Proyecto BBDD NoSQL

Nuestro proyecto se trata de un sistema de **gestión de stock de agro insumos**.

El mayor problema a largo plazo es la Escalabilidad, por lo tanto buscamos una BBDD (Cassandra) con las siguientes características
Escrituras Concurrentes: Cassandra maneja grandes volúmenes de escrituras rápidas (por ejemplo, actualizaciones de stock) sin afectar el rendimiento.
Desnormalización: Se optimizan las consultas duplicando datos en tablas especializadas, lo que permite que las lecturas sean rápidas
Escalabilidad Horizontal: Cassandra escala añadiendo nodos, lo que permite manejar un aumento en el volumen de productos, almacenes y consultas simultáneas. Posibilidad de aumentar la cantidad de usuarios/sucursales sin perder calidad

Esquema de base de datos de Cassandra:

Es un híbrido entre un almacén de claves y valores y un almacén tabular. Este esquema se caracteriza por:

Almacenamiento de datos: Los datos se almacenan en filas organizadas en familias de columnas o tablas. Cada fila tiene una clave principal que la identifica y divide los datos.

Estructura de datos: Cada clave corresponde a un valor que es un objeto, y cada clave tiene valores como columnas. Las columnas se agrupan en sets llamados familias de columnas.

Arquitectura Cassandra tiene una arquitectura sin maestro, por lo que todos los nodos actúan como iguales. Los datos se distribuyen automáticamente entre los nodos que se encuentran en un cluster. Esto nos parece muy importante si la empresa crece, siempre tendremos los datos a resguardo

Lenguaje de consulta: Cassandra utiliza CQL (Cassandra Query Language), un lenguaje de consulta similar a SQL.

Algunas empresas que utilizan Cassandra son: eBay, Reddit, Spotify, Uber, Cisco, Twitter, Walmart, Adobe

MODELADO DE DATOS

1. Productos de Agro Insumos

- **Descripción**: El catálogo de productos debe incluir información detallada sobre cada insumo agrícola, como el tipo de producto (fertilizante, pesticida, semillas, etc.), su composición química, volumen de envase, fecha de vencimiento, y si requiere almacenamiento especial.

Atributos:

- **producto_id**: Identificador único del producto (UUID).
- **nombre**: Nombre del producto (por ejemplo, “Fertilizante NPK 20-20-20”).
- **tipo**: Tipo de agro insumo (fertilizante, pesticida, semilla, herramienta).
- **categoria**: Categoría del producto (químico, biológico, orgánico, maquinaria).

- **composicion**: Composición química o biológica (por ejemplo, "NPK 20-20-20" o "Glifosato 480 g/L").
- **unidad_medida**: Unidad de medida del producto (kg, litros, unidades).
- **precio**: Precio del producto por unidad.
- **fecha_vencimiento**: Fecha de vencimiento o caducidad para productos perecederos.
- **almacenamiento_especial**: Requiere condiciones especiales de almacenamiento (temperatura, ventilación).

```
CREATE TABLE productos (
    producto_id UUID,
    nombre text,
    tipo text,
    categoria text,
    composicion text,
    unidad_medida text,
    precio decimal,
    fecha_vencimiento date,
    almacenamiento_especial boolean,
    PRIMARY KEY (producto_id)
);
```

2. Almacenes y Ubicaciones

- **Descripción**: En un proyecto de agro insumos, los productos se almacenan en distintos almacenes (por región, por tipo de producto, etc.). Se necesita una tabla para registrar los almacenes y su ubicación geográfica.

Atributos:

- **almacen_id**: Identificador único del almacén.
- **nombre_almacen**: Nombre del almacén o centro de distribución.
- **ubicacion**: Ubicación del almacén (región, ciudad).
- **capacidad_maxima**: Capacidad máxima del almacén en términos de volumen o cantidad de productos.

```
CREATE TABLE almacenes (
    almacen_id UUID,
    nombre_almacen text,
    ubicacion text,
    capacidad_maxima int,
    PRIMARY KEY (almacen_id)
);
```

3. Inventario de Agro Insumos en Almacenes

- **Descripción**: Cada producto en cada almacén tendrá un nivel de stock actual que debe mantenerse actualizado con cada movimiento de entrada o salida.

