***League of Legends* Database**



**By Brian Main**

**Database Management Fall 2014**

**Professor Labouseur**

**Table of Contents**

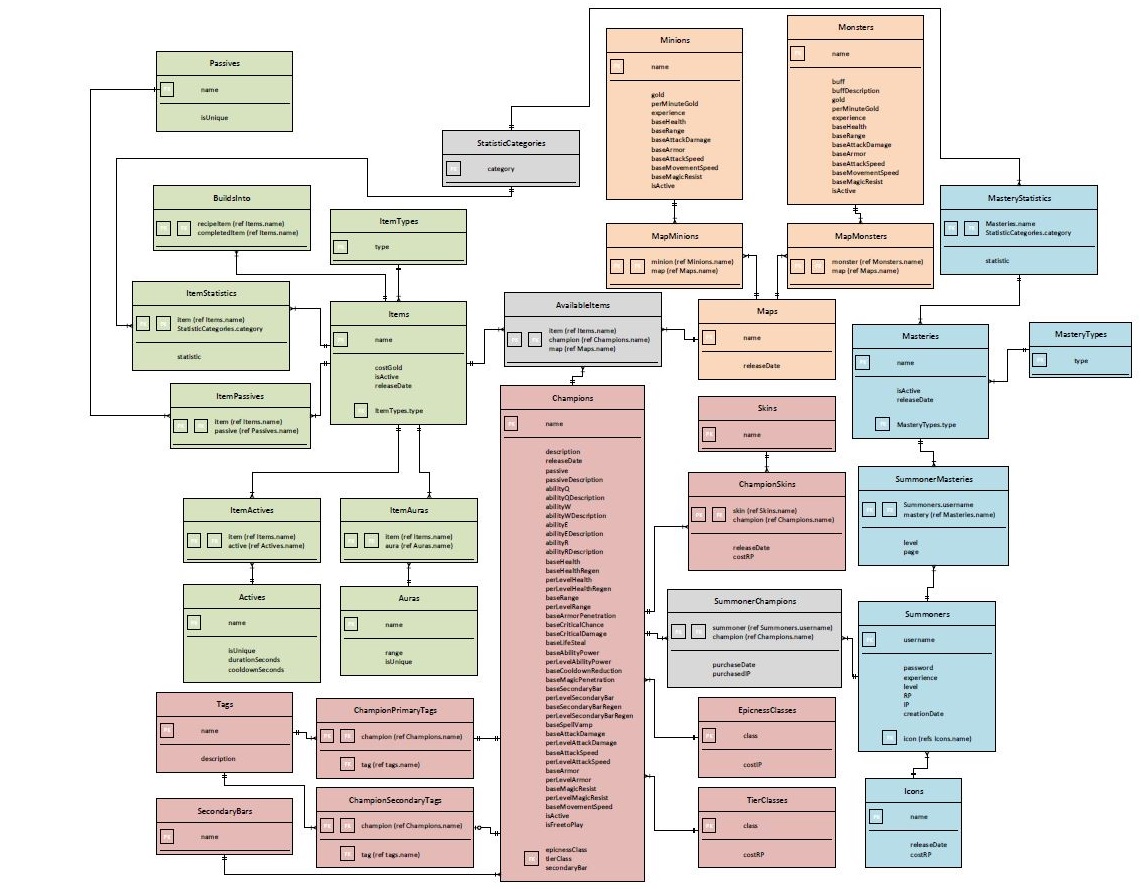
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Executive Summary | | | | | | | | | …………………………………………………………………………….3 | | | | | | | | | | | |
| Entity Relationship Diagram | | | | | | | | | | | | | | | | ……………………………………………………………………3 | | | | |
| Tables (create statements, functional dependencies, and sample data): | | | | | | | | | | | | | | | | | | | |  |
| Icons | | …………………………………………………………………………………….4 | | | | | | | | | | | | | | | | | | |
| Summoners | | | | | | | | ……………………………………………………………………………...4 | | | | | | | | | | | | |
| StatisticCategories | | | | | | | | | | | | | | | ……………………………………………………………………...4 | | | | | |
| MasteryTypes | | | | | | | | | | | …………………………………………………………………………..5 | | | | | | | | | |
| Masteries | | | | | | ………………………………………………………………………………..5 | | | | | | | | | | | | | | |
| MasteryStatistics | | | | | | | | | | | | | | ………………………………………………………………………..6 | | | | | | |
| SummonerMasteries | | | | | | | | | | | | | | | | ……………………………………………………………………6 | | | | |
| EpicnessClasses | | | | | | | | | | | | | ………………………………………………………………………...7 | | | | | | | |
| TierClasses | | | | | | | | ……………………………………………………………………………...7 | | | | | | | | | | | | |
| SecondaryBars | | | | | | | | | | | | ………………………………………………………………………….7 | | | | | | | | |
| Champions | | | | | | | | ……………………………………………………………………………...8 | | | | | | | | | | | | |
| Tags | | …………………………………………………………………………………….9 | | | | | | | | | | | | | | | | | | |
| ChampionPrimaryTags | | | | | | | | | | | | | | | | | | ………………………………………………………………...9 | | |
| ChampionSecondaryTags | | | | | | | | | | | | | | | | | | | ……………………………………………………………..10 | |
| Skins | | | …………………………………………………………………………………..10 | | | | | | | | | | | | | | | | | |
| ChampionSkins | | | | | | | | | | | | | ……………………………………………………………………….11 | | | | | | | |
| SummonerChampions | | | | | | | | | | | | | | | | | ………………………………………………………………...11 | | | |
