Ingeniería de Datos II - Trabajo Práctico Obligatorio

- **Título**: Desarrollo de Sistema de Gestión Políglota para una Plataforma de Comercio Electrónico

- **Profesor**: Salas Joaquin

- **Alumnos**: Coy Dafne (1177499), El Bacha, Nasser Abdel Malek (1124482) y Sismonda, Kevin Esteban Yoel (1155119).

- **Institución**: UADE (Universidad Argentina De la Empresa)

- **Turno**: Turno noche - Cursada de verano

- **Fecha de entrega**: 20 de febrero de 2025

- **Materia**: Ingeniería de datos II

- **Grupo**: 02

Desarrollo de Sistema de Gestión Políglota para una Plataforma de Comercio Electrónico

Objetivo: Construir una aplicación para una plataforma de comercio electrónico que gestione **la creación de pedidos, carritos de compras, conversión de carritos a pedidos, finalización de pedidos, generación de facturas, registro de pagos y gestión de usuarios con cuenta corriente.** Se enfoca en la persistencia políglota utilizando diferentes tipos de bases de datos NoSQL.

Parte 1: Definición de Modelos de Bases de Datos (BD):

1. **Justificación:**
   - Explique la elección de modelos de BD para cada componente del sistema (usuarios, carritos, pedidos, facturas, pagos, catálogo de productos).
2. **Modelado Físico:**
   - Presente el modelo físico de la estructura de cada BD utilizada, considerando la optimización para consultas y transacciones.

Parte 2: Desarrollo de la Aplicación: Desarrollar una aplicación que permita realizar las siguientes acciones:

1. **Autenticación y Sesión de Usuarios:**
   - Guardar y recuperar la sesión del usuario conectado junto con su actividad.
2. **Categorización de Usuarios:**
   - Registrar la actividad de los usuarios y determinar su categorización (TOP, MEDIUM, LOW).
3. **Gestión de Carritos de Compras:**
   - Agregar, eliminar o cambiar productos y/o cantidades en un carrito de compras.
   - Guardar, recuperar y volver a estados anteriores en las acciones realizadas sobre un carrito de compras activo.
4. **Conversión de Carrito a Pedido:**
   - Convertir el contenido del carrito de compras en un pedido con detalles del cliente, importes, descuentos e impuestos.
5. **Facturación y Registro de Pagos:**
   - Facturar el pedido y registrar el pago indicando la forma de pago.
6. **Control de Operaciones:**
   - Llevar el control de operaciones de facturación y pagos realizados por los usuarios.
7. **Catálogo de Productos:**
   - Mantener un catálogo de productos con información atractiva para mejorar la experiencia de compra.
8. **Lista de Precios:**
   - Mantener una lista de precios actualizada para las ventas.
9. **Registro de Cambios en el Catálogo:**
   - Llevar un registro de todas las actividades realizadas sobre el catálogo de productos.

Justificación de Elección de Bases de Datos

1. MongoDB → Usuarios, Categorización y Catálogo de Productos

Elegimos usar MongoDB para gestionar usuarios, categorización y el catálogo de productos porque es una base de datos documental que ofrece flexibilidad en la estructura de datos, permitiendo almacenar información semiestructurada de manera eficiente.

- Usuarios y Autenticación: MongoDB nos permite manejar datos de autenticación de manera eficiente sin necesidad de una estructura rígida. Su capacidad de indexación facilita búsquedas rápidas de usuarios, y el uso de agregaciones nos permite analizar datos para diferentes funcionalidades.

- Categorización de Usuarios: Para clasificar usuarios en TOP, MEDIUM y LOW, MongoDB es ideal porque permite almacenar información sobre compras y actividad en un solo documento, facilitando consultas eficientes para determinar la categoría del usuario.

- Catálogo de Productos: MongoDB es excelente para almacenar información sobre productos, incluyendo nombre, descripción y categorías. Su flexibilidad nos permite modificar la estructura sin afectar el rendimiento.

