League of Legends Database

By Brian Main

Database Management Fall 2014

Professor Labouseur

Table of Contents

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:

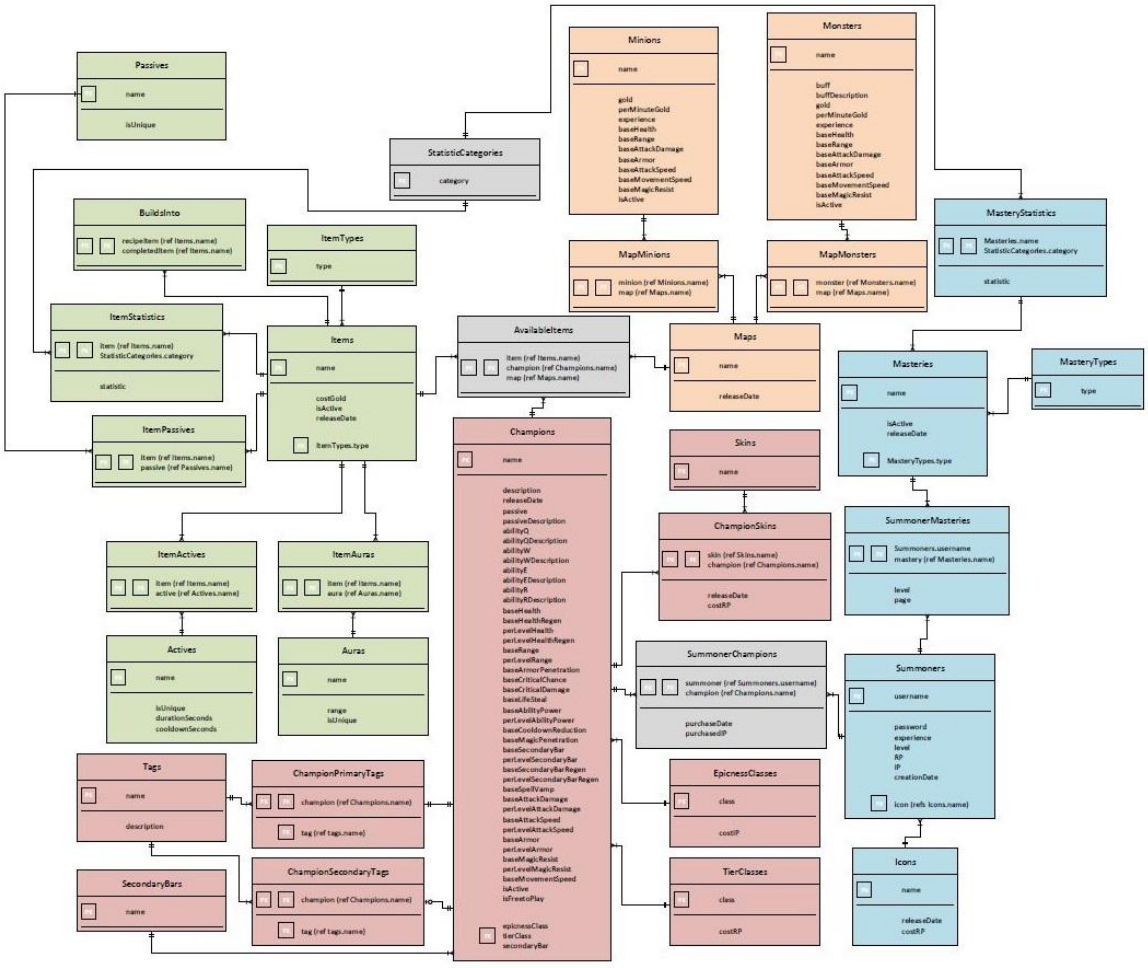

Table: Icons
Description: Table for icons used by summoner's profiles. Icons can be bought using Riot Points (RP).
Create statements:

```
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:

	name text	releasedate date	costrp integer
1	icon1	2010-01-01	250
2	blue	2014-02-20	300
3	red	2010-01-01	300

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:

	username character(24)	password character(16)	experience integer	level integer	rp integer	ip integer	creationdate date	icon text
1	Alan	pass123	999	29	0	0	2012-12-26	icon1
2	bcmain25	password	1000	30	999	125	2011-12-25	blue
3	guy	12348765	0	1	0	0	2013-03-01	red
4	girl	password	1000	30	50	0	2013-12-25	blue
5	TheOne	TheOne	1000	30	999999	999999	2012-01-01	red

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:

	category text
1	Attack Damage
2	Ability Power
3	Cooldown Reduction
4	Health

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:

	type text
1	Offensive
2	Defensive
3	Utility

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:

	name text	isactive boolean	releasedate date	type text
1	Double-Edged Sword	t	2013-11-25	Offensive
2	Expose Weakness	t	2013-11-25	Offensive
3	Block	t	2013-11-25	Defensive
4	Mastermind	f	2011-11-25	Utility
5	Intelligence	t	2013-11-25	Utility

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:

	name text	category text	statistic integer
1	Intelligence	Cooldown Reduction	15
2	Block	Health	100
3	Expose Weakness	Attack Damage	9
4	Expose Weakness	Ability Power	9
5	Mastermind	Cooldown Reduction	15

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:

	username text	mastery text	level integer	page integer
1	guy	Intelligence	1	4
2	guy	Expose Weakness	1	2
3	Alan	Block	2	1
4	bcmain25	Block	2	1

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:

	class text	costip integer
1	Heroic	450
2	Epic	1350
3	Legendary	6300

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:

	class text	costrp integer
1	Tier 1	260
2	Tier 2	790
3	Tier 3	975

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:

	name text
1	Mana
2	Health
3	Energy
4	Manaless

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:

	name text	description text	releasedate date	...	epicnessclass text	tierclass text	secondary text
1	Garen	the Might of Demacia	2010-04-27	...	Heroic	Tier 1	Manaless
2	Kalista	The Spear of Vengeance	2014-11-20	...	Legendary	Tier 3	Mana
3	Zed	the Master of Shadows	2012-11-13	...	Legendary	Tier 2	Energy

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:

	name text	description text
1	Mage	Damage-based champions whose abilities are typica
2	Fighter	A hybrid between DPS and tanking, these champions
3	Assassin	Champions which specialize in killing other heroe
4	Tank	Tanks are champions who are designed to take huge
5	Marksman	Champions that generally start off weaker than o

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:

	champion text	tag text
1	Garen	Fighter
2	Kalista	Marksman
3	Zed	Assassin

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:

	champion text	tag text
1	Garen	Tank
2	Zed	Fighter

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:

	name text
1	Shockblade
2	Rugged
3	Battlecast

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:

	skin text	champ text	releasedate date	cost inte
1	Battlecast	Garen	2011-01-01	900
2	Battlecast	Zed	2011-01-01	950
3	Shockblade	Zed	2013-02-22	350

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:

