Database Design Project: Playstation All-Stars Battle Royale

By: Sean Huban

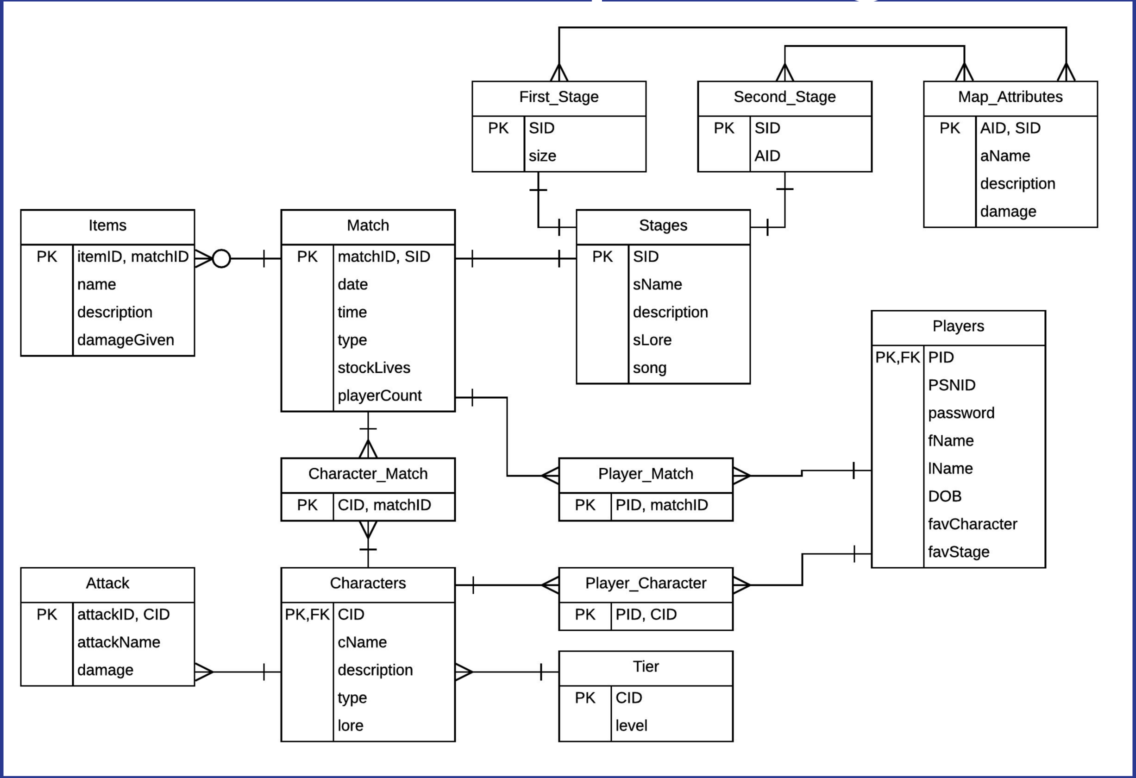
Table of Contents

Executive Summary

PlayStation All-Stars Battle Royale was released for the PlayStation 3 and PlayStation Vita in late 2012. The game took iconic characters from across all of Sony's exclusive franchises and pit them against one another in a platform-brawler style game. Characters fight on maps created as hybrids between two worlds and use items from across a wide variety of games. Players try to charge up their 'Ultimate' meters in order to execute devastating attacks. It is often compared to one of gaming's most notorious fighting games, *Super Smash Bros.* On a national scale, Sony's attempt to create a cult classic failed. Personally I consider P.A.B.R. to be one of the best fighting games ever created.

This database design proposal has been created to show all of the relationships between all of the data contained within the game and to show how they could be represented in a 3NF form. Example data includes things such as your character, map, items, special ability, stage, match, match type, etc. This proposal contains an E/R diagram showing relationship between tables. It also contains views, stored procedures, triggers, and reports. The database is loaded with a set amount of sample data so that these queries and other database modifiers I have drafted will return a set of data.

Entity/Relationship Diagram



Create Table Statements

Attack Table:
Every character in the game has 3 different attacks that are different in all 4 directions. The Attack table stores every move as a unique ID as well has other pertinent information such as damage and the name of the attack

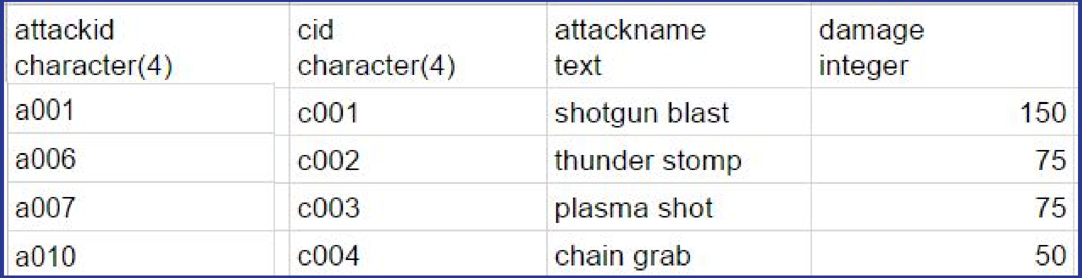
Functional Dependencies:
attackID , cID -> attackName, damage

Create Table Statements

Attack Table (cont.):

```
CREATE TABLE Attack (
 attackID CHAR(4) UNIQUE NOT NULL,
 cID CHAR(4) NOT NULL REFERENCES Characters(cID),
 attackName TEXT NOT NULL,
 damage INTEGER,
 CHECK(damage > 0),
 Primary key(attackID, cID)
);
```

Sample Data:



attackid character(4)	cid character(4)	attackname text	damage integer
a001	c001	shotgun blast	150
a006	c002	thunder stomp	75
a007	c003	plasma shot	75
a010	c004	chain grab	50

Create Table Statements

Characters Table:
This table contains data pertaining to each and every character contained within the game itself.

