# Project Direction Overview

I want to develop a website called LeagueAlytics that provides League of Legends players a tool for analyzing and improving at the game. LoL (League of Legends) is currently the most popular online video game in the world, and I have been playing it myself for almost a decade. It is extremely competitive and players who are passionate about the game are always looking for ways to improve. I believe that if people are willing to pay hundreds of dollars an hour to be coached, or watch thousands of hours of gameplay, then a website providing player analysis would be helpful. Currently, to the best of my knowledge, this sort of application does not exist or is behind a pay wall. There are some websites that show a person’s match history, but the data is minimal, and no analysis is given. I would like to offer players something that not only provides them with data but information about their good and bad tendencies.

First, I want to give a brief and simplified overview of how the game works and identify a few key metrics. LoL is played by two teams of five unique champions, or characters, with the objective being to destroy the other team’s nexus, or base. Defending each team’s nexus are structures that must first be destroyed. Killing opponents, destroying their structures, and conquering neutral objectives grants gold, levels, and buffs. Gold, levels, and buffs translate into increased combat stats which make it easier to win more fights, destroy enemy structures and conquer more neutral objectives.

Let’s say a player has been losing a lot of games recently and they cannot seem to figure out what is going wrong. They open their web browser, navigate to my website and login to their account which is tied to their LoL account. They are then shown an analysis of their performance and stats for each unique champion they have played. For instance, this person might notice that while playing a certain champion they lose 60% of their games while on another they win 55%. Now they may think to themselves, “I’ll stop playing X and start playing more of Y.” However, if they wish to improve on the champion they are losing with, then they have a few options to investigate the cause. One would be to compare the stats they have with their 55% win rate champion. In doing so they may find that on average they have double the deaths and one-third less gold. Now different champions have different strengths and weaknesses so perhaps they would prefer a comparison to how the average player does on the same character instead. In this analysis they may find that their average end of game level is lower, and they have half the number of kills. Either analysis, or both, can help the player identify the areas where they are underperforming and should focus on improving.

My initial impression is that the database hierarchy should be based around the user and then break down by each unique champion, with match specific information joined from other tables. I am sure as time progresses; I will want to add additional metrics for analysis so designing a database that allows for additional fields to be included will be a priority. I am also thinking players should be able to have some access to other accounts for comparison.

# Use Cases and Fields

Account Signup

1. Person visits LeagueAlytics website.
2. Website asks them to create account.
3. The user enters in administrative information.
4. They are then asked to link this account to their LoL account.

The database will need to store both the account information used to login to LeagueAlytics as well the LoL account that is associated with it.

|  |  |  |
| --- | --- | --- |
| **Field** | **What it Stores** | **Why it’s Needed** |
| AccountName | This field stores the unique account name used to sign in to LeagueAlytics. | For a person to review their analysis we must first know who we are reviewing. |
| Password | This field will store the password of the created accrount. | The password will ensure that the only the intended person has access to their account anlaysis. |
| Email | This field will store their email address. | Email addresses are often used as a backup if one forgets their password or username to sign in. |
| LolAccount | This field will store their LoL account name. | This will be used to connect this LeagueAlytics account to the LoL user creating the account. |
| Region | The field stores the region the LoL account is connected to. | This is important because no two account names can exist in the same. Two accounts can have the same name if on different regions however. |

Once a person has created an account, and connected their LoL account to it, their LoL match history will be pulled into the database as well as future games played. Since the analysis will be for each unique champion, we’ll need to start with information about said champion.

Data retrieval and monitoring.

1. Person visits LeagueAlytics website.
2. They then create an account and connect their LoL account.
3. Their match history data is then retrieved and stored into the database.
4. The website also will track any new match data that is created once a player plays additional games.

|  |  |  |
| --- | --- | --- |
| **Field** | **What it Stores** | **Why it’s Needed** |
| ChampionName | The name of the champion being played. | Used to connect the match data to the correct character for analysis |
| ChampionRole | The role the champion is played in (Top, Jungle, Mid, ADC, Support) | Champions can be flexed into other roles and they particular role they are played in dictates the best practices that should be followed in game. |
| ChampionType | The category that the champion belongs to (Tank, Fighter, Mage, Assassin, Marksmen, Enchanter) | Similar to the role field, this field will further identify the best practices needed to succeed in play. |
| ChampionPeak | This field will identify what point in the game the champions is said to be at it’s most powerful (Early game, Mid game, Late game) | Following the reasoning of the two fields above, knowing when a champion is most powerful helps player identify when they should play more aggressive and leverage their peak. |
| ChampionHealth | This field will store the base health given to a champion at the beginning of the game. | Health points are assigned to every champion are essential to know and understand when in combat. |
| ChampionAttack | This field will store the base attack given to a champion at the beginning of the game. | Similar to health, attack damage is important to know when in combat and also when fighting with neutral monsters, minions (enemy AI monsters) and destroying structures. |

Once we have captured data about the champion being played in a match, the next thing we would like to store is how this champion performed in said match. This will help in identifying which tendencies correlate to more wins and which correlate to more loses.

|  |  |  |
| --- | --- | --- |
| **Field** | **What it Stores** | **Why it’s Needed** |
| WinOrLose | This field will store the result of the match played (Win or Lose) | Winning is the objective metric used to determine how good someone plays a game. If you beat everyone you play in tennis you are not only good at tennis you’re the best there is. |
| MatchKills | This field will store the total amount of enemy champions a player has slain. | This field has variable important depending on the champion being played. Regardless, it is rarely ever bad if you play a match and have lots of kills. |
| MatchDeaths | This field will store the total amount of times a player has died. | This field is important to keep track of because unlike kills that have variable importance it is never good to die. There are numerous reasons why this is true so at the very least this must be tracked. |
| MatchAssists | This field will store the total amount of enemy champions a player has assisted in slaying. | Assists are important for players piloting, or playing, champions whose purpose is to help their allies get ahead of the enemy. Assists have an inverse variable importance to kills. Either you should be securing kills as a player or helping your team to do so. |
| MatchCS (Creep score) | This field will store the total amount of enemy AI a player has killed. | CS, or creep score, is an important metric for many champions and roles outside of the Support role. The reason it is so important is because it is the main income for a player to gain gold. |
| MatchGold | The field will store the total amount of gold gained in by a player in a match. | Gold is the reward given to a player for killing enemies, their structures, and or assisting in either of the above. Player can use the gold they have earned to buy items which increase their desired stats. |
| MatchRole | The field will store the role a player is assigned in a match (Top, Jungle, Mid, ADC, Support). | The role a player is assigned in a game essentially dictates the responsibilities they are expected to perform. |

As a final data source to base the analytics off it is important that the items a player decides to purchase be captured. Being mechanically good, meaning a player’s ability to play their champions to its peak potential, is important in being successful at the game. However, it is equally important that the player know which items best compliment their champion to further leverage its strengths. The following data will additionally need to be stored in the database.

|  |  |  |
| --- | --- | --- |
| **Field** | **What it Stores** | **Why it’s Needed** |
| ItemName | This field will store the name assigned to a particular item. | This is needed to uniquely identify the item and distinguish it for others. |
| ItemActive | This field will hold either a value of yes or no to identify whether it has an active on it. | Knowing whether an item has an active is an important consideration when deciding to buy it or not. If a player is playing an assassin for example, an item active that grants an extra 30% damage to enemies for 3 seconds would complement their inherent ability to do a lot of damage quickly. |
| ItemType | This field will store the type of buff to stats an item falls into (Defensive , Offensive, Supportive) | This field is important to keep track of because unlike kills that have variable importance it is never good to die. There are numerous reasons why this is true so at the very least this must be tracked. |
| ItemCost | This field will store the amount of gold needed to buy said item. | Comparing the cost of items is important when deciding which to choose. If a player is piloting a champion who is strong early, they might be inclined to buy items that are cheaper so that they can leverage their early advantages sooner. |
| ItemMythic | This field will store whether or not an item is a mythic item or not (Yes/No) | Mythic items are a special class of item, only one is allowed per champion, that once purchased give additional stats based on other non-mythic purchased. Deciding which mythic a player chooses is arguably the most important item decision they make so it is worthy of identifying. |

