INTRODUCCIÓN

Este documento presenta la documentación del proyecto relacionado con la implementación de bases de datos para gestionar información de distintos dominios.

Las bases de datos elegida es **pokemondb**.

SITUACIÓN PROBLEMÁTICA

Cada uno de los dominios presenta una serie de problemáticas que pueden ser resueltas con la implementación de bases de datos:

- **Pokemondb**: La gestión de datos relacionados con Pokémon suele ser extensa y compleja, siendo necesario gestionar no solo las características de un Pokémon sino también, sus evoluciones, movimientos, aprendizajes, entre otras.

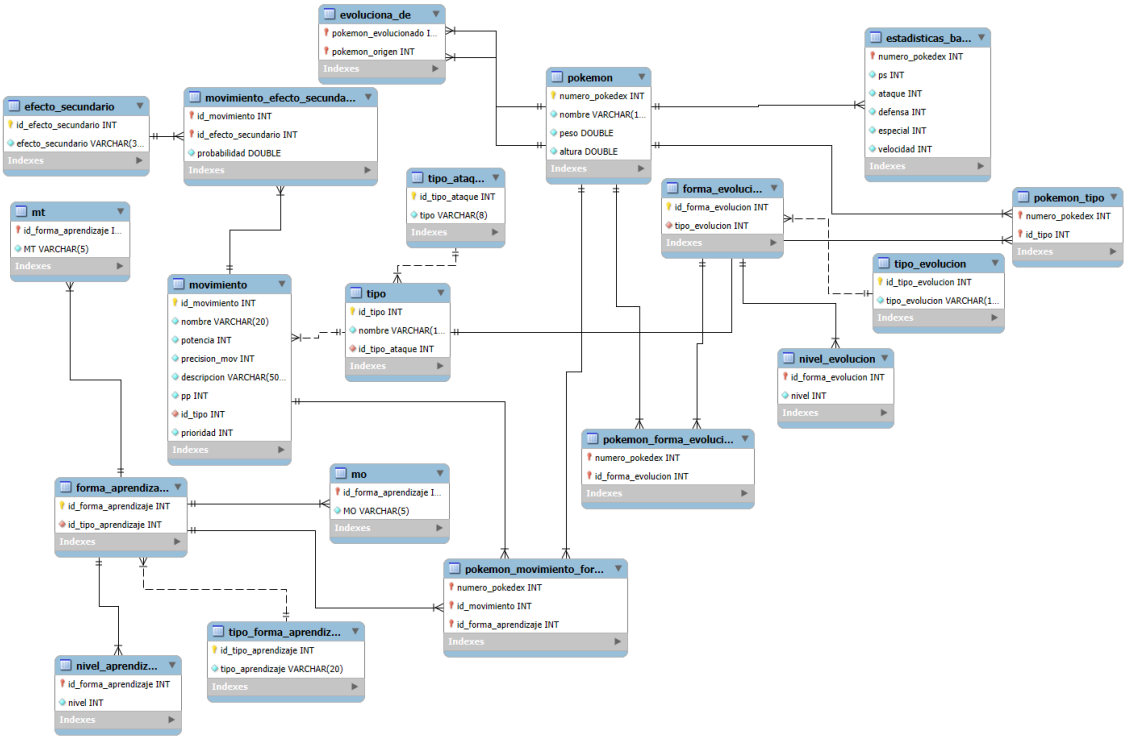
MODELO DE NEGOCIO

El proyecto busca desarrollar un sistema de bases de datos que permita almacenar, gestionar y consultar información relevante sobre la base seleccionada. Además, se busca optimizar el acceso a los datos, facilitar la generación de informes y garantizar la integridad de la información almacenada. El sistema cubrirá necesidades en áreas como:

- **Pokemondb:** Orientada a desarrolladores de videojuegos, investigadores y fanáticos de la franquicia que requieren acceso rápido y estructurado a información detallada sobre Pokémon.

