Submit design document (ERD, DDL, task division)

Group 11 - Pokemon Database
Pham Minh Hieu
Cao Lam Huy

May 2025

1 Conceptual & Logical Design

a. Functional and non-functional requirements

Functional requirements:

- Store region details, including unique region names.
- Manage trainer information, including name and level, associated with a region.
- Track abilities for Pokemon, with unique ability names and descriptions.
- Record battle outcomes, linking Pokemon, trainers, and regions, with timestamps and winner identification.
- Support queries to retrieve trainers and their Pokemon by region or battle history.
- Target 89.9% database uptime.

Functional requirements:

- Ensure data consistency with unique constraints (e.g., region names, ability names) and referential integrity.
- Achieve query response times under 1 second for common operations (e.g., retrieving Pokemon by trainer).
- Support scalability for up to 5,000 trainers and 10,000 Pokemon.
- Ensure maintainability with clear schema definitions and documentation.
- Support queries to retrieve trainers and their Pokemon by region or battle history.
- Allow updates to trainer levels and Pokemon stats.

b. Entity Relationship Diagram

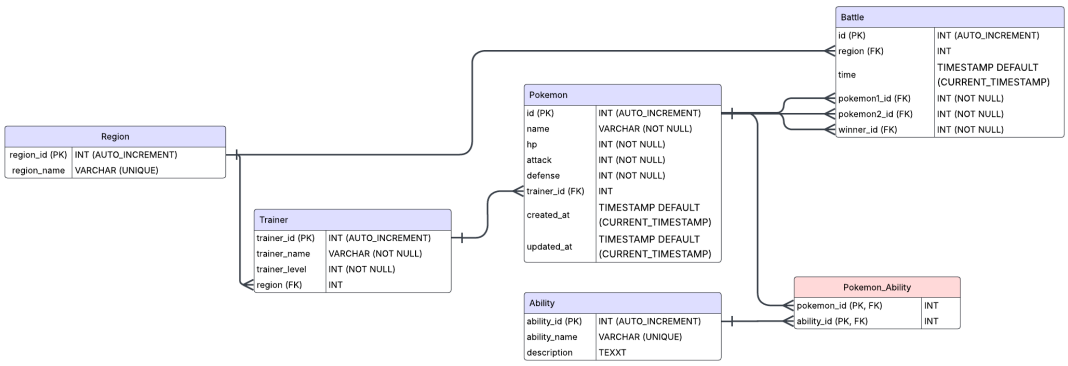

Figure 1: Pokemon entity relationship diagram

Region:

- One-to-many relationship with Trainer (one region can have many trainers)
- One-to-many relationship with Battle (one region can host many battles)

Trainer:

- Many-to-one relationship with Region (many trainers can be from one region)
- One-to-many relationship with Pokemon (one trainer can own many Pokemon)

Ability:

- Many-to-many relationship with Pokemon through Pokemon_Ability junction table

Pokemon:

- Many-to-one relationship with Trainer (many Pokemon can be owned by one trainer)
- Many-to-many relationship with Ability through Pokemon_Ability junction table
- One-to-many relationship with Battle (one Pokemon can participate in many battles)

Battle:

- Many-to-one relationship with Region (many battles can occur in one region)
- Many-to-one relationship with Pokemon for trainer1_id, trainer2_id, and winner_id

c. Normalization Proof up to Third Normal Form (3NF)

First Normal Form:

- All tables have a primary key
- No repeating groups
- All attributes are atomic

Second Normal Form:

- Non-key attributes depend fully on the primary key.
- Example: In Trainer, *trainer_level* depends on *trainer_id*, not a subset of a composite key.
- Junction table *Pokemon_Ability* uses composite key (*id*, *ability_id*) with no non-key attributes.

Third Normal Form:

- No transitive dependencies exist. Example: In Pokemon, hp, attack, etc., depend only on id, not on *trainer_id*.
- Region table has *region_name* dependent on *region_id*, with no other dependencies.
- All tables are verified to have no transitive dependencies.

