Informe de Diseño de Base de Datos NoSQL para E-Commerce

Proyecto: Taller - Diseño de bases de datos en MongoDB

Integrantes:

- Luis Eduardo Avila Belisle
- [Nombre de Compañero 1]
- [Nombre de Compañero 2]
-

Fecha: 31 de Octubre, 2025

1. Introducción y Filosofía de Diseño

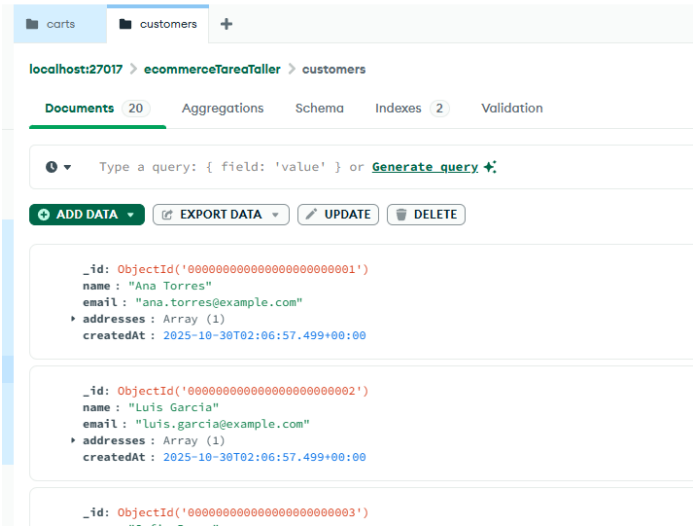
El presente documento detalla el proceso de modelado de datos en MongoDB para una aplicación de e-commerce, basado en tres wireframes clave: Perfil del Cliente, Carrito de Compras y Detalle del Pedido. Nuestra filosofía de diseño se ha centrado en el principio de "Modelado basado en el rendimiento y los casos de uso", una de las prácticas fundamentales de las bases de datos NoSQL. En lugar de adherirnos a una estructura relacional normalizada, hemos diseñado cada documento para que se corresponda directamente con la información que cada vista de la aplicación necesita. Esto nos permite optimizar las operaciones de lectura y escritura para cada caso de uso específico, resultando en una aplicación más rápida y escalable. Las decisiones clave sobre Incrustación vs. Referenciación se tomaron priorizando la eficiencia de las consultas críticas para la experiencia del usuario.

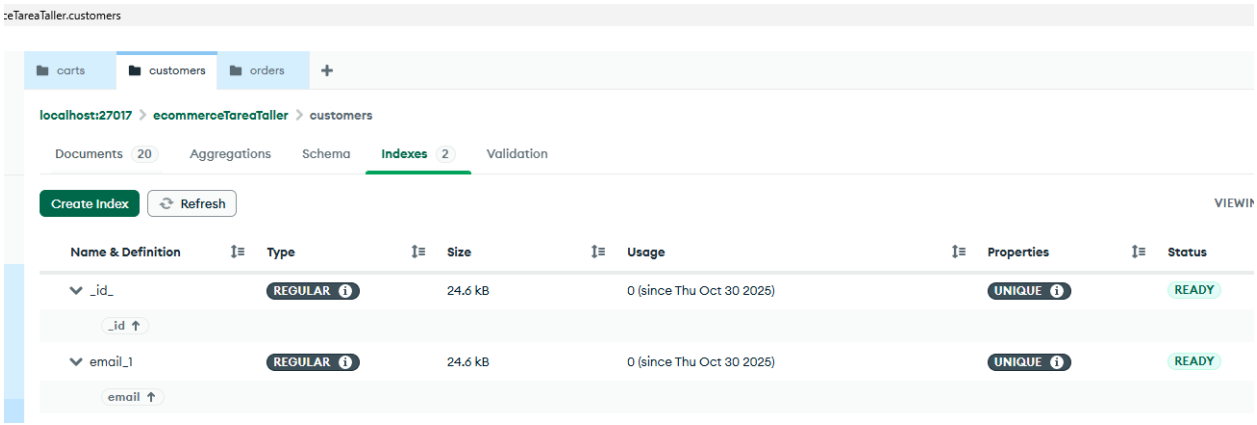
2. Ficha UIF: Perfil del Cliente

Esta ficha describe el modelo de datos diseñado para soportar la vista del perfil del cliente.

