Nicolas Ballén - 202310273
Maria Juliana Ballesteros - 202313216
Mauricio Martínez - 202314461

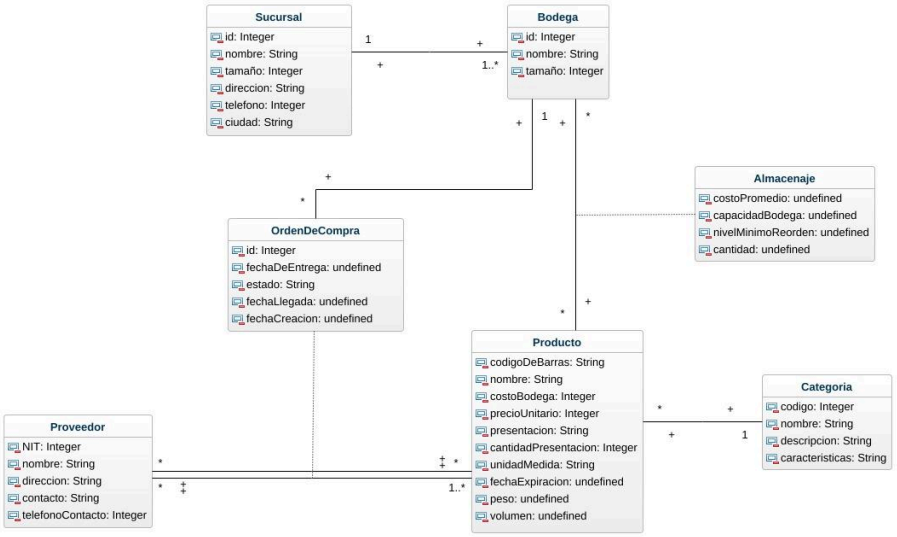
Proyecto Sistrans Entrega 3-Grupo C7

Contexto:

Tras la modelación del problema SuperAndes en bases de datos relacional (SQL), esta entrega nos propuso la solución del mismo problema pero con una BD no relacional (NoSQL) para facilitar el ingreso de datos y optimizar en pos de las consultas/modificaciones presupuestadas. Con ello en mente, a continuación se detallarán las consideraciones tenidas durante el desarrollo del nuevo proyecto, su funcionamiento, y finalmente los resultados de los escenarios de prueba.

1. Análisis y modelo conceptual

En primer lugar, hicimos las modificaciones correspondientes al diagrama UML. Ante la falta de ciertos requerimientos, como lo serían el manejo de registros de ingreso o de almacenamiento de ciudades, redujimos el modelo UML al siguiente:



—Resumen de Modificaciones

- Eliminación de entidad Ciudad: la nueva versión no requiere consulta o creación de ciudades, por lo que se redujo a un atributo de sucursal
- Eliminación Registro de Ingreso: Esto a falta de búsquedas o acciones que lo necesiten.

—Resumen de Entidades:

- **Sucursal:** No sufrió cambios considerables a excepción del atributo de ciudad. Resguarda un id, nombre, tamaño, dirección, y teléfono además. Tiene relación uno a muchos con Bodega.

- **Bodega:** Mantiene su estructura con un id, nombre y tamaño y su relación muchos a uno con sucursal.
- **Proveedor**: Mantiene su estructura de id, nombre e info de contacto, manteniendo una relación uno a muchos con producto (productos que ofrece), así como con órdenes de compra (múltiples solicitudes de compras a un proveedor).
- **Producto**: Mantiene su estructura desde código de barras hasta volumen por unidad. Mantiene la relación muchos a uno con categoría, muchos a uno con proveedor, y mucho a muchos con órdenes de compra.
- **Categoría**: Mantiene su estructura de código, descripción y características. Relación uno a muchos productos.
- **Órdenes de Compra**: Nuevamente, mantuvo su estructura, manteniendo una relación muchos a muchas con sucursal, producto, y muchas a uno con proveedor.
- **Almacenaje:** Reflejo de la información de inventario por producto en cada bodega, conserva la información de cantidad, capacidad, costo promedio, y nivel mínimo por producto en la bodega correspondiente de su relación muchos a uno con esta.

