Taller Práctico: De SQL a MongoDB

1. Tu Laboratorio (Copia y Pega)

No pierdas tiempo instalando cosas. Crea una carpeta y dentro guarda este archivo como docker-compose.yaml. Esto te dará una base de datos MongoDB lista para usar.

YAML

```
version: '3.8'
services:
  # Tu base de datos
  mongodb:
    image: mongo:6.0
    container_name: mi_mongo_taller
    ports:
      - "27017:27017"
    environment:
      - MONGO_INITDB_ROOT_USERNAME=admin
      - MONGO_INITDB_ROOT_PASSWORD=password123
    volumes:
      -./data:/data/db  # Guarda tus datos aquí
```

Arráncalo en tu terminal:

Bash

```
docker-compose up -d
```

Script de Conexión (Python)

Usa este script base para todos los ejemplos siguientes.

Python

```
from pymongo import MongoClient
from bson import ObjectId # Importante para buscar por ID
from pprint import pprint # Para imprimir bonito
```

```
# 1. Conexión (Equivalente al string de conexión JDBC/ODBC)
client = MongoClient("mongodb://admin:password123@localhost:27017/")

# 2. Seleccionar Base de Datos (Equivalente a: USE tiendadb;)
# Nota: En Mongo, si no existe, se crea sola al insertar el primer dato.
db = client['tiendadb']

# 3. Seleccionar Colección (Equivalente a una Tabla)
users_col = db['usuarios']
products_col = db['productos']
orders_col = db['pedidos']
```

---

2. El Cambio Mental: Tablas vs. Documentos

Imagina un **Usuario** con su **Carrito de Compras**.

- **En SQL (Relacional):** Necesitas 2 tablas obligatoriamente. Una tabla Usuarios y una tabla CarritoItems conectadas por una llave foránea (user_id).
- **En MongoDB (Documental):** Es **UN solo documento**. El carrito es una lista (array) dentro del usuario.

Visualización:

JSON

```
{
  "_id": ObjectId("..."),
  "nombre": "Juan Perez",
  "email": "juan@email.com",
  "carrito":
}
```

---

3. Capítulo 1: INSERT (Crear Datos)

Vamos a crear un catálogo de productos.

Caso 1: Insertar un solo registro

Acción	SQL	MongoDB (Python)

Código	INSERT INTO productos (nombre, precio, stock) VALUES ('Laptop', 1000, 10);	db.productos.insert_one({ "nombre": "Laptop", "precio": 1000, "stock": 10 })

Caso 2: Insertar múltiples registros

Acción	**SQL**	**MongoDB (Python)**
Código	INSERT INTO productos (nombre,...) VALUES ('Mouse',...), ('Teclado',...);	db.productos.insert_many()

Ejemplo Práctico en Python:

Python

```
# Limpiamos la colección para empezar de cero (como TRUNCATE TABLE)
products_col.delete_many({})

lista_productos = [
  { "sku": "A1", "nombre": "Laptop Gamer", "precio": 1500, "categoria": "compu", "tags": ["oferta", "nuevo"] },
  { "sku": "A2", "nombre": "Mouse Óptico", "precio": 20, "categoria": "accesorios", "tags": ["básico"] },
  { "sku": "A3", "nombre": "Monitor 4K", "precio": 300, "categoria": "compu", "tags": ["oferta"] }
]

resultado = products_col.insert_many(lista_productos)
print(f"Insertados {len(resultado.inserted_ids)} documentos.")
```

---

4. Capítulo 2: SELECT (Leer Datos)

Aquí es donde más tiempo pasarás. Fíjate cómo la sintaxis de llaves {} reemplaza al WHERE.

Caso 1: Select Simple (Todos)

- **SQL:** SELECT * FROM productos;
- **Mongo:** list(db.productos.find())

Caso 2: Filtrar por igualdad (WHERE)

- **SQL:** SELECT * FROM productos WHERE categoria = 'compu';
- **Mongo:**
  Python
  query = { "categoria": "compu" }
  resultados = products_col.find(query)

Caso 3: Operadores Lógicos (AND implícito)

- **SQL:** SELECT * FROM productos WHERE categoria = 'compu' AND precio = 1500;
- **Mongo:** (Simplemente pones más campos en el diccionario, es un AND automático).
  Python
  query = {
  "categoria": "compu",
  "precio": 1500
  }

Caso 4: Comparaciones (Mayor que, Menor que)

En SQL usas símbolos (>, <). En Mongo usas operadores con signo de dólar $.

- $gt (Greater Than - Mayor que)
- $lt (Less Than - Menor que)
- $gte (Mayor o igual)
- $lte (Menor o igual)
- $ne (No igual - Distinto)
- **SQL:** SELECT * FROM productos WHERE precio > 100;
- **Mongo:**
  Python
  # "Donde precio sea $gt (mayor que) 100"
  query = { "precio": { "$gt": 100 } }

Caso 5: El operador OR

- **SQL:** SELECT * FROM productos WHERE precio > 1000 OR categoria = 'accesorios';
- **Mongo:** (Usamos una lista para las opciones).
  Python
  query = {
  "$or": [
  { "precio": { "$gt": 1000 } },
  { "categoria": "accesorios" }
  ]
  }

Caso 6: Buscar dentro de una lista (IN)

Supongamos que el producto tiene tags: ["oferta", "nuevo"].

- **SQL:** (Difícil, requiere tablas extra o búsquedas lentas tipo LIKE).
- **Mongo:** ¡Es nativo! Si buscas "oferta", Mongo busca automáticamente dentro del array.
  Python
  ```
  # Busca productos que tengan la etiqueta "oferta" en su lista de tags
  query = { "tags": "oferta" }
  products_col.find(query)
  ```

---

5. Capítulo 3: UPDATE (Actualizar Datos)

Aquí MongoDB brilla. En lugar de sobrescribir toda la fila, usamos **Operadores de Actualización**.
¡Importante! Si en Mongo haces un update sin operadores (pasas solo un objeto), **reemplazas todo el documento**. Para modificar solo un campo, usa siempre $set.

