

Evaluación del módulo

Proyecto: Bases de Datos

Lección 1: Tecnologías de Base de Datos

Objetivo

Seleccionar y justificar las tecnologías de bases de datos que formarán parte de la solución.

1. Análisis comparativo entre bases de datos relacionales y NoSQL

Característica	Relacionales (SQL)	No Relacionales (NoSQL)
Modelo de datos	Tablas con filas y columnas (estructurado)	Documentos, grafos, pares clave-valor, columnas
Esquema	Fijo (estructurado y normalizado)	Flexible (puede variar entre documentos)
Consultas	SQL (Structured Query Language)	Lenguajes específicos o API de consultas
Integridad de datos	Alta (claves primarias y foráneas)	Variable según tipo de NoSQL
Escalabilidad	Vertical (más recursos en un solo servidor)	Horizontal (distribución entre múltiples nodos)
Rendimiento	Óptimo para transacciones ACID	Óptimo para grandes volúmenes de datos semiestruct.
Casos de uso	ERPs, CRMs, sistemas bancarios	Big Data, IoT, aplicaciones móviles y en tiempo real

2. Tecnologías seleccionadas

Bases de datos relacionales:

1. MySQL

- **Razón:** Es ampliamente utilizada, de código abierto y cuenta con soporte activo. Ideal para transacciones financieras, gestión de usuarios y sistemas estructurados.
- **Ventajas:** Madurez, comunidad activa, replicación, integridad referencial.

2. PostgreSQL

- **Razón:** Ofrece mayor soporte para tipos de datos complejos y funciones avanzadas. Ideal para escenarios donde se necesita integridad y personalización de consultas.
- **Ventajas:** Extensibilidad, soporte para JSON, funciones y procedimientos personalizados.

Bases de datos NoSQL:

1. MongoDB (tipo documento)

- **Razón:** Almacena datos en formato BSON (similar a JSON), ideal para manejar información de productos, catálogos, y perfiles de usuario.
- **Ventajas:** Flexibilidad, escalabilidad horizontal, consultas eficientes sobre documentos anidados.

2. Redis (clave-valor)

- **Razón:** Ideal para almacenamiento temporal, sesiones de usuarios y sistemas de caché.
- **Ventajas:** Velocidad extrema, soporte en memoria, ideal para datos de acceso rápido.