- **Caso de Uso y SLA Esperado:**
  - **Caso de Uso:** Un usuario visualiza y actualiza su información personal, direcciones de envío y métodos de pago. La operación principal es la carga rápida y completa de todos sus datos al visitar la página.
  - **SLA Esperado:** p95 < 100ms. La carga del perfil debe ser prácticamente instantánea.
- **Lecturas/Escrituras y Cardinalidad:**
  - **Operaciones:** Predominan las lecturas (findOne) de alta frecuencia. Las escrituras (updateOne) son de baja frecuencia (solo al editar).
  - **Cardinalidad:** La relación entre el cliente y sus direcciones/métodos de pago es uno a pocos (1-to-few). Un cliente típico tendrá un número bajo de direcciones (hogar, oficina) y métodos de pago.
- **Necesidad de Atomicidad:**
  - **Sí.** Acciones como establecer una dirección como predeterminada requieren actualizar varios elementos. Al estar todos los datos dentro de un único documento customer, MongoDB garantiza la atomicidad a nivel de documento, lo que simplifica enormemente estas operaciones sin necesidad de transacciones complejas.
- **Límites y Crecimiento:**
  - El crecimiento de los arrays addresses y paymentMethods es orgánico y limitado, no representa un riesgo de "array ilimitado" (Unbounded Array). El tamaño del documento se mantendrá muy por debajo del límite de 16MB de MongoDB.
- **Decisión Embed vs. Reference (Justificación Técnica):**
  - Se optó por Incrustar (Embed) las direcciones y métodos de pago dentro del documento del cliente. Dada la cardinalidad 1-to-few y el hecho de que estos datos casi siempre se necesitan junto con la información del cliente, este enfoque permite recuperar toda la información necesaria para la vista de perfil con una sola operación de lectura. Esto es fundamental para cumplir el estricto SLA de < 100ms. Crear colecciones separadas habría requerido operaciones $lookup o múltiples consultas, añadiendo latencia innecesaria.

- **Índices Derivados de Consultas Reales:**
    - Se creó un **índice único en { email: 1 }**. La consulta más fundamental para esta entidad es la búsqueda por correo electrónico (para login o recuperación de perfil). Este índice asegura que dicha búsqueda sea extremadamente rápida, como se demuestra en el archivo Consulta_1_Cliente_Email.js, que reporta un IXSCAN con totalDocsExamined: 1.
- **Validación ($jsonSchema):**
    - El schema de validación asegura que todos los clientes tengan los campos obligatorios (name, email), que el email siga un formato válido, y que el array addresses contenga la estructura correcta.
- **Evidencia de Implementación en MongoDB:**
    - "La siguiente imagen muestra la estructura de la colección customers poblada con 20 documentos, donde se puede observar el _id único, el email y el array de direcciones, validando nuestro modelo de incrustación."





3. Ficha UIF: Carrito de Compras

Esta ficha detalla el modelo de datos para el carrito de compras, una entidad altamente transaccional.

