Instituto Tecnológico de Buenos Aires

Informe TPO

Base de Datos II - 2C 2023

Integrantes:

Tobias Perry 62064

Manuel Esteban Dithurbide 62057

Tomas Camilo Gay Bare 62103

Martin Hecht 62041

Abstracto

Este informe describe el desarrollo del TPO de la materia Base de Datos II - ITBA. El proyecto tiene como objetivo el desarrollo de la base de datos para un sistema integral de facturación para gestionar los productos comprados por los clientes. El sistema se encargará de controlar el stock de productos, gestionar clientes, calcular el monto total de las facturas considerando impuestos y descuentos por volumen de compra. Se implementarán consultas, vistas y una API para facilitar el acceso y la manipulación de los datos.

Arquitectura:

SQL:

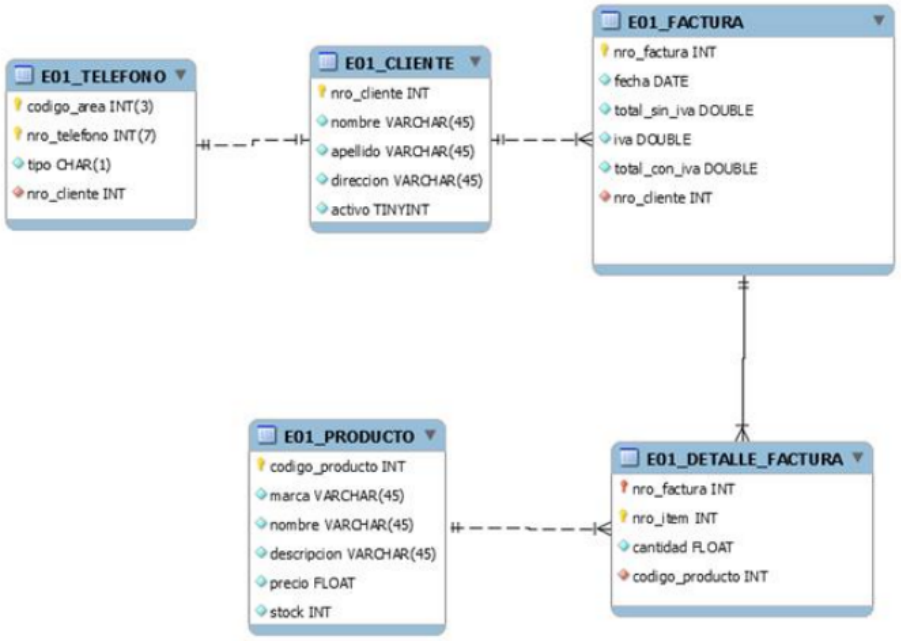

Figura 1: Diagrama DER del sistema de facturacion (Relacional/PostgreSQL)

MongoDB:

El formato de los objetos JSON en la base de datos es el siguiente.

Clientes:

```
[
  {
    "_id": "cliente_id_1",
    "nombre": "Nombre1",
    "apellido": "Apellido1",
    "direccion": "Direccion1",
    "activo": true
  },
  {
    "_id": "cliente_id_2",
    "nombre": "Nombre2",
    "apellido": "Apellido2",
    "direccion": "Direccion2",
    "activo": false
  },
//  ...
]
```

Productos:

```
[
  {
    "_id": "producto_id_1",
    "marca": "Marca1",
    "nombre": "Producto1",
    "descripcion": "Descripcion1",
    "precio": 10.99,
    "stock": 100
  },
  {
    "_id": "producto_id_2",
    "marca": "Marca2",
    "nombre": "Producto2",
    "descripcion": "Descripcion2",
    "precio": 20.50,
    "stock": 50
  },
  // ...
]
```

Teléfonos:

```
[
  {
    "codigo_area": "123",
    "nro_telefono": "456789",
    "nro_cliente": "cliente_id_1",
    "tipo": "celular"
  },
  {
    "codigo_area": "456",
    "nro_telefono": "123456",
    "nro_cliente": "cliente_id_2",
    "tipo": "residencial"
  },
  // ...
]
```