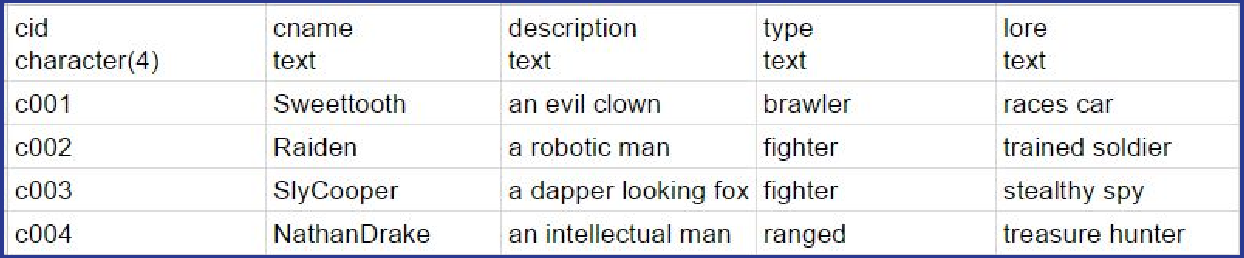
Functional Dependencies:
cID -> cName, description, type, lore

Create Table Statements

Characters Table (cont.):

```
CREATE TABLE Characters (
 cID CHAR(4) UNIQUE NOT NULL,
 cName TEXT NOT NULL,
 description TEXT,
 type TEXT NOT NULL,
 lore TEXT NOT NULL,
 PRIMARY KEY(cID)
);
```

Sample Data:



cid character(4)	cname text	description text	type text	lore text
c001	Sweettooth	an evil clown	brawler	races car
c002	Raiden	a robotic man	fighter	trained soldier
c003	SlyCooper	a dapper looking fox	fighter	stealthy spy
c004	NathanDrake	an intellectual man	ranged	treasure hunter

Create Table Statements

First-Stage Table:
PASBR uses 2 stages per match to create a transitional experience mid-game. This table keeps track of the stageID and the AttributeID, denoted as ‘AID’. This table tracks information for the first map of the match.

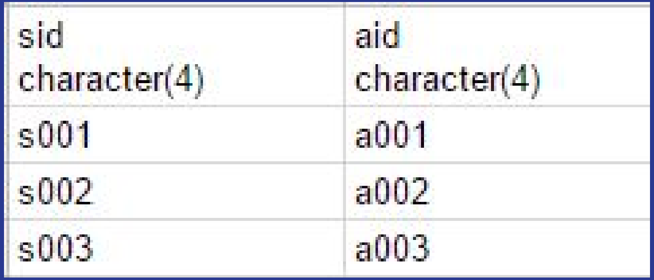
Functional Dependencies:
aID -> sID

Create Table Statements

First_Stage Table (cont.):

```
Create table First_Stage (
 sID CHAR(4) NOT NULL REFERENCES Stages(sID),
 aID CHAR(4) NOT NULL REFERENCES Map_Attributes(aID),
 PRIMARY KEY(sID)
);
```

Sample Data:



sid character(4)	aid character(4)
s001	a001
s002	a002
s003	a003

Create Table Statements

Second_Stage Table:
PASBR uses 2 stages per match to create a transitional experience mid-game. This table keeps track of the stageID and the AttributeID, denoted as ‘AID’. This table keeps track of the second map

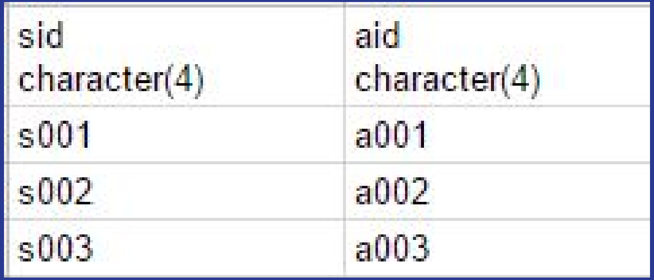
Functional Dependencies
aID -> sID

Create Table Statements

Second_Stage Table (cont.):

```
Create table Second_Stage (
 sID CHAR(4) NOT NULL REFERENCES Stages(sID),
 aID CHAR(4) NOT NULL REFERENCES Map_Attributes(aID),
 PRIMARY KEY(sID)
);
```

Sample Data:



sid character(4)	aid character(4)
s001	a001
s002	a002
s003	a003

Create Table Statements

Map_Attributes Table:
This table keeps track of the details for each of the map attributes which affect things like gravity and hazard damages on the map.