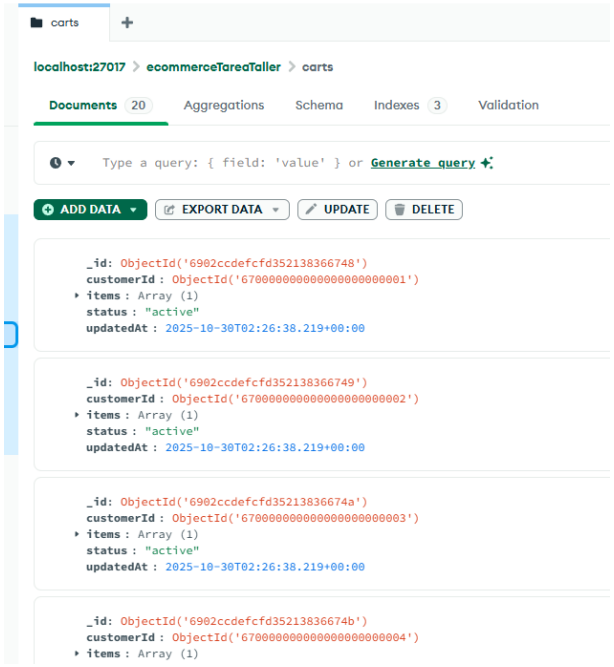
- **Caso de Uso y SLA Esperado:**
  - **Caso de Uso:** Los usuarios agregan productos, actualizan cantidades y eliminan productos de su carrito. La vista debe reflejar estos cambios en tiempo real y cargarse rápidamente para evitar abandonos.
  - **SLA Esperado:** p95 < 150ms. La interactividad debe ser fluida.
- **Lecturas/Escrituras y Cardinalidad:**
  - **Operaciones:** Frecuencia de escrituras muy alta (updateOne con $push, $pull, $inc). Lecturas de alta frecuencia para mostrar el contenido.
  - **Cardinalidad:** 1-a-1 entre un cliente y su carrito activo. 1-a-pocos entre el carrito y sus items.
- **Necesidad de Atomicidad:**
  - **Sí.** Cada modificación al carrito (ej. incrementar la cantidad de un producto) debe ser atómica. Nuestro modelo de items incrustados en un solo documento de carrito aprovecha perfectamente la garantía de atomicidad de MongoDB a nivel de documento para estas operaciones.
- **Límites y Crecimiento:**
  - Para prevenir un crecimiento excesivo, el validador del schema impone un límite de maxItems: 100 en el array de items, cumpliendo la restricción de "arrays con límite razonable".
- **Decisión Embed vs. Reference (Justificación Técnica):**
  - **Embed:** Los items se incrustan porque son intrínsecos al contexto del carrito y siempre se acceden juntos.
  - **Reference**: Se utiliza una Referencia (customerId) al documento del cliente. Esto evita la duplicación de datos del cliente y previene problemas de consistencia si el cliente actualiza su información personal.
- **Índices Derivados de Consultas Reales:**
  - Se creó un **índice único en { customerId: 1 }** para la consulta crítica "obtener el carrito de este usuario", asegurando una recuperación instantánea (validado en Consulta_2_Carrito_ID.js).
  - Se implementó un **índice TTL en { updatedAt: 1 }** con una expiración de 48 horas. Esta es una solución nativa y eficiente de MongoDB para cumplir con el caso de uso "detectar

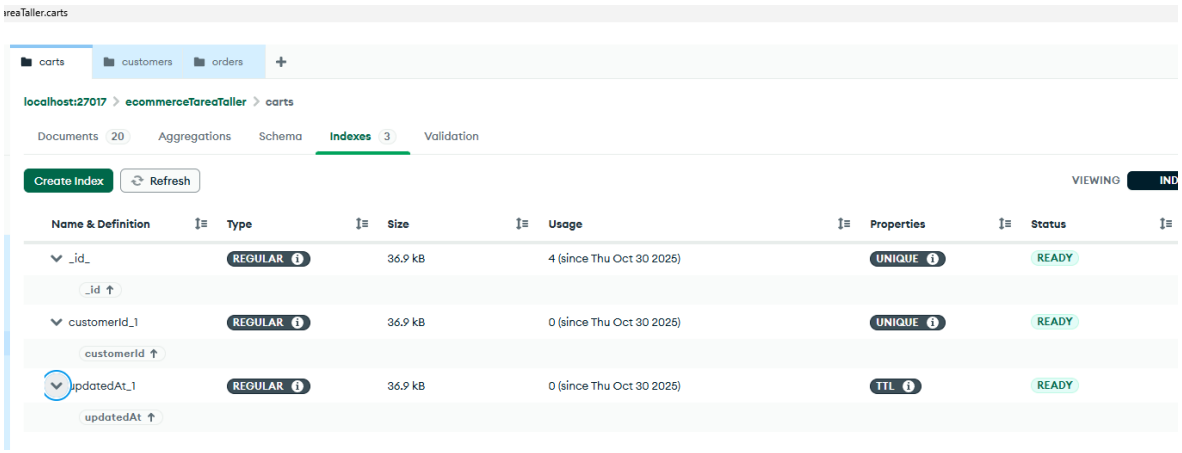
y eliminar carritos abandonados", automatizando la limpieza de la colección sin procesos externos.

