Serverstats Server Module

For silEnT mod, an Enemy Territory Modification

Version 1.0

TABLE OF CONTENTS

1. Background

1.1. General

Serverstats server module is an extension to silEnT mod, an Enemy Territory game modification. Enemy Territory is a free to play game created and released by SplashDamage as a sequel to Return to Castle Wolfenstein, a game created by id Software.

The server statistics module implements server side game data gathering and warehousing in a relational database. The module is designed to be able to use several different database managers. Including SQLite3 and PostgreSQL. Extending the possibilities is possible in the future as long as the licenses are compatible.

1.2. What it does and what it does not do

The module works only as a data collector. The module is not a replacement for the silEnT mod player database.

1.3. Database Requirements

The serverstats module requires the following privileges to the database: CREATE TABLE, CREATE INDEX, SELECT, INSERT and UPDATE. The module does not require any privileges for deleting or dropping data. Also, the module does not enforce mixed case on table column names. Do note that with all client connections to the databases.

1.4. DBMS

The module is compatible with SQlite3 and PostgreSQL DBMS. Client libraries to both of these are statically linked and do not require you to install anything additional. In case of the PostgreSQL, you need to have a configured PostgreSQL server to use this module.

1.5. Performance

The module is implemented with threading. This means that the interface facing the qagame, will work in high speed and also, that multicore CPUs are recommended for taking full advantage of the module.

1.6. Customization

The module does not require the database to be in any certain format. The database can be customized to the user needs freely by adding new tables, indexes and triggers. Even altering existing tables is possible. However, by doing customizations, users will take the risk that the database can be changed by the future versions of the module and the changes may overlap with the customizations. The module will not remove, by itself, any customizations.

1.7. Versions

The intent is to keep new modules and databases backward compatible to older versions of the client software. The following table describes the versions when the compatibility gets broken.

Module Version	Backwards Compatible	Description of Change
1.0	0.9.0	-

In the table, the Backwards Compatible field has the value of the oldest silEnT mod version for which this module is compatible.

2. Configuration

2.1. Configuring silEnT Mod Server

To enable the use of the statistics module, the server game must be configured for it. This is done with "modules.cfg" file and by adding specific data block about the statistics module into that file. The modules.cfg file is automatically read every time the server initializes the qagame game. A block like the following needs to be added to the configuration file:

```
[statsmodule]
path = servermodules
file = statsmodule
config = statmodule.cfg
enabled = 1
```

- *[statsmodule]* ; This identifies the module that is configured.
- *path* ; This is the path to the directory where the module shared library (statsmodule.dll/so) is located. This path is relative to the "fs_homepath\fs_game" directory. I.e. it is appended to a path that begins from the directory where your qagame for silEnT mod is located.
- *config* ; This is the configuration file which is read by the statistics module. Do note that this value can also include relative path to the file.
- *enabled* ; If this is set to 1, the module is enabled, if it is set to 0 it is disabled. This allows enabling and disabling the module without making other changes to the configurations.

2.2. Configuring the Statistics Module

Serverstats module is configured with "*key = value*" pairs read from a configuration file that is pointed to by the generic modules.cfg configuration. Commenting is done with ';' character. If the ';' character appears anywhere in the configuration file, the rest of that line is ignored. The file name is passed to the module from silEnT mod. The configuration accepts the following options.

2.2.1. sql_database

This option defines the name of the database the module is accessing. If SQLite3 is used, this value is practically a file name and therefore, it can also contain relative path to another directory if required.

2.2.2. sql_engine

This option defines the database engine that is used. Value can be either *sqlite* or *postgresql*. This option must be defined. By default: "sql_engine = sqlite".

2.2.3. sql_hostaddr

This value defines the IP address of the database. This value is relevant for databases that use sockets to communicate. With SQLite3, this value can be ignored in the configuration.

2.2.4. sql_hostport

This value defines the port in which the database is listening. Together with sql_hostaddr, these define IP:port pair. This value is relevant for databases that use sockets to communicate. With SQLite3, this value can be ignored in the configuration.

2.2.5. sql_username

This value defines the user name used to authenticate with database server. With SQLite3, this value can be ignored in the configuration.

2.2.6. sql_password

This value defines the password used to authenticate with database serer. With SQLite3, this value can be ignored in the configuration.

2.2.7. sql_debuglogfile

This value is optional. If set, all SQL actions are logged to a file with this name. Do note that the file will grow very large very fast.

2.2.8. sql_truncdebuglog

If this value and sql_debuglogfile is set, the SQL debug log will be truncated each time the module is loaded. This means every map change. Do note, that if this is set, errors that have happened a long time before they are getting noticed, will not be found from the SQL debug log.

2.2.9. sql_logexecutiontimes

Boolean value (true/false) that defines if the SQL debug log will include SQL query execution times. This requires sql_debuglogfile to be set. By default this is set to false.

2.2.10. track_deaths

Boolean value (true/false) that defines if the player deaths are stored to the database. This is part of the extended data collection.

2.2.11. track_teams

Boolean value (true/false) that defines if the team changes and participation is tracked. This is part of the extended data collection.

2.2.12. track_classes

Boolean value (true/false) that defines if player class changes are tracked. This is part of the extended data collection.

2.2.13. track_shots

Boolean value (true/false) that defines if shots are tracked. If enabled, every shot is stored into the database. This information includes the shooter position, the target position and the target head view angles. If the shot was made to "air" i.e. it did not hit any other player, the target position is the position in the map where the shot trace ended. It is possible to disable storing misses with "shots_ignoremisses" boolean.

2.2.14. shots_ignoremisses

Boolean value (true/false) that defines if missed shots are tracked. Requires "track_shots" to be set. This option exists to reduce the stored information to the database, if shots are collected, but misses are not interesting for the use case.

2.2.15. track_pickups

Boolean value (true/false) that defines if items picked up by players are inserted to the database.

2.2.16. track_positions

Boolean value (true/false) that defines if player positions are collected into the database. The interval is configurable between 1 seconds to any large value.

2.2.17. position_track_interval

An integer value that defines the position tracking interval in seconds. Do note that interval shorter than 1 second is not possible.

2.2.18. track_objectruns

Boolean value (true/false) that defines if object runs are stored into the database.

2.2.19. track_dynamites

Boolean value (true/false) that defines if dynamite plants on objectives is stored into the database.

2.2.20. track_constructibles

Boolean value (true/false) that defines if engineering for building objectives and any damage to destroy objectives is tracked.

2.2.21. coordinates_x_decimals

An integer value that defines how many decimals are stored for the X coordinate positions. If more decimals are stored, the database size increases more through the Position table. By default this value is 0, which is still enough to compare positions against command map with good accuracy. If the value is set to a negative integer, the rounding is done on the left side of the decimal point.

