Instituto Tecnológico de Buenos Aires

72.41 Bases de Datos II

TPO

Grupo 3

(62028) Nicolás Matías Margenat - nmargenat@itba.edu.ar

(62094) Juan Burda - jburda@itba.edu.ar

(62493) Saul Ariel Castañeda - scastaneda@itba.edu.ar

(62504) Elian Paredes - eparedes@itba.edu.ar

Tabla de contenidos

1. Introducción

En el presente informe se busca detallar las decisiones tomadas en el desarrollo del trabajo práctico obligatorio. En una primera instancia se explicará cómo utilizar la API. Luego, se dará una explicación de la estrategia de migración junto con las decisiones que llevaron al equipo a utilizar dicha estrategia. Finalmente, se presentarán las queries y vistas que se realizaron para PostgreSQL y MongoDB.

2. Uso de la API

Se cuentan con dos APIs, accesibles mediante puertos distintos. La única diferencia entre ambas APIs es el puerto en el que corren, los endpoints son exactamente iguales para ambas. La API del puerto 3000 utiliza PostgreSQL como base de datos, y en el puerto 3001 corre la API con MongoDB como base de datos.

Nótese que dentro de la carpeta `docs/` se encuentra una API realizada con Postman para poder probar todos los endpoints rápidamente.

A continuación se muestra una tabla con los endpoints.

Método	**Endpoint**	**Body**	**Descripción**
GET	`/clientes`	—	Buscar todos los clientes
	`/clientes/:id`	—	Buscar un solo cliente por su `id`
	`/productos`	—	Buscar todos los productos
	`/productos/:id`	—	Buscar un solo producto por su `id`
POST	`/clientes`	**nombre**: String (*requerido*): El nombre del cliente. **apellido**: String (*requerido*): El apellido del cliente **direccion**: String (*requerido*): La dirección del cliente **activo**: Integer (*requerido*): El activo del cliente	Crear un nuevo cliente
	`/productos`	**marca**: String (*requerido*): Marca del producto **nombre**: String (*requerido*): Nombre del producto **descripcion**: String (*requerido*): Descripción del producto	Crear un nuevo producto

		precio: Float (*requerido*): Precio del producto **stock**: Integer (*requerido*): Stock disponible del producto	
PUT	`/clientes/:id`	**nombre**: String (*opcional*): El nombre del cliente **apellido**: String (*opcional*): El apellido del cliente **direccion**: String (*opcional*): La dirección del cliente **activo**: Integer (*opcional*): El activo del cliente	Modifica un cliente existente por su `id`
	`/productos/:id`	**marca**: String (*opcional*): Marca del producto **nombre**: String (*opcional*): Nombre del producto **descripcion**: String (*opcional*): Descripción del producto **precio**: Float (*opcional*): Precio del producto **stock**: Integer (*opcional*): Stock disponible del producto	Modifica un producto existente por su `id`
DELETE	`/clientes/:id`	—	Borra un cliente existente por su `id`
	`/productos/:id`	—	Borra un producto existente por su `id`

3. Migración

Para migrar los datos se tomó la decisión de correr un script al iniciar los contenedores. No hay posibilidad de migrar los datos una vez se levanta el contenedor. Esto fue una decisión que se tomó para permitir el uso de ambas APIs a la vez, y no agregar lógica innecesaria a la aplicación.

En cuanto al proceso de migración, este se puede ver en el archivo “migrate.sh”. Lo que se hace es utilizar una funcionalidad de PostgreSQL para copiar las tuplas y pasarlas a formato JSON, cargándolas en un archivo. Luego se utiliza `mongoimport` para importar los archivos hacia Mongo y finalmente se borran los archivos generados.

El modelo de datos de Mongo se basó en el de PostgreSQL pero al tratarse la primera de una base de datos no relacional, se modificaron los esquemas para ajustarse mejor a dicho modelo. De esta manera, se tomaron las siguientes decisiones:

- Se embebió la relación *detalle_factura* dentro de *facturas*. Se tomó esta decisión pues la relación era 1-N, y se espera que en caso de ver una factura también se pidan los detalles de la misma en la mayoría de los casos. De esta manera, las operaciones de lectura tendrán un mejor rendimiento. Otro punto a tener en cuenta al tomar esta decisión son los arreglos mutables que crecen indefinidamente. En este caso, de todos modos, no resulta un problema pues se espera que una vez emitida una factura, los detalles no se vean modificados.
- Se embebió la relación *teléfonos* dentro de *clientes*. Se tomó esta decisión por las mismas razones que en *detalle_factura* y *facturas.* Asimismo, en ninguna situación de la vida real una persona tiene infinitos teléfonos, por lo que los arreglos mutables no son un problema.
- Se optó por crear un cuarto esquema: *sequence.* El objetivo de este esquema fue no perder el comportamiento de valores autoincrementales de las relaciones de *clientes* (con *nro_cliente*), *productos* (con *codigo_producto)* y con *facturas* (con *codigo_factura*). En este esquema, se cuenta con una única fila que contiene los valores de dichas columnas y se incrementan cada vez que se inserta un nuevo elemento a los tres esquemas previamente mencionados. Esta operación se hace transaccionalmente.