Atributos:

- **producto_id**: Identificador del producto.
- **almacen_id**: Identificador del almacén.
- **cantidad_actual**: Cantidad de stock actual en el almacén.

- **min_stock**: Cantidad mínima recomendada antes de activar una alerta de reabastecimiento.

```
CREATE TABLE inventario (
    producto_id UUID,
    almacen_id UUID,
    cantidad_actual int,
    min_stock int,
    PRIMARY KEY (producto_id, almacen_id)
);
```

Consulta: Obtener el stock actual de un agro insumo en un almacén específico.

```
SELECT cantidad_actual FROM inventario WHERE producto_id = ? AND almacen_id = ?;
```

4. Historial de Movimientos de Stock (Entradas y Salidas)

- **Descripción**: Para cada movimiento de stock (como una compra de insumos por parte de los agricultores o un reabastecimiento de almacenes), se debe mantener un historial detallado para rastrear entradas y salidas.

Atributos:

- **producto_id**: Identificador del producto.
- **almacen_id**: Identificador del almacén.
- **fecha**: Fecha y hora del movimiento.
- **tipo_movimiento**: Tipo de movimiento (venta, reabastecimiento, devolución).
- **cantidad**: Cantidad de productos involucrados en el movimiento.
- **usuario_responsable**: El usuario responsable del movimiento.

```
CREATE TABLE historial_movimientos (
    producto_id UUID,
    almacen_id UUID,
    fecha timestamp,
    tipo_movimiento text,  -- 'venta', 'reabastecimiento', 'devolución'
    cantidad int,
    usuario_responsable text,
    PRIMARY KEY (producto_id, fecha)
) WITH CLUSTERING ORDER BY (fecha DESC);
```

Consulta: Obtener el historial de movimientos de un producto en un almacén específico.

```
SELECT * FROM historial_movimientos WHERE producto_id = ? ORDER BY fecha DESC;
```

5. Gestión de Pedidos (Ordenes de Compra o Venta)

- **Descripción**: Se puede almacenar información sobre las órdenes de compra de agro insumos por parte de los agricultores o las ventas a distribuidores. Aquí se gestionan los pedidos y se registran las cantidades solicitadas y despachadas.

Atributos:

- **orden_id**: Identificador único de la orden.
- **cliente_id**: Identificador del cliente o agricultor.
- **producto_id**: Producto pedido.
- **fecha_pedido**: Fecha en la que se realizó el pedido.
- **cantidad_pedida**: Cantidad de productos solicitados.
- **estado**: Estado del pedido (pendiente, completado, cancelado).

```
CREATE TABLE ordenes_compra (
    orden_id UUID,
    cliente_id UUID,
    producto_id UUID,
    fecha_pedido timestamp,
    cantidad_pedida int,
    estado text,  -- 'pendiente', 'completado', 'cancelado'
    PRIMARY KEY (orden_id)
);
```

6. Alertas de Bajo Stock

- **Descripción**: Un sistema de alertas puede ayudar a identificar productos con bajo stock en tiempo real para iniciar el proceso de reabastecimiento.

Atributos:

- **producto_id**: Identificador del producto.
- **almacen_id**: Identificador del almacén.
- **cantidad_actual**: Cantidad actual de stock que está por debajo del mínimo.
- **fecha_alerta**: Fecha en la que se generó la alerta.

```
CREATE TABLE alertas_bajo_stock (
    producto_id UUID,
    almacen_id UUID,
    cantidad_actual int,
    fecha_alerta timestamp,
    PRIMARY KEY (producto_id, fecha_alerta)
);
```

Consulta: Obtener todos los productos con bajo stock para un almacén en particular.

```
SELECT * FROM alertas_bajo_stock WHERE almacen_id = ?;
```

7. Clientes (Agricultores/Distribuidores)

- **Descripción**: Para gestionar las ventas a los agricultores o distribuidores, se necesita una tabla que almacene la información de los clientes.

