EJERCICIO 3

Objetivo del ejercicio:

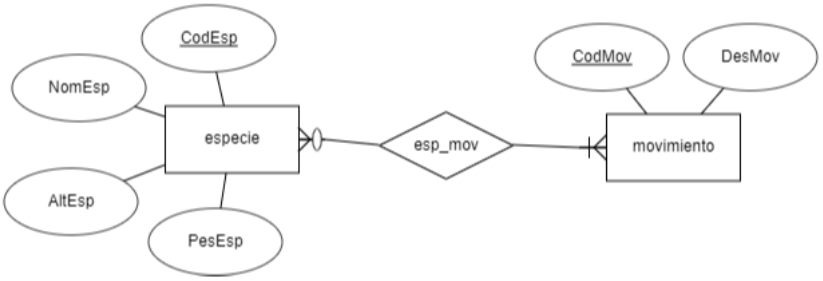
1. *Aprender el tipo de correspondencia muchos a muchos (m a n).*
2. *Repasar las cardinalidades mínimas.*
3. *Concepto de PK compuesta.*
4. *Primera Forma Normal (1FN).*
5. *En SQL: La cláusula ORDER BY de la sentencia SELECT y el tipo de datos DECIMAL.*

Se quiere diseñar una base de datos para almacenar las diferentes especies de Pokémon. Para cada especie almacenaremos su código, nombre, altura y peso. Cada especie tiene diferentes movimientos, de los que almacenaremos su código y descripción. Un movimiento lo pueden tener muchas especies y viceversa. Cuando se almacena una especie se debe indicar al menos un movimiento. Podemos tener almacenado un movimiento y que aún no pertenezca a ninguna especie.

Se pide:

1. Modelar la base de datos. Para ello haremos:

   a. Diseño Conceptual de Datos utilizando un Diagrama o Modelo Entidad-Relación. Lo hacemos en papel y lo pasamos a la Herramienta CASE ERD Plus.



   b. Diseño Lógico de Datos utilizando un Diagrama de Estructura de datos (DED). Lo hacemos en papel y lo pasamos a la Herramienta CASE MySql Workbench. En este apartado también vamos a poner el Diagrama Referencial que genera ERD Plus a partir del Modelo Entidad-Relación. Recuerda que el Diseño Lógico de Datos es hacer el modelo relacional y para ello podemos hacer un DED o un Diagrama Referencial.

      En el Diagrama Entidad-Relación anterior hemos detectado un tipo de correspondencia M:N (m a n, o también llamada muchos a muchos). Cuando ocurre esto, en el Diseño Lógico de Datos la **relación** que une las dos entidades implicadas **se convierte en una nueva entidad**. Como esta entidad se convertirá en tabla, debemos ponerle un nombre significativo. La nueva entidad llevará las FKs correspondientes a las PKs de las entidades que relacionaba. A veces, como veremos en otros ejercicios, llevará también otros atributos.

ELECCIÓN de la PK de la NUEVA ENTIDAD:
Esta nueva entidad al igual que todas, debe tener una PK. Por defecto, ERD Plus y MySQL Workbench ponen las dos FKs **JUNTAS** como PK compuesta (en este caso, clave primaria compuesta por dos atributos).

Podemos dejar la PK compuesta de las FKs, siempre que comprobemos que la combinación de los valores que pueden llegar a tomar estas FKs, NO tiene sentido que se repita. Si sí se pueden repetir , o bien no estamos seguro, hay que añadir un nuevo atributo que designaremos como PK.

NOTA:
1ª Forma Normal (1FN). No puede existir más de un valor en un atributo.
En caso de haber elegido mal el tipo de correspondencia, y haber elegido 1:N (en cualquiera de los dos sentidos) estaríamos incumpliendo la **1FN**. Puesto que, para poder almacenar la información, tendríamos que almacenar en un atributo más de un valor. Esto haría que nuestro **modelo incumpliera las normas para ser un Modelo Relacional**.