Consecuentemente, se terminó con cuatro esquemas: *facturas*, *clientes*, *productos* y *sequence*.

A continuación, se hablará de la migración per se,

Para la primera parte del trabajo (la de PostgreSQL), Prisma resultó de gran ayuda y aceleró muchísimo el trabajo. Gracias al ORM, se logró terminar la API funcional en muy poco tiempo. Esto no sucedió con Mongo, puesto que para hacer operaciones transaccionales Prisma requiere de una réplica de la base de datos de Mongo. Armar el Docker con dicho replica set resultó una tarea no trivial que sacó todo el tiempo ganado en la implementación de la primera parte del trabajo. Sin embargo, se considera que fue un aprendizaje y que valió la pena igualmente realizarlo de esta manera.

4. Queries/Vistas

4.1. Queries

4.1.1. Obtener el teléfono y el número de cliente del cliente con nombre "Wanda" y apellido "Baker".

PostgreSQL	`SELECT nro_telefono, nro_cliente` `FROM e01_telefono NATURAL JOIN e01_cliente` `WHERE nombre = 'Wanda' AND apellido = 'Baker';`
MongoDB	`db.clientes.aggregate([` `  {` `  $match: {` `  nombre: "Wanda",` `  apellido: "Baker"` `  },` `  },` `  {` `  $unwind: "$telefono",` `  },` `  {` `  $project: {` `  _id: 0,` `  nro_cliente: 1,` `  nro_telefono: { $toInt: "$telefono.nro_telefono" },` `  },` `  },` `]);`

4.1.2. Seleccionar todos los clientes que tengan registrada al menos una factura.

PostgreSQL	`SELECT *` `FROM e01_cliente c` `WHERE EXISTS (` `  SELECT 1` `  FROM e01_factura f` `  WHERE c.nro_cliente = f.nro_cliente` `);`
MongoDB	`db.facturas.aggregate([` `  {` `  $lookup: {`

	from: "clientes", localField: "nro_cliente", foreignField: "nro_cliente", as: "clientes", }, }, { $unwind: "$clientes", }, { $group: { _id: "$clientes.nro_cliente", nombre: { $first: "$clientes.nombre" }, apellido: { $first: "$clientes.apellido" }, direccion: { $first: "$clientes.direccion" }, activo: { $first: "$clientes.activo" }, }, }, { $project: { _id: 0, nro_cliente: "$_id", nombre: 1, apellido: 1, direccion: 1, activo: 1, }, },]);

4.1.3. Seleccionar todos los clientes que no tengan registrada una factura.

PostgreSQL	SELECT * FROM e01_cliente c WHERE NOT EXISTS (SELECT 1 FROM e01_factura f WHERE c.nro_cliente = f.nro_cliente);
MongoDB	db.clientes.aggregate([{ $lookup: {

	from: "facturas", localField: "nro_cliente", foreignField: "nro_cliente", as: "facturas", }, }, { $unwind: { path: "$facturas", preserveNullAndEmptyArrays: true }, }, { $match: { $or: [{ facturas: { $eq: [] } }, { facturas: { $exists: false } }, { facturas: { $eq: null } },], }, }, { $project: { _id: 0, nro_cliente: 1, nombre: 1, apellido: 1, direccion: 1, activo: 1, }, },]);

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

PostgreSQL	SELECT * FROM e01_producto p WHERE EXISTS (SELECT 1 FROM e01_detalle_factura df WHERE p.codigo_producto = df.codigo_producto);
MongoDB	db.productos.aggregate([{ $lookup: { from: "facturas", localField: "codigo_producto",

	foreignField: "detalle_factura.codigo_producto", as: "facturas" } }, { $match: { facturas: { $exists: true, $ne: [] } } }, { $project: { _id: 0, codigo_producto: 1, marca: 1, nombre: 1, descripcion: 1, precio: 1, stock: 1 } }]);

4.1.5. Seleccionar los datos de los clientes junto con sus teléfonos.

PostgreSQL	SELECT c.*, t.nro_telefono FROM e01_cliente c NATURAL JOIN e01_telefono t;
MongoDB	db.clientes.aggregate([{ $match: { telefono: { $ne: null } } }, { $unwind: "$telefono", }, { $project: { _id: 0, nro_cliente: 1, nombre: 1, apellido: 1, direccion: 1, activo: 1, nro_telefono: { $toInt: "$telefono.nro_telefono" } }, },

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

4.1.6. Devolver todos los clientes, con la cantidad de facturas que tienen registradas (admitir nulos en valores de Clientes).

