USE CASE STUDY REPORT

Group No.: Group 5

Student Names: Anyuan Xu Zichen Zhou

Executive Summary:

Access to match history is crucial for game designers and players to design games with analyzable data and improve gaming performance. The primary goal for this study was to design a match history database of League of Legends, the most popular online video games in the world. With the development of video games and analyzable data requirements from increasing players and designers. More and more after-game applications are designed for players. Therefore, using a relational database can apparently enhance the efficiency of players to check their game records(death, kill, assistant, damage, money) etc. In order to achieve it, we retrieved nearly 1000 data from Riot game API, including the player's name, level, Icon, the creation time of the match, game mode, map, and important statistics of player(kills, deaths, assists, champion used, Damage dealt, money giant) etc.

The database was modelled taking most basic requirements of data records by every player in League of Legends after each game, along with input including the player's name, level, Icon, the creation time of the match, duration of the match, game mode, map mode, map_id, etc.

Data cleaning and preparation are essential because the original identities of match and player are way too complicated due to several game servers in the world and multiple API versions used at the same time. After data preparation, we built the EER and UML diagrams and relational models. Next, we imported data into MySQL as well as architected the schema and linked each table by foreign key. Finally, we connected our database with Python to do some analysis and visualization to find out the top 5 hottest champions and who is the best player as well as each player's strengths and weaknesses.

I. Introduction

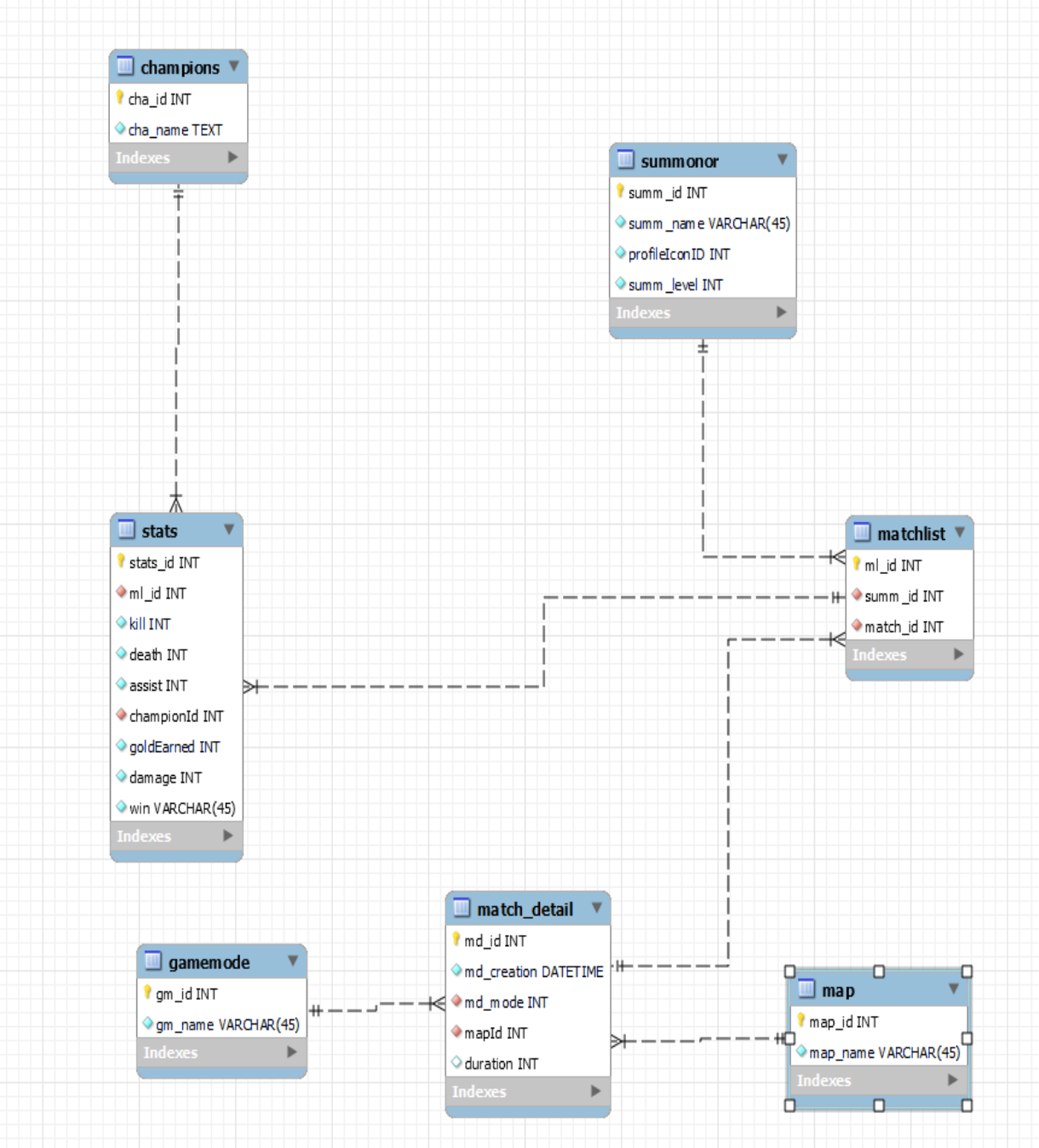
League of Legends (LoL), commonly referred to as League, is a 2009 multiplayer online battle arena video game developed and published by Riot Games. In the game, two teams of five players battle in player versus player combat, each team occupying and defending their half of the map. Each of the ten players controls a character, known as a "champion", with unique abilities and differing styles of play. During a match, champions become more powerful by collecting experience points, earning gold, and purchasing items to defeat the opposing team.

