Relational Model Normalization: Pokémon Database

1 INTRODUCTION

This document details the conversion of the Entity–Relationship (ER) diagram into a Relational Model, followed by an iterative normalization process through First, Second, and Third Normal Forms (1NF, 2NF, 3NF).

2 STAGE 1: INITIAL RELATIONAL MODEL

(AFTER ER MAPPING & REDUNDANCY CHECK)

This stage involves a direct translation of the ER entities into tables, removing derived attributes, and refining entity definitions to eliminate data redundancy across the schema.

2.1 Explanations

1. Derived Attributes Removed

- **Trainer.age**: Removed because it can be calculated from `birth date`.
- **Champion.defense_streak**: Removed because it can be derived from historical match data.

2. Entity Redundancy (Generalization of People) To ensure a "Single Source of Truth" for character data, we identify that **Gym Leaders** and **Champions** are essentially specialized **Trainers**. Storing their names in multiple tables creates redundancy and potential update anomalies.

- **Trainer:** Remains the master entity for all individuals (stores `name`, `gender`, `birth date`, `contact`).
- **Gym Leader:**
  - **Action:** Remove `leader name`.
  - **Link:** `leader id` is now linked to `Trainer.trainer id`. The Gym Leader table now strictly stores gym affiliation and professional details, referencing personal details from the Trainer table.
- **Champion:**
  - **Action:** Remove `champion name` and `region id`.
  - **Link:** `champion id` is linked to `Trainer.trainer id`. Region is inferred from the Trainer's origin or the League/Tournament context, preventing double storage.

3 STAGE 2: CONVERSION TO FIRST NORMAL FORM (1NF)

The goal of 1NF is to ensure that all data is atomic (no multi-valued or composite attributes) and that the table structure is flat.

3.1 Explanations

1. Handling Multivalued Attributes (Junction Tables)

- **Trainer.contact_info** is split into atomic fields (`email`, `phone number`).
- Multivalued attributes are removed and replaced with junction tables:
  - `PokemonSpeciesAbility`: Links species to abilities.
  - `RegisteredPokemonMove`: Links individual Pokémon to the moves they know.
  - `TypeStrength`: Links a Type to its multiple strengths.
  - `TypeWeakness`: Links a Type to its multiple weaknesses.

2. Handling Composite Attributes (Base Stats Split)

- **Issue:** The `base stats` attribute in `Pokémon Species` is a composite field (e.g., {HP: 45, Atk: 49...}). This violates 1NF atomicity.
- **Action:** `base stats` is removed and split into distinct atomic columns in the `Pokémon Species` table:
  - `base_hp`, `base_attack`, `base_defense`, `base_speed`

---

4 STAGE 3: CONVERSION TO SECOND NORMAL FORM (2NF)

A relation is in 2NF if it is in 1NF and every non-key attribute is **fully functionally dependent** on the entire primary key (no partial dependencies).

4.1 Explanations

The `GymBadge` table has a composite primary key: (`gym id`, `badge number`).

- **Issue:** The `badge name` depends only on `gym id`, not on the specific `badge number` or the trainer earning it.
- **Action:** Create a lookup table `GymBadgeName(gym id, badge name)` to store the badge name once per gym.
- **Result:** The original `GymBadge` table now only records the transaction of a trainer earning a badge: (`gym id`, `badge number`, `trainer id`, `date earned`).

---

5 STAGE 4: CONVERSION TO THIRD NORMAL FORM (3NF)

A relation is in 3NF if it is in 2NF and contains **no transitive dependencies** (non-key attributes depending on other non-key attributes).

5.1 Explanations

1. Tournament Location (Dropping Region ID)

- **Issue:** `Tournament` contains (`tournament id`, `location`, `region id`). Since a `location` (city) belongs to a specific region, `location` determines `region id`. This is a transitive dependency.
- **Action:** Drop `Tournament.region_id`.
- **Result:** The region is now derived solely from the `location`.

2. Gym Location (City Entity Integration)

- **Issue:** `Gym` contains (`gym id`, `city`, `region id`). The `city` determines the `region id`.
- **Action:**
  - Drop `Gym.city` and `Gym.region_id`.
  - Add `Gym.city_id` (Foreign Key).
- **Result:** The `Gym` table now references a dedicated `City` entity. To find the region of a gym, you join `Gym` $\rightarrow$ `City` $\rightarrow$ `Region`. This eliminates the transitive dependency within the Gym table.