2.2.22. coordinates_y_decimals

An integer value that defines how many decimals are stored for the Y coordinate positions. If more decimals are stored, the database size increases more through the Position table. By default this value is 0, which is still enough to compare positions against command map with good accuracy. If the value is set to a negative integer, the rounding is done on the left side of the decimal point.

2.2.23. coordinates_z_decimals

An integer value that defines how many decimals are stored for the Z coordinate positions. If more decimals are stored, the database size increases more through the Position table. By default the database stores the Z coordinates as command map layer numbers. Most use cases would look at the coordinates from top down where the accurate Z coordinate is not needed. If the value is set to a negative integer, the rounding is done on the left side of the decimal point.

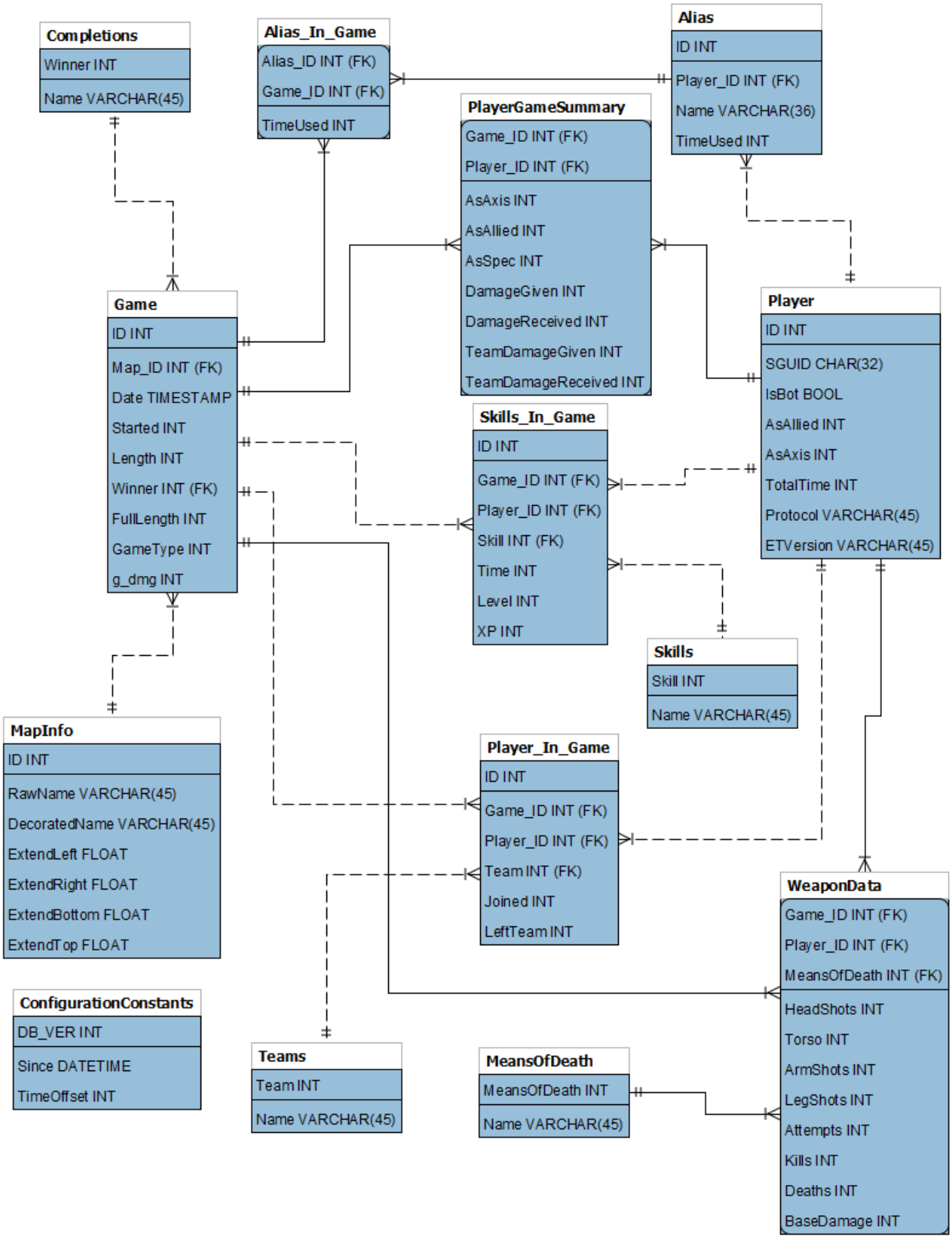
2.2.24. coordinates_usezlayers

A boolean value (true/false) that defines if the Z coordinate positions are stored as command map layer numbers. By default this is set to true as this reduces significantly the amount of variations in the Position table and most use cases look at the positions from top down direction.

3. Database Architecture

3.1. Basic Statistics

The database is divided into basic and extended statistics. What is collected is configurable. The extended part can grow the database with great speed. The basic statistics are the minimal collected data when enabled. The following displays the tables and describes them with detail.



3.1.1. ConfigurationConstants

The database has some information that may be version dependent. This table provides the version dependent information and how to manage this data. The TimeOffset field tells the offset seconds to add to various second times used by many of the game data tables, to convert the time into seconds since epoch. The reason why this offset is used is because of the way the game handles calendar time, without overflowing integer types. The **Since** field can be used to determine which database version has been used for which game. The Game table also includes a time stamp field and the Game table is linked to all entries which may use seconds to determine time.

Field	Type	Description
DB_VER	INT	The database version.
Since	DATETIME	When this version has been used the first time.
TimeOffset	INT	The number of seconds to add to get seconds since epoch.

3.1.2. Game

The Game table holds single games. Almost all collected data is tied to games. Notable thing about the table is the Date field. Using this field it is possible to compare against the **ConfigurationConstants.Since** field and see under which version the data related to this game is stored.

Field	Type	Description
ID	INT	Integer value for the sole purpose to uniquely identify the games.
Map_ID	INT	Foreign key constraint to **MapInfo.ID**.
Date	TIMESTAMP	This is the calendar time when the game was created to the database.
Started	INT	Seconds when the game started.
Length	INT	Seconds the game lasted.
Winner	INT	The team that won the game, ot the ending condition. Human readable interpretation of this value is found from the **Completions** table.
FullLength	INT	Seconds the map should have lasted if it was played the full time.
GameType	INT	This is the game type. This value is g_gametype value of the map.
g_dmg	INT	The g_dmg setting of the map. This value can be interesting for evaluating hits.

3.1.3. Completions

This is a supportive table which includes all different **Game.Winner** values and the human readable explanations. The module fills this table to the database on every start up.

Field	Type	Description
Winner	INT	The game completion condition identifier.
Name	VARCHAR	Human readable explanation of the condition.

3.1.4. Player

The Player table is one of the most central tables in the database. This table is linked to almost everything that is collected. The silEnT GUID is automatically updated in the database if the silEnT mod can itself link the new GUID to the old GUID.

