Bases de Datos 2

Trabajo Práctico Obligatorio

Mariano Agopian - 62317- maagopian@itba.edu.ar

Lautaro Farias - 60505 - lfarias@itba.edu.ar

Axel Castro Benza - 62358 - acastrobenza@itba.edu.ar

Índice

Decisiones tomadas durante el desarrollo

Se optó por utilizar MongoDB como la base de datos principal debido a varios factores clave. En primer lugar, MongoDB permite escalabilidad horizontal mediante sharding, lo que lo convierte en una opción ideal para manejar grandes volúmenes de datos, como productos y clientes. Su capacidad de adaptarse a esquemas flexibles es otra ventaja importante, ya que no requiere un esquema fijo. Esto facilita la incorporación de cambios en el sistema de facturación, como la adición de nuevos campos (por ejemplo, "tipo_de_pago" o "promoción_aplicada"). Además, MongoDB gestiona los datos en documentos BSON, lo que resulta adecuado para representar entidades como "Facturas" que incluyen múltiples atributos.

Otro aspecto relevante es que MongoDB provee tolerancia a particiones, lo que asegura que el sistema de facturación pueda seguir operando incluso durante fallos de red. Por último, su capacidad para soportar consultas avanzadas permite manejar consultas complejas sin problemas, garantizando un rendimiento óptimo en las respuestas a las consultas.

Por otro lado, se utilizó Redis como capa de caché para mejorar el rendimiento y optimizar la gestión de datos en el sistema de facturación. Redis acelera las consultas más frecuentes de productos y clientes, reduciendo significativamente la necesidad de acceder constantemente a MongoDB. Esto a su vez, disminuye la carga sobre MongoDB, reservándose para operaciones críticas

Redis también mejora la experiencia del usuario al minimizar la latencia en las consultas, especialmente en operaciones de alta concurrencia, como la verificación de disponibilidad de stock o el acceso a la información de productos durante el proceso de facturación. Gracias a su almacenamiento en memoria y su alta velocidad de respuesta, Redis es ideal para manejar datos dinámicos y transitorios que requieren respuestas inmediatas, como los precios de productos y los estados actuales del stock.

Diseño de modelo de datos

En cuanto al modelo de datos en Redis, las tablas Clientes y Productos se almacenan en hashes. Los datos de cada cliente se almacenan en una estructura de hash con la clave "Cliente:{nro_cliente}" mientras que los datos de cada producto se almacenan en otro hash con la clave "Producto:{codigo_producto}". Se utilizan estas estructuras debido a su naturaleza transitoria y la necesidad de acceso rápido para las consultas frecuentes.

Por su parte, en MongoDB se almacenan las tablas Teléfono, Cliente, Factura, Producto y Detalle de Factura, ya que MongoDB ofrece una excelente solución para almacenar y gestionar documentos más complejos y persistentes, que requieren un esquema flexible y el soporte para consultas avanzadas.

Configuración de entorno y carga de datos

MongoDB

Para cargar la información de los csv dentro de nuestra base de datos MongoDB se subieron los archivos csv a docker con el comando *cp* de la siguiente manera:

```
(Si el directorio no esta creado : docker exec Mymongo mkdir -p /files)
docker cp ./data/e01_cliente.csv Mymongo:/files/e01_cliente.csv
docker cp ./data/e01_detalle_factura.csv
Mymongo:/files/e01_detalle_factura.csv
docker cp ./data/e01_factura.csv Mymongo:/files/e01_factura.csv
docker cp ./data/e01_telefono.csv Mymongo:/files/e01_telefono.csv
docker cp ./data/e01_producto.csv Mymongo:/files/e01_producto.csv
```

Luego dentro de docker se utilizó el comando *sed* para cambiar los separadores del csv de “;” a “,” debido a que surgían problemas al momento de importarlos dentro de la base de datos. Esto se realizó de la siguiente manera:

```
sed 's/;/,/g' /files/e01_cliente.csv > /files/e01_cliente_comas.csv
sed 's/;/,/g' /files/e01_detalle_factura.csv >
/files/e01_detalle_factura_comas.csv
sed 's/;/,/g' /files/e01_factura.csv > /files/e01_factura_comas.csv
sed 's/;/,/g' /files/e01_telefono.csv > /files/e01_telefono_comas.csv
sed 's/;/,/g' /files/e01_producto.csv > /files/e01_producto_comas.csv
```

Tras esto, se importaron los datos de los csv a colecciones dentro de la base de datos con el comando *mongoimport*. Esto se realizó de la siguiente manera:

```
mongoimport --host localhost --db mymongo --collection Cliente --type csv
--file /files/e01_cliente_comas.csv --headerline
```

```
mongoimport --host localhost --db mymongo --collection DetalleFactura --type
csv --file /files/e01_detalle_factura_comas.csv --headerline
mongoimport --host localhost --db mymongo --collection Factura --type csv
--file /files/e01_factura_comas.csv --headerline
mongoimport --host localhost --db mymongo --collection Telefono --type csv
--file /files/e01_telefono_comas.csv --headerline
mongoimport --host localhost --db mymongo --collection Producto --type csv
--file /files/e01_producto_comas.csv --headerline
```

Redis

Se utiliza el contenedor de docker que se instala en la práctica de la materia. Los datos se cargan automáticamente una vez inicializada la API.

