Modulo 4

Proyecto: Portafolio ABP Modulo 4

Nombre: Marco Neira

Lección 1 – Tecnologías de Base de Datos

Objetivo:

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

1. Investigar y documentar las características de bases de datos relacionales y NoSQL.

Característica	Bases de Datos Relacionales (SQL)	Bases de Datos No Relacionales (NoSQL)
Modelo de datos	Tablas y relaciones	Documentos, clave-valor, columnas, grafos
Lenguaje	SQL	Propio de cada motor (CQL, BSON, etc.)
Transacciones (ACID)	Sí	Parcial o eventual según tipo
Escalabilidad	Vertical	Horizontal
Casos de uso típicos	ERP, CRM, sistemas financieros	Big Data, IoT, análisis en tiempo real
Estructura de los datos	Estrictamente estructurada	Semiestructurada o no estructurada
Ejemplos	MySQL, PostgreSQL	MongoDB, Cassandra, DynamoDB, Redis

2. Seleccionar al menos dos tecnologías relacionales y dos NoSQL para resolver los requerimientos planteados.

Relacionales:

- PostgreSQL
- MySQL

NoSQL:

- MongoDB (documental)
- Cassandra (columnar)

3. Justificar técnicamente la elección de cada tecnología según tipo de datos, escalabilidad y eficiencia.

PostgreSQL: Motor relacional robusto, open source, soporta transacciones ACID, ideal para sistemas transaccionales complejos y consultas SQL avanzadas. Su soporte a JSON también permite cierta flexibilidad semiestructurada.

MySQL: Muy eficiente para cargas transaccionales ligeras a medianas, con gran comunidad y herramientas de administración. Ideal para aplicaciones web de alto volumen.

MongoDB: Orientado a documentos, excelente para almacenar datos semiestructurados como perfiles de usuarios o catálogos de productos. Ofrece gran flexibilidad y escalabilidad horizontal.

Cassandra: Sistema distribuido columnar ideal para grandes volúmenes de datos, altamente disponible y escalable horizontalmente. Perfecto para logs históricos o eventos en tiempo real.

Lección 2 - Optimización de consultas SQL

Objetivo:

Diseñar y optimizar un esquema de base de datos relacional, utilizando las tecnologías seleccionadas en la Lección 1.

1. **Diseñar un esquema relacional normalizado para un escenario transaccional**

Entidad	Atributos principales
Clientes	id_cliente, nombre, email, teléfono, fecha_registro
Direcciones	id_direccion, id_cliente (FK), ciudad, comuna, calle
Pedidos	id_pedido, id_cliente (FK), fecha_pedido, estado_pedido, id_pago (FK)
Métodos_Pago	id_pago, tipo_pago, proveedor
Productos	id_producto, nombre_producto, precio, stock
Detalle_Pedido	id_detalle, id_pedido (FK), id_producto (FK), cantidad, precio_unitario

