Trabajo Práctico Obligatorio:

Base de datos II 2C 2023

Fecha de entrega: 17/11/2023

Alumnos:
Camila Sierra Pérez 60242
Martín E. Zahnd 60401
Pedro López Guzmán 60711

Docentes:
Cecilia Rodriguez Babino
Guillermo Rodriguez

Grupo 5

Índice

1. Introducción

El objetivo de este Trabajo Práctico es realizar un sistema de Facturación, llevando el control de los productos comprados por los clientes. La facturación de productos a un cliente consiste enchequear la disponibilidad en el stock de los productos, decrementar la cantidad vendida y calcular el monto total de la factura considerando el IVA y los descuentos que se aplican de acuerdo al volumen de productos que se compran. El sistema deberá proporcionar los mecanismos necesariospara dar de alta a nuevos clientes, baja y modificación de los ya existentes y dar de alta a nuevos productos y modificación de los ya existentes.

2. Trabajo realizado

2.1. Consultas requeridas

2.1.1. Postgres

1. Obtener el teléfono y el número de cliente del cliente con nombre “Wanda” y apellido “Baker”.

```
SELECT
    T.nro_telefono,
    T.nro_cliente
FROM
    E01_CLIENTE C
JOIN
    E01_TELEFONO T ON C.nro_cliente = T.nro_cliente
WHERE
    C.nombre = 'Wanda' AND
    C.apellido = 'Baker';
```

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

```
SELECT DISTINCT
    C.*
FROM
    E01_CLIENTE C
JOIN
    E01_FACTURA F ON C.nro_cliente = F.nro_cliente;
```

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

```
SELECT *
FROM E01_CLIENTE C
WHERE NOT EXISTS (
    SELECT 1
    FROM E01_FACTURA F
    WHERE C.nro_cliente = F.nro_cliente
);
```

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

```
SELECT DISTINCT P.*
FROM E01_PRODUCTO P
JOIN E01_DETALLE_FACTURA DF ON P.codigo_producto = DF.codigo_producto;
```

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

```
SELECT C.*, T.*
FROM E01_CLIENTE C
LEFT JOIN E01_TELEFONO T ON C.nro_cliente = T.nro_cliente;
```

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

```
SELECT C.*, COUNT(F.nro_factura) AS cantidad_facturas
FROM E01_CLIENTE C
LEFT JOIN E01_FACTURA F ON C.nro_cliente = F.nro_cliente
GROUP BY C.nro_cliente;
```

7. Listar todas las Facturas que hayan sido compradas por el cliente de nombre "Pandoraz apellido "Tate".

```
SELECT F.*
FROM E01_FACTURA F
JOIN E01_CLIENTE C ON F.nro_cliente = C.nro_cliente
WHERE C.nombre = 'Pandora' AND C.apellido = 'Tate';
```

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

```
SELECT DISTINCT F.*
FROM E01_FACTURA F
JOIN E01_DETALLE_FACTURA DF ON F.nro_factura = DF.nro_factura
JOIN E01_PRODUCTO P ON DF.codigo_producto = P.codigo_producto
WHERE P.marca = 'In Faucibus Inc.';
```

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

```
SELECT C.*, T.*
FROM E01_CLIENTE C
LEFT JOIN E01_TELEFONO T ON C.nro_cliente = T.nro_cliente;
```

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

```
SELECT C.nombre, C.apellido, SUM(F.total_con_iva) AS gasto_total
FROM E01_CLIENTE C
LEFT JOIN E01_FACTURA F ON C.nro_cliente = F.nro_cliente
GROUP BY C.nombre, C.apellido;
```

2.1.2. MongoDB

1. Obtener el teléfono y el número de cliente del cliente con nombre “Wanda” y apellido “Baker”.

```
db.collection("clientes").find(
    { nombre: "Wanda", apellido: "Baker" })
    .project({ _id: 0, nro_cliente: 1, telefono: 1 })
```

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

```
db.collection("clientes").aggregate([
        {
            $lookup: {
                "from": "facturas",
                "localField": "nro_cliente",
                "foreignField": "nro_cliente",
                "as": "facturas"
            }
        },
        {
            "$match": { "facturas": { "$exists": true, "$ne": [] } }
        },
        {
            "$project": { "facturas": 0 }
        }
])
```

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

```
db.collection("clientes").aggregate([
        {
            "$lookup": {
                "from": "facturas",
                "localField": "nro_cliente",
                "foreignField": "nro_cliente",
                "as": "facturas"
            }
        },
        {
            "$match": { "facturas": { "$exists": true, "$eq": [] } }
        },
        {
```

```
            "$project": { "facturas": 0 }
        }
])
```

