

Descripción general

El candidato debe implementar un pipeline para migrar documentos de MongoDB a tablas relacionales en PostgreSQL usando Java con Apache Camel. Se evaluará también manejo de PL/pgSQL, modelado de datos en MongoDB y buenas prácticas de desarrollo con Git.

Módulo 1: MongoDB y modelado

- Diseña una colección clientes con los campos:
 - `_id` (`ObjectId`)
 - `nombre` (`string`)
 - `correo` (`string`)
 - `direccion` (documento embebido con calle, ciudad, pais)

Respuesta:

Se usa docker para levantar MongoDB en contenedores.

```
{  
  "_id": ObjectId(...),           // generado por Mongo  
  "nombre": "Ana Pérez",        // string  
  "correo": "ana@example.com",   // string (clave natural para upsert)  
  "direccion": {                // documento embebido (1:1)  
    "calle": "Calle 10 #5-12",  
    "ciudad": "Bogotá",  
    "pais": "Colombia"  
  }  
}
```

Dirección embebida: como es 1-a-1 y se lee siempre junto al cliente, embebemos (lecturas atómicas y menos “joins”/lookups).

correo es la **clave natural** que usamos después para el *upsert* en Postgres (en Postgres queda UNIQUE).

_id lo deja Mongo por defecto (`ObjectId`).

- Inserta 10 documentos de prueba con datos variados.

Respuesta:

Se crea un archivo .sh que contiene la estructura solicitada de la colección en MongoDB. De esta forma vamos a tener un seed capaz de alimentar la colección. Desde el seed podremos manipular los datos iniciales que la componen.

```
#!/bin/bash
set -eux pipefail
ROOT="${PWD}/prueba-camel-mongo-postgres"
cat > "$ROOT/mongo/init/01_seed.js" << JS
db = db.getSiblingDB('demo');
db.clientes.drop();
db.clientes.insertMany([
  { nombre:"Ana Pérez",correo:"ana@example.com",direccion:{calle:"Calle 10 #5-12",ciudad:"Bogotá",pais:"Colombia"} },
  { nombre:"Luis Gómez",correo:"luis@example.com",direccion:{calle:"Av. Reforma 123",ciudad:"CDMX",pais:"México"} },
  { nombre:"María López",correo:"maria@example.com",direccion:{calle:"Gran Vía 45",ciudad:"Madrid",pais:"España"} },
  { nombre:"John Smith",correo:"john@example.com",direccion:{calle:"742 Evergreen",ciudad:"Springfield",pais:"USA"} },
  { nombre:"Sofía Ramírez",correo:"sofia@example.com",direccion:{calle:"Cra 7 #12-80",ciudad:"Medellín",pais:"Colombia"} },
  { nombre:"Pedro Sánchez",correo:"pedro@example.com",direccion:{calle:"Av. Libertador 500",ciudad:"Buenos Aires",pais:"Argentina"} },
  { nombre:"Lucía Fernández",correo:"lucia@example.com",direccion:{calle:"Rue Augusta 100",ciudad:"São Paulo",pais:"Brasil"} },
  { nombre:"Carlos Ruiz",correo:"carlos@example.com",direccion:{calle:"Av. Arequipa 200",ciudad:"Lima",pais:"Perú"} },
  { nombre:"Elena Petrova",correo:"elena@example.com",direccion:{calle:"Nevsky 1",ciudad:"San Petersburgo",pais:"Rusia"} },
  { nombre:"Marta Díaz",correo:"marta@example.com",direccion:{calle:"C/ Colón 8",ciudad:"Valencia",pais:"España"} }
]);
JS
echo "Seed de Mongo creado."
~
```

- Crea consultas Mongo para:
 - Listar clientes por país.
 - Actualizar el correo electrónico de un cliente.