API

Tras tener los archivos csv en MongoDB, se deben ejecutar los siguientes comandos para generar las dependencias necesarias para el funcionamiento de la misma:

```
# Navegar al directorio del proyecto
cd polyglot_system
# Crear un entorno virtual en Python
python3 -m venv venv
# Activar el entorno virtual:
# En Mac/Linux:
source venv/bin/activate
# En Windows:
venv\Scripts\activate
# Instalar las dependencias necesarias
pip install fastapi uvicorn pymongo redis pandas
```

Luego, para inicializar la API:

```
uvicorn main:app --reload
```

Tras esto, se puede acceder a la misma en: http://127.0.0.1:8000/docs#/

Queries

Las queries se encuentran dentro de la carpeta “consultas” y están escritas en JavaScript, que es el lenguaje de programación que se utiliza en la terminal de MongoDB.

1. Obtener los datos de los clientes junto con sus teléfonos.

```
db.Cliente.aggregate([
  {
    $lookup: {
      from: "Telefono",
      localField: "nro_cliente",
      foreignField: "nro_cliente",
      as: "telefonos",
    },
  },
  {
    $project: {
      _id: 0,
      nombre: 1,
      apellido: 1,
      direccion: 1,
      activo: 1,
      nro_cliente: 1,
      telefonos: {
        codigo_area: 1,
        nro_telefono: 1,
        tipo: 1,
      },
    },
  },
  {
    $unwind: "$telefonos",
  },
  {
    $group: {
      _id: "$nro_cliente",
      nombre: { $first: "$nombre" },
      apellido: { $first: "$apellido" },
      direccion: { $first: "$direccion" },
      activo: { $first: "$activo" },
      telefonos: { $push: "$telefonos" },
    },
  },
  {
    $project: {
      _id: 0,
      nombre: 1,
      apellido: 1,
      direccion: 1,
      activo: 1,
      nro_cliente: "$_id",
```

```
      telefonos: 1,
    },
  },
  {
    $sort: { nro_cliente: 1},
  },
])
```

2. Obtener el/los teléfonos/s y el número de cliente del cliente con nombre “Jacob” y apellido “Cooper”.

```
db.Cliente.aggregate([
  {
    $match: {
      nombre: "Jacob",
      apellido: "Cooper"
    }
  },
  {
    $lookup: {
      from: "Telefono",
      localField: "nro_cliente",
      foreignField: "nro_cliente",
      as: "telefonos"
    }
  },
  {
    $project: {
      _id: 0,
      nombre: 1,
      apellido: 1,
      nro_cliente: 1,
      telefonos: {
        codigo_area: 1,
        nro_telefono: 1,
        tipo: 1
      }
    }
  }
]);
```

3. Mostrar cada teléfono junto con los datos del cliente.

```
db.Cliente.aggregate([
  {
    $lookup: {
      from: "Telefono",
      localField: "nro_cliente",
      foreignField: "nro_cliente",
      as: "telefonos"
    }
```

```
  },
  {
    $project: {
      _id: 0,
      nro_cliente: 1,
      nombre: 1,
      apellido: 1,
      direccion: 1,
      activo: 1,
      telefonos: 1
    }
  },
  {
    $unwind: "$telefonos"
  },
  {
    $project: {
      _id: 0,
      nro_cliente: 1,
      nombre: 1,
      apellido: 1,
      direccion: 1,
      activo: 1,
      codigo_area: "$telefonos.codigo_area",
      nro_telefono: "$telefonos.nro_telefono",
      tipo: "$telefonos.tipo"
    }
  }
]);
```

4. Obtener a todos los clientes que tengan registrada al menos una factura.

```
db.Cliente.aggregate([
  {
    $lookup: {
      from: "Factura",
      localField: "nro_cliente",
      foreignField: "nro_cliente",
      as: "has_factura",
    },
  },
  {
    $match: {
      has_factura: { $exists: true, $ne: [] },
    },
  },
  {
    $project: {
      _id: 0,
      nro_cliente: 1,
      nombre: 1,
      apellido: 1,
    },
```

```
  },
]);
```

5. Identificar todos los clientes que no tengan registrada ninguna factura.

```
db.Cliente.aggregate([
  {
    $lookup: {
      from: "Factura",
      localField: "nro_cliente",
      foreignField: "nro_cliente",
      as: "has_factura",
    },
  },
  {
    $match: {
      has_factura: { $exists: true, $eq: [] },
    },
  },
  {
    $project: {
      _id: 0,
      nro_cliente: 1,
      nombre: 1,
      apellido: 1,
    },
  },
]);
```

