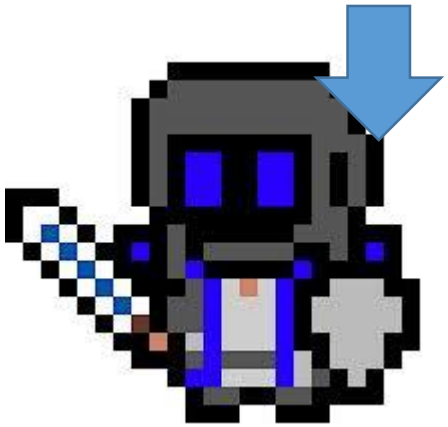
Level Down



By Richard Liao

Table of Contents

Executive Summary

The goal of this database is to create a simple yet powerful way to store all actors and data in the game “Level Down”. While it is possible to store all of the parts of data as “objects” (i.e. Enemies and items) in Level Down, it is extremely inefficient and difficult to manage. A database is required to keep all objects and other events tracked such as high-scores or player data. This will allow an admin to utilize a special set of tools to manage “Level Down” and hopefully maintain balance where ever necessary.

Game Overview

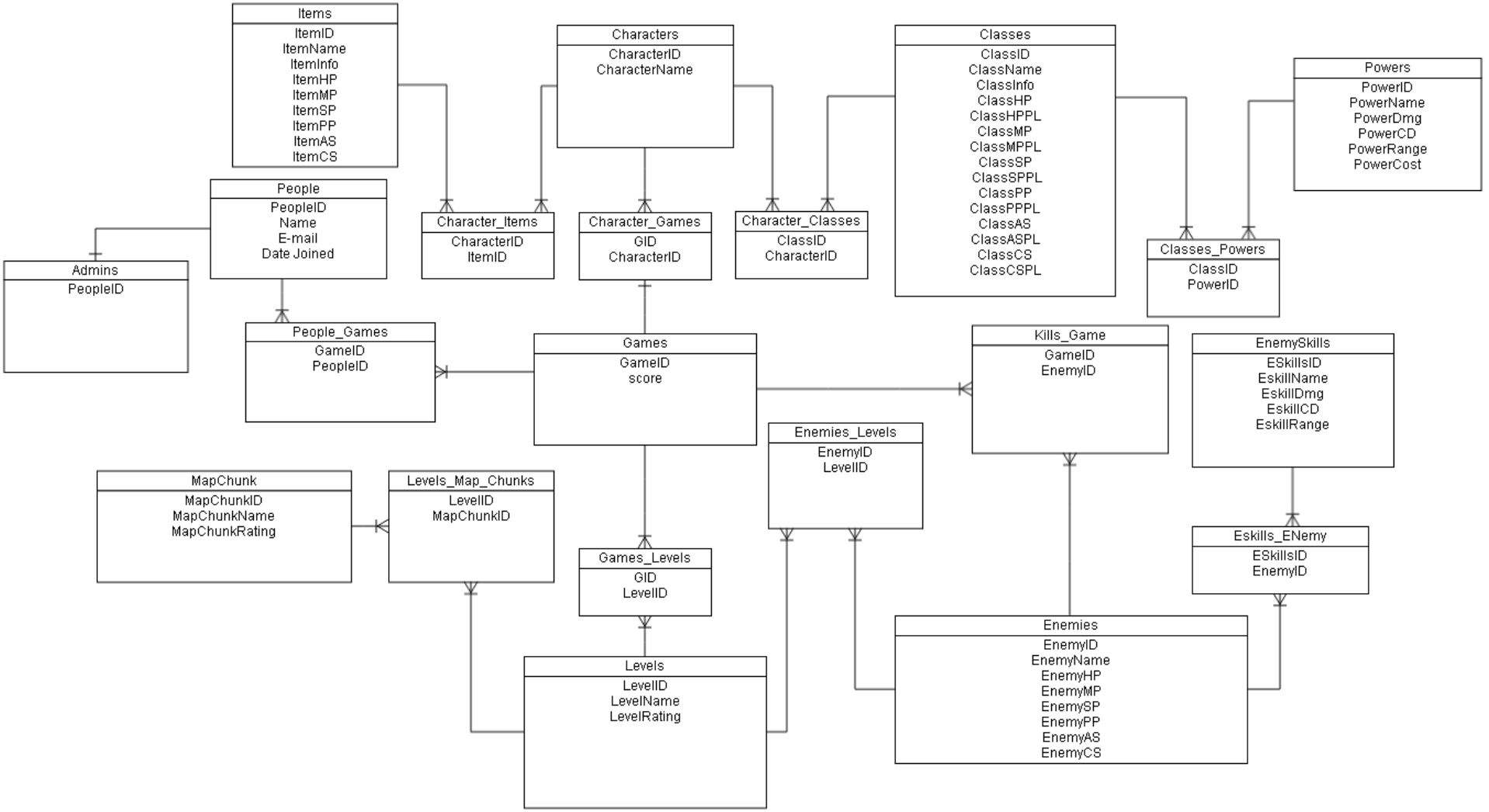
To fully understand this database and its existence, it is imperative the game is understood first. The premise of the game is as follows: You are the demon king, but instead of conquering and slaughtering the masses, you have opted to stay secluded and mostly uninvolved. This is a problem for the Hero's Guild as they are not benefitting from the usual wartime economy. To "combat" your inaction, they have assaulted your castle in an attempted coup. Although you managed to repel the attack, the Guild Leader left you with a curse that drains your powers and a note detailing a back-up plan to start pillaging and pin the deaths on you, the demon king. You must now stop the guild's leader before it is too late.