Respuesta:

```

bash-3.2$ 
bash-3.2$ 
bash-3.2$ 
bash-3.2$ 
bash-3.2$ open /Applications/Docker.app
bash-3.2$ 
bash-3.2$ 
bash-3.2$ 
bash-3.2$ 
bash-3.2$ docker --version
Docker version 28.3.2, build 578ccf6
bash-3.2$ 
bash-3.2$ 
bash-3.2$ 
bash-3.2$ 
bash-3.2$ docker info | head -n 20
Client:
  Version:    28.3.2
  Context:   desktop-linux
  Debug Mode: false
  Plugins:
    ai: Docker AI Agent - Ask Gordon (Docker Inc.)
      Version: v1.9.11
      Path:   /Users/franciscopena/.docker/cli-plugins/docker-ai
    buildx: Docker Buildx (Docker Inc.)
      Version: v0.26.1-desktop.1
      Path:   /Users/franciscopena/.docker/cli-plugins/docker-buildx
    cloud: Docker Cloud (Docker Inc.)
      Version: v0.4.11
      Path:   /Users/franciscopena/.docker/cli-plugins/docker-cloud
    compose: Docker Compose (Docker Inc.)
      Version: v2.39.1-desktop.1
      Path:   /Users/franciscopena/.docker/cli-plugins/docker-compose
    debug: Get a shell into any image or container (Docker Inc.)
      Version: 0.0.42
      Path:   /Users/franciscopena/.docker/cli-plugins/docker-debug
WARNING: DOCKER_INSECURE_NO_IPTABLES_RAW is set
bash-3.2$ 

```

Evidencia Docker MongoDB arriba:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
9ab79b5f2e4d	mongo:6	"docker-entrypoint.s..."	3 hours ago	Up 3 hours	0.0.0.0:27017->27017/tcp, [::]:27017->27017/tcp	mongo

Dentro de la consola de comando de MongoDB:

```

bash-3.2$ docker exec -it mongo mongosh demo
Current Mongosh Log ID: 6899fb8b9b64eb3e1774e399
Connecting to:      mongodb://127.0.0.1:27017/demo?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.5.6
Using MongoDB:     6.0.25
Using Mongosh:     2.5.6

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

To help improve our products, anonymous usage data is collected and sent to MongoDB periodically (https://www.mongodb.com/legal/privacy-policy).
You can opt-out by running the disableTelemetry() command.

-----
  The server generated these startup warnings when booting
  2025-08-11T11:40:39.614+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/prodnotes-filesystem
  2025-08-11T11:40:40.125+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
  2025-08-11T11:40:40.125+00:00: /sys/kernel/mm/transparent_hugepage/enabled is 'always'. We suggest setting it to 'never' in this binary version
  2025-08-11T11:40:40.125+00:00: vm.max_map_count is too low
-----
demo> 

```

Listar clientes por país:

```

demo> db.clientes.find(
...   { "direccion.pais": "Colombia" },
...   { _id: 0, nombre: 1, correo: 1, "direccion.ciudad": 1 }
... ).sort({ nombre: 1 });
...
[
  {
    nombre: 'Ana Pérez',
    correo: 'ana@example.com',
    direccion: { ciudad: 'Bogotá' }
  },
  {
    nombre: 'Sofía Ramírez',
    correo: 'sofia@example.com',
    direccion: { ciudad: 'Medellín' }
  }
]
demo> 

```

Actualizar el correo de un cliente (por nombre)

```

demo> db.clientes.updateOne(
...   { nombre: "Ana Pérez" },
...   { $set: { correo: "ana.perez@nuevo.com" } }
... );
...
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
demo> 

```

```

demo> db.clientes.find(
...   { "direccion.pais": "Colombia" },
...   { _id: 0, nombre: 1, correo: 1, "direccion.ciudad": 1 }
... ).sort({ nombre: 1 });
[...
[ {
  nombre: 'Ana Pérez',
  correo: 'ana.perez@nuevo.com' Se actualiza el correo
},
{
  nombre: 'Sofía Ramírez',
  correo: 'sofia@example.com',
  dirección: { ciudad: 'Medellín' }
}
]
demo>

```

- Explica brevemente las ventajas y desventajas de usar documentos embebidos vs referencias en MongoDB.