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

```
db.collection("productos").aggregate([
        {
            "$lookup": {
                "from": "facturas",
                "localField": "codigo_producto",
                "foreignField": "detalles.codigo_producto",
                "as": "facturas"
            }
        },
        {
            "$match": { "facturas": { "$exists": true, "$ne": [] } }
        },
        {
            "$project": { "facturas": 0 }
        }

])
```

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

```
db.collection("clientes").find({})
```

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

```
db.collection("facturas").aggregate([
    {
        "$group": {
            "_id": "$nro_cliente",
            "total": { "$sum": 1 }
        }
    },
    {
        "$lookup": {
            "from": "clientes",
            "localField": "_id",
            "foreignField": "nro_cliente",
            "as": "cliente_info"
        }
    },
    {
```

```
            "$unwind": {"path": "$cliente_info", "preserveNullAndEmptyArrays": true}
        },
        {
            "$project": {
                "_id": 0,
                "cliente_info": {
                    nro_cliente: { $ifNull: ["$_id","NULL"] },
                    nombre: { $ifNull: ["$cliente_info.nombre","NULL"] },
                    apellido: { $ifNull: ["$cliente_info.apellido","NULL"] },
                    direccion: { $ifNull: ["$cliente_info.direccion","NULL"] },
                    activo: { $ifNull: ["$cliente_info.activo","NULL"] },
                    telefono: { $ifNull: ["$cliente_info.telefono","NULL"] },
                    total_facturas: "$total"
                }
            }
        }
    ])
```

7. Listar todas las Facturas que hayan sido compradas por el cliente de nombre "Pandoraz apellido "Tate".

```
db.collection("facturas").aggregate([
    {
        "$lookup": {
            "from": "clientes",
            "localField": "nro_cliente",
            "foreignField": "nro_cliente",
            "as": "cliente_info"
        }
    },
    {
        "$match": { "cliente_info.nombre": "Pandora", "cliente_info.apellido": "Tate" }
    },
    {
        "$project": { "_id": 0, "cliente_info": 0, "detalles": 0 }
    }
])
```

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

```
db.collection("facturas").aggregate([
    {
        "$lookup": {
            "from": "productos",
            "localField": "detalles.codigo_producto",
            "foreignField": "codigo_producto",
            "as": "productos"
        }
```

```
    },
    {
        "$match": { "productos.marca": "In Faucibus Inc." }
    },
    {
        "$project": { "_id": 0, "detalles": 0, "productos": 0 }
    }
])
```

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

```
db.collection("clientes").aggregate([
    {
        "$unwind": "$telefono"
    },
    {
        "$project": {
            "_id": 0,
            "telefono": "$telefono",
            "info_cliente": {
                "nro_cliente": "$nro_cliente",
                "nombre": "$nombre",
                "apellido": "$apellido",
                "direccion": "$direccion",
                "activo": "$activo",
                // "telefono": 0
            }
        }
    }
])
```

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

```
db.collection("clientes").aggregate([
    {
        "$lookup": {
            "from": "facturas",
            "localField": "nro_cliente",
            "foreignField": "nro_cliente",
            "as": "facturas"
        }
    },
    {
        "$project": {
            "_id": 0,
            "nombre": "$nombre",
            "apellido": "$apellido",
            "total_gastado": { "$sum": "$facturas.total_con_iva" }
```

```
            }
        }
    ])
```

2.2. Vistas

2.2.1. Postgres

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

```
CREATE VIEW Vista_FacturasOrdenadas AS
SELECT *
FROM E01_FACTURA
ORDER BY fecha;
```

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

```
CREATE VIEW Vista_ProductosNoFacturados AS
SELECT P.*
FROM E01_PRODUCTO P
LEFT JOIN E01_DETALLE_FACTURA DF ON P.codigo_producto = DF.codigo_producto
WHERE DF.codigo_producto IS NULL;
```

2.2.2. MongoDB

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

```
db.createCollection("facturas_ordenadas_por_fecha", { viewOn: "facturas", pipeline: [{ $s
```

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

```
db.createCollection("productos_sin_facturas", { viewOn: "productos", pipeline: [
    {
        "$lookup": {
            "from": "facturas",
            "localField": "codigo_producto",
            "foreignField": "detalles.codigo_producto",
            "as": "facturas"
        }
    },
    {
        "$match": { "facturas": { "$eq": [] } }
    },
    {
        "$project": { "facturas": 0 }
```

```
        }
    ]})
```

3. Guía de ejecución

Para correr el trabajo se debe contar con:

- Docker
- Docker compose
- Node 19
- Python (3.11.5)

Desde la carpeta root del proyecto, ejecutar el siguiente comando:

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

Si se quiere apagar, también desde la carpeta root del proyecto se debe ejecutar:

```
docker compose down
```

Para correr las queries de Mongo, debe ejecutar, desde el directorio 'queries_and_views/mongo/' el comando:

```
npm install
```

Y luego:

```
npm run queries
```

el mismo genera 10 archivos del estilo queryXoutput.json en el directorio 'mongo'.