Field	Type	Desription
ID	INT	Integer value for the sole purpose of being primary key.
SGUID	CHAR(32)	32 character player silEnT GUID. Be sure not to reveal the full GUID publicly to avoid issues with GUID spoofing.
IsBot	BOOL	1 if the player is server bot. 0 otherwise.
AsAllied	INT	Time in seconds as allied.
AsAxis	INT	Time in seconds as axis.
TotalTime	INT	Total time in seconds the player has been on the server. This time includes time from warmups and all non gaming states.
Protocol	VARCHAR	The protocol version the client game uses.
ETVersion	VARCHAR	The game version the player is using, as reported by the player without binary checks.

3.1.5. Player_In_Game

Table stores all the team changes during the games done by players. Players can have several records for each played game. Note that when the player joins the server, he automatically joins the spectator team.

Field	Type	Description
ID	INT	Integer value for the sole purpose to uniquely identify the record.
Game_ID	INT	Foreign key constraint to **Game** table to uniquely identify the game.
Player_ID	INT	Foreign key constraint to **Player** table to uniquely identify the player.
Team	INT	The team, this is also foreign key constraint to **Teams** table.
Joined	INT	Seconds since the game start when the player joined the team.
LeftTeam	INT	Seconds since the game start when the player left the team.

3.1.6. Alias

The Alias table stores every alias any player has used while on the server. This includes warmup times and as spectator. Each game also stores which aliases were used and for how long. Notable is, that even a single color code change creates a new alias for the player.

Field	Type	Description
ID	INT	Integer value for the sole purpose to uniquely identify aliases.
Player_ID	INT	Foreign key constraint that uniquely identifies the player.
Name	VARCHAR	The name of the player. This includes the color codes.
TimeUsed	INT	Number of seconds this alias has been used by this player.

3.1.7. Alias_In_Game

The Alias_In_Game table connects aliases to games.

Field	Type	Description
ID	INT	Integer value for the sole purpose to uniquely identify the record.
Alias_ID	INT	Foreign key constraint to **Alias** table to uniquely identify the alias.
Game_ID	INT	Foreign key constraint to **Game** table to uniquely identify the game.
TimeUsed	INT	Time in seconds how much this alias was used during the game.

3.1.8. Teams

This is a supportive table that links different team identifiers to human readable texts. The module fills this table to the database on startup.

Field	Type	Description
Team	INT	Integer value that identifies the team.
Name	VARCHAR	Human readable explanation to team.

3.1.9. Skills_In_Game

This table collects player skills (XP) on each game. For every game there will be at least two records. One for the time when the player joins the game and one for the time the game ends or the player leaves the game. Also, a new record is added every time a player gains a skill upgrade. In case of upgrades, the *Time* field tells the second when the skill upgrade came to effect.

Field	Type	Description
ID	INT	Integer value for the sole purpose to uniquely identify the record.
Game_ID	INT	Foreign key constraint to **Game** table to uniquely identify the game.
Player_ID	INT	Foreign key constraint to **Player** table to uniquely identify the player.
Skill	INT	The skill this record describes. Also a foreign key constraint to **Skills** table.
Time	INT	Seconds since the game start when this record came to effect.
Level	INT	The skill level the player is on.
XP	INT	The amount of XP the player has on this skill.

3.1.10. Skills

This is a supportive table that has all the possible skills in the game and their human readable names. This table is automatically filled by the module on every game startup.

Field	Type	Description
Skill	INT	The skill identifier.
Name	VARCHAR	Human readable name for the skill.

3.1.11. MapInfo

This table holds details of specific maps. This table requires that the raw map names, i.e. the bsp names of the maps are unique. Otherwise, some details may be interpreted wrong per map. None of the stored information is vital though. The **ExtendLeft**, **ExtendRight**, **ExtendBottom** and **ExtendTop** fields are comparable to coordinate positions with extended statistics.

Field	Type	Description
ID	INT	Integer value for the sole purpose to uniquely identify record.
RawName	VARCHAR	The bsp -name of the map. The value must be unique with the rows.
DecoratedName	VARCHAR	The long name of the map. As seen at the map vote screen.
ExtendLeft	FLOAT	Command map extends to left.
ExtendRight	FLOAT	Command map extends to right.
ExtendBottom	FLOAT	Command map extends to bottom.
ExtendTop	FLOAT	Command map extends to top.

3.1.12. WeaponData

This table holds individual weapon statistics of a game.

Field	Type	Description
Game_ID	INT	Foreign key constraint to **Game.ID** to uniquely identify the game.
Player_ID	INT	Foreign key constraint to **Player.ID** to uniquely identify the player.
MeansOfDeath	INT	Foreign key constraint to **MeansOfDeath.MeansOfDeath**.
HeadShots	INT	Number of head shots with the weapon.
Torso	INT	Number of hits to the torso.
ArmShots	INT	Number of arm hits with the weapon.
LegShots	INT	Number of leg hits with the weapon.
Attempts	INT	Number of time the weapon was fired.
Kills	INT	Number of kills made with the weapon.
Deaths	INT	Number of deaths caused by the weapon.
BaseDamage	INT	Damage to health with this weapon. When hit to the body. For arm and leg shots the damage may be reduced based on the g_dmg setting.

3.1.13. MeansOfDeath

This is a supportive table that links identifying MeansOfDeath integers to human readable texts.

Field	Type	Description
MeansOfDeath	INT	Integer value to uniquely identify the means of death.
Name	VARCHAR	Human readable name for the death.

