Apuntes de MongoDB – Ampliación de Bases de Datos

1. Fundamentos del modelo documental

- **Documentos y colecciones**: MongoDB almacena datos en documentos BSON (Binary JSON); una colección agrupa documentos semejantes.
- **Schema-less con validación opcional**: los campos no son obligatorios, pero puedes usar `$jsonSchema` para imponer reglas de tipo y obligatoriedad.

```
// Ejemplo de validación
 db.createCollection("students", {
   validator: {
     $jsonSchema: {
       required: ["name", "gpa"],
       properties: {
         name:  { bsonType: "string" },
         gpa:   { bsonType: "double", minimum: 0, maximum: 4 }
       }
     }
   }
 })
```

Diseño de esquemas (embed vs reference)

Buen momento para *embed*	Buen momento para *referenciar*
Lecturas de un único documento suponen >90 % del patrón de acceso	El subdocumento crece sin control (listas enormes)
Subdocumentos altamente coherentes y poco compartidos	El subdocumento se comparte entre muchos padres
Operaciones ACID locales importantes	Necesitas leer/escribir el subdocumento por separado

Red flags: colecciones excesivas, arrays con muchos elementos, documentos muy grandes (> 16 MB) o demasiados índices.

---

2. Índices y optimización

2.1 Tipos de índice

- **Single-field**: `{campo: 1 | -1}`
- **Compuesto**: `{campoA: 1, campoB: -1}`
- **Multikey** (arrays), **Text**, **2d**, **2dsphere**, **Hashed**, **TTL**, **Parciales**.

```
// Ejemplo examen: geoespacial
 db.zips.createIndex({ location: "2dsphere" });

// Índice compuesto para filtros habituales
 db.zips.createIndex({ state: 1, city: 1 });

// Texto (búsqueda de palabras)
 db.products.createIndex({ descripcion: "text" }, { default_language:
"spanish" });

// Índice parcial (solo docs con stock > 0)
 db.items.createIndex(
   { sku: 1 },
   { partialFilterExpression: { stock: { $gt: 0 } } }
 );
```

Comandos útiles

```
 db.collection.getIndexes();
 db.collection.dropIndex("state_1_city_1");
 db.collection.explain().find({state:"CA"});
```

Consejo: `explain()` te muestra si la consulta usa `IXSCAN` (índice) o `COLLSCAN` (barrido completo).

---

3. Operaciones CRUD

3.1 Inserción

```
 db.usuarios.insertOne({nombre:"Ana", edad:30});
 db.usuarios.insertMany([
   {nombre:"Luis"},
   {nombre:"María"}
 ]);
```

3.2 Lectura

```
// Filtro y proyección
 db.usuarios.find({ edad: { $gte: 18 } }, { nombre: 1, _id: 0 });

// Ordenar, paginar
 db.usuarios.find().sort({ edad:-1 }).skip(10).limit(5);
```

Operadores frecuentes

- **Comparación**: `$eq`, `$ne`, `$gt`, `$gte`, `$lt`, `$lte`, `$in`, `$nin`.

- **Lógicos**: `$and`, `$or`, `$not`, `$nor`.
- **Arrays**: `$all`, `$elemMatch`, `$size`, `$slice`.
- **Texto**: `$regex`, `$text`.
- **Geo**: `$geoWithin`, `$geoIntersects`, `$near`, `$nearSphere`.

