EarthBound

By Philip Kirwin

December 8, 2016
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table of Contents</td>
<td>2</td>
</tr>
<tr>
<td>Executive Summary</td>
<td>3</td>
</tr>
<tr>
<td>E/R Diagram</td>
<td>4</td>
</tr>
<tr>
<td>Tables</td>
<td>5-28</td>
</tr>
<tr>
<td>Views</td>
<td>29-31</td>
</tr>
<tr>
<td>Stored Procedures</td>
<td>32-33</td>
</tr>
<tr>
<td>Triggers</td>
<td>34-35</td>
</tr>
<tr>
<td>Reports</td>
<td>36-37</td>
</tr>
<tr>
<td>Roles</td>
<td>38</td>
</tr>
<tr>
<td>Implementation Notes</td>
<td>39</td>
</tr>
<tr>
<td>Known Issues</td>
<td>40</td>
</tr>
<tr>
<td>Future Enhancements</td>
<td>41</td>
</tr>
</tbody>
</table>
As is the case with most classic RPG video games, EarthBound suffers from containing a large volume of data in the form of vast arrays of characters, enemies, locations, items, and powers at the player’s disposal. Naturally, new players may find the game frustrating at first, at least in part due to the sheer volume of data that is placed before them even from the game’s introduction that can be difficult to remember and keep track of. EarthBounddb sets out to solve this problem as it is built to manage a majority of the core information that new EarthBound players can easily lose track of.

The following pages first demonstrate the core design of the database via an E/R Diagram. Next, each table in the database will be described in depth with sample data included. Following that is a demonstration of some of the views, stored procedures, triggers, and report queries that will be implemented along with the database for the sake of improving one’s ability to access specific data. Then, the roles included with the database will be described along with the privileges held by each. Finally, implementation notes will be provided along with known issues and future enhancements as to maximize the utility of this database.
E/R Diagram
Locations

This table is made to keep track of the various locations that exist in EarthBound.

CREATE TABLE Locations(
    lid INT NOT NULL UNIQUE,
    locationName TEXT NOT NULL,
    PRIMARY KEY(lid)
);

Dependencies: lid → locationName
Your Sanctuary

This table acts as a subgroup to Locations that focuses on the Locations where Ness receives melodies.

CREATE TABLE YourSanctuary(
    lid INT NOT NULL UNIQUE REFERENCES Locations(lid),
    nessEnvisions TEXT NOT NULL,
    melodyLearned INT NOT NULL UNIQUE,
    PRIMARY KEY(lid)
);

Dependencies: lid \(\rightarrow\) nessEnvisions, melodyLearned
This table organizes general information about the many towns one finds within the game.

```
CREATE TABLE Towns(
    lid INT NOT NULL UNIQUE REFERENCES Locations(lid),
    humanPopulation INT,
    dogPopulation INT,
    zombiePopulation INT,
    averageTempF INT,
    annualRainfallInches INT,
    PRIMARY KEY(lid)
);
```

Items

This table helps to organize items that players may find or purchase.

CREATE TABLE Items(
    itemId INT NOT NULL UNIQUE,
    itemName TEXT NOT NULL,
    itemCostDollars INT,
    itemDescription TEXT NOT NULL,
    PRIMARY KEY(itemId)
);

Dependencies: itemId → itemName, itemCostDollars, itemDescription
Healing Items

This table focuses on items whose purpose is to replenish health or Psychic Points.

CREATE TABLE HealingItems(
    itemId INT NOT NULL UNIQUE REFERENCES Items(itemId),
    HPRestored INT,
    PPRestored INT,
    PooHPRestored INT,
    PRIMARY KEY(itemId)
);

Dependencies: itemId → HPRestored, PPRestored, PooHPRestored
Weapons

This table includes items that help improve a character’s offensive abilities.

CREATE TABLE Weapons(
    itemId INT NOT NULL UNIQUE REFERENCES Items(itemId),
    offenseUpBy INT NOT NULL,
    errorRate DECIMAL (8, 7),
    PRIMARY KEY(itemId)
);

Dependencies: itemId $\rightarrow$ offenseUpBy, errorRate
Armor

This table focuses on items that allow a character to suffer less damage when attacked.

```
CREATE TABLE Armor(
    itemId INT NOT NULL UNIQUE REFERENCES Items(itemId),
    defenseUpBy INT NOT NULL,
    PRIMARY KEY(itemId)
);
```

Dependencies: itemId $\rightarrow$ defenseUpBy

<table>
<thead>
<tr>
<th>itemid</th>
<th>defenseupby</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>15</td>
</tr>
</tbody>
</table>
Contains Item

This table shows where items can be found by acting as a link between Items and Locations.