There are many websites and applications that gather data from Riot game API and present it to users for checking their Summonor, Live Spectate and even real-time stats. The most popular one is OP.GG. This website allows you to find match history, champion stats, summoner data, game items, champion builds and even analyzed data presented visually. Users can easily access these features by just typing in summonors' names.

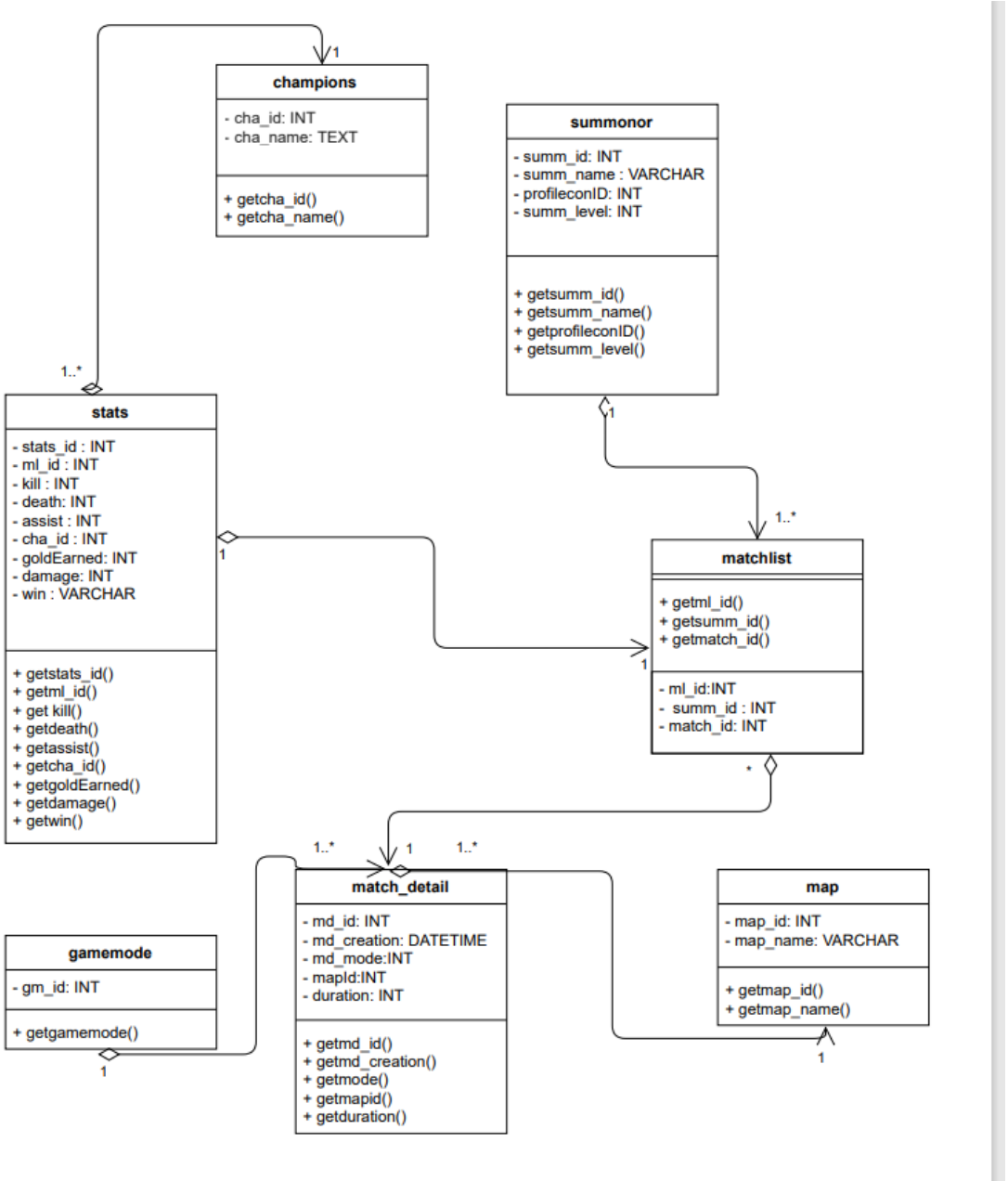
In this study, we aim to establish a match history database which contains information about player(in LOL called summonors) name, profileIcon, and their game level,besides, backend of a match history website so that we could build a website in the future to complete this project and make it landed. It is necessary to introduce the Riot game API. Due to the large number of players and stats we retrieved from it, we can not accept that all, so we only include some players that are my friends in the game and a subset of stats that can represent the ability of one player. Python does the rest of the job of storing data into dataframe and writing it into a .xlxs file.