Podríamos haber usado Neo4j para representar relaciones entre usuarios o entre productos (por ejemplo, recomendaciones basadas en compras anteriores), pero dado que no necesitamos un análisis de relaciones complejo, su uso no era necesario. También consideramos Redis, pero como el catálogo de productos es información persistente y no temporal, no era la mejor opción.

2. Redis (Sesión de Usuario, Carrito de Compras y Lista de Precios)

Elegimos usar Redis porque es una base de datos clave-valor extremadamente rápida, ideal para manejar datos volátiles y consultas de baja latencia. La velocidad de Redis es esencial para proporcionar una experiencia de usuario fluida en operaciones que requieren acceso inmediato.

- Sesión de Usuario: Redis nos permite almacenar información sobre la sesión activa del usuario, como su último inicio de sesión y actividad reciente. Dado que estos datos no necesitan persistencia a largo plazo, Redis es una excelente opción por su rapidez.

- Carrito de Compras: Redis es ideal para gestionar carritos de compras porque estos cambian constantemente y necesitan actualizaciones en tiempo real. Su capacidad para realizar operaciones atómicas permite agregar y eliminar productos de manera eficiente. Además, su función de expiración automática nos ayuda a eliminar carritos abandonados sin necesidad de procesos adicionales.

- Lista de Precios: Como los precios de los productos pueden cambiar frecuentemente y deben ser accesibles de forma inmediata, Redis es una opción óptima. Su

velocidad permite realizar actualizaciones y consultas en milisegundos sin afectar el rendimiento del sistema.

Podríamos haber usado MongoDB para almacenar sesiones o carritos, pero no es tan eficiente en operaciones en tiempo real debido a su modelo de almacenamiento basado en documentos. También consideramos Cassandra, pero su enfoque en datos estructurados y su optimización para escrituras masivas lo hacen menos adecuado para datos altamente dinámicos como carritos o sesiones.

3. Cassandra (Pedidos, Facturación, Registro de Pagos y Registro de Cambios en el Catálogo)

Elegimos usar Cassandra porque está diseñado para manejar grandes volúmenes de datos, altas tasas de escritura y garantizar disponibilidad. Como estos datos deben ser persistentes y consultados en orden cronológico, Cassandra es una opción sólida.

- Pedidos: Al convertir un carrito en pedido, Cassandra permite registrar datos del usuario, productos comprados, precios, descuentos e impuestos de manera eficiente. Su capacidad para manejar escalabilidad horizontal y escrituras rápidas lo hace ideal para un alto volumen de pedidos.

- Facturación: Cassandra es excelente para almacenar facturas porque maneja bien las series temporales, lo que facilita consultas por fecha y garantiza la integridad de los datos financieros. Como las facturas deben ser persistentes, necesitamos una base de datos con alta durabilidad y resistencia a fallos.

- Registro de Pagos: Cada pago debe guardarse junto con su fecha, monto y método de pago. Cassandra es ideal porque prioriza la disponibilidad y tolerancia a fallos, asegurando que ninguna transacción se pierda.

- Registro de Cambios en el Catálogo: Para llevar un historial de modificaciones en el catálogo de productos, Cassandra es útil porque permite almacenar grandes volúmenes de datos sin comprometer el rendimiento. Cada cambio en un producto (como actualización de precios o descripciones) se almacena con una marca de tiempo, permitiendo auditorías eficientes.

Podríamos haber usado MongoDB para pedidos y facturas, pero no maneja escrituras masivas tan eficientemente como Cassandra. Redis no era una opción viable porque no está diseñado para almacenamiento de largo plazo, y Neo4j no tenía sentido, ya que no necesitamos analizar relaciones entre pedidos y pagos.

Conclusión

Nuestra elección de bases de datos responde a la necesidad de equilibrar velocidad, persistencia, escalabilidad y eficiencia en consultas. Usamos MongoDB para datos estructurados y flexibles, Redis para datos volátiles y de acceso rápido, y Cassandra para almacenamiento robusto de datos transaccionales.