- **Validación ($jsonSchema):**
  - El schema fuerza la presencia de un customerId, valida la estructura de cada item incluyendo que price sea double y quantity >= 1, y restringe el campo status.

Evidencia de Implementación en MongoDB:

- "A continuación, se presenta la colección carts, donde cada documento contiene la referencia al customerId y un array de items incrustados. La existencia de 20 carritos confirma la correcta población de datos."





4. Ficha UIF: Detalle del Pedido

Esta ficha explica el modelo para los pedidos, que actúan como registros históricos inmutables.

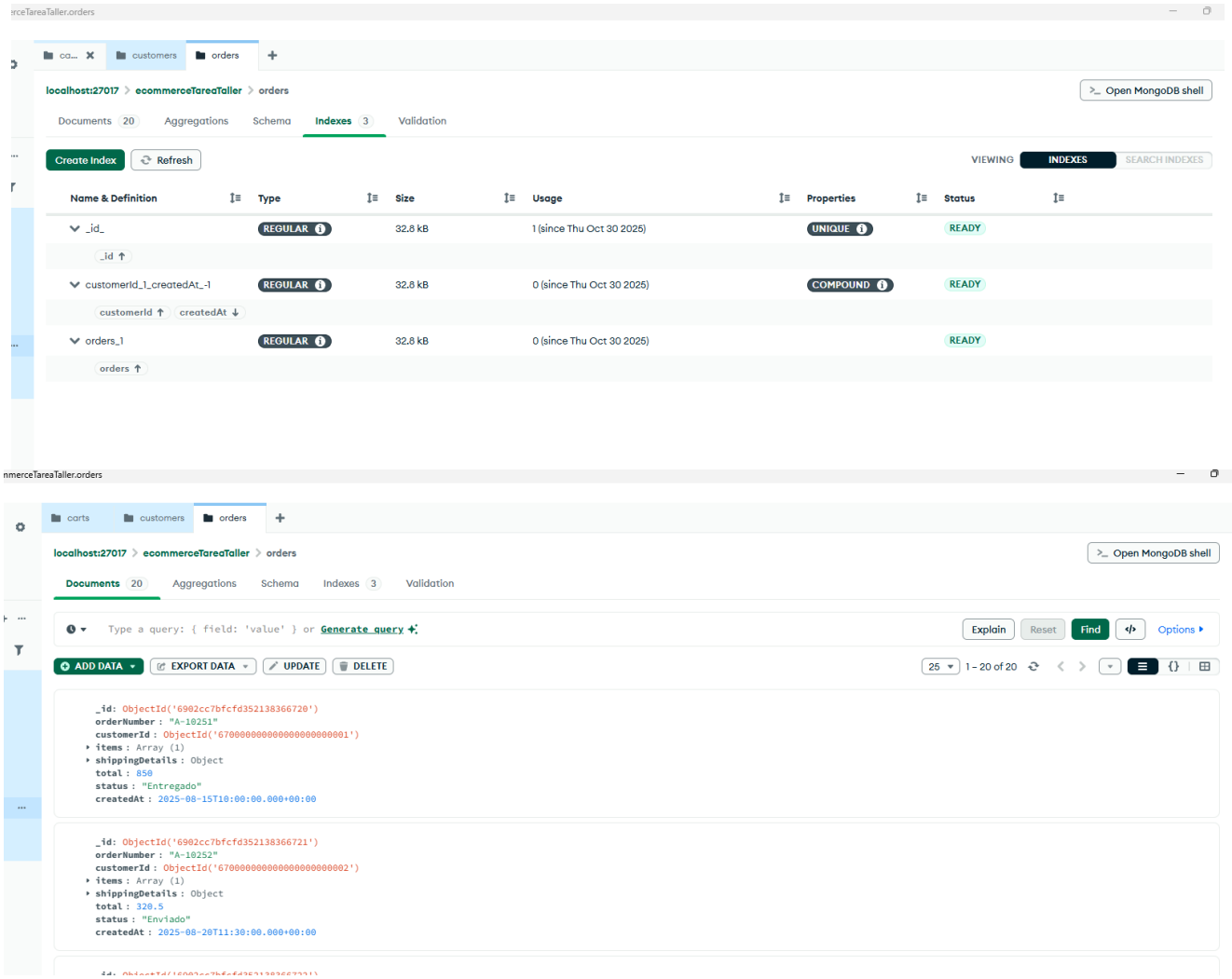
- **Caso de Uso y SLA Esperado:**
  - **Caso de Uso:** Un usuario consulta el historial y los detalles de un pedido pasado. La información mostrada debe ser la "fotografía" exacta del momento en que se realizó la compra.
  - **SLA Esperado:** p95 < 200ms. La consulta no es tan crítica como el carrito, pero debe ser eficiente.
- **Lecturas/Escrituras y Cardinalidad:**
  - **Operaciones:** Las escrituras (insertOne) ocurren una sola vez. Después de la creación, la entidad es de solo lectura (find). Las lecturas son de frecuencia media-alta.
  - **Cardinalidad:** 1-a-muchos (1-to-many) entre un cliente y sus pedidos.
- **Necesidad de Atomicidad:**
  - La creación del pedido es inherentemente atómica al ser una inserción de un solo documento.
- **Decisión Embed vs. Reference (Justificación Técnica):**
  - **Reference:** Se mantiene una **Referencia (customerId)** al cliente que realizó el pedido.