Atributos:

- **cliente_id**: Identificador único del cliente.
- **nombre_cliente**: Nombre del agricultor o distribuidor.
- **ubicacion**: Ubicación del cliente.
- **contacto**: Información de contacto.

```
CREATE TABLE clientes (
    cliente_id UUID,
    nombre_cliente text,
```

```
    ubicacion text,
    contacto text,
    PRIMARY KEY (cliente_id)
);
```

8. Instalaciòn y sus pasos

1. Instalación de Docker: Se descarga el programa desde la página oficial y se instala siguiendo el wizard https://www.docker.com/products/docker-desktop/
2. Instalación de Cassandra con el comando “Docker pull cassandra:latest”

```
C:\Users\Facun>docker pull cassandra:latest
latest: Pulling from library/cassandra
7478e0ac0f23: Downloading [==============>                                    ]  8.719MB/30.44MB
90a925ab929a: Downloading [============================================>      ]  11.48MB/12.87MB
7d9a34308537: Downloading [============>                                      ]  11.49MB/47.28MB
80338217a4ab: Waiting
1a5fd5c7e184: Waiting
a3b96c66dc7e: Waiting
213adabb5d23: Waiting
7875f1dc8197: Waiting
fb1bc1159cf2: Waiting
608923faa1bd: Waiting
```

3. Utilizamos el comando Docker run -p con los puertos correspondientes para dar de alta la base de datos.
4. Generamos la conexión a la base de datos con docker exec -it *identificador* bash y luego cqlsh.

```
C:\Users\Facun>docker ps
CONTAINER ID   IMAGE              COMMAND                  CREATED         STATUS          PORTS
                                                                                              NAMES
335f9bdd6235   cassandra:latest   "docker-entrypoint.s…"   4 minutes ago   Up 21 seconds   0.0.0.0:7000-7001->7000-7001/
tcp, 0.0.0.0:7199->7199/tcp, 0.0.0.0:9042->9042/tcp, 0.0.0.0:9160->9160/tcp   cassandra

C:\Users\Facun>docker exec -it 335f9bdd6235 bash
root@335f9bdd6235:/# cqlsh
Connected to Test Cluster at 127.0.0.1:9042
[cqlsh 6.2.0 | Cassandra 5.0.0 | CQL spec 3.4.7 | Native protocol v5]
Use HELP for help.
cqlsh>
```

5. Creamos un keyspace con create keyspace nombre_bbdd with replication = {'class': 'SimpleStrategy', 'replication_factor': 1};
6. Usamos el keyspace con Use nombre_bbdd;
7. Generamos la consulta para crear la tabla “almacén”.

```
cqlsh> create keyspace test with replication={'class':'SimpleStrategy','replication_factor':1};
cqlsh> desc keyspaces;

system       system_distributed  system_traces  system_virtual_schema
system_auth  system_schema       system_views   test

cqlsh> use test
   ... uste test;
Improper use command.
cqlsh> use test;
cqlsh:test> create table almacen (almacen_id UUID, nombre_almacen text, ubicacion text, capacidad_maxima int, PRIMARY KE
Y (almacen_id));
cqlsh:test> desc almacen

CREATE TABLE test.almacen (
    almacen_id uuid PRIMARY KEY,
    capacidad_maxima int,
    nombre_almacen text,
```

8. Generamos el primer insert dentro de la tabla, nótese que no se pasa Id porque este es una clave autogenerada.

```
cqlsh:test> INSERT INTO almacen (almacen_id, nombre_almacen, ubicacion, capacidad_maxima) VALUES (uuid(), 'Almacen Centr
al', 'Ciudad A', 1000);
cqlsh:test>
```

9.
10. Para testear la base de datos, generamos un select *

```
cqlsh:test> SELECT *
        ... FROM almacen
        ... ;

 almacen_id                           | capacidad_maxima | nombre_almacen  | ubicacion
--------------------------------------+------------------+-----------------+-----------
 79fffa11-d48b-4b53-bc9c-a777f119cc97 |             1000 | Almacen Central |  Ciudad A

(1 rows)
cqlsh:test>
```