| Maps | | | …………………………………………………………………………………..11 | | | | | | | | | | | | | | | | | |
| Minions | | | | | ……………………………………………………………………………….12 | | | | | | | | | | | | | | | |
| MapMinions | | | | | | | | | | …………………………………………………………………………..12 | | | | | | | | | | |
| Monsters | | | | | | ………………………………………………………………………………13 | | | | | | | | | | | | | | |
| MapMonsters | | | | | | | | | | | …………………………………………………………………………13 | | | | | | | | | |
| ItemTypes | | | | | | | ……………………………………………………………………………..14 | | | | | | | | | | | | | |
| Items | | | …………………………………………………………………………………..14 | | | | | | | | | | | | | | | | | |
| BuildsInto | | | | | | | ......................................................................................................................14 | | | | | | | | | | | | | |
| ItemStatistics | | | | | | | | | | …………………………………………………………………………15 | | | | | | | | | | |
| Passives | | | | | ……………………………………………………………………………….15 | | | | | | | | | | | | | | | |
| ItemPassives | | | | | | | | | | …………………………………………………………………………..15 | | | | | | | | | | |
| Actives | | | | ………………………………………………………………………………...16 | | | | | | | | | | | | | | | | |
| ItemActives | | | | | | | | | …………………………………………………………………………...16 | | | | | | | | | | | |
| Auras | | | …………………………………………………………………………………..17 | | | | | | | | | | | | | | | | | |
| ItemAuras | | | | | | | ……………………………………………………………………………..17 | | | | | | | | | | | | | |
| AvailableItems | | | | | | | | | | | | ………………………………………………………………………...17 | | | | | | | | |
| Queries | ………………………………………………………………………………………...19 | | | | | | | | | | | | | | | | | | | |
| Stored Procedures | | | | | | | ……………………………………………………………………………..19 | | | | | | | | | | | | | |
| Security | ………………………………………………………………………………………...21 | | | | | | | | | | | | | | | | | | | |
| Implementation Notes | | | | | | | | | | | …………………………………………………………………………21 | | | | | | | | | |
| Known Problems | | | | | | ………………………………………………………………………………21 | | | | | | | | | | | | | | |
| Future Enhancements | | | | | | | | | | | …………………………………………………………………………21 | | | | | | | | | |