DIAGRAMA ENTIDAD-RELACIÓN (DER)

POKEMON



LISTA DE TABLAS

POKEMON

Tabla	Campo	Tipo de Dato	Nulo	Clave
efecto_secundario	id_efecto_secundario	int	NO	PRI
	efecto_secundario	varchar(30)	NO	
estadísticas_base	numero_pokedex	int	NO	PRI
	ps	int	NO	
	ataque	int	NO	
	defensa	int	NO	
	especial	int	NO	
	velocidad	int	NO	
evoluciona_de	pokemon_evolucionado	int	NO	PRI
	pokemon_origen	int	NO	PRI
forma_aprendizaje	id_forma_aprendizaje	int	NO	PRI
	id_tipo_aprendizaje	int	NO	MUL
forma_evolucion	id_forma_evolucion	int	NO	PRI
	tipo_evolucion	int	NO	MUL
mo	id_forma_aprendizaje	int	NO	
	MO	varchar(5)	NO	
movimiento	id_movimiento	int	NO	PRI
	nombre	varchar(20)	NO	

	potencia	int	NO	
	precision_mov	int	NO	
	descripcion	varchar(500)	NO	
	pp	int	NO	
	id_tipo	int	NO	MUL
	prioridad	int	NO	
movimiento_efecto_secundario	id_movimiento	int	NO	PRI
	id_efecto_secundario	int	NO	PRI
	probabilidad	double	NO	
mt	id_forma_aprendizaje	int	NO	PRI
	MT	varchar(5)	NO	
nivel_aprendizaje	id_forma_aprendizaje	int	NO	PRI
	nivel	int	NO	
nivel_evolucion	id_forma_evolucion	int	NO	PRI
	nivel	int	NO	
pokemon	numero_pokedex	int	NO	PRI
	nombre	varchar(15)	NO	
	peso	double	NO	
	altura	double	NO	
pokemon_forma_evolucion	numero_pokedex	int	NO	PRI
	id_forma_evolucion	int	NO	PRI
pokemon_movimiento_forma	numero_pokedex	int	NO	PRI
	id_movimiento	int	NO	PRI
	id_forma_aprendizaje	int	NO	PRI
pokemon_tipo	numero_pokedex	int	NO	PRI
	id_movimiento	int	NO	PRI
	id_forma_aprendizaje	int	NO	PRI
tipo	id_tipo	int	NO	PRI
	nombre	varchar(10)	NO	
	id_tipo_ataque	int	NO	MUL
tipo_ataque	id_tipo_ataque	int	NO	PRI
	tipo	varchar(8)	NO	
tipo_evolucion	id_tipo_evolucion	int	NO	PRI
	tipo_evolucion	varchar(15)	NO	
tipo_forma_aprendizaje	id_tipo_aprendizaje	int	NO	PRI
	tipo_aprendizaje	varchar(20)	NO	

IMPLEMENTACIÓN SQL: VISTAS, FUNCIONES, PROCEDIMIENTOS ALMACENADOS Y TRIGGERS EN POKEMONDB

Este documento describe las funciones, vistas, procedimientos almacenados y triggers definidos en el código SQL proporcionado.

CREACIÓN DE VISTAS

Las vistas son consultas guardadas que se tratan como tablas virtuales. Facilitan la consulta de datos complejos al abstraer las uniones y selecciones necesarias.

1. **vista_pokemon_tipos:**
    a. Descripción: Muestra el número de Pokédex, el nombre del Pokémon y su tipo.
    b. Objetivo: Simplificar la consulta de los tipos de cada Pokémon.
    c. Tablas Afectadas:
        i. pokemon: Se utiliza para obtener el numero_pokedex y el nombre.
        ii. pokemon_tipo: Se utiliza como tabla de enlace para relacionar Pokémon con sus tipos a través de numero_pokedex y id_tipo.
        iii. tipo: Se utiliza para obtener el nombre del tipo a través de id_tipo.
    d. Cómo afecta: Permite consultar de manera sencilla la información combinada de Pokémon y sus tipos sin necesidad de realizar las uniones explícitamente en cada consulta.
    e. Código:

```
CREATE VIEW vista_pokemon_tipos AS
SELECT p.numero_pokedex, p.nombre, t.nombre AS tipo
FROM pokemon p
JOIN pokemon_tipo pt ON p.numero_pokedex = pt.numero_pokedex
JOIN tipo t ON pt.id_tipo = t.id_tipo;
SELECT *
FROM vista_pokemon_tipos;
```

