DotABase

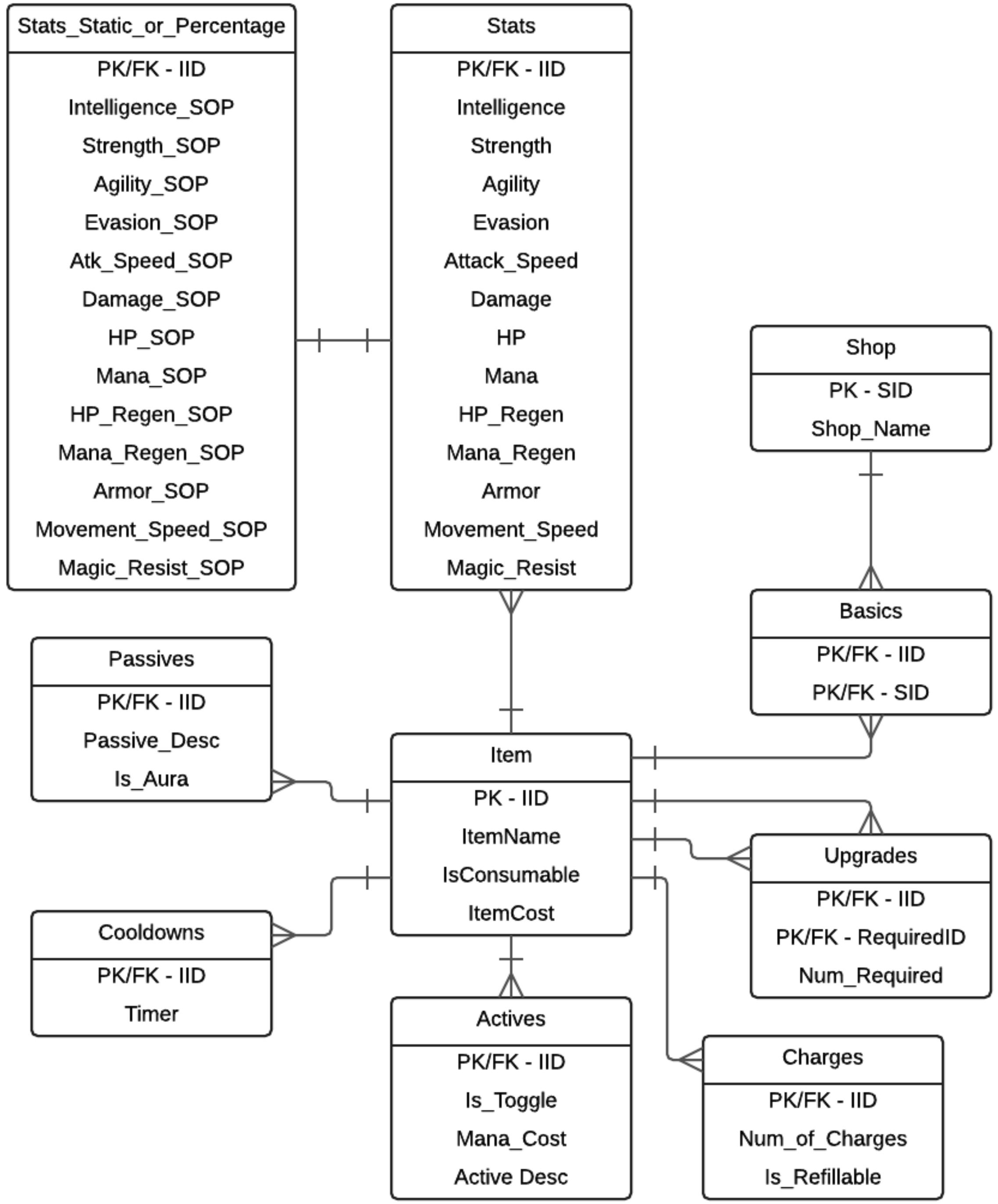
By Clayton Szelestey

Table of Contents

Executive Summary

DotA 2 is a free competitive multiplayer online battle arena video game. One of the draws of the game is the sheer amount of creativity the game offers you in terms of how you play. One of the main contributing factors to this freedom is the large number of items that the game allows you to use with any character. These items are all extremely varied in their affects and attributes. Unfortunately this can be daunting to new players, and the developers of DotA 2 do a poor job of creating learning tools within the game, and so it is often left up to the community to deconstruct and understand the mechanics of the game. The goal of DotABase is to create an organizational system for the items in DotA 2 so they can easily be sorted through and understood. This essay will go over the design of DotABase, the implementation of DotABase, some of the features of DotABase, and the future of DotAbase.

