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 and store procedures. There will also be implementations of different reports.

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).

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

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

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

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

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

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

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, level)
);
```

Functional Dependencies

name,level —> cid

Sample Output

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

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)
);
```

Functional Dependencies

itemID, matchID —> name, description, damageGiven

Sample Output

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

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

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

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