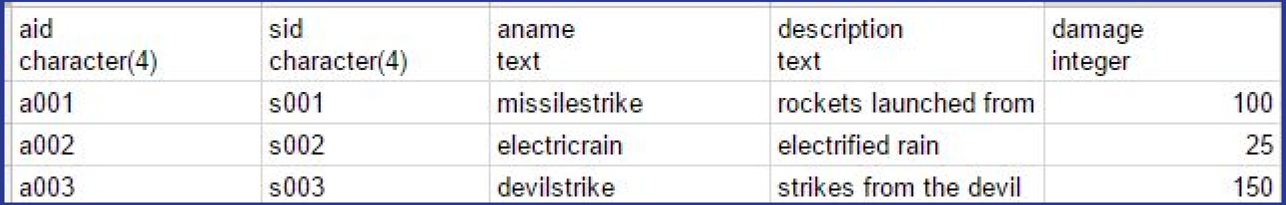
Functional Dependencies:
aID , sID -> aName , description , damage

Create Table Statements

Map_Attributes Table (cont.):

```
CREATE TABLE Map_Attributes (
 aID CHAR(4) UNIQUE NOT NULL,
 sID CHAR(4) NOT NULL REFERENCES Stages(sID),
 aName TEXT,
 description TEXT,
 damage INTEGER,
 CHECK (damage > 0),
 PRIMARY KEY(aID)
);
```

Sample Data:

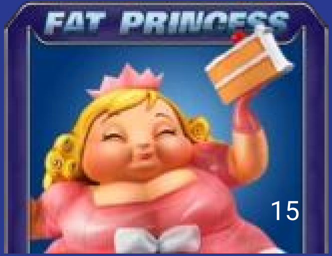


aid character(4)	sid character(4)	aname text	description text	damage integer
a001	s001	missilestrike	rockets launched from	100
a002	s002	electricrain	electrified rain	25
a003	s003	devilstrike	strikes from the devil	150

Create Table Statements

Tier Table:
When looking at a game at a professional level, the characters are sorted into various tiers separated by skill level and overall perceived advantage in-game. The order of tiers, from highest to lowest, is: SS,S,A,B,C,D, and F.

Functional Dependencies:
tID , level -> cID

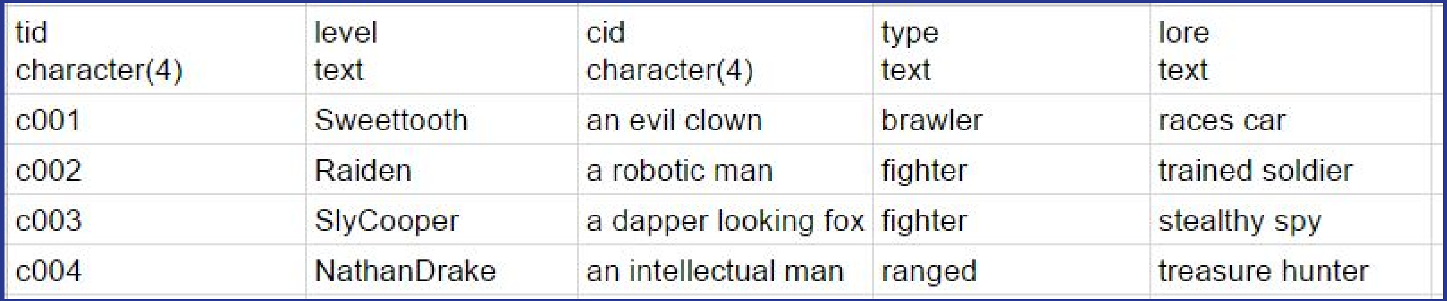


Create Table Statements

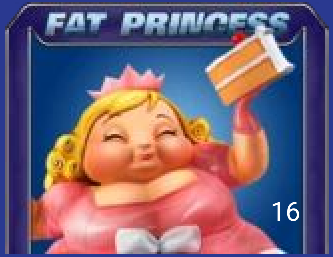
Tier Table (Cont.):

```
CREATE TABLE Tier (
 tID CHAR(4) NOT NULL,
 level TEXT NOT NULL,
 cID CHAR(4) NOT NULL REFERENCES Characters(cID),
 PRIMARY KEY(tID, cID)
);
```

Sample Data:



tid character(4)	level text	cid character(4)	type text	lore text
c001	Sweettooth	an evil clown	brawler	races car
c002	Raiden	a robotic man	fighter	trained soldier
c003	SlyCooper	a dapper looking fox	fighter	stealthy spy
c004	NathanDrake	an intellectual man	ranged	treasure hunter



Create Table Statements

Player_Character Table:
This table serves to easily associate a player with the character that they have chosen to play as.

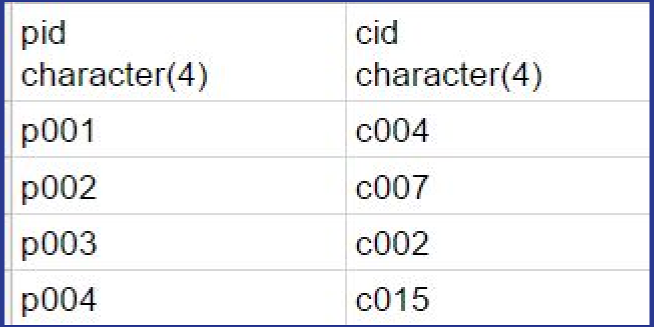
Functional Dependencies:
N/A

Create Table Statements

Player_Character Table (Cont.):

```
CREATE TABLE Player_Character (
 pID CHAR(4) NOT NULL REFERENCES Players(pID),
 cID CHAR(4) NOT NULL REFERENCES Characters(cID),
 PRIMARY KEY(pID, cID)
);
```

Sample Data:



pid character(4)	cid character(4)
p001	c004
p002	c007
p003	c002
p004	c015

Create Table Statements

Players Table:
This table keeps track of all data pertaining to the user playing the game. Every player is uniquely identified with a player ID (aka, pID)

Functional Dependencies:
pID -> psnID, password, fName, lName, DOB, favCharacter, favStage

