SQL Relational Database Project

League of Legends - Videogame

Pascal Raos - 19335585

Contents of Project Report

Section A: Description of Database Application area and ER Model

Section B Explanation of data and SQL Code:

SECTION A

1. APPLICATION DESCRIPTION

1.1 INTRODUCTION

My Database will be based on the popular videogame “League of Legends” developed and published by Riot Games. Due to the complex nature of videogames, I will be making certain assumptions on the entities I have created that are as true as possible to the underlying mechanics of the game. I will explain these assumptions following a general description of how the game functions independent to my model, and then what aspects of the game I have decided to model.

League of Legends (which I will refer to for the remainder of this report in its colloquial form, “League”) is categorized as a Mobile Online Battle Arena (MOBA) style videogame. MOBAs typically function as a competitive team-based player-vs-player (PVP) experience.

Since many demonstrators/lecturers may not be intuitively familiar with videogame knowledge compared to real life systems, Sections 1.2, 1.3, and 1.4 are here to explain in extensive knowledge the core gameplay of League of Legends. Not all aspects of the game will be modelled in my relational database as there is an excessive number of things to choose from. Section 1.5 onwards will talk about the specific features pertaining to my project.

1.2 CHAMPION DESCRIPTION

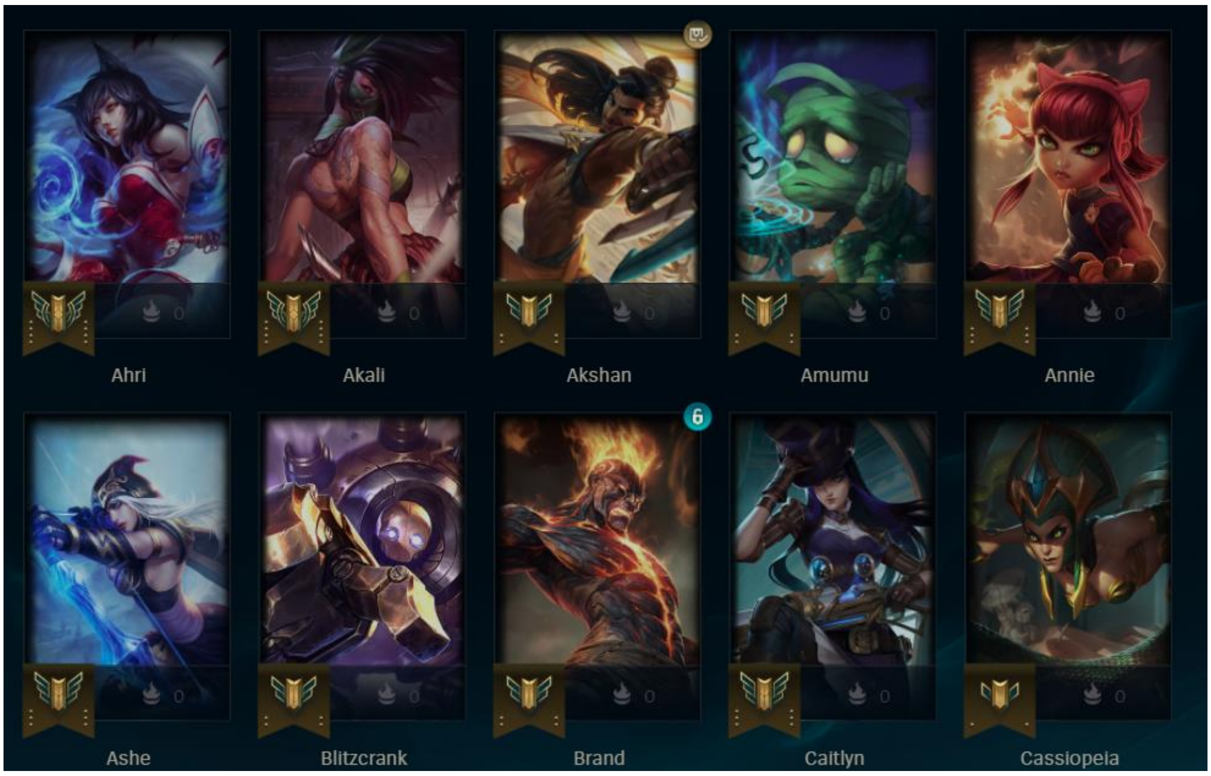


Figure 1 Shows 10 playable champions from the game

In League, two teams of five players compete against each other using characters known as champions. Each champion has its own unique name, a difficulty arbitrarily assigned to it, and five abilities commonly denoted by which button you press on your keyboard to activate them. The five abilities are a passive ability that is always active, a Q ability, a W ability, an E ability, and an R ability. By the specific implementations strategies undertaken by the developers of League, each character's abilities are unique, i.e. no other champion has the exact same ability (however some may be similar in certain ways).