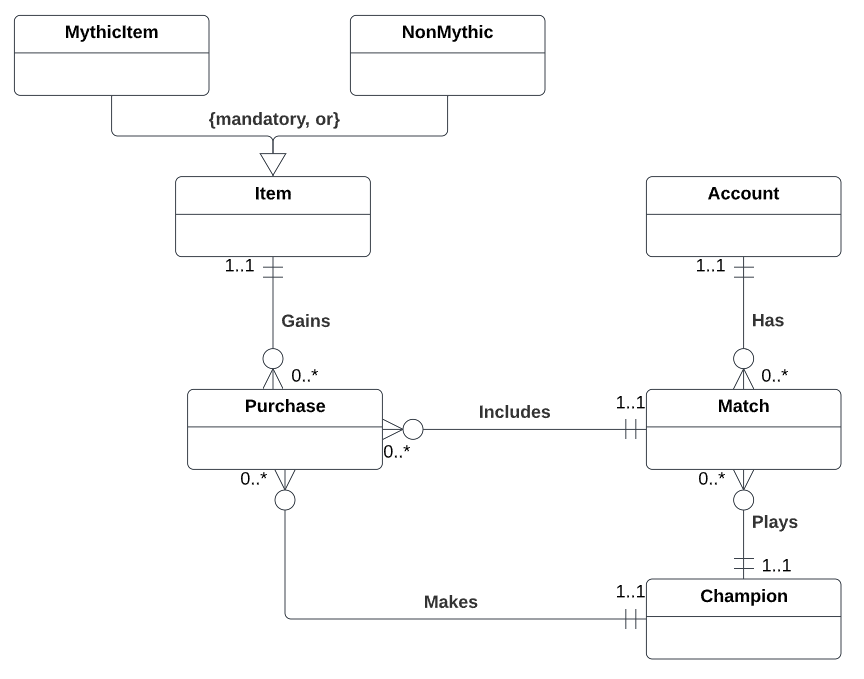
Now that we have stored all the following data above the application can support any number of questions asked of it. Use case questions that utilize the data above include:

* How many games did I win when I played Katarina and I bought the Hextech Rocketbelt mythic item?
* On average, how many kills a game do I secure when playing Akali?
* What tank champions do I play the most?
* How much total gold do I have on average when my CS is at least over 100?

# Structural Database Rules

1. An account may have many matches, each match is had by an account.
2. A champion may play matches, a match is played by a champion.
3. A match may include many purchases, a purchase is included in a match.
4. A champion may make many purchases, each purchase is made by a champion.
5. A purchase gains an item, an item may be gained by a purchase.

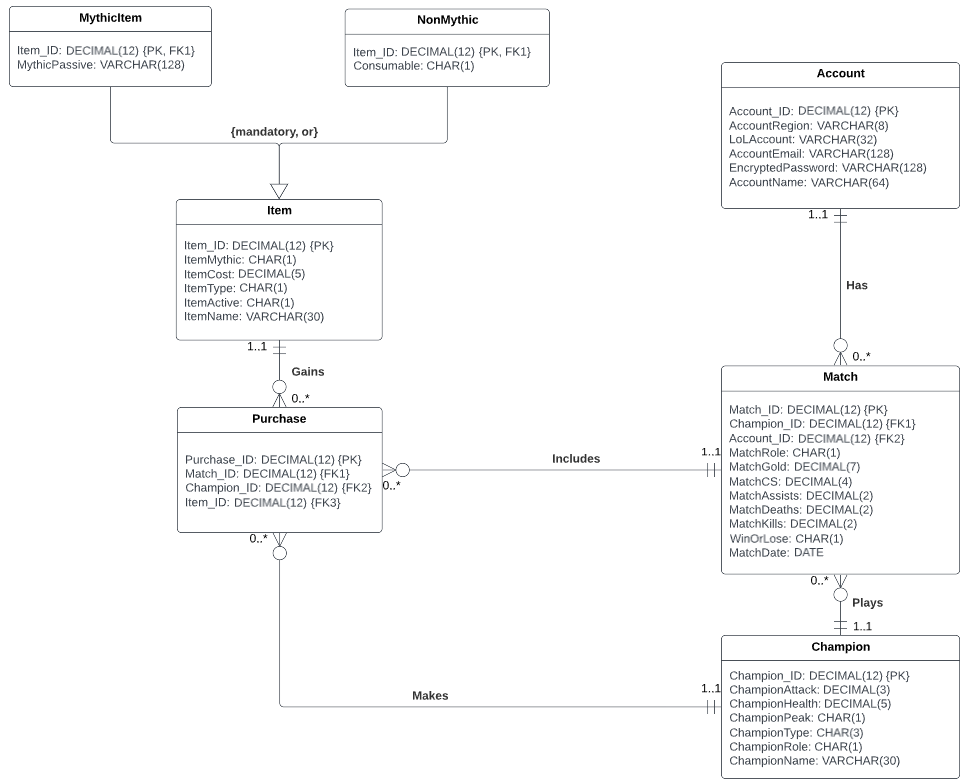
# Conceptual Entity-Relationship Diagram



# Full DBMS Physical ERD

The following are the attributes necessary to allow my application to provide players with a rich and comprehensive analysis of their play.