Entity-Relationship Diagram

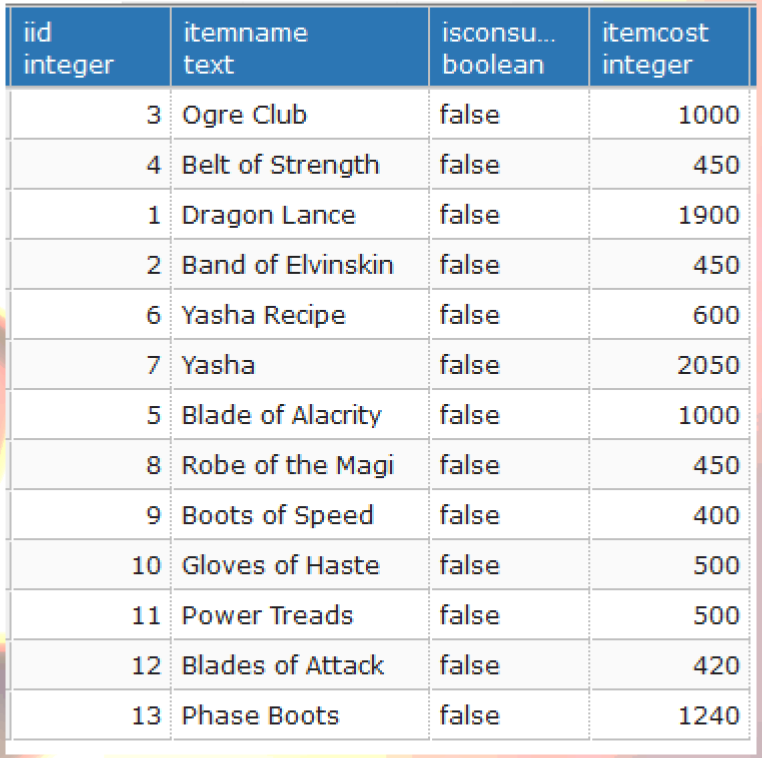


Items - Table

This table is the base for all items in DotA 2.

```
CREATE TABLE Items(
        IID int not null unique,
        ItemName text not null,
        IsConsumable bool not null,
        ItemCost int not null,
        PRIMARY KEY(IID)
);
```

IID → ItemName, IsConsumable, ItemCost



iid integer	itemname text	isconsu... boolean	itemcost integer
3	Ogre Club	false	1000
4	Belt of Strength	false	450
1	Dragon Lance	false	1900
2	Band of Elvinskin	false	450
6	Yasha Recipe	false	600
7	Yasha	false	2050
5	Blade of Alacrity	false	1000
8	Robe of the Magi	false	450
9	Boots of Speed	false	400
10	Gloves of Haste	false	500
11	Power Treads	false	500
12	Blades of Attack	false	420
13	Phase Boots	false	1240

Stats - Table

This table determines the basic boosts that your character gets if they are holding an item that gives stat boosts.

IID → Intelligence, Strength, Agility, Evasion, Attack_Speed, Damage, HP, Mana, HP_Regen, Armor, Movement_Speed, Magic_Resist

```
CREATE TABLE Stats(
   IID int not null unique references Items(IID),
   Intelligence int,
   Strength int,
   Agility int,
   Evasion int,
   Attack_Speed int,
   Damage int,
   HP int,
   Mana int,
   HP_Regen int,
   Mana_Regen int,
   Armor int,
   Movement_Speed int,
   Magic_Resist int,
   PRIMARY KEY(IID)
);
```

Stats_Static_or_Percentage - Table

This table determines whether or not the stats in the stats table are based on simply adding a number of points, or if they are percentage based.