1.3 GAMEPLAY DESCRIPTION

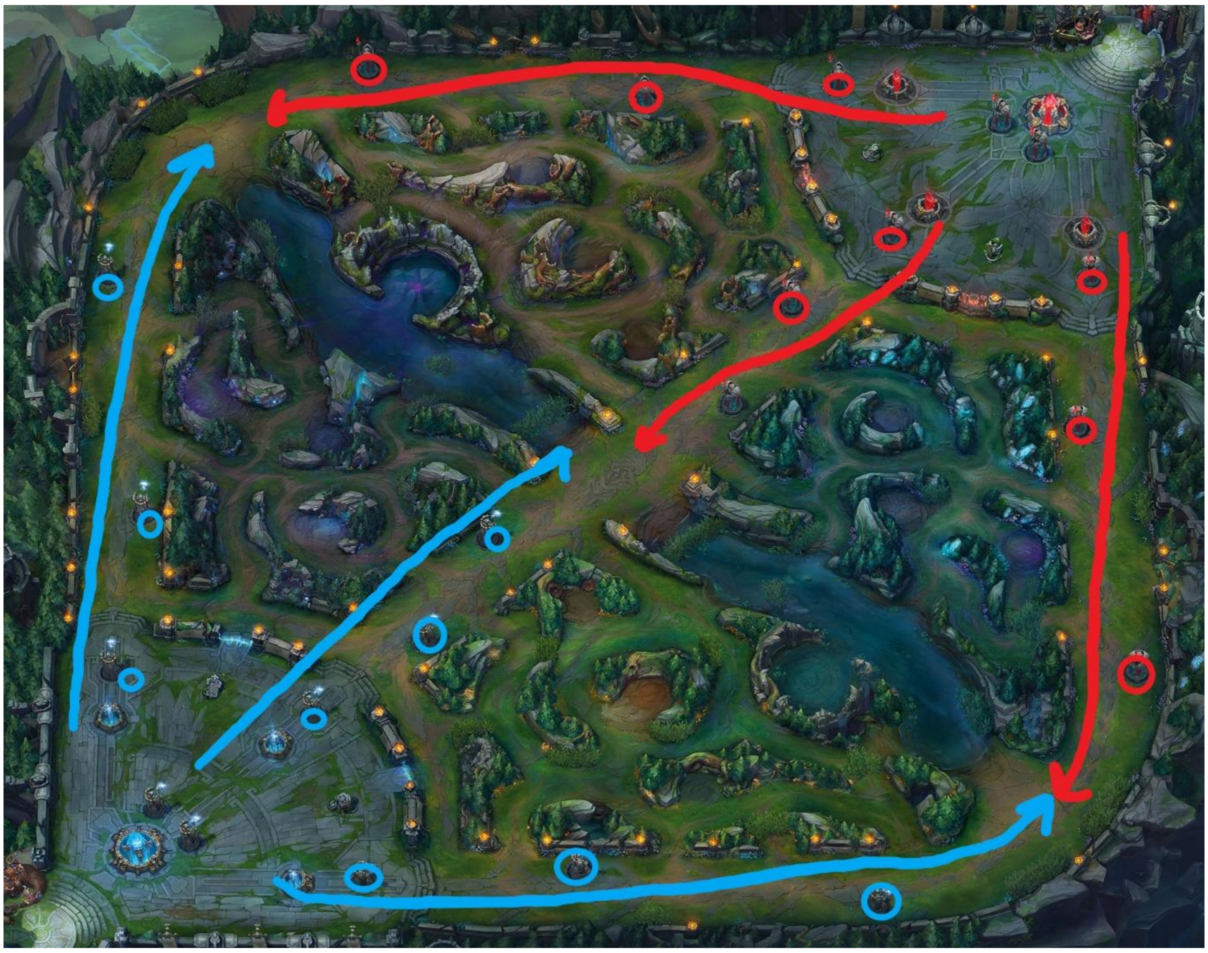


Figure 2 The coloured circles represent the position of each teams' towers (red or blue team), and the arrows represent the movement of the minions as they spawn in each teams respective base, the minions will meet at a common front at the centre of each lane

Each individual game is played on an almost symmetrical map, with three distinct lanes running down the centre and sides of the map. Each team has a main base that the opposing team is trying to destroy, and extending outward from that base along the lanes are three rows of powerful towers that protect the base from the enemy team by dealing heavy damage to enemies in their proximity.

In order to reach the enemy base, each team's base continuously produces a non-playable character (NPC) known as a minion. These minions are here to be the focus of a tower's fire, so that the player can attack the tower without receiving any harm. In order for the minions to reach the opposing tower, the player must kill/destroy the opposing teams minions so that there are no opposing minions to stop the player's from advancing forward. Once you kill a minion you increment your Creep Score value (CS, counts how many minions you have killed)

The goal of each game is to destroy enough towers in any or all lanes and attack the opponents main base until it is destroyed, at which point that team wins the game. Essentially it is a tactics based game where you must distribute the five players in your team along the three lanes in such a manner that your towers are not destroyed, and the enemy does not advance to your main base.

If a player kills another player, the player deficit can tip the scales of map control, and make it harder for the enemy team to protect all three lanes simultaneously. After a player is killed by another player, they must wait a set amount of time before they respawn (come back to life) at their team's start location (top right or bottom left of the map, depending on which team you are on).