II. Conceptual Data Modeling

1. EER Diagram



2. UML Diagram



III. Mapping Conceptual Model to Relational Model

Primary Key- Underlined **Foreign Key- *Italicized***

Champions(cha_id, cha_name)

Stats(stats_id, *champion_id*, *ml_id*, kill, death, assist, goldEarned, damage, win)

FOREIGN KEY champion_id refers to champion_id in Champions; NULL NOT ALLOWED

FOREIGN KEY ml_id refers to ml_id in matchlist; NULL NOT ALLOWED

Summonor(summ_id, summ_name, profileIconID, summ_level)

Match_detail(md_id, *md_mode, mapId,* md_creation, duration)

FOREIGN KEY md_mode refers to gm_id in gamemode; NULL NOT ALLOWED

FOREIGN KEY mapId refers to map_id in map; NULL NOT ALLOWED

Matchlist(ml_id, *summ_id*, *match_id*)

FOREIGN KEY summ_id refers to summ_id in summoner; NULL NOT ALLOWED

FOREIGN KEY match_id refers to md_id in match_detail; NULL NOT ALLOWED

Map(map_id, map_name)

gamemode(gm_id, gm_name)

IV. Implementation of Relation Model via MySQL and NoSQL

MySQL Implementation

The database was created in MySQL and the following queries were performed:

Query 1: Find matches that achieve 10 plus kills by a player named '88Rising'. Retrieve player name, champion name, kill and map.

```
select s.summ_id, s.summ_name, c.cha_name, stats.kill, map.map_name
from summonor s, stats, matchlist m, champions c, match_detail md, map
where s.summ_id = m.summ_id and m.ml_id =stats.ml_id
and m.match_id = md.md_id
and md.mapId = map.map_id
and stats.championId = c.cha_id
and summ_name = '88Rising' and stats.kill > 10
order by 4 desc;
```

summ_id	summ_name	cha_name	kill	map_name
1	88Rising	Renekton	33	summonor's rift
1	88Rising	Lucian	16	summonor's rift
1	88Rising	Kennen	12	summonor's rift
1	88Rising	Jinx	11	summonor's rift