The main twist of the game is that instead of the normal convention of leveling up, the player will instead level down. The game is a rogue-like game; the player will have one chance to reach the end of a slightly randomized set of levels in order to beat the game. The player starts from their most powerful state and end up at their weakest. If they fail, their character is gone forever and they must try again. At the start, the player chooses up to 3 classes that make up a character. These characters start with a set of items and skills. Every time a player kills a boss or levels down, they will choose a power that they want to give up. However, whenever a power is lost by the player, the last boss gains it. This will force players to pick and choose powers based on their usefulness to their current item set, skill set, power, and what they want to sabotage the boss with. A power may eventually cost too much MP, so it may be beneficial to drop it because, while it does a large amount of damage, it is too costly to use and will hurt the final boss' MP pool. Leveling down, instead of up, creates a new kind of strategy that can change on the fly.

The issue arises that a player might try to simply run past the level without killing anything. This is where the enemies play a large role. As the player proceeds, they will fight bosses that force them to lose a level (if they win). As such, their current equipment will no longer be useable as they have become too low level to use it. Enemies scattered through-out the level will drop gear, but also level the player down, creating a risk reward style of game play. Learning enemy attack patterns is much more important than anything else in the game; items and skills are unreliable.

Database: E/R Diagram

Note: This E/R Diagram is set up in a very awkward way; due to its size it had to be altered to fit in this document. It is still logically the same, even though the tables may not be presented in a way that is pleasing to the eye.



Database: Items Table

The purpose of this table is to store all the items that will be obtainable by the player. All items give the player HP(hit points), MP(Magic Points), SP(Spell power), PP(Physical power), AS(Attack Speed), and CS(Cast Speed). Items are tracked by a unique ID called ItemID, the primary key.

