Proyecto Final del Trimestre

“Sistema Administrativo de Base de Datos para una Panadería”

Informe de Arquitectura de la Base de Datos

Presentado por

Juan Pablo Daza Alcázar

Johan Sebastián Castro Gonzales

Juan Pablo Mozuca Chaparro

David Santiago Beltran Pedraza

Servicio Nacional de Aprendizaje - Centro de Biotecnología Agropecuaria

Tecnólogo de análisis y desarrollo de software

Mosquera, Cundinamarca, Colombia

(diciembre 2025)

INDICE

INDICE ____ *2*

Justificación Arquitectónica ____ *3*

1. ¿Por qué SQL para este tipo de información? ____ **3**

Respuesta: ____ 3

Transacciones confiables de venta y caja ____ 3

Reglas de negocio e integridad del inventario ____ 3

Reportes operativos del día a día ____ 3

2. ¿Por qué MongoDB para esto? ____ **4**

Respuesta: ____ 4

Catálogo de productos enriquecido ____ 4

Pedidos como “foto completa” de la venta ____ 4

Empleados con historial y documentación ____ 4

3. ¿Por qué Redis Aporta en Términos de Rendimiento o Simplicidad? ____ **5**

Respuesta: ____ 5

Sesiones y autenticación ligeras ____ 5

Métricas en tiempo real sin consultas pesadas ____ 5

Simplicidad en la lógica de negocio ____ 5

Carrito de compras sencillo de manejar ____ 5

Cola de producción sin infraestructura compleja ____ 5

Visibilidad operativa inmediata ____ **6**

Ranking de productos más vendidos al instante ____ 6

Mejor soporte a decisiones rápidas ____ 6

4. Beneficios y Riesgos de la Arquitectura Propuesta de Datos Propuesta ____ **6**

Respuesta: ____ 6

Beneficios de la arquitectura propuesta para el sistema de panadería ____ **6**

Operación diaria más fluida ____ 6

Mejor planificación de producción y compras ____ 7

Experiencia de uso rápida para el personal ____ 7

Evolución del sistema sin romper lo que ya funciona ____ 7

Riesgos para la panadería y su administración ____ **7**

Riesgo de incoherencias entre lo que se vende y lo que se analiza ____ 7

Mayor complejidad de operación y soporte técnico ____ 8

Dependencia de la caché para la operación diaria ____ 8

Riesgo de mal uso de los datos ____ 8

BIBLIOGRAFÍA ____ *8*

Justificación Arquitectónica

1. ¿Por qué SQL para este tipo de información?

Respuesta:

En este proyecto el SQL es el lugar donde se guarda lo oficial, lo exacto y lo que realmente define cuánto vendió y cuánto ganó la panadería o en este sistema de bases de datos.
Las tablas clientes, empleados, productos, pedidos, pedido_detalle y pagos modelan las operaciones diarias de la panadería: quién compra, quién atiende, qué se vende y cuánto se cobra.

Transacciones confiables de venta y caja

Cada vez que un vendedor registra un pedido en el sistema, se crea el encabezado en pedidos, sus líneas en pedido_detalle y el cobro en pagos.
El motor SQL garantiza que todo ese bloque se guarde de forma atómica; si algo falla, no queda un pedido a medias ni pagos sin pedido asociado, algo clave para cuadrar caja y reportes contables.

Reglas de negocio e integridad del inventario

Restricciones como CHECK (stock >= 0), claves foráneas entre pedido_detalle y productos, y campos obligatorios evitan que se vendan productos inexistentes o que se registren montos negativos.
Esto da confianza a la administración: los reportes de stock y de ingresos generados desde SQL (JOINs, CTE de ventas, etc.) reflejan exactamente lo que ocurre en el local.

Reportes operativos del día a día

Consultas como “ventas por cliente”, “productos más pedidos”, “stock total por tipo” o “pedidos atendidos por cada empleado” se resuelven de forma natural con JOINs y GROUP BY.
Son los reportes que el administrador revisa al final del día para saber cuánto se vendió, qué hay que producir mañana y si hay clientes mayoristas que están creciendo.

2. ¿Por qué MongoDB para esto?

Respuesta:

MongoDB se utiliza en nuestro sistema para guardar información que en el mundo real es mucho más flexible y detallada que el modelo puramente transaccional.

Catálogo de productos enriquecido

En SQL se guarda lo mínimo para vender (nombre, tipo, precio, stock), mientras que en MongoDB la colección productos almacena descripción, costo, calorías, tiempo de preparación, lista de ingredientes y proveedores dentro del mismo documento.
Esto le sirve al área de producción y a compras: pueden filtrar rápidamente por ingredientes, revisar qué proveedor suministra qué insumo, o calcular ingresos potenciales y márgenes por tipo de producto usando pipelines de agregación.