Query 2: Find the top 5 hottest champions.

```
select c.cha_name as 'top 5', count(stats.championId) as 'frequency'
from stats, champions c where c.cha_id =  stats.championId
group by c.cha_name order by count(stats.championId)
desc limit 5
```

top 5	frequency
Nautilus	7
Gragas	4
Pantheon	4
Yuumi	4
Blitzcrank	3

Query 3: Calculate the total WinRate for each player.

```
select s.summ_id, s.summ_name,
CONCAT((sum(case when s1.win = 'TRUE' then 1 else 0 end)/count(s1.win))*100, '%') as 'win rate'
from summonor s, matchlist m, stats s1
where s.summ_id = m.summ_id and m.ml_id =s1.ml_id group by s.summ_id;
```

summ_id	summ_name	win rate
1	88Rising	60.0000%
2	yogurtQAQ	70.0000%
3	Annes	60.0000%
4	cudii	60.0000%
5	Fight for mount	50.0000%
6	Lalafell	60.0000%
7	Vita C	60.0000%
8	After March	50.0000%
9	ChaosBerzerker99	40.0000%
10	Dubblelift	60.0000%

Query 4: Calculate total KDA((kill+assist)/death) for each player.

```
select s.summ_id, s.summ_name, (sum((s1.kill+s1.assist)/s1.death)/count(*)) as 'kd/a'
from summonor s, matchlist m, stats s1
where s.summ_id = m.summ_id and m.ml_id =s1.ml_id
group by s.summ_id;
```

summ_id	summ_name	kd/a
1	88Rising	3.85180927
2	yogurtQAQ	3.63333333
3	Annes	4.18350816
4	cudii	3.40698413
5	Fight for mount	3.80500000
6	Lalafell	5.10841270
7	Vita C	4.22361111
8	After March	3.87857143
9	ChaosBerzerker99	3.73000000
10	Dubblelift	3.70563404

Query 5: Calculate the total farming score(goldEarned/matchDuration/game mode weighting factor) for each player.

```
select s.summ_id, s.summ_name,
(case when md.mapId = '1' then sum((s1.goldEarned/md.duration)/1000) else sum((s1.goldEarned/md.duration)/1500) end) as 'Farming score'
from summonor s, matchlist m, match_detail md, stats s1
where s.summ_id = m.summ_id and m.match_id = md.md_id and m.ml_id =s1.ml_id group by s.summ_id;
```

summ_id	summ_name	Farming score
1	88Rising	6.47948509
2	yogurtQAQ	2.42720963
3	Annes	7.06942703
4	cudii	5.25043761
5	Fight for mount	2.90746461
6	Lalafell	5.16219713
7	Vita C	3.35585743
8	After March	3.07526177
9	ChaosBerzerker99	3.10051069
10	Dubblelift	5.19545419

Query 6: Calculate total damage conversion(damage/goldEarned) for each player along with map name.

```
select s.summ_id, s.summ_name, (sum(s1.damage)/sum(s1.goldEarned)) as 'damage conversion rate', map.map_name
from summonor s, matchlist m, match_detail md, stats s1, map
where s.summ_id = m.summ_id and m.match_id = md.md_id and m.ml_id =s1.ml_id and md.mapId = map.map_id
 group by s.summ_id, map.map_name
 order by s.summ_id;")
```

summ_id	summ_name	damage conversion rate	map_name
1	88Rising	3.9587	howling abyss
1	88Rising	5.6655	summonor's rift
2	yogurtQAQ	2.4606	howling abyss
2	yogurtQAQ	3.9541	summonor's rift
3	Annes	3.8700	howling abyss
3	Annes	5.8263	summonor's rift
4	cudii	3.7372	howling abyss
4	cudii	5.5475	summonor's rift
5	Fight for mount	5.2468	howling abyss
5	Fight for mount	6.4262	summonor's rift
6	Lalafell	3.3507	howling abyss
6	Lalafell	4.6303	summonor's rift
7	Vita C	3.9736	howling abyss
7	Vita C	2.8656	summonor's rift
8	After March	4.4430	howling abyss
8	After March	2.7904	summonor's rift
9	ChaosBerzerk...	6.6272	howling abyss
9	ChaosBerzerk...	4.8594	summonor's rift
10	Dubblelift	6.0849	howling abyss
10	Dubblelift	7.0679	summonor's rift