**Executive Summary:**

This document contains my design and implementation of a database for the online computer game *League of Legends*. The main purpose of this database is to see past and present Champions (characters), Summoners (people), Items, and Maps in the video game. Potential users of this database would be Riot (the game’s makers) employees, people who play *League of Legends*, and gamers in general to get a sense of how the game works.

The Entity Relationship Diagram (ERD) is shown below, which consists of all the relationships between the tables in the database. Following the ERD is an explanation of each table, the create code, functional dependencies, and sample data created for the database. Some reports are generated after. Lastly, security and more information regarding implementation and future enhancements are discussed.

**Entity Relationship Diagram:**



**Table:** Icons

**Description:** Table for icons used by summoner’s profiles. Icons can be bought using Riot Points (RP).

**Create statements:**

create table Icons (

name text not null,

releaseDate date not null,

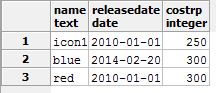
costRP integer not null,

primary key (name)

);

**Functional dependencies:** name → releaseDate, costRP

**Sample data:**

****

**Table:** Summoners

**Description:** Table for summoners, the term used for people playing the game *League of Legends*. Summoners have login information, experience and levels, RP and Influence points (IP), and a valid icon from the Icons table.

**Create statements:**

create table Summoners (

username char(24) not null,

password char(16) not null,

experience integer not null,

level integer not null,

RP integer not null,

IP integer not null,

creationDate date not null,

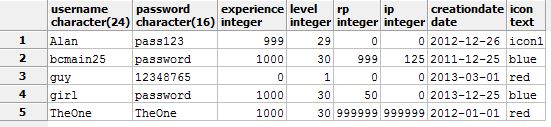
icon text not null references Icons(name),

primary key (username)

);

**Functional dependencies:** username → password, experience, level, RP, IP, creationDate, icon

**Sample data:**



**Table:** StatisticCategories

**Description:** Table for in-game statistic bonus categories.

**Create statements:**

create table StatisticCategories (

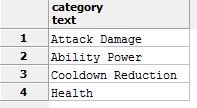
category text not null,

primary key (category)

);

**Functional dependencies:** category→

**Sample data:**

****

**Table:** MasteryTypes

**Description:** Table for mastery types used for masteries.

**Create statements:**

create table MasteryTypes (

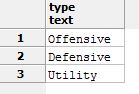
type text not null,

primary key (type)

);

**Functional dependencies:** type →

**Sample data:**

****

**Table:** Masteries

**Description:** Table for masteries, or bonuses, given to any champion and chosen by a summoner. These use the mastery types offense, defense, and utility.

**Create statements:**

create table Masteries (

name text not null,

isActive boolean not null,

releaseDate date not null,

type text not null references MasteryTypes(type),

primary key (name)

);

**Functional dependencies:** name → isActive, releaseDate, type

**Sample data:**

****

**Table:** MasteryStatistics

**Description:** Table for statistics given by specific masteries.

**Create statements:**

create table MasteryStatistics (

name text not null references Masteries(name),

category text not null references StatisticCategories(category),

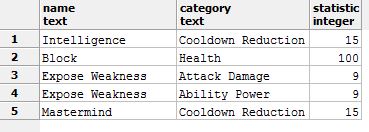
statistic integer not null,

primary key (name, category)

);

**Functional dependencies:** name, category → statistic

**Sample data:**

****

**Table:** SummonerMasteries

**Description:** Table showing which summoners have which masteries. Summoners are allowed to enter points to increase level on mastery and can make mastery pages.

**Create statements:**

create table SummonerMasteries (

username text not null references Summoners(username),

mastery text not null references Masteries(name),

level integer not null,

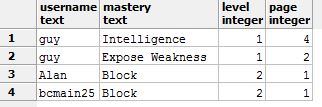
page integer not null,

primary key (username, mastery)

);

**Functional dependencies:** username, mastery → level, page

**Sample data:**

****

**Table:** EpicnessClasses

**Description:** Table for the classes used for buying champions through IP.

**Create statements:**

create table EpicnessClasses (

class text not null,

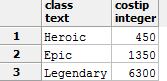
costIP integer not null,

primary key (class)

);

**Functional dependencies:** class → costIP

**Sample data:**

****

**Table:** TierClasses

**Description:** Table for the classes used for buying champions through RP. Note that while most champions have the same epicness and tier class, this is not always the case.

**Create statements:**

create table TierClasses (

class text not null,

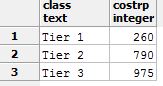
costRP integer not null,

primary key (class)

);

**Functional dependencies:** class → costRP

**Sample data:**

****

**Table:** SecondaryBars

**Description:** Table for secondary bars. Typically, champions use mana, but some are manaless or use another type of energy.