Pedidos como “foto completa” de la venta

La colección pedidos guarda, en un solo documento, el snapshot del cliente, los items, descuentos, método de pago, notas y el historial de cambios de estado.
Esta estructura está pensada para el seguimiento operativo: desde una pantalla de detalle de pedido se ve todo el contexto sin múltiples consultas ni joins, y se pueden generar análisis como ventas por cliente, ganancia por ítem o tiempos desde “pendiente” hasta “completado”.

Empleados con historial y documentación

La colección empleados amplía la ficha simple de SQL con especialidades, turnos asignados, historial de producción diario, documentos y capacitaciones.
Esta información no es necesaria para registrar una venta, pero es muy útil para gestión de talento: permite medir productividad (cantidad producida por hora), ver qué panadero domina qué productos y decidir capacitaciones o asignación de turnos según desempeño.

Es decir básicamente el SQL se enfoca en qué se vendió y cuánto, mientras que MongoDB es cómo, con qué ingredientes y con qué personas se logró esa venta, habilitando análisis más cercanos a la realidad de la panadería.

3. ¿Por qué Redis Aporta en Términos de Rendimiento o Simplicidad?

Respuesta:

Básicamente Redis aporta en ese rendimiento de nuestro sistema de administración para la panadería ya que este mismo genera menos carga para SQL y MongoDB y atiende a esos procesos que cambian frecuentemente o no son muy estáticos por así decirlo.

Sesiones y autenticación ligeras

Las sesiones de los usuarios se guardan como STRING con expiración (SETEX sesion:user:404 1800 "token_valido_abcd"), de modo que el sistema no necesita consultar la tabla de clientes en SQL cada vez que un vendedor o cliente hace una petición.
Esto reduce tiempos de respuesta en el login y en las operaciones del día a día, y además libera memoria automáticamente cuando la sesión expira, manteniendo el servidor liviano.

Métricas en tiempo real sin consultas pesadas

Contadores como las visitas a un producto (INCR visitas:pan_masa_madre) se actualizan en memoria, evitando ejecutar UPDATE o SELECT COUNT sobre la base relacional.
Para la panadería esto significa poder mostrar rápidamente, en un panel, qué productos están siendo más vistos en la tienda online o en el sistema interno, sin afectar el rendimiento de las ventas ni los reportes.

Simplicidad en la lógica de negocio

Carrito de compras sencillo de manejar

El carrito se modela como un HASH (HSET carrito:user:101 pan_trigo_integral 5, HSET carrito:user:101 donas_vainilla 10), lo que permite modificar cantidades o agregar productos con operaciones muy simples.
Cuando el cliente confirma la compra, el sistema usa HGETALL carrito:user:101 para construir el pedido definitivo y recién ahí se inicia la transacción ACID en SQL, de forma clara y separando bien “borrador” (Redis) de “venta cerrada” (SQL).

Cola de producción sin infraestructura compleja

La cola de pedidos para el obrador se gestiona con una LIST: RPUSH cola:produccion_dia PED-2025-001 para encolar y LPOP cola:produccion_dia para que el panadero tome el siguiente pedido.
Esto evita diseñar tablas especiales de cola, manejar estados intermedios complicados o

implementar sistemas de mensajería externos; la lógica FIFO queda resuelta con dos comandos muy claros.

Visibilidad operativa inmediata

Ranking de productos más vendidos al instante

Con un ZSET (ZINCRBY top_vendidos 5 pan_trigo_integral), el sistema actualiza el ranking de productos en cuanto una venta se confirma en SQL.
Luego, con ZRANGE top_vendidos 0 -1 REV WITHSCORES, el administrador obtiene al instante el top de productos más vendidos del día o del periodo, sin ejecutar agregaciones pesadas sobre tablas de detalle.

Mejor soporte a decisiones rápidas

Gracias a estas estructuras, el sistema puede mostrar en tiempo real qué productos lideran las ventas, cuántos pedidos hay en cola o cuántos usuarios están activos, lo que ayuda al administrador a decidir si se necesita hornear más, abrir otra caja o priorizar ciertos pedidos grandes.

En conjunto, Redis le da al Sistema de la Panadería esa capacidad muy rápida y sencilla para manejar sesiones, carritos, colas y rankings, permitiendo que SQL se concentre en las ventas oficiales y MongoDB en el análisis profundo, sin que se sacrifique rendimiento ni complicar la lógica de negocio.