SQL Create Statement

```
create table Items
(
ItemID SERIAL not null,
ItemName varchar (255),
ItemInfo varchar(255),
ItemHP int,
ItemMP int,
ItemSP int,
ItemPP int,
ItemAS int,
ItemCS int,
primary key (ItemID));
```

Functional Dependency

ItemID→ ItemName, ItemInfo, ItemHP, ItemMP, ItemSP, ItemPP, ItemAS, ItemCS

Sample Data

	itemid integer	itemname character varying(255)	iteminfo character varying(255)	itemhp integer	itemmp integer	itemsp integer	itempp integer	itemas integer	itemcs integer
1	1	Demonic Longsword	This sword swirls with evil energy	90	20	10	10	20	50
2	2	Book of the Dead	An empty book	0	200	100	0	0	500
3	3	Flabbos Leg of Meat	A leg of ham. Why are you using this	1000	0	0	0	0	0
4	4	BB gun	You ll shoot yer eye out	0	0	10	0	500	0
5	5	Slingshot	Some weapon you stole from a small child.	0	0	0	140	120	0

Database: Characters table

This table will hold all names of characters. It will serve as the “hub” where characters and their information is stored.

SQL Create Statement

create table Characters

(

CharacterID SERIAL not null,

CharacterName varchar (255) not null,

primary key (CharacterID)

);

Functional Dependency

CharacterID→ CharacterName

Sample Data

	characterid integer	charactername character varying(255)
1	1	Jimbo
2	2	Salami Sam
3	3	Bacon Barry
4	4	Cherry Canute
5	5	Shrimp Shrek
6	6	Salad Sally

Database: Classes Table

This table stores Classes and all their properties. Classes have HP, HP per level, MP, MP per level, etc.

SQL Create Statement

```
create table Classes(
ClassID SERIAL not null,
ClassName varchar(255),
ClassInfo varchar(255),
ClassHP int,--Hitpoints
ClassHPPL int,--hitpoints per level (lost per level)
ClassMP int,
ClassMPPL int,
ClassSP int,
ClassSPPL int,
ClassPP int,
ClassPPPL int,
ClassAS int,
ClassASPL int,
ClassCS int,
ClassCSPL int,
primary key (ClassID));
```

Functional Dependency

ClassID→ ClassName, ClassInfo, ClassHP, ClassHPPL, ClassMP, ClassMPPL, ClassSP, ClassSPPL,
. ClassPP, ClassPPPL, ClassAS, ClassASPL, ClassCS, ClassCSPL.

Sample Data

classid integer	classname character varying(255)	classinfo character varying(255)	classhp integer	classhppl integer	classmp integer	classmppl integer	classsp integer	classsppl integer	classpp integer	classpppl integer	classas integer	classaspl integer	classcs integer	classcspl integer
1	Bacon Mage	All the powers of bacon are at your disposal	100	10	20	30	10	10	10	10	10	10	10	10
2	Onion Knight	Your prowess to make the breath of enemies smell is unmatched	200	130	210	310	110	110	110	110	120	150	110	120
3	Vampire Archer	You.Do.Not.Sparkle	1000	10	10	30	70	20	1320	140	160	110	120	10
4	Mage	What? Thats it?	100	1330	420	3560	1650	540	1560	140	160	10	10	100
5	Innocent Bystande	You dont know why youre here, but you really feel like you should start running	1300	160	220	350	810	510	810	140	120	120	120	140
6	Shadow	Youre what happens when an object blocks rays of light	1200	170	280	330	10	130	10	1550	10	1880	10	130

Database: MapChunks Table

This table will hold the map chunks for use in the levels Table.

SQL Create Statement

create table MapChunk(

MapChunkID SERIAL not null,

MapChunkName varchar(255),

MapChunkRating int,

primary key(MapChunkID)

);

Functional Dependency

MapChunkID→ MapChunkName, MapChunkRating

Sample Data

	mapchunkid integer	mapchunkname character varying(255)	mapchunkrating integer
1	1	Alter	2
2	2	Dessert	1
3	3	Desert	3
4	4	Dark Musty Bathroom	4
5	5	Boss Room	5
6	6	Woods	2
7	7	Cow Level	12

Database: Levels table

This table holds all the levels. Each level has a rating and will be organized as such in the actual game.

SQL Create Statement

create table Levels(

LevelID Serial not null,

LevelName varchar(255),

LevelRating int,

primary key(LevelID));

Functional Dependency

LevelID→LevelName, LevelRating

Sample Data

	levelid integer	levelname character varying(255)	levelrating integer
1	1	Desert or Dessert?	10
2	2	Cow Level	12
3	3	Dreams	8
4	4	To the End	17

Database: Games Table

This table will serve as a hub for all the game information.