2. **Crear las tablas, claves primarias y foráneas**

“-- ==========

-- Creación de esquema relacional normalizado

```
-- ============================================

-- Tabla: Clientes
CREATE TABLE Clientes (
  id_cliente SERIAL PRIMARY KEY,
  nombre VARCHAR(100) NOT NULL,
  email VARCHAR(100) UNIQUE NOT NULL,
  telefono VARCHAR(20),
  fecha_registro DATE DEFAULT CURRENT_DATE
);

-- Tabla: Direcciones
CREATE TABLE Direcciones (
  id_direccion SERIAL PRIMARY KEY,
  id_cliente INT NOT NULL,
  ciudad VARCHAR(50),
  comuna VARCHAR(50),
  calle VARCHAR(100),
  FOREIGN KEY (id_cliente) REFERENCES Clientes(id_cliente)
);

-- Tabla: Métodos de Pago
CREATE TABLE Metodos_Pago (
  id_pago SERIAL PRIMARY KEY,
  tipo_pago VARCHAR(50) NOT NULL,
  proveedor VARCHAR(50)
);

-- Tabla: Pedidos
```

```
CREATE TABLE Pedidos (
  id_pedido SERIAL PRIMARY KEY,
  id_cliente INT NOT NULL,
  fecha_pedido TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  estado_pedido VARCHAR(50),
  id_pago INT,
  FOREIGN KEY (id_cliente) REFERENCES Clientes(id_cliente),
  FOREIGN KEY (id_pago) REFERENCES Metodos_Pago(id_pago)
);

-- Tabla: Productos
CREATE TABLE Productos (
  id_producto SERIAL PRIMARY KEY,
  nombre_producto VARCHAR(100) NOT NULL,
  precio DECIMAL(10, 2) NOT NULL,
  stock INT DEFAULT 0
);

-- Tabla: Detalle del Pedido
CREATE TABLE Detalle_Pedido (
  id_detalle SERIAL PRIMARY KEY,
  id_pedido INT NOT NULL,
  id_producto INT NOT NULL,
  cantidad INT NOT NULL,
  precio_unitario DECIMAL(10, 2) NOT NULL,
  FOREIGN KEY (id_pedido) REFERENCES Pedidos(id_pedido),
  FOREIGN KEY (id_producto) REFERENCES Productos(id_producto)
);”
```

3. **Definir índices y vistas materializadas para consultas eficientes**

Crear índices en campos de búsqueda frecuente como:

- email en Clientes
- fecha_pedido en Pedidos
- id_producto en Detalle_Pedido

“CREATE INDEX idx_cliente_email ON Clientes(email);

CREATE INDEX idx_pedido_fecha ON Pedidos(fecha_pedido);”

4. **Realizar consultas SQL sobre los datos del pipeline anterior**

“-- Total de pedidos por cliente

SELECT c.nombre, COUNT(p.id_pedido) AS total_pedidos

FROM Clientes c

JOIN Pedidos p ON c.id_cliente = p.id_cliente

GROUP BY c.nombre;”

“-- Productos más vendidos

SELECT pr.nombre_producto, SUM(dp.cantidad) AS vendidos

FROM Detalle_Pedido dp

JOIN Productos pr ON dp.id_producto = pr.id_producto

GROUP BY pr.nombre_producto

ORDER BY vendidos DESC;”

Lección 3 - Bases de datos No Relacionales

Objetivo:

Identificar y documentar casos de uso específicos para bases NoSQL, **complementando** el esquema relacional diseñado en la Lección 2.

1. **Profundizar en los modelos NoSQL seleccionados**

Tecnología	Tipo de NoSQL	Características principales
MongoDB	Documental	Manejo de documentos JSON/BSON, ideal para datos semiestructurados, permite estructuras flexibles.
Cassandra	Columnar distribuida	Altamente escalable, orientado a escritura intensiva, ideal para datos históricos o eventos.

2. **Determinar qué tipo de datos y consultas serán gestionados en bases NoSQL**

Tecnología	Datos gestionados
MongoDB	Perfiles de clientes, carritos de compra, historial de navegación, preferencias de usuario
Cassandra	Logs de actividad, registros históricos de pedidos, analítica de eventos por timestamp

3. **Definir escenarios concretos para el uso de cada tecnología**

Tecnología	Escenario de uso	Justificación
MongoDB	Almacenamiento de documentos dinámicos para clientes o productos con atributos variables	MongoDB permite trabajar sin esquema rígido, ideal para catálogos con distintos atributos.
MongoDB	Carritos de compra temporales con actualizaciones frecuentes	La flexibilidad de estructura y soporte para operaciones CRUD complejas en documentos lo permite.
Cassandra	Registro de eventos de pedidos: cuándo se creó, procesó, despachó, entregó (eventos con timestamp)	Cassandra es ideal para series de tiempo distribuidas y para escribir grandes volúmenes rápidamente.
Cassandra	Métricas de rendimiento o visitas en tiempo real en e-commerce	Gracias a su replicación y lectura/escritura distribuida, es ideal para métricas de alta frecuencia.