IID → Intelligence_SOP, Strength_SOP, Agility_SOP, Evasion_SOP, Attack_Speed_SOP, Damage_SOP, HP_SOP, Mana_SOP, HP_Regen_SOP, Armor_SOP, Movement_Speed_SOP, Magic_Resist _SOP

```
CREATE TABLE Stats_Static_or_Percentage(
    IID int not null unique references Stats(IID),
    Intelligence_SOP bool,
    Strength_SOP bool,
    Agility_SOP bool,
    Evasion_SOP bool,
    Attack_Speed_SOP bool,
    Damage_SOP bool,
    HP_SOP bool,
    Mana_SOP bool,
    HP_Regen_SOP bool,
    Mana_Regen_SOP bool,
    Armor_SOP bool,
    Movement_Speed_SOP bool,
    Magic_Resist_SOP bool,
    PRIMARY KEY(IID)
);
```

Passives - Table

This table describes the bonuses that items give players that are innate to the item, and do not have to be manually activated.

```
CREATE TABLE Passives(
        IID int not null unique references Items(IID),
        Passive_Desc text not null,
        Is_Aura bool not null,
        PRIMARY KEY(IID)
);
```

IID → Passive_Desc, Is_Aura

Cooldowns - Table

This table contains the timer for the period of time that is required before an items ability can be used a second time.

```
CREATE TABLE Cooldowns(
        IID int not null unique references Items(IID),
        Timer int not null,
        PRIMARY KEY(IID)
);
```

IID → Timer

Actives Table - Table

This table describes the abilities that items have that need to be manually activated by the player in order to take effect.

```
CREATE TABLE Actives(
	IID int not null unique references Items(IID),
	Active_Desc text not null,
	Is_Toggle bool not null,
	Mana_Cost int,
	PRIMARY KEY(IID)
);
```

IID → Active_Desc, Is_Toggle, Mana_Cost

Charges - Table

Some items can only be used a certain number of times before they expire or need to be refilled, as this table shows.

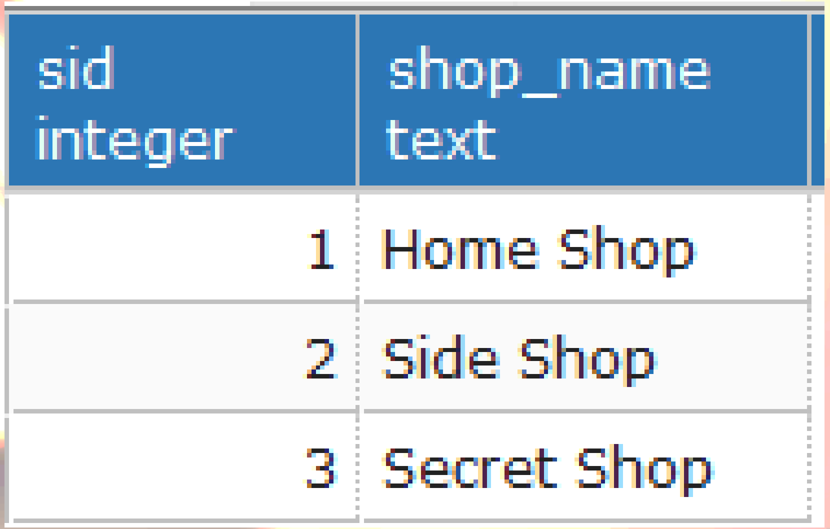
```
CREATE TABLE Charges(
	IID int not null unique references Items(IID),
	Num_of_Charges int not null,
	Is_Refillable bool not null,
	PRIMARY KEY(IID)
);
```

IID → Num_of_Charges, Is_Refillable

Shops - Table

There are three different kinds of shops in DotA that all contain different items. This table sets up IDs and names for the shops.

```
CREATE TABLE Shop(
        SID int not null unique,
        Shop_Name text not null,
        PRIMARY KEY(SID)
);
```



sid integer	shop_name text
1	Home Shop
2	Side Shop
3	Secret Shop

SID → Shop_Name

Basics - Table

Basic items are items that you can buy in a shop on their own.

```
CREATE TABLE Basics(
        IID int not null references Items(IID),
        SID int not null references Shop(SID),
        PRIMARY KEY(IID, SID)
);
```

(IID, SID)

Upgrades - Table

Upgrades are items that can only be obtained purchasing two or more basic items and combining them into one more powerful item.