3.3 Actualización

```
// upsert (inserta si no existe)
 db.usuarios.updateOne(
   { nombre:"Ana" },
   { $set:{ edad:31 }, $setOnInsert:{ activo:true } },
   { upsert:true }
 );

// Modificar arrays
 db.alumnos.updateOne(
   { nombre:"Perico", "notas.asignatura":"Maths" },
   { $set:{ "notas.$.asignatura":"Applied Maths" } }
 );

// Incremento masivo
 db.inventario.updateMany({}, { $inc:{ stock:-1 } });
```

3.4 Eliminación

```
 db.logs.deleteMany({ timestamp: { $lt: new Date("2025-01-01") } });
```

---

4. Consultas geoespaciales

1. **Definir índice 2dsphere**

```
 db.zips.createIndex({ location:"2dsphere" });
```

1. **Encontrar documentos en un polígono**

```
const polygon = {
  type: "Polygon",
  coordinates: [[
    [-72.292215, 44.671914],
    [-73.694505, 43.015552],
    [-70.899497, 41.893637],
    [-72.292215, 44.671914]  // cerrar el anillo
  ]]
};
```

```
db.zips.find({ location: { $geoWithin: { $geometry: polygon } } });
```

1. **Actualizar ciudades dentro del polígono (examen)**

```
db.zips.updateMany(
  {
    state:"VT",
    location: { $geoWithin:{ $geometry: polygon } }
  },
  { $inc:{ population: 1000 } }
);
```

1. **\$geoNear en aggregate**

```
db.tiendas.aggregate([
  { $geoNear: {
      near: { type:"Point", coordinates:[ -3.70379, 40.41678 ] },
      distanceField: "dist",
      spherical: true,
      maxDistance: 100000  // 100 km
  }},
  { $limit: 10 }
]);
```

---

5. Framework de Agregación

El *pipeline* transforma documentos etapa a etapa.

Etapas esenciales con ejemplos

Stage	Propósito	Ejemplo breve
`$match`	Filtrar temprano, aprovecha índices	`{ $match: { estado:"CA" } }`
`$project`	Seleccionar / calcular campos	`{ $project:{ año:{ $year:"$fecha" } } }`
`$group`	Agregar con acumuladores	`{ $group:{ _id:"$estado", total: { $sum:"$ventas" } } }`
`$sort`	Ordenar	`{ $sort:{ total:-1 } }`
`$unwind`	Descomponer arrays	`{ $unwind:"$lineas" }`

Stage	Propósito	Ejemplo breve
`$lookup`	*Join* con otra colección	`{ $lookup:{ from:"clientes", localField:"clienteId", foreignField:"_id", as:"cliente" } }`
`$addFields`	Añadir campos derivados	`{ $addFields:{ iva:{ $multiply:["$total", 0.21] } } }`
`$facet`	Pipelines paralelos	`{ $facet:{ resumen:[...], detalle:[...] } }`
`$out` / `$merge`	Persistir resultado	`{ $out:"reporte_mensual" }`

Acumuladores populares

`$sum`, `$avg`, `$min`, `$max`, `$push`, `$addToSet`, `$first`, `$last`.

Ejemplo 1 – Contar ciudades >10 000 hab por estado (examen)

```
db.zips.aggregate([
  { $match:{ population:{ $gt:10000 } } },
  { $group:{ _id:"$state", numCiudades:{ $sum:1 } } },
  { $sort:{ numCiudades:-1 } }
]);
```

Ejemplo 2 – Pipeline tipo práctica

> «Listado de todas las compras de un cliente»

```
db.compras.aggregate([
  { $match:{ clienteId:ObjectId("...") } },
  { $lookup:{
      from:"proveedores",
      localField:"proveedorId",
      foreignField:"_id",
      as:"proveedor"
  }}
]);
```

---

6. MapReduce (cuando la agregación no basta)

```
db.ventas.mapReduce(
  function() { emit(this.producto, this.cantidad); },
  function(clave, valores) { return Array.sum(valores); },
```

```
  { out:"totales_producto" }
);
```

Tip: En la mayoría de casos el framework de agregación es más rápido y simple.