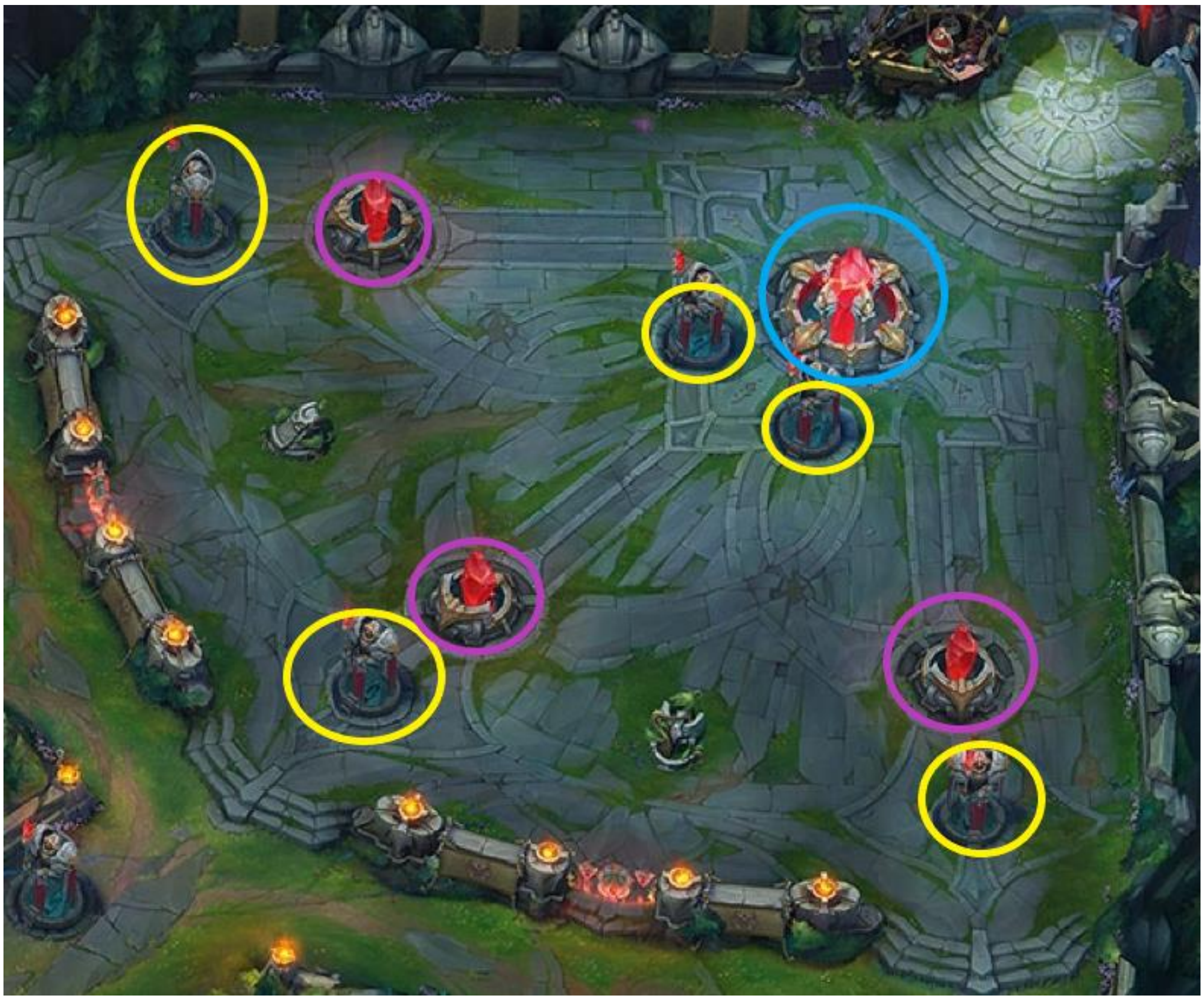


Figure 3 Shows a magnified view of Red Team's base. Towers are circled in yellow, the nexus (the main base, the thing that must be destroyed to win the game), and in yellow, inhibitors.

In total there are three main structures on the map to destroy, towers, inhibitors, and the nexus (The enemy base). These structures must be destroyed in a linear fashion, where only the outward most structure can be attacked, with the inward ones being un-targetable (invulnerable to damage). To make it easier to finish games once the main base's outer towers have been breached, inhibitors when destroyed cause the opposing team's minion to be more powerful, making it harder for them to be killed before they reach the central base, ensuring there is constant pressure by the enemy team.

1.4 GOLD, ITEMS AND RUNES

Where the competitive nature of the game shines the most, is in the way that a players skill counts towards their ability to win the game. This is done through the gold and item system.

Essentially at the start of the game, a player's goal is to kill the opponent and the opponent's minions. Killing their minions means their own minions get closer to the enemy's tower, allowing the player to hit the enemy tower without the tower attacking them. Killing a player similarly allows the player to attack the enemy tower without the other player attacking them.

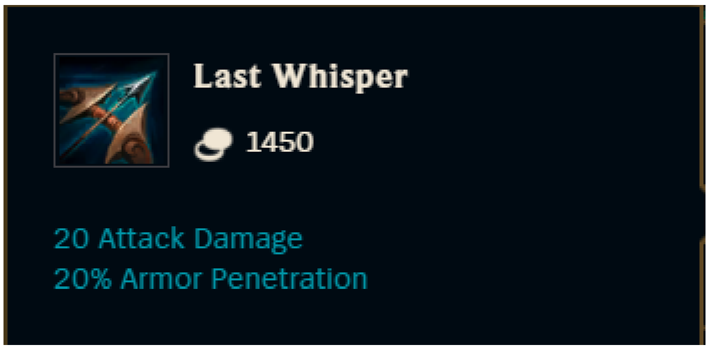
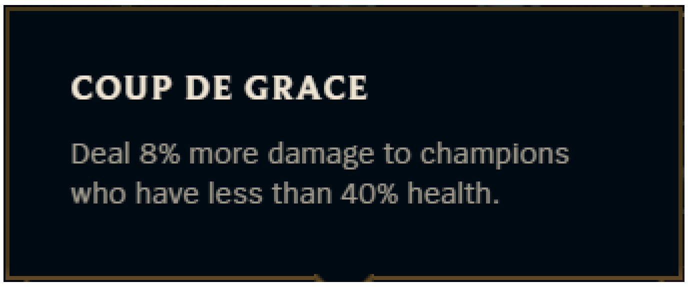

