

Universidad San Pablo de Guatemala  
Facultad de Ingeniería  
Licenciatura en Ingeniería en Sistemas y Ciencias de la  
Computación



Taller - Diseño de bases de datos en MongoDB

Bases de datos II

Jonatan Emanuel Sandoval Guerra

Rodrigo Leonel Bonilla Barreda

Luis Eduardo Ávila Belisle

Dylan Adrián Telón Zarco

1/11/2025

## Introducción

Esta documentación comprende el análisis exhaustivo de un sistema de e-commerce implementado en MongoDB, diseñado para gestionar clientes, carritos de compras y órdenes de pedido. El sistema está estructurado en tres colecciones principales que trabajan de forma integrada para proporcionar una experiencia completa de compra en línea.

## Objetivo del Sistema

El sistema e-commerce está diseñado para manejar el ciclo completo de ventas en línea, desde el registro del cliente hasta la entrega del pedido, incluyendo la gestión de carritos de compras activos y el historial de órdenes.

## Estructura de Colecciones

### 1. Colección customers

- **Propósito:** Gestionar la información de clientes registrados
  - **Características clave:**
    - 20 clientes de prueba con datos realistas
    - Validación estricta de emails con regex
    - Soporte para múltiples direcciones de envío
    - Timestamps de registro para análisis de antigüedad

### 2. Colección carts

- **Propósito:** Manejar carritos de compras activos
  - **Características clave:**
    - Relación 1:1 con clientes
    - Límite de 100 productos por carrito
    - Estados: active, abandoned, converted
    - Actualización en tiempo real con timestamps

### 3. Colección orders

- **Propósito:** Gestionar el historial de pedidos completados
  - **Características clave:**
    - 20 órdenes con datos históricos realistas (Agosto-Octubre 2025)
    - Estados completos del ciclo de vida: Pagado, Enviado, Entregado, Cancelado
    - Información detallada de envío y productos
    - Totales que incluyen costos adicionales

## **Esquemas de Validación**

Cada colección cuenta con un esquema de validación JSON que garantiza:

- **Integridad de datos:** Campos requeridos y tipos correctos
- **Consistencia:** Valores predefinidos para estados y enumeraciones
- **Relaciones válidas:** Referencias entre colecciones
- **Calidad:** Validación de formatos (emails, precios, cantidades)

## **Casos de Uso Cubiertos**

1. **Autenticación y perfil:** Búsqueda rápida de clientes por email
2. **Experiencia de compra:** Gestión eficiente de carritos
3. **Historial y tracking:** Consulta de pedidos con ordenamiento
4. **Dashboard administrativo:** Reportes de ventas y estados
5. **Business Intelligence:** Agregaciones para análisis de ventas

## ecommerceTareaTaller.carts.json

**Propósito:** Almacenar los carritos de compra de los clientes en el sistema

**Estructura del archivo:**

```
{
  "_id": {
    "$oid": "6902ccdefcfd352138366748"
  },
  "customerId": {
    "$oid": "67000000000000000000000001"
  },
  "items": [
    {
      "productId": {
        "$oid": "6902ccdefcfd352138366734"
      },
      "sku": "AUD-001",
      "name": "Auriculares Pro",
      "price": 450,
      "quantity": 1
    }
  ],
  "status": "active",
  "updatedAt": {
    "$date": "2025-10-30T02:26:38.219Z"
  }
}
```

## **Campos explicados:**

### **\_id (Identificador único)**

- Tipo: ObjectId de MongoDB
- Formato: `{ $oid: "hexadecimal_24_caracteres" }`
- Función: Identificador único del carrito en la base de datos
- Ejemplo: "6902ccdefcfd352138366748"

### **customerId (ID del cliente)**

- Tipo: ObjectId referenciando al cliente
- Función: Vincula el carrito con un cliente específico
- Función: Vincula el carrito con un cliente específico

### **items (Productos en el carrito)**

- Tipo: Array de objetos
- Estructura por ítem:
- productId: ObjectId del producto
- sku: Código único del producto (ej: "AUD-001")
- name: Nombre descriptivo del producto
- price: Precio unitario en moneda local
- quantity: Cantidad seleccionada

### **status (Estado del carrito)**

- Tipo: String
- Valor: Siempre "active" en este dataset
- Función: Indica que el carrito está activo y pendiente de compra

### **updatedAt (Fecha de actualización)**

- Tipo: Fecha ISO 8601
- Formato: `{ "$date": "YYYY-MM-DDTHH:mm:ss.sssZ" }`
- Función: Marca la última modificación del carrito