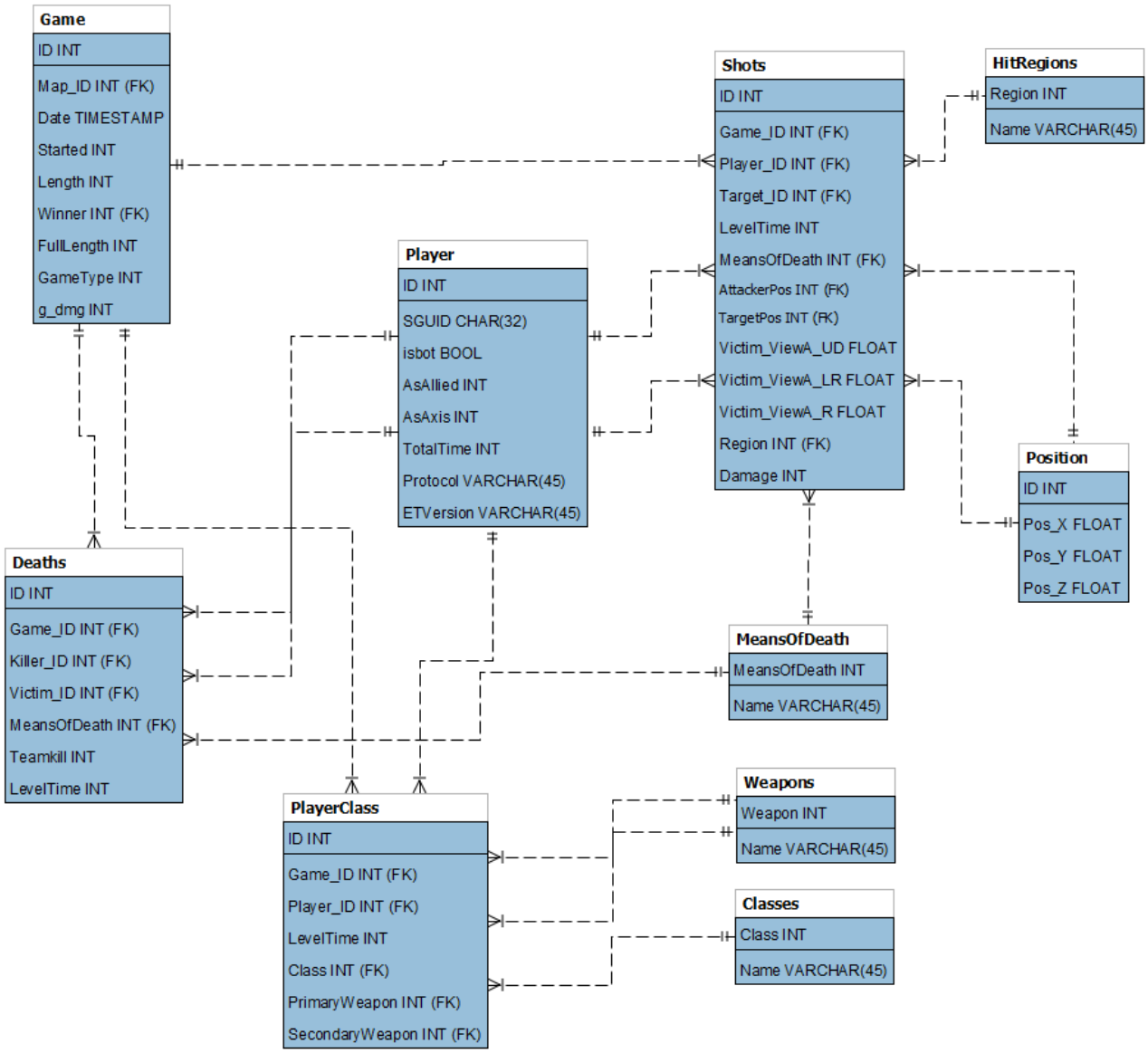
3.1.14. PlayerGameSummary

This table collects summaries of players in games. There is one row fir every player that participated in one game. The Game_ID and Player_ID columns together form the table primary key.

Field	Type	Description
Game_ID	INT	Foreign key constraint to Game.ID.
Player_ID	INT	Foreign key constraint to Player.ID.
AsAxis	INT	Seconds the player played for the axis team.
AsAllied	INT	Seconds the player played for the allied team.
AsSpec	INT	Seconds the player sat in spectators.
DamageGiven	INT	Damage inflicted to enemy team members.
DamageReceived	INT	Damage received from enemy team members.
TeamDamageGiven	INT	Damage inflicted to team members.
TeamDamageReceived	INT	Damage received from team members.

3.2. Extended Statistics

Extended statistics provide many separately configurable statistics collection options. These include deaths, team joins, class choices and shots. The **Game** and **Player** tables are described in the basic statistics.



3.2.1. PlayerClass

This table holds all class changes players have done. This is updated only when players spawn to the game, and class, primary weapon or secondary weapon has changed.

Field	Type	Description
ID	INT	Integer value for the sole purpose to uniquely identify the record.
Game_ID	INT	Foreign key constraint to **Game.ID** to uniquely identofy the game.
Player_ID	INT	Foreign key constraint to **Player.ID** to uniquely identify the player.
LevelTime	INT	Milliseconds since the map start for this spawn/change.
Class	INT	Foreign key constraint to **Classes.Class** table to identify the class.
PrimaryWeapon	INT	Foreign key constraint to **Weapons.Weapon** to identify the weapon.
SecondaryWeapon	INT	Foreign key constraint to **Weapons.Weapon** to identify the weapon.

3.2.2. Weapons

This is a supportive table. It links all weapon numbers to weapon names. This data should not be confused with MeansOfDeath or WeaponStats that also use weapon names. These are all and strictly all weapons unlike the other mentioned datas that are mixed/combined information.

Field	Type	Description
Weapon	INT	Unique integer value for every weapon.
Name	VARCHAR	Human readable name for the weapon.

3.2.3. Classes

This is a supportive table that links all class identifiers to human readable class names.

Field	Type	Description
Class	INT	Unique integer value for every class.
Name	VARCHAR	Human readable name for the class.

3.2.4. Deaths

Table collects all player kills and deaths. However, sometimes the killer might not be another player. For these cases, the Player table includes two predefined constant players: none and world.

Field	Type	Description
ID	INT	Integer value for the sole purpose to uniquely identify the record.
Game_ID	INT	Foreign key constraint to **Game.ID** to identiy the game.
Killer_ID	INT	Foreign key constraint to **Player.ID** to identify the killer.
Victim_ID	INT	Foreign key constraint to **Player.ID** to identify the victim.
MeansOfDeath	INT	Foreign key constraint to **MeansOfDeaths.MeansofDeath** to identify how the kill was made.
Teamkill	INT	Non zero integer if the death was a teamkill. Otherwise it is 0.
LevelTime	INT	Milliseconds from map start when the death happened.

3.2.5. Shots

This table can store every shot made in a game. This data can be used to create heat maps or any deeper analysis of player shots. If enabled, this data will consume space with great speed.

Field	Type	Description
ID	INT	Integer value for the sole purpose to uniquely identify this record.
Game_ID	INT	Foreign key constraint to **Game.ID** to identiy the game.
Player_ID	INT	Foreign key constraint to **Player.ID** to identify the shooter.
Target_ID	INT	Foreign key constraint to **Player.ID** to identify the target. If the target is None or World pre-defined constant, the shot missed all players.
LevelTime	INT	Milliseconds from the map start when the shot was fired (by server).
MeansOfDeath	INT	Foreign key constraint to **MeansOfDeaths.MeansofDeath** to identify the method of attack.
AttackerPos	INT	Foreign key constraint to **Position.ID**.
TargetPos	INT	Foreign key constraint to **Position.ID**.
Victim_ViewA_UD	FLOAT	The Up/Down view angle of the victim if hit. 0.0 if the shot missed all players.
Victim_ViewA_LR	FLOAT	The Left/Right view angle of the victim if hit. 0.0 if the shot missed all players.
Victim_ViewA_R	FLOAT	The Roll view angle of the victim if hit. 0.0 if the shot missed all players.
Region	INT	Foreign key constraint to **HitRegions.Region** to identify where the shot hit.
Damage	INT	If the shot hit a player, the exact damage that was inflicted to the target.