Facturas:

```
[
  {
    "_id": "factura_id_1",
    "fecha": "2023-01-01",
    "total_sin_iva": 100.0,
    "iva": 10.0,
    "total_con_iva": 110.0,
    "nro_cliente": "cliente_id_1",
    "detalles": [
      {
        "codigo_producto": "producto_id_1",
        "cantidad": 2,
        "nro_item": 1
      },
      {
        "codigo_producto": "producto_id_2",
        "cantidad": 1,
        "nro_item": 2
      }
    ]
  },
  // ...
]
```

Parte 1: Consultas SQL y MongoDB

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

SQL:

```sql
select nro_telefono, nro_cliente
from e01_telefono natural join e01_cliente
where nombre = 'Wanda' and apellido = 'Baker';
```

MongoDB:

```
db.clients.aggregate([
    {
        $lookup: {
            from: "phones",
            localField: "_id",
            foreignField: "nro_cliente",
            as: "clientPhones"
        }
    },
    {
        $match: {
            nombre: "Wanda",
            apellido: "Baker"
        }
    },
    {
        $unwind: "$clientPhones"
    },
    {
        $project: {
            _id: 0,  // excluye _id field
            client_id: "$_id",  // renombra _id to client_id
            client_name: "$nombre",  // incluye client name
            client_lastname: "$apellido",  // incluye client last name
            codigo_area: "$clientPhones.codigo_area",
            nro_telefono: "$clientPhones.nro_telefono"
        }
    }
])
```

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

SQL:

```sql
select distinct nombre, apellido, direccion, activo, nro_cliente
from e01_cliente natural join e01_factura;
```

MongoDB:

```
db.clients.aggregate([
    {
        $lookup:{
            from: "tickets",
            localField: "_id",
            foreignField: "nro_cliente",
            as: "clientsTickets"
        }
    },
    {
        $match: {
            "clientsTickets": {$ne: []}
        }
    }
])
```

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

SQL:

```sql
select distinct *
from e01_cliente cli
where cli.nro_cliente not in (select nro_cliente
                              from e01_cliente natural join e01_factura);
```

MongoDB:

```
db.clients.aggregate([
    {
        $lookup:{
            from: "tickets",
            localField: "_id",
            foreignField: "nro_cliente",
            as: "clientsTickets"
        }
    },
    {
        $match: {
            "clientsTickets": {$eq: []}
        }
    }
])
```

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

SQL:

```
select codigo_producto, marca, nombre, descripcion, precio, stock
from e01_producto
where codigo_producto in (select codigo_producto
                          from e01_detalle_factura);
```

MongoDB:

```
db.products.aggregate([
    {
        $lookup:{
            from: "tickets",
            localField: "_id",
            foreignField: "_id",
            as: "factura"
        }
    },
    {
        $match:{
            factura: {$ne: []}
        }
    },
    {
        $project:{
            nombre: 1,
            marca: 1,
            _id: 0
        }
    }
]);
```

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

SQL:

```
select distinct nro_cliente, nombre, apellido, direccion, activo, nro_telefono
from e01_cliente natural join e01_telefono;
```

MongoDB:

```
db.clients.aggregate([
    {
        $lookup: {
            from: "phones",
            localField: "_id",
            foreignField: "nro_cliente",
            as: "clientPhones"
        }
    },
    {
        $unwind: "$clientPhones"
    },
    {
        $project: {
            _id: 0,  // excluye _id field
            client_id: "$_id",  // renombra _id a client_id
            client_name: "$nombre",  // incluye client name
            client_lastname: "$apellido",  // incluye client last name
            codigo_area: "$clientPhones.codigo_area",
            nro_telefono: "$clientPhones.nro_telefono",
        }
    }
])
```

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

SQL:

```
select e01c.nro_cliente, count(distinct nro_factura) as facturas
from e01_factura full outer join e01_cliente e01c
    on e01c.nro_cliente = e01_factura.nro_cliente
group by e01c.nro_cliente;
```

MongoDB:

```
db.clients.aggregate([
    {
        $lookup:{
            from: "tickets",
            localField: "_id",
            foreignField: "nro_cliente",
            as: "clientsTickets"
        }
    },
    {
        $project:{
            nombre: 1,
            apellido: 1,
            count: {$size: "$clientsTickets"}
        }
    }
])
```

