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

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

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.origin, s.song, c.name, 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
```

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.origin, s.song, c.name,
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
```

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
```

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;
```

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;
```

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
```

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;
```

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;
```

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;
```

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
```