|  |  |  |  |
| --- | --- | --- | --- |
| **Table** | **Attribute** | **Datatype** | **Reasoning** |
| Account | AccountName | VARCHAR(64) | Any player who wishes to make an account will have an account name associated to it. This will be used to login to the website. I have allocated 64 characters to this field to allow for millions of users to create unique and meaningful names. |
| Account | EncryptedPassword | VARCHAR(128) | Every account created will need a password to login. It is best practice to encrypt passwords in a database, and since for PGP encryption 1024 bits is sufficient, I have allocated 128 characters of space. |
| Account | AccountEmail | VARCHAR(128) | This field will hold the account email. In case the user has forgotten their password and needs to reset it a link to do so will be sent here. Most email addresses are usually a combination of a person’s first name, last name, date of birth and email provider domain I image 128 characters should suffice. |
| Account | LoLAccount | VARCHAR(32) | This field will store the players in game account name. Since account names are limited to 16 characters, and no two players can have the same name in the same region I have allocated 32 characters. That should cover the current limit and also accommodate names double in length if ever this limit changes. |
| Account | AccountRegion | VARCHAR(8) | This field will store the region, and therefore server, the account plays on. Regions usually can be abbreviated with 2 to 3 letters like NA (North America) or EUW (Europe West). Therefore 8 characters should be more than enough to accommodate existing and new regions should additional ones be added. |
| Champion | ChampionName | VARCHAR(30) | This field will store the name of the champion. Currently, the shortest champion’s name is 2 characters and the longest is 14. An extra 16 characters added to the longest should cover any new names added. |
| Champion | ChampionRole | CHAR(1) | This field will hold one of the following characters: “T”, “J”, “M”, “A” or “S”. The characters respectively refer to the roles: Top, Jungle, Mid, ADC and Support. Technically, any champion can be played in any role, this field serves to identify the current role a champion best fits in the meta. |
| Champion | ChampionType | CHAR(3) | Champion types categorize champion playstyles. This field will hold one of the following three-character strings: “TNK”, “FGH”, “MGE”, “ASN”, “MRK”, “ENC”. The strings respectively refer to the following types: Tank, Fighter, Mage, Assassin, Marksmen, Enchanter. All champions fall into at least one of these types. |
| Champion | ChampionPeak | CHAR(1) | Champions have attributes and abilities that make them stronger and weaker at certain points in the game. The three phases are broken down into early game, mid game, and late game. One of the following characters will identify that peak: “E”, “M” or “L”. |
| Champion | ChampionHealth | DECIMAL(5) | This field will contain the health points given to a champion. No champion starts with health over 9,999, and it is extremely unlikely a new one would, so a length of 5 should be plenty. |
| Champion | ChampionAttack | DECIMAL(3) | This field will contain the attack damage given to a champion. No champion starts with over 99 attack damage, and it is unlikely a new one would. Therefore, a max length of 3 is allocated. |
| Match | WinOrLose | CHAR(1) | This field stores whether the player has won “W” or lost “L” his or her match. Since there are no ties, a single character is all that is needed. |
| Match | MatchKills | DECIMAL(2) | This field stores the number of kills a player has secured. Since most games result in players having kills around 5 to 15, 99 should be high enough. Even further, any game where a player has even close to 50 kills would suggest cheating or the enemy team intentionally dying. Allowing for such data to be stored would only ruin the purity of the analysis. |
| Match | MatchDeaths | DECIMAL(2) | This field stores the number of times a player has been slain. The reasoning for having decimal 2 is the same as “MatchKills”. |
| Match | MatchAssists | DECIMAL(2) | This field stores the number of times a player has helped an ally kill an enemy. The reasoning for having decimal 2 is the same as “MatchKills”. |
| Match | MatchCS | DECIMAL(4) | This field stores the number of creeps (enemy AI) killed. As the most reliable and easiest way to gain gold this is an essential stat to track. Typical CS numbers are around 150, where even a CS score like 400 is rarely seen even at the highest level of competition. Therefore, allowing for up to 9,999 should be plenty. |
| Match | MatchGold | DECIMAL(7) | This field will store the total amount of gold a player has earned. Based off [leaguemath.com](https://www.leaguemath.com/mapwide-resources-gold-xp/), at the highest level of play, meaning games played between the highest-ranking players, the peak of bell curve distribution for team gold is 120,000. Dividing that by 5 for each player on the team results in 24,000 per player. Therefore, allowing for gold amounts up to 999,999 should be enough. |
| Match | MatchRole | CHAR(1) | This field will store the role a player is assigned. See “ChampionRole” for reasoning and expected values. |
| Match | MatchDate | DATE | The field will store the date the match was played on. This will allow for match data to be analyzed based on different patch updates. |
| Item | ItemName | VARCHAR(30) | This field will store the name of the item. Most items are usually around 10 characters long with a max under 20. Therefore, a character length of 30 will store existing items and should cover new ones as well. |
| Item | ItemActive | CHAR(1) | This field identifies whether the item has an active ability or not. Therefore, 1 character with either a “Y” for yes or an “N” is all that is needed. |
| Item | ItemType | CHAR(1) | This field will contain data to identify which type an item is. All items can be broken down into three basic types: Offensive, Defensive, Supportive. Therefore, the following single character values “O”, “D” or “S” will populate this field. |
| Item | ItemCost | DECIMAL(5) | This field stores the gold cost of an item. The most expensive item in the game cost 3600 gold, so a 99,999 will cover all existing items and should cover new ones as well. |
| Item | ItemMythic | CHAR(1) | This field will store whether an item is a mythic or not. Mythic items are special for several reasons, most notably a passive that interacts with other non-mythic items. To simply identify this a single character with either a “Y” or “N” will suffice. |
| MythicItem | MythicPassive | VARCHAR(128) | This field will store a description of the mythic passive a mythic item has. Descriptions are usually a few sentences long, describing a basic correlation to other items such as, “For every non-mythic item in inventory gain ## ‘stat’”. Due to their basic and simple description allowing for 128-character descriptions should be plenty to store. |
| NonMythic | Consumable | CHAR(1) | This field stores whether a non-mythic item is a consumable or not. A consumable item is one in which it is used once and then it exits a player’s inventory. A health potion, for example, grants healing for a small duration then is gone. For this a simple “Y” or “N” character will work. |



# Stored Procedure Execution and Explanations

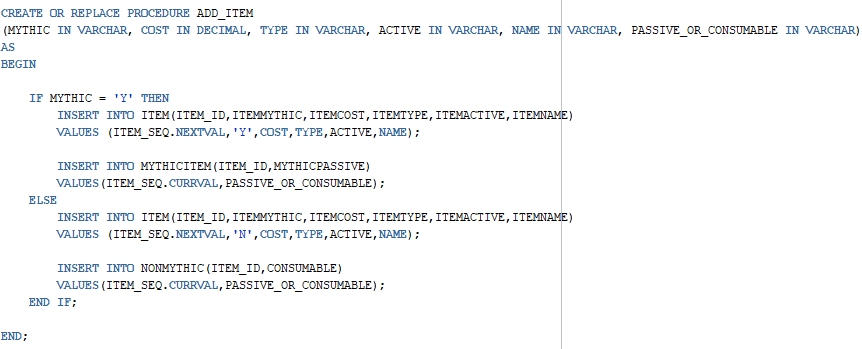
Before the website is made available to the public, the database must first be initialized with some data. For example, before a match can include purchases, it must first have the items available to be purchased. If a new item has been added to the game recently, this item must be included because it might be purchased.

Item Register.

1. Developers add a new item to League of Legends.
2. Background process monitors League of Legends items via API and checks for new items.
3. New item is created and data about it is captured and sent to database.
4. Stored procedure is executed to insert a new item into database.

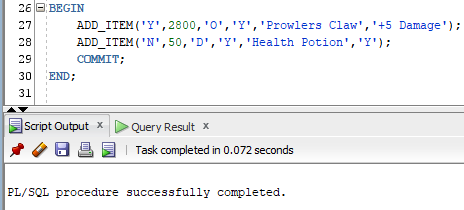
For this use case, I will implement a single stored procedure to insert either a mythic item or a non-mythic item.

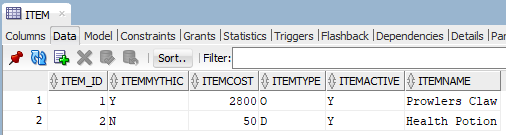
Below is a screenshot of the procedure definition.

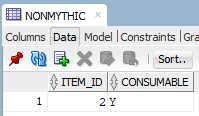
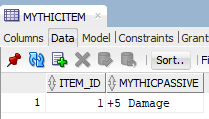


The stored procedure above is named “ADD\_ITEM”, and it passes all the parameters that correspond to columns in the Item table. I also add a parameter at the end called “PASSIVE\_OR\_CONSUMABLE”. This should include either a yes/no flag or an item passive description depending on the value given for “ITEMMYTHIC”. If the item is a mythic then, in addition to creating a record for the item table, a record for the “MYTHICITEM” table is created and uses the “PASSIVE\_OR\_CONSUMABLE” parameter to populate the “MYTHICPASSIVE” field of the table. If the item is not a mythic, then instead a “NONMYTHIC” record is created, and the parameter is used to populate the “CONSUMABLE” field.

This is the screenshot of my stored procedure used to add both a mythic and non-mythic item.





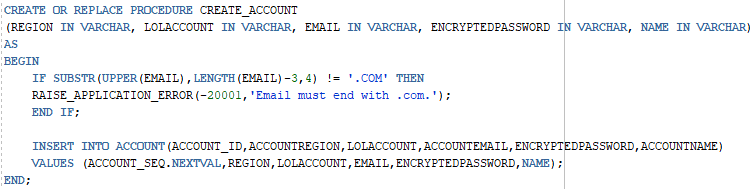


Once all the current game data needed has been added, the website is now ready to allow users to create their accounts.

Account Signup

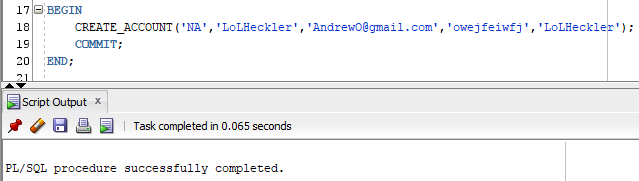
1. Person visits LeagueAlytics website.
2. Website asks them to create account.
3. The user enters in administrative information.
4. They are then asked to link this account to their LoL account.

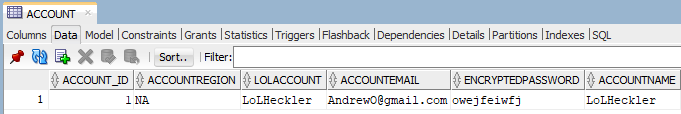
To allow users to create accounts, I have created the following stored procedure below named “ADD\_ACCOUNT”.