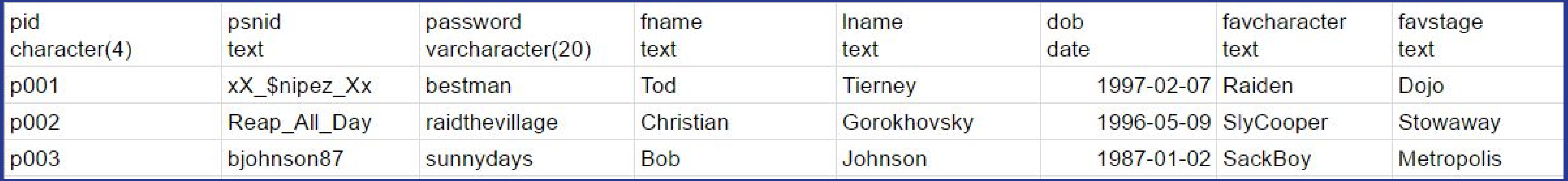
Create Table Statements

Players Table (cont.):

```
CREATE TABLE Players (
 pID CHAR(4) UNIQUE NOT NULL,
 psnID TEXT NOT NULL,
 password VARCHAR(20) NOT NULL,
 fName TEXT,
 lName TEXT,
 DOB DATE NOT NULL,
 favCharacter TEXT,
 favStage TEXT,
 PRIMARY KEY (pID)
);
```

Create Table Statements

Players Table (cont.):
Sample Data:



pid character(4)	psnid text	password varcharacter(20)	fname text	lname text	dob date	favcharacter text	favstage text
p001	xX_$nipez_Xx	bestman	Tod	Tierney	1997-02-07	Raiden	Dojo
p002	Reap_All_Day	raidthevillage	Christian	Gorokhovsky	1996-05-09	SlyCooper	Stowaway
p003	bjohnson87	sunnydays	Bob	Johnson	1987-01-02	SackBoy	Metropolis

Create Table Statements

Player_Match Table:
This table is used to show the relationship between players and the match that they competed in.

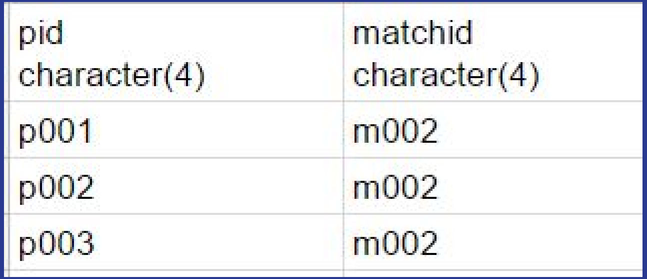
Functional Dependencies:
N/A

Create Table Statements

Player_Match Table (cont.):

```
CREATE TABLE Player_Match (
 pID CHAR(4) NOT NULL REFERENCES Players(pID),
 matchID CHAR(4) NOT NULL REFERENCES Match(matchID),
 PRIMARY KEY(pID, matchID)
);
```

Sample Data:



pid character(4)	matchid character(4)
p001	m002
p002	m002
p003	m002

Create Table Statements

Character_Match Table:
This table illustrates the relationship between Characters and the most recent match of the game that they have been involved in.

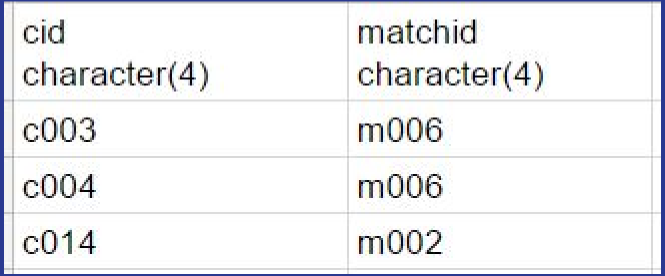
Functional Dependencies:
N/A

Create Table Statements

Character_Match Table (cont.):

```
CREATE TABLE Character_Match (
 cID CHAR(4) NOT NULL REFERENCES Characters(cID),
 matchID CHAR(4) NOT NULL REFERENCES Match(matchID),
 PRIMARY KEY(cID, matchID)
);
```

Sample Data:



cid character(4)	matchid character(4)
c003	m006
c004	m006
c014	m002

Create Table Statements

Match Table:
This table stores all of the important information pertaining to the specifics an in-game match.

Functional Dependencies:
matchID, sID -> date, time, type, stockLives, playerCount

Create Table Statements

Match Table (cont.):

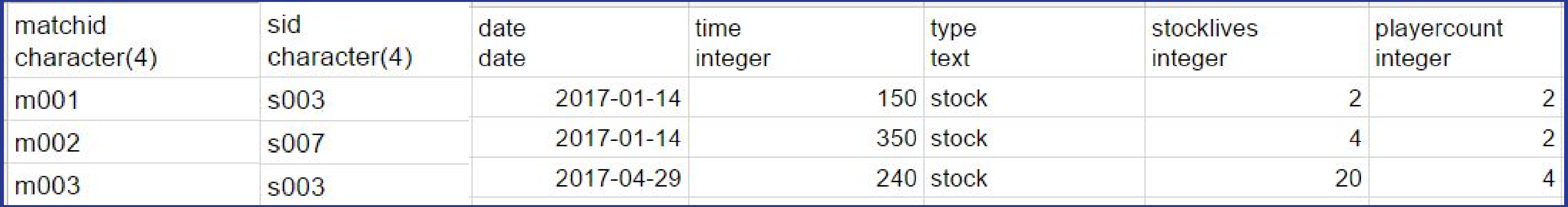
```
CREATE TABLE Match (
 matchID CHAR(4) UNIQUE NOT NULL,
 date DATE NOT NULL,
 time INTEGER,
 type TEXT NOT NULL,
 stockLives INTEGER,
 playerCount INTEGER NOT NULL,
 sID CHAR(4) UNIQUE NOT NULL REFERENCES Stages(sID),
 CHECK (time > 0),
 CHECK(stockLives > 0),
 CHECK(playerCount >= 1),
 PRIMARY KEY(matchID,sID)
);
```

Sample Data:

Create Table Statements

Match Table (cont.):

Sample Data:



matchid character(4)	sid character(4)	date date	time integer	type text	stocklives integer	playercount integer
m001	s003	2017-01-14	150	stock	2	2
m002	s007	2017-01-14	350	stock	4	2
m003	s003	2017-04-29	240	stock	20	4

Create Table Statements

Items Table:
This table contains all of the information pertaining to the ingame items including its description, damage amount, and unique item ID.