### **Funcionamiento del Sistema**

1. Creación: Se genera automáticamente cuando un cliente agrega el primer producto
2. Actualización: Se modifica al agregar/eliminar productos o cambiar cantidades
3. Persistencia: Mantiene los items seleccionados entre sesiones
4. Conversión: Se convierte en orden al proceder al checkout

### **Características Observadas**

- Cada cliente tiene exactamente un carrito activo
- Los carritos contienen entre 1-5 productos
- Precios varían desde 150 hasta 4500 unidades monetarias
- Todos los carritos fueron actualizados el mismo timestamp

## ecommerceTareaTaller.customers.json

**Propósito:** Almacenar la información de clientes registrados dentro del sistema

### Estructura

```
{
  "_id": {
    "$oid": "00000000000000000000000001"
  },
  "name": "Ana Torres",
  "email": "ana.torres@example.com",
  "addresses": [
    {
      "alias": "Casa",
      "street": "Calle Falsa 123",
      "city": "Ciudad Capital"
    }
  ],
  "createdAt": {
    "$date": "2025-10-30T02:06:57.499Z"
  }
}
```

## **Campos Explicados:**

### **\_id (Identificador único)**

- Tipo: ObjectId de MongoDB
- Rango: 000000000000000000000001 a 000000000000000000000020
- Función: Clave primaria del cliente

### **name (Nombre completo)**

- Tipo: String
- Formato: "Nombre Apellido"
- Función: Nombre completo del cliente para identificación

### **email (Correo electrónico)**

- Tipo: String con formato email
- Dominio: @example.com
- Función: Identificador único para login y comunicación

### **addresses (Direcciones de envío)**

- Tipo: Array de objetos
- Estructura por dirección:
  - alias: Identificador amigable ("Casa", "Trabajo", "Oficina")
  - street: Dirección física completa
  - city: Ciudad de destino

### **createdAt (Fecha de registro)**

- Tipo: Fecha ISO 8601
- Función: Marca cuando el cliente se registró en el sistema



### **Funcionamiento del Sistema**

1. Registro: Se crea un nuevo documento al registrarse un cliente
2. Autenticación: El email sirve como identificador único
3. Gestión de Direcciones: Permite múltiples direcciones de envío
4. Vinculación: Relaciona órdenes y carritos mediante el \_id

### **Características Observadas**

- 20 clientes registrados en total
- 3 ciudades principales: Ciudad Capital, Metropolis, Pueblo Nuevo
- 4 tipos de alias de dirección: Casa, Trabajo, Oficina, Ninguno específico
- Todos registrados el mismo día con timestamps muy cercanos

ecommerceTareaTaller.orders.json

Propósito: Gestión de las ordenes de compra por clientes

```
{
  "_id": {
    "$oid": "6902cc7bfcfd352138366720"
  },
  "orderNumber": "A-10251",
  "customerId": {
    "$oid": "67000000000000000000000001"
  },
  "items": [
    {
      "productId": {
        "$oid": "6902cc7bfcfd35213836670b"
      },
      "name": "Teclado Mecánico",
      "price": 750,
      "quantity": 1
    }
  ],
  "shippingDetails": {
    "address": "Calle Falsa 123",
    "city": "Ciudad Capital"
  },
  "total": 850,
  "status": "Entregado",
  "createdAt": {
    "$date": "2025-08-15T10:00:00.000Z"
  }
}
```

## **Campos Explicados:**

### **\_id (Identificador único)**

- Tipo: ObjectId de MongoDB
- Función: Clave primaria de la orden

### **orderNumber (Número de orden legible)**

- Tipo: String
- Formato: "A-XXXXX" donde X es numérico
- Rango: "A-10251" a "A-10270"
- Función: Número de referencia para el cliente

### **customerId (ID del cliente)**

- Tipo: ObjectId
- Función: Vincula la orden con el cliente que la realizó

### **items (Productos comprados)**

- Tipo: Array de objetos
- Estructura:
  - productId: ObjectId del producto
  - name: Nombre del producto
  - price: Precio unitario al momento de la compra
  - quantity: Cantidad comprada

### **shippingDetails (Información de envío)**

- Tipo: Objeto
- Campos:
  - address: Dirección completa de envío
  - city: Ciudad de destino

**total (Monto total)**

- Tipo: Number
- Función: Suma total de la orden (incluye posiblemente envío/impuestos)
- Observación: No siempre coincide con  $\text{price} * \text{quantity}$