3.2.6. Position

Position table is used to store coordinate positions. One row in the Position table can be referenced by many rows in many tables. The intent of the Position table is to reduce space requirements in the long run. In time it becomes more and more likely that the same positions get used repeatedly. The columns *Pos_X* and *Pos_Y* are used to automatically create an index to the table. This index is very important and without it, the execution times of most actions will sky rocket.

Field	Type	Description
ID	INT	Integer value for the sole purpose to uniquely identify the record.
Pos_X	FLOAT	The X coordinate position.
Pos_Y	FLOAT	The Y coordinate position.
Pos_Z	FLOAT	The Z coordinate position.

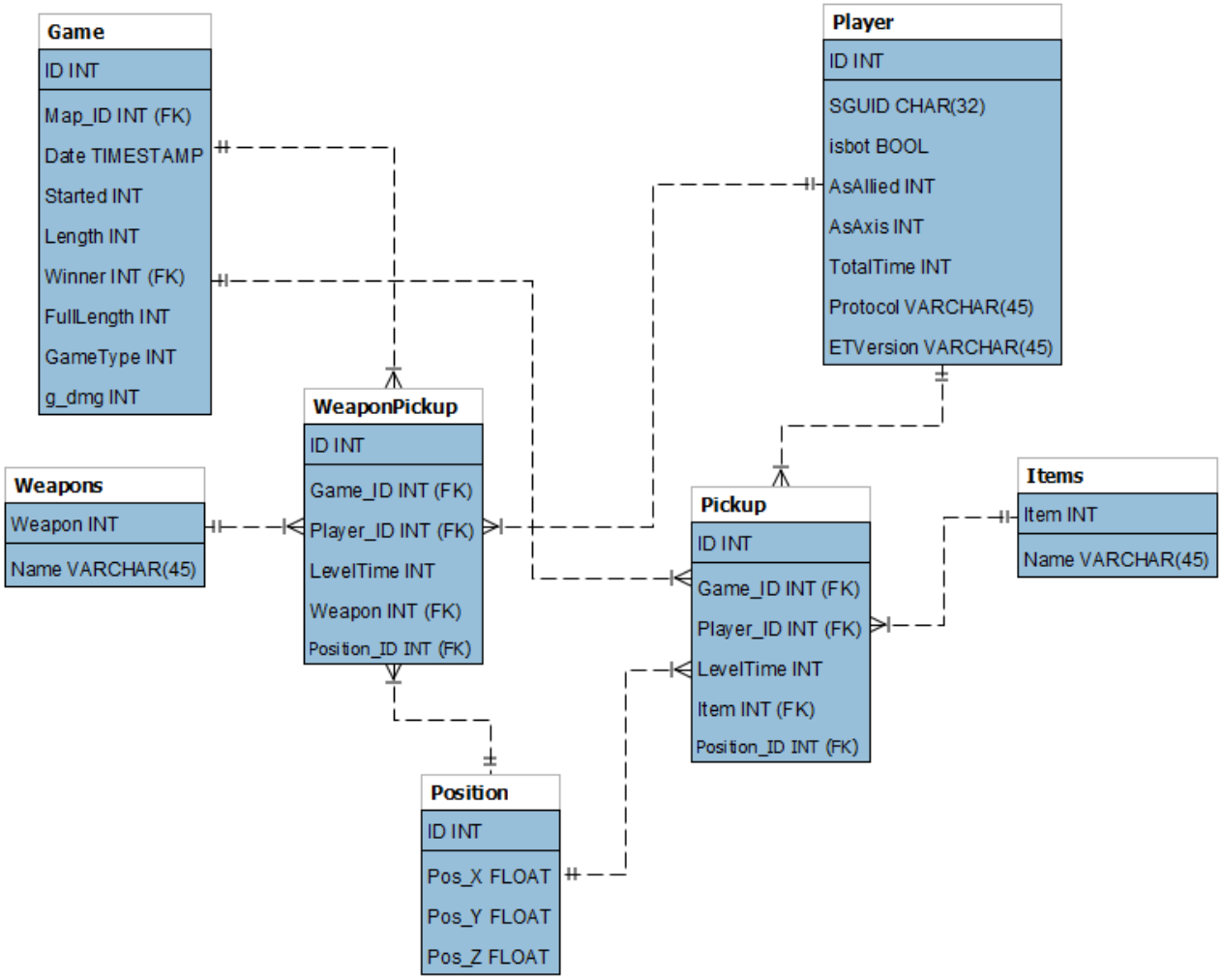
3.2.7. HitRegions

This is a supportive table that links different hit regions to human readable names.

Field	Type	Description
Region	INT	Integer value that identifies the hit region.
Name	VARCHAR	Human readable name for the hit region.

3.3. Pickup Statistics

Pickup statistics are part of the extended data collection. All items players pick up can be collected to the database. The tables **Player**, **Weapons** and **Game** are described in the basic statistics.



3.3.1. Pickup

This table collects all pickups that do not have specialized tables.

Field	Type	Description
ID	INT	Integer value for the sole purpose to uniquely identify the record.
Game_ID	INT	Foreign key constraint to **Game.ID**.
Player_ID	INT	Foreign key constraint to **Player.ID**.
LevelTime	INT	Milliseconds from the map start.
Item	INT	Foreign key constraint to **Items.Item**.
Position_ID	INT	Foreign key constraint to **Position.ID**.

3.3.2. Item

This is a supportive table that links item numbers to human readable names.

Field	Type	Description
Item	INT	Unique integer value to identify the item.
Name	VARCHAR	Human readable name for the item.

