Mario Super Sluggers

The quest to rescue Baseball Kingdom!

Trevor Pirone
DBMS 308
12/04/17
# Table of Contents

Executive Summary.................................................................................. 3  
ER Diagram................................................................................................. 4  
Tables........................................................................................................... 5-17  
Views........................................................................................................... 18-21  
Reports........................................................................................................ 22-26  
Stored Procedures.................................................................................... 27-29  
Triggers....................................................................................................... 30-31  
Security....................................................................................................... 32  
Implementation Notes............................................................................. 33-34  
Known Problems....................................................................................... 35  
Future Enhancements............................................................................. 36  
Conclusion.................................................................................................. 37
Executive Summary

Mario Super Sluggers is the Wii sequel to the GCN game Mario Superstar Baseball. Not only are Mario and his friends tasked to relinquish Baseball Kingdom back from the wrath of Bowser’s evil claws, he was also tasked to create a database to keep track of all of the players, teams, items, stadiums, captains, and minigames on the island. Due to Mario’s occupation (plumber), he is not technically competent enough to do so which is why he hired Trevor Pirone (that’s me) to create it for him.

The database consists of create statements for all elements that appear in the game (i.e. Characters, Stats, Captains, Teams, Stadiums, etc.). From these tables, queries, views, triggers, and stored procedures are used to retrieve more in-depth information about the game (i.e. Return the characters that play for the Mario Fireballs.) Specific users have roles that can modify the database in such a way that is needed to accommodate any changes that occur in the kingdom (i.e. creating a new player, adding a new team, removing an item, etc.)

Due to some extra requirements that are needed at the request of Trevor's boss, he added an additional character and item in the game, Alan “The Pleasant Valley Pulverizer” Labouseur and his magical staff!
Characters Tables

The characters tables includes all of the characters in the game.

CREATE TABLE Characters (  
  char_id INT NOT NULL,  
  name TEXT NOT NULL,  
  biography TEXT NOT NULL,  
  bats TEXT NOT NULL,  
  throws TEXT NOT NULL,  
  running TEXT NOT NULL,  
  fielding TEXT NOT NULL,  
  UNIQUE(char_id),  
  PRIMARY KEY(char_id)
);

Functional Dependencies
char_id \rightarrow name  
char_id \rightarrow biography  
char_id \rightarrow bats  
char_id \rightarrow throws  
char_id \rightarrow running  
char_id \rightarrow fielding
Captains Table

The captains table includes all of the players that are captains of a team.

CREATE TABLE Captains (  
cap_id INT NOT NULL,  
star_pitch TEXT NOT NULL,  
star_swing TEXT NOT NULL,  
UNIQUE(cap_id),  
PRIMARY KEY(cap_id)
);

Functional Dependencies:
cap_id \rightarrow star_pitch
cap_id \rightarrow star_swing

<table>
<thead>
<tr>
<th>cap_id</th>
<th>star_pitch</th>
<th>star_swing</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Fireball</td>
<td>Fire Swing</td>
</tr>
<tr>
<td>2</td>
<td>Tornado Ball</td>
<td>Tornado Swing</td>
</tr>
<tr>
<td>3</td>
<td>Heart Ball</td>
<td>Heart Swing</td>
</tr>
<tr>
<td>4</td>
<td>Flower Ball</td>
<td>Flower Swing</td>
</tr>
<tr>
<td>5</td>
<td>Rainbow Ball</td>
<td>Egg Swing</td>
</tr>
<tr>
<td>6</td>
<td>Suction Ball</td>
<td>Cannon Swing</td>
</tr>
<tr>
<td>7</td>
<td>Phony Ball</td>
<td>Phony Swing</td>
</tr>
<tr>
<td>8</td>
<td>Liar Ball</td>
<td>Liar Swing</td>
</tr>
<tr>
<td>9</td>
<td>Barrel Ball</td>
<td>Barrel Swing</td>
</tr>
<tr>
<td>10</td>
<td>Banana Ball</td>
<td>Banana Swing</td>
</tr>
<tr>
<td>11</td>
<td>Killer Ball</td>
<td>Breath Swing</td>
</tr>
<tr>
<td>12</td>
<td>Graffiti Ball</td>
<td>Graffiti Swing</td>
</tr>
</tbody>
</table>
Special Characters Table

This table acts as a link between the Characters and Captains tables. It represents the characters in the game that are also captains of a team.