SQL Create Statement

Create table Games(

GameID Serial not null,

score int,

primary key(GameID)

);

Functional Dependency

GameID→ Score

Sample Data

	gameid integer	score integer
1	1	123242
2	2	123242422
3	3	1252522
4	4	633733
5	5	99999
6	6	6666666
7	7	0

Database: People

This table holds all the people. People are all players, but not all players will be admins

SQL Create Statement

Create table People(

PeopleID SERIAL not null,

Name varchar(255),

Email varchar(255),

JoinDate DATE,

Primary Key(PeopleID)

);

Functional Dependency

PeopleID→Name, Email, JoinDate

Sample Data

	peopleid integer	name character varying(255)	email character varying(255)	joindate date
1	1	Richard	Richard@gmail.com	1010-10-10
2	2	Ricardo	Ricardo@gmail.com	2110-12-13
3	3	Richy Rich	RichyR321@gmail.com	10010-01-04
4	4	Dick	HamMan@gmail.com	6010-11-10
5	5	Rich	Rich321@gmail.com	2010-09-10
6	6	Richy	R61345@gmail.com	2000-03-23

Database: Admins

The Admins table will hold the PeopleID of all the people will special permissions.

SQL Create Statement

Create table Admins(

PeopleID Int,

Foreign key (PeopleID) references People(PeopleID)

);

Functional Dependency

PeopleID→

Sample Data

	peopleid integer
1	1
2	2

Database: Powers Table

This table contains all the powers that the player can use. Each power has damage, cool down, range, and MP cost

SQL Create Statement

Create table Powers(

PowerID SERIAL not null,

PowerName varchar(255),

PowerDmg int,

PowerCD int,

PowerRange int,

PowerCost int,

Primary key(PowerID));

Functional Dependency

PowerID→ PowerName, PowerDmg, PowerCD, PowerRange, PowerCost

Sample Data

	powerid integer	powername character varying(255)	powerdmg integer	powercd integer	powerrange integer	powercost integer
1	1	Laser of Death	12230	5	100	500
2	2	Death	9999999	0	100	5020
3	3	Ham Slap	240	150	2500	1200
4	4	Onion Breath	120	60	300	100
5	5	Lightning Aura	25203	0	30	300
6	6	Blend and stab	50	10	20	10
7	7	Corpse Explosion	1000	10	320	100
8	8	Corpse rotation	0	0	110	300

Database: Enemies Table

The Enemies table holds all the enemies that the player will fight in the game.

SQL Create Statement

```
Create table Enemies(
EnemyID SERIAL not null,
EnemyName varchar(255),
EnemyInfo varchar(255),
EnemyType varchar(255),
ETypeHP int,
ETypeMP int,
ETypeSP int,
ETypePP int,
ETypeAS int,
ETypeCS int,
Primary key(EnemyID)
);
```

Functional Dependency

EnemyID→EnemyName, EnemyInfo, EnemyType,

Sample Data

enemyid integer	enemyname character varying(255)	enemyinfo character varying(255)	enemytype character varying(255)	enemyhp integer	enemymp integer	enemysp integer	enemypp integer	enemyas integer	enemycs integer
1	Cat	Looks like a cat. Sounds like a cat. Is a cat	Ranged	10	0	0	0	0	0
2	Destroyer of World	Admittedly small worlds, but worlds none the less	Melee	10	50	40	30	90	10
3	Mind Slug	Eats the minds of youths. Also connects users wirelessly and cheaply!	Magic	10	30	760	10	160	30
4	Explosive Elemental	EXPLODES.DONT TOUCH. BOOM	Magic	30	30	60	10	70	240
5	Reverse Centaur	Half man, half horse. Looks a bit lopsided	Magic	240	2404	470	2470	850	10
6	Scythe Wielding man	Cuts people like wheat	Melee	840	560	960	960	30	20
7	Glass Cannon	Doesnt even shoot the cannon ball far...	Ranged	970	450	450	220	20	20
8	Pots	The mortal enemy of a particular adventurer.	Melee	240	720	230	830	3580	350
9	The Black Knight	Pretty much immortal. Dismemberment does nothing.	Melee	35	350	220	30	220	20

Database: Enemy Skills Table

This table contains the skills that enemies can use.