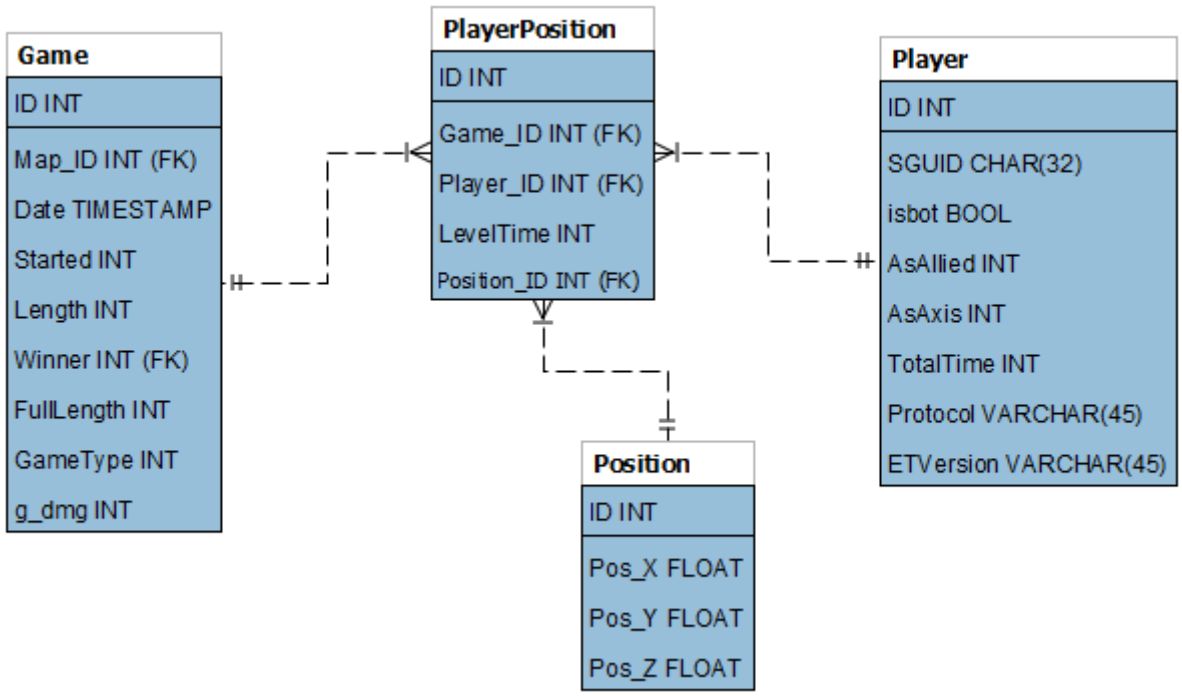
3.3.3. WeaponPickup

This table collects all weapon pickups.

Field	Type	Description
ID	INT	Integer value for the sole purpose to uniquely identify the record.
Game_ID	INT	Foreign key constraint to **Game.ID**.
Player_ID	INT	Foreign key constraint to **Player.ID**.
LevelTime	INT	Milliseconds from the map start.
Weapon	INT	Foreign key constraint to **Weapons.Weapon**.
Position_ID	INT	Foreign key constraint to **Position.ID**.

3.4. Player Positions

Player positions can be collected to the database. The interval can be configured only in seconds.



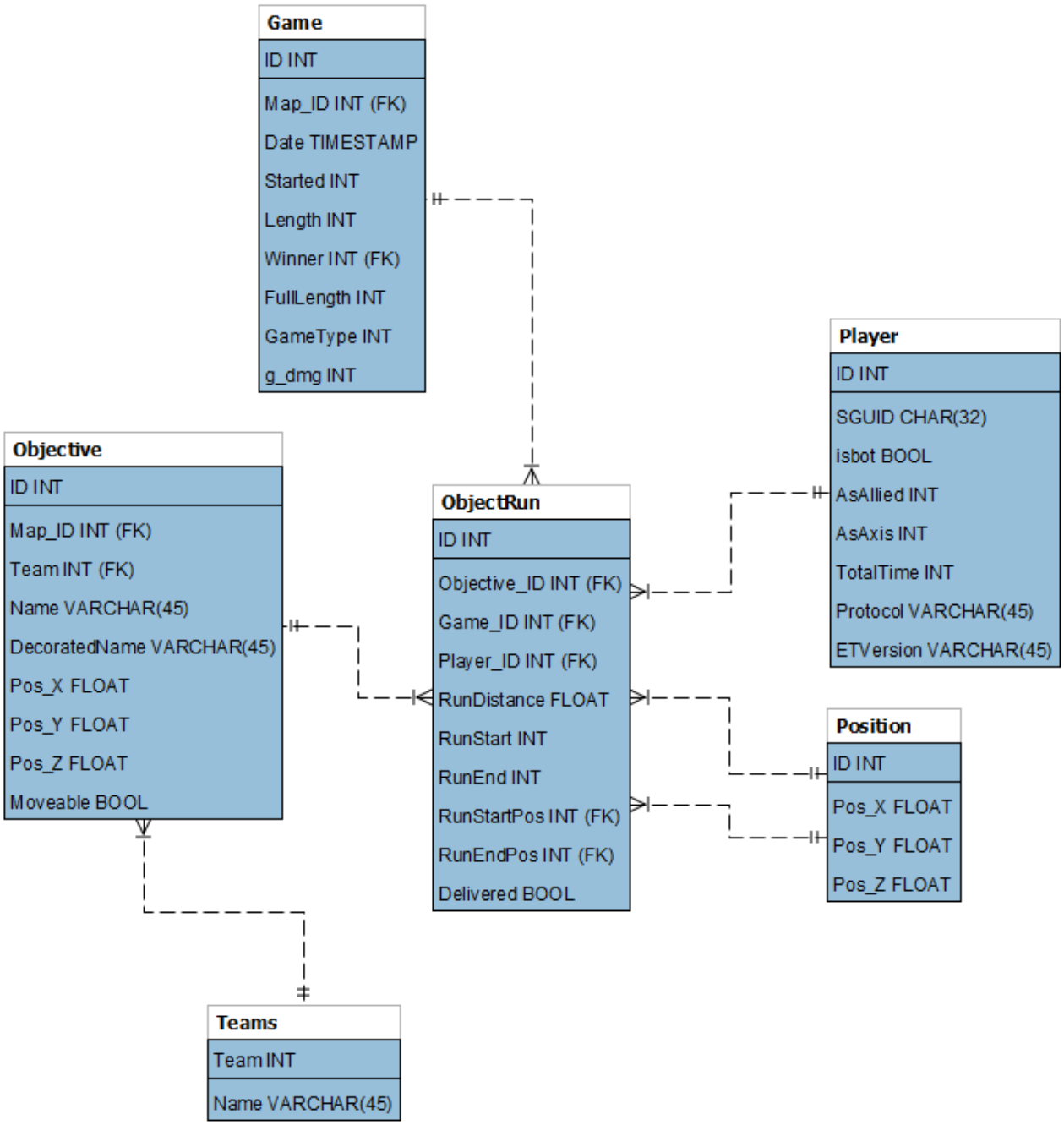
3.4.1. PlayerPosition

This table collects all the positions.

Field	Type	Description
ID	INT	Integer value for the sole purpose to uniquely identify the record.
Game_ID	INT	Foreign key constraint to **Game.ID** to identify the game.
Player_ID	INT	Foreign key constraint to **Player.ID** to identify the player.
LevelTime	INT	Milliseconds from the map start.
Position_ID	INT	Foreign key constraint to **Position.ID**.

3.5. Objective Runs

Objective runs are runs where a single player carries one objective. This objective can be gold or documents for example.



3.5.1. Objective

The objective table is used to store all objectives of all maps. This table is referenced always when something related to objectives is stored into the database.