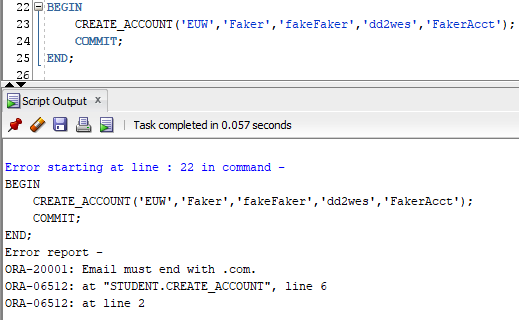


In addition to inserting a new record into the Account table, the stored procedure also validates the “EMAIL” parameter passed to it. This check ensures that at the very least the email address contains the usual “.com” appended to it. If the user tries to create an account with a bogus email address the stored procedure will raise an error stating that it must end with “.com”.

The following are screenshots of my stored procedure inserting a new record and rejecting another with an incorrect email address.





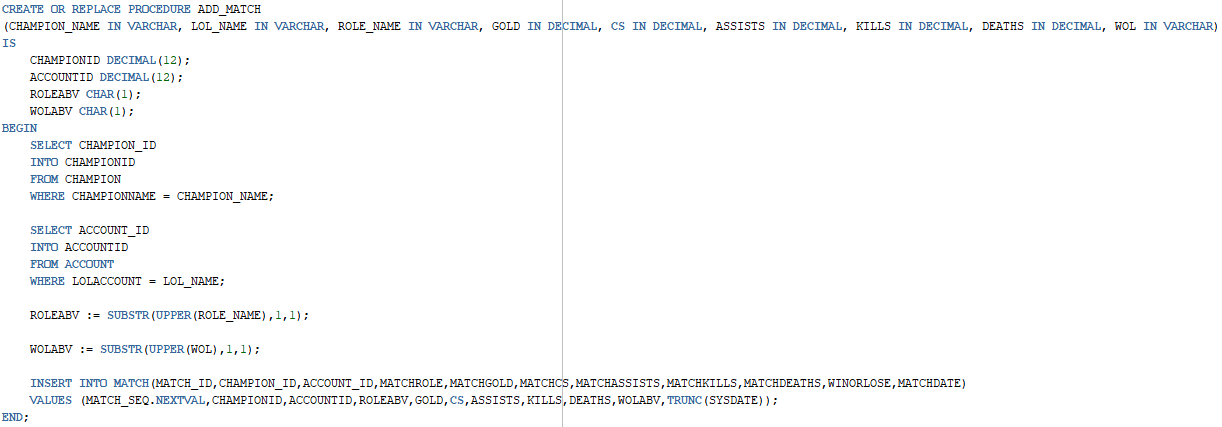


Once the database has been initialized and a user has created an account, we are now ready to begin pulling in that player’s match history.

Data retrieval and monitoring.

1. Person visits LeagueAlytics website.
2. They then create an account and connect their LoL account.
3. Their match history data is then retrieved and stored into the database.
4. The website also will track any new match data that is created once a player plays additional games.

To facilitate the process of adding match records to the database I have created a stored procedure named “ADD\_MATCH”.

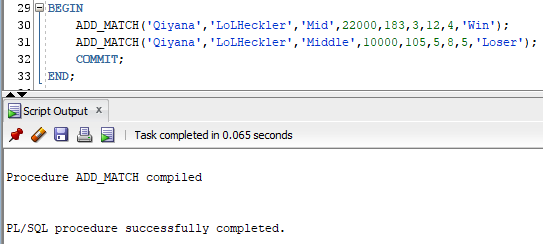


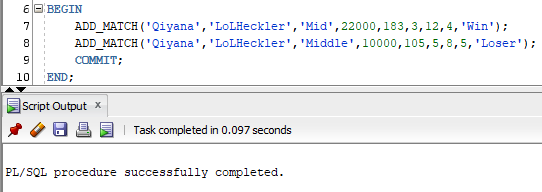
There are two key functional differences between this stored procedure and the ones that preceded it. One is the reliance on data that already exists in the database. Before any match data may be collected it is assumed that the player involved has created an account, and that the champion they are playing is already recorded in the database. The other distinction is the frequency at which the procedure will be called. A player will only create a user account once, and a champion, or item, will only be added a couple times in a year. For both reasons, I have included variables within the procedure to help with data insertion.

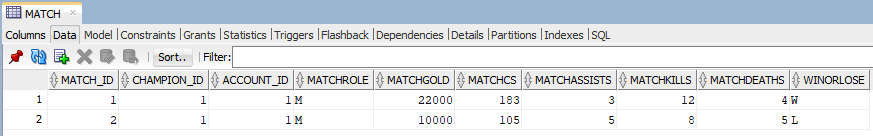
The variables “CHAMPIONID” and “ACCOUNTID” will help map champion and account names to their corresponding primary key. The match data retrieved has no knowledge of my database, nor the key values given to each champion and account. However, what the match data does know and can send is the name of the champion and name of the League of Legends account. Within the procedure I therefore find the primary key values for both champion and account based on their names.

The variables “ROLEABV” and “WOLABV” will assist in the conversion of the string values sent to the string values expected in the table. As stated above, the data retrieved has no knowledge of my database, nor the constraints and conventions in place. Therefore, the role and win/lose values sent are most likely not single letter abbreviations as the match table requires its values to be. Also, the values sent are subject to change. Maybe a win is sent as “Win” but then changes to “WINNER”. Or maybe Top Lane is “TOP” but then is changed to “Top Lane”. Changes like these, over the course of just one day, could be the reason why hundreds of thousands of matches are not recorded.

The result of executing this stored procedure with data from two different matches is captured below.







# Question Identification and Explanations

**First Query**

Before a player can improve, they must first know where they currently stand. For this reason, the first question I would ask myself is, “What is my win-rate percentage on Qiyana when I build Prowlers Claw?”.

Like many team sports, each player has a unique role and responsibilities. For the same reason a hockey player doesn’t play forward one game and goalie the next, or a football player doesn’t punt one game and quarterback the next, LoL players pick and practice a single role to specialize in. Then, within the role they have chosen, they will pick a few Champions to master. With a role and champion pool selected the last decision a player must make is which items to purchase. Itemization is by far the most controversial topic discussed amongst the community. Professionals, streamers, and coaches are always theory crafting and debating which items are superior to others. Everyone is trying to figure out which items best compliment the identity of a champion, and what does the success or failure of their ability to utilize certain items say about their identity. With so many different combinations, and in game situations that might warrant one item over another, it is something that can be overwhelming.

**Second Query**

Figuring out which mythic item a player should buy can be hard, but what if a player isn’t even able to get enough gold to even buy one? Therefore, a question one might even ask before then is, “How many games did I play and not buy a mythic?”.

What can happen in games, especially on champions who peak in the late game and are weak early, is a player can lose early and not get enough gold to buy a mythic before the game ends. This would suggest that a player has issues that need to be address before they even start to worry about mythic choice.

**Third Query**

The queries above can be helpful when a player has already decided which role and champions they are committed to. Of course, there are players who have not picked either, or maybe would like to try something new. A question they might ask is, “Who are the best players, and what Champions do they play?”

