Super Smash Brothers

A Database Design Proposal
By:

Frank Siderio

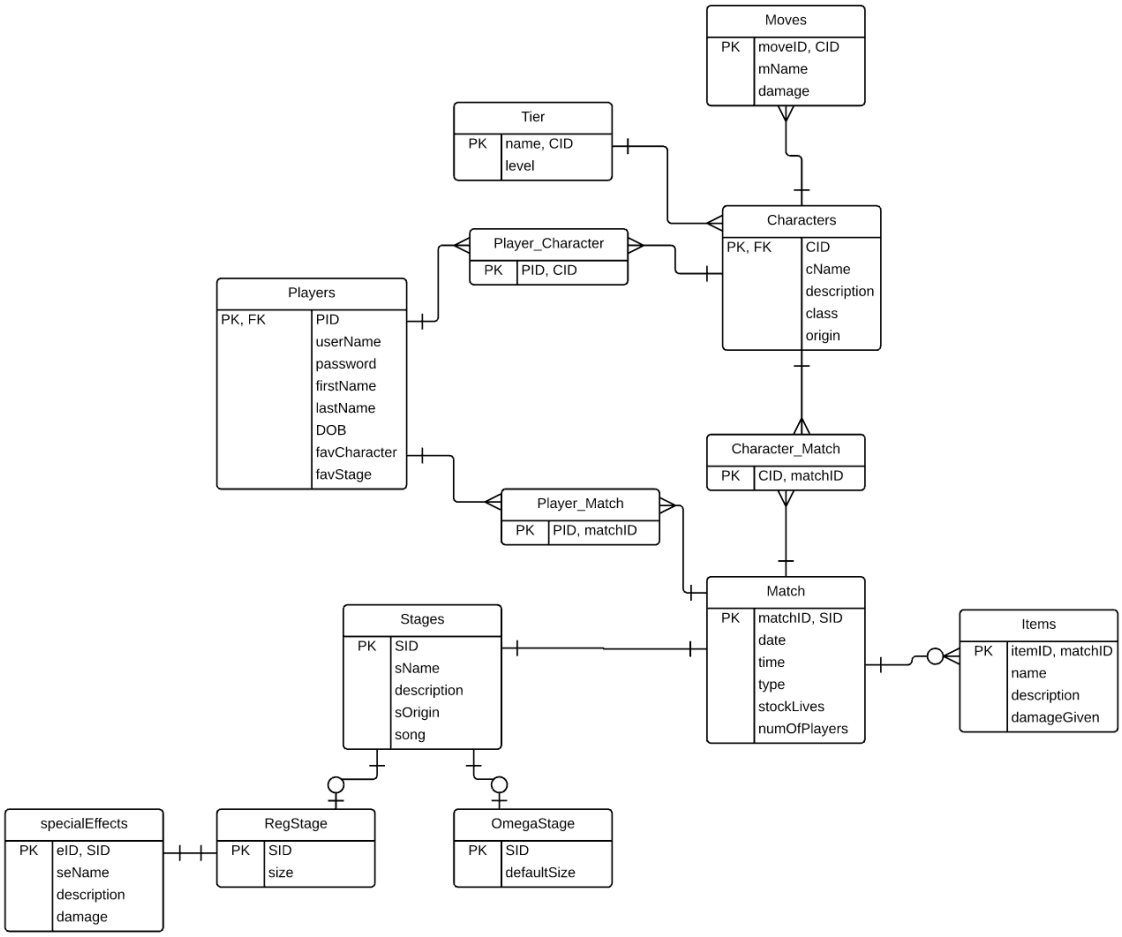
Table of Contents

Executive Summary

This document shows the design and implementation of the game, Super Smash Brothers. This is a way to see all the different relationships throughout the game. The document will give an ER diagram along with create statements for all the different tables. It will also show an implementation of views, store procedures, reports, and triggers.

The purpose of this database is the show the different relationships between all the aspects of the game. This could be useful for generating reports and statistics about the game. The main goal of this document is to design and implement a third normal form database (or even better codd normal form).