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

SQL:

```
select * from e01_factura
where nro_cliente = (select nro_cliente
                     from e01_cliente
                     where nombre = 'Pandora' and apellido = 'Tate');
```

MongoDB:

```
db.clients.aggregate([
    {
        $lookup:{
            from: "tickets",
            foreignField: "nro_cliente",
            localField: "_id",
            as: "clientTickets"
        }
    },
    {
        $match: {
            nombre: "Pandora",
            apellido: "Tate"
        }
    },
    {
        $unwind: "$clientTickets"
    },
    {
        $project: {
            nombre: 1,
            apellido: 1,
            factura_datos: {
                id: "$clientTickets._id",
                fecha_factura: "$clientTickets.fecha"
            }
        }
    }
])
```

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

SQL:

```sql
select *
from e01_factura
where nro_factura in (select nro_factura
                      from e01_detalle_factura natural join e01_producto
                      where marca = 'In Faucibus Inc.');
```

MongoDB:

```
db.tickets.aggregate([
    {
        $lookup: {
            from: "products",
            localField: "detalles.codigo_producto",
            foreignField: "_id",
            as: "ticketAndProduct"
        }
    },
    {
        $match: {
            "ticketAndProduct.marca": "In Faucibus Inc."
        }
    }
])
```

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

SQL:

```sql
select nro_cliente, codigo_area, nro_telefono, nombre, apellido, direccion, activo
from e01_telefono natural join e01_cliente;
```

MongoDB:

```
db.clients.aggregate([
    {
        $lookup: {
            from: "phones",
            localField: "_id",
            foreignField: "nro_cliente",
            as: "clientPhones"
        }
    },
    {
        $unwind: "$clientPhones"
    },
    {
        $project: {
            _id: 0,  // Exclude the _id field
            client_name: "$nombre",  // Include client name
            client_lastname: "$apellido",  // Include client last name
            codigo_area: "$clientPhones.codigo_area",
            nro_telefono: "$clientPhones.nro_telefono"
        }
    }
])
```

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

SQL:

```
select c.nombre, c.apellido, SUM(f.total_con_iva) as total
from e01_cliente c join e01_factura f on c.nro_cliente = f.nro_cliente
group by c.nro_cliente, c.nombre, c.apellido
```

MongoDB:

```
db.clients.aggregate([
    {
        $lookup: {
            from: "tickets",
            localField: "_id",
            foreignField: "nro_cliente",
            as: "clientsTickets"
        }
    },
    {
        $project: {
            _id: 0, // exluye _id en el output
            nombre: 1,
            apellido: 1,
            gasto: {$sum: "$clientsTickets.total_con_iva"}
        }
    }
])
```

Parte 2: Vistas

1. Las facturas ordenadas por fecha.

SQL:

```sql
create view facturas_ordenadas_por_fecha as
    select *
    from e01_factura
    order by fecha;
```

MongoDB:

```
db.createView(
    "facturas_por_fecha",
    "tickets",
    [{
        $sort: {
            fecha: 1
        }
    }]
)
```

2. Los productos que aún no han sido facturados.

SQL:

```sql
create view productos_no_facturados as
    select *
    from e01_producto
    where codigo_producto not in (select codigo_producto
                                  from e01_detalle_factura);
```

MongoDB:

```
db.createView(
    "productos_no_facturados",
    "products",
    [
        {
            $lookup: {
                from: "tickets",
                foreignField: "detalles.codigo_producto",
                localField: "_id",
                as: "ticketsAndProducts"
            }
        },
        {
            $match: {
                ticketsAndProducts: {$eq: []}
            }
        }
    ]
)
```

Parte 3: Guía de ejecución de los endpoints

1. Iniciar las BDs

Correr dentro de su sistema o un contenedor docker una BD postgreSQL.