6. Devolver todos los clientes, con la cantidad de facturas que tienen registradas (si no tienen considerar cantidad en 0)

```
db.Cliente.aggregate([
{
  $lookup: {
    from: "Factura",
    localField: "nro_cliente",
    foreignField: "nro_cliente",
    as: "facturas"
  }
},
{
  $addFields: {
    cantidad_facturas: { $size: { $ifNull: ["$facturas", []] } }
  }
},
{
  $project: {
    _id: 0,
    nro_cliente: 1,
    nombre: 1,
    apellido: 1,
```

```
    cantidad_facturas: 1
  }
}
]);
```

7. Listar los datos de todas las facturas que hayan sido compradas por el cliente de nombre "Kai" y apellido "Bullock".

```
db.Cliente.aggregate([
  {
    $match: {
      nombre: "Kai",
      apellido: "Bullock"
    }
  },
  {
    $lookup: {
      from: "Factura",
      localField: "nro_cliente",
      foreignField: "nro_cliente",
      as: "facturas"
    }
  },
  {
    $unwind: "$facturas",
  },
  {
    $replaceRoot: { newRoot: "$facturas" },
  }
]);
```

8. Seleccionar los productos que han sido facturados al menos 1 vez.

```
db.Producto.aggregate([
  {
    $lookup: {
      from: "DetalleFactura",
      localField: "codigo_producto",
      foreignField: "codigo_producto",
      as: "has_factura",
    },
  },
  {
    $match: {
      has_factura: { $exists: true, $ne: [] },
    },
  },
  {
    $project: {
      _id: 0,
      codigo_producto: 1,
      marca: 1,
```

```
      nombre: 1,
      descripcion: 1,
      precio: 1,
      stock: 1,
    },
  },
]);
```

9. Listar los datos de todas las facturas que contengan productos de las marcas "Ipsum".

```
db.DetalleFactura.aggregate([
  {
    $lookup: {
      from: "Factura",
      localField: "nro_factura",
      foreignField: "nro_factura",
      as: "factura_info"
    }
  },
  {
    $unwind: "$factura_info"
  },
  {
    $lookup: {
      from: "Producto",
      localField: "codigo_producto",
      foreignField: "codigo_producto",
      as: "producto_info"
    }
  },
  {
    $unwind: "$producto_info"
  },
  {
    $match: {
      "producto_info.marca": { $regex: "Ipsum", $options: "i" }
    }
  },
  {
    $project: {
      _id: 0,
      nro_factura: "$factura_info.nro_factura",
      fecha: "$factura_info.fecha",
      total_sin_iva: "$factura_info.total_sin_iva",
      iva: "$factura_info.iva",
      total_con_iva: "$factura_info.total_con_iva",
      codigo_producto: "$producto_info.codigo_producto",
      marca: "$producto_info.marca",
      nombre: "$producto_info.nombre",
      descripcion: "$producto_info.descripcion",
      cantidad: "$cantidad"
    }
```

```
  }
]);
```

10. Mostrar nombre y apellido de cada cliente junto con lo que gastó en total, con IVA incluido.

```
db.Cliente.aggregate([
  {
    $lookup: {
      from: "Factura",
      localField: "nro_cliente",
      foreignField: "nro_cliente",
      as: "facturas"
    }
  },
  {
    $unwind: "$facturas"
  },
  {
    $group: {
      _id: { nro_cliente: "$nro_cliente", nombre: "$nombre", apellido:
"$apellido" },
      total_gastado: { $sum: "$facturas.total_con_iva" }
    }
  },
  {
    $project: {
      _id: 0,
      nombre: "$_id.nombre",
      apellido: "$_id.apellido",
      total_gastado: 1
    }
  }
]);
```

Vistas

1. Se necesita una vista que devuelva los datos de las facturas ordenadas por fecha.

```
db.createCollection("FacturasOrdenadas", {
  viewOn: "Factura",
  pipeline: [
    { $sort: { fecha: 1 } }
  ]
});
```

2. Se necesita una vista que devuelva todos los productos que aún no han sido facturados.

```
db.createView("ProductosNoFacturados", "Producto", [
  {
```

```
    $lookup: {
      from: "DetalleFactura",
      localField: "codigo_producto",
      foreignField: "DetalleFactura.codigo_producto",
      as: "facturas",
    },
  },
  {
    $match: {
      facturas: [],
    },
  },
  {
    $project: {
      _id: 0,
      codigo_producto: 1,
      marca: 1,
      nombre: 1,
      descripcion: 1,
      precio: 1,
      stock: 1,
    },
  },
]);
```

Funcionalidades

Las funcionalidades como la inserción, remoción y actualización tanto de clientes como de productos se encuentran en la propia API. Dentro de la sección de “consultas” se encuentran las tablas que explican el funcionamiento de cada método junto con sus requisitos a la hora de ejecutarlos.