2. **vista_movimientos_efectos:**
    a. Descripción: Muestra el nombre del movimiento y su efecto secundario (si lo tiene). También incluye la potencia del movimiento.
    b. Objetivo: Proporcionar una visión rápida de los movimientos y sus efectos.
    c. Tablas Afectadas:
        i. movimiento: Se utiliza para obtener el nombre y la potencia del movimiento a través de id_movimiento.
        ii. movimiento_efecto_secundario: Se utiliza como tabla de enlace para relacionar movimientos con sus efectos secundarios a través de id_movimiento y Id_efecto_secundario.
        iii. efecto_secundario: Se utiliza para obtener la descripción del efecto_secundario a través de id_efecto_secundario.
    d. Cómo afecta: Facilita la consulta de los movimientos junto con sus efectos secundarios y potencia, combinando información de varias tablas.
    e. Código:

```
CREATE VIEW vista_movimientos_efectos AS

SELECT m.nombre, m.potencia, es.efecto_secundario

FROM movimiento m

JOIN movimiento_efecto_secundario mes ON m.id_movimiento =
mes.id_movimiento

JOIN efecto_secundario es ON mes.id_efecto_secundario =
es.id_efecto_secundario;

SELECT *

FROM vista_movimientos_efectos;
```

3. **vista_evoluciones_pokemon:**
    a. Descripción: Muestra el número de Pokédex, el nombre del Pokémon y su tipo y nivel de evolución.
    b. Objetivo: Simplificar la consulta de la información de evolución de los Pokémon.
    c. Tablas Afectadas:
        i. pokemon: Se utiliza para obtener el número_pokedex y el nombre.
        ii. pokemon_forma_evolucion: Se utiliza como tabla de enlace entre Pokémon y sus formas de evolución a través de numero_pokedex y id_forma_evolucion.
        iii. forma_evolucion: Se utiliza para obtener el tipo_evolucion a través de id_forma_evolucion.
        iv. tipo_evolucion: Se utiliza para obtener la descripción del tipo_evolucion a través de id_tipo_evolucion.

v. nivel_evolucion: Se utiliza para obtener el nivel de la evolución a través de id_forma_evolucion.

d. Cómo afecta: Permite consultar de forma sencilla la información relacionada con la evolución de los Pokémon, incluyendo el tipo y nivel requeridos.

e. Código:

```
CREATE VIEW vista_evoluciones_pokemon AS

SELECT p.numero_pokedex, p.nombre, te.tipo_evolucion, ne.nivel

FROM pokemon p

JOIN pokemon_forma_evolucion pfe ON p.numero_pokedex = pfe.numero_pokedex

JOIN forma_evolucion fe ON pfe.id_forma_evolucion = fe.id_forma_evolucion

JOIN tipo_evolucion te ON fe.tipo_evolucion = te.id_tipo_evolucion

JOIN nivel_evolucion ne ON fe.id_forma_evolucion = ne.id_forma_evolucion;
```

4. **viista_pokemon_movimientos:**
    a. Descripción: Muestra los movimientos que posee un pokemon, siendo este el nombre, el nivel de daño y si tiene un efecto secundario.
    b. Objetivo: Simplificar la consulta de los movimientos que tiene en su haber un pokemon
    c. Tablas Afectadas:
        i. pokemon: Se utiliza para obtener el número_pokedex y el nombre.
        ii. pokemon_movimiento_forma: Se utiliza como tabla de enlace entre Pokémon y sus formas de movimiento a través de numero_pokedex y id_movimiento.
        iii. movimiento: Se utiliza para obtener el moviento que caracteriza al pokemon
    d. Cómo afecta: Permite consultar de forma sencilla la información relacionada los movimientos que tiene el pokemon.
    e. Código:

```
CREATE OR REPLACE VIEW vista_pokemon_movimientos AS

SELECT p.numero_pokedex, p.nombre AS nombre_pokemon,

m.nombre AS nombre_movimiento, m.potencia, m.precision_mov, m.descripcion

FROM pokemon p

JOIN pokemon_movimiento_forma pmf ON p.numero_pokedex =
pmf.numero_pokedex

JOIN movimiento m ON pmf.id_movimiento = m.id_movimiento;
```

CREACIÓN DE FUNCIONES

Las funciones son bloques de código SQL que realizan una tarea específica y devuelven un valor.

