Taller de CRUD: UPDATE en MongoDB

Requisitos previos

Antes de comenzar, asegúrate de tener **uno de los siguientes entornos de ejecución**:

- MongoDB instalado localmente
- MongoDB Compass (interfaz gráfica)
- Docker instalado para contenedor MongoDB
- Mongo Shell (mongosh) o conexión mediante terminal

Crear la base de datos LabTestDB y la colección clientes

A) Si usas la terminal con mongosh:

1. Abre el terminal y ejecuta: `mongosh`
2. Crea o cambia a la base de datos: `use LabTestDB`
3. Verifica que estás en la base correcta: `db`

B) Con Docker (entorno desechable para pruebas):

1. Levanta un contenedor de MongoDB:

```
docker run -d -p 27017:27017 --name mongo-taller mongo
```

2. Accede al contenedor: `docker exec -it mongo-taller mongosh`
3. Luego crea la base: `use LabTestDB`

B) Importar los datos del archivo .json

Puedes utilizar el archivo LabTestDB.clientes.json para precargar datos de ejemplo. En el entorno local, haz lo siguiente:

1. Guarda el archivo como clientes.json
2. Ejecuta desde la terminal:

```
mongoimport --db LabTestDB --collection clientes --file clientes.json --
jsonArray
```

Asegúrate de estar en el mismo directorio donde está el archivo LabTestDB.clientes.json.

Verifica que la colección clientes no está vacía (para los ejercicios de UPDATE)

```
db.clientes.find()
```

UPDATE en MongoDB

Parte 1: updateOne()

1. Cambiar el nombre de un cliente específico

```
db.clientes.updateOne(
 { _id: "CL1" },
 { $set: { nombre: "William Sun" } }
)
// Cambia el nombre del cliente con ID "CL1" usando el operador $set.
```

2. Activar las notificaciones para un cliente

```
db.clientes.updateOne(
 { email: "john.booth@example.com" },
 { $set: { "preferencias.notificaciones": true } }
)
// Actualiza un campo anidado con notación de puntos.
```

3. Incrementar puntos de fidelidad

```
db.clientes.updateOne(
 { _id: "CL2" },
 { $inc: { puntos_fidelidad: 50 } }
)
// Usa $inc para incrementar un campo numérico.
```

4. Establecer nuevo campo si no existe (upsert)

```
db.clientes.updateOne(
 { _id: "CL999" },
 { $set: { nombre: "Nuevo Cliente", puntos_fidelidad: 10 } },
 { upsert: true }
)
// Crea un documento nuevo si no se encuentra uno que cumpla el filtro.
```

5. Agregar un nuevo valor a array ($push)

```
db.clientes.updateOne(
 { _id: "CL3" },
 { $push: { "preferencias.categorias_favoritas": "juguetes" } }
)
// Añade un valor al final del array.
```

6. Usar collation para comparación sin sensibilidad a mayúsculas

```
db.clientes.updateOne(
 { nombre: "william suh" },
 { $set: { estatus: "verificado" } },
 { collation: { locale: "en", strength: 2 } }
)
// Permite buscar sin distinguir entre mayúsculas/minúsculas.
```

7. Remover un campo ($unset)

```
db.clientes.updateOne(
 { _id: "CL4" },
 { $unset: { telefono: "" } }
)
// Elimina el campo 'telefono' del documento.
```

8. Usar let para pasar variables

```
db.clientes.updateOne(
 { $expr: { $eq: ["$estatus", "$$estadoBuscado"] } },
 { $set: { puntos_fidelidad: 0 } },
 { let: { estadoBuscado: "inactivo" } }
)
// Usa una variable dentro del $expr para filtrar.
```

9. Aplicar hint para forzar uso de un índice

```
db.clientes.updateOne(
 { email: "william.suh@example.com" },
 { $set: { puntos_fidelidad: 1000 } },
 { hint: { email: 1 } }
)
// Usa un índice específico para la consulta.
```

10. Ordenar antes de aplicar la actualización

```
db.clientes.updateOne(
 { estatus: "activo" },
 { $set: { estatus: "verificado" } },
 { sort: { puntos_fidelidad: -1 } }
)
// Aplica la actualización al cliente activo con más puntos.
```

Parte 2: updateMany()

11. Cambiar el estatus a "suspendido" para todos los clientes inactivos

```
db.clientes.updateMany(
 { estatus: "inactivo" },
 { $set: { estatus: "suspendido" } }
)
// Actualiza múltiples documentos.
```

12. Añadir nueva categoría a todos los clientes

```
db.clientes.updateMany(
 {},
 { $addToSet: { "preferencias.categorias_favoritas": "hogar" } }
)
// Añade un valor sin duplicados al array.
```