---

7. Plantillas de soluciones para la práctica LocalStore

1. **Compras de un cliente**

```
[
  { $match:{ clienteId: ObjectId("…") } }
]
```

1. **Proveedores para un producto**

```
[
  { $match:{ productoId:ObjectId("…") } },
  { $lookup:{ from:"proveedores", localField:"proveedorId",
foreignField:"_id", as:"proveedor" } },
  { $project:{ _id:0, proveedor:{ $arrayElemAt:["$proveedor",0] } } }
]
```

1. **Productos diferentes comprados por cliente** – `$group` + `$addToSet`.
2. **Productos de "Modas Paqui" que contienen "manga corta"** – `$match` con regex y proveedor fixed.
3. **Peso y volumen total en fecha** – `$match` por fecha + `$group` sumando `peso` y `volumen`.
4. **Media de envíos por mes y almacén** – `$group` por `{mes, almacen}` y `$avg`.
5. **Top 3 proveedores por facturación** – `$group` + `$sort` + `$limit:3`.
6. **Almacenes cercanos (<100 km)** – `$geoNear` con `maxDistance:100000`.
7. **Compras con destino en polígono** – `$match` `destino:{$geoWithin:{ $geometry: poly}}`.
8. **Guardar compras a enviar en día X** – pipeline + `$out:"pendientes_YYYYMMDD"`.

---

8. Buenas prácticas y consejos rápidos

- **Filtra lo antes posible** (`$match` Stage 1).
- **Proyecta solo lo necesario** para reducir transferencia.
- **No abuses de **``: si se puede embed, mejor.
- **Mide con **`` y ajusta índices.
- **Limita tamaño de arrays** (`$slice`), evita que superen varias decenas de elementos si se consultan completos.
- **Versiona tu esquema** añadiendo un campo `_schemaVersion`.

¡Con esto tienes un repositorio completísimo de ejemplos para afrontar tus actividades y exámenes! Revisa cada bloque, prueba en tu *shell* y ajusta al contexto de tus colecciones.

9. Ejercicios de preparación para el examen (con soluciones)

> Todos los ejercicios parten de colecciones de ejemplo (`zips`, `tiendas`, `ventas`, `usuarios`, `compras`, `productos`, `posts`, `hoteles`, `pedidos`). Ajusta nombres/campos a tu propio esquema.

9.1 Índices y optimización

#	Enunciado	Solución	Explicación
1	Ejecutar eficientemente `find({state:"CA", city:"SAN DIEGO"})`	`db.zips.createIndex({state:1, city:1})`	Índice compuesto con ord predicado.
2	Mejorar `find({categoria:"ropa", precio:{$lt:50}}).sort({precio:1})`	`db.productos.createIndex({categoria:1, precio:1})`	**Cubriendo**: incluye campo `SORT`.
3	Borrar índices que nunca se usan	`db.collection.dropIndexes(["campo1_1", "foo_1_bar_-1"])`	Usa `db.collection.aggreg {}}])` para detectar huér
4	TTL para logs mayores a 30 días	`db.logs.createIndex({timestamp:1}, {expireAfterSeconds:2592000})`	Borra docs automáticame
5	Índice parcial solo para artículos con stock > 0	`db.items.createIndex({sku:1}, {partialFilterExpression:{stock:{$gt:0}}})`	Reduce tamaño de índice.

9.2 Aggregation Pipeline

Ejercicio A – Promedio de población de las 3 ciudades más pobladas por estado

```
db.zips.aggregate([
  { $sort:{ population:-1 } },
  { $group:{
      _id:"$state",
      topCities:{ $push:{ city:"$city", pop:"$population" } }
  }},
  { $project:{
      top3:{ $slice:["$topCities",3] }
  }},
  { $unwind:"$top3" },
  { $group:{
      _id:"$_id",
      avgTop3:{ $avg:"$top3.pop" }
  }},
```

```
  { $sort:{ avgTop3:-1 } }
]);
```

Ejercicio B – Número de productos distintos comprados por cada cliente y total gastado

```
db.compras.aggregate([
  { $group:{
      _id:"$clienteId",
      totalGasto:{ $sum:"$importe" },
      productos:{ $addToSet:"$productoId" }
  }},
  { $project:{
      numProductos:{ $size:"$productos" },
      totalGasto:1
  }},
  { $sort:{ totalGasto:-1 } }
]);
```