2. **Diseño de BD:**

Análisis de Carga de Trabajo:

-Cantidades de Entidades:

- **Sucursales:** 150
- **Bodegas:** 900 entre las diferentes sucursales
- **Proveedores**: 10000
- **Productos**: 20000 distribuidos en diferentes proveedores
- **Órdenes:** 200000 en un plazo de 3 años con estimado de 70mil al año.
- **Almacenajes**: x número de productos por bodega (mínimo 900).

-Operaciones:

Para diseñar nuestra base de datos, primero cuantificamos las diferentes acciones pedidas dentro de los requisitos funciones, denotando aquellas entidades necesarias, que información de las mismas vamos a necesitar y con qué frecuencia van a necesitar.

Tabla de Operaciones (Anexo A+B)

Entidades	Operación	Información necesaria	Tipo	tasa
Sucursal	Crear una sucursal	Información de la sucursal	Escritura	Creación: 1 vez/mes Consulta: 1 vez/semana
Sucursal, Bodega	Crear/borrar una bodega	Identificación de la sucursal + información de la bodega	Escritura	Creación: 1 vez/mes

				Consulta: 1 vez/semana
Proveedores	Crear/actualizar proveedor	Información del proveedor	Escritura	Creación: 10 veces por día Consulta: 200 veces/día
Categoría	Crear una categoría	Información de la categoría	Escritura/lectura	NO definido
Producto	Crear un producto	Información del producto	Escritura/lectura	1000 veces/día
Orden, Producto	Crear una orden	Información de la orden de compra + identificación de los productos + precios + cantidades	Escritura	200 vez/día
Orden, Producto	Leer una orden	Encabezado + detalles orden	Lectura	1000 veces/día
Producto, Categoría	Mostrar los productos que cumplen ciertas características	Información de los productos	Lectura	2000 veces/día
Sucursal, Almacenaje	Inventario de productos de una sucursal	Bodega + nombre productos + cantidad actual + cantidad mínima + costo promedio	Lectura	Modificación: 10000 veces/día Consulta: 2000 veces/día

Colecciones Escogidas:

Tras analizar los datos necesarios en conjunto, reducimos las entidades a las siguientes colecciones (*esquemas anexados en el repositorio en docs/EsquemasValidacion*):

- **Sucursales:** Sus documentos mantienen los atributos de la entidad, más agrega un atributo de ‘bodegas’ tipo arreglo que modela una relación embebida uno a muchos con bodegas (id de bodega reemplazado por número).
- **Categorías:** Sus documentos mantienen los mismos atributos de la entidad.
- **Proveedores**: Sus documentos mantienen los mismos atributos más un atributo de ‘productos’ tipo arreglo que guarda identificadores en una relación de referencia uno a muchos con productos.
- **Productos:** Sus documentos mantienen los atributos, más un atributo de ‘categoría’ tipo int que sirve a una relación de referencia muchos a uno con categoría.

- **Inventarios:** Representa la información de almacenajes indexada por la sucursal y bodega correspondiente.
  Atributos Documento:
  - ☐ Sucursal: int referenciando a una sucursal
  - ☐ Bodega: int referenciando a una bodega que pertenece a la sucursal
  - ☐ Inventarios: Arreglo con los datos de almacenaje por producto. Cada doc. embebido lo identifica un producto, la cantidad que hay, la capacidad, nivel mínimo, y costo promedio.

  (Más allá de lo indicado, estos docs. embebidos no sirven a una relación. En su lugar es más una agrupación de las entidad Almacenaje)

- **EXTRA**: Secuencia: Esta colección no sirve al escenario problema de SuperAndes. En su lugar regula los ids tipo int que usamos con cada colección para generarlos automáticamente (menos el NIT de los proveedores a ser necesario un input humano). Los docs. solo cuentan con un id tipo string que distingue la secuencia de cada colección real, y un int 'sec' que aumenta cada que se intenta agregar un documento a la colección correspondiente (siendo el valor aumentado el nuevo _id). Dados los tamaño de prueba de este problema (hasta los 3 años presupuestados), los ids son de tipo int y crecientes, siendo suficientes números diferentes para dicho tamaño (máximo 200000 con las órdenes de compra).

