**Taller SQL (RELACIONALES)**

🧩Parte 1: Crear las tablas

Crea 3 tablas:

1.Productos: Debe tener ID (entero, clave primaria), Nombre (texto) y Precio (número decimal).

2.Categorías: Debe tener ID (entero, clave primaria) y Nombre (texto).

3.Productos\_Categorías: Relaciona productos con categorías. Debe tener las columnas necesarias y usar claves foráneas.

DROP TABLE IF EXISTS Productos\_Categorias;

DROP TABLE IF EXISTS Productos;

DROP TABLE IF EXISTS Categorias;

CREATE TABLE Categorias (

ID INT PRIMARY KEY,

Nombre VARCHAR(100)

);

CREATE TABLE Productos (

ID INT PRIMARY KEY,

Nombre VARCHAR(100),

Precio DECIMAL(10, 2)

);

CREATE TABLE Productos\_Categorias (

ProductoID INT,

CategoriaID INT,

PRIMARY KEY (ProductoID, CategoriaID),

FOREIGN KEY (ProductoID) REFERENCES Productos(ID),

FOREIGN KEY (CategoriaID) REFERENCES Categorias(ID)

);

🧪Parte 2: Insertar Datos Agrega al menos:

2 categorías: por ejemplo “Bebidas” y “Dulces”.

3 productos: con nombres y precios diferentes.

Relaciona los productos con las categorías en la tabla intermedia.

👉Usa INSERT INTO, decide tú los valores que usarás.

INSERT INTO Categorias (ID, Nombre) VALUES

(1, 'Bebidas'),

(2, 'Dulces');

INSERT INTO Productos (ID, Nombre, Precio) VALUES

(1, 'Jugo de Naranja', 2000),

(2, 'Chocolate', 800),

(3, 'Galletas', 1200);

INSERT INTO Productos\_Categorias (ProductoID, CategoriaID) VALUES

(1, 1), -- Jugo de Naranja -> Bebidas

(2, 2), -- Chocolate -> Dulces

(3, 2); -- Galletas -> Dulces

🔍Parte 3: Consultas

Escribe consultas SQL para responder:

1.¿Cuáles son todos los productos disponibles y sus precios?

SELECT Nombre, Precio

FROM Productos;

2.¿Qué productos cuestan más de $10?

SELECT Nombre, Precio

FROM Productos

WHERE Precio > 10;

3.¿Qué productos pertenecen a cada categoría? (JOIN)

SELECT p.Nombre AS Producto, c.Nombre AS Categoria

FROM Productos p

JOIN Productos\_Categorias pc ON p.ID = pc.ProductoID

JOIN Categorias c ON c.ID = pc.CategoriaID;

4.Muestra todas las categorías, incluso si no tienen productos asignados. (LEFT JOIN)

SELECT c.Nombre AS Categoria, p.Nombre AS Producto

FROM Categorias c

LEFT JOIN Productos\_Categorias pc ON c.ID = pc.CategoriaID

LEFT JOIN Productos p ON p.ID = pc.ProductoID;

✏️Parte 4: Preguntas cortas

1.¿Para qué se usan las claves foráneas?

Las claves foráneas o foreign keys se usan para relacionar tablas entre sí, para asegurar que los datos estén conectados correctamente. Manteniendo la integridad referencial, en donde los valores de una tabla (como ProductoID) correspondan a valores válidos en otra tabla (como Productos).

Por ejemplo: si ProductoID = 1 en la tabla Productos\_Categorias, debe existir un producto con ID = 1 en la tabla Productos.

2.¿Qué diferencia hay entre INNER JOIN y LEFT JOIN?

INNER JOIN: nos muestra los registros que coinciden en ambas tablas, si una categoría no tiene productos, no se mostrará. Mientras que, LEFT JOIN: nos muestra todos los registros de la tabla de la izquierda, aunque no tengan coincidencias en la tabla de la derecha, si una categoría no tiene productos, igual aparece, pero con valores NULL para el producto.

**// Taller: MongoDB con JSON de Pokémon**

**// 📚 Por: Yurani Hurtado Lozano**

🔹Parte 1: Consultas Básicas

1. 🕵️‍♂️Encuentra todos los Pokémon de tipo “Electric”.

db.pokemones.find({ Type1: "Electric" })

2. 📊Muestra solo los nombres y el ataque de los Pokémon con más de 100 de ataque.

db.pokemones.find(

{ Attack: { $gt: 100 } },

{ \_id: 0, Name: 1, Attack: 1 }

)

3. 🧱Encuentra los Pokémon cuya defensa esté entre 80 y 100.

db.pokemones.find(

{ Defense: { $gte: 80, $lte: 100 } }

)

🔹Parte 2: Agregaciones

4. 📈Muestra el promedio de ataque de los Pokémon por tipo (Type1).

db.pokemones.aggregate([

{

$group: {

\_id: "$Type1",

promedioAtaque: { $avg: "$Attack" }

}

}

])

5.🔝Encuentra el Pokémon con más HP de cada tipo.

db.pokemones.aggregate([

{

$sort: { Type1: 1, Type2: 1, HP: -1 } // Ordena por ambos tipos y luego HP

},

{

$group: {

\_id: { type1: "$Type1", type2: "$Type2" },

nombre: { $first: "$Name" },

HP: { $first: "$HP" }

}

}

])

6. 🚀Muestra los 5 Pokémon más rápidos

db.pokemones.find()

.sort({ Speed: -1 }) // Orden descendente

.limit(5)

🔹Parte 3: Combinaciones de $match, $group y $sort

7. 🌊Muestra el promedio de ataque de los Pokémon tipo “Water” ordenado de mayor a menor.

db.pokemones.aggregate([

{

$match: { Type1: "Water" }

},

{

$group: {

\_id: "$Type1",

promedioAtaque: { $avg: "$Attack" }

}

},

{

$sort: { promedioAtaque: -1 }

}

])

-------- Type2

db.pokemones.aggregate([

{

$match: { Type2: "Water" }

},

{

$group: {

\_id: "$Type2",

promedioAtaque: { $avg: "$Attack" }

}

},

{

$sort: { promedioAtaque: -1 }

}

])

8.🧬Encuentra el Pokémon con más ataque por generación y ordénalos de mayor a menor.

db.pokemones.aggregate([

{

$sort: { Generation: 1, Attack: -1 }

},

{

$group: {

\_id: "$Generation",

nombre: { $first: "$Name" },

ataque: { $first: "$Attack" }

}

},

{

$sort: { ataque: -1 }

}

])

Parte 4: Indexación y Rendimiento

9. ⚙️Crea un índice en el campo Type1.

db.pokemones.createIndex({ Type1: 1 })

10. 📊Usa explain() para analizar el rendimiento de una búsqueda.

db.pokemones.find({ Type1: "Fire" }).explain("executionStats")

11.🧩Crea un índice compuesto en Type1 y Speed, y analiza una búsqueda.

db.pokemones.createIndex({ Type1: 1, Speed: -1 })

db.pokemones.find({ Type1: "Electric" }).sort({ Speed: -1 }).explain("executionStats")