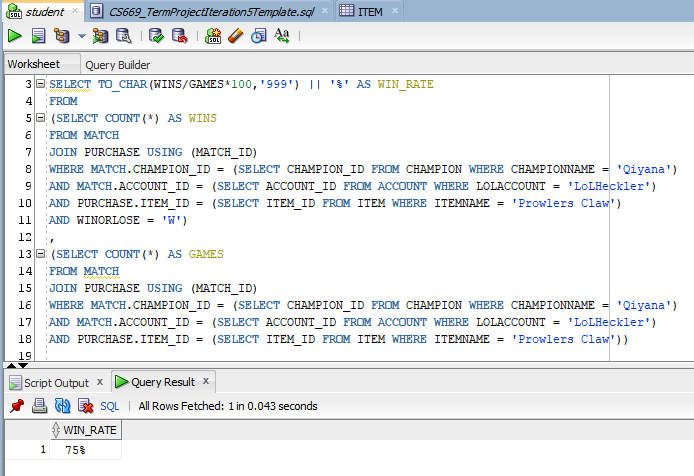
One of the best aspects of playing LoL is the size of the player base around the world. As I mentioned in the overview, League of Legends is the most popular video game in the world. Like any competition, as the number of people that play increases, so does level of skill needed to win. With international tournaments held annually, the best players from each region now to get compete to determine who is the best on the planet. To be the very best takes an unimaginable amount of time and dedication. The best players in the world play on average 10 hours every single day. Now not everyone can match such a strict regiment, but what they can do is watch and learn from these players. For the player who is thinking about a path to the top, seeing who the best players win with makes a lot of sense.

# Query Executions and Explanations

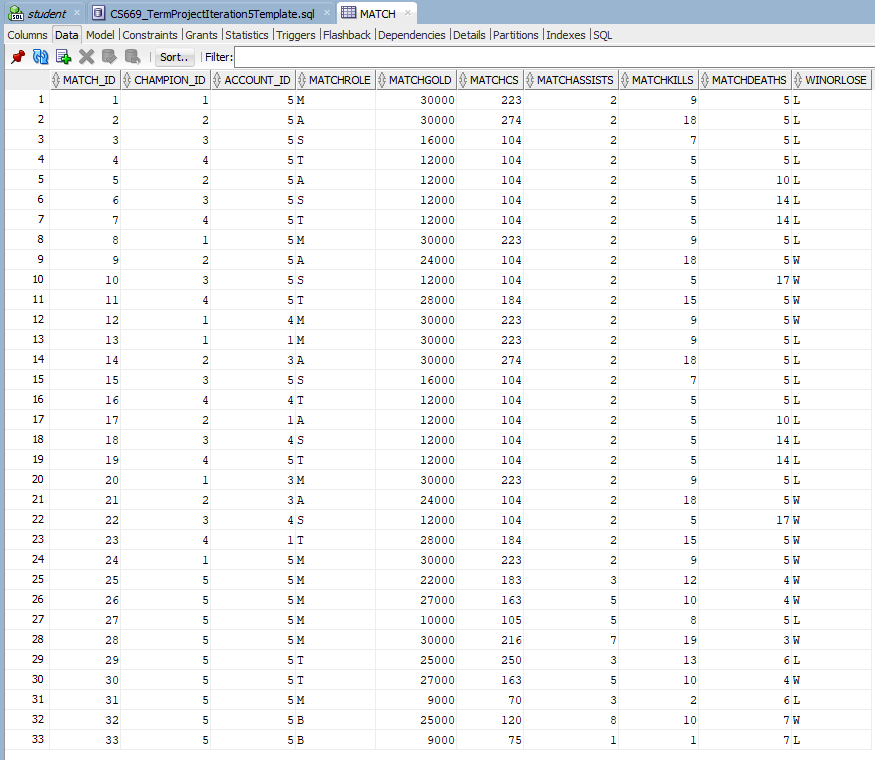
**First Query**

The screenshot below shows the query in use.

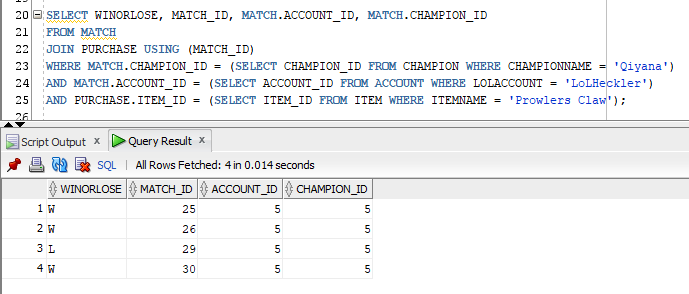
“What is my win-rate percentage on Qiyana when I build Prowlers Claw?”



To get the win rate, I first join the Purchase table the Match table to get access to the items I built in my matches. I first then pull filtered Match data in my two outer subqueries. Only Match records where I, “LoLHeckler”, played “Qiyana” and built “Prowlers Claw” are returned. I select the necessary keys needed to filter Match data via inner subqueries filtered on the name of the champion, name of the LoL account and name of item. Then, the subqueries differ so that the first one only counts the games I have won and the one beneath it counts all the games I played. I then calculate the win perfect and format it for output.

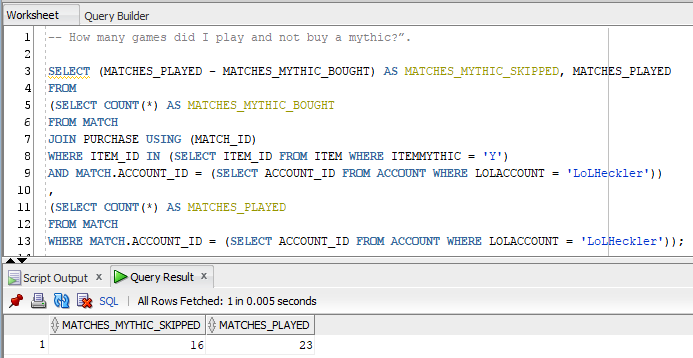


Above shows the Match table data where my ID is 5, Qiyana is 5. Here is a query to help make it clearer to see.

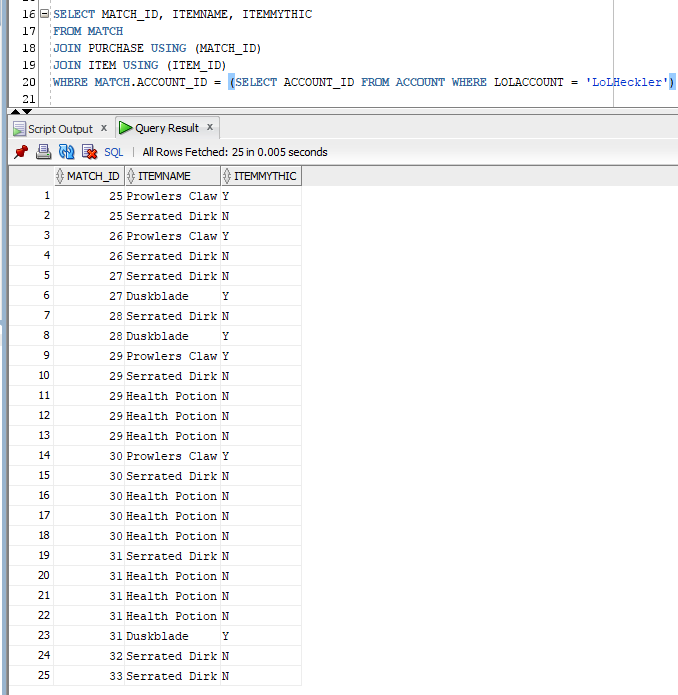


**Second Query**

“How many games did I play and not buy a mythic?”.



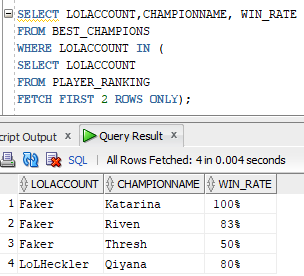
To find the number of mythic items bought I first join Purchase to Match on “MATCH\_ID”. I then narrow the query by matches played by me, then matches where a mythic item was purchased. Like the first query, I select the needed keys via a sub query on the same tables. I also include the total amount of matches I have played in the bottom sub query for context. These two queries are basically the same except I do not filter out matches without purchases of mythic items. The last point worth mentioning is on why I counted “MATCHES\_MYTHIC\_BOUGHT” and did a calculation in the select instead of “MATCHES\_MYTHIC\_SKIPPED” directly. The reason for this is the purchase constraint in game that only allows a champion to purchase a single mythic. Non-mythic items however may be purchased multiple times. It is common for consumable items for instance to be purchased dozens of times a game. The uniqueness of each mythic item to each match made it much simpler to query, although the alternative query is possible.