CREATE TABLE ContainsItem(
    lid INT NOT NULL REFERENCES Locations(lid),
    itemId INT NOT NULL REFERENCES Items(itemId),
    PRIMARY KEY(lid,itemId)
);

Dependencies: lid, itemId
Characters

This table provides data about the multitude of characters one may interact with.

CREATE TABLE Characters(
    cid INT NOT NULL UNIQUE,
    characterName TEXT NOT NULL,
    characterAge INT,
    lid INT NOT NULL REFERENCES Locations(lid),
    PRIMARY KEY(cid)
);

Dependencies: cid → characterName, characterAge, lid
Non-Player Characters

This table focuses on characters that the player can interact with, but not directly control.

CREATE TABLE NonPlayerCharacters(
    cid INT NOT NULL UNIQUE REFERENCES Characters(cid),
    dialogue TEXT NOT NULL,
    PRIMARY KEY(cid)
);

Dependencies: cid → dialogue
Quest Givers

This table focuses on NPCs who are capable of giving the player a quest or completing one.

CREATE TABLE QuestGivers(
    cid INT NOT NULL UNIQUE REFERENCES NonPlayerCharacters(cid),
    questTrigger TEXT NOT NULL,
    PRIMARY KEY(cid)
);

Dependencies: cid → questTrigger
Player Characters

This table focuses on characters that the player is in direct control of.

CREATE TABLE PlayerCharacters(
    cid INT NOT NULL UNIQUE REFERENCES Characters(cid),
    currentLevel INT NOT NULL,
    maxHitPoints INT NOT NULL,
    maxPsychicPoints INT NOT NULL,
    currentHitPoints INT NOT NULL,
    currentPsychicPoints INT NOT NULL,
    characterOffense INT NOT NULL,
    characterDefense INT NOT NULL,
    characterSpeed INT NOT NULL,
    characterIQ INT NOT NULL,
    characterGuts INT NOT NULL,
    PRIMARY KEY(cid)
);

Dependencies: cid → currentLevel, maxHitPoints, maxPsychicPoints, currentHitPoints, currentPsychicPoints, characterOffense, characterDefense, characterSpeed, characterIQ, characterGuts
Inventory

This table provides a link between Items and Player Characters, allowing them to interact.

```
CREATE TABLE Inventory(
    cid INT NOT NULL REFERENCES PlayerCharacters(cid),
    itemId INT NOT NULL REFERENCES Items(itemId),
    itemQuantity INT NOT NULL,
    PRIMARY KEY(cid, itemId)
);
```

Dependencies: cid, itemId → itemQuantity
Enemies

This table organizes the adversaries that the player will face throughout the game.

CREATE TABLE Enemies(
    eid INT NOT NULL UNIQUE,
    enemyName TEXT NOT NULL,
    enemyHitPoints INT NOT NULL,
    enemyPsychicPoints INT NOT NULL,
    enemyOffense INT NOT NULL,
    enemyDefense INT NOT NULL,
    enemySpeed INT NOT NULL,
    enemyIQ INT NOT NULL,
    enemyGuts INT NOT NULL,
    enemyEXPReward INT NOT NULL,
    enemyBountyDollars INT NOT NULL,
    enemyStatus TEXT NOT NULL,
    PRIMARY KEY(eid)
);

Dependencies: eid → enemyName, enemyHitPoints, enemyPsychicPoints, enemyOffense, enemyDefense, enemySpeed, enemyIQ, enemyGuts, enemyEXPReward, enemyBountyDollars, enemyStatus
Enemy Habitat

This table links Enemies to Locations by indicating where enemies may be found.

CREATE TABLE EnemyHabitat(
    eid INT NOT NULL REFERENCES Enemies(eid),
    lid INT NOT NULL REFERENCES Locations(lid),
    PRIMARY KEY(eid, lid)
);

Dependencies: eid, lid →
Enemy Loot

This table links Enemies to Items and thus allows enemies to drop loot when defeated.

```
CREATE TABLE EnemyLoot(
    eid INT NOT NULL REFERENCES Enemies(eid),
    itemId INT NOT NULL REFERENCES Items(itemId),
    dropChance DECIMAL(8,7) NOT NULL,
    PRIMARY KEY(eid, itemId)
);
```

Dependencies: eid, itemId → dropChance
Battle

This table links Enemies to Player Characters by storing data relevant to battles between them.

CREATE TABLE Battle(
    bid INT NOT NULL UNIQUE,
    cid INT NOT NULL REFERENCES PlayerCharacters(cid),
    eid INT NOT NULL REFERENCES Enemies(eid),
    victory BOOLEAN,
    PRIMARY KEY(bid)
);

Dependencies: bid → cid, eid, victory
This table organizes the many psychic abilities seen throughout the game.

```sql
CREATE TABLE PSI(
    PSId INT NOT NULL UNIQUE,
    psiName TEXT NOT NULL,
    psiDescription TEXT NOT NULL,
    powerLevel TEXT NOT NULL,
    PPCost INT NOT NULL,
    target TEXT NOT NULL,
    PRIMARY KEY(PSId)
);
```

Dependencies: PSId → psiName, psiDescription, powerLevel, PPCost, target
Offense PSI

This table focuses on PSI whose primary purpose is for combat.

