.92 |
| MP-7 | 40 | 0.78 |
| SPAZ-12 | 3 | 0.2 |

Secondary Weapons Table

| secondary_weapon_name | rate_of_fire | accuracy |
|---|---|---|
| Beretta 92 | 12 | 0.65 |
| Desert Eagle | 8 | 0.75 |
| M1911 | 10 | 0.58 |
| P228 | 14 | 0.55 |
| P99 | 10 | 0.68 |

Perks Table

| perk_name | rate_of_fire_boost | accuracy_boost |
|---|---|---|
| Eagle-eye | -5 | 0.18 |
| Hi-powered weapons | 16 | -0.07 |
| Quick Trigger | 10 | 0 |
| Sharpshooter | 0 | 0.11 |
| Superman | 5 | 0.06 |

Loadouts Table

| **Username** | **loadout_name** | **primary _weapon _name** | **secondary_weapon_ name** | **perk_name** |
|---|---|---|---|---|
| bmalone100 | win | SPAZ-12 | P99 | Superman |
| DKing1543 | Loadout 1 | AK-47 | Desert Eagle | Sharpshooter |
| lmadden518 | a loadout | AK-47 | P99 | Sharpshooter |
| nikolanikolov | Loadout 1 | AW50 | M1911 | Quick Trigger |
| SoulsIsGood | Accuracy | MP-7 | Desert Eagle | Eagle-eye |
| SoulsIsGood | DPS | MP-7 | P228 | Hi-Powered Weapons |

## FUNCTIONAL DEPENDENCIES

Accounts

- email = A
- username = B
- password = C
- level = D

- $A \rightarrow B$
- $A \rightarrow C$
- $A \rightarrow D$

Statistics

- username = A
- kills = B
- deaths = C
- assists = D

- $A \rightarrow B$
- $A \rightarrow C$
- $A \rightarrow D$

Primary Weapons

- primary_weapon_name = A
- rate_of_fire = B
- accuracy = C

- $A \rightarrow B$
- $A \rightarrow C$

Secondary Weapons

- secondary_weapon_name = A
- rate_of_fire = B
- accuracy = C
- $A \rightarrow B$

- A → C

Perks

- perk_name = A
- rate_of_fire_boost = B
- accuracy_boost = C
- A → B
- A → C

Loadouts

- username = A
- loadout_name = B
- primary_weapon_name = C
- secondary_weapon_name = D
- perk_name = E
- AB → C
- AB → D
- AB → E

## PROOF OF THIRD NORMAL FORM

### Accounts

- Input: A → B
  Since, {A}+ = {A}, we cannot remove A from the LHS.
  Output: A → B
- Input A → C
  Since, {A}+ = {A}, we cannot remove A from the LHS.
  Output: A → C
- Input A → D
  Since, {A}+ = {A}, we cannot remove A from the LHS.
  Output: A → D

Thus, table accounts is in third normal form.

### Statistics

- Input: A → B
  Since, {A}+ = {A}, we cannot remove A from the LHS.
  Output: A → B
- Input A → C
  Since, {A}+ = {A}, we cannot remove A from the LHS.
  Output: A → C
- Input A → D
  Since, {A}+ = {A}, we cannot remove A from the LHS.
  Output: A → D

Thus, table statistics is in third normal form.

Primary Weapons

- Input: A → B
  Since, {A}+ = {A}, we cannot remove A from the LHS.
  Output: A → B
- Input A → C
  Since, {A}+ = {A}, we cannot remove A from the LHS.
  Output: A → C

Thus, table primary weapons is in third normal form.

Secondary Weapons

- Input: A → B
  Since, {A}+ = {A}, we cannot remove A from the LHS.
  Output: A → B
- Input A → C
  Since, {A}+ = {A}, we cannot remove A from the LHS.
  Output: A → C

Thus, table secondary weapons is in third normal form.

Perks

- Input: A → B
  Since, {A}+ = {A}, we cannot remove A from the LHS.
  Output: A → B
- Input A → C
  Since, {A}+ = {A}, we cannot remove A from the LHS.
  Output: A → C

Thus, table perks is in third normal form.

Loadouts

- Input: AB → C
  Since, {A}+ = {A} AND {B}+ = {B}, we cannot remove either A or B from the LHS.
  Output: AB → C
- Input: AB → D
  Since, {A}+ = {A} AND {B}+ = {B}, we cannot remove either A or B from the LHS.
  Output: AB → D
- Input: AB → E
  Since, {A}+ = {A} AND {B}+ = {B}, we cannot remove either A or B from the LHS.
  Output: AB → E

Thus, table loadouts is in third normal form.