Diagrama Referencial

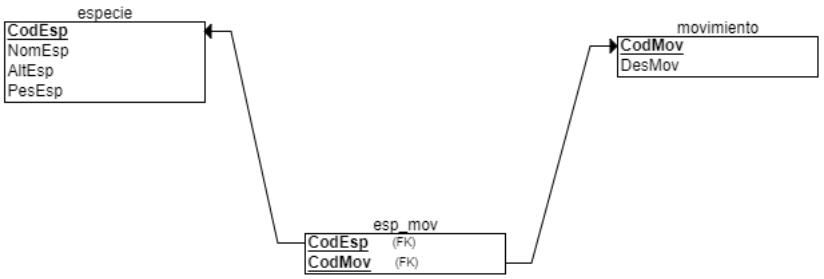
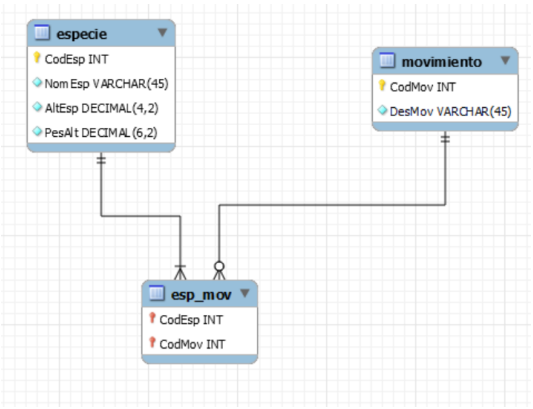
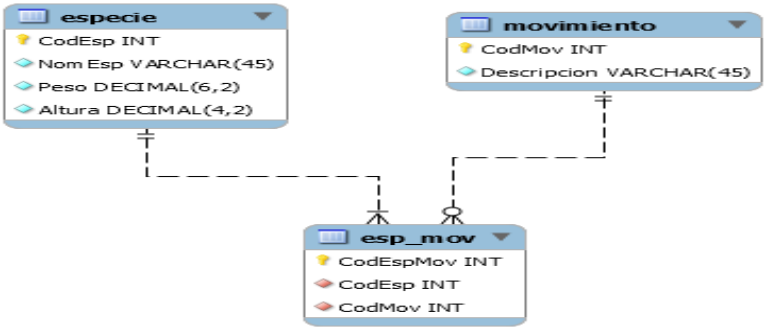


Diagrama de Estructura de Datos (DED)



En este caso la combinación de CodEsp y CodMov, no tiene sentido que se repitan nunca. Por lo tanto, podemos dejar las dos juntas como PK (esto se llama PK compuesta).

Pero si queremos (NO ES OBLIGATORIO), Podemos poner un PK diferente y dejar CodEsp y CodMov solo como FKs.



c. Diseño Físico de Datos. Creamos la base de datos y las tablas en SQL.

```
CREATE DATABASE EJERCICIO3;
USE EJERCICIO3;

CREATE TABLE especie
(
  CodEsp INT NOT NULL,
  NomEsp VARCHAR(45) NOT NULL,
  AltEsp DECIMAL(4,2) NOT NULL,
  PesEsp DECIMAL(6,2) NOT NULL,
  PRIMARY KEY (CodEsp)
);

CREATE TABLE movimiento
(
  CodMov INT NOT NULL,
  DesMov VARCHAR(45) NOT NULL,
  PRIMARY KEY (CodMov)
);

CREATE TABLE esp_mov
(
  CodEsp INT NOT NULL,
  CodMov INT NOT NULL,
  PRIMARY KEY (CodEsp, CodMov),
  FOREIGN KEY (CodEsp) REFERENCES especie(CodEsp),
  FOREIGN KEY (CodMov) REFERENCES movimiento(CodMov)
```

```
);
```

2. Insertar datos desde phpmyadmin.

```
INSERT INTO  especie (CodEsp, NomEsp, AltEsp, PesEsp)
VALUES  (1, 'Dragón', 10.22, 50.60),
        (2, 'Pikachu', 1.30, 7.50);

INSERT INTO movimiento (CodMov,DesMov)
VALUES (1,'Salto Salvaje'), (2, 'Movimiento2'),(3, 'Movimiento3'),(4, 'Movimiento4'),(5, 'Movimiento5'),(6,'Nuevo Movimiento');


INSERT INTO esp_mov (CodEsp,CodMov)
VALUES  (1,1), (1,2),(1,3),
        (2,1),(2,4),(2,5);
```

3. Realizar las siguientes consultas en SQL:

- Muestra todas las filas y todos los campos de las tablas.

  ```
  SELECT * FROM especie;

  SELECT * FROM movimiento;

  SELECT * FROM esp_mov;
  ```