CREATE TABLE Special_Characters (  
    char_id INT REFERENCES Characters(char_id),  
    cap_id INT REFERENCES Captains(cap_id),  
    PRIMARY KEY(char_id, cap_id)
);

Functional Dependencies:  
char_id, cap_id →

<table>
<thead>
<tr>
<th>char_id</th>
<th>cap_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>
The stats table keeps track of a player's skill level in four categories. Each stat is ranked on a score out of 10.

```sql
CREATE TABLE Stats (  
    char_id INT REFERENCES Characters(char_id),
    pitch INT NOT NULL,
    CHECK (pitch >= 0),
    bat INT NOT NULL,
    CHECK (bat >= 0),
    field INT NOT NULL,
    CHECK (field >= 0),
    run INT NOT NULL,
    CHECK (run >= 0),
    UNIQUE(char_id),
    PRIMARY KEY(char_id)
);
```

Functional Dependencies:

<table>
<thead>
<tr>
<th>char_id</th>
<th>pitch</th>
<th>bat</th>
<th>field</th>
<th>run</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>20</td>
<td>7</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>21</td>
<td>21</td>
<td>6</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>22</td>
<td>22</td>
<td>5</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>23</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>24</td>
<td>24</td>
<td>4</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>25</td>
<td>25</td>
<td>5</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>26</td>
<td>26</td>
<td>6</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>27</td>
<td>27</td>
<td>4</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>28</td>
<td>28</td>
<td>3</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>29</td>
<td>29</td>
<td>6</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>30</td>
<td>30</td>
<td>3</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>31</td>
<td>31</td>
<td>4</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>32</td>
<td>32</td>
<td>8</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>33</td>
<td>33</td>
<td>4</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>34</td>
<td>34</td>
<td>4</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>35</td>
<td>35</td>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>36</td>
<td>36</td>
<td>7</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>37</td>
<td>37</td>
<td>4</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>38</td>
<td>38</td>
<td>3</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>39</td>
<td>39</td>
<td>5</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>40</td>
<td>40</td>
<td>4</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>41</td>
<td>41</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>42</td>
<td>42</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>43</td>
<td>43</td>
<td>4</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>
The teams table includes all of the teams that can be played with in the game.

CREATE TABLE Teams (  
    team_id INT NOT NULL,  
    team_name TEXT NOT NULL,  
    team_info TEXT NOT NULL,  
    UNIQUE(team_id),  
    PRIMARY KEY(team_id)
);

Functional Dependencies:  
team_id \rightarrow team_name  
team_id \rightarrow team_info
has Team Table

This table acts as a link between the Captains and Teams tables. It represents the teams that are owned by a captain.

CREATE TABLE hasTeam (  
cap_id INT REFERENCES Captains(cap_id),  
team_id INT REFERENCES Teams(team_id),  
PRIMARY KEY (cap_id, team_id)
);

Functional Dependencies:  
cap_id, team_id →

```
<table>
<thead>
<tr>
<th>cap_id</th>
<th>team_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>
```
plays For Table

This table acts as a link between the Characters and Teams tables. It represents the rosters for each team or what characters play on which team.

CREATE TABLE playsFor (  
  char_id INT REFERENCES Characters(char_id),  
  team_id INT REFERENCES Teams(team_id),  
  PRIMARY KEY(char_id, team_id)
);

Functional Dependencies:  
char_id, team_id →
Stadiums Table

The stadiums tables includes all of the stadiums that teams can play within for games.

CREATE TABLE Stadiums (
    stadium_id INT NOT NULL,
    field_name TEXT NOT NULL,
    day_mode BOOLEAN NOT NULL,
    night_mode BOOLEAN NOT NULL,
    hazards TEXT NOT NULL,
    UNIQUE(stadium_id),
    PRIMARY KEY(stadium_id)
);

Functional Dependencies:

- stadium_id → field_name
- stadium_id → night_mode
- stadium_id → day_mode
- stadium_id → hazards
has Stadium Table

This table acts as a link between the Teams and Stadiums tables. It represents all of the teams that own their own stadium.

CREATE TABLE hasStadium (  
    team_id INT REFERENCES Teams(team_id),  
    stadium_id INT REFERENCES Stadiums(stadium_id),  
    PRIMARY KEY(team_id, stadium_id) 
);

Functional Dependencies:  
team_id, stadium_id →

<table>
<thead>
<tr>
<th>team_id</th>
<th>stadium_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>
**Items Table**

The items table includes all of the items players can use in the game on offense to attack the defense.