1. **peso_promedio_pokemon():**
    a. Descripción: Devuelve el peso promedio de todos los Pokémon registrados en la tabla pokemon.
    b. Objetivo: Calcular y obtener el peso promedio de los Pokémon.
    c. Tablas Afectadas:
        i. pokemon: Se utiliza para calcular el promedio de la columna peso.
    d. Cómo afecta: Proporciona un valor agregado calculado a partir de los datos de la tabla pokemon.
    e. Código:

```
DELIMITER //

CREATE FUNCTION peso_promedio_pokemon()

RETURNS DOUBLE DETERMINISTIC

BEGIN

RETURN (SELECT AVG(peso) FROM pokemon);

END //

DELIMITER ;

SELECT peso_promedio_pokemon()
```

2. **nombre_pokemon**:
    a. Descripción: Recibe un número de Pokédex como entrada y devuelve el nombre del Pokémon correspondiente de la tabla pokemon.
    b. Objetivo: Obtener el nombre de un Pokémon específico dado su número de Pokédex.
    c. Tablas Afectadas:
        i. pokemon: Se consulta para obtener el nombre basado en el numero_pokedex.
    d. Cómo afecta: Permite buscar el nombre de un Pokémon individualmente utilizando su identificador único.
    e. Código:

```
DELIMITER //

CREATE FUNCTION nombre_pokemon(numero INT)

RETURNS VARCHAR(100) DETERMINISTIC

BEGIN

RETURN (SELECT nombre FROM pokemon WHERE numero_pokedex = numero);

END //

DELIMITER ;

SELECT nombre_pokemon(25);
```

3. **potencia_movimiento(id_mov INT):**
    a. Descripción: Recibe el ID de un movimiento como entrada y devuelve su potencia desde la tabla movimiento.
    b. Objetivo: Obtener la potencia de un movimiento específico dado su ID.
    c. Tablas Afectadas:
        i. movimiento: Se consulta para obtener la potencia basada en el id_movimiento.
    d. Cómo afecta: Permite consultar la potencia de un movimiento individualmente.
    e. Código:

```
DELIMITER //

CREATE FUNCTION potencia_movimiento(id_mov INT)

RETURNS INT DETERMINISTIC

BEGIN

RETURN (SELECT potencia FROM movimiento WHERE id_movimiento = id_mov
LIMIT 1);

END //

DELIMITER ;


SELECT potencia_movimiento(5);
```

4. **tipo_ataque_movimiento(id_mov INT):**
    a. Descripción: Recibe el ID de un movimiento como entrada y devuelve el nombre del tipo al que pertenece ese movimiento.
    b. Objetivo: Obtener el tipo de un movimiento específico.
    c. Tablas Afectadas:
        i. movimiento: Se utiliza para obtener el id_tipo del movimiento basado en el id_movimiento.
        ii. tipo: Se utiliza para obtener el nombre del tipo basado en el id_tipo obtenido de la tabla movimiento.
    d. Cómo afecta: Permite conocer el tipo de un movimiento específico combinando información de dos tablas.
    e. Código:

```
DELIMITER //

CREATE FUNCTION tipo_ataque_movimiento(id_mov INT)

RETURNS VARCHAR(50) DETERMINISTIC

BEGIN

RETURN (SELECT t.nombre FROM movimiento m

            JOIN tipo t ON m.id_tipo = t.id_tipo

            WHERE m.id_movimiento = id_mov LIMIT 1);

END //

DELIMITER ;


SELECT tipo_ataque_movimiento(5);
```

5. **evolucion_pokemon(numero INT):**
    a. Descripción: Recibe el número de Pokédex de un Pokémon y devuelve su tipo de evolución.
    b. Objetivo: Obtener el tipo de evolución de un Pokémon específico.
    c. Tablas Afectadas:
        i. pokemon: Se utiliza para identificar el Pokémon por su numero_pokedex.
        ii. pokemon_forma_evolucion: Se utiliza para encontrar la forma de evolución asociada al Pokémon.
        iii. forma_evolucion: Se utiliza para obtener el id_tipo_evolucion de la forma de evolución.
        iv. tipo_evolucion: Se utiliza para obtener la descripción del tipo_evolucion basado en el id_tipo_evolucion.

v. nivel_evolucion: Aunque se une, en la selección final no se utiliza directamente, pero es parte del camino para llegar al tipo de evolución.

d. Cómo afecta: Permite consultar el tipo de evolución de un Pokémon específico.
e. Código:

```
DELIMITER //

CREATE FUNCTION evolucion_pokemon(numero INT)

RETURNS VARCHAR(100) DETERMINISTIC

BEGIN

    RETURN (SELECT te.tipo_evolucion FROM pokemon p

            JOIN pokemon_forma_evolucion pfe ON p.numero_pokedex =
pfe.numero_pokedex

            JOIN forma_evolucion fe ON pfe.id_forma_evolucion =
fe.id_forma_evolucion

            JOIN tipo_evolucion te ON fe.tipo_evolucion =
te.id_tipo_evolucion

            JOIN nivel_evolucion ne ON fe.id_forma_evolucion =
ne.id_forma_evolucion

            WHERE p.numero_pokedex = numero LIMIT 1);

END //

DELIMITER ;


SELECT evolucion_pokemon (1)
```

STORE PROCEDURES

Los procedimientos almacenados son conjuntos de instrucciones SQL guardadas que se pueden ejecutar varias veces. Pueden recibir parámetros de entrada y realizar operaciones en la base de datos.