Respuesta:

Documentos embebidos (sub-documentos dentro del mismo doc)

Ventajas

- **Lecturas rápidas**: todo junto en 1 lectura.
- **Atomicidad local**: una sola operación actualiza todo el “aggregate”.
- **Menos complejidad**: menos colecciones y relaciones.

Desventajas

- **Duplicación** si el mismo dato se repite en muchos documentos.
- **Tamaño del doc** puede crecer (límite 16 MB por documento).
- **Actualizaciones costosas** si hay que cambiar el mismo dato en muchos docs.

Referencias (IDs a otra colección):

Ventajas

- **Evita duplicados** y facilita **reutilizar** datos compartidos.
- Funciona bien con **1:many / many:many** de tamaño variable.

Desventajas

- **Más lecturas** o `$lookup` (joins) → mayor latencia/complexidad.
- **No atómico** entre colecciones (si necesitas atomicidad, usa transacciones).
- Modelado y consultas un poco más complejos.

Módulo 2: PostgreSQL y PL/pgSQL

- Diseña tablas clientes y direcciones con integridad referencial adecuada para almacenar los datos migrados.

Respuesta:

La información se separó en dos tablas. Clientes y direcciones.

- Clientes: guarda la identidad (id, nombre, correo)
- Direcciones: guarda la dirección **1:1** con el cliente y referencia su id
- Correo se configura como UNIQUE como clave natural para soportar el upsert.
- La FK tiene ON DELETE CASCADE para mantener integridad: si borras un cliente, su dirección cae sola
- Esto normaliza el embebido de Mongo a un modelo relacional limpio y consulta amigable

```
#!/bin/bash
set -euo pipefail
ROOT="${PWD}/prueba-camel-mongo-postgres"
cat > "$ROOT/postgres/init/01_schema.sql" <<'SQL'
CREATE TABLE IF NOT EXISTS clientes (
    id SERIAL PRIMARY KEY,
    nombre TEXT NOT NULL,
    correo TEXT UNIQUE NOT NULL
);

CREATE TABLE IF NOT EXISTS direcciones (
    id SERIAL PRIMARY KEY,
    cliente_id INT NOT NULL REFERENCES clientes(id) ON DELETE CASCADE,
    calle TEXT NOT NULL,
    ciudad TEXT NOT NULL,
    pais TEXT NOT NULL
);
```

- Escribe una función PL/pgSQL que inserte o actualice un cliente y su dirección en las tablas correspondientes (upsert).

Respuesta:

Para insertar/actualizar desde Camel, creé una **función PL/pgSQL** que hace **UPSERT** por correo y sincroniza la dirección:

```

CREATE OR REPLACE FUNCTION upsert_cliente(
    p_nombre TEXT,p_correo TEXT,p_calle TEXT,p_ciudad TEXT,p_pais TEXT
) RETURNS INT AS $$ 
DECLARE v_cliente_id INT;
BEGIN
    INSERT INTO clientes(nombre, correo) VALUES (p_nombre, p_correo)
    ON CONFLICT (correo) DO UPDATE SET nombre = EXCLUDED.nombre
    RETURNING id INTO v_cliente_id;

    IF EXISTS (SELECT 1 FROM direcciones WHERE cliente_id = v_cliente_id) THEN
        UPDATE direcciones SET calle=p_calle, ciudad=p_ciudad, pais=p_pais WHERE cliente_id=v_cliente_id;
    ELSE
        INSERT INTO direcciones(cliente_id,calle,ciudad,pais) VALUES (v_cliente_id,p_calle,p_ciudad,p_pais);
    END IF;

    RETURN v_cliente_id;
END; $$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION clientes_por_pais(p_pais TEXT)
RETURNS TABLE(id INT, nombre TEXT, correo TEXT, calle TEXT, ciudad TEXT, pais TEXT) AS $$ 
BEGIN
    RETURN QUERY SELECT c.id,c.nombre,c.correo,d.calle,d.ciudad,d.pais
    FROM clientes c JOIN direcciones d ON d.cliente_id=c.id WHERE d.pais=p_pais;
END; $$ LANGUAGE plpgsql;
SQL
Echo "Esquema y funciones de Postgres creados."