CREATE TABLE Items (  
    item_id INT NOT NULL,  
    item_name TEXT NOT NULL,  
    item_desc TEXT NOT NULL,  
    UNIQUE(item_id),  
    PRIMARY KEY(item_id)  
);  

Functional Dependencies:  
item_id \(\rightarrow\) item_name  
item_id \(\rightarrow\) item_desc

<table>
<thead>
<tr>
<th>item_id</th>
<th>item_name</th>
<th>item_desc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Green Shell</td>
<td>Shoots at a player and dazes if it hits.</td>
</tr>
<tr>
<td>2</td>
<td>Fire Ball</td>
<td>Some fireballs bounce to burn someone.</td>
</tr>
<tr>
<td>3</td>
<td>Mini Boos</td>
<td>Makes the ball and its shadow invisible for about six seconds. The ball</td>
</tr>
<tr>
<td>4</td>
<td>Bob-omb</td>
<td>When a Bob-omb gets shot in the outfield, it sits and stays until it exp</td>
</tr>
<tr>
<td>5</td>
<td>POW-Ball</td>
<td>All fielders are stunned by a small earthquake for about four seconds w</td>
</tr>
<tr>
<td>6</td>
<td>Banana Peel</td>
<td>Five bananas are shot out to slip the outfielders.</td>
</tr>
<tr>
<td>7</td>
<td>Lightning Bolt</td>
<td>Will strike any characters on the screen.</td>
</tr>
<tr>
<td>8</td>
<td>Alan's Magic Staff</td>
<td>All max stats on every player for the rest of the inning. Very rare!</td>
</tr>
</tbody>
</table>
The minigames table includes all of the minigames that can be played in the game.

```sql
CREATE TABLE Minigames (
    minigame_id INT NOT NULL,
    game_name TEXT NOT NULL,
    game_desc TEXT NOT NULL,
    hi_score INT NOT NULL,
    CHECK (hi_score >= 0),
    UNIQUE(minigame_id),
    PRIMARY KEY(minigame_id)
);
```

Functional Dependencies:
- `minigame_id` → `game_name`
- `minigame_id` → `game_desc`
- `minigame_id` → `hi_score`

<table>
<thead>
<tr>
<th>minigame_id</th>
<th>game_name</th>
<th>game_desc</th>
<th>hi_score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bob-omb Bash</td>
<td>Hit bob-ombs into the sky and earn as many points as you can!</td>
<td>6000</td>
</tr>
<tr>
<td>2</td>
<td>Wall Ball</td>
<td>Throw the ball at the walls to break them and earn the most points!</td>
<td>4000</td>
</tr>
<tr>
<td>3</td>
<td>Piranha Panic</td>
<td>Prevent the piranha plants from reaching your player by throwing baseballs at them!</td>
<td>3500</td>
</tr>
<tr>
<td>4</td>
<td>Sea Catch</td>
<td>Maneuver the player to catch gems to earn points! The more valuable gems, the more points can be earned!</td>
<td>3500</td>
</tr>
<tr>
<td>5</td>
<td>Barrel Bash</td>
<td>Pend on barrels and bob-omb by hitting them with baseballs protecting your shield!</td>
<td>2500</td>
</tr>
<tr>
<td>6</td>
<td>Short E</td>
<td>Throw baseballs at the ghosts on the screen to make them disappear!</td>
<td>2500</td>
</tr>
<tr>
<td>7</td>
<td>Ringer Ringer</td>
<td>Collect the ringer points by running around the bases! Be careful! Don't hit the Ginger Ringer's tentacles!</td>
<td>2000</td>
</tr>
<tr>
<td>8</td>
<td>Graffiti Runner</td>
<td>Paint as much of the canvas as you can by running around with the paintbrush!</td>
<td>1500</td>
</tr>
<tr>
<td>9</td>
<td>Bowser Pinball</td>
<td>Keep a spiked ball within the area and hit coins and walls to earn points!</td>
<td>1000</td>
</tr>
</tbody>
</table>
has Items Table

This table acts as a link between the Stadiums and Items tables. It represents all of the items a player can receive whilst playing a game at a stadium.