Lección 4 - Apache Cassandra

Objetivo:

Diseñar e implementar un modelo de base de datos **distribuida columnar** en Cassandra.

1. **Crear un modelo columnar para almacenar registros históricos**

Registro de eventos por pedido, como:

- Pedido creado
- Pedido confirmado
- Pedido despachado
- Pedido entregado

“CREATE TABLE eventos_pedidos (

id_pedido UUID,

timestamp_evento TIMESTAMP,

tipo_evento TEXT,

estado TEXT,

descripcion TEXT,

PRIMARY KEY (id_pedido, timestamp_evento)

) WITH CLUSTERING ORDER BY (timestamp_evento ASC);”

2. **Generar un script CQL para la creación de tablas y consultas**

“-- Crear keyspace

CREATE KEYSPACE ecommerce_ks WITH REPLICATION = {

'class': 'SimpleStrategy',

'replication_factor': 2

};

-- Usar keyspace

USE ecommerce_ks;

-- Crear tabla de eventos

CREATE TABLE eventos_pedidos (

  id_pedido UUID,

  timestamp_evento TIMESTAMP,

  tipo_evento TEXT,

  estado TEXT,

  descripcion TEXT,

  PRIMARY KEY (id_pedido, timestamp_evento)

) WITH CLUSTERING ORDER BY (timestamp_evento ASC);”

3. **Realizar inserciones y consultas básicas**

“INSERT INTO eventos_pedidos (id_pedido, timestamp_evento, tipo_evento, estado, descripcion)

VALUES (uuid(), toTimestamp(now()), 'CREACION', 'CREADO', 'Pedido generado por el cliente');

INSERT INTO eventos_pedidos (id_pedido, timestamp_evento, tipo_evento, estado, descripcion)

VALUES (uuid(), toTimestamp(now()), 'ENVIO', 'ENVIADO', 'Pedido despachado al cliente');”

“SELECT * FROM eventos_pedidos

WHERE id_pedido = 123e4567-e89b-12d3-a456-426614174000

ORDER BY timestamp_evento ASC;”

4. **Documentar decisiones de replicación, escalabilidad y eficiencia**

Resumen técnico:

- Se eligió **SimpleStrategy** con replication_factor = 2 para balancear disponibilidad sin complejidad de multi-datacenter.
- La clave de partición id_pedido permite una distribución equitativa entre nodos.
- El ordenamiento por timestamp_evento permite recuperar fácilmente el historial cronológico.

- El modelo está optimizado para **escrituras frecuentes** y **lecturas por clave** específicas (no se recomiendan joins ni agregaciones complejas).

Lección 5 - MongoDB

Objetivo:

Diseñar y manipular una base de datos documental con operaciones CRUD y justificar su estructura dentro del ecosistema.

1. **Diseñar la estructura de colecciones y documentos**

Manejo de **perfiles de clientes**, incluyendo preferencias, historial de navegación y carritos de compra. Ejemplo de clientes:

“{

"_id": ObjectId("64f2d9cbbf31b2f9eac74c3d"),

"nombre": "Carlos Soto",

"email": "csoto@example.com",

"telefono": "987654321",

"fecha_registro": "2024-01-10T00:00:00Z",

"preferencias": ["tecnología", "hogar"],

"carrito": [

{

"id_producto": "P123",

"nombre": "Smartphone X",

"cantidad": 1,

"precio_unitario": 399.99

},

{

"id_producto": "P789",

"nombre": "Auriculares Bluetooth",

"cantidad": 2,

"precio_unitario": 29.99

```
  }
 ]
}”
```

2. **Crear los documentos y realizar operaciones CRUD**

```
// Crear base de datos y colección
use ecommerce;
db.clientes.insertOne({
 nombre: "Carlos Soto",
 email: "csoto@example.com",
 telefono: "987654321",
 fecha_registro: new Date("2024-01-10"),
 preferencias: ["tecnología", "hogar"],
 carrito: [
  { id_producto: "P123", nombre: "Smartphone X", cantidad: 1, precio_unitario: 399.99 },
  { id_producto: "P789", nombre: "Auriculares Bluetooth", cantidad: 2, precio_unitario: 29.99 }
 ]
});
```