Entity-Relationship Diagram



Create Table Statements

Players Table:
The players table represents the users who are playing the game. Each player has their own player id (pid)

```
--Players Table--
CREATE TABLE Players (
    pid             CHAR(4) UNIQUE NOT NULL,
    userName        TEXT    NOT NULL,
    password        VARCHAR(20)    NOT NULL,
    firstName       TEXT,
    lastName        TEXT,
    dob             DATE    NOT NULL,
    favCharacter    TEXT,
    favStage        TEXT,
  PRIMARY KEY (pid)
);
```

Functional Dependencies
pid —> userName, password, firstName, lastName, dob, favCharacter, favStage

Sample Data

Output pane

Data Output | Explain | Messages | History

	pid character(4)	username text	password character varying(20)	firstname text	lastname text	dob date	favcharacter text	favstage text
1	p000	FrankBond007	IamAwesome	Frank	Siderio	1996-02-19	Pikachu	Final Destination
2	p001	baseballDude	tgife@34567	John	Doe	1996-07-09	Bowser	Final Destination
3	p002	coolDude5	IamCool	Josh	Smith	1995-12-11	Link	Battlefied
4	p003	jamesGriff	IamAnicePerson	James	Griffin	1995-06-09	Marth	Dream Land
5	p004	Alan007	alpaca	Alan	Labouseur	1986-01-03	Kirby	Battlefield

Characters Table:
Represents all the different characters in the game.

```
--Characters Table--
CREATE TABLE Characters (
    cid             CHAR(4) UNIQUE NOT NULL,
    name            TEXT NOT NULL,
    description     TEXT,
    class           TEXT NOT NULL,
    origin          TEXT NOT NULL,
  PRIMARY KEY(cid)
);
```

Functional Dependencies
cid —> name, description, class, origin

Sample Data

	cid character(4)	cname text	cdescription text	class text	corigin text
1	c000	Pikachu	Down B	lightweight	pokemon
2	c001	Marth	Uses a sword	middleweight	fire emblem
3	c002	Bowser	Is a big dude	heavyweight	Mario
4	c003	Link	Has a lot of items	middleweight	Zelda
5	c004	Ness	swings a bat	lightweight	Earth Bound

Player_Character Table:
Since many players can be many different characters we need an associative table between the Players and Character table.

```
--Player_Character Table--
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 Output

	pid character(4)	cid character(4)
1	p000	c000
2	p001	c002
3	p000	c002
4	p002	c003
5	p003	c001
6	p004	c004

Match Table:
Stores all the information about each match played.

```
--Match Table--
Create table Match (
    matchID          CHAR(4) UNIQUE NOT NULL,
    DATE               DATE              NOT NULL,
    time               INTEGER,
    type               TEXT              NOT NULL,
    stockLives         INTEGER,
    numOfPlayers  INTEGER          NOT NULL,
    sid                CHAR(4) UNIQUE NOT NULL REFERENCES
Stages(sid),
    CHECK (time > 0),
    CHECK(stockLives > 0),
    CHECK(numOfPlayers >= 2),
  PRIMARY KEY(matchID,sid)
);
```

Functional dependencies
matchID,sid —> date, time, type, stockLives, numOfPlayers

Sample Output

	matchid character(4)	date date	time integer	type text	stocklives integer	numofplayers integer	sid character(4)
1	m000	2016-01-02	100	Stock	2	2	s000
2	m001	2016-01-02	200	Stock	2	2	s001
3	m002	2016-01-02	300	Stock	2	2	s002
4	m003	2016-01-02	400	Stock	2	2	s003
5	m004	2016-01-09	340	Stock	6	8	s004

Player_Match Table:
The associative table between the Players and Match table. This table is necessary because there is a many to many relationship between the Players and Match tables.