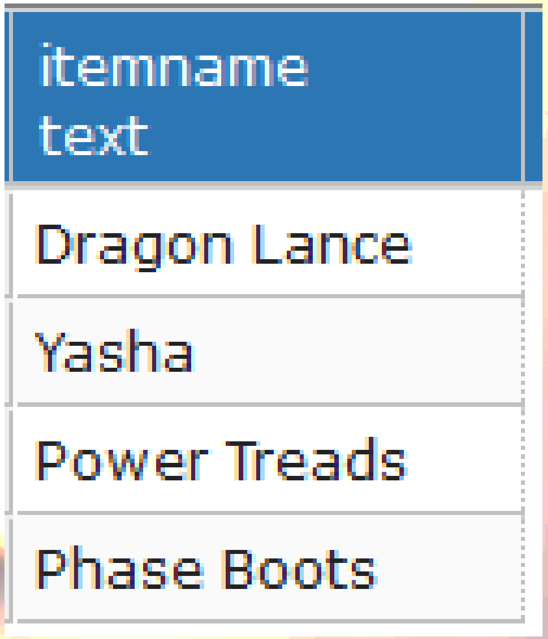
```
CREATE TABLE Upgrades(
	IID int not null references Items(IID),
	RequiredID int not null references Items(IID),
	Num_Required int not null,
	PRIMARY KEY(IID, RequiredID)
);
```

IID → RequiredID, Num_Required

UpgradableItems - View

This view displays a list of all Upgrade type items.

```
Create or Replace View UpgradableItems AS
        select ItemName
        from Items
        where IID in(
           select IID
           from Upgrades);
```

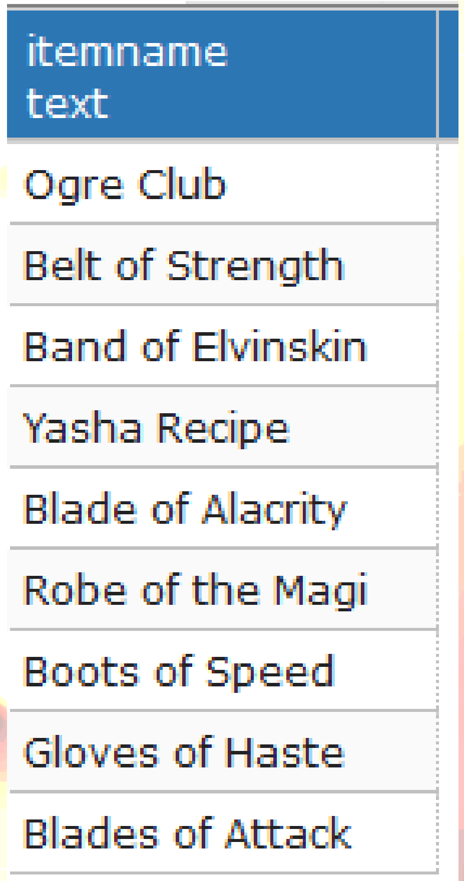


itemname text
Dragon Lance
Yasha
Power Treads
Phase Boots

BasicItems - View

This view displays a list of all Basic type items.

```
Create or Replace View BasicItems AS
        select ItemName
        from Items
        where IID in(
          select IID
          from Basics);
```



itemname text
Ogre Club
Belt of Strength
Band of Elvinskin
Yasha Recipe
Blade of Alacrity
Robe of the Magi
Boots of Speed
Gloves of Haste
Blades of Attack

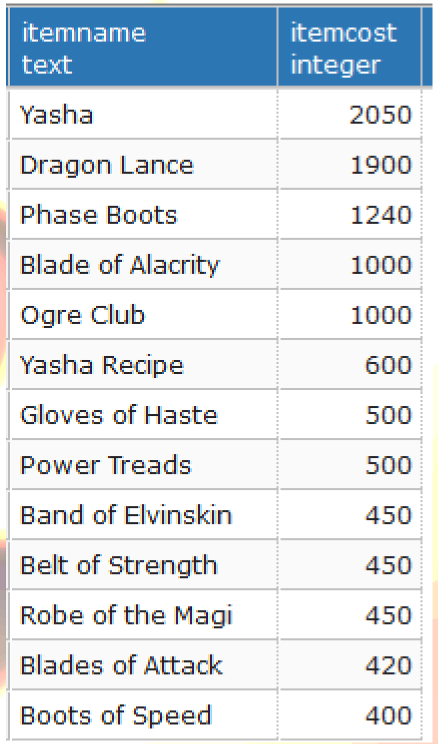
Reports

This query shows a list of all items in the game, from the most expensive to the least expensive.