Last Whisper
1450
20 Attack Damage
20% Armor Penetration

Figure 4 An example of a simple item, providing extra damage and pierces enemy armour, the cost of the item is listed as 1450 gold

Once a player kills a minion or another player, they receive a currency known as gold. A player may return to their base where they can purchase items with their accrued gold. Items augment the player's attack values (how much damage they do) or provide them with some unique utility (Moving quickly, making the opponent move slower, etc). If a player has killed more minions and players than people on the opposing team, they become much stronger as they can purchase more/better items than the opposing team, making it easier again to kill the opponents, and thus getting stronger again. The appeal in the game comes from outplaying an opponent who may be stronger (have more items than you), or by being so strong yourself that you essentially can kill all five of the opposing players without the help of your teammates.



COUP DE GRACE

Deal 8% more damage to champions who have less than 40% health.

Figure 5 Example of a simple rune

Runes are similar to items in that they provide the player with some sort of augmentation to their damage or provide some sort of unique utility. However, unlike items, you choose which runes you want to use prior to the game starting. Each player chooses a "Rune Page" before the game begins.

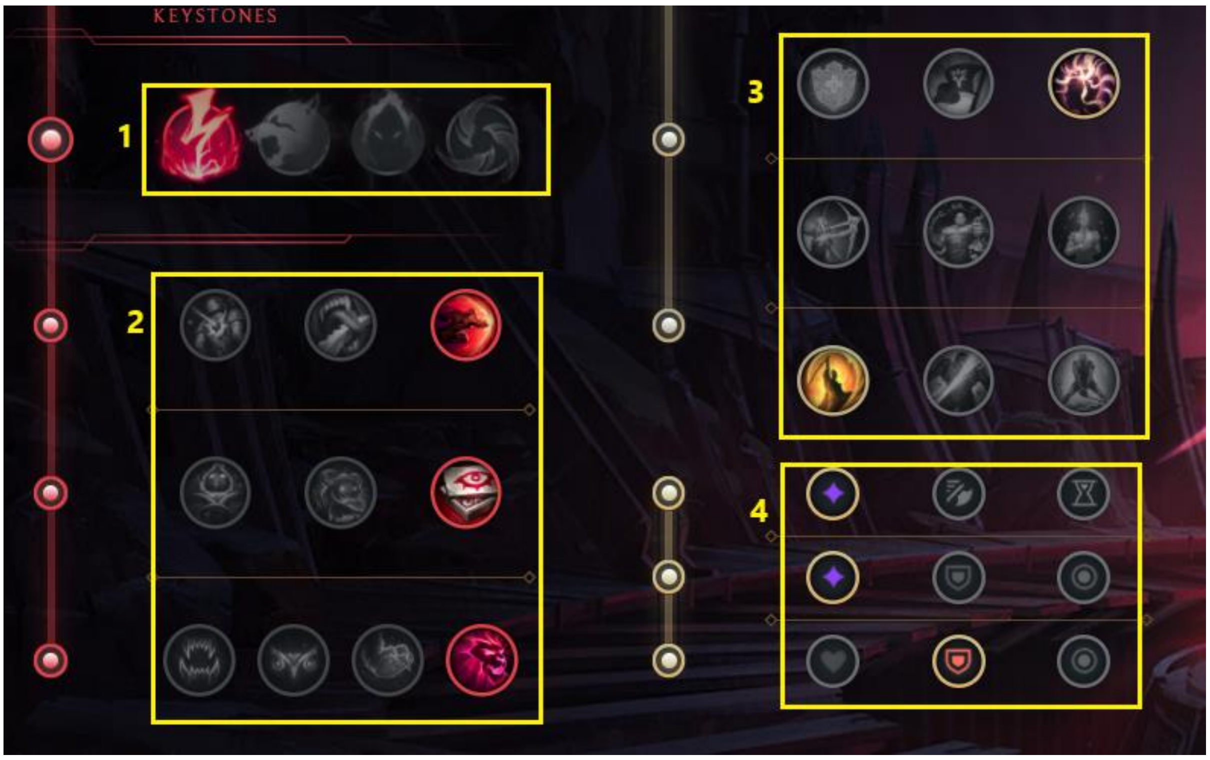


Figure 6 Shows at (1) 4 Primary Runes from a chosen path. (2) Shows three rows of sub-runes that relate to the primary path. (3) shows two sub-runes that relate to a path that isn't from the primary rune. (4) Show 3 augments that are independent of either path.