**status (Estado de la orden)**

- Tipo: String
- Valores posibles:
  - "Entregado" - Orden completada exitosamente
  - "Enviado" - En proceso de entrega
  - "Pagado" - Pago confirmado, preparando envío
  - "Cancelado" - Orden cancelada

**createdAt (Fecha de creación)**

- Tipo: Fecha ISO 8601
- Función: Marca cuando se realizó la orden

**Funcionamiento del Sistema**

1. Creación: Se genera al finalizar el proceso de checkout
2. Procesamiento: Pasa por diferentes estados hasta completarse
3. Inmutabilidad: Los datos se congelan al crear la orden
4. Seguimiento: Permite tracking del estado de entrega

**Características Observadas**

- 20 órdenes en el sistema
- Estados distribuidos: 6 Entregado, 6 Pagado, 5 Enviado, 2 Cancelado
- Período de órdenes: Agosto a Octubre 2025
- Algunos clientes tienen múltiples órdenes
- Los totales incluyen costos adicionales (envío/impuestos)

## **Relaciones entre Archivos:**

### **Customer → Cart (1:1)**

- Cada cliente tiene un carrito activo
- Relación: customers.\_id → carts.customerId

### **Customer → Orders (1:N)**

- Un cliente puede tener múltiples órdenes
- Relación: customers.\_id → orders.customerId

### **Cart → Orders (Potencial 1:1)**

- El carrito se convierte en una orden al completar la compra
- No hay relación directa en los datos, pero es el flujo lógico

**Esta estructura permite un flujo completo de e-commerce: registro → carrito → orden → entrega.**

## Consultas

### Consulta 1: Búsqueda de Cliente por Email

**Propósito:** Optimizar la búsqueda de clientes por dirección de correo electrónico, operación crítica para login y recuperación de perfiles.

#### Estructura de la consulta

```
db.customers.find({ email: "ana.torres@example.com" }).explain("executionStats");
```

#### Análisis de Ejecución

##### Plan de Ejecución (winningPlan)

- Estrategia: EXPRESS\_I\_XSCAN (Escaneo rápido de índice)
- Índice Utilizado: email\_1
- Patrón de Clave: { email: 1 }

##### Métricas de Rendimiento

- Documentos Retornados: 1
- Tiempo de Ejecución: 0ms
- Claves Examinadas: 1
- Documentos Examinados: 1
- Eficiencia: 100% (1 documento retornado por 1 examinado)

##### Funcionamiento Detallado

##### Proceso de Búsqueda

1. Acceso Directo al Índice: MongoDB utiliza el índice email\_1 para localizar inmediatamente la entrada correspondiente al email
2. Recuperación Eficiente: Solo se accede al documento específico sin escanear otros registros
3. Respuesta Instantánea: La operación se completa en tiempo despreciable

### **Ventajas del Diseño**

- Índice Único: Garantiza búsquedas  $O(1)$  para emails
- Prevención de Duplicados: Asegura unicidad de cuentas
- Optimización para Login: Operación crítica optimizada al máximo

### **Casos de Uso**

- Autenticación de usuarios
- Recuperación de perfil
- Verificación de existencia de cuenta
- Procesos de checkout

## Consulta 2: Recuperación de Carrito por Customer ID

Propósito: Carga eficiente del carrito de compras de un cliente específico.

### Estructura

```
db.carts.find({ customerId: ObjectId("670000000000000000000003") }).explain("executionStats");
```

### Análisis de Ejecución

#### Plan de Ejecución

- **Estrategia:** EXPRESS\_IXSCAN
- **Índice Utilizado:** customerId\_1
- **Filtro:** customerId: ObjectId("670000000000000000000003")

#### Métricas de Rendimiento

- **Documentos Retornados:** 1
- **Tiempo de Ejecución:** 0ms
- **Claves Examinadas:** 1
- **Documentos Examinados:** 1

### Funcionamiento Detallado

#### Proceso de Recuperación

1. **Búsqueda por Relación:** Utiliza el índice customerId\_1 para encontrar el carrito asociado
2. **Acceso Directo:** Salta directamente al documento del carrito sin escanear otros
3. **Relación 1:1:** Asume un carrito activo por cliente

#### Arquitectura del Modelo

- **Separación de Concerns:** Carritos separados de órdenes completadas
- **Estado Activo:** Solo carritos con status: "active"
- **Actualización en Tiempo Real:** Campo updatedAt para tracking

#### Optimizaciones Clave

- **Índice Dedicado:** customerId\_1 para búsquedas rápidas
- **Estructura Simple:** Documentos ligeros para operaciones frecuentes
- **Cache Amigable:** Fácil de cachear en memoria