13. Establecer puntos mínimos a 50

```
db.clientes.updateMany(
 { puntos_fidelidad: { $lt: 50 } },
 { $set: { puntos_fidelidad: 50 } }
)
// Usa un operador de comparación.
```

14. Marcar clientes con notificaciones activas

```
db.clientes.updateMany(
 { "preferencias.notificaciones": true },
 { $set: { notificacion_activa: true } }
)
// Crea un nuevo campo booleano.
```

15. Reemplazar teléfonos nulos por uno genérico

```
db.clientes.updateMany(
 { telefono: null },
 { $set: { telefono: "+52-1-000-0000" } }
)
// Normaliza datos faltantes.
```

16. Reemplazar emails de cierto dominio

```
db.clientes.updateMany(
 { email: /@hotmail\.com$/ },
 { $set: { email: "actualizar@dominio.com" } }
)
// Usa expresión regular para filtrar.
```

17. Quitar un campo innecesario

```
db.clientes.updateMany(
 {},
 { $unset: { comentarios: "" } }
)
// Limpieza de campos extras.
```

18. Incrementar puntos según condición

```
db.clientes.updateMany(
 { puntos_fidelidad: { $gte: 900 } },
 { $inc: { puntos_fidelidad: 100 } }
)
// Bonus para clientes leales.
```

19. Usar writeConcern con nivel de seguridad

```
db.clientes.updateMany(
 { ciudad: "Puebla" },
 { $set: { actualizado: true } },
 { writeConcern: { w: "majority", wtimeout: 5000 } }
)
// Requiere confirmación de mayoría de nodos en replica set.
```

20. Aplicar hint para índice compuesto

```
db.clientes.updateMany(
 { ciudad: "Guadalajara", estatus: "activo" },
 { $set: { zona_prioritaria: true } },
 { hint: { ciudad: 1, estatus: 1 } }
)
// Mejora el rendimiento usando índice sugerido.
```

Parte 3: replaceOne()

21. Reemplazar documento completo

```
db.clientes.replaceOne(
 { _id: "CL5" },
 {
   _id: "CL5",
   nombre: "Cliente Reemplazado",
   email: "reemplazo@example.com"
 }
)
// Sustituye completamente el documento, preserva solo el _id.
```

22. Usar upsert en reemplazo

```
db.clientes.replaceOne(
 { _id: "CL999" },
 { _id: "CL999", nombre: "Cliente Nuevo" },
 { upsert: true }
)
// Crea nuevo documento si no existe.
```

23. Reemplazar con orden (sort)

```
db.clientes.replaceOne(
 { estatus: "activo" },
 { nombre: "Ordenado", email: "ordenado@example.com" },
 { sort: { puntos_fidelidad: -1 } }
)
// Reemplaza el más fiel de los activos.
```

24. Aplicar collation en búsqueda de reemplazo

```
db.clientes.replaceOne(
 { nombre: "william suh" },
 { nombre: "Nombre Normalizado", email: "normalizado@example.com" },
 { collation: { locale: "en", strength: 2 } }
)
// Permite búsqueda insensible a mayúsculas.
```

25. Usar hint en reemplazo

```
db.clientes.replaceOne(
 { ciudad: "Querétaro" },
 { nombre: "Reemplazado", email: "nuevo@email.com" },
 { hint: { ciudad: 1 } }
)
// Sugiere uso de índice para la consulta.
```

Casos adicionales y avanzados

26. Añadir campo de auditoría a todos los documentos

```
db.clientes.updateMany({}, { $set: { actualizado_por: "sistema" } });
```

27. Vaciar array de historial

```
db.clientes.updateOne(
 { _id: "CL6" },
 { $set: { historial_acceso: [] } }
);
```

28. Eliminar método de pago PayPal

```
db.clientes.updateOne(
 { "metodos_pago.tipo": "PayPal" },
 { $pull: { metodos_pago: { tipo: "PayPal" } } }
);
```

29. Cambiar dispositivo en historial

```
db.clientes.updateOne(
 { _id: "CL7", "historial_acceso.dispositivo": "iPad" },
 { $set: { "historial_acceso.$.dispositivo": "Tablet" } }
);
```

30. Añadir múltiples valores con $each

```
db.clientes.updateOne(
 { _id: "CL8" },
 {
```

```
  $push: {
    "preferencias.categorias_favoritas": {
      $each: ["música", "deportes"]
    }
  }
 }
);
```

31. Usar arrayFilters para editar elementos específicos

```
db.clientes.updateOne(
 { _id: "CL9" },
 { $set: { "metodos_pago.$[elem].numero": "**** **** **** 0000" } },
 {
  arrayFilters: [{ "elem.tipo": "tarjeta_credito" }]
 }
);
```

32. Añadir campo con fecha actual

```
db.clientes.updateOne(
 { _id: "CL10" },
 { $currentDate: { ultima_actualizacion: true } }
);
```