- Usar la terminal de línea de comandos para descargar la versión oficial de PostgreSQL:

```
docker pull postgres
```

- Levantar el contenedor:

```
docker run --name MypostgreSQL -e POSTGRES_PASSWORD=mysecretpassword
-p 5432:5432 -d postgres
```

- Conectarse a PostgreSQL utilizando su gestor de bd de preferencia.
- Ejecutar el script ITBA_2023_esquema_facturacion.sql

Correr dentro de su sistema una BD MongoDB o también puede optar por otro contenedor docker:

- Para descargar la versión oficial de MongoDB:

```
docker pull mongo
```

- Levantar el contenedor:

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

- Para popular los documentos primero debe transferir los JSON con los datos a su docker. Desde la raiz del repositorio ejecutar:

```
docker cp ./tickets.json mongodb:/
docker cp ./clients.json mongodb:/
docker cp ./phones.json mongodb:/
docker cp ./products.json mongodb:/
```

- Luego para importar los datos corremos los siguientes comandos:

```
docker exec -it mongodb bash

mongosh

use E01

db.createUser({user: "username", pwd: "password",roles: [ { role: "readWrite", db: "E01" } ]})

db.auth("username", "password")

exit

mongoimport --db E01 --collection phones --jsonArray --file phones.json

mongoimport --db E01 --collection tickets --jsonArray --file tickets.json

mongoimport --db E01 --collection clients --jsonArray --file clients.json

mongoimport --db E01 --collection products --jsonArray --file products.json

exit
```

2. Configuración del entorno de la API

- Navegar a la ubicación BD2_TPO/api-related/config.json

- En el archivo config.json, ajustar los siguientes datos:

    - port: Este es el puerto local en el que se va a ejecutar la API, elegir a gusto.

    - postgres: Aquí hay que poner los detalles de conexión a la bd PostgreSQL.

    - mongo: Aquí hay que poner los detalles de conexión a la db MongoDB.

3. Correr la API

- Navegar al directorio e instalar npm en caso de no estar instalado

```
cd ./api-related
npm install
```

- Correr la API

```
node api.js
```

- Usar Postman para pegarle a la api ejecutando cualquiera de los siguientes endpoints:

Nota:
- {bd} = “pg” o “mongo” dependiendo de a que bd quiero mandar la request
- {puerto} = puerto donde se inició la API en el paso 3

  - GET:

```
http://localhost:{puerto}/clients/{bd}

http://localhost:{puerto}/products/{bd}
```

  - POST:

```
http://localhost:{puerto}/clients/{bd}

http://localhost:{puerto}/products/{bd}
```

    - con un json en el body con los datos del cliente/producto nuevo en el siguiente formato respectivamente:

```
-----------------
Cliente
-----------------
{
    “nro_cliente”:nro_cliente,
   “nombre”:”nombre”,
   “apellido”:”apellido”,
   “direccion”:”direccion”,
   “activo”:activo
```

```
}

-----------------
Producto
----------------
{
    “codigo_producto”: codigo_producto,
   “marca”:”marca”,
   “nombre”:”nombre”,
   “descripcion”:”descripcion”,
   “precio”:precio,
   “stock”:stock
}
```

- PUT:

```
http://localhost:{puerto}/clients/{bd}/:nro_cliente
http://localhost:{puerto}/products/{bd}/:codigo_producto
```

- nro_cliente es el número del cliente a modificar y un json en el body con los datos a cambiar del cliente

- codigo_producto es el número del producto a modificar y un json en el body con los datos a cambiar del producto

- DELETE

```
http://localhost:{puerto}/clients/{bd}/:nro_cliente

http://localhost:{puerto}/products/{bd}/:codigo_producto
```

  - nro_cliente es el número del cliente a borrar
  - codigo_producto es el número del producto a borrar

Parte 4: Estrategias utilizadas

Para llevar a cabo el trabajo práctico decidimos utilizar distintas herramientas y tecnologías para probar y desarrollar las consultas/API.

Para correr la BD decidimos utilizar el contenedor de docker de Mongo y de Postgres para facilitar y evitar la instalación de estos programas localmente en nuestras computadoras.

Luego, para desarrollar la API lo hicimos en JavaScript vainilla creando los endpoints usando la biblioteca Express. Luego, para probar estos endpoints utilizamos la aplicación Postman para asegurarnos que funcionen correctamente y devuelvan lo que deberían.

Al migrar los datos a Mongo, tuvimos que tomar decisiones sobre cómo embeber los datos. Decidimos embeber la información de la tabla factura y meterla toda en el archivo de detalle_factura. De esta manera, cada ítem de detalle_factura pertenece a una factura y dentro del mismo archivo está la información de la factura referenciada.