Caso 1: Modificar un campo simple

- **SQL:** UPDATE productos SET precio = 1400 WHERE sku = 'A1';
- **Mongo:** Usamos $set.
  Python
  ```
  filtro = { "sku": "A1" }
  cambio = { "$set": { "precio": 1400 } }

  products_col.update_one(filtro, cambio)
  ```

Caso 2: Incrementar un valor (Contadores)

Imagina que vendiste uno y quieres bajar el stock o subir las visitas.

- **SQL:** UPDATE productos SET stock = stock - 1 WHERE sku = 'A1';
- **Mongo:** Usamos $inc (Incrementar). Para restar, usa números negativos.
  Python
  ```
  # ¡Súper eficiente! No necesitas leer el valor anterior.
  products_col.update_one(
     { "sku": "A1" },
     { "$inc": { "stock": -1 } }
  )
  ```

Caso 3: Poner un campo nuevo (Migración al vuelo)

¿Decidiste que ahora los productos tienen "garantía"? En SQL tendrías que hacer ALTER TABLE. En Mongo solo haces el update.

- **Mongo:**
  Python
  # Agrega el campo 'garantia' a TODOS los productos de categoria 'compu'
  products_col.update_many(
  { "categoria": "compu" },
  { "$set": { "garantia": "2 años" } }
)

---

6. Capítulo 4: Arrays - El Carrito de Compras (Crucial)

Aquí aprenderemos a manejar relaciones 1-a-Muchos sin crear tablas extra.
Creemos un usuario con carrito vacío.

Python

users_col.insert_one({
"_id": 1,
"nombre": "Maria",
"carrito": # Array vacío
})

Acción 1: Agregar item al carrito (INSERT en tabla relacionada)

- **SQL:** INSERT INTO carrito_items (user_id, producto, cant) VALUES (1, 'Mouse', 1);
- **Mongo:** Usamos $push (Empujar al array).
  Python
  users_col.update_one(
  { "_id": 1 },
  { "$push": { "carrito": { "producto": "Mouse", "cantidad": 1 } } }
)

Acción 2: Quitar item del carrito (DELETE en tabla relacionada)

- **SQL:** DELETE FROM carrito_items WHERE user_id=1 AND producto='Mouse';
- **Mongo:** Usamos $pull (Jalar/Sacar del array).
  Python
  users_col.update_one(

```
    { "_id": 1 },
    { "$pull": { "carrito": { "producto": "Mouse" } } }
)
```

Acción 3: Modificar item DENTRO del carrito (UPDATE complejo)

María ya tiene el Mouse en el carrito, pero quiere cambiar la cantidad de 1 a 2.
Esto es avanzado. Necesitamos encontrar a María, encontrar el Mouse en su array, y actualizar solo ese elemento.

- **Mongo:** Usamos el operador posicional $. El $ representa el índice del elemento que encontró el filtro.
  Python
  ```
  users_col.update_one(
      # 1. Filtro: Usuario 1 Y que tenga 'Mouse' en el carrito
      { "_id": 1, "carrito.producto": "Mouse" },

      # 2. Update: Incrementa la cantidad del elemento encontrado ($)
      { "$inc": { "carrito.$.cantidad": 1 } }
  )
  ```

---

7. Capítulo 5: Relaciones (JOIN vs $lookup)

A veces no podemos meter todo en un documento (ej. un Pedido y un Usuario). El pedido debe tener los datos del usuario, pero si el usuario cambia de email, queremos que se actualice.

Estructura:

1. Colección usuarios: { "_id": 1, "nombre": "Maria" }
2. Colección pedidos: { "_id": 100, "user_id": 1, "total": 50 }

El JOIN en MongoDB: $lookup

Mongo no hace JOINs en el find(). Los hace en una tubería de agregación (aggregate). Piensa en aggregate como una máquina de pasos.

- **SQL:**
  SQL
  ```
  SELECT * FROM pedidos
  JOIN usuarios ON pedidos.user_id = usuarios._id
  ```

- **Mongo (Python):**
  Python
  ```
  pipeline =
  ```

```
resultados = list(orders_col.aggregate(pipeline))
pprint(resultados)
```

Resultado: Mongo te devolverá el pedido y un campo datos_usuario que será una **lista** con la información encontrada (porque podría haber coincidido con varios, aunque aquí sea uno).

---

8. Resumen de Traducción Rápida

Operación SQL	Operador MongoDB	Ejemplo Python
WHERE col = val	{ "col": val }	find({"edad": 18})
WHERE col > val	{ "col": { "$gt": val } }	find({"edad": {"$gt": 18}})
WHERE col IN (1,2)	{ "col": { "$in": [1, 2] } }	find({"id": {"$in": [1, 2]}})
ORDER BY col DESC	sort("col", -1)	find().sort("edad", -1)
LIMIT 5	limit(5)	find().limit(5)
COUNT(*)	count_documents({})	col.count_documents({})
UPDATE... SET...	{ "$set": {... } }	update_one(..., {"$set":...})
DELETE	delete_one / delete_many	delete_one({"_id": 1})

Ejercicio Final Propuesto

Para graduarte de este "Nivel 1", intenta hacer este script:

1. Crea un producto "Iphone 15" con stock 10.
2. Crea un usuario "Carlos".
3. Agrega el "Iphone 15" al carrito de "Carlos" (usando $push).
4. Resta 1 al stock del producto original (usando $inc).
5. Imprime el documento de "Carlos" para ver su carrito.