Comparativa técnica: del modelo relacional a NoSQL (caso real simulado)

Juan Carlos Rojas Pestana - 202022178
David Leonardo Mariño Ardila - 202112483
Nicolas Santiago Caceres Torres - 202111929

UPTC
Sogamoso, Boyacá
2025

Índice

1. Diagnóstico del Problema y Contexto

Una empresa ficticia dedicada a la gestión de eventos ha operado exitosamente con un sistema basado en una base de datos relacional (como PostgreSQL o MySQL). Sin embargo, a medida que su negocio crece y la naturaleza de los eventos se vuelve más dinámica, el modelo de datos actual comienza a presentar fricciones.

El contexto es el de una plataforma que necesita manejar información diversa y en constante evolución: eventos con estructuras de precios variables, promociones temporales y perfiles de asistentes con atributos personalizados (ej. preferencias dietéticas, intereses específicos, información corporativa).

El problema central es que el esquema rígido de la base de datos relacional dificulta la adaptación rápida a estas nuevas necesidades. La necesidad de obtener vistas completas de un evento (con todos sus tickets, promociones y asistentes) en tiempo real para miles de usuarios concurrentes expone las limitaciones del modelo actual, amenazando la escalabilidad y el rendimiento del sistema.

2. Análisis de Limitaciones del Enfoque Tradicional (SQL)

El enfoque relacional, si bien garantiza la consistencia y la integridad de los datos (ACID), impone varias limitaciones significativas para este caso de uso específico.

2.1. Rigidez del Esquema

El principal obstáculo es la falta de flexibilidad. Para agregar un nuevo campo a la información de los asistentes, como .empresa.º "talla de camiseta", se requiere una alteración de la estructura de la tabla (`ALTER TABLE`). Esto es un proceso costoso y resulta ineficiente cuando muchos campos son opcionales, generando una gran cantidad de celdas nulas o requiriendo modelos de datos complejos (EAV) que perjudican la claridad y el rendimiento.

2.2. Complejidad de las Consultas

El modelo de datos normalizado obliga a la aplicación a reconstruir la información mediante el uso intensivo de operaciones `JOIN`. Una simple solicitud para ver la página de un evento puede requerir unir las tablas `Evento`, `Ticket`, `Promocion`, `Asistencia` y `Asistente`. A medida que la base de datos crece, estas operaciones se convierten en un cuello de botella de rendimiento, aumentando la latencia para el usuario final.

2.3. Escalabilidad Horizontal Limitada

Las bases de datos SQL están diseñadas principalmente para escalar verticalmente (aumentando la potencia del servidor: CPU, RAM). Escalar horizontalmente (distribuyendo los datos en múltiples servidores) es un proceso complejo que no es nativo en la mayoría de los motores SQL y requiere arquitecturas de *sharding* o *clustering* difíciles de implementar

y mantener. Esto representa un riesgo directo para la capacidad del sistema de soportar picos de alta concurrencia, como durante la venta de boletos para un evento popular.

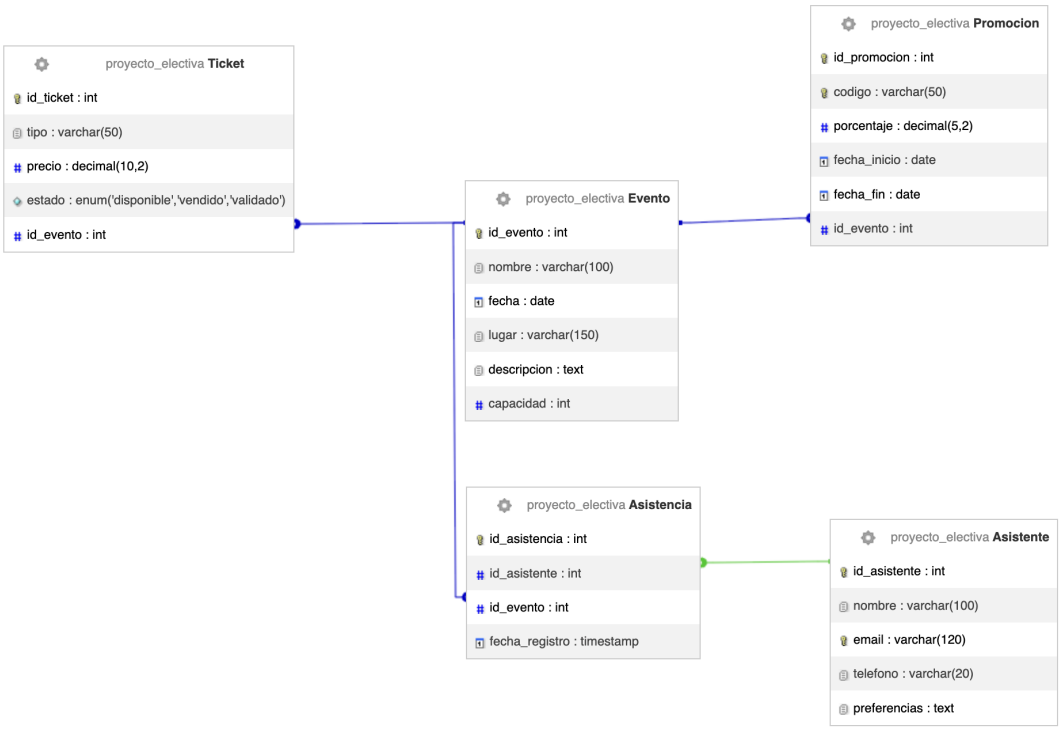

Figura 1: Diagrama Entidad-Relación del modelo de datos tradicional.