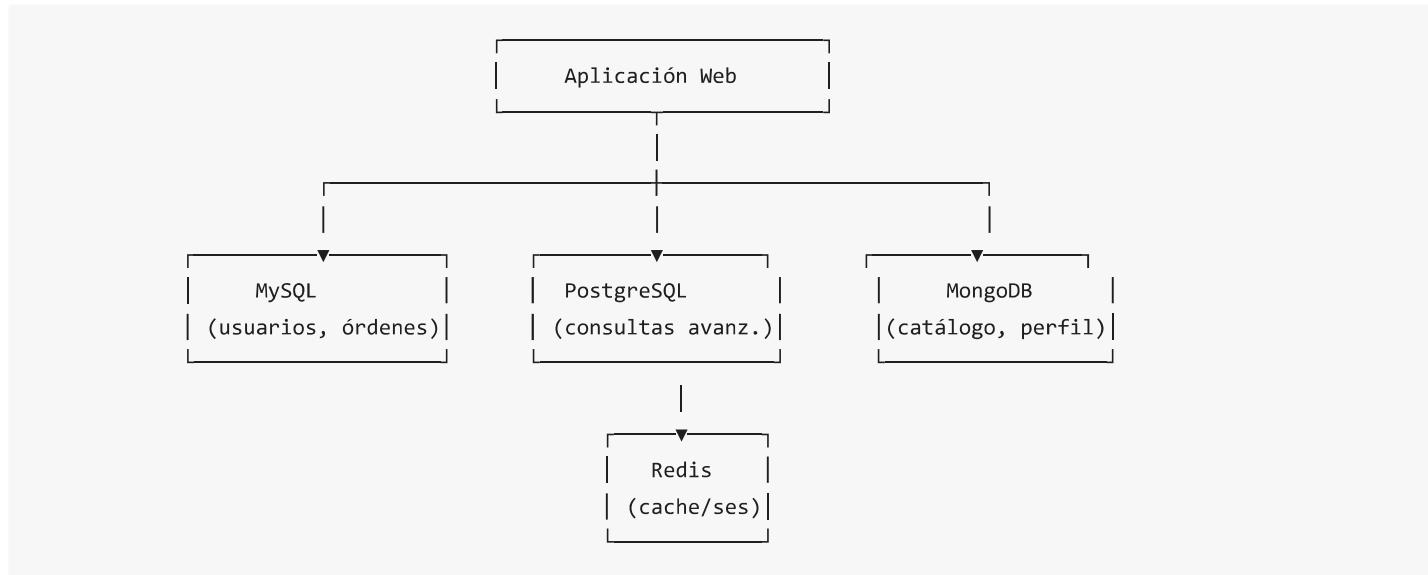
3. Justificación técnica

La solución combina bases relacionales y NoSQL para aprovechar lo mejor de ambos mundos:

- **Relacionales** como MySQL y PostgreSQL ofrecen integridad y estructura para transacciones críticas (órdenes, usuarios, pagos).
- **NoSQL** como MongoDB y Redis permiten escalar el sistema, almacenar grandes volúmenes de datos heterogéneos y ofrecer rendimiento en tiempo real para ciertas funcionalidades (catálogos dinámicos, historial, cacheo de resultados).

Esta arquitectura híbrida permite adaptarse a diversos volúmenes, tipos y velocidades de datos, maximizando eficiencia y escalabilidad.

4. Diagrama del ecosistema de bases de datos



Lección 2: Optimización de consultas SQL

Objetivo

Diseñar un esquema relacional normalizado y optimizado que forme parte del ecosistema de integración de datos de la tienda digital, utilizando SQL relacional.

1. Decisiones de Diseño y Normalización

Se diseñó un esquema relacional basado en un escenario transaccional de gestión de pedidos, usuarios y productos. Se aplicaron las siguientes normalizaciones:

- **1FN (Primera Forma Normal)**: No hay atributos multivaluados ni repetitivos.
- **2FN (Segunda Forma Normal)**: Todos los atributos no clave dependen completamente de la clave primaria.
- **3FN (Tercera Forma Normal)**: No hay dependencias transitivas.

Relaciones diseñadas:

- Usuarios (clientes que hacen pedidos).
- Productos (catálogo).
- Ordenes (compras hechas por los usuarios).
- DetalleOrden (productos incluidos en cada orden).

2. Técnicas de Optimización Aplicadas

Se aplicaron varias técnicas para mejorar el rendimiento de las consultas:

- **Índices:**

- Sobre columnas de búsqueda frecuente: `usuario_id`, `fecha_orden`, nombre de producto.
- Para optimizar joins en `DetalleOrden`.

- **Vistas optimizadas:**

- La vista `vista_ordenes_usuarios` permite acceder rápidamente a información cruzada entre órdenes y usuarios, útil para dashboards o reportes.

3. Integración con el módulo anterior

El pipeline de datos ETL del módulo anterior carga datos limpios y estructurados. Este modelo relacional está diseñado para:

- Recibir los datos ya transformados.
- Validar la consistencia con claves primarias/foráneas.
- Ofrecer una estructura optimizada para análisis transaccional, reportes y exploración.

Por ejemplo, los datos de usuarios, productos y órdenes limpios del pipeline pueden insertarse directamente aquí.

Conclusión

Este modelo relacional proporciona una base robusta, normalizada y optimizada para realizar operaciones eficientes y servir como origen o destino de integración con otros sistemas como bases NoSQL o lagos de datos.