2 Physical Schema Definition

a. Data Definition Language

```
-- Drop tables if they exist
DROP TABLE IF EXISTS Battle;
DROP TABLE IF EXISTS Pokemon_Ability;
DROP TABLE IF EXISTS Pokemon;
DROP TABLE IF EXISTS Ability;
DROP TABLE IF EXISTS Trainer;
DROP TABLE IF EXISTS Region;
```

```sql
-- Initialize tables
CREATE TABLE Region (
    region_id INT PRIMARY KEY AUTO_INCREMENT,
    region_name VARCHAR(100) UNIQUE NOT NULL
);

CREATE TABLE Trainer (
    trainer_id INT PRIMARY KEY AUTO_INCREMENT,
    trainer_name VARCHAR(100) NOT NULL,
    trainer_level INT NOT NULL CHECK (trainer_level >= 1),
    region_id INT,
    FOREIGN KEY (region_id) REFERENCES Region(region_id)
);

CREATE TABLE Ability (
    ability_id INT PRIMARY KEY AUTO_INCREMENT,
    ability_name VARCHAR(100) UNIQUE NOT NULL,
    description TEXT
);

CREATE TABLE Pokemon (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL UNIQUE,
    hp INT NOT NULL CHECK (hp >= 1),
    attack INT NOT NULL CHECK (attack >= 1),
    defense INT NOT NULL CHECK (defense >= 1),
    trainer_id INT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
        CURRENT_TIMESTAMP,
    FOREIGN KEY (trainer_id) REFERENCES Trainer(trainer_id)
);

CREATE TABLE Pokemon_Ability (
    pokemon_id INT,
    ability_id INT,
    PRIMARY KEY (pokemon_id, ability_id),
    FOREIGN KEY (pokemon_id) REFERENCES Pokemon(id),
    FOREIGN KEY (ability_id) REFERENCES Ability(ability_id)
);

CREATE TABLE Battle (
    battle_id INT PRIMARY KEY AUTO_INCREMENT,
    battle_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    trainer1_id INT NOT NULL,
    trainer2_id INT NOT NULL,
    winner_id INT NOT NULL,
    region_id INT,
    FOREIGN KEY (trainer1_id) REFERENCES Pokemon(id),
```

```
    FOREIGN KEY (trainer2_id) REFERENCES Pokemon(id),
    FOREIGN KEY (winner_id) REFERENCES Pokemon(id),
    FOREIGN KEY (region_id) REFERENCES Region(region_id),
    CONSTRAINT CHK_Different_Pokemon CHECK (pokemon1_id != pokemon2_id),
    CONSTRAINT CHK_Valid_Winner CHECK (winner_id IN (pokemon1_id,
        pokemon2_id))
);
```

b. Definitions of views, indexes, and any partitioning strategy

Views: Simplify retrieval of trainers, Pokémon, region, battle.

```
-- View for Pokemon with their trainers and regions
CREATE VIEW v_pokemon_details AS
SELECT
    p.id,
    p.name AS pokemon_name,
    p.hp,
    p.attack,
    p.defense,
    t.trainer_id,
    t.trainer_name,
    t.trainer_level,
    r.region_id,
    r.region_name
FROM
    Pokemon p
LEFT JOIN
    Trainer t ON p.trainer_id = t.trainer_id
LEFT JOIN
    Region r ON t.region_id = r.region_id;

-- View for Pokemon with their abilities
CREATE VIEW v_pokemon_abilities AS
SELECT
    p.id,
    p.name AS pokemon_name,
    GROUP_CONCAT(a.ability_name SEPARATOR ', ') AS abilities
FROM
    Pokemon p
LEFT JOIN
    Pokemon_Ability pa ON p.id = pa.pokemon_id
LEFT JOIN
    Ability a ON pa.ability_id = a.ability_id
GROUP BY
    p.id, p.name;

-- View for Battle details
```

```sql
CREATE VIEW v_battle_details AS
SELECT
    b.battle_id,
    b.battle_time,
    p1.name AS pokemon1_name,
    p2.name AS pokemon2_name,
    pw.name AS winner_name,
    r.region_name,
    t1.trainer_name AS trainer1_name,
    t2.trainer_name AS trainer2_name
FROM
    Battle b
JOIN
    Pokemon p1 ON b.trainer1_id = p1.id
JOIN
    Pokemon p2 ON b.trainer2_id = p2.id
JOIN
    Pokemon pw ON b.winner_id = pw.id
LEFT JOIN
    Region r ON b.region_id = r.region_id
LEFT JOIN
    Trainer t1 ON p1.trainer_id = t1.trainer_id
LEFT JOIN
    Trainer t2 ON p2.trainer_id = t2.trainer_id;

-- View for Region statistics
CREATE VIEW v_region_stats AS
SELECT
    r.region_id,
    r.region_name,
    COUNT(DISTINCT t.trainer_id) AS trainer_count,
    COUNT(DISTINCT p.id) AS pokemon_count,
    COUNT(DISTINCT b.battle_id) AS battle_count
FROM
    Region r
LEFT JOIN
    Trainer t ON r.region_id = t.region_id
LEFT JOIN
    Pokemon p ON t.trainer_id = p.trainer_id
LEFT JOIN
    Battle b ON r.region_id = b.region_id
GROUP BY
    r.region_id, r.region_name;
```