Operaciones CRUD:

```
“db.clientes.find({ email: "csoto@example.com" });”
“
db.clientes.updateOne(
 { email: "csoto@example.com" },
 {
  $push: {
   carrito: { id_producto: "P456", nombre: "Tablet Z", cantidad: 1, precio_unitario: 199.99 }
  }
 }
);”
```

“db.clientes.updateOne(

{ email: "csoto@example.com" },

{ $set: { carrito: [] } }

);”

3. **Documento técnico explicando la estructura y ventajas del modelo documental**

Justificación:

- MongoDB permite **esquemas flexibles**, ideal para manejar datos con variabilidad en la estructura (ej. distintos tipos de carritos o preferencias).
- Soporta **subdocumentos embebidos**, lo cual reduce la necesidad de joins y mejora la eficiencia de lectura para objetos anidados.
- Excelente para escenarios donde los datos cambian dinámicamente o tienen distintas formas según el cliente o producto.
- Se integra bien con pipelines ETL y herramientas de análisis, como **MongoDB Atlas** o **Compass**.

Lección 6 - DynamoDB

Objetivo:

Implementar una base de datos NoSQL serverless en AWS DynamoDB, integrada al flujo de datos del ecosistema.

1. **Diseñar y crear una tabla en DynamoDB**

Almacenar **eventos de interacción de usuarios**, como clics, inicios de sesión, o cambios de estado.

Atributo	Tipo	Observaciones
id_usuario	String	Clave de partición (Partition Key)
timestamp_evento	String	Clave de ordenamiento (Sort Key)
tipo_evento	String	"login", "clic", "compra", etc.
dispositivo	String	móvil, desktop
detalle	String	Descripción u objeto serializado (JSON)

Ejemplo de comando en AWS CLI para crear tabla:

“aws dynamodb create-table \

--table-name EventosInteraccion \

--attribute-definitions \

AttributeName=id_usuario,AttributeType=S \

AttributeName=timestamp_evento,AttributeType=S \

--key-schema \

AttributeName=id_usuario,KeyType=HASH \

AttributeName=timestamp_evento,KeyType=RANGE \

--provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5”

2. **Ejecutar operaciones CRUD (con AWS CLI o SDK)**

Inserción de datos (PutItem):

“aws dynamodb put-item \

--table-name EventosInteraccion \

--item '{

"id_usuario": {"S": "U123"},

"timestamp_evento": {"S": "2025-08-03T10:00:00Z"},

"tipo_evento": {"S": "login"},

"dispositivo": {"S": "móvil"},

"detalle": {"S": "Inicio de sesión exitoso"}

}'"

Consulta por usuario:

"aws dynamodb query \

--table-name EventosInteraccion \

--key-condition-expression "id_usuario = :id" \

--expression-attribute-values '{":id":{"S":"U123"}}'"

Eliminación de item:

"aws dynamodb delete-item \

--table-name EventosInteraccion \

--key '{

"id_usuario": {"S": "U123"},

"timestamp_evento": {"S": "2025-08-03T10:00:00Z"}

}'"

3. **Justificación técnica de DynamoDB e integración**

Ventajas del modelo:

- **Alta disponibilidad y escalabilidad automática**.
- Ideal para **eventos atómicos, rápidos y frecuentes**.
- No requiere provisión de infraestructura (serverless).
- Su modelo de clave-partición + clave-ordenamiento permite consultar eventos de un usuario en orden temporal fácilmente.
- Integración directa con pipelines en AWS (por ejemplo, Lambda, Kinesis, API Gateway).