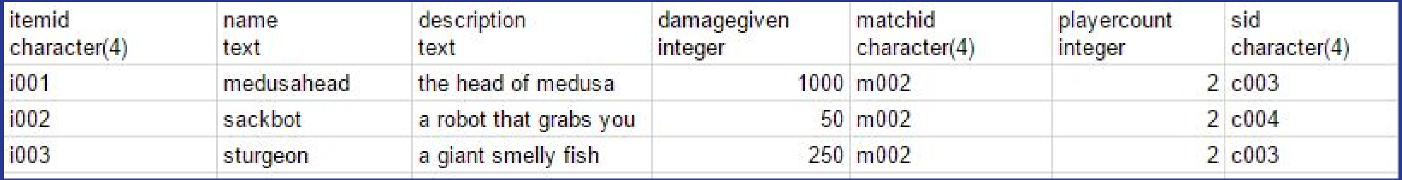
Functional Dependencies:
itemID , MatchID -> name , description, damageGiven

Create Table Statements

Items Table (cont.):

```
CREATE TABLE Items (
 itemID CHAR(4) UNIQUE NOT NULL,
 name TEXT NOT NULL,
 description TEXT,
 damageGiven INTEGER NOT NULL,
 matchID CHAR(4) NOT NULL REFERENCES Match(matchID),
 CHECK (damageGiven > 0),
 Primary key(itemID, matchID)
);
```

Sample Data:



itemid	name	description	damagegiven	matchid	playercount	sid
character(4)	text	text	integer	character(4)	integer	character(4)
i001	medusahead	the head of medusa	1000	m002	2	c003
i002	sackbot	a robot that grabs you	50	m002	2	c004
i003	sturgeon	a giant smelly fish	250	m002	2	c003

Create Table Statements

Stages Table:
This table represents the information pertaining to each of the separate stages in the game that the characters can choose to battle on.

Functional Dependencies:

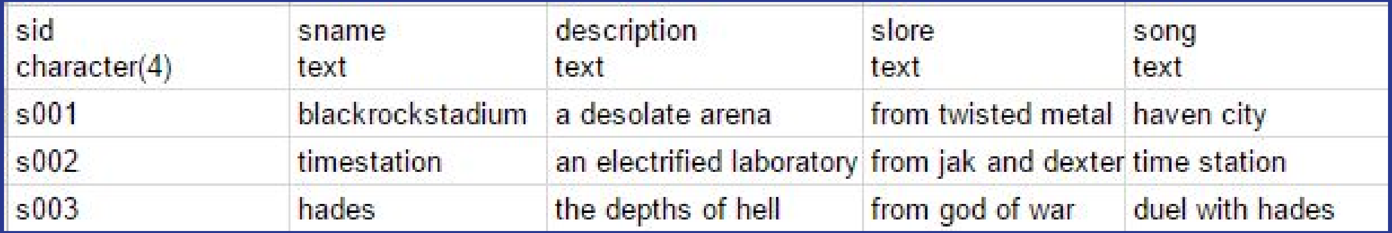
sID -> sName , description , sLore , song

Create Table Statements

Stages Table (cont.):

```
CREATE TABLE Stages (
 sID CHAR(4) UNIQUE NOT NULL,
 sName TEXT NOT NULL,
 description TEXT,
 sLore TEXT,
 song TEXT NOT NULL,
 PRIMARY KEY(sID)
);
```

Sample Data:



sid character(4)	sname text	description text	slore text	song text
s001	blackrockstadium	a desolate arena	from twisted metal	haven city
s002	timestation	an electrified laboratory	from jak and dexter	time station
s003	hades	the depths of hell	from god of war	duel with hades

View Definitions

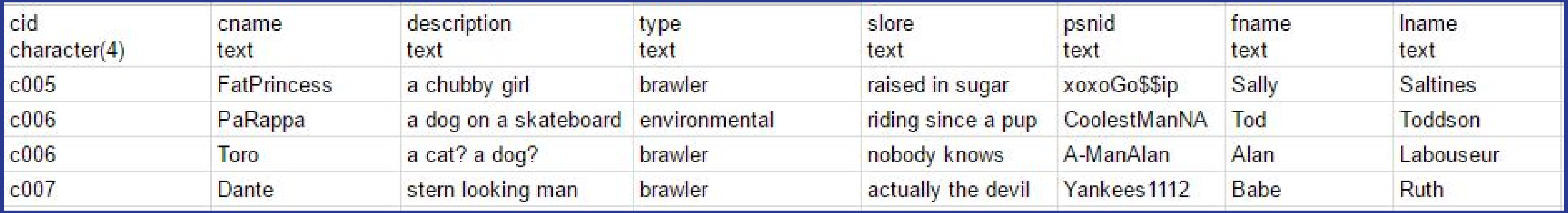
CharacterPlayer:
This view was created in order to easily see what character each player (psnID) has chosen for the match

```
CREATE view CharacterPlayer AS
SELECT Characters.*, psnID, fName, favCharacter
FROM Players, Characters, Player_Character
WHERE Players.pID = Player_Character.pID
AND Characters.cID = Player_Character.cID
;
```