Select ItemName, ItemCost

from Items

Order By ItemCost DESC;

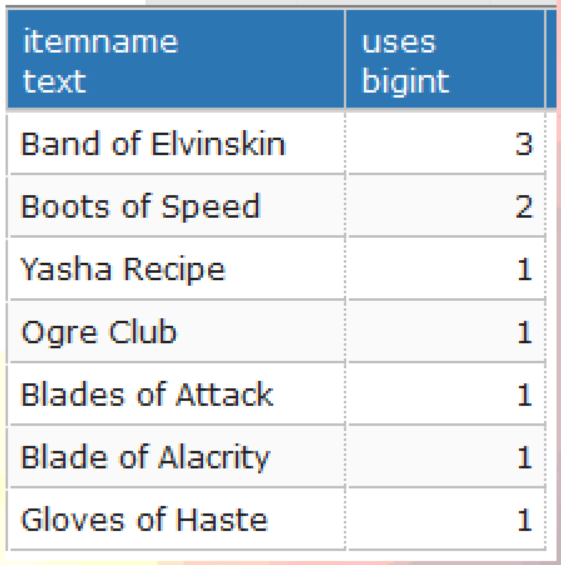


itemname text	itemcost integer
Yasha	2050
Dragon Lance	1900
Phase Boots	1240
Blade of Alacrity	1000
Ogre Club	1000
Yasha Recipe	600
Gloves of Haste	500
Power Treads	500
Band of Elvinskin	450
Belt of Strength	450
Robe of the Magi	450
Blades of Attack	420
Boots of Speed	400

Reports

This query displays a list of the most common basic items that are used in combining to create upgraded items.

```
Select ItemName, Count(Num_Required) as Uses
from Items, Upgrades
Where Items.IID = Upgrades.RequiredID
Group By ItemName
Order By Uses DESC;
```

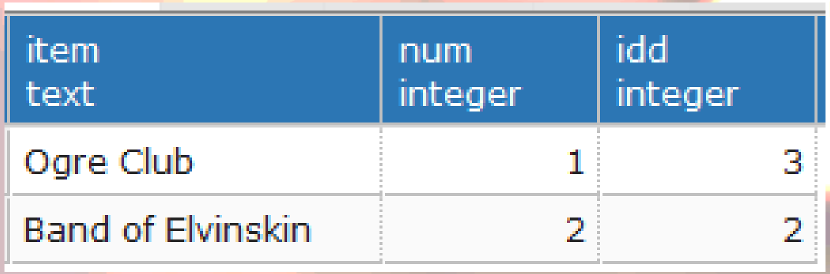


itemname text	uses bigint
Band of Elvinskin	3
Boots of Speed	2
Yasha Recipe	1
Ogre Club	1
Blades of Attack	1
Blade of Alacrity	1
Gloves of Haste	1

Required - Stored Procedures

This procedure takes in the ID of an Upgraded item and displays the necessary components to create it.

Dragon Lance (ID – 1):



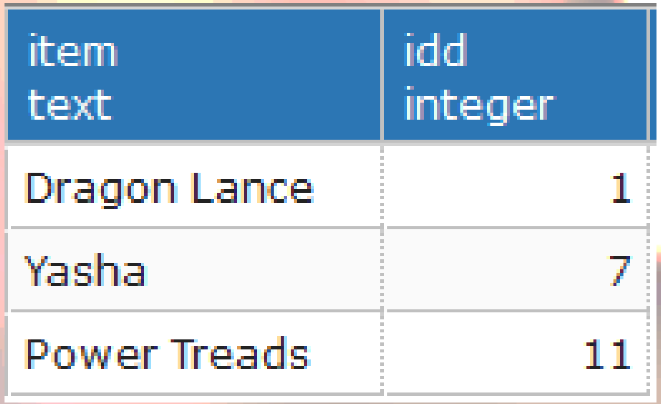
item text	num integer	idd integer
Ogre Club	1	3
Band of Elvinskin	2	2

```
create or replace function Required(int) returns TABLE(item text, num int, idd int) as
$$
declare
  searchIID   int      := $1;
begin
  return QUERY
  select
     Items.ItemName,
     Upgrades.Num_Required,
     Items.IID
  from Items join Upgrades on Upgrades.RequiredID = Items.IID
  where Upgrades.IID = searchIID;
end;
$$
language plpgsql;
```

RequiredFor - Stored Procedures

This procedure takes in the ID of a Basic item and displays which Upgraded items can be made out of it.

