Strong Entities:

- **User:** Stores our site's user data. We assume each user will only link to one Riot account.
    - Smurf/Alt users will be treated the same as player smurfs; They will be considered separate unique users
- **Player:** Represents players from Valorant matches. Cannot be a User because not every Valorant player is a user of our site.
    - In the case of Smurf/Alt accounts, we will treat this as a separate unique player
    - Stats might be skewed and more inaccurate but realistically we are unable to accurately and reliably predict if a player is a smurf/ alt and identify which is their main based on their performance stats alone
- **Pro_Player:** A pro player is still a Player, but has more attributes that a normal player won't have on our site (Social media links, Twitch, Pro team, profile picture, etc).
    - We assume that a pro player does not switch teams throughout their career.
- **Game:** Stores information about the game and acts as a way to relate other entities. Date will be a unix timestamp that sorts the games by when it was played.
    - Assumes all games are played on the same patch
- **Player_Stats:** Holds a player's stats for each game that they play
    - Team_side indicates which side the player is playing on depending on the half of the game
    - Each game will result in 2 rows in this table, 1 for attack half and 1 for defend half
    - Overtime rounds will be lumped with their respective side's data depending on the side that team plays during the round
    - Team_id indicates what team the player is playing on, helps for querying due to having 2 rows per game and team_side just keeping track of attacking/ defending
- **Maps:** Holds the average win rates for each map and has a map_name to relate the Blitz.gg data with the Riot API and VLR.gg data. We assume that there are no collisions between map names and we store it as a string.
- **Weapons:** Represents each weapon and their id
- **Agents:** Represents each agent and their id
- **Tiers:** Represents each rank and their id in competitive, named as such due to rank being a reserved keyword in SQL

Weak Entities

- **Agent_Stats:** This contains data about how each Valorant agent performs in a particular map and rank.
    - Requires foreign keys for: map_id, agent_id and tier_id
- **Map_Stats:** Win rate of each map for attacker side depending on rank
    - Requires foreign keys for: map_id, tier_id
- **Weapon_Stats:** Statistics for each weapon depending on the map and rank
    - Requires foreign keys for: map_id, weapon_id and tier_id

One-to-one relationships:
User -> Player (a user can only have one Valorant account linked)

Pro_Player -> Player (a pro player is still a player)

One-to-many relationships:

Player -> Player_Stats

- (each row in Player_Stats represents a game that the player played in)

Game -> Player_Stats

- (for every game, there will be 20 rows in Player_Stats, 2 each for the 10 individual players)

Agent_Stats -> Player_Stats

- Agent_stats is accumulated data mainly from blitz.gg data but we can get some data for it from Player_stats which is from VLR data

Maps -> Agent_Stats, Map_Stats, Weapon_Stats, Games

- Many of the stats are weak entities that require multiple primary keys and foreign keys to uniquely identify the statistics

Weapons -> Weapon_Stats

Ranks -> Weapon_Stats, Map_Stats, Agent_Stats, Player, Player_Stats

Agents -> Agent_Stats

Many-to-many relationships:

None

Functional Dependencies

We chose to normalize using 3NF. Our data thus adheres to 3NF as each non-key attribute is dependent on the primary keys.

For example, we can look at the Player_Stats table. Each non-key attribute (tier_id, agent_id, kills, deaths, assists, kill_assist_trade_survive, combat_score, dmg_per_round, head_shot_ratio, first_blood, first_death) is dependent on all 3 primary keys (player_id, game_id, team_side). To be able to identify the kills, deaths, assists (KDA), for example, we would be required to know the game we would like to access the KDA from, and which player's KDA we would like to access. Then, due to the way we have collected the data, where our stats were collected for each game's half (team_side), the KDA would also be dependent on knowing the team_side.

Player

player_id → riot_id, current_rank

Pro_Player

player_id → twitch, team_name

User

user_id → riot_api_credentials, player_id, pro_lookalike

Player_Stats

player_id, game_id, team_side → team_id, tier_id, agent_id, kills, deaths, assists, kill_assist_trade_survive, combat_score, dmg_per_round, head_shot_ratio, first_blood, first_death

Game

game_id → map_id, date

Agents

agent_id → agent_name

Agent_Stats

agent_id, map_id, tier_id → kills, deaths, assists, win_rate, pick_rate, first_blood, num_matches, q_usage, w_usage, e_usage, r_usage

Maps

map_id → map_name

Map_Stats

map_id, tier_id → win_rate_attacks

Tiers

tier_id → tier_name

Weapons

weapon_id → weapon_name

Weapon_Stats

map_id, tier_id, weapon_id → leg_shot_p, body_shot_p, head_shot_p, dmg_per_round, kills_per_match

Relational Schema

Player_Stats(player_id: int [PK, FK], game_id: int [PK, FK], team_side: varchar(20)[PK], team_id: int, tier_id:int [FK], agent_id: int [FK], kills: int, deaths: int, assists: int, kill_assist_trade_survive: int, combat_score: int, dmg_per_round: int, head_shot_ratio: int, first_blood: int, first_death: int)

Player(player_id: int [PK], riot_id: varchar(20), current_tier_id: int)

Pro_Player(player_id: int [PK], twitch: varchar(20), team_name: varchar(20))

User(user_id: int [PK], player_id: int [FK], riot_api_credentials: varchar(256), pro_lookalike: varchar(20))

Game(game_id: int [PK], map_id: int [FK], date: varchar(20))

Weapons(weapon_id: int [PK], weapon_name: varchar(20))

Weapon_Stats(map_id: int [PK, FK], tier_id: int [PK, FK], weapon_id: int [PK, FK], body_shot_p: float, leg_shot_p: float, head_shot_p: float, dmg_per_round: int, kill_per_match: float)

Agents(agent_id: int [PK], agent_name: varchar(20))

Agent_Stats(agent_id: int [PK, FK] , map_id: int [PK, FK], tier_id:int [PK, FK], kills: float, deaths: float, assists: float, win_rate: float, pick_rate: float, first_blood: int, num_matches: int, q_usage: float, w_usage: float, e_usage: float, r_usage: float)

Maps(map_id: int [PK], map_name: varchar(20))

Map_Stats(map_id: int [PK, FK], tier_id: int [PK, FK], win_rate_attacks: int)

Tiers(tier_id: int [PK], tier_name: varchar(20))