View Definitions

CharacterPlayer (cont.):

Sample Data:



cid	cname	description	type	slore	psnid	fname	lname
character(4)	text	text	text	text	text	text	text
c005	FatPrincess	a chubby girl	brawler	raised in sugar	xoxoGo$$ip	Sally	Saltines
c006	PaRappa	a dog on a skateboard	environmental	riding since a pup	CoolestManNA	Tod	Toddson
c006	Toro	a cat? a dog?	brawler	nobody knows	A-ManAlan	Alan	Labouseur
c007	Dante	stern looking man	brawler	actually the devil	Yankees1112	Babe	Ruth

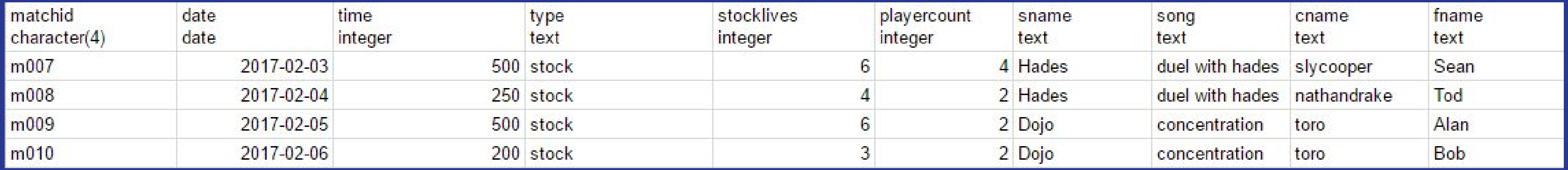
View Definitions

MatchView:
This view gives you details about recent matches that have occurred in game.

```
CREATE view matchView AS
SELECT m.matchID, m.date, m.time, m.type, m.stockLives,m.playerCount,
s.sid, s.sName, s.description, s.sLore, c.cName, p.lName
    FROM match m, Stages s, Characters c, Character_Match cm,Players p,
        Player_Match pm
WHERE m.sID = s.sID
    AND c.cID = cm.cID
    AND cm.matchID = m.matchID
    AND p.pID = pm.pID
    AND pm.matchID = m.matchID
;
```

View Definitions

MatchView(cont.):
Sample Data:



matchid character(4)	date date	time integer	type text	stocklives integer	playercount integer	sname text	song text	cname text	fname text
m007	2017-02-03	500	stock	6	4	Hades	duel with hades	slycooper	Sean
m008	2017-02-04	250	stock	4	2	Hades	duel with hades	nathandrake	Tod
m009	2017-02-05	500	stock	6	2	Dojo	concentration	toro	Alan
m010	2017-02-06	200	stock	3	2	Dojo	concentration	toro	Bob

View Definitions

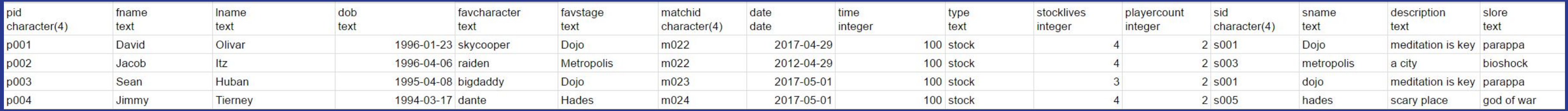
Database:

This view does a good job of presenting the user with almost all of the information contained within the database.

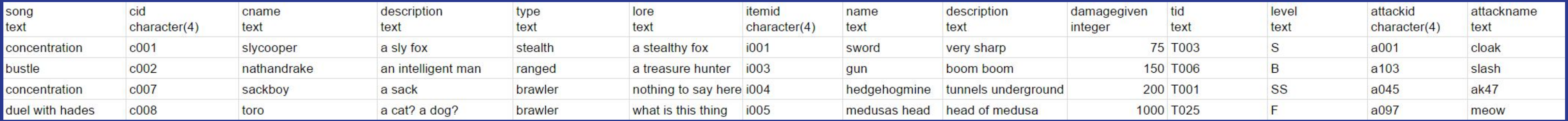
```
CREATE view dbView AS
SELECT p.pID, p.fName, p.lName, p.dob, p.favCharacter,p.favStage,
     m.matchID, m.date, m.time, m.type, m.stockLives, m.playerCount, s.*, c.*,
     i.itemID, i.Name, i.Description, i.damageGiven,t.tID, t.level,
     a.sID, a.attackName, a.damage
FROM Players p, Match m, Player_Match pm, Stages s, Characters
     c, Player_Character pc, Character_Match cm,Items i, Tier t, Attack a
WHERE p.pID = pm.pID
     AND pm.matchID = m.matchID
     AND m.sID = s.sID
     AND c.cID = cm.cID
     AND p.pID = pc.pID
     AND c.cID = pc.cID
     AND c.cID = cm.cID
     AND i.matchID = m.matchID
     AND t.cID = c.cID
     AND a.cID = c.cID
;
```