	summoner text	champion text	purchasedate date	pu bo
1	bcmain25	Zed	2013-02-22	t
2	bcmain25	Garen	2013-02-23	f
3	bcmain25	Kalista	2013-02-24	t
4	Alan	Zed	2013-02-22	f
5	Alan	Kalista	2013-02-25	t

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:

	name text	releasedate date
1	Rift	2011-06-01
2	Treeline	2011-06-01
3	Abyss	2013-01-24

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:

	name text	go int	pe in	ex int	bas inte	bas inte	ba int	ba int	bi in	bas inte	ba int	is bo
1	Caster	15	1	30	100	300	5	5	1	300	5	t
2	Siege	20	2	40	300	400	10	10	2	300	10	t

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:

	minion text	map text
1	Caster	Rift
2	Siege	Rift
3	Caster	Treeline
4	Siege	Abyss

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:

	name text	buff text	buffdescription text	gold inte	pe in	exp inte	basel integ	base integ	bas inte	bas inte	ba in	ba int	bas inte	is bo
1	Dragon	Global Gold	Gives team who slains Dr	300	0	400	3000	900	90	90	4	5	90	t
2	Baron	Head of Baron	Gives team who slains Ba	0	0	400	9000	1000	100	100	5	10	100	t

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:

	monste text	map text
1	Dragon	Rift
2	Baron	Rift
3	Dragon	Treeline

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:

	type text
1	Basic
2	Legendary
3	Mythic

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:

	name text	costg integ	is b(	releasedate date	type text
1	Sword	360	t	2013-01-04	Basic
2	BF Sword	1550	t	2013-01-04	Legendary
3	Infinity Edge	3000	t	2013-01-04	Mythic
4	Black Spear	0	t	2014-11-20	Basic

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:

	recipeitem text	completeditem text
1	Sword	BF Sword
2	BF Sword	Infinity Edge

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:

	item text	category text	stat inte
1	Sword	Attack Damage	30
2	BF Sword	Attack Damage	50
3	Infinity Edge	Attack Damage	100
4	Infinity Edge	Cooldown Reduction	10

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:

	name text	is bo
1	Life	f
2	Blood	f
3	Heal	t

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:

	item text	passiv text
1	BF Sword	Blood
2	Infinity Edge	Heal
3	Infinity Edge	Blood

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:

	name text	is bo	du int	cool inte
1	Slow	t	3	90
2	Damage	t	15	120

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:

	item text	active text
1	BF Sword	Damage
2	Black Spear	Slow

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:

	name text	rang inte	is bo
1	Slow	500	t
2	Heal	200	f

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:

	item text	aura text
1	Black Spear	Heal

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:

	item text	champion text	map text
1	Infinity Edge	Garen	Rift
2	Infinity Edge	Kalista	Rift
3	Infinity Edge	Zed	Rift
4	Sword	Garen	Rift
5	Sword	Garen	Treeline
6	Sword	Garen	Abyss
7	Sword	Kalista	Rift
8	Sword	Kalista	Treeline
9	Sword	Kalista	Abyss
10	Sword	Zed	Rift
11	Sword	Zed	Treeline
12	Sword	Zed	Abyss
13	Black Spear	Kalista	Rift
14	Black Spear	Kalista	Treeline
15	Black Spear	Kalista	Abyss
16	BF Sword	Garen	Rift
17	BF Sword	Kalista	Rift
18	BF Sword	Zed	Rift

Queries:
Query: Find all summoners who own champion “Kalista.”
Code:
```
select summoner
from SummonerChampions
where champion = 'Kalista'
```
Sample Output:

	summoner text
1	bcmain25
2	Alan

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:

	champion text
1	Kalista

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:

	completeditem text
1	BF Sword

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:

	recipeitem text
1	Sword

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*.