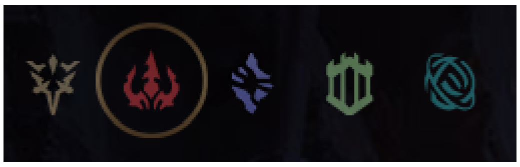


Figure 7 Shows the five possible rune paths for the primary rune. The secondary sub-runes are chosen from the remaining four unchosen paths. Path names from left to-right are Precision, Domination, Sorcery, Resolve, Inspiration. Notice a general colour scheme associated with the paths and the sub-runes in figure 6.

Each page has a one of four strong primary rune from a specific rune path, and three weaker sub-runes associated with the primary rune's path. Besides this, players also choose two sub-runes that must be from a different primary rune to the first one chosen (called secondary runes), and they must also choose three small augments that are independent of the primary runes. Players can only choose one sub-rune and augment per row, with the exception of the secondary runes, where you choose 2 but they are also row limited.

1.5 MY APPLICATION DESCRIPTION

In this section I will talk about the aspects of the game that I have decided to model. Firstly, I decided not to model anything that occurs in real-time during the progress of the game, as it is misguided to model things that are constantly changing as the modelled tables may contain lots of NULL values. Instead, I decided to model the aspects of the game that are recorded once the game is complete.

The game's software is split into two parts, a launcher that requires you to log-in and contains many of the social and purchasing aspects related to the game, and then the game itself, which after queuing to find 9 other people to play with, opens and allows the user to begin playing the game.

I have decided to model a person's account, the champions that a person can play in the game, the runes that augment a player's capabilities, a player instance, which is an instance of one player playing a game, and finally an instance of the game itself. The exact specifics of these entities is discussed in the following sections of the report

2. ENTITY RELATIONSHIP DIAGRAM

Since Player Instance is such a central entity in the diagram, it will be explained last after the others are already described.

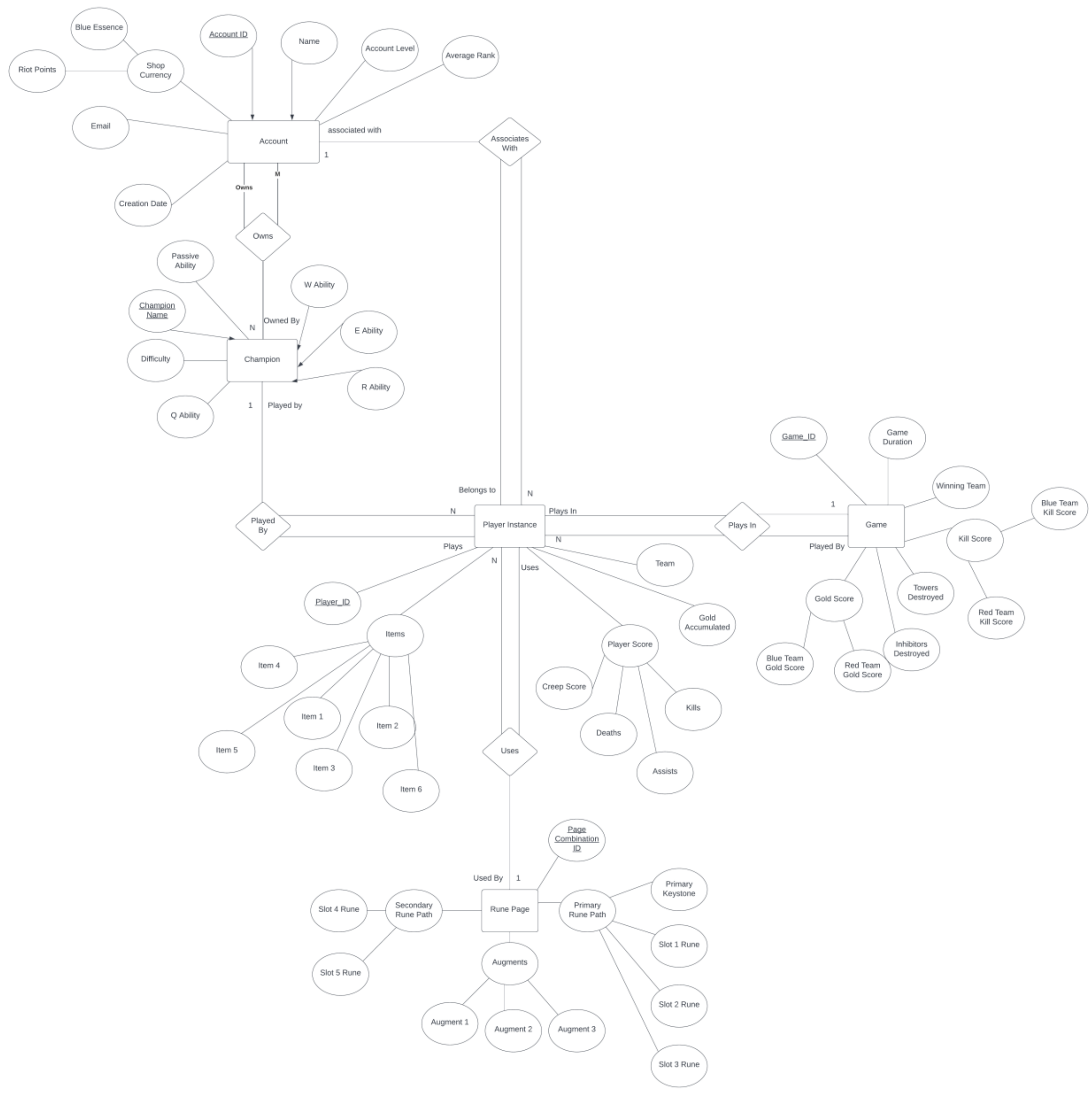


Figure 8 The above is an Entity Relationship Diagram for the 5 things I have decided to model about League of Legends. These 5 combined with the M-to-N relationship between Account and Champion will create 6 relational tables. Due to its size, you may need to zoom in to see the diagram. The diagram was also uploaded separately as a PNG image, and it may be easier to look at that while reading the rest of this section of the report.