- Muestra algunos campos de las tablas.

  ```
  SELECT NomEsp,AltEsp,PesEsp

  FROM especie;

  SELECT DesmOV

  FROM movimiento;
  ```

- Para cada especie, mostrar todos los movimientos que tiene almacenado. Muestra primero toda la información y posteriormente muestra solo el nombre de la especie y el nombre del movimiento. Ordenar ascendentemente por nombre de la especie.

  ```
  SELECT  *

  FROM (especie e JOIN esp_mov em ON (e.CodEsp=em.CodEsp));


  SELECT *

  FROM (especie e JOIN esp_mov em ON (e.CodEsp=em.CodEsp))
  ```

JOIN movimiento m ON (em.CodMov=m.CodMov);

SELECT NomEsp, Desmov

FROM (especie e JOIN esp_mov em ON (e.CodEsp=em.CodEsp)) JOIN movimiento m ON (em.CodMov=m.CodMov)

ORDER BY NomEsp;

También podemos poner:

SELECT NomEsp, DesMov

FROM (especie e JOIN esp_mov em ON (e.CodEsp=em.CodEsp)) JOIN movimiento m ON (em.CodMov=m.CodMov)

ORDER BY 1;

Dónde 1 indica que ordenamos por la primera expresión del SELECT.

Si no se especifica nada en la cláusula ORDER BY, el orden por defecto es ascendente, es decir de menor a mayor.

SELECT NomEsp, Desmov

FROM (especie e JOIN esp_mov em ON (e.CodEsp=em.CodEsp)) JOIN movimiento m ON (em.CodMov=m.CodMov)

ORDER BY 1 ASC;

Observa que el resultado es el mismo.

Si queremos ordenar descendentemente debemos especificar DESC en la cláusula ORDER BY.

SELECT NomEsp, Desmov

FROM (especie e JOIN esp_mov em ON (e.CodEsp=em.CodEsp)) JOIN movimiento m ON (em.CodMov=m.CodMov)

ORDER BY NomEsp DESC;

También podemos ordenar por varias expresiones del SELECT, por ejemplo

SELECT NomEsp, Desmov

FROM (especie e JOIN esp_mov em ON (e.CodEsp=em.CodEsp)) JOIN movimiento m ON (em.CodMov=m.CodMov)

ORDER BY 1 ASC, 2 DESC;

- Mostrar para cada movimiento las especies que lo tienen. Muestra primero toda la información y posteriormente muestra solo el nombre del movimiento y nombre de la especie que lo tiene. ¿Qué ocurre con los movimientos que aún no lo tienen ninguna especie?

  SELECT *

  FROM (especie e JOIN esp_mov em ON (e.CodEsp=em.CodEsp));

  SELECT *

  FROM (especie e JOIN esp_mov em ON (e.CodEsp=em.CodEsp)) JOIN movimiento m ON (em.CodMov=m.CodMov);

  SELECT DesMov, NomEsp

  FROM (especie e JOIN esp_mov em ON (e.CodEsp=em.CodEsp)) JOIN movimiento m ON (em.CodMov=m.CodMov);

  Lo que ocurre con los movimientos que aún no lo tienen ninguna especie es que no aparecen. Para eso tenemos los diferentes tipos de JOIN. En este caso hemos puesto RIGHT JOIN para que aparezca la información que no sale de la tabla que hay a nuestra derecha según miramos la sentencia.

  SELECT DesMov, NomEsp

  FROM (especie e JOIN esp_mov em ON (e.CodEsp=em.CodEsp)) RIGHT JOIN movimiento m ON (em.CodMov=m.CodMov);

- Para el movimiento Salto salvaje, mostrar las especies que lo tienen.

  SELECT DesMov, NomEsp

  FROM (especie e JOIN esp_mov em ON (e.CodEsp=em.CodEsp)) JOIN movimiento m ON (em.CodMov=m.CodMov)

  WHERE DesMov='Salto Salvaje';

- Para la especie Dragón, mostrar todos los movimientos que tiene almacenados.

  SELECT NomEsp, Desmov

```
FROM (especie e JOIN esp_mov em ON (e.CodEsp=em.CodEsp)) JOIN movimiento m
ON (em.CodMov=m.CodMov)

WHERE NomEsp='Dragón';
```