NoSQL Implementation

Three collections(Match, Stats, Summoner) have been built in the MongoDB database.
Summoner collection stored players' information such as name, level, ID.
Stats collection stored performance information for each player in each game.
Match collection stored map, datetime information and linked summoners and stats.

Query 1: Calculate total KDA((kill+assist)/death) for each player.

```
db.match.aggregate([{
    $lookup: {
        from: 'summonor',
        localField: 'summonor ID',
        foreignField: '_id',
        as: 'summonor'
    }
}, {
    $unwind: {
        path: "$summonor"
    }
}, {
    $lookup: {
        from: 'stats',
        localField: 'stats ID',
        foreignField: '_id',
        as: 'stats'
    }
}, {
    $unwind: {
        path: "$stats"
    }
}, {
    $addFields: {
        "kill": "$stats.kill",
        "death": "$stats.death",
        "assist": "$stats.assist",
        "name": "$summonor.summ_name",
        "kd": {
            $add: ["$stats.kill", "$stats.assist"]
        }

    }
}, {
    $project: {
        name: 1,
        kda: {
            $divide: ["$kd", "$stats.death"]
        }
    }
}, {
    $group: {
        _id: "$name",
        kda: {
            "$avg": "$kda"
        }
    }
}, {
    $sort: {
        "kda": -1
    }
}])
```

```
{'_id': '88rising', 'kda': 7.75}
{'_id': 'yogurtQAQ', 'kda': 6.133333333333333}
{'_id': 'Annes', 'kda': 3.6717948717948716}
{'_id': 'cudii', 'kda': 2.803030303030303}
```

Query 2: Calculate average damage each player dealt in one game.

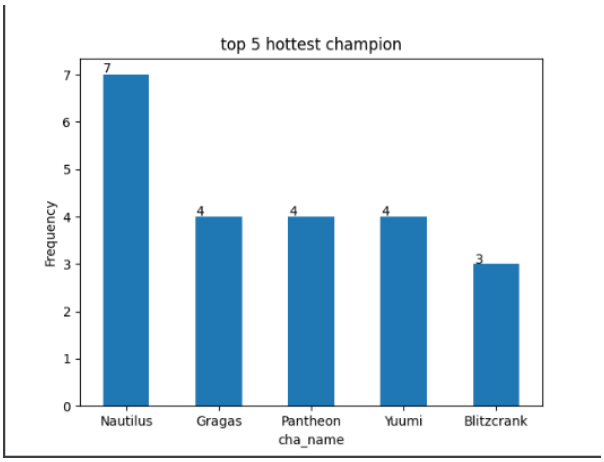
```
db.match.aggregate[{
    $lookup: {
        from: 'summonor',
        localField: 'summonor ID',
        foreignField: '_id',
        as: 'summonor'
    }
}, {
    $unwind: {
        path: "$summonor"
    }
}, {
    $lookup: {
        from: 'stats',
        localField: 'stats ID',
        foreignField: '_id',
        as: 'stats'
    }
}, {
    $unwind: {
        path: "$stats"
    }
}, {
    $addFields: {
        "name": "$summonor.summ_name",
        damage: "$stats.damage"
    }
}, {
    $project: {
        "name": 1,
        damage: 1
    }
}, {
    $group: {
        _id: "$name",
        dmg: {
            "$avg": "$damage"
        }
```

```
{'_id': 'cudii', 'dmg': 16402.666666666668}
{'_id': '88rising', 'dmg': 11015.0}
{'_id': 'Annes', 'dmg': 10648.666666666666}
{'_id': 'yogurtQAQ', 'dmg': 7398.333333333333}
```