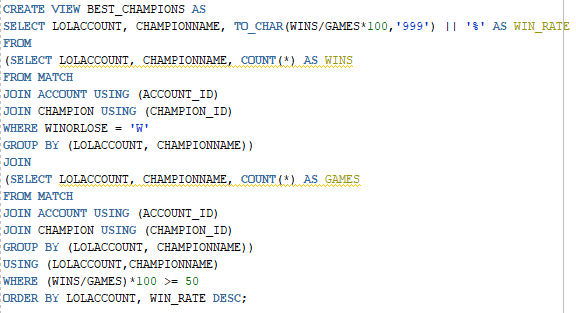
The snippet above shows all the purchases I have made in all my games played. Notice how there are several rows that have the same “MATCH\_ID” but not a single pair of those rows contain two mythic items.

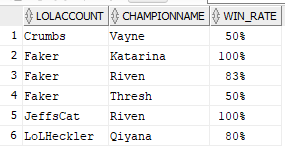
**Third Query**

“Who are the best players, and what Champions do they play?”

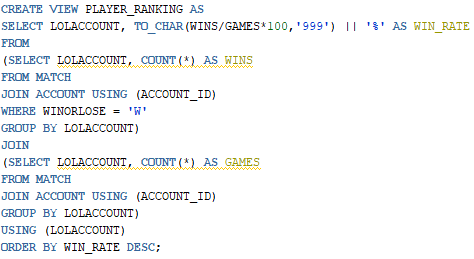


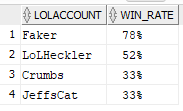
The simple query above selects the top two ranked players along with their win rate on best champions. The query is made simple due to the two views above. Below are snippets of their definitions.





Above is the “BEST\_CHAMPIONS” view. This was created so that players could see the champions they have found the most success with in their games. The first subquery grabs all the matches won then returns the total amount of wins each account has grouped by the champion they won with. The second query does the same thing as the first except it grabs matches lost as well. Together, the win rate for each account on each champion they play is calculated. Lastly, since this list is an accounts best champions, there is a where clause at the end to disregard win rates that are below 50%.





Above is the view definition for “PLAYER\_RANKING”. The view was created to keep track of and compare the ranking of all players. The players with the highest win rates are ordered from top to bottom with the top row being the best player. To get each players “WIN\_RATE”, the first sub query must select all of the matches won by each account. The second sub query must then select all the total games an account has played. With these two tables it is a simple calculation to find each accounts win rate.

# Index Identification and Creations

The following list includes all the primary keys for each table created.

* Account.Account\_ID
* Match.Match\_ID
* Champion.Champion\_ID
* Purchase.Purchase\_ID
* Item.Item\_ID
* MythicItem.Item\_ID
* NonMythic.Item\_ID

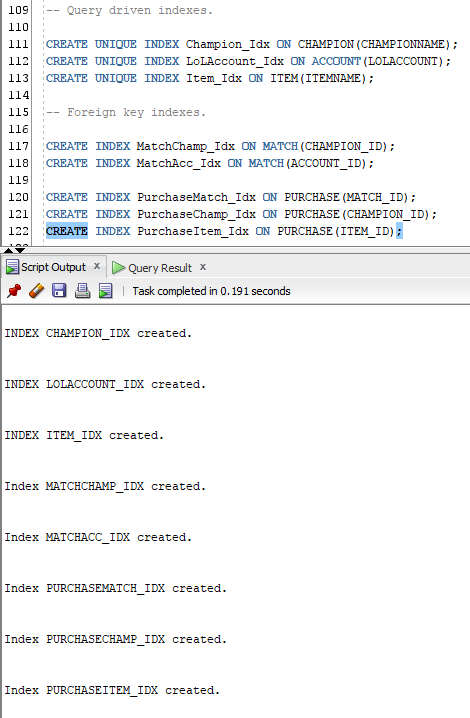
The follow table lists all foreign keys and an explanation on whether it must be unique.

|  |  |  |
| --- | --- | --- |
| **Column** | **Unique/Not Unique** | **Description** |
| Match.Champion\_ID | Not Unique | This foreign key is not unique because a champion may play many matches. |
| Match.Account\_ID | Not Unique | This foreign key is not unique because an account may have many matches. |
| Purchase.Match\_ID | Not Unique | This foreign key is not unique because a match may include many purchases. |
| Purchase.Champion\_ID | Not Unique | This foreign key is not unique because a champion may make many purchases. |
| Purchase.Item\_ID | Not Unique | This foreign key is not unique because an item may be gained in many purchases. |
| MythicItem.Item\_ID | Unique | This foreign key is also the primary key, and therefore already unique. |
| NonMythic.Item\_ID | Unique | This foreign key is also the primary key, and therefore already unique. |

Three query driven indexes that make sense for this database are the columns: “ChampionName”, “LoLAccount” and “ItemName”.

The reason why these three columns all deserve their own index is simple, they hold the unique columns values players already know. As demonstrated by my own queries above, the questions and answers players will ask and want will be based on these values. When someone asks a question about an account within the database, they will do so using the accounts LoL name. When a question about a champion is asked it will be with the champion’s name. The same goes for any item question, it will be asked with the name of the item.

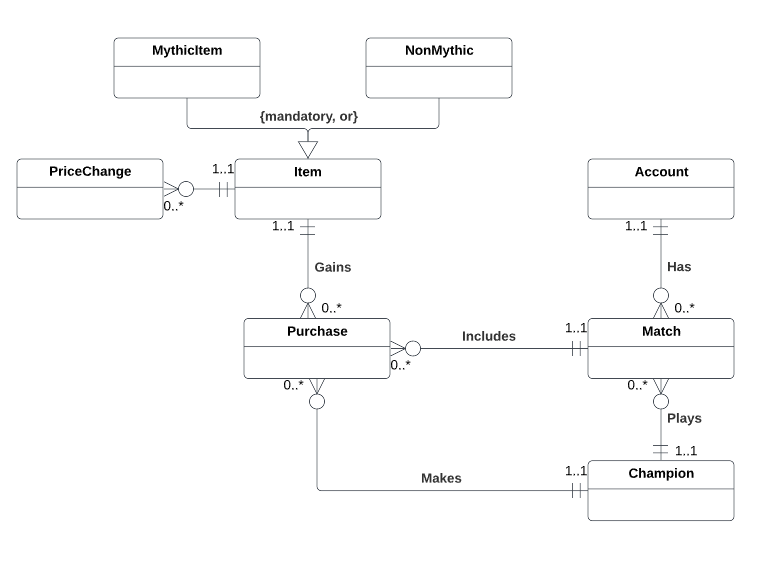
There maybe other ways in which queries are asked information about these tables that will not include their names. A question such as “How many items do fighter type champions buy on average in a game?” However, whenever a question will be asked about a specific champion, account or item, it will undoubtedly be with their respective name.