Para crear las views:

```
npm run views
```

Para elegir que base de datos va a usar la api, hay un '.env' en la carpeta root que tiene el siguiente contenido:

```
API_DB="mongodb"
```

Para usar Postgres hay que cambiarla por:

```
API_DB="postgresql"
```

4. Estrategias utilizadas

Por empezar, para realizar la migración de PostgreSQL a MongoDB decidimos utilizar Python. Migramos los datos directamente desde la base, y no desde el archivo otorgado. Para ello fueron utilizadas dos librerías, 'psycopg' para Postgre y 'pymongo' para Mongo.

Realizamos un trabajo asincronico con tres workers, uno para los clientes y los teléfonos, otro para los productos y otro para las facturas y los detalles de las facturas. Decidimos que las facturas tengan los detalles embebidos, utilizando un array. Lo mismo hicimos con los teléfonos, dado que descubrimos que un cliente podía tener muchos números de teléfono. Finalmente decidimos que la relación entre cliente y teléfono sea Strong, dado que si se borra un cliente también debería hacerlo su número. Cada uno de los workers lo que hace es sacar una colección con los datos de Postgres y subirla a Mongo.

En cuanto a los scripts de Mongo, no contamos con archivos nativos como sí tenemos para SQL, por lo que investigamos y dado que la tecnología utilizada para hacerlos es JavaScript, optamos por utilizar este lenguaje para realizarlos. Creamos un paquete de node que tiene dos comandos:

```
npm run queries
```

para correr las queries, dejando los resultados en 'queryXoutput.json' y

```
npm run views
```

para hacer las views.

Por último, dockerizamos todo dentro de un Docker compose, dado que estabamos utilizando uno para cada base de datos. Para la parte de migración, tenemos un contenedor nuevo que lo que hace es cargar los datos a la base de Postgres y después migrarlos a Mongo. El mismo no se inicia hasta que los otros dos esten listos y una vez que confirma que la conexión con las bases esté bien, comienza la migración.

Con respecto a la API, hicimos una misma API para las dos bases de datos, por lo cual la misma debe ser compatible con ambas, y es por eso que, por ejemplo, al hacer un GET del client te devuelve el id de Mongo y el número de cliente que es la PK en Postgres, Esto se debe a que no hay garantías en mongo del que número de cliente sea Unique y un object id solamente puede ser un string de 24 caracteres hexa, entonces por eso optamos por esta solución. Por otro lado, si se levanta la misma API pero con Postgres, el id y el nro de cliente devuelven lo mismo, siendo id un string y nro de cliente un int, que es el valor de la primary key. Lo mismo aplica para el resto de los endpoints, con el objetivo de mantener la compatibilidad entre ambas bases. Como requería la consigna, se permite dar de alta, baja y modificar clientes y productos. Para las facturas, se calcula el total con y sin IVA al realizar un GET en base a los items en los detalles.

Otra decisión de diseño fue setear y configurar las bases con las siguientes variables de entorno:

API_DB, permite elegir la base de datos a utilizar entre Mongo y Postgres

$PSQL\_DB\_NAME$

$PSQL\_USER$

$PSQL\_PASSWORD$

PSQL_HOST

PSQL_PORT

MONGO_HOST

MONGO_PORT

Los endpoints de la API se configuran en el archivo 'main.py', dentro de la carpeta 'api'. En el mismo directorio hay una carpeta 'service' que recibe todos los endpoints, realiza validaciones y luego llama a la carpeta 'persistance', donde dependiendo de la base de datos escogida le pide que realice las queries. Ambas son intercambiables y no se comparten datos entre una y la otra. Dentro del directorio 'persistence' se encuentra la carpeta 'model' que contiene los modelos de la base de datos.

Las tecnologías utilizadas fueron Python (3.11.5), Psycopg, Pymongo, FastAPI, node, Mongo, Docker, Docker Compose y Postgres.