```

- Crea un procedimiento almacenado que devuelva clientes de un país específico.

Respuesta:

Ejecutando dentro del contenedor:

```
docker exec -it postgres psql -U demo -d demo -c "select * from clientes_por_pais('Colombia');"
```

```

bash-3.2$ cd prueba-camel-mongo-postgres/
bash-3.2$ 
bash-3.2$ 
bash-3.2$ 
bash-3.2$ docker compose up -d postgres mongo
WARN[0000] /Users/franciscopena/Prueba_Camel/prueba-camel-mongo-postgres/docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion
[+] Running 2/2
  ✓ Container mongo      Running
  ✓ Container postgres   Started
bash-3.2$ 
bash-3.2$ 
bash-3.2$ docker exec -it postgres psql -U demo -d demo -c "select * from clientes_por_pais('Colombia');"
+-----+-----+-----+-----+-----+-----+
| id  | nombre | correo          | calle            | ciudad          | pais           |
+-----+-----+-----+-----+-----+-----+
| 1   | Ana Pérez | ana@example.com | Calle 10 #5-12 | Bogotá         | Colombia       |
| 5   | Sofía Ramírez | sofia@example.com | Cra 7 #12-80  | Medellín       | Colombia       |
+-----+-----+-----+-----+-----+-----+
(2 rows)

bash-3.2$ 

```

- Inserta datos de prueba manualmente para validar las tablas y funciones.

Entro directo a ejecutar bloque SQL:

```
bash-3.2$ docker exec -i postgres psql -U demo -d demo <<'SQL'
>
> SELECT upsert_cliente('Ana Pérez','ana@example.com','Calle 10 #5-12','Bogotá','Colombia');
> SELECT upsert_cliente('Luis Gómez','luis@example.com','Av. Reforma 123','CDMX','México');
> SELECT upsert_cliente('María López','maria@example.com','Gran Vía 45','Madrid','España');
>
>
>
> SELECT c.id, c.nombre, c.correo, d.calle, d.ciudad, d.pais
> FROM clientes c JOIN direcciones d ON d.cliente_id=c.id
> ORDER BY c.id;
>
>
>
> SELECT upsert_cliente('Ana P.','ana@example.com','Calle 11 #6-20','Bogotá','Colombia');
>
>
>
> SELECT c.id, c.nombre, c.correo, d.calle, d.ciudad, d.pais
> FROM clientes c JOIN direcciones d ON d.cliente_id=c.id
> WHERE c.correo='ana@example.com';
> SQL
```

Resultados:

```
upsert_cliente
-----
1
(1 row)

upsert_cliente
-----
2
(1 row)

upsert_cliente
-----
3
(1 row)
```

Upser de Clientes

```
id | nombre | correo | calle | ciudad | pais
---+-----+-----+-----+-----+-----+
1 | Ana Pérez | ana@example.com | Calle 10 #5-12 | Bogotá | Colombia
2 | Luis Gómez | luis@example.com | Av. Reforma 123 | CDMX | México
3 | María López | maria@example.com | Gran Vía 45 | Madrid | España
4 | John Smith | john@example.com | 742 Evergreen | Springfield | USA
5 | Sofía Ramírez | sofia@example.com | Cra 7 #12-80 | Medellín | Colombia
6 | Pedro Sánchez | pedro@example.com | Av. Libertador 500 | Buenos Aires | Argentina
7 | Lucía Fernández | lucia@example.com | Rue Augusta 100 | São Paulo | Brasil
8 | Carlos Ruiz | carlos@example.com | Av. Arequipa 200 | Lima | Perú
9 | Elena Petrova | elena@example.com | Nevsky 1 | San Petersburgo | Rusia
10 | Marta Diaz | marta@example.com | C/ Colón 8 | Valencia | España
(10 rows)
```