CREATE TABLE OffensePSI(
    PSId INT NOT NULL UNIQUE REFERENCES PSI(PSId),
    damageDealt INT NOT NULL,
    statusInflicted TEXT,
    PRIMARY KEY(PSId)
);

Dependencies: PSId \(\rightarrow\) damageDealt, statusInflicted
Recover PSI

This table focuses on PSI whose primary purpose is to heal allies.

```
CREATE TABLE RecoverPSI(
    PSId INT NOT NULL UNIQUE REFERENCES PSI(PSId),
    HPRestored INT,
    PPREstored INT,
    PRIMARY KEY(PSId)
);
```

Dependencies: PSId → HPRestored, PPREstored
Assist PSI

This table focuses on PSI whose primary purpose is to support allies in battle.

CREATE TABLE AssistPSI(
    PSId INT NOT NULL UNIQUE REFERENCES PSI(PSId),
    statusInflicted TEXT,
    statAffected TEXT,
    statModifier INT,
    PRIMARY KEY(PSId)
);

Dependencies: PSId → statusInflicted, statAffected, statModifier
Other PSI

This table focuses on PSI whose primary purpose is to be fulfilled outside of combat.

```
CREATE TABLE OtherPSI(  
    PSId INT NOT NULL UNIQUE REFERENCES PSI(PSId),  
    PSIEffect TEXT,  
    PSILimitations TEXT,  
    PRIMARY KEY(PSId)
);
```

Dependencies: → PSId, PSIEffect, PSILimitations

<table>
<thead>
<tr>
<th>Data Output</th>
<th>Explain</th>
<th>Messages</th>
<th>History</th>
</tr>
</thead>
<tbody>
<tr>
<td>psid</td>
<td>psieffect</td>
<td>psilimitations</td>
<td>text</td>
</tr>
<tr>
<td>1</td>
<td>4 teleport to previously visited location must run in a straight line without any collisions for a somewhat low distance or it fails</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Character Has PSI

This table provides a link between Player Characters and PSI based on what a character may use.

```
CREATE TABLE CharacterHasPSI(
    cid INT NOT NULL REFERENCES PlayerCharacters(cid),
    PSId INT NOT NULL REFERENCES PSI(PSId),
    PRIMARY KEY(cid, PSId)
);
```

Dependencies: cid, PSId →
Enemy Has PSI

This table provides a link between Enemies and PSI based on what they may use.

```
CREATE TABLE EnemyHasPSI(
    eid INT NOT NULL REFERENCES Enemies(eid),
    PSId INT NOT NULL REFERENCES PSI(PSId),
    PRIMARY KEY(eid, PSId)
);
```

Dependencies: eid, PSId
Character and PSI

This view provides the user with a list of characters, the PSI they can learn, and other relevant data.

CREATE VIEW CharacterAndPSI AS

SELECT characterName, maxPsychicPoints, psiName, powerLevel
FROM PlayerCharacters, Characters, CharacterHasPSI, PSI
WHERE PlayerCharacters.cid = Characters.cid
  AND PlayerCharacters.cid = CharacterHasPSI.cid
  AND PSI.PSId = CharacterHasPSI.PSId;
Enemy and PSI

In a similar fashion as CharacterAndPSI, EnemyAndPSI created a list of Enemies that can learn PSI, the actual moves they can learn, and other related information.

CREATE VIEW EnemyAndPSI AS

    SELECT enemyName, enemyPsychicPoints, psiName, powerLevel
    FROM Enemies, EnemyHasPSI, PSI
    WHERE Enemies.eid = EnemyHasPSI.eid
    AND PSI.PSId = EnemyHasPSI.PSId;
Area Boss

This view returns Enemies that have the “Boss” status as well as the area that they guard and information about defeating them.

CREATE VIEW AreaBoss AS

    SELECT locationName, enemyName, enemyHitPoints, enemyEXPReward, enemyBountyDollars
    FROM Locations, Enemies, EnemyHabitat
    WHERE Locations.lid = EnemyHabitat.lid
    AND Enemies.eid = EnemyHabitat.eid
    AND Enemies.enemyStatus = 'Boss';
Find Person

Given a character name, this procedure returns the name of that character’s original location.