2.1 ACCOUNT

The account entity is the entity that models the fundamental information surrounding a persons League of Legends account. This account contains basic details such as the email address used to create the account, the username chosen at creation, and assigns a unique account ID.

Each account can level up and receive rewards by playing the game, and those that play a more competitive ranked mode will have their average rank displayed. Accounts must purchase additional champions with a currency called blue essence (earnable by playing the game), otherwise they are limited to the few champions they receive upon creating their account. A more premium currency called Riot Points is obtainable by spending real world money, and allows Accounts to purchase cosmetic features to apply to their champions.

I have decided not to model these cosmetic features do to the relationship between them and the Account entity being identical to the relationship between Account and Champion. Like stated before, an account can purchase additional champions with blue essence, to which they now permanently own that champion. **Each account can own N champions, and each champion can be owned by M accounts**

2.2 CHAMPION

Each champion has its own unique name. Also by the design of the game, no two champions share the same ability, therefore all five champion abilities, the passive, Q, W, E, and R abilities are unique to each champion. The difficulty attribute is arbitrarily decided by the game developers and has no direct dependency on a champions abilities.

League of Legends has 141 unique champions, of which each account can own all of them by purchasing them with in-game currency. Since every player (N) can own all (M) champions, there is an M-to-N relationship between Champion and Account entities.

2.3 RUNE PAGE

As previously stated in section 1.4, a Rune Page consists of a primary rune/keystone chosen from a rune path, and three sub-runes (slots 1-3) associated with the primary one’s path. Also two additional sub-runes (slots 4-5) are chosen that are associated with a primary rune from a path that was not already chosen.

Since a path cannot uniquely identify a primary rune and its three chosen sub-runes, nor a secondary rune's two chosen sub-runes, it cannot be a candidate key. A primary rune cannot uniquely identify its sub-runes either due to different primary runes can use the same sub-runes.

Finally, since the augments are independent of the other runes, it seems that there is no way to uniquely identify tuples in the rune table with the current information. Initially I thought that maybe the Rune Page entity can be a weak entity, dependant on the player that choses the runes, however since different players can have identical rune pages, even their relationship to a player cannot uniquely identify them.

Instead, since there is a limited number of combinations for primary and secondary runes combined with their sub-runes and augments, I can assign a unique rune combination ID as the primary key to each combination, and a player would have this rune combination ID as a foreign key in their tuple if they intend to use a specific rune combination.

Since each player instance will use one rune page, but each page can belong to N players, there is a 1-to-N relationship

2.4 GAME

The Game entity is an entity that describes the attributes related to a game of League. Please recall from sections 1.2 to 1.4 the specifics of how a game progresses.

At the end of each game there are certain features that are recorded about the game state. These are the duration of the game, the game ID, the team that won the game, the number of towers destroyed, the number of inhibitors destroyed, the gold accrued by each team, and the number of kills accrued by each team. For each game there are ten players (five per team), therefore there is a one-to-many relationship between the Game and Player Instance entities

2.5 PLAYER INSTANCE

Player Instance is the most complex entity in the model, as it is also the most important aspect of the League of Legends model. The players themselves are associated in every aspect of the game process.

A player instance can be thought of as one occasion in which an account plays a game of league. Therefore there is a one-to-many relationship between account and player instance, since one account can have multiple instances where they play. There is a one-to-many relationship between champion and player instance, since one champion is played by each player. Similarly there is a one-to-many relationship between rune page and player instance, since each player uses one rune page. Finally, there is a one-to-many relationship between game and player instance since a player instance is unique to one game, and one game has many players.

A notable assumption I have made is that even if a single account plays the same champion, uses the same rune page, gets the same score, and purchases the same items in two separate games, they are treated as separate player instances due to being in separate games (the Player_ID primary key will be different).

Another assumption is that only one copy of each champion can be played in any one game, this follows exactly how the real game works, preventing the boredom of potentially playing against the champion you yourself have chosen.

The final assumption is that each account can only have one player instance per game. This means that in a game of 10 people, all 10 people are unique individuals.