Indexes: Enhance performance for searches.

```sql
-- Indexes for Region table
CREATE INDEX idx_region_name ON Region(region_name);

-- Indexes for Trainer table
```

```sql
CREATE INDEX idx_trainer_name ON Trainer(trainer_name);
CREATE INDEX idx_trainer_region ON Trainer(region_id);

-- Indexes for Ability table
CREATE INDEX idx_ability_name ON Ability(ability_name);

-- Indexes for Pokemon table
CREATE INDEX idx_pokemon_name ON Pokemon(name);
CREATE INDEX idx_pokemon_trainer ON Pokemon(trainer_id);
CREATE INDEX idx_pokemon_stats ON Pokemon(hp, attack, defense);

-- Indexes for Battle table
CREATE INDEX idx_battle_region ON Battle(region_id);
CREATE INDEX idx_battle_pokemon1 ON Battle(trainer1_id);
CREATE INDEX idx_battle_pokemon2 ON Battle(trainer2_id);
CREATE INDEX idx_battle_winner ON Battle(winner_id);
CREATE INDEX idx_battle_time ON Battle(battle_time);
```

Strategy: Battles are time-sensitive queries, and partitioning improves performance for historical data analysis.

```sql
ALTER TABLE Battle
PARTITION BY RANGE (UNIX_TIMESTAMP(battle_time)) (
    PARTITION p0 VALUES LESS THAN (UNIX_TIMESTAMP('2024-01-01 00:00:00'))
        ,
    PARTITION p1 VALUES LESS THAN (UNIX_TIMESTAMP('2025-01-01 00:00:00'))
        ,
    PARTITION p2 VALUES LESS THAN (UNIX_TIMESTAMP('2026-01-01 00:00:00'))
        ,
    PARTITION p3 VALUES LESS THAN (UNIX_TIMESTAMP('2027-01-01 00:00:00'))
        ,
    PARTITION p4 VALUES LESS THAN MAXVALUE
);
```

3 Task Division & Project Plan

a. Task Division - Pham Minh Hieu, Cao Lam Huy

Planning and Requirements

- Clarify requirements for the Pokémon database web app – *Hieu*
- Design application structure – *Huy*
- Write `README.md` – *Hieu, Huy*

Application Setup

– *Hieu, Huy*

- Create Flask application structure
- Set up MySQL database connection
- Create database schema, data definition language, entity relation diagram
- Set up virtual environment and install dependencies

Database Models

– *Hieu*

- Implement Region model
- Implement Trainer model
- Implement Ability model
- Implement Pokémon model
- Implement `Pokemon_Ability` relationship model
- Implement Battle model

Core Features

– *Hieu*

- Implement CRUD operations for Region
- Implement CRUD operations for Trainer
- Implement CRUD operations for Ability
- Implement CRUD operations for Pokémon
- Implement CRUD operations for `Pokemon_Ability` relationships
- Implement CRUD operations for Battle

Frontend

– *Huy*

- Create base templates and layout
- Implement Region management pages
- Implement Trainer management pages

- Implement Ability management pages
- Implement Pokémon management pages
- Implement Battle management pages
- Add search functionality for all entities

Testing

– *Hieu, Huy*

- Test database connections and models
- Test CRUD operations for all entities
- Test search functionality
- Perform end-to-end testing