Field	Type	Description
ID	INT	Integer value for the sole purpose to uniquely identify the record.
Map_ID	INT	Foreign key constraint to **Map.ID** to identify the map.
Team	INT	Foreign key constraint to **Teams.Team** to identify the object target team.
Name	VARCHAR	Uniquely identifying name for the object. There can not be more than one objective with the same **Map_ID**, **Team** and **Name** values.
DecoratedName	VARCHAR	Publicly shown name for the objective. This looks better on outputs than the Name value.
Pos_X	FLOAT	The X coordinate of the objective when put to map in the beginning.
Pos_Y	FLOAT	The Y coordinate of the objective when put to map in the beginning.
Pos_Z	FLOAT	The Z coordinate of the objective when put to map in the beginning.
Moveable	BOOL	If the objective can be moved from its original position, this is a positive value or true, otherwise zero or false.

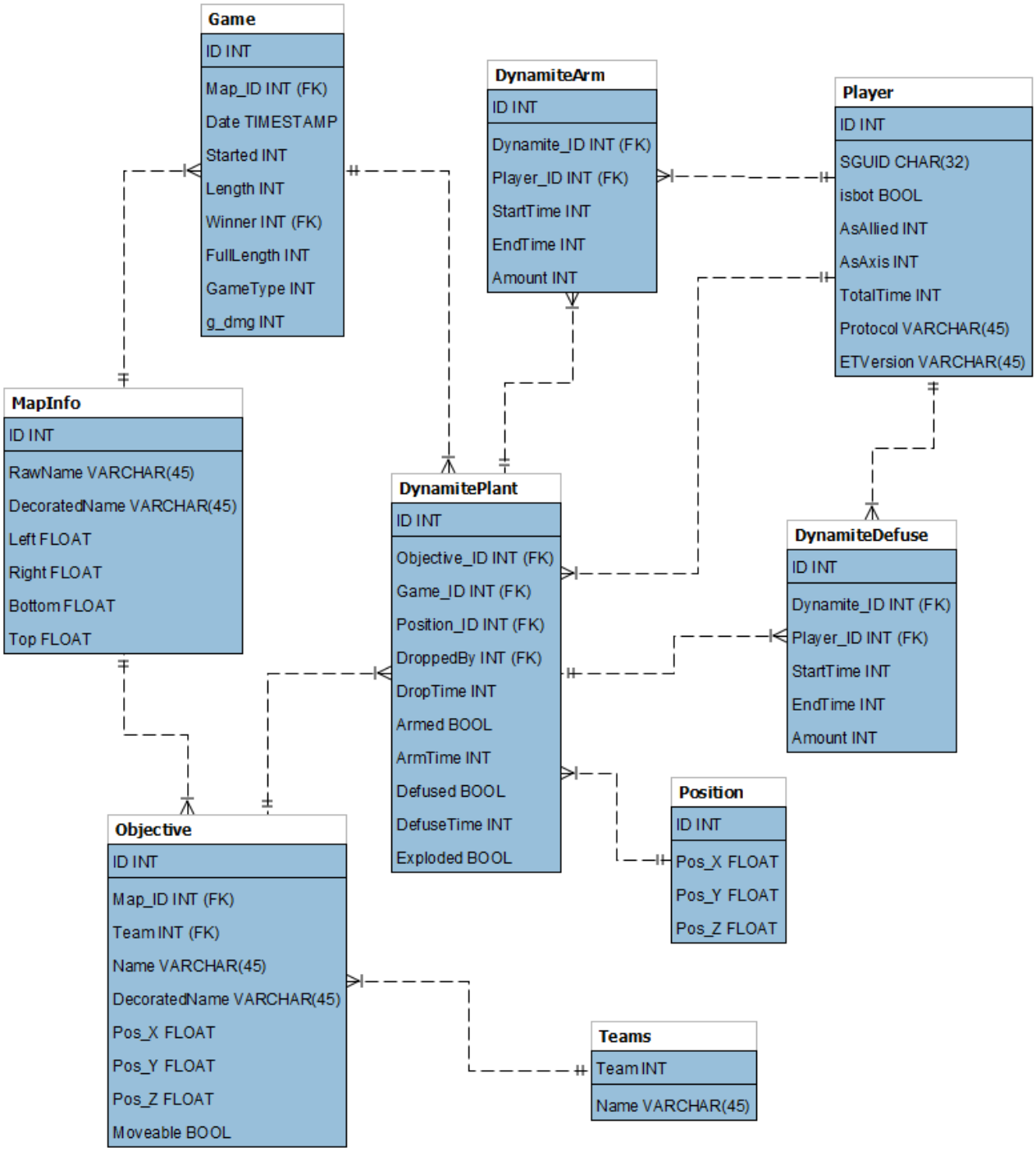
3.5.2. ObjectRun

This table stores all runs made while carrying an objective. The stored run distance is updated on every server frame while running (not to database).

Field	Type	Description
ID	INT	Integer value for the sole purpose to uniquely identify the record.
Objective_ID	INT	Foreign key constraint to **Objective.ID**.
Game_ID	INT	Foreign key constraint to **Game.ID**.
Player_ID	INT	Foreign key constraint to **Player.ID**.
RunDistance	FLOAT	The total amount of traveled length between coordinate points.
RunStart	INT	Millisecond from the map start when player grabbed objective.
RunEnd	INT	Milliseconds from map start when the player dropped objective.
RunStartPos	INT	Foreign key constraint to **Position.ID**.
RunEndPos	INT	Foreign key constraint to **Position.ID**.
Delivered	BOOL	True(or positive value) if the objective was delivered.

3.6. Dynamites

Dynamite drops, plants and defuses can be collected to the database, if they are done on objectives. The tables **Objective** and **Position** are described in Object Runs.



3.6.1. DynamitePlant

This table stores every dynamite drop and plant that can influence an objective. The dynamite is dropped by one player, but it can be armed and defused by multiple players. Information of these actions is collected to **DynamiteArm** and **DynamiteDefuse** tables.

Field	Type	Description
ID	INT	Integer value for the sole purpose to uniquely identify the record.
Objective_ID	INT	Foreign key constraint to **Objective.ID** to identoy the object.
Game_ID	INT	Foreign key constraint to **Game.ID** to identify the game.
Position_ID	INT	Foreign key constraint to **Position.ID**. This position tells the coordinate position of the dynamite.
DroppedBy	INT	Foreign key constraint to **Player.ID**. This tells which player dropped the dynamite.
DropTime	INT	Milliseconds from the map start when the dynamite was dropped.
Armed	BOOL	True (positive value) if the dynamite was armed. 0 or false otherwise.
ArmTime	INT	Milliseconds from map start when the dynamite was armed.
Defused	BOOL	True (positive value) if the dynamite was defused before explosion. 0 or false otherwise.
DefuseTime	INT	Milliseconds from the map start when the dynamite was defused.
Exploded	BOOL	True (positive value) if the dynamite exploded. 0 or false otherwise.

3.6.2. DynamiteArm

This table stores all actions to arm dynamites on objectives. Even if the dynamite didn't get fully armed. Each continuous use of pliers on the dynamite is stored as a single row. A completely armed dynamite can be constructed from several rows.