View Definitions

Database(cont.):
Sample data:



pid character(4)	fname text	lname text	dob text	favcharacter text	favstage text	matchid character(4)	date date	time integer	type text	stocklives integer	playercount integer	sid character(4)	sname text	description text	slore text
p001	David	Olivar	1996-01-23	skycooper	Dojo	m022	2017-04-29	100	stock	4	2	s001	Dojo	meditation is key	parappa
p002	Jacob	Itz	1996-04-06	raiden	Metropolis	m022	2012-04-29	100	stock	4	2	s003	metropolis	a city	bioshock
p003	Sean	Huban	1995-04-08	bigdaddy	Dojo	m023	2017-05-01	100	stock	3	2	s001	dojo	meditation is key	parappa
p004	Jimmy	Tierney	1994-03-17	dante	Hades	m024	2017-05-01	100	stock	4	2	s005	hades	scary place	god of war



song text	cid character(4)	cname text	description text	type text	lore text	itemid character(4)	name text	description text	damagegiven integer	tid text	level text	attackid character(4)	attackname text
concentration	c001	slycooper	a sly fox	stealth	a stealthy fox	i001	sword	very sharp	75	T003	S	a001	cloak
bustle	c002	nathandrake	an intelligent man	ranged	a treasure hunter	i003	gun	boom boom	150	T006	B	a103	slash
concentration	c007	sackboy	a sack	brawler	nothing to say here	i004	hedgehogmine	tunnels underground	200	T001	SS	a045	ak47
duel with hades	c008	toro	a cat? a dog?	brawler	what is this thing	i005	medusas head	head of medusa	1000	T025	F	a097	meow

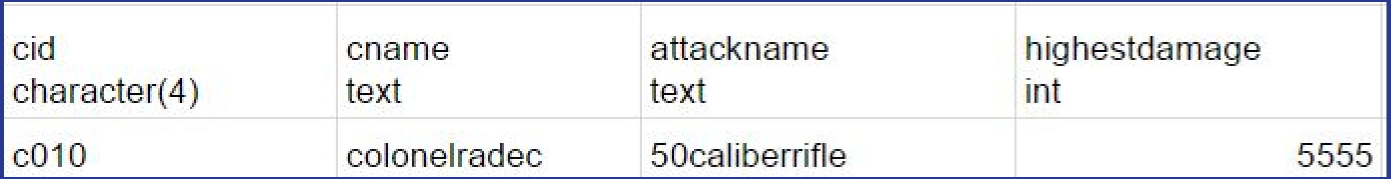
Queries and Reports

Strongest Attack:
This query returns the data for the character with the most damaging attack.

```
SELECT c.cID, c.cName, a.attackName, a.damage
    AS highestDamage
FROM Characters c, Attack a
    WHERE c.cid = a.cID
ORDER BY a.damage DESC
limit 1
;
```

Queries and Reports

Strongest Attack(cont.):
Sample Data:



cid character(4)	cname text	attackname text	highestdamage int
c010	colonelradec	50caliberrifle	5555

Queries and Reports

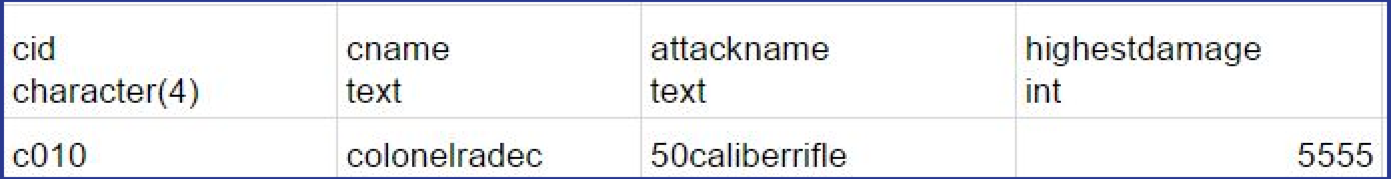
WhoAttack:
This query, slightly connected to the last pulls the results for the player and the character they used to execute the strongest attack.

```
SELECT p.pid, p.firstName, c.cid, c.cName, a.attackName,
a.damage
    FROM Characters c, Players p, Player_Character pc, Attack a
WHERE p.pID = pc.pID
    AND pc.cID = c.cID
    AND c.cID = mo.cID
ORDER BY a.damage DESC
limit 1
;
```

Queries and Reports

WhoAttack(cont.):

Sample Data:



cid character(4)	cname text	attackname text	highestdamage int
c010	colonelradec	50caliberrifle	5555

Stored Procedures

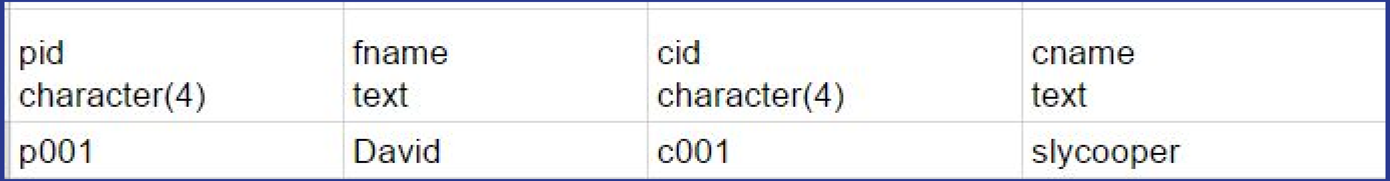
Player-Character/Stage: This procedure returns the players and characters on a chosen stage

```
CREATE OR REPLACE function playerCharacterStage(text,REFCURSOR)
     RETURNS REFCURSOR AS
$$
DECLARE
 stage text := $1;
 resultSet REFCURSOR := $2;
BEGIN
 OPEN resultSet FOR
 select p.pID, p.psnID, c.cID, c.cName
from Players p, Characters c, Player_Match pm,Character_Match cm, Match m, Stages s
     where p.pID = pm.pID
     and pm.matchID = m.matchID
     and c.cID = cm.cID
     and cm.matchID = m.matchID
     and m.sID = s.sID
     and s.sName = stage;
 RETURN resultSet;
END;
$$
language plpgsql;
```