SQL Create Statement

Create table EnemySkills(

ESkillsID SERIAL not null,

ESkillsName varchar(255),

ESkillsDMG int,

ESkillsCD int,

ESkillsRange int,

primary key(ESkillsID));

Functional Dependency

ESkillsID→EskillsName, EskillsDMg, EskillsCD, EskillsRange

Sample Data

	eskillsid integer	eskillsname character varying(255)	eskillsdmg integer	eskillscd integer	eskillsrange integer
1	1	Dark Touch	10	0	110
2	2	Dark Blast	100	10	10
3	3	Dark Ascension	1000	20	10
4	4	Dark Dark	110	0	110
5	5	Arrow Shower	1101	10	160
6	6	Fire Arrow	50	5	110
7	7	Barbed Arrow	110	0	1110
8	8	Explosive Arrow	1150	0	2110
9	9	Hide	0	0	0
10	10	Do nothing	0	0	0
11	11	Avoid Danger	0	0	0
12	12	Self Preservation	0	0	0

Database: Character_Items Table

This table stores which characters have which items

SQL Create Statement

Create table Character_Items(

CharacterID int,

ItemID int,

Foreign Key (CharacterID) References Characters(CharacterID),

Foreign Key(ItemID) References Items(ItemID));

Functional Dependency

ItemID→

CharacterID→

Sample Results

	characterid integer	itemid integer
1	1	1
2	2	1
3	1	2
4	2	2
5	3	4
6	1	3
7	6	1
8	5	5
9	4	4

Database: Character_Classes Table

This table stores all the classes that each character has.

SQL Create Statement

Create table Character_classes(

CharacterID int,

ClassID int,

Foreign Key (CharacterID) References Characters(CharacterID),

Foreign Key(ClassID) References Classes(ClassID));

Functional Dependency

CharacterID→

ClassID→

Sample Data

	characterid integer	classid integer
1	1	1
2	3	1
3	2	2
4	5	3
5	4	3
6	4	6
7	2	4
8	4	4
9	4	5
10	1	5

Database: Character_Games Table

This table stores all the games that characters have been in.

SQL Create Statement

Create table Character_Games(

CharacterID int,

GameID int unique,

Foreign Key (CharacterID) References Characters(CharacterID),

Foreign Key(GameID) References Games(GameID));

Functional Dependency

CharacterID→

GameID→

Sample Data

	characterid integer	gameid integer
1	6	7
2	1	1
3	2	2
4	3	3
5	6	4
6	4	5
7	5	6

Database: People_Games Table

The People_Games table shows what people have been in what games.

SQL Create Statement

Create table People_Games(

GameID int unique,

PeopleID int,

Foreign Key (GameID) References Games(GameID),

Foreign Key(PeopleID) References People(PeopleID));

Functional Dependency

GameID→

PeopleID→

Sample Data

	gameid integer	peopleid integer
1	7	6
2	6	6
3	5	5
4	4	4
5	3	3
6	2	2
7	1	1

Database: Games_Levels Table

This table shows what levels are in what games.

SQL Create Statement

Create table Games_Levels(

GameID int,

LevelID int,

Foreign Key (GameID) References Games(GameID),

Foreign Key(LevelID) References Levels(LevelID)

);

Functional Dependency

GameID→

LevelID→

Sample Data

	gameid integer	levelid integer
1	7	4
2	6	4
3	7	3
4	6	1
5	5	3
6	5	1
7	4	4
8	4	3

Database: Classes_Powers

This table holds all the powers that each class has.