1. **insertar_pokemon(IN nombre VARCHAR(100), IN peso, DOUBLE, IN altura DOUBLE):**
    a. Descripción: Inserta un nuevo registro en la tabla pokemon con los valores proporcionados para el número de Pokédex, nombre, peso y altura.
    b. Objetivo: Permitir la adición de nuevos Pokémon a la base de datos.
    c. Tablas Afectadas:
        i. pokemon: Se insertan nuevos registros en esta tabla.

d. Cómo afecta: Modifica la tabla pokemon añadiendo una nueva fila con la información del nuevo Pokémon.
e. Código:

```
DELIMITER //
CREATE PROCEDURE insertar_nuevo_pokemon(
    IN nombre VARCHAR(100),
    IN peso DOUBLE,
    IN altura DOUBLE
)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        -- Si hay error, hacer rollback
        ROLLBACK;
    END;
    START TRANSACTION;
    -- Insertar en las diferentes tablas
    INSERT INTO pokemon (nombre, peso, altura)
    VALUES (nombre, peso, altura);
    COMMIT;
END //
DELIMITER ;
```

2. **actualizar_potencia_movimiento(IN id_mov INT, IN nueva_potencia INT):**
    a. Descripción: Actualiza la potencia de un movimiento específico en la tabla movimiento utilizando el ID del movimiento y la nueva potencia proporcionada.
    b. Objetivo: Permitir la modificación de la potencia de los movimientos existentes.
    c. Tablas Afectadas:
        i. movimiento: Se actualiza la columna potencia para el registro con el id_movimiento especificado.
    d. Cómo afecta: Modifica la tabla movimiento actualizando la potencia de un movimiento.
    e. Código:

```
DELIMITER //
CREATE PROCEDURE actualizar_potencia_movimiento(
IN id_mov INT,
IN nueva_potencia INT
)
BEGIN
UPDATE movimiento
SET potencia = nueva_potencia
WHERE id_movimiento = id_mov;
END //
DELIMITER ;
```

3. **eliminar_pokemon(IN numero INT):**
    a. Descripción: Elimina un registro de la tabla pokemon basándose en el número de Pokédex proporcionado.
    b. Objetivo: Permitir la eliminación de Pokémon de la base de datos.
    c. Tablas Afectadas:
        i. pokemon: Se eliminan registros de esta tabla.
    d. Cómo afecta: Modifica la tabla pokemon eliminando la fila correspondiente al número de Pokédex especificado.
    e. Código:

```
DELIMITER //
CREATE PROCEDURE eliminar_pokemon(IN numero INT)
BEGIN
DELETE FROM pokemon WHERE numero_pokedex = numero;
END //
DELIMITER ;
```

TRIGGERS

Los triggers son bloques de código SQL que se ejecutan automáticamente en respuesta a ciertos eventos (como INSERT, UPDATE o DELETE) que ocurren en una tabla específica.

1. **validar_pokemon_insert:**
    a. Descripción: Se ejecuta antes de que se intente insertar un nuevo registro en la tabla pokemon. Verifica que el peso y la altura del nuevo Pokémon sean mayores que 0. Si no lo son, la inserción se cancela y se genera un error.
    b. Objetivo: Asegurar la integridad de los datos en la tabla pokemon al evitar la inserción de Pokémon con peso o altura no válidos.
    c. Tablas Afectadas:
        i. pokemon: Este trigger opera sobre la tabla pokemon durante la operación de INSERT.
    d. Cómo afecta: Impide que se inserten registros en la tabla pokemon si los valores de peso o altura no cumplen con la condición especificada, manteniendo la calidad de los datos.
    e. Código:

```
DELIMITER //

CREATE TRIGGER validar_pokemon_insert

BEFORE INSERT ON pokemon

FOR EACH ROW

BEGIN

    IF NEW.peso <= 0 OR NEW.altura <= 0 THEN

        SIGNAL SQLSTATE '45000'

        SET MESSAGE_TEXT = 'Error: El peso y la altura del Pokémon deben
ser mayores a 0.';

    END IF;

END //

DELIMITER ;
```