Análisis selección de esquema de asociación (Referenciado vs Embebido)

La decisión de usar documentos embebidos o referenciados depende de las consultas que se van a realizar, la frecuencia con la que se harán, y el tamaño de la información de los documentos participes en dichas consultas/operaciones. Por lo mismo explicaremos a continuaciones las razones y modelamiento de cada relación mencionada previamente.

Sucursal - Bodegas(One-To-Many): La relación entre sucursal y bodega se representa como un documento embebido, ya que, cada sucursal tiene bodegas **embebidas** con su respectivo id. Para tomar esta decisión se tuvieron en cuenta los siguiente saspectos.

- **Simplicidad:** Al tener las bodegas embebidas dentro de sucursal, el modelo de datos se mantiene simple y organizado. Se evita que exista una bodega que pertenezca a más de una sucursal (algo imposible), ya que las bodegas de una sucursal estarán dentro del mismo documento.
- **Go-Together/Individualidad:** Las bodegas están estrechamente relacionadas con su sucursal. Una bodega no podría existir sin una sucursal.
- **Atomicidad/Crecimiento**: Al consultar la información de una sucursal es necesario mostrar la información de sus bodegas. Embediendo las bodegas se puede obtener toda esta información de forma simple. Súmese a ello la poca frecuencia con las que se revisan o modifican ambas, por lo cual los tamaños de bodegas no llegaran a ser un impacto significativo una vez establecidas (mucho menos con solo 6 por sucursal aproximadamente, y con tan poca info. repartida en solo 3 atributos básicos).
- **Actualización:** En este caso, las bodegas y la sucursal suelen actualizarse juntas(registrar o actualizar una bodega), por lo que tener las bodegas embebidas simplifica las operaciones de escritura.

—Representacion:

Sucursal	**Bodega**
{... ‘bodegas’:array {items:bodega} …}	**{numero:int tamaño:int nombre:string }**

Proveedor-Productos (One-to-Many): Cada proveedor tiene su listado de ids de productos que **referencian** los productos que ofrecen. Ello se escogió así por lo siguiente:

- **Duplicación de Información/Tamaño/Actualización:** Un enfoque embebido daría lugar a una gran duplicación de información, siendo que varios proveedores pueden ofrecer el mismo producto. Con la referenciación evitamos este problema. Súmale a ello las complicaciones de actualizar un mismo producto en todos los proveedores.
- **Individualidad fallida:** Un producto existe sin proveedor así como proveedor sin productos (no tiene sentido pero puede existir en caso de no poder suministrar más dicho producto por ejemplo). Mantiene una relación de agregación, no complementación. Por ende no tiene sentido un enfoque embebido.
- **Atomicidad**: Las consultas sobre el proveedor son frecuentes, más no las de producto. El embebido implicaría modificaciones constantes en una entidad que no lo necesita. Súmese a ello poder modificar un producto sin la necesidad de modificar el proveedor y viceversa.

—Representacion:

Proveedor	**Producto**
{... productos:array {items:[id1, id2, id3]} …}	**{... _id:id1 … } {... _id:id2 … } {... _id:id3 … }**

Producto-Categoria(Many-To-One): Los productos hacen **referencia** a una categoría, como varios productos pueden tener la misma categoría no hay necesidad de que esta esté embebida dentro de cada producto.

- **Duplicación de datos:** Dado que varios productos pueden tener la misma categoria, embebiendo la categoría dentro de cada producto se duplicará innecesariamente la información.
- **Complejidad de actualización:** Como muchos productos pueden compartir una misma categoría, si esta está referenciada es más fácil actualizar su información.
- **Cardinalidad:** Varios productos pueden tener la misma categoria, si categoria estuviera embebido en productos el tamaño de los documentos podría volverse demasiado grande.
- **Individualidad:** Una categoría no depende de un producto, por lo que es mejor referenciar dentro de este

—Representación:

Producto	**Categoría**
{... **categoria:id1** **…}**	**{...** **_id:id1** **…** **}**

Órdenes-Proveedor (Many to one): La orden referencia a un único proveedor con el atributo de tipo int del mismo nombre. Consideramos esta referencia por:

- **Duplicación de datos/Actualización:** A un proveedor le pueden hacer múltiples órdenes, escalando el tamaño excesivamente. Ello peor siendo que los proveedores tienen presupuestados modificaciones frecuentes, aumentando mucho el costo de trabajo de mantener actualizadas las órdenes.
- **Individualidad**: No existe orden de compra sin proveedor, pero si proveedores sin órdenes. No hay razón para resguardar la totalidad del proveedor.
- **Atomicidad:** Al consultar la orden no se requieren de todos los datos del proveedor.