SQL Create Statement

Create table Classes_Powers(

ClassID int,

PowerID int,

Foreign Key (ClassID) References Classes(ClassID),

Foreign Key(PowerID) References Powers(PowerID));

Functional Dependency

ClassId→

PowerID→

Sample Data

	classid integer	powerid integer
1	1	1
2	1	2
3	1	3
4	1	4
5	2	1
6	2	8
7	2	4
8	3	5

Database: Enemies_Levels Table

This table contains all the levels that enemies will appear on

SQL Create Statement

Create table Enemies_Levels(

EnemyID int,

LevelID int,

Foreign Key (EnemyID) References Enemies(EnemyID),

Foreign Key(LevelID) References Levels(LevelID)

);

Functional Dependency

EnemyID→

Level→

Sample Data

	enemyid integer	levelid integer
1	9	4
2	8	4
3	7	2
4	6	3
5	5	4
6	4	3
7	3	1
8	2	4
9	1	1

Database: Kills_Game Table

This table is used to show how many kills, what enemies, and which game they occurred in.

SQL Create Statement

Create table Kills_Game(

GameID int,

EnemyID int,

Foreign Key (GameID) References Games(GameID),

Foreign Key(EnemyID) References Enemies(EnemyID)

);

Functional Dependency

GameID→

EnemyID→

Sample Data

	gameid integer	enemyid integer
1	7	9
2	6	8
3	5	7
4	4	6
5	3	5
6	2	4
7	1	3
8	2	3
9	4	5
10	5	7

Database: Levels_Map_ChunksTable

This table is used to show levels and the map chunks that make them up.

SQL Create Statement

Create table Levels_Map_Chunks(

LevelID int,

MapChunkID int,

Foreign Key(LevelID) References Levels(LevelID),

Foreign Key(MapchunkID) References MapChunk(MapChunkID)

);

Functional Dependency

LevelID→

MapChunkID→

Sample Data

	levelid integer	mapchunkid integer
1	1	1
2	1	3
3	1	5
4	2	2
5	2	3
6	2	4
7	2	5
8	3	1
9	3	2
10	3	3

Views: Kill count and scores per game

This views help the user access data that is normally not easily used.

SQL Create Statement

Drop View if exists KillsGame;

Create View KillsGame as(Select kg.GameID,g.score,Count(EnemyID) as KGcount

From Kills_game kg, games g

where kg.GameID=g.gameID

Group by kg.GameID,g.score);

SQL Query

select *

from killsgame

Order By GameID desc

Sample Results

	gameid integer	score integer	kgcount bigint
1	1	123242	1
2	2	123242422	2
3	3	1252522	1
4	4	633733	2
5	5	99999	2
6	6	6666666	1
7	7	0	1

Views: Character info

This View lets admins see each character's classes.

SQL Create Statement

```
Drop View if exists CharClass;

Create View CharClass as( SELECT distinct c.CharacterID,CharacterName, ClassName
                          From Classes CL, Characters C, Character_Classes CC
                          Where C.CharacterID=CC.CharacterID
                              and CL.ClassID=CC.ClassID
Order By CharacterName Desc );
```

SQL Query

```
Select *
 From CharClass
 Order by characterID asc
```

Sample Results

	characterid integer	charactername character varying(255)	classname character varying(255)
1	1	Jimbo	Bacon Mage
2	1	Jimbo	Innocent Bystander
3	2	Salami Sam	Mage
4	2	Salami Sam	Onion Knight
5	3	Bacon Barry	Bacon Mage
6	4	Cherry Canute	Innocent Bystander
7	4	Cherry Canute	Mage
8	4	Cherry Canute	Shadow
9	4	Cherry Canute	Vampire Archer
10	5	Shrimp Shrek	Vampire Archer

Views: Class info

This View lets admins see each character's classes.