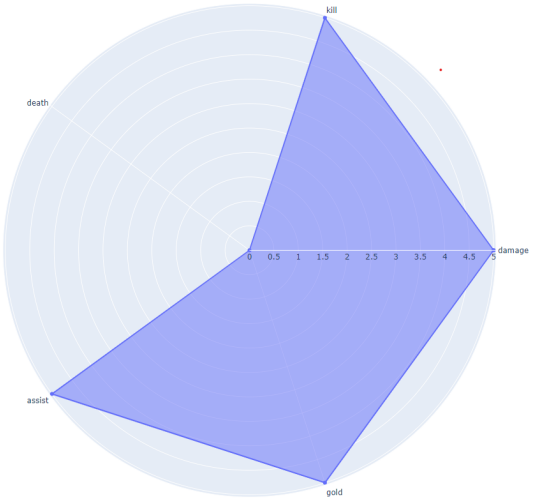
V. Database Access via R or Python

We used Python as our analysis programming language. *mysql.connector* package was imported to connect to the MySQL database, then passed sql query into cursor.execute(query) and stored the result in a list, transforming the list into dataframe at the end. For visualization, we used matplotlib and plotly packages.

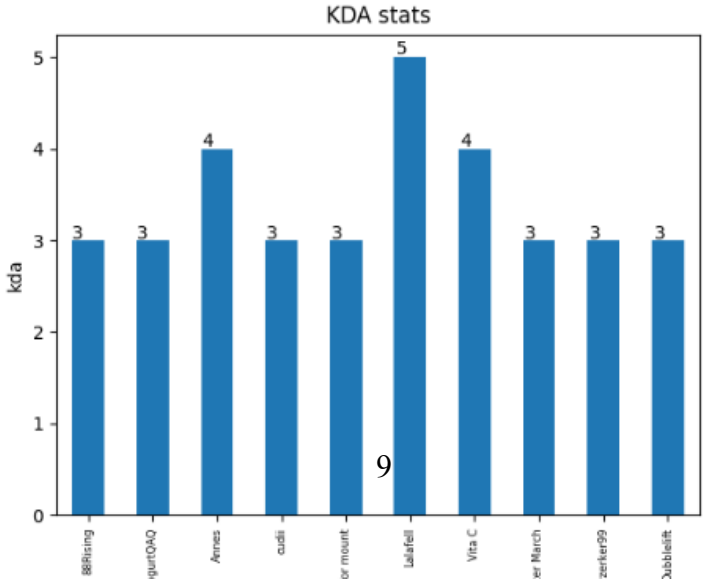
Graph 1 : Top five Hottest League of Legends Champions



Graph 2: Radar chart for League of Legends players (88Rising)



Graph 3: KDA Stats for each League of Legends player



VII. Summary and recommendation

Large-scale online video games like League of Legends have tens of millions of players all over the world. It is very important for each player to have an opportunity to access a match history database with comprehensive information and complete functions through websites or mobile applications. Players can obtain game information through the database to have a clear insight of their skills and then make improvements. Game developers can get the whole picture of the performance of a particular object. Is the new champion popular? Which champion is popular in all ranks? These are the questions game developers are trying to answer through analyzing stats in the match history database. The match history database we have built and data access and analytics applications we have created using python are useful and ready to use for League of Legends players and development teams.

There are still some known shortages and flaws which could be improved in the future. Taking the Riot game database as a reference, our database is relatively simple in volume aspect and complexity aspect. For volume, we can add more players and their match history records in our database since we only had 10 players and a total 100 match history records right now. For complexity, we can add more tables such as items, spells, Icon and more columns like double kill, lane, turret destroyed for instance. There is still much data we didn't use due to the limit of time and page limit. We can do much more analytics in the future to perfect this project.

In the course of building MySQL and MongoDB databases, I found it is difficult to scale for a traditional relational database, especially for building a massive database and scaling thousands of concurrent gamers, analyzing user statistics. In contrast, NoSQL has flexible schemas which allow data scientists to easily make changes to database as new requirements occur. Queries are also fast in NoSQL as there is no need to join tables which saves a lot of time for users. In general, I recommend using NoSQL to build a League of Legends database.