3. MAPPING TO RELATIONAL SCHEMA

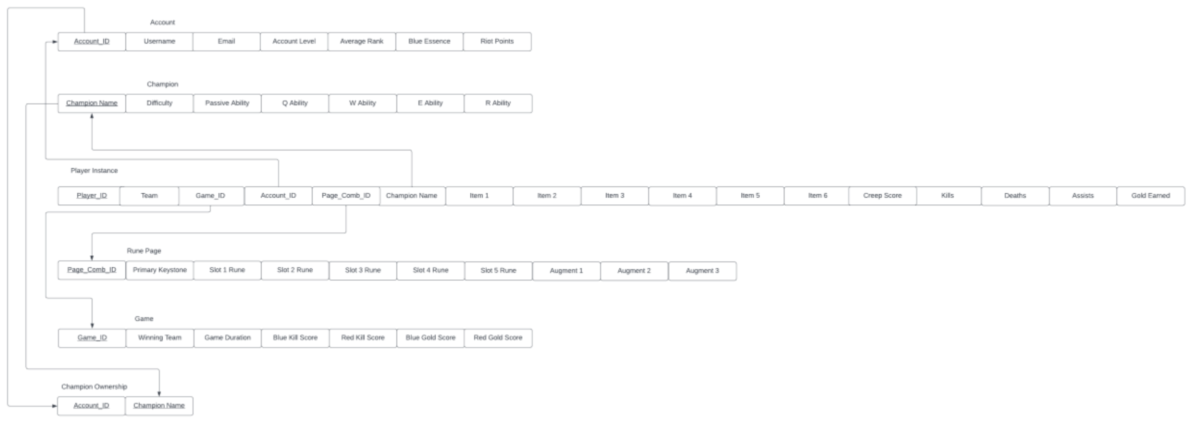


Figure 9 The above describes the mapping of the entity relationship diagram to tables based on the cardinality of relationships between each one. Again you may need to zoom in or refer to the standalone image attached in the submission

3.1 ACCOUNT

The account entity has no direct changes done to it from the logical mapping to the relational schema but does have an indirect one in the Champion Ownership table and will be explained in that section.

3.2 CHAMPION

The champion entity has no direct changes done to it from the logical mapping to the relational schema, but does have an indirect one in the Champion Ownership table, and will be explained in that section

3.3 CHAMPION OWNERSHIP

The champion_ownership table describes the M-to-N relationship discussed in the entity relationship diagram section between the champion and account entities. Since many accounts can own many champions, a separate table is required with two attributes being the primary key (The primary key from each related table), as them individually cannot identify the other.

3.4 RUNE PAGE

The rune page entity has no direct changes done to it from the logical mapping to the relational schema.

3.5 GAME

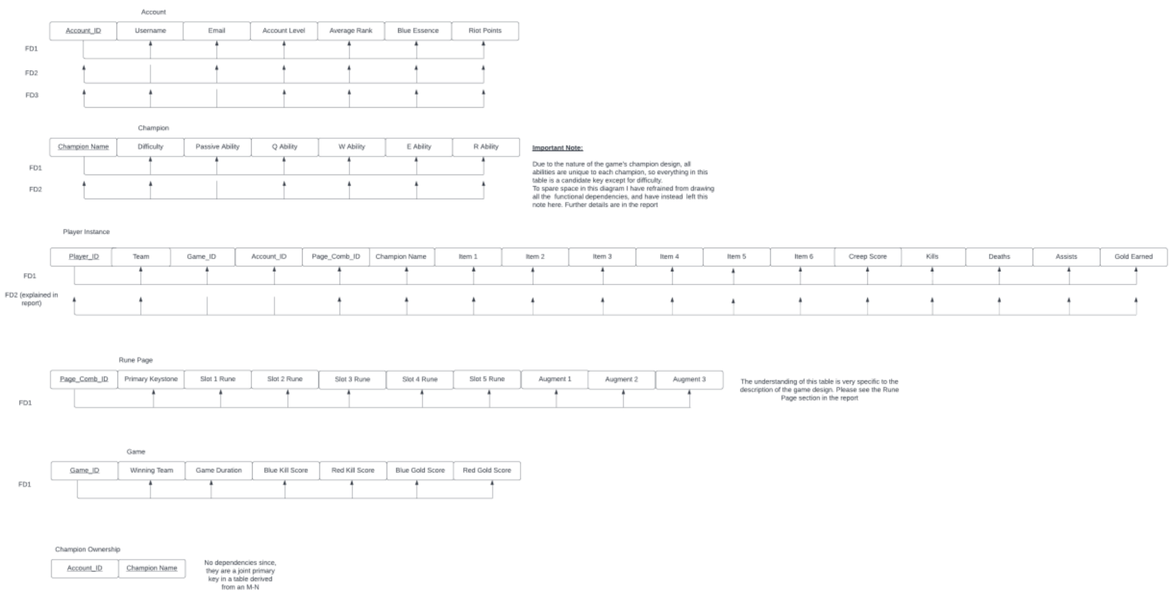
The game entity has no direct changes done to it from the logical mapping to the relational schema.

3.6 PLAYER INSTANCE

Player instance is the entity changed the most by the mapping to the relational schema, this is because it is on the N side of four one-to-many relationships between Account, Game, Rune Page, and Champion.

The primary keys of Account, Game, Rune Page, and Champion, are placed as foreign keys in the Player Instance table, as they are what makes up the necessary information to determine a player instance

4. FUNCTIONAL DEPENDANCY DIAGRAM



4.1 ACCOUNT

The account table has three distinct functional dependencies. FD1 describes the standard relationship between the primary key and the tuple. FD2 describes it using the Username, and FD3 describes it using the email of the account. Since the game developers have made it so that you can only have one account per email, every username is unique, and the account_id is already stated as unique, all three of these attributes are superkeys, and therefore these functional dependencies do not cause any table modification issues.