SQL Create Statement

```
Drop View if Exists CharItem;

Create View CharItem as(
        Select C.CharacterID ,CharacterName, ItemName
        From Items I, Characters c, Character_Items CI
        WHERE I.ItemID=CI.ItemID
                and c.CharacterID=CI.CharacterID
        Order By Charactername desc          );
```

SQL Query

```
Select *
 From CharItem
 Order by characterID asc
```

Sample Results

	characterid integer	charactername character varying(255)	itemname character varying(255)
1	1	Jimbo	Demonic Longsword
2	1	Jimbo	Flabbos Leg of Meat
3	1	Jimbo	Book of the Dead
4	2	Salami Sam	Demonic Longsword
5	2	Salami Sam	Book of the Dead
6	3	Bacon Barry	BB gun
7	4	Cherry Canute	BB gun
8	5	Shrimp Shrek	Slingshot
9	6	Salad Sally	Demonic Longsword

Reports: Highscore board

This report illustrates how the highscore board will be populated.

SQL Query

Select P.Name, g.GameID, KillsGame.KgCount, score

From People p, Games g, People_Games PG, KillsGame

Where P.PeopleID=PG.PeopleID

and g.GameID=PG.GameID

and g.GameID=KillsGame.GameID

Order By G.GameID desc

Sample Results

	name character varying(255)	gameid integer	kgcount bigint	score integer
1	Ricardo	2	2	123242422
2	Richy	6	1	6666666
3	Richy Rich	3	1	1252522
4	Dick	4	2	633733
5	Richard	1	1	123242
6	Rich	5	2	99999
7	Richy	7	1	0

Reports: Character Classes and Items

This report uses two previously create views to show all of the character's classes and items.

SQL Query

SELECT distinct CharItem.CharacterName, ClassName, ItemName

From CharClass inner join CharItem on CharClass.CharacterID=CharItem.CharacterID

Where CharItem.CharacterID=CharClass.CharacterID

Order by CharItem.CharacterName desc

Sample Results

	charactername character varying(255)	classname character varying(255)	itemname character varying(255)
1	Shrimp Shrek	Vampire Archer	Slingshot
2	Salami Sam	Mage	Book of the Dead
3	Salami Sam	Mage	Demonic Longsword
4	Salami Sam	Onion Knight	Book of the Dead
5	Salami Sam	Onion Knight	Demonic Longsword
6	Jimbo	Bacon Mage	Book of the Dead
7	Jimbo	Bacon Mage	Demonic Longsword
8	Jimbo	Bacon Mage	Flabbos Leg of Meat
9	Jimbo	Innocent Bystander	Book of the Dead
10	Jimbo	Innocent Bystander	Demonic Longsword
11	Jimbo	Innocent Bystander	Flabbos Leg of Meat
12	Cherry Canute	Innocent Bystander	BB gun
13	Cherry Canute	Mage	BB gun
14	Cherry Canute	Shadow	BB gun
15	Cherry Canute	Vampire Archer	BB gun
16	Bacon Barry	Bacon Mage	BB gun

Reports: Admins

This report illustrates how the highscore board will be populated.

SQL Query

```
Select *
From People
Where PeopleID in(
        Select PeopleID
        From admins
        );
Order by PeopleID desc
```

Sample Results

	peopleid integer	name character varying(255)	email character varying(255)	joindate date
1	1	Richard	Richard@gmail.com	1010-10-10
2	2	Ricardo	Ricardo@gmail.com	2110-12-13

Stored Procedures

This sample stored procedure will execute every time an update is applied to Classes_Powers ensuring that no more than 7 powers are added to a single class.

SQL Query

```
CREATE OR REPLACE FUNCTION CountClassPowers()
returns trigger as $$
Declare countpwr integer =1;
BEGIN
loop
        IF ((select count(CP.powerID)From Classes_Powers CP Where ClassID= countpwr)>7)
        THEN
        Raise Exception 'You can not add more than 7 powers to a class';
          END if;
 countpwr=countpwr+1;
 exit when countpwr < (select Max(ClassID) From Classes_Powers);
 end loop;
 countpwr=0;
 return new;
END
 $$LANGUAGE plpgsql;
```

Triggers

This trigger will call CountClassPower() every time a power is added to a class.