**Create statements:**

create table SecondaryBars (

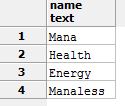
name text not null,

primary key (name)

);

**Functional dependencies:** name →

**Sample data:**

****

**Table:** Champions

**Description:** Table of champions, the term used for playable characters in the game *League of Legends*. Note that given a specific champion, all of the statistics of the champion can be found.

**Create statements:**

create table Champions (

name text not null,

description text not null,

releaseDate date not null,

passive text not null,

passiveDescription text not null,

abilityQ text not null,

abilityQDescription text not null,

abilityW text not null,

abilityWDescription text not null,

abilityE text not null,

abilityEDescription text not null,

abilityR text not null,

abilityRDescription text not null,

baseHealth integer not null,

baseHealthRegen integer not null,

perLevelHealth integer not null,

perLevelHealthRegen integer not null,

baseRange integer not null,

perLevelRange integer not null,

baseArmorPenetration integer not null,

baseCriticalChance integer not null,

baseCriticalDamage integer not null,

baseLifeSteal integer not null,

baseAbilityPower integer not null,

perLevelAbilityPower integer not null,

baseCooldownReduction integer not null,

baseMagicPenetration integer not null,

baseSecondaryBar integer not null,

perLevelSecondaryBar integer not null,

baseSecondaryBarRegen integer not null,

perLevelSecondaryBarRegen integer not null,

baseSpellVamp integer not null,

baseAttackDamage integer not null,

perLevelAttackDamage integer not null,

baseAttackSpeed integer not null,

perLevelAttackSpeed integer not null,

baseArmor integer not null,

perLevelArmor integer not null,

baseMagicResist integer not null,

perLevelMagicResist integer not null,

baseMovementSpeed integer not null,

isActive boolean not null,

isFreetoPlay boolean not null,

epicnessClass text not null references EpicnessClasses(class),

tierClass text not null references TierClasses(class),

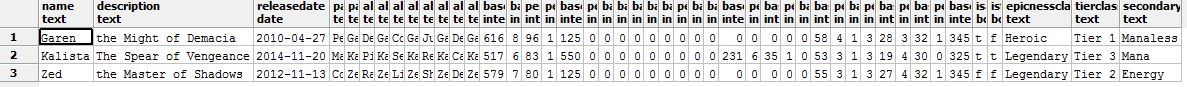
secondaryBar text not null references SecondaryBars(name),

primary key (name)

);

**Functional dependencies:** name → description, releaseDate, passive, passiveDescription, abilityQ, abilityQDescription, abilityW, abilityWDescription, abilityE, abilityEDescription, abilityR, abilityRDescription, baseHealth, baseHealthRegen, perLevelHealth, perLevelHealthRegen, baseRange, perLevelRange, baseArmorPenetration, baseCriticalChance, baseCriticalDamage, baseLifeSteal, baseAbilityPower, perLevelAbilityPower, baseCooldownReduction, baseMagicPenetration, baseSecondaryBar, perLevelSecondaryBar, baseSecondaryBarRegen, perLevelSecondaryBarRegen, baseSpellVamp, baseAttackDamage, perLevelAttackDamage, baseAttackSpeed, perLevelAttackSpeed, baseArmor, perLevelArmor, baseMagicResist, perLevelMagicResist, baseMovementSpeed, isActive, isFreetoPlay, epicnessClass, tierClass, secondaryBar

**Sample data:**

****

**Table:** Tags

**Description:** Table for tags used for champion categorizing.

**Create statements:**

create table Tags (

name text not null,

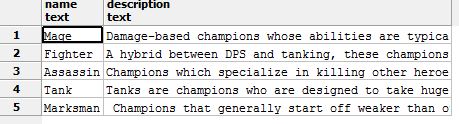
description text not null,

primary key (name)

);

**Functional dependencies:** name → description

**Sample data:**

****

**Table:** ChampionPrimaryTags

**Description:** Table for which champions are which primary tag. A primary tag is given based on the champion’s typical role in the game.