3. Justificación de la Tecnología Seleccionada (MongoDB)

Para superar las limitaciones identificadas, se seleccionó **MongoDB** como la base de datos NoSQL para el rediseño del sistema. Esta elección se justifica por las siguientes razones técnicas:

- **Modelo de Datos Flexible (Documentos JSON):** MongoDB almacena datos en documentos similares a JSON, lo que permite que cada registro (por ejemplo, un asistente) tenga su propia estructura. Esto resuelve directamente el problema de la rigidez del esquema, permitiendo agregar campos personalizados de forma natural y sin afectar a los demás documentos en la colección.
- **Rendimiento Optimizado para el Caso de Uso:** Al permitir embeber datos relacionados dentro de un mismo documento (como los tipos de tickets y promociones dentro del documento del evento), MongoDB elimina la necesidad de `JOINs`. Las operaciones de lectura más comunes se resuelven consultando un único documento, lo que resulta en una latencia significativamente menor y un mayor rendimiento.

- **Escalabilidad Nativa:** MongoDB fue diseñado desde su origen para escalar horizontalmente a través de un proceso llamado *sharding*. Esto permite distribuir los datos a través de un clúster de servidores de manera automática, garantizando que el sistema pueda manejar un crecimiento masivo en volumen de datos y número de usuarios sin degradar el rendimiento.
- **Ecosistema y Facilidad de Integración:** La elección de MongoDB se alinea perfectamente con la pila tecnológica propuesta (Node.js y Express.js). La librería Mongoose facilita la interacción con la base de datos. Además, la amplia comunidad y la disponibilidad de servicios en la nube como MongoDB Atlas aseguran un buen soporte y un camino claro para la futura producción y escalamiento del proyecto.

4. Requisitos

4.1. Funcionales

- Registrar y gestionar eventos (nombre, fecha, lugar, descripción, capacidad).
- Gestionar asistentes (datos personales, historial de participación, preferencias).
- Controlar tickets (tipo, precio, estado, validación de entrada).
- Manejar promociones y descuentos (códigos, porcentajes, fechas de validez).
- Ofrecer una API CRUD (crear, leer, actualizar, eliminar) para aplicaciones móviles.

4.2. No funcionales

- **Escalabilidad:** Soportar miles de usuarios concurrentes.
- **Rendimiento:** Consultas rápidas en colecciones grandes (ej. búsqueda de tickets).
- **Flexibilidad:** Almacenar datos semi-estructurados (ej. asistentes con diferentes atributos).
- **Portabilidad:** La base debe integrarse con microservicios o aplicaciones web.

5. Solución NoSQL elegida

5.1. Candidatos

- **MongoDB:** Base de datos documental, ideal para datos semi-estructurados como eventos y asistentes. Flexible, con consultas potentes, muy usado en apps de gestión.
- **DynamoDB (AWS):** Opción en la nube altamente escalable, serverless y con baja latencia.

Criterio	MongoDB	DynamoDB
Modelo de datos	Documentos JSON, flexible	Clave-valor y documentos
Escalabilidad	Alta, requiere configuración	Automática (serverless)
Facilidad de uso	Muy extendido, mucha comunidad	Integración profunda con AWS
Costos	Puede ser local o en la nube	Pago por uso (escala sin preocuparse)
Consultas complejas	Muy buenas (filtros, agregaciones)	Más limitado, accesos simples

Cuadro 1: Comparación entre MongoDB y DynamoDB.

5.2. Comparación

5.3. Elección final para nuestro caso: MongoDB

- Permite rediseñar el modelo relacional fácilmente con documentos.
- Ideal para un prototipo simulado (sin depender de AWS).
- Ofrece gran flexibilidad para eventos con distinta información, promociones dinámicas y asistentes con datos variables.
- Existe MongoDB Atlas si después se quiere escalar a la nube.

6. Lenguajes y tecnologías

6.1. Backend

Node.js + Express.js

- Popular en sistemas de eventos, apps web y móviles.
- Integración nativa con MongoDB a través de Mongoose (ODM).
- Permite crear rápido una API RESTful CRUD.

6.2. Base de datos

MongoDB

6.3. API

- Express.js para la API CRUD.
- Postman para pruebas.

6.4. Frontend

React (interfaz web básica) o Volt.

6.5. Infraestructura

Desarrollo: Docker para levantar servicios (MongoDB + API).