Band of Elvinskin (ID – 2):



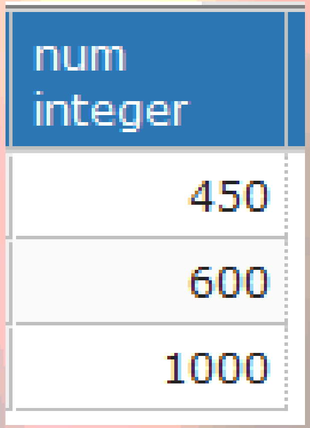
item text	idd integer
Dragon Lance	1
Yasha	7
Power Treads	11

```
create or replace function RequiredFor(int) returns TABLE(item text, idd int) as
$$
declare
  searchIID   int       := $1;
begin
  return QUERY
  select
     Items.ItemName,
     Items.IID
  from Items join Upgrades on Upgrades.IID = Items.IID
  where Upgrades.RequiredID = searchIID;
end;
$$
language plpgsql;
```

ComponentPrice - Stored Procedures

This procedure takes in the ID of an Upgraded weapon and displays the prices of the individual components of the item.

Yasha (ID – 7):



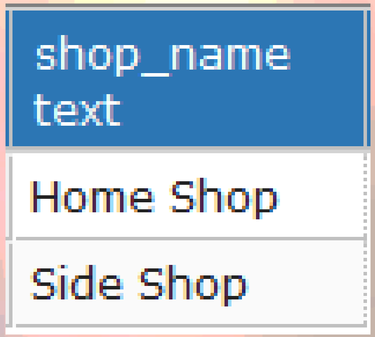
num integer
450
600
1000

```
create or replace function componentPrice(int) returns TABLE(num int) as
$$
declare
  searchIID   int       := $1;
begin
  return QUERY
  select
     Items.ItemCost * Upgrades.Num_Required
  from Items join Upgrades on Upgrades.RequiredID = Items.IID
  where Upgrades.IID = searchIID;
end;
$$
language plpgsql;
```

findLocation - Stored Procedures

This procedure takes in the ID of a Basic item and displays which stores the item can be purchased at.

Belt of Strength (ID – 4):

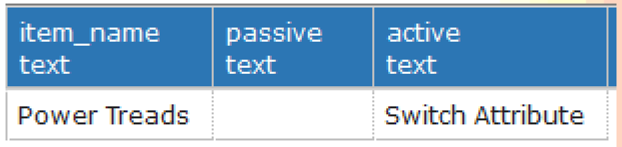


shop_name text
Home Shop
Side Shop

```
create or replace function findLocation(int) returns TABLE (Shop_Name text) as
$$
declare
  searchIID   int       := $1;
begin
  return QUERY
  select Shop.Shop_Name
  from Shop
  where SID in (
      select SID
      from Basics
      where IID = searchIID
   );
end;
$$
language plpgsql;
```

ItemFunc - Stored Procedures

```
create or replace function ItemFunc(int) returns TABLE (Item_Name text, Passive text, Active text) as
$$
declare
  searchIID   int       := $1;
begin
  return QUERY
  select Items.ItemName,
         case when Exists(Select Passives.IID from Passives where Passives.IID = Items.IID) then (select Passives.Passive_Desc from passives where Passives.IID = Items.IID)
         else null end as Passive,
         case when Exists(Select Actives.IID from Actives where Actives.IID = Items.IID) then (select Actives.Active_Desc from Actives where Actives.IID = Items.IID)
         else null end as Active

  from Items
  where Items.IID = searchIID;
end;
$$
language plpgsql;
```



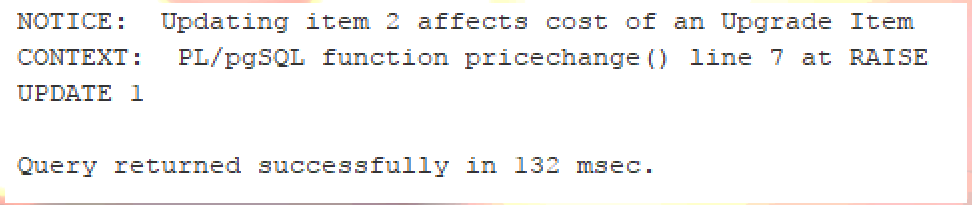
item_name text	passive text	active text
Power Treads		Switch Attribute

This procedure takes in an item ID and displays both passive effects and active effects if the item happens to have any.