Ejercicio C – Clientes sin pedidos en los últimos 90 días

```
const limite = new Date();
limite.setDate(limite.getDate()-90);

db.pedidos.aggregate([
  { $match:{ fecha:{ $gte:limite } } },
  { $group:{ _id:"$clienteId" } },
  { $rightOuterLookup:{
      from:"clientes",
      localField:"_id",
      foreignField:"_id",
      as:"inactivos"
  }},
  { $unwind:"$inactivos" },
  { $match:{ _id:null } },
  { $replaceWith:"$inactivos" }
]);
```

9.3 Operaciones sobre arrays

1. **Encontrar productos con ≥ 3 opiniones 5★**

```
db.productos.find({
  "opiniones":{ $elemMatch:{ rating:5 } },
  $expr:{ $gte:[ { $size:{ $filter:{ input:"$opiniones", as:"o", cond:{ $eq:
```

```
["$$o.rating",5] } } }, 3 ] }
 }});
```

1. **Añadir comentario a la primera opinión**

```
db.productos.updateOne(
  { _id:ObjectId("…"), "opiniones.0":{ $exists:true } },
  { $set:{ "opiniones.0.comentario":"Revisado por control calidad" } }
);
```

9.4 Texto completo

Consulta	Sintaxis
Posts que contengan "mongodb" y "index"	`db.posts.find({ $text:{ $search:"mongodb index" } })`
Relevancia (score)	`db.posts.find({ $text:{ $search:"nosql" } }, { score: { $meta:"textScore" } }).sort({ score: { $meta:"textScore" } })`

9.5 Geoespacial

Ejercicio D – Hoteles a < 5 km de la Sagrada Familia ordenados por precio

```
const sagrada = { type:"Point", coordinates:[ 2.174355, 41.403627 ] };
db.hoteles.aggregate([
  { $geoNear:{
      near:sagrada,
      distanceField:"dist",
      maxDistance:5000,
      spherical:true
  }},
  { $sort:{ precio:1 } }
]);
```

Ejercicio E – Precio medio de Airbnbs dentro de un barrio definido como polígono *(supongamos colección ``).*

```
 const barrio = { type:"Polygon", coordinates:[ [ [x1,y1], [x2,y2], [x3,y3],
[x1,y1] ] ] };
 db.airbnbs.aggregate([
  { $match:{ location:{ $geoWithin:{ $geometry:barrio } } } },
  { $group:{ _id:null, precioMedio:{ $avg:"$precio" } } }
 ]);
```

9.6 Actualizaciones masivas

#	Escenario	Comando
1	Reducir `stock` en 5 para productos sin ventas desde 2024-01-01	`db.productos.updateMany({ ultimaVenta:{ $lt:new Date("2024-01-01") } },{ $inc:{ stock:-5 } })`
2	Añadir campo `etiquetas:["legacy"]` a docs sin `version`	`db.docs.updateMany({ version:{ $exists:false } }, { $set:{ etiquetas:["legacy"] } })`
3	Eliminar clientes marcados como `baja:true`	`db.clientes.deleteMany({ baja:true })`

9.7 MapReduce avanzado

Total de ventas por mes y canal (online/offline)

```
db.ventas.mapReduce(
  function(){
    const mes = this.fecha.getUTCFullYear()+"-"+(this.fecha.getUTCMonth()
+1).toString().padStart(2,"0");
    emit({mes, canal:this.canal}, this.importe);
  },
  function(key, values){ return Array.sum(values); },
  { out:"ventas_mes_canal" }
);
```

9.8 Validación de esquema rápida

```
db.runCommand({
  collMod:"productos",
  validator:{
    $jsonSchema:{
      required:["sku","precio"],
      properties:{
        sku:{ bsonType:"string", pattern:"^[A-Z]{3}-\d{4}$" },
        precio:{ bsonType:"double", minimum:0 }
      }
    }
  },
  validationLevel:"strict"
});
```