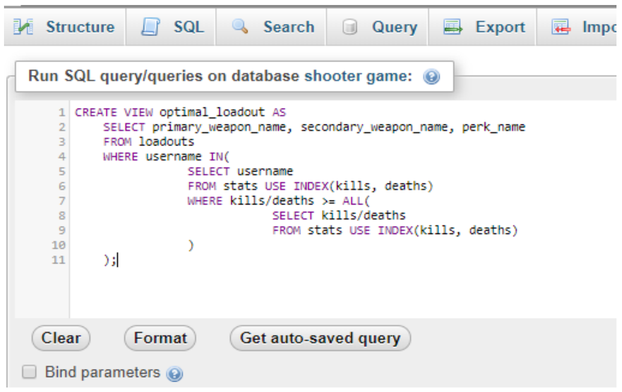
Therefore, since all of the above tables are in third normal form, our entire database is in third normal form.

## Views and Indexes

### VIEWS

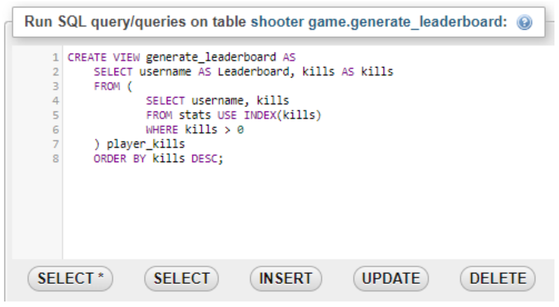
For the purpose of this database the views save information from queries which is useful for determining aspects of the game. The information is saved as tables resulting from the execution of the queries.

The 1st query demonstrates our use of sub-queries to obtain the 'optimal' loadout that is used by the best player in the game using the indexes we've made.



```sql
CREATE VIEW optimal_loadout AS
    SELECT primary_weapon_name, secondary_weapon_name, perk_name
    FROM loadouts
    WHERE username IN(
                SELECT username
                FROM stats USE INDEX(kills, deaths)
                WHERE kills/deaths >= ALL(
                            SELECT kills/deaths
                            FROM stats USE INDEX(kills, deaths)
                )
    );
```

*Figure 2: Query 1 getting the optimal loadout*

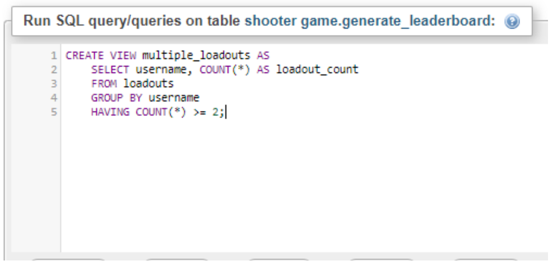

Run SQL query/queries on table shooter game.generate_leaderboard:
SELECT *
SELECT
INSERT
UPDATE
DELETE

```sql
CREATE VIEW generate_leaderboard AS
    SELECT username AS Leaderboard, kills AS kills
    FROM (
            SELECT username, kills
            FROM stats USE INDEX(kills)
            WHERE kills > 0
    ) player_kills
    ORDER BY kills DESC;
```

*Figure 3: Query making the leaderboard*

The 2nd query uses a sub-query and an order by clause to create a view of a leaderboard. This leaderboard displays the players and their total kills, ordering them in descending order.

The 3rd query utilizes a group by and having clauses to create a view table containing the list of players that make use of more than one loadout. It uses a COUNT statement to check for the number of loadouts that each player has and places that player in the table in the case they have more than 1 loadout.


Run SQL query/queries on table shooter game.generate_leaderboard:

```sql
CREATE VIEW multiple_loadouts AS
    SELECT username, COUNT(*) AS loadout_count
    FROM loadouts
    GROUP BY username
    HAVING COUNT(*) >= 2;
```

*Figure 4: Query showing players with more than 1 loadout.*

## JUSTIFICATION

The use of these views is to store information from a query that would otherwise have to be written again. These views are stored for when you need them again. Should you no longer need them you can just drop them. This is useful for gathering data on different aspects of the game.