priceChange - Triggers

```
CREATE OR REPLACE FUNCTION priceChange()
RETURNS TRIGGER AS
$$
DECLARE
   affectedID        int;
BEGIN
   IF new.ItemCost != Items.ItemCost from Items where IID = new.IID AND (EXISTS(select idd from RequiredFor(new.IID))) THEN
   RAISE NOTICE 'Updating item % affects cost of an Upgrade Item',
      new.IID;
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

create trigger priceWarning
BEFORE INSERT OR UPDATE ON Items
                       FOR EACH ROW EXECUTE PROCEDURE priceChange();
```

This trigger warns the user if they are changing the price of a basic item that will affect the price of its upgraded counterpart, as Upgraded item prices are simply the sum of their parts.

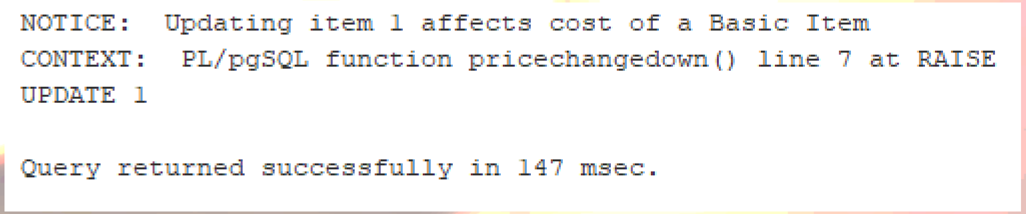


```
NOTICE:  Updating item 2 affects cost of an Upgrade Item
CONTEXT:  PL/pgSQL function pricechange() line 7 at RAISE
UPDATE 1

Query returned successfully in 132 msec.
```

priceChangeDown - Triggers

```
CREATE OR REPLACE FUNCTION priceChangeDown()
RETURNS TRIGGER AS
$$
DECLARE
  affectedID      int;
BEGIN
  IF new.ItemCost != Items.ItemCost from Items where IID = new.IID AND (EXISTS(select idd from Required(new.IID))) THEN
  RAISE NOTICE 'Updating item % affects cost of a Basic Item',
    new.IID;
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

create trigger priceWarningDown
BEFORE INSERT OR UPDATE ON Items
                FOR EACH ROW EXECUTE PROCEDURE priceChangeDown();
```

This trigger warns the user if they are changing the price of an upgraded item that will cause inconsistencies with the pricing of its base components, as Upgraded items are priced based on the sum of their parts



```
NOTICE:  Updating item 1 affects cost of a Basic Item
CONTEXT:  PL/pgSQL function pricechangedown() line 7 at RAISE
UPDATE 1

Query returned successfully in 147 msec.
```

Security

There are two roles for DotABase, dev and player. Devs have the ability to change anything they wish, and players do not have the ability to change anything. This mimics how the actual game works and players cannot go into the files and change the values however they like.

```
CREATE ROLE dev;
GRANT ALL ON ALL TABLES IN SCHEMA public TO dev;

CREATE ROLE player;
REVOKE ALL ON ALL TABLES IN SCHEMA public FROM player;
```

Implementation Notes

- Each stat in DotA can be applied in either a static fashion (+6 HP REGEN) or in a percentage fashion (+6% HP REGEN). The static fashion would mean you gain an extra 6 HP per second, while the percentage fashion would mean you gain 6% of your total health per second. This is why the two stat tables were necessary
- There is some irrelevant data that I left out of certain tables. For example, each store type actually has two different sides (Radiant and Dire side). However each type of store still has the same items in its inventory so this information is not useful in the current implementation of DotABase

Known Problems

- There is an Upgraded item in DotA 2 that can be assembled three separate ways (Power Treads), but is the same item once assembled. This is not currently reflected in DotABase
- The componentPrice procedure, while accurate, does not display separate instances of an item if more than one is used to create an Upgraded item
- The Stats and Stats_Static_or_Percentage could be combined into a single table to be more efficient, but DotABase does not deal with extremely large amounts of data so I found this to be unnecessary. I found separating the tables helped with readability in terms of the E/R Diagram

Future Enhancements

- More functionality would be ideal in the future, as DotA 2 is a very robust game.
- Ability to see win rates of certain items
- The ability to see only the attributes that pertain to a specific item
- More search functionality, such as looking up agility based items, or items that restore mana
- Expanding to include not only items, but heroes so that it can be complete DotABase that encompasses all aspects of the game
- Enhanced readability and polish