Esquema relacional normalizado para la tienda digital

```
-- Tabla de usuarios
CREATE TABLE Usuarios (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100),
    correo VARCHAR(100) UNIQUE,
    fecha_registro DATE
);

-- Tabla de productos
CREATE TABLE Productos (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100),
    descripcion TEXT,
    precio DECIMAL(10,2),
    stock INT
);

-- Tabla de órdenes
CREATE TABLE Ordenes (
    id INT AUTO_INCREMENT PRIMARY KEY,
    usuario_id INT,
    fecha_orden DATETIME,
    total DECIMAL(10,2),
    FOREIGN KEY (usuario_id) REFERENCES Usuarios(id)
```

```

);

-- Tabla de detalles de cada orden
CREATE TABLE DetalleOrden (
    id INT AUTO_INCREMENT PRIMARY KEY,
    orden_id INT,
    producto_id INT,
    cantidad INT,
    precio_unitario DECIMAL(10,2),
    FOREIGN KEY (orden_id) REFERENCES Ordenes(id),
    FOREIGN KEY (producto_id) REFERENCES Productos(id)
);

-- Índices para optimizar búsquedas
CREATE INDEX idx_usuario_fecha ON Ordenes(usuario_id, fecha_orden);
CREATE INDEX idx_producto_nombre ON Productos(nombre);
CREATE INDEX idx_detalle_orden ON DetalleOrden(orden_id);
CREATE INDEX idx_detalle_producto ON DetalleOrden(producto_id);

-- Vista optimizada para reportes rápidos
CREATE VIEW vista_ordenes_usuarios AS
SELECT o.id AS orden_id, u.nombre AS usuario, o.fecha_orden, o.total
FROM Ordenes o
JOIN Usuarios u ON o.usuario_id = u.id;

```

Lección 3: Bases de Datos No Relacionales

Objetivo: Identificar y documentar casos de uso específicos para bases NoSQL, complementando el esquema relacional creado.

1. Características de las tecnologías NoSQL seleccionadas

MongoDB (Modelo de Documentos)

- **Modelo de almacenamiento:** Documentos BSON (formato binario de JSON).
- **Esquema flexible:** Permite cambios en la estructura sin afectar los datos existentes.
- **Consultas ricas:** Usa una sintaxis similar a JSON para consultas complejas.
- **Escalabilidad horizontal:** Alta capacidad para escalar en clústeres distribuidos.
- **Alta disponibilidad:** Mediante replicación y balanceo de carga.
- **Uso común:** Aplicaciones con estructuras de datos variadas o cambiantes.

Redis (Clave-Valor en memoria)

- **Modelo de almacenamiento:** Clave-valor con soporte para listas, conjuntos, hashes y más.
- **Alto rendimiento:** Opera íntegramente en memoria, ideal para lecturas y escrituras rápidas.
- **Persistencia opcional:** Puede configurarse para persistencia en disco o solo en RAM.
- **Soporte para pub/sub y streams:** Ideal para mensajería y procesamiento de eventos en tiempo real.
- **Uso común:** Caching, sesiones de usuarios, contadores en tiempo real.

2. Tipos de datos y consultas gestionados en NoSQL

Tecnología	Tipo de datos	Tipos de consultas
MongoDB	Documentos JSON flexibles	Búsqueda por campos, filtros por fecha, agregaciones
Redis	Claves con estructuras simples o compuestas	Acceso directo por clave, lectura/escritura ultrarrápida

3. Casos de uso específicos por tecnología

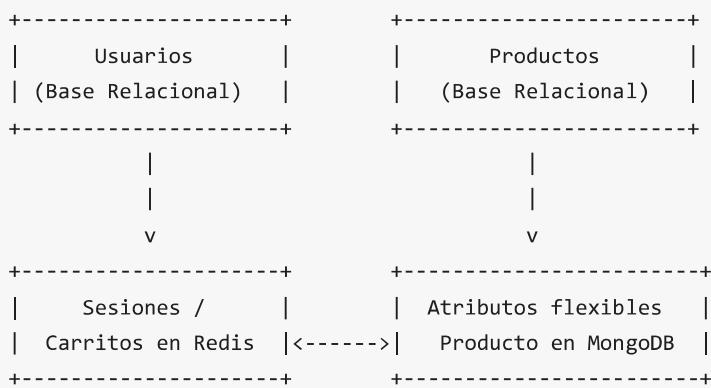
MongoDB

- **Descripción del uso:** Almacenar la información de productos con descripciones complejas, atributos variables y múltiples versiones de presentación (colores, tallas, etc.).
- **Ventaja:** Flexibilidad de esquema para adaptarse a distintos catálogos de productos sin necesidad de migraciones de esquema.
- **Escenario concreto:** Aplicación de catálogo donde los productos varían según temporada o proveedor.