**Create statements:**

create table ChampionPrimaryTags (

champion text not null references Champions(name),

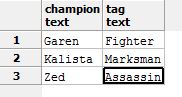
tag text not null references Tags(name),

primary key (champion)

);

**Functional dependencies:** champion → tag

**Sample data:**

****

**Table:** ChampionSecondaryTags

**Description:** Table for which champions are which secondary tag. A secondary tag can be given based on the champion’s typical alternate role in the game.

**Create statements:**

create table ChampionSecondaryTags (

champion text not null references Champions(name),

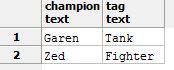
tag text not null references Tags(name),

primary key (champion)

);

**Functional dependencies:** champion → tag

**Sample data:**

****

**Table:** Skins

**Description:** Table for skin categories for champions.

**Create statements:**

create table Skins (

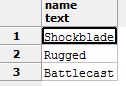
name text not null,

primary key (name)

);

**Functional dependencies:** name →

**Sample data:**

****

**Table:** ChampionSkins

**Description:** Table for which champions have which skins.

**Create statements:**

create table ChampionSkins (

skin text not null references Skins(name),

champion text not null references Champions(name),

releaseDate date not null,

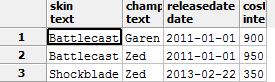
costRP integer not null,

primary key (skin, champion)

);

**Functional dependencies:** skin, champion → releaseDate, costRP

**Sample data:**

****

**Table:** SummonerChampions

**Description:** Huge table for all of the current combinations of summoners (people) and champions (characters) in the game. If an entry lives in this table, then the summoner owns the champion. If purchasedIP is false, then the champion was bought through RP.

**Create statements:**

create table SummonerChampions (

summoner text not null references Summoners(username),

champion text not null references Champions(name),

purchaseDate date not null,

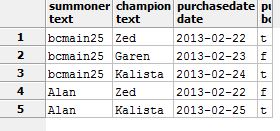
purchasedIP boolean not null,

primary key (summoner, champion)

);

**Functional dependencies:** summoner, champion → purchaseDate, purchasedIP

**Sample data:**

****

**Table:** Maps

**Description:** Table of playable maps.

**Create statements:**

create table Maps (

name text not null,

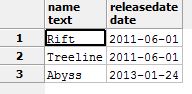
releaseDate date not null,

primary key (name)

);

**Functional dependencies:** name → releaseDate

**Sample data:**

****

**Table:** Minions

**Description:** Table of minions, non-playable allies.

**Create statements:**

create table Minions (

name text not null,

gold integer not null,

perMinuteGold integer not null,

experience integer not null,

baseHealth integer not null,

baseRange integer not null,

baseAttackDamage integer not null,

baseArmor integer not null,

baseAttackSpeed integer not null,

baseMovementSpeed integer not null,

baseMagicResist integer not null,

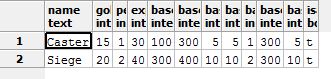
isActive boolean not null,

primary key (name)

);

**Functional dependencies:** name → gold, perMinuteGold, experience, baseHealth, baseRange, baseAttackDamage, baseArmor, baseAttackSpeed, baseMovementSpeed, baseMagicResist, isActive

**Sample data:**

****

**Table:** MapMinions

**Description:** Table for which minions play on which map.

**Create statements:**

create table MapMinions (

minion text not null references Minions(name),

map text not null references Maps(name),

primary key (minion, map)

);

**Functional dependencies:** minion, map →

**Sample data:**

****

**Table:** Monsters

**Description:** Table of monsters, non-playable neutral entities that can give champions buff when killed.

**Create statements:**

create table Monsters (

name text not null,

buff text not null,

buffDescription text not null,

gold integer not null,

perMinuteGold integer not null,

experience integer not null,

baseHealth integer not null,

baseRange integer not null,

baseAttackDamage integer not null,

baseArmor integer not null,

baseAttackSpeed integer not null,

baseMovementSpeed integer not null,

baseMagicResist integer not null,

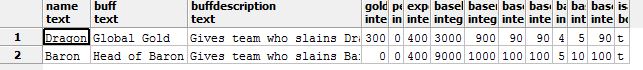
isActive boolean not null,

primary key (name)