```
--Player_Match Table--
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 Output

	pid character(4)	matchid character(4)
1	p000	m000
2	p001	m001
3	p002	m002
4	p003	m003
5	p004	m004

Character_Match Table:
The associative table between the Character and Match tables. Many characters can play many matches.

```
--Character_Match Table--
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 Output

	cid character(4)	matchid character(4)
1	c000	m000
2	c001	m001
3	c002	m002
4	c003	m003
5	c004	m004

Tier Table:
In the world of professional smash there are tiers to represent how good a character is. There are 3 tiers that each have their own level.

```
--Tier Table--
CREATE TABLE Tier (
    name   TEXT NOT NULL,
    level     TEXT NOT NULL,
    cid       CHAR(4)  NOT NULL REFERENCES Characters(cid),
  PRIMARY KEY(name, cid)
);
```

Functional Dependencies
name,level —> cid

Sample Output

	tname text	level text	cid character(4)
1	Tier 1	B	c000
2	Tier 2	A	c001
3	Tier 3	A	c002
4	Tier 1	A	c003
5	Tier 1	B+	c004

Moves Table:
Each character has their own set of moves. This table stores each move and what character the move belongs to along with some more information about the move.

```
--Moves Table--
CREATE TABLE Moves (
    moveID   CHAR(4) UNIQUE NOT NULL,
    cid          CHAR(4) NOT NULL REFERENCES Characters(cid),
    name       TEXT NOT NULL,
    damage   INTEGER,
    CHECK(damage > 0),
  Primary key(moveID, cid)
);
```

Functional Dependencies
moveID, cid —> name, damage

Sample Output

	moveid character(4)	cid character(4)	mname text	damage integer
1	m000	c000	down B	10
2	m001	c001	B Special	15
3	m002	c002	neutral B	7
4	m003	c003	Side smash	12
5	m004	c004	Side smash	20

Items Table:
This table represents all the different items in a match. Not every match has items depending on the settings.

```
--Items--
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)
);
```

Functional Dependencies
itemID, matchID —> name, description, damageGiven

Sample Output

	itemid character(4)	iname text	idescription text	damagegiven integer	matchid character(4)
1	i000	bat	one hit KO	250	m000
2	i001	bomb	blows stuff up	10	m001
3	i002	gun	shoots stuff	5	m002
4	i003	mine	explodes when activated	20	m003
5	i004	Mr. Saturn	hurts	25	m004

Stages Table:
The table that represents all the different stages. It has a one to one relationship with the matches table because there is only one stage per match.

```
--Stages Table--
CREATE TABLE Stages (
    sid             CHAR(4) UNIQUE NOT NULL,
    name            TEXT NOT NULL,
    description     TEXT,
    origin          TEXT,
    song            TEXT NOT NULL,
  PRIMARY KEY(sid)
);
```

Functional Dependencies
sid —> name, description, origin, song

Sample Output

	sid character(4)	sname text	sdescription text	sorigin text	song text
1	s000	Final Destination	Good stage	some place	a song
2	s001	Dream Land	nice stage	a place	dream song
3	s002	Battlefield	a good stage	a nice place	a good song
4	s003	Donkey Kong Land	big stage	a place	kong song
5	s004	Temple	large stage	zelda	zelda song

RegStage Table:
A stage can be either a regular stage with special effects or an Omega Stage (with no special effects). This table represents the regular stages.

```
--RegStage Table--
CREATE TABLE RegStage (
    sid  CHAR(4)  NOT NULL REFERENCES Stages(sid),
    size INTEGER NOT NULL,
    CHECK (size > 0),
  PRIMARY KEY(sid)
);
```

Functional Dependencies
sid —> size

Sample Output

	sid character(4)	size integer
1	s001	8
2	s002	15

OmegaStage Table:
Represents the stages that are only omega. So their size is just the default size (different size from the regular stage)

```
--OmegaStage Table--
Create table OmegaStage (
    sid          CHAR(4)  NOT NULL REFERENCES Stages(sid),
    defaultSize  INTEGER NOT NULL,
    CHECK (defaultSize > 0),
  PRIMARY KEY(sid)
);
```

Functional Dependencies
sid —> defaultSize

Sample Output

	sid character(4)	defaultsize integer
1	s000	10
2	s002	12
3	s003	8
4	s004	8

Special Effects Table:
Each stage has its own special effects and this table stores all the information about the special effect.