  - **Embed con Denormalización:** Esta es la decisión de diseño más importante aquí. Se incrusta una copia de los detalles del producto (y de la dirección de envío en el momento de la compra. Esta denormalización controlada es intencional y crucial para la integridad histórica. Garantiza que si un producto cambia de precio o un cliente actualiza su dirección, los registros de pedidos pasados permanezcan inalterados y precisos

- **Índices Derivados de Consultas Reales:**
  - Se creó un **índice compuesto en { customerId: 1, createdAt: -1 }**. Este índice optimiza perfectamente el caso de uso "mostrar los últimos pedidos de un cliente". Permite a MongoDB filtrar por customerId y obtener los resultados ya ordenados por fecha de forma nativa desde el índice, eliminando la necesidad de una costosa etapa de ordenamiento en memoria. La eficiencia de este índice se valida en Consulta_3_Pedidos_Cliente.js.
  - Se añadió un índice simple en { status: 1 } para optimizar consultas de back-office o dashboards administrativos.

- **Validación ($jsonSchema):**
    - El schema asegura que todos los campos requeridos para un registro histórico completo estén presentes, incluyendo la estructura de los ítems y los detalles de envío copiados.

Evidencia de Implementación en MongoDB:

- "El modelo de orders se implementó siguiendo un patrón de denormalización controlada. La captura muestra cómo los detalles del pedido, como shippingDetails y los items (con su precio al momento de la compra), se copian en el documento para asegurar la integridad histórica."



Wireframes de Referencia