);

**Functional dependencies:** name → buff, buffDescription, gold, perMinuteGold, experience, baseHealth, baseRange, baseAttackDamage, baseArmor, baseAttackSpeed, baseMovementSpeed, baseMagicResist, isActive

**Sample data:**

****

**Table:** MapMonsters

**Description:** Table for which monsters play on which map.

**Create statements:**

create table MapMonsters (

monster text not null references Monsters(name),

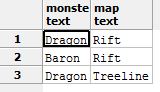
map text not null references Maps(name),

primary key (monster, map)

);

**Functional dependencies:** monster, map →

**Sample data:**

****

**Table:** ItemTypes

**Description:** Table for the types of items.

**Create statements:**

create table ItemTypes (

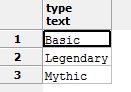
type text not null,

primary key (type)

);

**Functional dependencies:** type →

**Sample data:**

****

**Table:** Items

**Description:** Table of items that champions can use.

**Create statements:**

create table Items (

name text not null,

costGold integer not null,

isActive boolean not null,

releaseDate date not null,

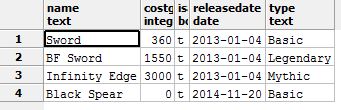
type text not null references ItemTypes(type),

primary key (name)

);

**Functional dependencies:** name → costGold, isActive, releaseDate, type

**Sample data:**

****

**Table:** BuildsInto

**Description:** Table that shows which items can be combined into a “better” item.

**Create statements:**

create table BuildsInto (

recipeItem text not null references Items(name),

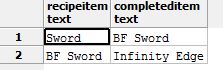
completedItem text not null references Items(name),

primary key (recipeItem, completedItem)

);

**Functional dependencies:** recipeItem, completedItem →

**Sample data:**

****

**Table:** ItemStatistics

**Description:** Table for bonuses given by a certain item.

**Create statements:**

create table ItemStatistics (

item text not null references Items(name),

category text not null references StatisticCategories(category),

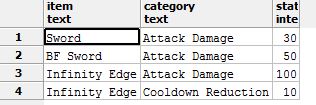
statistic integer not null,

primary key (item, category)

);

**Functional dependencies:** item, category → statistic

**Sample data:**

****

**Table:** Passives

**Description:** Table for passives of items. Unique items cannot give a champion the same passive if 2 or more are owned.

**Create statements:**

create table Passives (

name text not null,

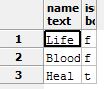
isUnique boolean not null,

primary key (name)

);

**Functional dependencies:** name → isUnique

**Sample data:**

****

**Table:** ItemPassives

**Description:** Table for which items get which passives.

**Create statements:**

create table ItemPassives (

item text not null references Items(name),

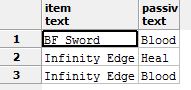
passive text not null references Passives(name),

primary key (item, passive)

);

**Functional dependencies:** item, passive →

**Sample data:**

****

**Table:** Actives

**Description:** Table for actives of items. Unique items cannot give a champion the same active ability if 2 or more are owned. Active items have a cooldown and duration timer.

**Create statements:**

create table Actives (

name text not null,

isUnique boolean not null,

durationSeconds integer not null,

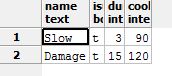
cooldownSeconds integer not null,

primary key (name)

);

**Functional dependencies:** name → isUnique, durationSeconds, cooldownSeconds

**Sample data:**

****

**Table:** ItemActives

**Description:** Table for which items get which actives.

**Create statements:**

create table ItemActives (

item text not null references Items(name),

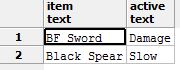
active text not null references Actives(name),

primary key (item, active)

);

**Functional dependencies:** item, active →

**Sample data:**

****

**Table:** Auras

**Description:** Table for auras of items. Unique items cannot give a champion the same aura if 2 or more are owned. Auras are basically passives with a range around the champion.

