

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:

- **Descripción:** Muestra el número de Pokédex, el nombre del Pokémon y su tipo.
- **Objetivo:** Simplificar la consulta de los tipos de cada Pokémon.
- **Tablas Afectadas:**
 - pokemon: Se utiliza para obtener el numero_pokedex y el nombre.
 - pokemon_tipo: Se utiliza como tabla de enlace para relacionar Pokémon con sus tipos a través de numero_pokedex y id_tipo.
 - tipo: Se utiliza para obtener el nombre del tipo a través de id_tipo.
- **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.
- **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:

- **Descripción:** Muestra el nombre del movimiento y su efecto secundario (si lo tiene). También incluye la potencia del movimiento.
- **Objetivo:** Proporcionar una visión rápida de los movimientos y sus efectos.
- **Tablas Afectadas:**
 - movimiento: Se utiliza para obtener el nombre y la potencia del movimiento a través de id_movimiento.
 - 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.

- efecto_secundario: Se utiliza para obtener la descripción del efecto_secundario a través de id_efecto_secundario.
- **Cómo afecta:** Facilita la consulta de los movimientos junto con sus efectos secundarios y potencia, combinando información de varias tablas.
- **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:

- **Descripción:** Muestra el número de Pokédex, el nombre del Pokémon y su tipo y nivel de evolución.
- **Objetivo:** Simplificar la consulta de la información de evolución de los Pokémon.
- **Tablas Afectadas:**
 - pokemon: Se utiliza para obtener el numero_pokedex y el nombre.
 - 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.
 - forma_evolucion: Se utiliza para obtener el tipo_evolucion a través de id_forma_evolucion.
 - tipo_evolucion: Se utiliza para obtener la descripción del tipo_evolucion a través de id_tipo_evolucion.
 - nivel_evolucion: Se utiliza para obtener el nivel de la evolución a través de id_forma_evolucion.
- **Cómo afecta:** Permite consultar de forma sencilla la información relacionada con la evolución de los Pokémon, incluyendo el tipo y nivel requeridos.
- **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;
```

```
SELECT *  
FROM vista_evoluciones_pokemon;
```

Creación de Funciones

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

1. **peso_promedio_pokemon():**

- **Descripción:** Devuelve el peso promedio de todos los Pokémon registrados en la tabla pokemon.
- **Objetivo:** Calcular y obtener el peso promedio de los Pokémon.
- **Tablas Afectadas:**
 - **pokemon:** Se utiliza para calcular el promedio de la columna peso.
- **Cómo afecta:** Proporciona un valor agregado calculado a partir de los datos de la tabla pokemon.

- **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(numero INT):**

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

- **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):**

- **Descripción:** Recibe el ID de un movimiento como entrada y devuelve su potencia desde la tabla movimiento.
- **Objetivo:** Obtener la potencia de un movimiento específico dado su ID.
- **Tablas Afectadas:**
 - movimiento: Se consulta para obtener la potencia basada en el id_movimiento.
- **Cómo afecta:** Permite consultar la potencia de un movimiento individualmente.
- **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):**

- **Descripción:** Recibe el ID de un movimiento como entrada y devuelve el nombre del tipo al que pertenece ese movimiento.
- **Objetivo:** Obtener el tipo de un movimiento específico.
- **Tablas Afectadas:**
 - movimiento: Se utiliza para obtener el id_tipo del movimiento basado en el id_movimiento.
 - tipo: Se utiliza para obtener el nombre del tipo basado en el id_tipo obtenido de la tabla movimiento.
- **Cómo afecta:** Permite conocer el tipo de un movimiento específico combinando información de dos tablas.
- **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):**

- **Descripción:** Recibe el número de Pokédex de un Pokémon y devuelve su tipo de evolución.
- **Objetivo:** Obtener el tipo de evolución de un Pokémon específico.
- **Tablas Afectadas:**
 - **pokemon:** Se utiliza para identificar el Pokémon por su `numero_pokedex`.
 - **pokemon_forma_evolucion:** Se utiliza para encontrar la forma de evolución asociada al Pokémon.
 - **forma_evolucion:** Se utiliza para obtener el `id_tipo_evolucion` de la forma de evolución.
 - **tipo_evolucion:** Se utiliza para obtener la descripción del `tipo_evolucion` basado en el `id_tipo_evolucion`.
 - **nivel_evolucion:** Aunque se une, en la selección final no se utiliza directamente, pero es parte del camino para llegar al tipo de evolución.
- **Cómo afecta:** Permite consultar el tipo de evolución de un Pokémon específico.
- **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 numero INT, IN nombre VARCHAR(100), IN peso DOUBLE, IN altura DOUBLE):**

- **Descripción:** Inserta un nuevo registro en la tabla pokemon con los valores proporcionados para el número de Pokédex, nombre, peso y altura.
- **Objetivo:** Permitir la adición de nuevos Pokémon a la base de datos.
- **Tablas Afectadas:**
 - pokemon: Se insertan nuevos registros en esta tabla.
- **Cómo afecta:** Modifica la tabla pokemon añadiendo una nueva fila con la información del nuevo Pokémon.
- **Código:**

```
DELIMITER //
CREATE PROCEDURE insertar_pokemon(
    IN numero INT,
    IN nombre VARCHAR(100),
    IN peso DOUBLE,
    IN altura DOUBLE
)
BEGIN
    INSERT INTO pokemon (numero_pokedex, nombre, peso, altura)
    VALUES (numero, nombre, peso, altura);
END //
DELIMITER ;
```

2. **actualizar_potencia_movimiento(IN id_mov INT, IN nueva_potencia INT):**

- **Descripción:** Actualiza la potencia de un movimiento específico en la tabla movimiento utilizando el ID del movimiento y la nueva potencia proporcionada.
- **Objetivo:** Permitir la modificación de la potencia de los movimientos existentes.
- **Tablas Afectadas:**
 - movimiento: Se actualiza la columna potencia para el registro con el id_movimiento especificado.
- **Cómo afecta:** Modifica la tabla movimiento actualizando la potencia de un movimiento.
- **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):**

- **Descripción:** Elimina un registro de la tabla pokemon basándose en el número de Pokédex proporcionado.
- **Objetivo:** Permitir la eliminación de Pokémon de la base de datos.
- **Tablas Afectadas:**
 - pokemon: Se eliminan registros de esta tabla.
- **Cómo afecta:** Modifica la tabla pokemon eliminando la fila correspondiente al número de Pokédex especificado.

- **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:**

- **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.
- **Objetivo:** Asegurar la integridad de los datos en la tabla pokemon al evitar la inserción de Pokémon con peso o altura no válidos.
- **Tablas Afectadas:**
 - pokemon: Este trigger opera sobre la tabla pokemon durante la operación de INSERT.
- **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.

- **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:**

- **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.
- **Objetivo:** Mantener la coherencia entre las tablas `pokemon` y `pokemon_forma_evolucion`, asegurando que no se registren evoluciones para Pokémon inexistentes.
- **Tablas Afectadas:**
 - `pokemon_forma_evolucion`: Este trigger opera sobre esta tabla durante la operación de INSERT.
 - `pokemon`: Se consulta esta tabla para verificar la existencia del `numero_pokedex`.
- **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.
- **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 ;

```



```
END IF;  
END //  
DELIMITER;
```