4. Beneficios y Riesgos de la Arquitectura Propuesta de Datos Propuesta

Respuesta:

Beneficios de la arquitectura propuesta para el sistema de panadería

Operación diaria más fluida

SQL sostiene el flujo de caja y el inventario: cada venta, pago y ajuste de stock queda registrado con precisión, lo que facilita cierres de turno, arqueo de caja y control de pérdidas.
MongoDB aporta en la parte de visualizar y manejar los pedidos, productos y empleados, de forma que el administrador puede ver, por ejemplo, qué productos generan más margen, qué clientes compran con más frecuencia o qué panadero es más productivo sin afectar el rendimiento de la base transaccional.

Mejor planificación de producción y compras

Con SQL se sabe cuánto se vendió de cada producto; con los pipelines de MongoDB se combinan precios, costos, stock y producción histórica para decidir qué hornear más mañana, qué productos conviene descontinuar o qué insumos hay que pedir con urgencia.
Redis, una vez integrado, permitirá tener contadores en vivo de pedidos del día, pedidos en cola de horneado y productos más pedidos en la última hora, de forma que el jefe de producción pueda reaccionar rápido (por ejemplo, hornear más donas si se disparan las ventas en la tarde).

Experiencia de uso rápida para el personal

El hecho de que las sesiones, carritos y algunas métricas se manejen en memoria hace que las pantallas del sistema respondan rápido incluso en horas pico, sin tiempos de espera largos para consultar SQL o ejecutar agregaciones pesadas.
Para el vendedor esto se traduce en poder atender más clientes por hora; para el administrador, en dashboards que se actualizan prácticamente en tiempo real con información relevante.

Evolución del sistema sin romper lo que ya funciona

El modelo relacional se mantiene estable para todo lo que afecta contabilidad y stock, mientras que MongoDB permite ir agregando nuevos atributos a productos, pedidos o empleados (por ejemplo, etiquetas de marketing, comentarios de clientes, nuevas métricas de calidad) sin migraciones invasivas.
Esto facilita que el sistema crezca con la panadería: se puede empezar con un solo local y, más adelante, incluir sucursales, canales online, programas de fidelización, etc., aprovechando la flexibilidad documental.

Riesgos para la panadería y su administración

Riesgo de incoherencias entre lo que se vende y lo que se analiza

Si por un error de la aplicación un pedido se registra en SQL pero no se replica correctamente en MongoDB, los reportes analíticos podrían no coincidir con la caja, generando decisiones equivocadas (por ejemplo, creer que cierto cliente compra menos de lo real).
Para mitigarlo, el sistema debe tratar a SQL como fuente de verdad y tener flujos claros de sincronización (reintentos, colas de eventos) además de herramientas internas para detectar desajustes y reconstruir datos analíticos a partir de las tablas transaccionales cuando sea necesario.

Mayor complejidad de operación y soporte técnico

El administrador de la panadería no solo depende de que “la base de datos funcione”, sino de que funcionen tres motores distintos; si uno se cae (por ejemplo, Redis), el impacto puede ir desde perder métricas en tiempo real hasta que los carritos web no se carguen correctamente.
Esto exige buenas prácticas: monitorización básica (estado de cada servicio), copias de seguridad de SQL y MongoDB, documentación de procedimientos ante fallos y un entorno de despliegue que simplifique levantar todos los componentes juntos (contenedores, scripts, etc.).

Dependencia de la caché para la operación diaria

Si Redis se usa de forma agresiva para sesiones y contadores, un error en la invalidez de caché puede mostrar información desactualizada en paneles (por ejemplo, stock aparente mayor al real) o mantener carritos caducados.
Para el negocio esto puede significar vender más unidades de las disponibles o tomar decisiones de producción basadas en números erróneos. La mitigación es clara: TTL bien definidos, uso de Redis solo para datos temporales o derivados y siempre recalcular información crítica (como stock y totales de caja) desde SQL cuando se trate de decisiones de alto impacto.

Riesgo de mal uso de los datos

El personal técnico que mantenga el sistema necesita entender qué información está en cada motor y cuándo usarla; de lo contrario, podrían terminar implementando reportes contables sobre MongoDB o tomando decisiones de inventario basadas en datos de caché.
Para la panadería, eso se traduce en posibles errores de gestión. La solución pasa por una buena documentación funcional: para cada módulo (ventas, producción, inventario, RRHH) dejar claro qué fuentes se usan y para qué, además de establecer revisiones periódicas de reportes críticos comparando SQL y MongoDB.

BIBLIOGRAFÍA

- Repositorio del Proyecto: https://github.com/Santiago-Beltran1/Proyecto-Final-de-Trimestre---DB-T3.git