33. Usar $min para establecer valor mínimo

```
db.clientes.updateOne(
 { _id: "CL11" },
 { $min: { puntos_fidelidad: 100 } }
);
```

34. Usar $max para establecer valor máximo

```
db.clientes.updateOne(
 { _id: "CL12" },
 { $max: { puntos_fidelidad: 900 } }
);
```

35. Activar notificaciones solo si están desactivadas

```
db.clientes.updateOne(
 { "preferencias.notificaciones": false },
 { $set: { "preferencias.notificaciones": true } }
);
```

36. Añadir tags como array

```
db.clientes.updateOne(
 { _id: "CL13" },
 { $set: { tags: ["VIP", "reciente"] } }
);
```

37. Usar $rename para renombrar campo

```
db.clientes.updateOne(
 {},
 { $rename: { telefono: "telefono_movil" } }
);
```

38. Incrementar puntos y agregar categoría en una sola operación

```
db.clientes.updateOne(
 { _id: "CL14" },  {
   $inc: { puntos_fidelidad: 10 },
   $addToSet: { "preferencias.categorias_favoritas": "hogar" }
 }
);
```

39. Eliminar todos los campos menos _id y nombre (solo con replaceOne)

```
db.clientes.replaceOne(
 { _id: "CL15" },
 {
   _id: "CL15",
   nombre: "Anónimo"
 }
);
```

40. Reasignar ciudad en dirección

```
db.clientes.updateOne(
 { _id: "CL16" },
 { $set: { "direccion.ciudad": "Tijuana" } }
);
```

Evaluación: Ejercicios sobre UPDATE en MongoDB

Crea un fichero evaluacion_update.js.

Incluye al inicio:

```
use("LabTestDB");
const coleccion = db.getCollection("clientes_eval_update");
```

Completa los siguientes ejercicios escribiendo las operaciones updateOne, updateMany o replaceOne según corresponda.

1. Actualiza solo el cliente que tenga más puntos de fidelidad y cámbiale el campo estatus a "premium". Usa sort.

```
coleccion.updateOne(
 { estatus: "activo" },
 { /* completa aquí */ },
 { sort: { /* completa aquí */ } }
);
```

2. Incrementa en 25 los puntos de fidelidad de todos los clientes que tengan más de 3 métodos de pago registrados. Usa un operador de comparación adecuado.

```
coleccion.updateMany(
 { /* condición sobre longitud del array */ },
 { /* operador de incremento */ }
);
```

3. Establece "verificado" como nuevo estatus para los clientes con nombre "sofia", sin importar si está en mayúsculas o minúsculas. Usa collation.

```
coleccion.updateMany(
 { nombre: /* expresión aquí */ },
 { $set: { estatus: "verificado" } },
 {
   collation: { /* completa aquí */ }
 }
);
```

4. Usa upsert para crear un cliente si no existe con _id: "CL_TEST01" y establece nombre y email.

```
coleccion.updateOne(
 { _id: "CL_TEST01" },
 { $set: { nombre: "Estudiante Nuevo", email: /* completa aquí */ } },
 { upsert: /* true o false */ }
);
```

5. Reemplaza completamente el documento del cliente con _id: "CL_REPLACE1" por uno nuevo con solo los campos _id, nombre y email. Usa replaceOne.

```
coleccion.replaceOne(
 { _id: "CL_REPLACE1" },
 { /* documento nuevo aquí */ }
);
```

6. Renombra el campo telefono a telefono_contacto en todos los documentos que tengan ese campo.

```
coleccion.updateMany(
 { telefono: { $exists: true } },
 { /* operador para renombrar */ }
);
```

7. Añade un nuevo valor "belleza" al array preferencias.categorias_favoritas solo si no existe. Evita duplicados.

```
coleccion.updateMany(
 {},
 { /* operador adecuado para arrays sin duplicados */ }
);
```

8. En el array historial_acceso, cambia el campo dispositivo a "Tablet" solo en las entradas que actualmente sean "iPad". Usa arrayFilters.

```
coleccion.updateOne(
 { _id: "CL_EJEMPLO_ARRAY" },
 { $set: { /* usa $[elem] */ } },
 {
   arrayFilters: [ /* filtro para elem */ ]
 }
);
```

9. Elimina todos los campos comentarios de los documentos que contengan esa propiedad.

```
coleccion.updateMany(
 { comentarios: { $exists: true } },
 { /* operador de eliminación de campo */ }
);
```

10. Actualiza el email de todos los clientes que tienen una cuenta con dominio @outlook.com, cambiándolo a "anonimo@correo.com". Usa una expresión regular.

```
coleccion.updateMany(
 { email: /* expresión regular */ },
 { /* nueva dirección */ }
);
```