Revisión de
Upser de clientes

Modificación de registro:

```
upsert_cliente
-----
1
(1 row)

id | nombre | correo | calle | ciudad | pais
---+-----+-----+-----+-----+-----+
1 | Ana P. | ana@example.com | Calle 11 #6-20 | Bogotá | Colombia
(1 row)
```

Módulo 3: Java + Apache Camel

- Crea un proyecto Java (preferible Maven) que use Apache Camel para:
 - Leer documentos de la colección clientes en MongoDB.
 - Transformar esos documentos para que encajen con las tablas relacionales.
 - Invocar la función PL/pgSQL en PostgreSQL para insertar o actualizar los datos.
- Implementa logging y manejo de errores.
- Crea un test unitario o de integración para validar que un documento de MongoDB se inserte correctamente en PostgreSQL.
- Documenta brevemente cómo correr el proyecto y pruebas.

Respuesta:

Configurar las dependencias de camel a usar:

```
<properties>
    <maven.compiler.release>17</maven.compiler.release>
    <camel.version>3.22.2</camel.version>
    <junit.version>5.10.2</junit.version>
    <slf4j.version>2.0.13</slf4j.version>
</properties>

<dependencies>
    <!-- Camel core/main -->
    <dependency>
        <groupId>org.apache.camel</groupId>
        <artifactId>camel-main</artifactId>
        <version>${camel.version}</version>
    </dependency>

    <!-- Componentes Camel que usamos -->
    <dependency>
        <groupId>org.apache.camel</groupId>
        <artifactId>camel-mongodb</artifactId>
        <version>${camel.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apache.camel</groupId>
        <artifactId>camel-sql</artifactId>
        <version>${camel.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apache.camel</groupId>
        <artifactId>camel-timer</artifactId>
        <version>${camel.version}</version>
    </dependency>
```

En App.java creo los beans y los **registro** para que Camel los auto-inyecte:

```

package com.demo;
import org.apache.camel.CamelContext;
import org.apache.camel.impl.DefaultCamelContext;
import org.apache.camel.support.DefaultRegistry;
import org.postgresql.ds.PGSimpleDataSource;
import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoClients;
public class App {
    public static void main(String[] args) throws Exception {
        DefaultRegistry reg = new DefaultRegistry();
        PGSimpleDataSource ds = new PGSimpleDataSource();
        ds.setURL("jdbc:postgresql://localhost:5432/demo"); ds.setUser("demo"); ds.setPassword("demo");
        reg.bind( id: "postgresDS", ds);
        MongoClient client = MongoClients.create("mongodb://localhost:27017");
        reg.bind( id: "client", client);
        try (CamelContext ctx = new DefaultCamelContext(reg)) {
            ctx.addRoutes(new MongoToPostgresRoute()); ctx.start(); Thread.currentThread().join();
        }
    }
}

```

Camel detecta postgresDS para camel-sql y client para camel-mongodb.

Lectura desde MongoDB:

Como operation=findAll **no** puede ir en un *consumer*(from(...)), diseñé la ruta con **timer**:

- `from("timer:runOnce?repeatCount=1")` dispara una sola vez
- `to("mongodb:...&operation=findAll")` hace la consulta
- `split(body())` procesa cada documento

```

/*
 * En Camel, operation=findAll se usa en un endpoint productor (to:), no en un consumer (from:).
 * Por eso: usamos un timer (consumer), luego llamamos a Mongo con to(...findAll), y después split.
 */
from( uri: "timer:runOnce?repeatCount=1" ) RouteDefinition
    .routeId("mongo-to-postgres")
    // Cuerpo vacío para findAll (consulta = {}):
    .setBody(constant( value: "{}"))
    // Productor: consulta a Mongo y devuelve List<Document>
    .to( uri: "mongodb:client?database=demo&collection=clientes&operation=findAll")
    // Iterar cada Document del List
    .split(body()) SplitDefinition
        .process( Exchange exchange -> {
            // El body es un org.bson.Document; lo convertimos a JSON y extraemos campos
            org.bson.Document doc = exchange.getIn().getBody(org.bson.Document.class);
            String json = doc.toJson();
            JsonNode root = mapper.readTree(json);

```

Transformación a parámetros relacionales

Con la librería Jackson, parseo el Document y mapeo a los parámetros que espera la función PL/pgSQL `upsert_cliente(...)`:

```
.split(body()) SplitDefinition
    .process( Exchange exchange -> {
        // El body es un org.bson.Document; lo convertimos a JSON y extraemos campos
        org.bson.Document doc = exchange.getIn().getBody(org.bson.Document.class);
        String json = doc.toJson();
        JsonNode root = mapper.readTree(json);

        Map<String, Object> p = new HashMap<>();
        p.put("p_nombre", root.get("nombre").asText());
        p.put("p_correo", root.get("correo").asText());

        JsonNode d = root.get("direccion");
        p.put("p_calle", d.get("calle").asText());
        p.put("p_ciudad", d.get("ciudad").asText());
        p.put("p_pais", d.get("pais").asText());

        exchange.getIn().setBody(p);
    }) ExpressionNode
    .to( uri: "sql:SELECT upsert_cliente(:#p_nombre, :#p_correo, :#p_calle, :#p_ciudad, :#p_pais)")
    .log("Upsert OK -> resultado: ${body}")
.end();
```

Inserción/actualización en PostgreSQL

Invoco la función con camel-sql usando **named params**:

```
} ExpressionNode
    .to( uri: "sql:SELECT upsert_cliente(:#p_nombre, :#p_correo, :#p_calle, :#p_ciudad, :#p_pais)")
    .log("Upsert OK -> resultado: ${body}")
.end();
}
```

Logging y manejo de errores:

- Logs con `.log(...)` en la ruta y `slf4j-simple` en runtime.
- Bloque global de errores:

```

1 package com.demo;
2
3 import com.fasterxml.jackson.databind.JsonNode;
4 import com.fasterxml.jackson.databind.ObjectMapper;
5 import org.apache.camel.*;
6 import org.apache.camel.builder.RouteBuilder;
7
8 import java.util.HashMap;
9 import java.util.Map;
10
11 public class MongoToPostgresRoute extends RouteBuilder { 1 usage
12
13     private final ObjectMapper mapper = new ObjectMapper(); 1 usage
14
15     @Override
16     public void configure() {
17
18         onException(Exception.class)
19             .handled(true)
20             .log(LogLevel.ERROR, message: "Error procesando mensaje: ${exception.message}")
21             .maximumRedeliveries(0);

```

Test:

Creé un RouteSmokeTest (JUnit 5) que:

- Registra postgresDS y client (igual que App)
- Arranca el CamelContext con la ruta
- Verifica que el contexto **inicia sin excepciones**

Módulo 4: Git y entrega

- Inicializa un repositorio Git para el proyecto.
- Realiza commit de manera clara y significativos durante el desarrollo.
- Crea un README.md con:
 - Descripción del proyecto.
 - Instrucciones para levantar MongoDB y PostgreSQL (pueden ser con Docker).
 - Cómo compilar, correr y probar la aplicación.
- Sube el repositorio a GitHub o similar y entrega el link.

Entrega

- Repositorio Git con código, scripts y documentación.
- Evidencia de la migración exitosa (logs, screenshots, salida consola).

Respuesta: En el cuerpo del correo se adjunta link del repositorio Git; donde encontrarán todas las evidencias complementarias a este documento.