CREATE TABLE hasItems (  
  stadium_id INT REFERENCES Stadiums(stadium_id),  
  item_id INT REFERENCES Items(item_id),  
  PRIMARY KEY(stadium_id, item_id)
);

Functional Dependencies:  
stadium_id, item_id →

<table>
<thead>
<tr>
<th>stadium_id</th>
<th>item_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
</tr>
<tr>
<td>17</td>
<td>3</td>
</tr>
<tr>
<td>18</td>
<td>3</td>
</tr>
<tr>
<td>19</td>
<td>3</td>
</tr>
<tr>
<td>20</td>
<td>3</td>
</tr>
<tr>
<td>21</td>
<td>3</td>
</tr>
<tr>
<td>22</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>stadium_id</th>
<th>item_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>5</td>
</tr>
<tr>
<td>37</td>
<td>5</td>
</tr>
<tr>
<td>38</td>
<td>5</td>
</tr>
<tr>
<td>39</td>
<td>5</td>
</tr>
<tr>
<td>40</td>
<td>5</td>
</tr>
<tr>
<td>41</td>
<td>6</td>
</tr>
<tr>
<td>42</td>
<td>6</td>
</tr>
<tr>
<td>43</td>
<td>6</td>
</tr>
<tr>
<td>44</td>
<td>6</td>
</tr>
<tr>
<td>45</td>
<td>6</td>
</tr>
<tr>
<td>46</td>
<td>6</td>
</tr>
<tr>
<td>47</td>
<td>6</td>
</tr>
</tbody>
</table>
has Minigame Table

This table acts as a link between the Stadiums and Minigames tables. It represents all of the stadiums that have a minigame and where each one is held.

CREATE TABLE hasMinigame (
    stadium_id INT REFERENCES Stadiums(stadium_id),
    minigame_id INT REFERENCES Minigames(minigame_id),
    PRIMARY KEY(stadium_id, minigame_id)
);

Functional Dependencies:

stadium_id, minigame_id →
Views

View 1: Create a view that returns all of the characters ids, names, name of their team, and their stats ordered by the team name and character name alphabetically, respectively.

CREATE VIEW view_all_characters_with_team_and_stats AS
SELECT s.char_id, c.name, t.team_name, pitch, bat, field, run
FROM Teams t
INNER JOIN playsFor p ON t.team_id = p.team_id
INNER JOIN Characters c ON c.char_id = p.char_id
INNER JOIN Stats s ON s.char_id = c.char_id
GROUP BY s.char_id, c.name, t.team_name
ORDER BY t.team_name ASC, c.name ASC;

SELECT *
FROM view_all_characters_with_team_and_stats;
Views

View 2: Create a view that returns all information about stadiums with their corresponding minigames.

```sql
CREATE VIEW view_all_stadiums_with_minigames
AS
SELECT s.stadium_id, s.field_name, s.day_mode, s.night_mode, s.hazards, m.minigame_id,
       m.game_name, m.game_desc, m.hi_score
FROM Stadiums s
FULL OUTER JOIN hasMinigame h ON s.stadium_id = h.stadium_id
FULL OUTER JOIN Minigames m ON h.minigame_id = m.minigame_id
WHERE s.stadium_id != 0 OR m.minigame_id != 0
GROUP BY s.stadium_id, s.field_name, s.day_mode, s.night_mode, s.hazards, m.minigame_id,
       m.game_name, m.game_desc, m.hi_score
ORDER BY s.stadium_id, m.minigame_id;
```

```sql
SELECT *
FROM view_all_stadiums_with_minigames;
```
Views

View 3: Return the ids, star pitches, and star swings of captains that have a Banana Swing or Fireball pitch.

CREATE VIEW view_captain_banana_swing_or_fireball AS
SELECT *
FROM Captains
WHERE (star_swing = 'Banana Swing'
OR star_pitch = 'Fireball')
AND cap_id IN(SELECT char_id
FROM Special_Characters
WHERE char_id IN (SELECT char_id
FROM Characters
WHERE char_id > 0 AND char_id <= 12
))
);

SELECT *
FROM view_captain_banana_swing_or_fireball;

<table>
<thead>
<tr>
<th>cap_id</th>
<th>star_pitch</th>
<th>star_swing</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Fireball</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>Banana Ball</td>
</tr>
</tbody>
</table>
Views

View 4: Return the id, name, and score of all minigames with a high score greater than 4000.