```
--Special Effects Table--
CREATE TABLE specialEffects (
    eID  CHAR(4) UNIQUE NOT NULL,
    sid  CHAR(4)  NOT NULL REFERENCES Stages(sid),
    name TEXT,
    description  TEXT,
    damage       INTEGER,
    CHECK (damage > 0),
  PRIMARY KEY(eID, sid)
);
```

Functional Dependencies
eID, sid —> name, description, damage

Sample Output

	eid character(4)	sid character(4)	sename text	sedescription text	damage integer
1	e000	s001	wind	wind	1
2	e001	s003	bombs	blow stuff up	10

Views

Player Character View:
A view into the Players and Characters tables. This is useful so we can see what players play as what characters.

```
create view PlayerCharacter as
select Characters.*, userName, firstName, favCharacter
from Players, Characters, Player_Character
where Players.pid = Player_Character.pid
  and Characters.cid = Player_Character.cid
```

Sample Output

	cid character(4)	cname text	cdescription text	class text	corigin text	username text	firstname text	favcharacter text
1	c000	Pikachu	Down B	lightweight	pokemon	FrankBond007	Frank	Pikachu
2	c002	Bowser	Is a big dude	heavyweight	Mario	baseballDude	John	Bowser
3	c002	Bowser	Is a big dude	heavyweight	Mario	FrankBond007	Frank	Pikachu
4	c003	Link	Has a lot of items	middleweight	Zelda	coolDude5	Josh	Link
5	c001	Marth	Uses a sword	middleweight	fire emblem	jamesGriff	James	Marth
6	c004	Ness	swings a bat	lightweight	Earth Bound	Alan007	Alan	Kirby

Match View (without items):
A look into the match without items

```
create view matchView as
select m.matchID, m.date, m.time, m.type, m.stockLives,
m.numOfPlayers,
      s.sid, s.sName, s.sOrigin, s.song, c.cName, p.firstName
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
```

Sample Output

	matchid character(4)	date date	time integer	type text	stocklives integer	numofplayers integer	sid character(4)	sname text	sorigin text	song text	cname text	firstname text
1	m000	2016-01-02	100	Stock	2	2	s000	Final Destination	some place	a song	Pikachu	Frank
2	m001	2016-01-02	200	Stock	2	2	s001	Dream Land	a place	dream song	Marth	John
3	m002	2016-01-02	300	Stock	2	2	s002	Battlefield	a nice place	a good song	Bowser	Josh
4	m003	2016-01-02	400	Stock	2	2	s003	Donkey Kong Land	a place	kong song	Link	James
5	m004	2016-01-09	340	Stock	6	8	s004	Temple	zelda	zelda song	Ness	Alan

Match View (with items):
A look into the match with items

```
create view matchViewItems as
select m.matchID, m.date, m.time, m.type, m.stockLives,
m.numOfPlayers, s.sid, s.sName, s.sOrigin, s.song, c.cName,
p.firstName, i.itemID, i.iName, i.damageGiven
from match m, stages s, Characters c, Character_Match cm,
Players p, Player_Match pm, Items i
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
  and i.matchID = m.matchID
```

Sample Output

	matchid character(4)	date date	time integer	type text	stocklives integer	numofplayers integer	sid character(4)	sname text	sorigin text	song text	cname text	firstname text	itemid character(4)	iname text	damagegiven integer
1	m000	2016-01-02	100	Stock	2	2	s000	Final Destination	some place	a song	Pikachu	Frank	i000	bat	250
2	m001	2016-01-02	200	Stock	2	2	s001	Dream Land	a place	dream song	Marth	John	i001	bomb	10
3	m002	2016-01-02	300	Stock	2	2	s002	Battlefield	a nice place	a good song	Bowser	Josh	i002	gun	5
4	m003	2016-01-02	400	Stock	2	2	s003	Donkey Kong Land	a place	kong song	Link	James	i003	mine	20
5	m004	2016-01-09	340	Stock	6	8	s004	Temple	zelda	zelda song	Ness	Alan	i004	Mr. Saturn	25

Database View:
A view into almost the entire database.