Stored Procedures

Player/Character-Stage(cont.):

Sample Data:



pid character(4)	fname text	cid character(4)	cname text
p001	David	c001	slycooper

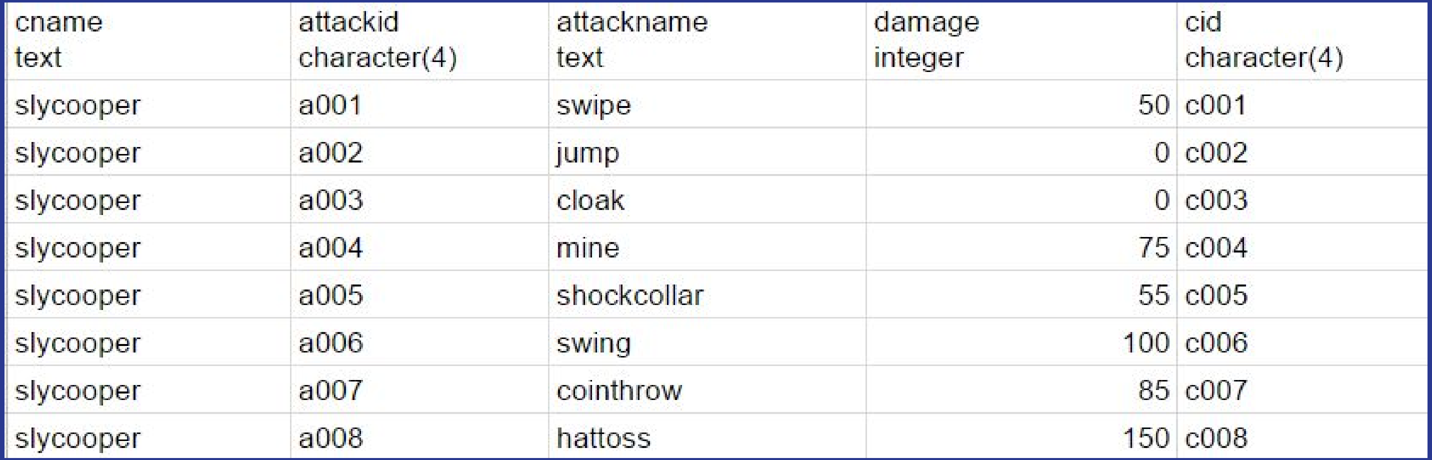
Stored Procedures

Character-Attacks: **This will return all of the moves for a chosen character**

```
CREATE OR REPLACE function characterAttacks(text, REFCURSOR)
    RETURN REFCURSOR as
$$
DECLARE
    character text := $1;
    resultSet REFCURSOR := $2;
BEGIN
 OPEN resultSet FOR
 SELECT c.cName, a.attackID, a.attackName, a.damage, c.cid
    FROM Attack a, Characters c
    WHERE c.cID = a.ciID
        AND c.cName = character;
RETURN resultSet;
END;
$$
language plpgsql;
```

Stored Procedures

Character-Attacks:
Sample Data:



cname	attackid	attackname	damage	cid
text	character(4)	text	integer	character(4)
slycooper	a001	swipe	50	c001
slycooper	a002	jump	0	c002
slycooper	a003	cloak	0	c003
slycooper	a004	mine	75	c004
slycooper	a005	shockcollar	55	c005
slycooper	a006	swing	100	c006
slycooper	a007	cointhrow	85	c007
slycooper	a008	hattoss	150	c008

Triggers

addPlayer: This trigger is activated when an attempt to add a new player is started and NULL data is detected.

```
CREATE OR REPLACE FUNCTION addPlayer() RETURNS trigger AS
$$
BEGIN
IF NEW.pID is null THEN
      raise exception 'Invalid pid';
 END IF;
IF NEW.psnID IS NULL THEN
      raise exception 'Invalid PlayStation Network ID';
 END IF;
IF NEW.password IS NUL THEN
      raise exception 'Invalid password';
 END IF;
INSER INTO Players(pID, psdID, password, fName,lName, dob, favCharacter, favStage)
values (NEW.pID, NEW.psnID, NEW.password,NEW.fName, NEW.lName, NEW.dob, NEW.favCharacter,
NEW.favStage);
RETURN new;
END;
$$ language plpgsql;
CREATE trigger addPlayer
AFTER INSERT ON Players
FOR EACH ROW EXECUTE procedure addPlayer();
```

Security

For this database there are only 2 simple levels of security. The first level being a database admin who is able to view, add, edit, and delete all information in the database. The second level is the user(player) level. The user is only able to view the database, and is not allowed to make any changes.

Admin

```
CREATE role admin
GRANT SELECT, INSERT, UPDATE, DELETE
ON all tables IN schema PUBLIC
to admin
```

Player

```
CREATE role player
GRANT SELECT
on Players, Stages, first_Stage, Second_Stage, Map_Attributes,
Match, Characters, Attack, Tier, Items
to player
```

Implementation Notes, Known Problems, Future Enhancements

The implementation of this database was completed without any major roadblocks. This database can be used to take data and turn it into useful statistical information for this game. Using the information in this database could help players learn about their playstyle as well as their preferences in-game.

There are not any known issues that I noted with this database that prevent it from operating properly. There is however a lot of room for future expansion. Video games can be very complex and deal with tons of data at a time. This database could be expanded in numerous ways in order to provide users with a much more in-depth look at all of the information that is contained within the game. The Map_Attributes table could be updated to contain a lot more detail about the specifics of the map. The multiple sets of data that deal with damage could be updated to be more accurate; not all of the damage that is caused in game is delivered in one instance, some attacks do damage over time.