### **Consulta 3: Historial de Pedidos con Ordenamiento**

**Propósito:** Recuperar los pedidos más recientes de un cliente, ordenados por fecha de creación.

## Estructura

```
db.orders.find({ customerId: customerIdPedidos })
  .sort({ createdAt: -1 })
  .limit(10)
  .explain("executionStats");
```

### Análisis de Ejecución

#### Plan de Ejecución

- **Estrategia Compuesta:** LIMIT → FETCH → IXSCAN
- **Índice Utilizado:** customerId\_1\_createdAt\_-1 (compuesto)
- **Ordenamiento:** Pre-calculado en el índice

#### Métricas de Rendimiento

- **Documentos Retornados:** 2 (de 2 existentes)
- **Tiempo de Ejecución:** 0ms
- **Claves Examinadas:** 2
- **Documentos Examinados:** 2

### Funcionamiento Detallado

#### Proceso de Tres Etapas

1. IXSCAN: Búsqueda eficiente en índice compuesto
2. FETCH: Recuperación de documentos completos
3. LIMIT: Aplicación del límite después de ordenar

#### Ventajas del Índice Compuesto

- **Ordenamiento Gratuito:** Los datos ya vienen ordenados por createdAt: -1
- **Evita SORT en Memoria:** No necesita operación costosa de ordenamiento
- **Búsqueda Rápida:** Combina filtro y orden en una operación

#### Optimización para UX

- **Paginación Eficiente:** limit(10) para vistas de historial
- **Ordenamiento Natural:** Más recientes primero
- **Rendimiento Constante:** Mismo performance sin importar cantidad de pedidos

#### **Consulta 4: Búsqueda de Pedidos por Estado**

**Propósito** Recuperar todos los pedidos con estado específico para dashboards administrativos.

Estructura

```
db.orders.find({ status: "Enviado" }).explain("executionStats");
```

## Análisis de Ejecución

### Plan de Ejecución

- **Estrategia:** COLLSCAN (Escaneo completo de colección)
- **Filtro Aplicado:** status: "Enviado"
- **Sin Uso de Índice:** A pesar de existir índice status\_1

### Métricas de Rendimiento

- **Documentos Retornados:** 5
- **Tiempo de Ejecución:** 0ms
- **Claves Examinadas:** 0
- **Documentos Examinados:** 20 (toda la colección)

## Análisis de Decisiones del Optimizador

### ¿Por qué COLLSCAN y no IXSCAN?

- **Colección Pequeña:** Solo 20 documentos
- **Costo de Indirección:** Para colecciones pequeñas, es más eficiente escanear directamente
- **Selectividad Baja:**  $5/20 = 25\%$  - no suficientemente selectivo para justificar índice

### Escalabilidad

- **Comportamiento Adaptativo:** Para millones de documentos, MongoDB elegiría automáticamente el índice
- **Umbral de Cambio:** Approx. 10-20% de selectividad

## Consulta 5: Agregación de Ventas por Mes

**Propósito** Generar reportes de ventas mensuales para business intelligence.

Estructura

```
db.orders.aggregate([
  {
    $match: {
      status: { $in: ["Pagado", "Enviado", "Entregado"] }
    }
  },
  {
    $group: {
      _id: {
        year: { $year: "$createdAt" },
        month: { $month: "$createdAt" }
      },
      totalSales: { $sum: "$total" },
      orderCount: { $sum: 1 }
    }
  },
  {
    $sort: {
      "_id.year": 1,
      "_id.month": 1
    }
  }
]).explain();
```

## Análisis de Ejecución

### Pipeline de Tres Etapas

1. \$match: Filtra órdenes completadas (excluye canceladas)
2. \$group: Agrupa por año/mes y calcula métricas

3. \$sort: Ordena cronológicamente

### Plan de Ejecución

- **Estrategia:** COLLSCAN → GROUP → SORT
- **Filtro Inicial:** Reduce dataset antes de agrupación
- **Procesamiento en Memoria:** Cálculos realizados en RAM

### Funcionamiento Detallado

#### Proceso de Agregación

1. **Filtrado Eficiente:** Solo procesa órdenes relevantes (15/20)
2. **Extracción de Fechas:** Convierte createdAt en componentes de año/mes
3. **Agrupación y Cálculo:** Suma totales y cuenta órdenes
4. **Ordenamiento Final:** Organiza resultados por tiempo

#### Optimizaciones para Reportería

- **Filtro Temprano:** \$match al inicio reduce carga
- **Agrupación Natural:** Año/mes como clave natural
- **Métricas Compuestas:** Ventas totales + volumen de órdenes