SQL Query

```
drop trigger CountPowers on classes_Powers;

Create Trigger  CountPowers

Before Insert or Update

 On Classes_Powers

 FOR EACH ROW Execute Procedure CountClassPowers();
```

Security

Although the end user's interaction with the database is minimal and automatic in the future, security is still an issue. There are only two levels of security with the database: Admins and players. Admins have full access to the database while players can only view data in the database.

SQL Grants(Admin)

Create User admin With Password 'admin';

Revoke All on Items from admin;

Revoke All on admins from admin;

Revoke All on People from admin;

Revoke All on MapChunk from admin;

Revoke All on Character_Items from admin;

Revoke All on People_Games from admin;

Revoke All on Levels_Map_Chunks from admin;

Revoke All on Characters from admin;

Revoke All on Character_Games from admin;

Revoke All on Games from admin;

Revoke All on Games_Levels from admin;

Revoke All on Levels from admin;

Revoke All on Character_Classes from admin;

Revoke All on Character_Games from admin;

Revoke All on Classes from admin;

```
Revoke All on Kills_Game from admin;
Revoke All on Enemies from admin;
Revoke All on Classes_Powers from admin;
Revoke All on ESkills_Enemies from admin;
Revoke All on Powers from admin;
Revoke All on EnemySkills from admin;

Grant insert, update, delete, select on Items to admin;
Grant insert, update, delete, select on admins to admin;
Grant insert, update, delete, select on People to admin;
Grant insert, update, delete, select on MapChunk to admin;
Grant insert, update, delete, select on Character_Items to admin;
Grant insert, update, delete, select on People_Games to admin;
Grant insert, update, delete, select on Levels_Map_Chunks to admin;
Grant insert, update, delete, select on Characters to admin;
Grant insert, update, delete, select on Character_Games to admin;
Grant insert, update, delete, select on Games to admin;
Grant insert, update, delete, select on Games_Levels to admin;
Grant insert, update, delete, select on Levels to admin;
Grant insert, update, delete, select on Character_Classes  to admin;
Grant insert, update, delete, select on Character_Games  to admin;
```

Grant insert, update, delete, select on Classes to admin;

Grant insert, update, delete, select on Kills_Game to admin;

Grant insert, update, delete, select on Enemies to admin;

Grant insert, update, delete, select on Classes_Powers to admin;

Grant insert, update, delete, select on ESkills_Enemies to admin;

Grant insert, update, delete, select on Powers to admin;

Grant insert, update, delete, select on EnemySkills to admin;

SQL Grants(Players)

Create User Player With Password 'pass';

Revoke All on Items from player;

Revoke All on admins from player;

Revoke All on People from player;

Revoke All on MapChunk from player;

Revoke All on Character_Items from player;

Revoke All on People_Games from player;

Revoke All on Levels_Map_Chunks from player;

Revoke All on Characters from player;

Revoke All on Character_Games from player;

Revoke All on Games from player;

Revoke All on Games_Levels from player;

Revoke All on Levels from player;

```
Revoke All on Character_Classes  from player;
Revoke All on Character_Games  from player;
Revoke All on Classes from player;
Revoke All on Kills_Game from player;
Revoke All on Enemies from player;
Revoke All on Classes_Powers from player;
Revoke All on ESkills_Enemies from player;
Revoke All on Powers from player;
Revoke All on EnemySkills from player;

Grant select on Items to player;
Grant select on admins to player;
Grant select on Characters to player;
Grant select on Games to player;
Grant select on Levels to player;
Grant select on Enemies to player;
Grant select on Powers to player;
Grant select on EnemySkills to player;
```

Known Problems

- Map Chunks can be assigned to levels even if their ratings do not match
- Currently no way to know what Items will drop from enemies
- Classes_powers and eskills_enemy will allow duplicate powers.

Future Improvements

- Randomize map chunks to levels
- Randomize Character names based on class.
- Randomize items that enemies have/drop
- Show which level kills occurred on.