**Create statements:**

create table Auras (

name text not null,

range integer not null,

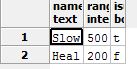
isUnique boolean not null,

primary key (name)

);

**Functional dependencies:** name → range, isUnique

**Sample data:**

****

**Table:** ItemAuras

**Description:** Table for which items get which auras.

**Create statements:**

create table ItemAuras (

item text not null references Items(name),

aura text not null references Auras(name),

primary key (item, aura)

);

**Functional dependencies:** item, aura →

**Sample data:**

**ItemAuras.JPG**

**Table:** AvailableItems

**Description:** Huge table for all of the current combinations of items, champions (characters), and maps in the game. If an entry lives in this table, then a summoner (person) can buy the item when playing the champion and the map that the entry contains.

**Create statements:**

create table AvailableItems (

item text not null references Items(name),

champion text not null references Champions(name),

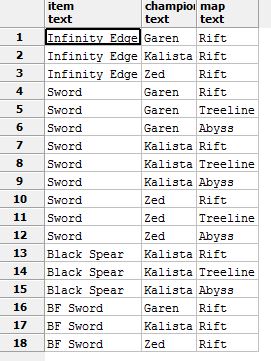
map text not null references Maps(name),

primary key (item, champion, map)

);

**Functional dependencies:** item, champion, map →

**Sample data:**

****

**Queries:**

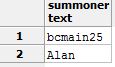
**Query:** Find all summoners who own champion “Kalista.”

**Code:**select summoner

from SummonerChampions

where champion = 'Kalista'

**Sample Output:**



**Query:** Find all champions who can buy an item that has the “Slow” active.

**Code:**select distinct ai.champion

from AvailableItems ai, Items i, ItemActives ia

where ai.item = i.name

and i.name = ia.item

and ia.active = 'Slow'

**Sample Output:**

**2.JPG**

**Stored Procedures:**

**Procedure:** Stored Procedure to find BuildsInto completedItems

**Code:**

create or replace function BuildsInto(text, REFCURSOR) returns refcursor as

$$

declare

item text := $1;

resultset REFCURSOR := $2;

begin

open resultset for

select completedItem

from BuildsInto

where recipeItem = item;

return resultset;

end;

$$

language plpgsql;

**Sample Procedure:**

select BuildsInto('Sword', 'results');

Fetch all from results;

**Sample Output:**

**3.JPG**

**Procedure:** Stored Procedure to find BuildsInto recipeItems

**Code:**

create or replace function BuildsInto(text, REFCURSOR) returns refcursor as

$$

declare

item text := $1;

resultset REFCURSOR := $2;

begin

open resultset for

select recipeItem

from BuildsInto

where completedItem = item;

return resultset;

end;

$$

language plpgsql;

**Sample Procedure:**

select BuildsInto('BF Sword', 'results');

Fetch all from results;

**Sample Output:**

**4.JPG**

**Security:**

This database consists of two types of users. The first is considered an administrator who is allowed to change, update, and maintain the database. To create this admin, the following code is used:

CREATE ROLE admin

GRANT SELECT, INSERT, UPDATE, ALTER

ON ALL TABLES IN SCHEMA PUBLIC

TO admin

The second is considered a typical user who can only see the database and view queries. To create this user, the following code is used:

CREATE ROLE user

GRANT SELECT

ON ALL TABLES IN SCHEMA PUBLIC

TO user

**Implementation Notes:**

The implementation of the above database was successful. Implementing this database in a real scenario would take much more time as the game contains over 170 champions, almost 2,000 items, and 27 million summoners. The sample data also just shows a small portion of the real database governing the game, since certain aspects of the game, such as Runes and Summoner Spells were ignored for simplicity. The code above has been ordered to avoid “orphan” tables as all parent tables are created before the child tables are.

**Known Problems:**

The dates of entries can be entered to not make sense. For example, a summoner can purchase a champion before the champion was released or the summoner was even created. This field will need to be auto-populated. Other known problems include: some of the columns are ambiguous as to if the statistic is a percentage or flat amount, champions cannot be created unless they have all of the columns even though there exists a champion “Urf” who was never finished but is still a champion with moves with no release date.

**Future Enhancements:**

The sample data created for this project was taken from the *League of Legends* wikia page. In future enhancements, data from the game would feed into the database and account for all parts of the game, such as runes and spells. Expansion of the database to contain more information regarding lore would also make the database much more accepting to the public as it will contain more information regarding to *League of Legends*.