4.2 CHAMPION

FD1 states the dependency between the primary key and the rest of the attributes. As previously stated in the report and also within the Functional Dependency Diagram, all champions have unique abilities and therefore almost every attribute in the champion table can uniquely identify a tuple in the champion table (superkeys). The only exception is the difficulty attribute, but as there is no dependency between the difficulty in uniquely identifying the other attributes, and all other dependencies do not cause table issues, the current state is acceptable

4.3 PLAYER INSTANCE

FD1 states the dependency between the primary key and the rest of the attributes. FD2 is interesting in the way that given a Game_ID and an Account_ID, a specific player instance can be uniquely identified due to the two assumptions that have been made, that in a specific game each player is unique (Account_ID), and that each Player_ID is unique to each game (Game_ID).

To simplify since there are N players in each game, Game_ID cannot identify a specific player alone. Also since each Account can have multiple Player_IDs, Account_ID cannot identify a specific player. But a combination of Account_ID, and Game_ID can uniquely identify a Player Instance tuple.

I'll be honest, but I could not find any example online of what to do when two foreign keys together can act as a candidate key. Do I do some normalization on the Player Instance table and separate stuff out? It seems that the table would lose clarity if I took out any of these three keys

4.4 RUNE PAGE

FD1 states the dependency between the primary key and the rest of the attributes. There are no other attributes that are dependant on each other as I have stated in section 2.3 when explaining the roles of the individual Primary runes/keystones

4.5 GAME

FD1 states the dependency between the primary key and the rest of the attributes. There are no other dependencies since no other attributes can uniquely identify the others. You could make some inference between a team's kill and gold scores and what team won, but there are many exceptions where a team had lower scores than the other and still won the game

4.6 CHAMPION OWNERSHIP

There are no functional dependencies here since the table is simply a table where two attributes act as a single primary key

SECTION B – SQL

5.1 Explanation of Table creation

```
CREATE TABLE PlayerInstance (

        Player_ID int NOT NULL,
        Team varchar(255) NOT NULL,
        Game_ID int NOT NULL,
        Account_ID int NOT NULL,
        Page_Comb_ID int NOT NULL,
        ChampionName varchar(255) NOT NULL,
        Item1 varchar(255),
        Item2 varchar(255),
        Item3 varchar(255),
        Item4 varchar(255),
        Item5 varchar(255),
        Item6 varchar(255),
        CreepScore int DEFAULT 0,
        Kills int DEFAULT 0,
        Deaths int DEFAULT 0,
        Assists int DEFAULT 0,
        GoldEarner int DEFAULT 0,

        PRIMARY KEY (Player_ID),
        FOREIGN KEY (Game_ID) REFERENCES Game(Game_ID),
        FOREIGN KEY (Account_ID) REFERENCES Account(Account_ID),
        FOREIGN KEY (ChampionName) REFERENCES Champion(ChampionName),
        FOREIGN KEY (Page_Comb_ID) REFERENCES RunePage(Page_Comb_ID)

);
```

Figure 10 Code for the Player Instance Table

The above code describes the creation of the PlayerInstance table modelled after the Player Instance entity. I chose this one as it has the most substance in it constrain wise. Game_ID, Account_ID, Page_Comb_ID, and ChampionName are all foreign keys from their respective tables.

The table also makes use of the DEFAULT keyword in assigning default values of 0 to the attributes that relate to scoring in the game, as you will always begin with a score of zero in these categories

6. Explanation of Table altering

```
UPDATE Account
SET AccountLevel = AccountLevel + 1
Where Account_ID = 1;
```

This code alters the account by incrementing their level. This is in essence when an account levels up after acquiring enough experience from playing games

7. Explanation of Trigger

```
CREATE TRIGGER RemoveChampion
BEFORE DELETE ON Champion
FOR EACH ROW
    SIGNAL SQLSTATE '2234' SET MESSAGE_TEXT = "Cannot remove champions from the game";
```

This trigger operation prevents a person from removing a champion from the game. This is done due to the microtransaction model that league of legends uses. Players buy cosmetics for champions, therefore if you remove a champion you are effectively stealing money from players.

8. Explanation of View creation

```
CREATE VIEW SkilledPlayerInstance AS
        SELECT Account_ID, ChampionName, Kills, Deaths
        FROM PlayerInstance
  WHERE (Kills > 10) AND (Deaths < Kills/2);
```

This code is an example of a view that intends to find instances in which a player performed well. It gets moments where they have more than 10 kills but also only died less than half of the times they acquired kills. Therefore they have a kill to death ratio of > 2:1. It also returns the champion they played.

9. Explanation of Table Population

```
INSERT INTO Account
VALUES (1, "fungamer", "test@email.com", 120, "Bronze", 5000, 340),
        (2, "richard12", "test2@email.com", 10, "Iron", 5060, 0),
        (3, "sam223", "saw2@email.com", 87, "Gold", 6012, 45),
        (4, "xwadah23", "myemail@email.com", 403, "Diamond", 14568, 1980),
        (5, "proplayer", "unique@email.com", 653, "Challenger", 6022, 140);
```

The table above describes the insertion of rows into the Account table. Each account has an account_ID, a username, an email, their account level, their average rank, their total blue essence, and their total riot points. (all of these are explained in previous sections of the report).

10. Explanation of Table retrieve/joins

```
SELECT Account.Account_ID, PlayerInstance.Player_ID
FROM Account
JOIN PlayerInstance ON Account.Account_ID=PlayerInstance.Account_ID
```

The code above joins together all Accounts with every Player Instance they have played. Essentially shows them the ID of every time they have played. You could apply some summation technique to find out how many times each person has played a game.