Deployment

– *Huy*

- Prepare application for deployment
- Create final documentation
- Package and deliver the application to the user

b. Timeline/Gantt chart

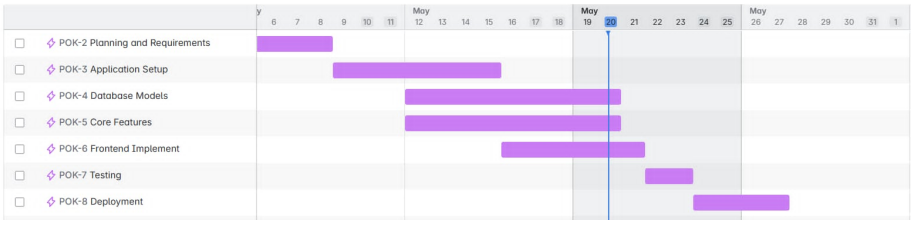

Figure 2: Timeline

4 Supporting Document

a. Database Decision Rationale

Separation of Region and Trainer Entities:

- Separating regions from trainers allows for better data organization and reduces redundancy. Multiple trainers can be from the same region, and this design allows for region-based queries and statistics.
- Including region as an attribute of Trainer would have simplified the schema but would have led to data redundancy and made it difficult to maintain region consistency.

Pokemon-Ability Many-to-Many Relationship:

- Using a junction table (*Pokemon_Ability*) allows Pokemon to have multiple abilities and abilities to be shared among multiple Pokemon, reflecting the real-world relationship in the Pokemon universe.
- Limiting Pokemon to a single ability would have simplified the schema but would not have accurately represented the domain.

Timestamps for Pokemon:

- Including *created_at* and *updated_at* timestamps allows for tracking when Pokemon are added and modified, which can be useful for auditing and sorting.
- Omitting timestamps would have simplified the schema but would have limited the ability to track changes over time.

Foreign Key Relationships:

- $ONDELETESETNULL$ vs $CASCADE$: For *trainer_id* in Pokemon and *region_id* in Trainer/Battle, we use ON DELETE SET NULL to preserve Pokemon and Trainer records when their parent entities are deleted. For battle participants and *Pokemon_Ability* relationships, we use ON DELETE CASCADE since these relationships don't make sense without both entities.
- Direct Pokemon References in Battle: The Battle table references Pokemon directly (rather than Trainers) to simplify battle recording and querying. This design choice prioritizes data integrity and query performance over complex relationship modeling.

b. Sample data loading

Data Generation

```
import random
from faker import Faker
import pandas as pd

fake = Faker()

# Generate trainers
trainers = []
for i in range(30):
    trainers.append({
        'trainer_name': fake.name(),
        'trainer_level': random.randint(5, 50),
        'region_id': random.randint(1, 8)
    })

# Generate Pokemon
pokemon = []
for i in range(100):
    pokemon.append({
        'name': f"{fake.word().capitalize()}{fake.word().capitalize()}",
        'hp': random.randint(30, 150),
        'attack': random.randint(20, 130),
        'defense': random.randint(20, 130),
        'trainer_id': random.randint(1, 30) if random.random() > 0.1 else None
    })

# Export to CSV
pd.DataFrame(trainers).to_csv('trainers.csv', index=False)
pd.DataFrame(pokemon).to_csv('pokemon.csv', index=False)
```

CSV Import

```
LOAD DATA INFILE 'trainers.csv'
INTO TABLE Trainer
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(trainer_name, trainer_level, region_id);

LOAD DATA INFILE 'pokemon.csv'
INTO TABLE Pokemon
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
```

```
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(name, hp, attack, defense, trainer_id);
```

Data Integrity Considerations

- Foreign Key Constraints: All data loading will respect foreign key constraints, loading parent tables before child tables.
- Unique Constraints: Scripts will check for and handle potential duplicate entries, especially for Pokemon and ability names.
- Realistic Relationships:

  - Most trainers will have 1-6 Pokemon (standard team size in Pokemon games)
  - Pokemon will have 1-4 abilities
  - Battles will only occur between Pokemon that exist in the database

- Data Validation:

  - Pokemon stats will be within reasonable ranges
  - Trainer levels will follow game progression logic
  - Battle winners will be one of the participating Pokemon