PostgreSQL	`SELECT c.nro_cliente, count(nro_factura) as "cantidad_facturas"` `FROM e01_cliente c LEFT OUTER JOIN e01_factura f ON c.nro_cliente = f.nro_cliente` `GROUP BY c.nro_cliente;`
MongoDB	`db.clientes.aggregate(` `{` `$lookup: {` `from: "facturas",` `localField: "nro_cliente",` `foreignField: "nro_cliente",` `as: "facturas"` `}` `},` `{` `$project: {` `_id: 0,` `nro_cliente: 1,` `cantidad_facturas: { $size: "$facturas" }` `}` `}` `);`

4.1.7. Listar todas las Facturas que hayan sido compradas por el cliente de nombre "Pandora" y apellido "Tate".

PostgreSQL	`SELECT f.nro_factura` `FROM e01_factura f NATURAL JOIN e01_cliente c` `WHERE nombre = 'Pandora' AND apellido = 'Tate';`
MongoDB	`db.facturas.aggregate([` `{` `$lookup: {` `from: "clientes",` `localField: "nro_cliente",` `foreignField: "nro_cliente",`

	`` as: "cliente" } }, { $match: { "cliente.nombre": "Pandora", "cliente.apellido": "Tate" } }, { $project: { _id: 0, nro_factura: 1 } }]); ```

4.1.8. Listar todas las Facturas que contengan productos de la marca "In Faucibus Inc.".

PostgreSQL	``` SELECT nro_factura FROM e01_factura WHERE nro_factura IN (SELECT nro_factura FROM e01_detalle_factura WHERE codigo_producto IN (SELECT codigo_producto FROM e01_producto WHERE marca = 'In Faucibus Inc.')); ```
MongoDB	``` db.facturas.aggregate([{ $lookup: { from: "productos", localField: "detalle_factura.codigo_producto", foreignField: "codigo_producto", as: "productos" } }, { $match: { "productos.marca": "In Faucibus Inc." } }, ```

	{ $project: { _id: 0, nro_factura: 1 } }]);

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

PostgreSQL	SELECT t.* FROM e01_cliente c RIGHT OUTER JOIN public.e01_telefono t on c.nro_cliente = t.nro_cliente;
MongoDB	db.clientes.aggregate([{ $match: { telefono: {$ne: null} } }, { $unwind: "$telefono" }, { $project: { _id: 0, nro_cliente: 1, nro_telefono: "$telefono.nro_telefono", codigo_area: "$telefono.codigo_area", tipo: "$telefono.tipo" } }]);

4.1.10. Mostrar nombre y apellido de cada cliente junto con lo que gastó en total (con IVA incluido).

PostgreSQL	SELECT c.nombre, c.apellido, coalesce(sum(f.total_con_iva), 0) as "gasto_total" FROM e01_cliente c NATURAL JOIN e01_factura f GROUP BY c.nro_cliente;

MongoDB	db.facturas.aggregate([{ $lookup: { from: "clientes", localField: "nro_cliente", foreignField: "nro_cliente", as: "cliente" } }, { $unwind: "$cliente" }, { $group: { _id: { nro_cliente: "$cliente.nro_cliente", nombre: "$cliente.nombre", apellido: "$cliente.apellido" }, gasto_total: { $sum: "$total_con_iva" } }, }, { $project: { _id: 0, nombre: "$_id.nombre", apellido: "$_id.apellido", gasto_total: 1 } }]);

4.2. Vistas

4.2.1. Se debe realizar una vista que devuelva las facturas ordenadas por fecha.

PostgreSQL	CREATE VIEW facturas_por_fecha AS SELECT * FROM e01_factura ORDER BY fecha;
MongoDB	db.createView("facturas_ordenadas_por_fecha", "facturas", [{ $sort: { fecha: 1 },

	`` }, { $project: { _id: 0 } }]); ```

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

PostgreSQL	``` CREATE VIEW productos_no_facturados AS SELECT p.* FROM e01_producto p WHERE NOT EXISTS (SELECT 1 FROM e01_detalle_factura df WHERE df.codigo_producto = p.codigo_producto); ```
MongoDB	``` db.createView("productos_no_facturados", "productos", [{ $lookup: { from: "facturas", localField: "codigo_producto", foreignField: "detalle_factura.codigo_producto", as: "facturas" } }, { $match: { facturas: [] } }, { $project: { _id: 0, codigo_producto: 1, marca: 1, nombre: 1, descripcion: 1, precio: 1, stock: 1 ```

```
        }
    }
]);
```

5. Conclusión

Este trabajo le permitió al equipo verificar las diferencias entre las bases de datos relacionales y no relacionales. La transformación del modelo de datos desde PostgreSQL hacia MongoDB fue esclarecedora y resultó de gran utilidad para consolidar los conocimientos adquiridos en la materia. Se tomaron decisiones no solo en base a la "forma" de los esquemas, sino también en base a la frecuencia esperada de la consultas y la facilidad de uso de un posible usuario de la API, lo que llevó a pensar el problema más allá del alcance de la materia, llevándolo a un plano más real.