2. **validar_evolucion_pokemon:**
    a. Descripción: Se ejecuta antes de que se intente insertar un nuevo registro en la tabla pokemon_forma_evolucion. Verifica que el numero_pokedex que se intenta registrar en la tabla de evoluciones exista previamente en la tabla pokemon. Si el Pokémon no existe, la inserción se cancela y se genera un error.
    b. Objetivo: Mantener la coherencia entre las tablas pokemon y pokemon_forma_evolucion, asegurando que no se registren evoluciones para Pokémon inexistentes.
    c. Tablas Afectadas:
        i. pokemon_forma_evolucion: Este trigger opera sobre esta tabla durante la operación de INSERT.
        ii. pokemon: Se consulta esta tabla para verificar la existencia del numero_pokedex.
    d. Cómo afecta: Evita la creación de registros de evolución para Pokémon que no están registrados en la tabla principal de Pokémon, manteniendo la integridad relacional entre las tablas.
    e. Código:

```
DELIMITER //

CREATE TRIGGER validar_evolucion_pokemon

BEFORE INSERT ON pokemon_forma_evolucion

FOR EACH ROW

BEGIN

    IF NOT EXISTS (SELECT 1 FROM pokemon WHERE numero_pokedex =

NEW.numero_pokedex) THEN

        SIGNAL SQLSTATE '45000'

        SET MESSAGE_TEXT = 'Error: No se puede registrar una evolución
para un Pokémon que

no existe.';

END IF;

END //

DELIMITER ;
```

POKEDEX

La pokedex permite combinar las diferentes fuentes de información para resumirla en una sola tabla.

```
-- POKEDEX --


CREATE VIEW vista_pokedex_completa AS

SELECT

    p.numero_pokedex,

    p.nombre AS nombre_pokemon,

    tipo1.nombre AS tipo_I,

    tipo2.nombre AS tipo_II,

    p.peso,

    p.altura,

    e.ataque AS ATK,

    e.defensa AS DEF,

    e.velocidad AS VEL,

    m.nombre AS movimiento,

    m.potencia

FROM pokemon p

LEFT JOIN estadisticas_base e ON p.numero_pokedex = e.numero_pokedex

LEFT JOIN pokemon_tipo pt1 ON p.numero_pokedex = pt1.numero_pokedex

LEFT JOIN tipo tipo1 ON pt1.id_tipo = tipo1.id_tipo

LEFT JOIN pokemon_tipo pt2 ON p.numero_pokedex = pt2.numero_pokedex

LEFT JOIN tipo tipo2 ON pt2.id_tipo = tipo2.id_tipo

LEFT JOIN pokemon_movimiento_forma pm ON p.numero_pokedex =
pm.numero_pokedex

LEFT JOIN movimiento m ON pm.id_movimiento = m.id_movimiento;


SELECT * FROM vista_pokedex_completa;
```

CONCLUSIÓN DEL PROYECTO

El desarrollo de esta base de datos relacional tuvo como propósito principal modelar de forma estructurada y funcional el universo Pokémon, integrando entidades clave como Pokémon, tipos, movimientos, estadísticas y relaciones entre ellos. A través de una arquitectura normalizada, se logró representar la complejidad del mundo Pokémon de manera eficiente, facilitando consultas avanzadas, mantenibilidad y expansión futura.

Durante el proyecto, se diseñaron e implementaron múltiples componentes esenciales:

- Tablas relacionales que reflejan con precisión las características individuales de cada Pokémon (peso, altura, estadísticas base).
- Relaciones muchos a muchos, como los tipos y movimientos de cada Pokémon, resueltas mediante tablas intermedias.
- Vistas que permiten visualizar datos complejos como una Pokédex completa o la lista de movimientos asociados a cada criatura.
- Procedimientos almacenados que encapsulan operaciones como la inserción segura de nuevos Pokémon, utilizando transacciones y control de errores mediante ROLLBACK.
- Triggers y funciones que añaden lógica adicional al momento de insertar, actualizar o validar datos.

Este proyecto no solo permitió aplicar conceptos fundamentales de bases de datos como claves primarias/foráneas, normalización, transacciones y vistas, sino también demostrar cómo una estructura correctamente diseñada puede servir tanto a fines analíticos como operativos.

En síntesis, la base de datos creada proporciona una plataforma robusta y escalable para consultar, mantener y analizar datos del mundo Pokémon, y representa un caso práctico aplicable a cualquier sistema de información con entidades interrelacionadas.