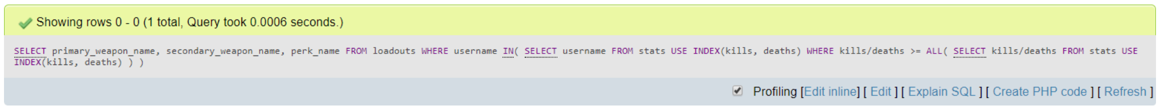
## EXAMPLES AND SCENARIOS

In our database we use the views to store information such as a leaderboard of the players with the most kills. This is useful for tracking player skill level and if we were to implement matchmaking in a larger database of players, we could use this information to that end.

We also use views to track individual players loadout choices to see how players vary their loadouts and in the current database there is no limit on the number of loadouts a player can have. In a larger player base this number would need a cap as to optimize the game's loading time. This information shows us how players like to swap out loadouts and mix up their strategies.

## INDEXES

For this database, we made two indexes, one for the 'kills' column of the stats table, and one for the 'deaths' column. The purpose of these indexes is to decrease the time spent resolving queries. Adding these queries however this does not seem to have decreased query resolution time at all.

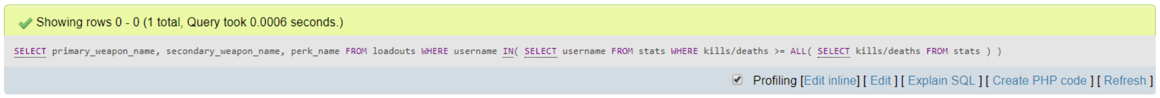


```
Showing rows 0 - 0 (1 total, Query took 0.0006 seconds.)

SELECT primary_weapon_name, secondary_weapon_name, perk_name FROM loadouts WHERE username IN( SELECT username FROM stats USE INDEX(kills, deaths) WHERE kills/deaths >= ALL( SELECT kills/deaths FROM stats USE
INDEX(kills, deaths) ) )
```

Profiling [Edit inline] [ Edit ] [ Explain SQL ] [ Create PHP code ] [ Refresh ]

*Figure 5: Query 1 using indexes*



```
Showing rows 0 - 0 (1 total, Query took 0.0006 seconds.)

SELECT primary_weapon_name, secondary_weapon_name, perk_name FROM loadouts WHERE username IN( SELECT username FROM stats WHERE kills/deaths >= ALL( SELECT kills/deaths FROM stats ) )
```

Profiling [Edit inline] [ Edit ] [ Explain SQL ] [ Create PHP code ] [ Refresh ]

*Figure 6: Query 1 without using indexes*

## JUSTIFICATION

While these indexes do not increase the performance of the database, indexes are important in larger databases, and we thought it was important to know how to properly create and use indexes.

## EXAMPLES AND SCENARIOS

The purpose of these indexes would be to increase database efficiency when resolving queries that involves the kills and deaths columns in the statistics table. If the database was used on a larger scale, these indexes would likely be invaluable in reducing query resolution time and increasing efficiency. These indexes would be useful in games such as Call of Duty and Battlefield, since we use these indexes to determine the optimal choice of weapons. This allows us to see which weapons are currently the most powerful in the game and gives us the ability to balance the weapons in future updates to the game.

## Triggers, Procedures, and Functions

In our database we wrote two triggers. One which calls the procedure after inserting into the accounts table, and one which calls the function after an update to the stats table. The procedure inserts into the stats table for the new account/username. The function calculates the average kill death ratio. It calculates the total kills and total deaths in the stats table and returns the total kills divided by the total deaths.

### JUSTIFICATION

The procedure ensures that each individual account has its' own statistics. This is useful as if the stats got mixed between accounts, the information would be jargon. It is sensible to ensure that this does not occur. Allowing the player the ability to track and 'admire' their statistics is always a popular feature in modern FPS (First Person Shooter) games.

The purpose of our function is to calculate data across the database and use this information to observe trends. Hence, why we calculate the average kill death ratio in this game's database. This figure allows us to estimate the difficulty of the game. This allows us to modify the game as necessary for balance.

### EXAMPLES AND SCENARIOS

In our database we use the triggers to call this function and procedure. We could potentially write other procedures to update information similar to the way we did above, as well as using other functions to gather other useful information. We update statistics and we observe kill death ratio. We also discussed updating the loadouts table to minimize the amount of variety players can switch between, so that they can maximize their current loadouts. We could also write a function to check the most commonly used weapons to potentially create recommended loadouts for any new players.

## Table of Figures