CREATE VIEW minigames_with_hi_score_greater_than_4000 AS
SELECT minigame_id, game_name, hi_score, COUNT(minigame_id)
FROM Minigames
WHERE hi_score > 4000
GROUP BY hi_score, minigame_id, game_name
HAVING COUNT(minigame_id) > 0
ORDER BY minigame_id;

SELECT *
FROM minigames_with_hi_score_greater_than_4000;
Reports

Query 1: Get the names, ids, bat stats, and pitch stats of all players on the Wario Muscles with a batting stat greater than 6 or a pitching stat less than 4.

```
SELECT playsFor.char_id, Characters.name, playsFor.team_id, bat, pitch
FROM Characters, Teams, playsFor, Stats
WHERE Characters.char_id = playsFor.char_id
AND Teams.team_id = playsFor.team_id
AND playsFor.team_id = 7
AND Characters.char_id = Stats.char_id
AND (bat > 6 OR pitch < 4)
GROUP BY playsFor.char_id, Characters.name, playsFor.team_id, bat, pitch
ORDER BY Characters.name;
```

<table>
<thead>
<tr>
<th>char_id</th>
<th>name text</th>
<th>team_id</th>
<th>bat integer</th>
<th>pitch integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Green Paratroopa</td>
<td>7</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>King Boo</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>Koopa Troopa</td>
<td>7</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>Wario</td>
<td>7</td>
<td>8</td>
<td>5</td>
</tr>
</tbody>
</table>
Reports

Query 2: Get the names, ids, and run stats of all captains that own a stadium where their run stat is greater than 5 in reverse alphabetical order.

```
SELECT DISTINCT Characters.name, Special_Characters.char_id, Captains.cap_id, hasTeam.team_id, hasStadium.stadium_id, run
FROM Characters, Special_Characters, Captains, hasTeam, Teams, hasStadium, Stadiums, Stats
WHERE Characters.char_id = Special_Characters.char_id
AND Special_Characters.cap_id = Captains.cap_id
AND Captains.cap_id = hasTeam.cap_id
AND hasTeam.team_id = Teams.team_id
AND Teams.team_id = hasStadium.team_id
AND hasStadium.stadium_id = Stadiums.stadium_id
AND Stadiums.stadium_id != 0
AND Characters.char_id = Stats.char_id
AND run > 5
ORDER BY Characters.name DESC;
```
Query 3: Return the name and id of the character with the highest average of all stats.

```
SELECT c.name, s.char_id, ROUND(AVG((pitch + bat + field + run) / 4.0), 2) AS Average
FROM Characters c
INNER JOIN Stats s ON c.char_id = s.char_id
GROUP BY s.char_id, c.name
ORDER BY Average DESC
LIMIT 1;
```
Query 4: Return the name and ids of the teams that play in a stadium that is only available in day or night mode. (i.e. one or the other, NOT both)

```
SELECT t.team_id, t.team_name
FROM Teams t
FULL OUTER JOIN hasStadium h ON t.team_id = h.team_id
FULL OUTER JOIN Stadiums s ON h.stadium_id = s.stadium_id
WHERE (s.day_mode = TRUE AND s.night_mode = FALSE)
OR (s.day_mode = FALSE AND s.night_mode = TRUE)
GROUP BY t.team_id, t.team_name
ORDER BY t.team_id ASC;
```

<table>
<thead>
<tr>
<th>team_id</th>
<th>team_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Luigi Knights</td>
</tr>
<tr>
<td>2</td>
<td>Bowser Monsters</td>
</tr>
<tr>
<td>3</td>
<td>Bowser Jr. Rookies</td>
</tr>
</tbody>
</table>
Query 5: Return the names, character ids, and the team name of all characters with a fielding stat larger than or equal to 7. (NOT including free agents.)

SELECT Stats.char_id, name, team_name, field
FROM Characters, Teams, playsFor, Stats
WHERE Characters.char_id = playsFor.char_id
AND playsFor.team_id = Teams.team_id
AND Characters.char_id = Stats.char_id
AND field >= 7
AND Teams.team_id <> 0
ORDER BY Teams.team_name ASC, field DESC, Characters.name;
Stored Procedures

Stored Procedure 1: Create a function that allows a user to input the name of the player and find the team they play for.