Redis

- **Descripción del uso:** Gestión de sesiones de usuarios, carritos de compra temporales y caché de productos populares.
- **Ventaja:** Operaciones de lectura y escritura en milisegundos, ideal para mejorar la experiencia del usuario.
- **Escenario concreto:** E-commerce de alto tráfico que necesita mantener los carritos activos aunque el usuario navegue varias páginas.

4. Diagrama esquemático del ecosistema NoSQL + Relacional



→ Redis complementa la gestión transaccional con rapidez en sesión y caché.
 → MongoDB permite manejar atributos variables sin alterar el esquema SQL.

Conclusión

La integración de tecnologías NoSQL como **MongoDB** y **Redis** permite ampliar la flexibilidad, escalabilidad y rendimiento de la solución planteada. Estas bases no relacionales complementan el modelo relacional en áreas donde se requiere mayor adaptabilidad o velocidad, como la gestión de productos dinámicos y datos temporales.

▼ Lección 4: Apache Cassandra

1. Decisiones de Modelización

Para complementar el modelo relacional (Lección 2), se diseñó una tabla columnar en Apache Cassandra llamada **transacciones**, enfocada en el almacenamiento de **registros históricos de transacciones** por usuario.

- **Clave primaria compuesta:** se eligió `id_usuario` como partición y `fecha`, `id_transaccion` como claves de ordenamiento para facilitar la consulta cronológica.
- **Clustering ordenado:** por fecha e ID de transacción en orden descendente, lo que permite obtener fácilmente los últimos movimientos del usuario.

Esta estructura permite una rápida lectura secuencial de las transacciones de un usuario, un patrón muy común en sistemas financieros o de e-commerce.

2. Estrategia de Replicación y Escalabilidad

- Se usó `SimpleStrategy` con un `replication_factor = 3`, adecuada para entornos de desarrollo o pruebas.
- En producción, se recomienda `NetworkTopologyStrategy` para múltiples datacenters.
- Cassandra escala horizontalmente de forma nativa: al aumentar nodos, se mejora tanto la capacidad de almacenamiento como el rendimiento de lectura y escritura.

3. Consultas y Rendimiento

Las consultas están optimizadas para el **modelo de acceso por usuario**, sin necesidad de `JOIN` ni agregaciones pesadas:

- Consulta por ID de usuario.
- Consulta por ID de usuario y fecha.
- Consulta limitada para obtener los últimos movimientos.

Esto garantiza **baja latencia** en escenarios de alto volumen de datos, como plataformas de ventas online.

4. Integración en el Ecosistema

Cassandra se integra como **componente NoSQL especializado en series temporales** dentro del ecosistema de bases de datos definido en las lecciones anteriores:

Tipo	Tecnología	Uso
Relacional	PostgreSQL / MySQL	Gestión estructurada (clientes, productos, pedidos)
NoSQL - Documentos	MongoDB	Almacén flexible para fichas de productos
NoSQL - Columnar	Cassandra	Historial de transacciones, consultas por usuario y tiempo

El modelo relacional sigue manejando relaciones estructuradas clave, mientras que Cassandra gestiona grandes volúmenes de datos de forma escalable y eficiente.

Script CQL (Lección 4: Apache Cassandra)

Archivo: `cassandra_modelo.cql`

```
-- Crear el Keyspace con estrategia de replicación
CREATE KEYSPACE IF NOT EXISTS ecommerce_transacciones
WITH REPLICATION = {
    'class': 'SimpleStrategy',
    'replication_factor': 3
};

-- Usar el keyspace
USE ecommerce_transacciones;

-- Crear tabla de transacciones históricas
CREATE TABLE IF NOT EXISTS transacciones (
    id_usuario UUID,
```

```

fecha DATE,
id_transaccion TIMEUUID,
total DECIMAL,
metodo_pago TEXT,
estado TEXT,
PRIMARY KEY ((id_usuario), fecha, id_transaccion)
) WITH CLUSTERING ORDER BY (fecha DESC, id_transaccion DESC);

-- Insertar datos simulados
INSERT INTO transacciones (id_usuario, fecha, id_transaccion, total, metodo_pago, estado)
VALUES (uuid(), '2025-07-01', now(), 99.90, 'tarjeta_credito', 'completado');

INSERT INTO transacciones (id_usuario, fecha, id_transaccion, total, metodo_pago, estado)
VALUES (uuid(), '2025-07-01', now(), 45.00, 'paypal', 'pendiente');

-- Consultas básicas
-- 1. Consultar todas las transacciones de un usuario
SELECT * FROM transacciones WHERE id_usuario = <UUID_DEL_USUARIO>;

-- 2. Consultar transacciones por usuario y fecha
SELECT * FROM transacciones WHERE id_usuario = <UUID_DEL_USUARIO> AND fecha = '2025-07-01';

-- 3. Obtener los últimos 5 movimientos de un usuario
SELECT * FROM transacciones WHERE id_usuario = <UUID_DEL_USUARIO> LIMIT 5;