9.9 Preguntas flash estilo test

1. **¿Qué operador geoespacial devuelve documentos dentro de un círculo?** → `$geoWithin + $centerSphere`.
2. **¿Cómo se fuerza un plan de índice concreto?** → `.hint({ campo:1 })`.
3. **¿Qué acumulador devuelve el primer valor de un grupo ordenado?** → `$first`.
4. **¿TTL admite índices compuestos?** → No, sólo un campo y ascendente.
5. **¿Cómo ignorar campos desconocidos en validación?** → `additionalProperties:false` dentro de `$jsonSchema`.

Asegúrate de practicar cada ejercicio con tus datos reales y usa `` donde sea posible para verificar que el plan es el óptimo.

9.10 Más ejercicios tipo examen

Ejercicio F – Top 3 productos con más unidades vendidas

```
db.ventas.aggregate([
  { $group:{ _id:"$productoId", total:{ $sum:"$cantidad" } } },
  { $sort:{ total:-1 } },
  { $limit:3 }
]);
```

Ejercicio G – Media y total de facturación por cliente de Madrid

```
db.clientes.aggregate([
  { $match:{ ciudad:"Madrid" } },
  { $lookup:{ from:"ventas", localField:"_id", foreignField:"clienteId",
as:"compras" } },
  { $unwind:"$compras" },
  { $group:{ _id:"$_id", nombre:{ $first:"$nombre" }, total:{
$sum:"$compras.importe" }, media:{ $avg:"$compras.importe" } } }
]);
```

Ejercicio H – Insertar campo "iva" en cada venta calculado como 21% del importe

```
db.ventas.updateMany({}, [
  { $set:{ iva:{ $multiply:["$importe", 0.21] } } }
]);
```

Ejercicio I – Encontrar almacenes sin coordenadas geoespaciales

```
db.almacenes.find({ location: { $exists: false } });
```

Ejercicio J – Cuántos usuarios se registraron cada mes del último año

```
db.usuarios.aggregate([
  { $match:{ fechaRegistro:{ $gte: new Date("2024-06-01") } } },
  { $project:{ mes: { $dateToString:{ format:"%Y-%m",
date:"$fechaRegistro" } } } },
  { $group:{ _id:"$mes", total:{ $sum:1 } } },
  { $sort:{ _id:1 } }
]);
```

Ejercicio K – Buscar productos cuya descripción incluya "rebajado" pero no "agotado"

```
db.productos.find({
  descripcion: { $regex: /rebajado/i, $not: { $regex: /agotado/i } }
});
```

Ejercicio L – Agrupar comentarios por usuario y contar cuántos ha hecho

```
db.comentarios.aggregate([
  { $group:{ _id:"$usuarioId", total:{ $sum:1 } } },
  { $sort:{ total:-1 } }
]);
```

Ejercicio M – Eliminar productos con 0 unidades en stock y sin ventas

```
db.productos.deleteMany({ stock:0, ventas:0 });
```

Ejercicio N – Mostrar usuarios que han comprado todos los productos del catálogo

Asume que hay una colección `productos` y otra `ventas`

```
const totalProductos = db.productos.countDocuments();
db.ventas.aggregate([
  { $group:{ _id:"$clienteId", productos:{ $addToSet:"$productoId" } } },
  { $project:{ total: { $size:"$productos" } } },
  { $match:{ total: totalProductos } }
]);
```

Ejercicio O – Clientes con más de 3 pedidos en el último mes

```
const fecha = new Date();
fecha.setMonth(fecha.getMonth() - 1);
db.pedidos.aggregate([
  { $match:{ fecha: { $gte: fecha } } },
  { $group:{ _id:"$clienteId", numPedidos:{ $sum:1 } } },
```

```
  { $match:{ numPedidos: { $gt:3 } } }
]);
```