CREATE OR REPLACE FUNCTION findPerson(TEXT)
RETURNS TABLE(characterName TEXT, locationName TEXT) AS $$
DECLARE
    soughtPerson TEXT := $1;
BEGIN
    RETURN QUERY
    SELECT Characters.characterName, Locations.locationName
    FROM Characters, Locations
    WHERE Locations.lid = Characters.lid
    AND Characters.characterName = soughtPerson;
END;
$$ LANGUAGE plpgsql;

Stored Procedures
Get Character PSI

Given a character name, this procedure returns the PSI (power level included) that can be learned by said character, if any.

CREATE OR REPLACE FUNCTION getCharacterPSI(TEXT)
RETURNS TABLE(characterName TEXT, psiName TEXT, powerLevel TEXT) AS $$
DECLARE
    searchCharacter TEXT := $1;
BEGIN
    RETURN QUERY
    SELECT Characters.characterName, PSI.psiName, PSI.powerLevel
    FROM Characters, PSI, CharacterHasPSI
    WHERE Characters.cid = CharacterHasPSI.cid
    AND PSI.PSId = CharacterHasPSI.PSId
    AND Characters.characterName = searchCharacter;
END;
$$ LANGUAGE plpgsql;

Stored Procedures
Battle Victory

This trigger sets the victory flag in the Battle table to true if an enemy in said battle has its HP fall to zero.

CREATE OR REPLACE FUNCTION battleVictory() RETURNS TRIGGER AS $$
BEGIN
    IF new.enemyHitPoints <= 0 THEN
        UPDATE Battle
        SET victory = true
        WHERE Battle.eid = new.eid;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER battleVictory
AFTER UPDATE on Enemies
FOR EACH ROW
EXECUTE PROCEDURE battleVictory();
Battle Defeat
This trigger sets the victory flag in the Battle table to false if a Player Character in said battle has its HP fall to zero.

CREATE OR REPLACE FUNCTION battleDefeat()
RETURNS TRIGGER AS $$
BEGIN
  IF new.currentHitPoints <= 0 THEN
    UPDATE Battle
    SET victory = false
    WHERE Battle.cid = new.cid;
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER battleDefeat
AFTER UPDATE on PlayerCharacters
FOR EACH ROW
EXECUTE PROCEDURE battleDefeat();
Reports

This query shows the current inventory size of any given Playable Character.

```
SELECT characterName, SUM(itemQuantity) AS InventorySize
FROM PlayerCharacters
INNER JOIN Characters
ON PlayerCharacters.cid = Characters.cid
INNER JOIN Inventory
ON PlayerCharacters.cid = Inventory.cid
GROUP BY characterName;
```
Reports

This query shows a list of PSI that can only be learned by one Playable Character.

```sql
SELECT psiName, powerLevel, COUNT(characterName) AS UserCount
FROM Characters, PlayerCharacters, CharacterHasPSI, PSI
WHERE Characters.cid = PlayerCharacters.cid
    AND PlayerCharacters.cid = CharacterHasPSI.cid
    AND CharacterHasPSI.PSId = PSI.PSId
GROUP BY psiName, PowerLevel
HAVING COUNT(characterName) = 1;
```
Roles

This database, in its current state, has two roles: admin and player.

Admin role: Has administrative permissions across the entirety of the database.

```
CREATE ROLE admin;
GRANT ALL ON ALL TABLES IN SCHEMA public TO admin;
```

Player role: Has permission to manipulate parts of the database, such as Inventory, Battle, or PlayerCharacters, as they play the game.

```
CREATE ROLE player;
REVOKE ALL ON ALL TABLES IN SCHEMA public FROM player;
GRANT SELECT ON ALL TABLES IN SCHEMA public TO player;
GRANT INSERT ON Inventory, Battle TO Player;
GRANT UPDATE ON Inventory, Battle, PlayerCharacters TO Player;
```
Implementation Notes

• It should be noted that the dropChance field in the enemyLoot table is a probability ranging from zero to one, rather than a percentage.

• The OtherPSI table refers exclusively to PSI abilities that have no effect inside of battle.

• It should be noted that Poo responds differently to most food items than other characters, and thus requires his own HPRestored field in the HealingItem table.

• Accuracy of a weapon is an attribute of the weapon itself (errorRate) in EarthBound, rather than being a character attribute.
Known Issues

• There is currently no method by which to attribute a status affliction to a Player Character of Enemy. As such, references to status afflictions has simply left as part of the description of corresponding PSI.

• There is currently no way to calculate damage, and therefore no way for this system to simulate battles beyond listing participants.

• Calculations for the stats of Player Characters currently does not take equipment (weapons or armor) into account.
Future Enhancements

• Support can be added in the future for additional game mechanics, including but not limited to statis ailments, damage calculations, and item use and consumption.

• Support can be made for data from other games in this series to be included in this database.