```

Reemplazar <UUID_DEL_USUARIO> por el UUID real de un usuario generado con `uuid()`.

Lección 5: MongoDB - Modelo Documental y Operaciones CRUD

Documento Técnico

1. Estructura de los Documentos

Para esta lección se ha diseñado un modelo documental usando MongoDB con las siguientes colecciones principales:

- **clientes**: gestiona perfiles de clientes con datos semiestructurados.
- **productos**: almacena información flexible sobre productos ofrecidos.

Ejemplo de documento en la colección clientes :

```
{
  "_id": ObjectId("..."),
  "nombre": "Laura Perez",
  "email": "laura.perez@gmail.com",
  "telefono": "+56 9 1234 5678",
  "direccion": {
    "calle": "Av. Principal 123",
    "ciudad": "Santiago",
    "region": "Metropolitana"
  },
  "preferencias": ["electrónica", "libros"],
  "historial_compras": [
    {"producto_id": "P1001", "fecha": "2024-06-01", "monto": 89.99},
    {"producto_id": "P1004", "fecha": "2024-07-10", "monto": 120.50}
  ]
}
```

```
]
}
```

Ejemplo de documento en la colección productos :

```
{
  "_id": "P1001",
  "nombre": "Audífonos Bluetooth",
  "precio": 89.99,
  "stock": 15,
  "caracteristicas": {
    "color": "Negro",
    "bateria": "10h",
    "conexion": "Bluetooth 5.0"
  },
  "categorias": ["tecnología", "audio"]
}
```

2. Ventajas del Modelo Documental

- **Flexibilidad:** Permite almacenar documentos con estructura variable (ideal para perfiles de clientes o productos que cambian).
- **Agilidad en consultas:** MongoDB permite acceder a documentos completos sin necesidad de joins.
- **Escalabilidad:** Es adecuado para aplicaciones que requieren escalar horizontalmente.
- **Integración Natural:** JSON-like structure facilita el desarrollo en aplicaciones modernas frontend/backend.

3. Integración al Ecosistema General

MongoDB complementa el ecosistema relacional (como MySQL o PostgreSQL) al encargarse de datos semiestructurados que no requieren una normalización estricta:

- Perfiles de clientes personalizados
- Catálogos de productos con atributos flexibles
- Historiales de interacciones no normalizados

En el flujo general de datos, MongoDB se usará para:

- Consultas rápidas de perfil y preferencias.
- Almacenamiento flexible de productos que pueden cambiar de atributos con el tiempo.
- Complementar las operaciones del sistema relacional, que gestiona inventarios y ventas de forma estructurada.

Archivo de creación de documentos y operaciones CRUD (MongoDB)

```
// Conexión a MongoDB y creación de colecciones
use ecommerce;

// Insertar documentos en la colección clientes

db.clientes.insertMany([
  {
    nombre: "Laura Perez",
    email: "laura.perez@gmail.com",
    telefono: "+56 9 1234 5678",
    direccion: {
      calle: "Av. Presidente Balmaceda 1234",
      ciudad: "Santiago",
      pais: "Chile"
    }
  }
])
```

```

        calle: "Av. Principal 123",
        ciudad: "Santiago",
        region: "Metropolitana"
    },
    preferencias: ["electrónica", "libros"],
    historial_compras: [
        {producto_id: "P1001", fecha: "2024-06-01", monto: 89.99},
        {producto_id: "P1004", fecha: "2024-07-10", monto: 120.50}
    ]
}
]);