```
create view dbView as
select p.pid, p.firstName, p.lastName, p.dob, p.favCharacter,
p.favStage,
        m.matchID, m.date, m.time, m.type, m.stockLives,
m.numOfPlayers,
        s.*, c.*,
        i.itemID, i.iName, i.iDescription, i.damageGiven,
        t.tName, t.level,
        mo.moveID, mo.mName, mo.damage
from Players p, Match m, Player_Match pm, Stages s, Characters
c, Player_Character pc, Character_Match cm,
     Items i, Tier t, Moves mo
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 mo.cid = c.cid
```

Sample Output

	pid character(4)	firstname text	lastname text	dob date	favcharacter text	favstage text	matchid character(4)	date date	time integer	type text	stocklives integer	numofplayers integer	sid character(4)	sname text	sdescription text	sorigin text
1	p000	Frank	Siderio	1996-02-19	Pikachu	Final Destination	m000	2016-01-02	100	Stock	2	2	s000	Final Destination	Good stage	some plac
2	p001	John	Doe	1996-07-09	Bowser	Final Destination	m001	2016-01-02	200	Stock	2	2	s001	Dream Land	nice stage	a place
3	p000	Frank	Siderio	1996-02-19	Pikachu	Final Destination	m000	2016-01-02	100	Stock	2	2	s000	Final Destination	Good stage	some plac
4	p002	Josh	Smith	1995-12-11	Link	Battlefied	m002	2016-01-02	300	Stock	2	2	s002	Battlefield	a good stage	a nice pl
5	p003	James	Griffin	1995-06-09	Marth	Dream Land	m003	2016-01-02	400	Stock	2	2	s003	Donkey Kong Land	big stage	a place
6	p004	Alan	Labouseur	1986-01-03	Kirby	Battlefield	m004	2016-01-09	340	Stock	6	8	s004	Temple	large stage	zelda

song text	cid character(4)	cname text	cdescription text	class text	corigin text	itemid character(4)	iname text	idescription text	damagegiven integer	tname text	level text	moveid character(4)	mname text	damage integer
a song	c000	Pikachu	Down B	lightweight	pokemon	i000	bat	one hit KO	250	Tier 1	B	m000	down B	10
dream song	c002	Bowser	Is a big dude	heavyweight	Mario	i001	bomb	blows stuff up	10	Tier 3	A	m002	neutral B	7
a song	c002	Bowser	Is a big dude	heavyweight	Mario	i000	bat	one hit KO	250	Tier 3	A	m002	neutral B	7
a good song	c003	Link	Has a lot of items	middleweight	Zelda	i002	gun	shoots stuff	5	Tier 1	A	m003	Side smash	12
kong song	c001	Marth	Uses a sword	middleweight	fire emblem	i003	mine	explodes when activated	20	Tier 2	A	m001	B Special	15
zelda song	c004	Ness	swings a bat	lightweight	Earth Bound	i004	Mr. Saturn	hurts	25	Tier 1	B+	m004	Side smash	20

Queries and Reports

Character with the Strongest Move:
Returns the character who has the strongest move.

```
select c.cid, c.cName, mo.mName, mo.damage as
highestDamage
from Characters c, Moves mo
where c.cid = mo.cid
order by mo.damage desc
limit 1;
```

Sample Output

	cid character(4)	cname text	mname text	highestdamage integer
1	c004	Ness	Side smash	20

Player who uses the Strongest Move:
Returns the player who uses the character with the strongest move.

```
select p.pid, p.firstName, c.cid, c.cName, mo.mName,
mo.damage
from Characters c, Players p, Player_Character pc, Moves mo
where p.pid = pc.pid
  and pc.cid = c.cid
  and c.cid = mo.cid
order by mo.damage desc
limit 1;
```

Sample Output

	pid character(4)	firstname text	cid character(4)	cname text	mname text	damage integer
1	p004	Alan	c004	Ness	Side smash	20

Special Effect with Players and Characters:
Returns the players and characters (they played with) on a stage with special effects.