Field	Type	Description
ID	INT	Integer value for the sole purpose to uniquely identify the record.
Dynamite_ID	INT	Foreign key constraint to **DynamitePlant.ID**.
Player_ID	INT	Foreign key constraint to **Player.ID**.
StartTime	INT	Milliseconds from the map start when the player started using pliers on the dynamite.
EndTime	INT	Milliseconds from the map start when the player stopped using pliers.
Amount	INT	The amount of health points added to the dynamite.

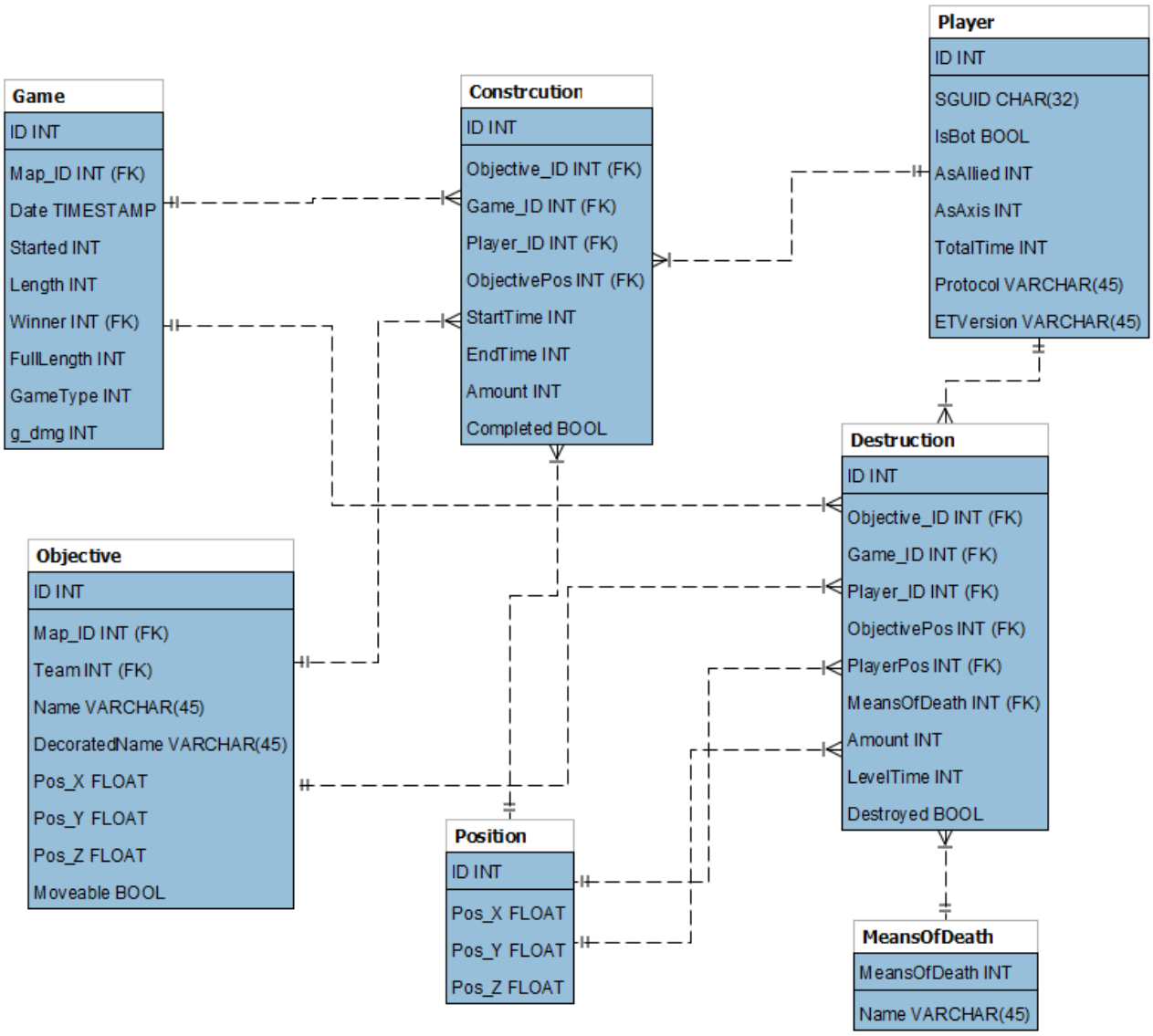
3.6.3. DynamiteDefuse

This table stores all defusing actions done to armed dynamites. One defuse can consist of several players defusing it or one player defusing it in several parts. Do note that health required for dynamite arming and defusing may not be equal.

Field	Type	Description
ID	INT	Integer value for the sole purpose to uniquely identify the record.
Dynamite_ID	INT	Foreign key constraint to **DynamitePlant.ID**.
Player_ID	INT	Foreign key constraint to **Player.ID**.
StartTime	INT	Milliseconds from the map start when the player started using pliers on the dynamite.
EndTime	INT	Milliseconds from the map start when the player stopped using pliers.
Amount	INT	The amount of dynamite health defused.

3.7. Construction and Destruction

The database can store all construction actions to map objectives and also, all destructive actions to map objectives excluding destructing movers such as tanks and trucks.



3.7.1. Construction

This table stores all engineering actions done to objectives. Including fixing movers. One fully constructed object may be constructed in several parts. Table stores also those construction acts that do not result in a fully constructed object.

Field	Type	Description
ID	INT	Integer value for the sole purpose to uniquely identify the record.
Objective_ID	INT	Foreign key constraint to **Objective.ID**.
Game_ID	INT	Foreign key constraint to **Game.ID**.
Player_ID	INT	Foreign key constraint to **Player.ID**.
ObjectivePos	INT	Foreign key constraint to **Position.ID**.
StartTime	INT	Milliseconds from map start when the player started using plier.
EndTime	INT	Milliseconds from map start when the player stopped one continuous use of pliers to the objective.
Amount	INT	The amount of health the player engineered into the objective.
Completed	BOOL	1 or true if the objective was completed with this engineering act, 0 or false otherwise.

3.7.2. Destruction

This table holds all destruction acts with any weapons to all objectives, apart from movers such as tanks or trucks.

Field	Type	Description
ID	INT	Integer value for the sole purpose to uniquely identify the record.
Objective_ID	INT	Foreign key constraint to **Objective.ID**.
Game_ID	INT	Foreign key constraint to **Game.ID**.
Player_ID	INT	Foreign key constraint to **Player.ID**.
ObjectivePos	INT	Foreign key constraint to **Position.ID**.
PlayerPos	INT	Foreign key constraint to **Position.ID**. This is the player position at the very moment of destruction. Not the position of the satchel drop for example.
MeansOfDeath	INT	Foreign key constraint to **MeansOfDeath.MeansOfDeath**.
Amount	INT	The amount of health that was taken.
LevelTime	INT	Milliseconds from map start when damage was made.
Destroyed	BOOL	1 or true if the objective was destroyed by this action, 0 or false otherwise.