```

// Insertar documentos en la colección productos

```

db.productos.insertOne({
    _id: "P1001",
    nombre: "Audífonos Bluetooth",
    precio: 89.99,
    stock: 15,
    características: {
        color: "Negro",
        batería: "10h",
        conexión: "Bluetooth 5.0"
    },
    categorías: ["tecnología", "audio"]
});

```

// CONSULTAS CRUD

// READ: Buscar cliente por email

```

db.clientes.find({email: "laura.perez@gmail.com"});

```

// UPDATE: Agregar nueva preferencia a cliente

```

db.clientes.updateOne(
    {email: "laura.perez@gmail.com"},
    {$push: {preferencias: "hogar"}}
);

```

// DELETE: Eliminar un producto por ID

```

db.productos.deleteOne({_id: "P1001"});

```

Este modelo está diseñado para optimizar la gestión de datos semiestructurados, mejorar la experiencia del cliente y facilitar el mantenimiento del sistema. MongoDB aporta versatilidad y agilidad, fortaleciendo el ecosistema general propuesto.

Lección 6: DynamoDB

1. Diseño de la tabla en DynamoDB

- **Nombre de la tabla:** EventosUsuario

•

Clave primaria:

- **Partition Key:** usuario_id (tipo: String)
- **Sort Key:** timestamp (tipo: String) → para ordenar cronológicamente los eventos por usuario.

Atributos adicionales:

- evento_tipo: tipo de evento (login, compra, clic, etc.)
- detalles: datos específicos del evento (como ID de producto, ubicación, etc.)
- ip: dirección IP de la interacción

❖ Estructura conceptual (key-value):

```
{  
    "usuario_id": "U001",  
    "timestamp": "2025-07-18T14:25:00Z",  
    "evento_tipo": "compra",  
    "detalles": {  
        "producto_id": "P2345",  
        "monto": 12000  
    },  
    "ip": "190.23.15.2"  
}
```

❖ 2. Script de configuración usando AWS CLI

Crear la tabla

```
aws dynamodb create-table \  
    --table-name EventosUsuario \  
    --attribute-definitions \  
        AttributeName=usuario_id,AttributeType=S \  
        AttributeName=timestamp,AttributeType=S \  
    --key-schema \  
        AttributeName=usuario_id,KeyType=HASH \  
        AttributeName=timestamp,KeyType=RANGE \  
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \  
    --region us-east-1
```

Insertar un registro

```
aws dynamodb put-item \  
    --table-name EventosUsuario \  
    --item '{  
        "usuario_id": {"S": "U001"},  
        "timestamp": {"S": "2025-07-18T14:25:00Z"},  
        "evento_tipo": {"S": "login"},  
        "detalles": {"M": {  
            "ubicacion": {"S": "Santiago"},  
            "navegador": {"S": "Chrome"}  
        }},  
        "ip": {"S": "190.23.15.2"}  
    }'
```

```
}' \n\n--region us-east-1
```

Leer datos

```
aws dynamodb query \n    --table-name EventosUsuario \n    --key-condition-expression "usuario_id = :uid" \n    --expression-attribute-values '{":uid":{"S":"U001"}}' \n    --region us-east-1
```

Actualizar datos

```
aws dynamodb update-item \n    --table-name EventosUsuario \n    --key '{"usuario_id": {"S": "U001"}, "timestamp": {"S": "2025-07-18T14:25:00Z"}}' \n    --update-expression "SET evento_tipo = :et" \n    --expression-attribute-values '{":et":{"S":"compra"}}' \n    --region us-east-1
```

Eliminar datos

```
aws dynamodb delete-item \n    --table-name EventosUsuario \n    --key '{"usuario_id": {"S": "U001"}, "timestamp": {"S": "2025-07-18T14:25:00Z"}}' \n    --region us-east-1
```

Documento técnico

1. Estructura de la tabla y datos

La tabla `EventosUsuario` sigue un modelo clave-valor con una clave compuesta (`usuario_id`, `timestamp`) para organizar eventos de interacción por usuario y ordenarlos cronológicamente. El atributo `detalles` permite almacenar información flexible en formato tipo mapa (Map), útil para distintos tipos de eventos.

2. Operaciones CRUD realizadas

Se implementaron las siguientes operaciones:

- **Create:** `put-item`
- **Read:** `query`
- **Update:** `update-item`
- **Delete:** `delete-item`

Todas se ejecutaron desde la terminal usando AWS CLI en la región `us-east-1`.