```
select p.firstName, c.cName, s.sName, se.seName, se.damage
from Players p, Characters c, Stages s, specialEffects se,
        Match m,
     Player_Match pm, Character_Match cm
where se.sid = s.sid
  and s.sid = m.sid
  and m.matchID = cm.matchID
  and c.cid = cm.cid
  and p.pid = pm.pid and pm.matchID = m.matchID
```

Sample Output

	firstname text	cname text	sname text	sename text	damage integer
1	John	Marth	Dream Land	wind	1
2	James	Link	Donkey Kong Land	bombs	10

Stored Procedures

Player Stage:
Returns the players who have played on the passed in stage.

```
create or replace function playerStage(text, REFCURSOR)
returns REFCURSOR as
$$
declare
  stage     text      := $1;
  resultSet REFCURSOR := $2;

begin
  open resultSet for
    select p.pid, p.firstName, p.lastName
    from Players p, Player_Match pm, Stages s, Match m
    where p.pid = pm.pid
      and pm.matchID = m.matchID
      and m.sid = s.sid
      and s.sName = stage;
  return resultSet;
end;
$$
language plpgsql;
```

Sample Output

	pid character(4)	firstname text	lastname text
1	p000	Frank	Siderio

Player Character Stage:
Returns the players and characters who have played on the passed in 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.userName, 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;
```

Sample Output

	pid character(4)	username text	cid character(4)	cname text
1	p000	FrankBond007	c000	Pikachu

Character Moves:
Returns all the moves for the passed in character.

```
create or replace function characterMoves(text, REFCURSOR)
returns REFCURSOR as
$$
declare
  character text      := $1;
  resultSet REFCURSOR := $2;

begin
  open resultSet for
  select c.cName, mo.moveID, mo.mName, mo.damage, c.cid
  from Moves mo, Characters c
  where c.cid = mo.cid
    and c.cName = character;
  return resultSet;
end;
$$
language plpgsql;
```

Sample Output

	cname text	moveid character(4)	mname text	damage integer	cid character(4)
1	Pikachu	m000	down B	10	c000

Add Player Trigger

```
create or replace function addPlayer() returns trigger as
$$
    begin
      if NEW.pid is null then
        raise exception 'Invalid pid';
      end if;
      if NEW.userName is null then
        raise exception 'Invalid userName';
      end if;
      if NEW.password is null then
        raise exception 'Invalid password';
      end if;
      insert into Players(pid, userName, password, firstName,
lastName, dob, favCharacter, favStage)
                values (NEW.pid, NEW.userName, NEW.password,
NEW.firstName, NEW.lastName, NEW.dob, NEW.favCharacter,
NEW.favStage);
      return new;
    end;
$$ language plpgsql;
```

Trigger

```
create trigger addPlayer
after insert on Players
for each row execute procedure addPlayer();
```

Security

With this database there is only a need for two different roles (an admin and a player). An admin has access to whatever he/she wants to do. A player only has access to view anything in the database.

```
create role db_admin
grant select, insert, update, delete
on all tables in schema public
to db_admin

create role player
grant select
on Players, Stages, OmegaStage, RegStage, specialEffects,
Match, Characters, Moves, Tier, Items
to player
```

Implementation Notes, Known Problems Future Enhancements

The implementation went well with not too many issues. This database is useful for generating reports and statistics about the game. It could be used for analyzing Players behavior to learn more about them to then make the game better.

There aren't any big issues in this design. Even though the database covers almost the entirety of the game it could be much more complex. For example, not all Items give damage there're some items that give health. If I wanted to enhance this database in the future that is something I could easily add. Also, the Moves table could be more complex. Not every move is the same. Some moves are projectiles and some do not give damage and can actually receive other projectiles and shoot them back out. With this design it allows for future implementations for the full game.