—Representación:

órdenes	**Proveedor**
{... **proveedor:id1** **…}**	**{...** **_id:id1** **…** **}**

Órdenes-Sucursal (Many to one): La orden referencia a una única sucursal con el atributo de tipo int del mismo nombre. Consideramos esta referencia por:

- **Duplicación de datos/Atomicidad:** A una sucursal le pueden destinar múltiples órdenes, escalando el tamaño excesivamente.
- **Individualidad**: No existe orden de compra sin sucursal, pero si sucursales sin órdenes. No hay razón para resguardar la totalidad del proveedor.
- **Atomicidad:** Si bien las sucursales poco se revisan o cambian, su información es poco relevante para levantar la orden, siendo esta una colección que si se revisa mucho.

—Representación:

Órdenes	**Sucursal**
{... **sucursal_destino:id1** **…}**	**{...** **_id:id1** **…** **}**

Órdenes-Productos (Many to Many): La orden contiene un arreglo de datos de productos. Cada dato define las cantidades y precios, así como un id que **referencia** al producto. Ello dado que:

- **Duplicación de datos/Actualización:** Nuevamente, varias órdenes a diferentes sucursales pueden pedir los mismos productos, duplicando información y dificultando su actualización.
- **Individualidad**: Los productos no dependen de una orden de compra.
- **Atomicidad:** Al consultar la orden no se requieren de todos los datos del producto, siendo que lo único que importa es la cantidad y precio, ambos datos definidos por el usuario y ya presentes dentro del documento..

—Representación:

Órdenes	**Producto**
{... **productos:array{items:** **[{producto:id1, cantidad:2, precio: 4}]** **}** **…}**	**{...** **_id:id1** **…** **}**

Escenarios de Prueba:

-Consultas en Violación de Esquemas de Validacion:

Con tal de verificar el funcionamiento de los esquemas, se nos solicita probar la creación de proveedores y órdenes violando las reglas de validación. Para ello ejecutamos los siguientes script en Mongo Shell con atributos faltantes:

- Proveedor:

  db.proveedores.insertOne({tel_contacto:1})

```
  operatorName: 'required',
  specifiedAs: {
    required: [
      'nombre',
      'direccion',
      'contacto',
      'tel_contacto',
      'productos'
    ]
  },
  missingProperties: [
    'contacto',
    'direccion',
    'nombre',
    'productos'
  ]
```

- Orden:

  db.ordenes.insertOne({sucursal_destino:1})

```
    operatorName: 'required',
    specifiedAs: {
      required: [
        'estado',
        'fecha_estimada',
        'fecha_creacion',
        'proveedor',
        'sucursal_destino',
        'productos'
      ]
    },
    missingProperties: [
      'estado',
      'fecha_creacion',
      'fecha_estimada',
      'productos',
      'proveedor'
    ]
```

-Población de Colecciones:

Con tal de probar los requerimientos en Postman, poblamos con esta misma herramienta cada colección siguiendo las reglas de integridad de las referencias. Las pruebas se encuentran adjuntas en el repositorio del proyecto en la carpeta docs. Como resultado de esta poblaciones, quedamos con:

- 5 categorías de productos
- 25 sucursales, cada una con 5 bodegas
- 10 órdenes de compra, cada una con 3 productos
- 10 proveedores,cada uno ofreciendo 3 productos, a excepción del décimo que ofrece 10
- 50 productos
- 5 registros de inventario o almacenamiento, cada uno con datos de 5 productos diferentes.

Estos datos servirán para evidenciar el funcionamiento de las consultas y demás requisitos.