3. Justificación del uso de DynamoDB

- **Serverless:** DynamoDB no requiere administración de infraestructura, lo que reduce costos y complejidad operativa.
- **Alta disponibilidad:** Datos replicados automáticamente en múltiples zonas de disponibilidad.
- **Escalabilidad automática:** Se adapta al tráfico sin necesidad de intervención manual.
- **Modelo flexible:** Permite almacenar distintos tipos de eventos con esquemas variables.

4. Integración con el ecosistema general

DynamoDB se integra como el componente de interacción en tiempo real de usuarios. Complementa:

- **Cassandra**: procesamiento de grandes volúmenes históricos.
- **MongoDB**: manejo de perfiles semiestructurados y productos.
- **RDBMS (Lección 2)**: mantiene los datos estructurados base del sistema (usuarios, pedidos).

Resumen del Proyecto Integrador

Este proyecto integrador abordó el diseño, implementación e integración de diferentes tecnologías de bases de datos relacionales y NoSQL en un ecosistema de datos moderno y escalable. A lo largo de seis lecciones, se exploraron múltiples enfoques y tecnologías con el fin de construir una arquitectura de datos robusta, flexible y adaptada a distintas necesidades del sistema.

Lección 1 - Análisis Inicial y Selección Tecnológica

Se identificaron las necesidades del sistema y se eligieron las tecnologías de base de datos más adecuadas para cubrir distintos tipos de datos (estructurados y no estructurados), consultas, escalabilidad y rendimiento. Las tecnologías seleccionadas fueron: MySQL (relacional), Cassandra (columnar), MongoDB (documental) y DynamoDB (clave-valor, serverless).

Lección 2 - Modelo Relacional (MySQL)

Se diseñó el modelo entidad-relación utilizando MySQL, centrado en entidades como usuarios, productos y transacciones. Se creó un esquema normalizado con integridad referencial y un script de creación de tablas SQL con datos de ejemplo.

Lección 3 - Bases NoSQL y Casos de Uso

Se documentaron las características, ventajas y casos de uso específicos de cada tecnología NoSQL seleccionada. Se explicó cómo cada una se complementa con el modelo relacional:

- **Cassandra**: para almacenamiento distribuido de registros históricos.
- **MongoDB**: para datos semiestructurados como perfiles de clientes y productos flexibles.
- **DynamoDB**: para registrar interacciones en tiempo real con alta disponibilidad.

Lección 4 - Apache Cassandra

Se diseñó un modelo columnar en Cassandra para registros históricos de transacciones. Se definió la estrategia de replicación y escalabilidad. Se elaboró un script `.cql` con la creación de tablas, inserciones y consultas básicas.

Lección 5 - MongoDB

Se diseñó una base documental con colecciones como `clientes` y `productos`, permitiendo estructuras flexibles. Se realizaron operaciones CRUD y se explicó cómo MongoDB facilita consultas sobre datos semiestructurados y cómo se integra al flujo general.

Lección 6 - DynamoDB

Se diseñó una tabla para almacenar eventos de interacción del usuario en DynamoDB. Se realizaron operaciones CRUD utilizando AWS CLI y se documentó cómo su modelo serverless garantiza disponibilidad y escalabilidad, complementando a Cassandra y MongoDB.

Conclusión Final

El proyecto permitió integrar una **arquitectura híbrida** de datos que aprovecha las fortalezas de distintas tecnologías:

- **MySQL** garantiza integridad para datos estructurados y relaciones complejas.
- **Cassandra** ofrece eficiencia en la escritura y lectura distribuida para grandes volúmenes históricos.
- **MongoDB** proporciona flexibilidad para estructuras cambiantes y consultas ágiles sobre datos semiestructurados.
- **DynamoDB**, con su enfoque serverless, permite una solución rápida, escalable y altamente disponible para registros en tiempo real.

Esta integración demuestra cómo una **arquitectura de datos moderna** no depende exclusivamente de una sola base, sino que **combina múltiples tecnologías para adaptarse a distintos tipos de información y necesidades de rendimiento**. Con esta base, el sistema está preparado para escalar, integrarse con servicios en la nube, y responder de forma eficaz a entornos dinámicos y de alto tráfico.