CREATE OR REPLACE FUNCTION locatePlayer(TEXT)
RETURNS TABLE(name TEXT, team_name TEXT) AS
$$
DECLARE
    findChar TEXT := $1;
BEGIN
    RETURN QUERY
    SELECT Characters.name, Teams.team_name
    FROM Characters, playsFor, Teams
    WHERE Characters.char_id = playsFor.char_id
    AND playsFor.team_id = Teams.team_id
    AND Characters.name = findChar;
END;
$$ LANGUAGE plpgsql;

SELECT locatePlayer('Dry Bones');
SELECT locatePlayer('Wiggler');
SELECT locatePlayer('Alan Labouseur');
Stored Procedure 2: Create a function that returns the name, team name, and team stadium of a player. If a player is not a captain, then the result will be empty.

```sql
CREATE OR REPLACE FUNCTION teamOwn(TEXT)
RETURNS TABLE(name TEXT, team_name TEXT, field_name TEXT) AS $$
DECLARE
    findOwner TEXT := $1;
BEGIN
    RETURN QUERY
    SELECT c.name, t.team_name, s.field_name
    FROM Characters c
    INNER JOIN Special_Characters sc ON c.char_id = sc.char_id
    INNER JOIN Captains a ON sc.cap_id = a.cap_id
    INNER JOIN hasTeam h ON a.cap_id = h.cap_id
    INNER JOIN Teams t ON h.team_id = t.team_id
    INNER JOIN hasStadium d ON t.team_id = d.team_id
    INNER JOIN Stadiums s ON d.stadium_id = s.stadium_id
    WHERE c.name = findOwner
    GROUP BY c.name, t.team_name, s.field_name
    LIMIT 1;
END;
$$ LANGUAGE plpgsql;
```

```
SELECT teamOwn('Daisy');
SELECT teamOwn('Waluigi');
SELECT teamOwn('Funky Kong');
```
Stored Procedures

Stored Procedure 3: Create a function that allows the user to search for a team’s roster by inputting the name of the team. (Also works for free agents!)

CREATE OR REPLACE FUNCTION showRoster(TEXT)
RETURNS TABLE(team_name TEXT, name TEXT, char_id INT) AS $$
DECLARE
   teamName TEXT := $1;
BEGIN
   RETURN QUERY
   SELECT t.team_name, c.name, c.char_id
   FROM Characters c
   FULL OUTER JOIN playsFor p ON c.char_id = p.char_id
   FULL OUTER JOIN Teams t ON p.team_id = t.team_id
   WHERE t.team_name = teamName
   GROUP BY t.team_name, c.name, c.char_id
   ORDER BY c.name, c.char_id;
END
$$ LANGUAGE plpgsql;

SELECT showRoster('Free Agent');
SELECT showRoster('DK Wilds');
SELECT showRoster('Peach Monarchs');
Triggers

Trigger 1: Create a trigger that checks if a player is added to the Characters table, when added to the stats table, if they have any stat less than zero or greater than 10, remove them from the database.

CREATE OR REPLACE FUNCTION newStat()
RETURNS TRIGGER AS $$
BEGIN
  IF (NEW.bat < 0 OR NEW.bat > 10) OR (NEW.pitch < 0 OR NEW.pitch > 10) OR (NEW.field < 0 OR NEW.field > 10) OR (NEW.run < 0 OR NEW.run > 10) THEN
    DELETE FROM Stats WHERE pitch = NEW.pitch;
    DELETE FROM Stats WHERE bat = NEW.bat;
    DELETE FROM Stats WHERE field = NEW.field;
    DELETE FROM Stats WHERE run = NEW.run;
    DELETE FROM Stats WHERE char_id = NEW.char_id;
    DELETE FROM Characters WHERE char_id = NEW.char_id;
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER newStat
AFTER INSERT ON Stats
FOR EACH ROW
EXECUTE PROCEDURE newStat();

INSERT INTO Characters(char_id, name, biography, bats, throws, running, fielding) VALUES (75, 'Trevor Pirone', 'Really cute.', 'Right', 'Right', 'Enlarge', 'None');
SELECT * FROM Characters;
INSERT INTO Stats(char_id, pitch, bat, field, run) VALUES (75, 11, 11, 11, 11);
SELECT * FROM Stats;
SELECT * FROM Characters;
Triggers

Trigger 2: Create a trigger that checks if the high score of a new minigame is greater than 9999 (the max value allowed in the game). If so, delete the minigame from the database.