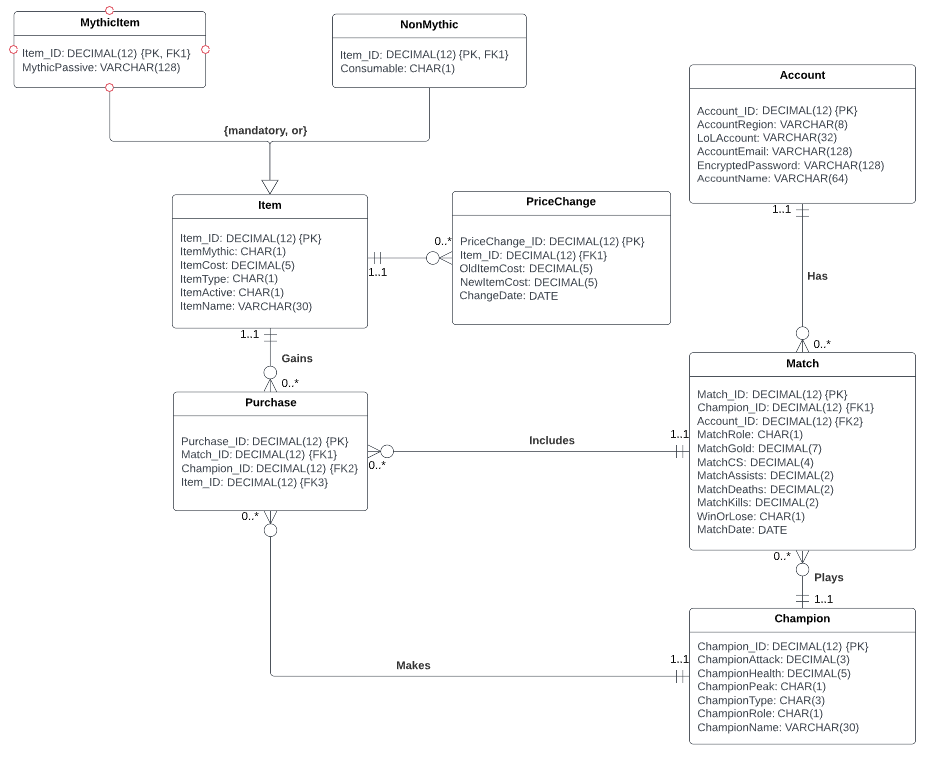
# History Table Demonstration

LoL is a game that is constantly changing and evolving. There are new patch updates roughly once a month that include changes of all shapes and sizes. One example of a possible change might be the price of an item. By keeping track of an items old price, my database would be able to quantify the effect the change has had by looking at matches before and after the update.

My updated ERD below shows a new “PriceChange” table that will be storing this data. It of course is related to the “Item” table which stores item data.



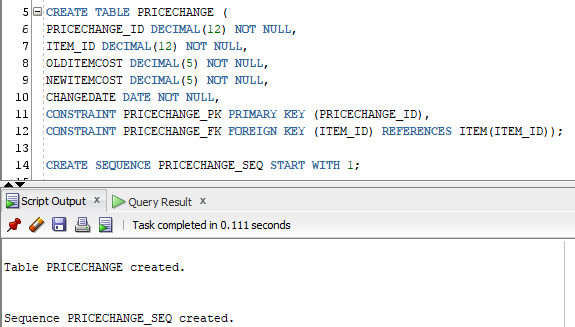
The updated DBMS physical ERD is below.



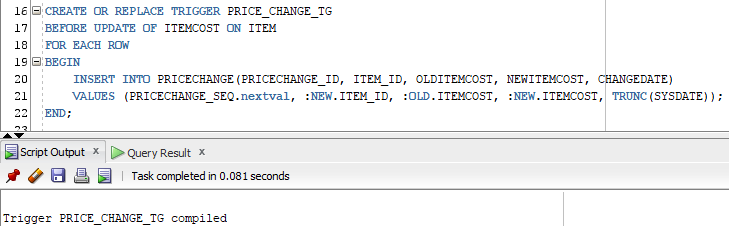
The “PriceChange” entity above captures the relational constraints and attributes needed. Below is a description and explanation for each attribute present.

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| PriceChange\_ID | This is the primary key of the PriceChange table. It is DECIMAL(12) to allow for many values similar to all other primary keys. |
| Item\_ID | This is a foreign key to the Item table. This is used to identify the item that had a change in price. |
| OldItemCost | This is the item cost before the patch update changed it. The datatype mirrors that of the ItemCost in the Item table. |
| NewItemCost | This is the item cost after the patch update changed it. The datatype mirrors that of the ItemCost in the Item table. |
| ChangeDate | This is the date the item price changed, naturally the datatype is DATE. |

The SQL to create this table and a sequence to be used for record insertion is shown below.

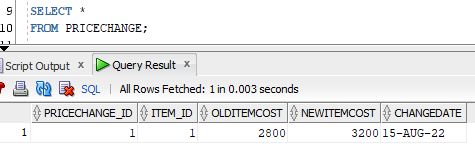
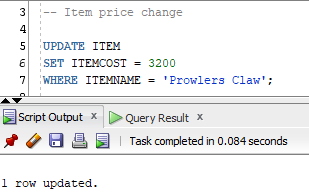


I will also need a trigger to capture when an item price changes. The SQL for such a trigger is shown below.



The trigger is quite simple, for each item with a price change a record of the price change is recorded.

As an example, let’s say that the item “Prowlers Claw” is too good in the current meta. By too good what I mean is the item is being picked in most games, and players who do pick is have an above average win-rate. The game designers might look at this and decide to release a new patch where they increase the price.

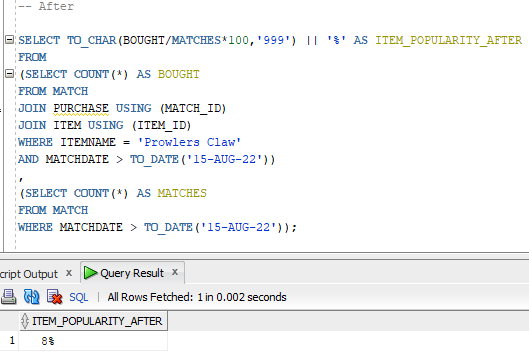
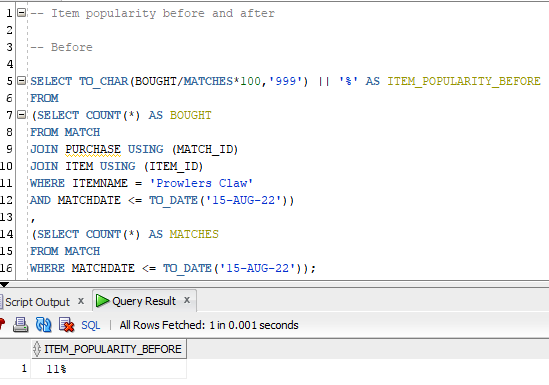


After about one month of the price change being live, we can begin to ask questions about the impact it had.

First Question: Item popularity

“How many people are buying Prowlers Claw now that the price has been increased?”

This is an important first question to ask before any further analysis of the item takes place. If the item is only being bought in 2% of games, whereas before it was 30%, determining whether the change in price influenced the outcome of a game will have to be done from a drastically smaller sample size. At the same time however, it might just be the case that players valued the cheapness of the item as one of its strengths, and an increase has led them to make other choices.

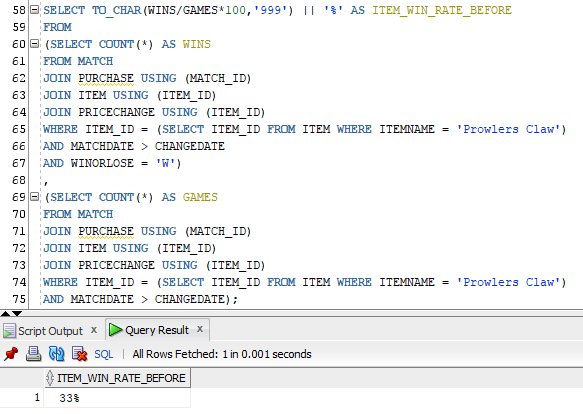
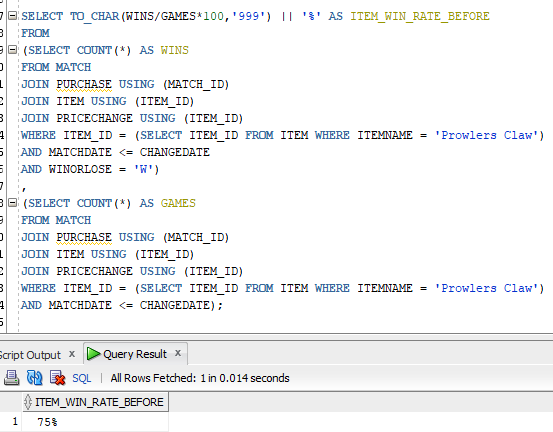


The two queries above show the popularity before and after the price of the item changed. The queries are almost identical except for the operator between the match date and the date the price changed. For each respective date range, the bottom subquery gets the total amount of matches played, and the top subquery gets the total amount of matches the item was purchased. From the results above it appears the item has decreased in popularity by 3%.

Second Question: Item Success

“What is the win-rate of Prowlers Claw after the price increase?”

This is arguably the most important question one would have after any sort of change. The game designers who increased the price did so to lower the success so many people were having. However, it is never the intention to ruin an item, rather the designers aim to balance it. Ideally, they would like to have win-rates around 50%, and anything too high or low warrants a change. For players, understanding the results of these changes can aid them in their games. For a player who is always building this item they might need to reconsider other options that have better success.



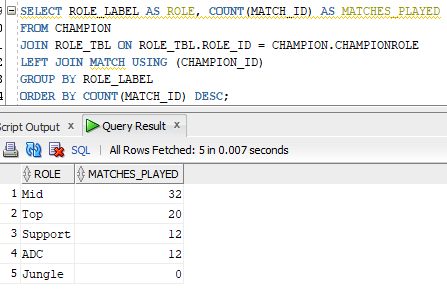
The two queries above show the win-rate before and after the price of the item changed. The queries are almost identical except for the operator between the match date and the date the price changed. For each respective date range, the bottom subquery gets the total amount of matches played where “Prowlers Claw” was purchased, and the top subquery gets the total amount of matches the item was purchased and won. From the results above it appears the item has significantly decreased in win-rate by 42%. Although the result of this price change was probably more than the game designers would have wanted it is better to have an item underpowered than overpowered. For players who have faithfully building this item the data would suggest it is time to find an alternative.

# Data Visualizations

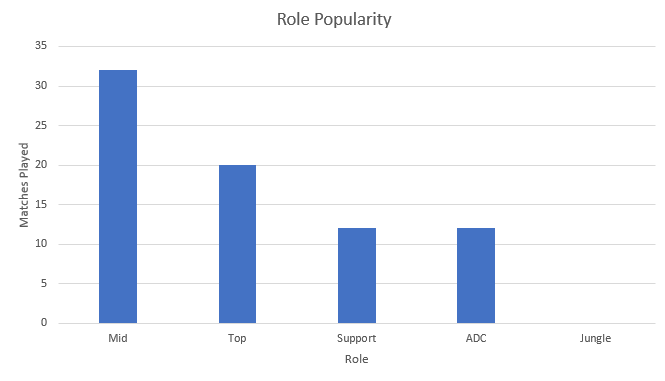
The point of the application is to help the players who are using it improve at the game. In addition to the aid the analysis provides, it would be great to have professional, or high-ranking player, give a personal guide on how to play their role. There are so many nuisance aspects of the game that make it impossible for just data to articulate fully. Having a human who understands all aspects of how the game should be played at the highest level, and who can explain it verbally will provide invaluable help.

Now of course, when offering a free service such as this, budget is limited. I can not practically be able to afford to pay a player for each role and champion they are an expert at. If I am only able to afford maybe to pay one or two players, I want to match sure that the players I choose are playing the same role and champions as my users. The question then is, “What roles are the majority of my players playing?”

To answer this question, I have written the following query that counts all the matches played for each of the five roles available.



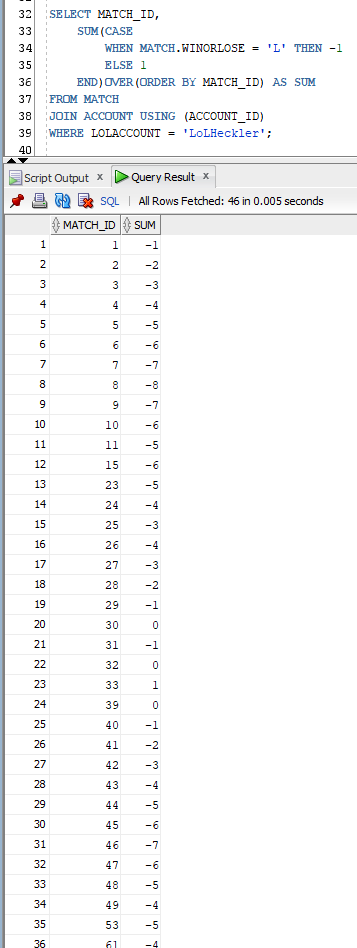
The following is a bar graph that gives a visual representation of the results above.



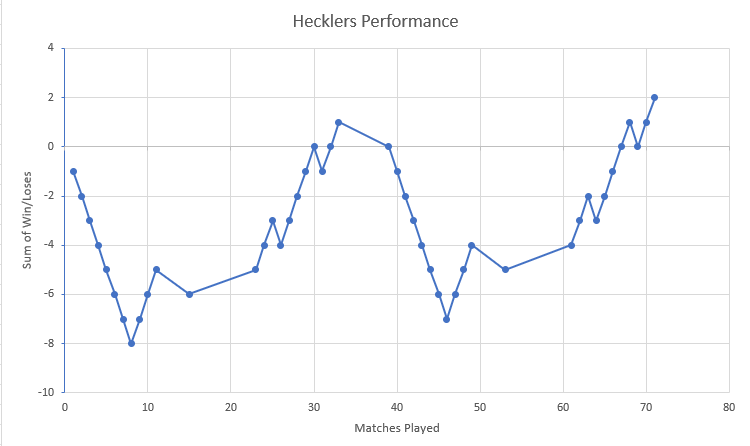
After reviewing the chart above it is extremely clear that Mid is the most popular role my users play. What is also surprising is the lack of Jungle players using the application. Based on this information, hiring a Mid lane player to create a guide for LeagueAlytics would serve most of the users on the site. Another take away from this graph is that the site should think about alternative ways to attract Jungle players.

Playing and trying to climb the rank ladder in LoL can be mentally exhausting. With game times that can last anywhere from 20 minutes to an hour, and the fact that any match won is effectively negated by one lost, it’s easy to understand why. Players grinding out games day after day can lose motivation and drive. That is why a great way to show a player how they have been improving since joining LeagueAlytics is to show their performance over time.

Let’s see how I have been performing since joining the site. The following query returns a row for each match played and a sum of the wins and losses, which correspond to +1 and -1, up until that match.



To better understand the data above I have created the scatter plot below.



There are a few key observations that immediately present themselves to me. First, since joining LeagueAlytics I have won more games than I have lost. That’s great! This is the goal of both the website and the players who join, to win more games than they lose. It also appears the way in which I progressed was in win streaks followed by shorter losing streaks. This makes sense because I typically only seek help if I’m losing. After going on a loss streak I would analyze my play and try to figure out what I needed to do differently.

# Summary and Reflection

My database will help facilitate the analysis offered on the LeagueAlytics website, which helps players get a better idea of how they stack up against other players and themselves. There are some websites that track or pull match history data from players and simply offer a dump of the data. This website aims to go one step further and offer players the tools to take that data and turn it into useful information.

The database captures all the fundamental concepts via entities that are needed to really break down and analyze how a player performs and the trends that accompany it. The entitles purposely are left as generic as possible with only essential attributes needed and several low length CHAR attributes which act as flags and tags to discretely identify the entity data they represent. The idea behind having so many of these flags and tags is so subtype entitles can be easily added for more specialized data storage without having to alter existing tables. For example, the Champion entity could easily be broken down into subtypes with a relationship of “mandatory” and “or” using the “ChampionType” attribute. Each type has unique attributes and questions relevant to specifically them which can further describe their champion and maybe add additional insight to playing them.

The SQL scripts define and contain all the conceptional ERD constraints and structural database rules indicated and diagramed above to ensure efficient and scalable data storage. It precisely mirrors the physical ERD with all the attributes and keys defined exactly as presented. The scripts also now include data inserts, stored procedures, query questions, views, and indexes. These additional components demonstrate the databases capability to not only store and maintain data but to manipulate it to answer any number of queries as well.

I am happy with how the database has turned out, and seeing it work with data has finally brought it to life in my eyes. I had to make many alterations early on which made me really think about how I wanted this to be designed. After a couple weeks of struggling, I came up with my current design and since then it has been much easier to work through iterations.