CREATE OR REPLACE FUNCTION newMinigame()
RETURNS TRIGGER AS
$$
BEGIN
    IF (NEW.hi_score > 9999) THEN
        DELETE FROM Minigames WHERE minigame_id = NEW.minigame_id;
        DELETE FROM Minigames WHERE game_name = NEW.game_name;
        DELETE FROM Minigames WHERE game_desc = NEW.game_desc;
        DELETE FROM Minigames WHERE hi_score = NEW.hi_score;
    END IF;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER newMinigame
AFTER INSERT ON Minigames
FOR EACH ROW
EXECUTE PROCEDURE newMinigame();

INSERT INTO Minigames(minigame_id, game_name, game_desc, hi_score) VALUES
(10, 'Coin Robbers', 'Catch the coins that are being thrown around the field. The player that catches the most amount of coins before the time runs out is the winner.', 10000);
SELECT * FROM Minigames;
Security

Admin Role:
The admin will have access to modify all tables present in the database.

CREATE ROLE ADMIN;
GRANT ALL
ON ALL TABLES IN SCHEMA PUBLIC
TO ADMIN;

Captain Role:
The captain will have the privilege to select, update, and delete information on teams, stadiums, and minigames.

CREATE ROLE CAPTAIN;
GRANT SELECT, UPDATE, DELETE
ON Teams, Stadiums, Minigames
TO CAPTAIN;

Guest Role:
The guest can use select statements on all tables, but is revoked from inserting, updating, or deleting while interacting with the database.
CREATE ROLE GUEST;
GRANT SELECT
ON ALL TABLES IN SCHEMA PUBLIC
TO GUEST;
REVOKE INSERT, UPDATE, DELETE
ON ALL TABLES IN SCHEMA PUBLIC
FROM GUEST;
Implementation Notes

It should be noted that the Stats table only reflects the stats of characters without enabling star players. (When a player is a star all stats not maxed out increase by 1.)

This database does not take into account scout missions that appear in the game’s story mode due to sheer size and time.

Items are available in every game and stadium, but when playing an exhibition game, items can be turned off, so the database is assuming that items are always enabled.
Implementation Notes

Similar to the items, star powers (star pitches and star swings) can also be disabled when playing an exhibition game.

In the game, the player has the option to pick from 12 captains. However, captains also play for other captains teams (i.e. Luigi on Mario’s team, Daisy on Peach’s, etc.) The captains that play for other captains are known as sub-captains which is not covered in the database. So, if a user wanted to see the roster for the Luigi Knights, the team would be empty because of the aforementioned reason.
Known Problems

Some known problems include...

- The teamOwned stored procedure, although it works, when specifying a non-captain character, it returns an empty result instead of having a result specifying only captains own teams.

- Missing scout missions from the story mode due to size and time. Does not break the database, however. (This would have made for some cool triggers though.)

- Not a groundbreaking issue, but items and stadiums do not really relate because all items can be used in every stadium. In fact, I would say items are on their own, but because it had to be implemented it best fit with stadium.

- (Maybe a more ideal fit would have been relating it to an instance of game and having the ability to enable or disable the use of items?)
Future Enhancements

Including, but not limited to...

- Adding scout missions into the database.

- Adding a stats table that takes into account a player when they are a star (and if they even have a star)!

- Rearranging the items table in a manner that makes it more involved in the database.

- Creating a games table which takes into account enabling rules specific to that game (allow star pitches/swings, mercy rule, innings, items, etc.).

- Chemistry table (which characters have good chemistry/bad chemistry).
Conclusion

Regardless of the outcome of the project, I felt that I learned a lot about building and managing a database system. I also found the use of views, stored procedures and triggers very interesting as I had no prior knowledge of them before this class.

For what it was worth, this game happens to be one of my favorite games of all time and implementing a database for it has made me learn even more about the game that I did not know prior to this project (I found which characters had the highest and lowest stat averages thanks to Query 3.).

Although there are elements of the game missing in the database, I can always go back and modify it at my leisure. It’s not perfect, but I think it’s definitely satisfying enough for Mario to pick out his best lineups to defeat Bowser and rescue the Baseball Kingdom!
"Buttered broccoli! I can't believe I lost!"
-Bowser Jr.