#### Casos de Uso Empresarial

- **Dashboard de Ventas:** Tendencias mensuales
- **Análisis Estacional:** Patrones de compra
- **Planificación:** Forecast basado en histórico
- **KPIs:** Crecimiento interanual

---

### Resumen de Estrategias de Indexación

#### Índices Críticos Identificados

1. customers.email\_1: Búsqueda por email (login)
2. carts.customerId\_1: Carrito por cliente
3. orders.customerId\_1\_createdAt\_-1: Historial ordenado
4. orders.status\_1: Filtrado por estado (futuro uso)

#### Patrones de Optimización

- **Operaciones Críticas:** Índices dedicados para UX
- **Consultas Administrativas:** COLLSCAN aceptable para datasets pequeños

- **Agregaciones Complejas:** Pipeline optimizado con filtros tempranos
- **Escalabilidad:** Comportamiento adaptativo según tamaño de datos

### **Costumers.js**

**Propósito** Script para poblar la colección customers con datos de clientes de prueba para el sistema e-commerce.

### **Estructura**

```
{
  _id: ObjectId("670000000000000000000001"),
  name: "Ana Torres",
  email: "ana.torres@example.com",
  addresses: [
    {
      alias: "Casa",
      street: "Calle Falsa 123",
      city: "Ciudad Capital"
    }
  ],
  createdAt: new Date()
}
```

## Campos Explicados

### \_id (Identificador único)

- **Tipo:** ObjectId explícito
- **Formato:** ObjectId("670000000000000000000001") a ObjectId("6700000000000000000000000020")
- **Función:** Clave primaria predefinida para facilitar relaciones
- **Patrón:** Secuencial numérico en hexadecimal

### **name** (Nombre completo)

- **Tipo:** String
- **Formato:** "Nombre Apellido"



- **Ejemplos:** "Ana Torres", "Luis Garcia", "Sofia Reyes"
- **Función:** Identificación humana del cliente

#### **email (Correo electrónico)**

- **Tipo:** String con validación de formato
- **Dominio:** @example.com (dominio de prueba)
- **Estructura:** nombre.apellido@example.com
- **Función:** Identificador único para login y comunicación

#### **addresses (Direcciones de envío)**

- **Tipo:** Array de objetos (permite múltiples direcciones)
- **Estructura por dirección:**
  - **alias:** String descriptivo ("Casa", "Trabajo", "Oficina")
  - **street:** Dirección física completa
  - **city:** Ciudad (3 valores: "Ciudad Capital", "Metropolis", "Pueblo Nuevo")

#### **createdAt (Fecha de registro)**

- **Tipo:** Date
- **Valor:** new Date() (fecha/hora actual de inserción)
- **Función:** Auditoría y análisis de antigüedad de clientes

### **Características de los Datos**

#### **Distribución Geográfica**

- **Ciudad Capital:** 8 clientes (40%)
- **Metropolis:** 7 clientes (35%)
- **Pueblo Nuevo:** 5 clientes (25%)

#### **Tipos de Dirección**

- **Casa:** 12 clientes (60%)
- **Trabajo:** 4 clientes (20%)
- **Oficina:** 4 clientes (20%)

#### **Patrones de Nombres**

- **Nombres:** Comunes en español (Ana, Luis, Sofia, Carlos, etc.)
- **Apellidos:** Diversidad hispana (Torres, Garcia, Reyes, Peña, etc.)

#### **Funcionamiento del Script**

## Comando Principal

```
use("ecommerceTareaTaller");  
db.customers.insertMany([...]);
```

## Flujo de Ejecución

1. **Selección de BD:** use("ecommerceTareaTaller")
2. **Inserción Masiva:** insertMany() con array de 20 documentos
3. **Confirmación:** print(">>> 20 clientes insertados.")

## Ventajas del Diseño

- **IDs Predefinidos:** Facilita relaciones con otras colecciones
- **Datos Realistas:** Nombres y estructuras coherentes
- **Escalabilidad:** Fácil de extender con más clientes

## Carts.js

**Propósito:** Script para crear carritos de compras activos asociados a los clientes existentes.

## Estructura

```

{
  customerId: ObjectId("67000000000000000000000001"),
  items: [
    {
      productId: ObjectId(),
      sku: "AUD-001",
      name: "Auriculares Pro",
      price: Double(450),
      quantity: 1
    }
  ],
  status: "active",
  updatedAt: new Date()
}

```

## Campos Explicados

### customerId (Relación con cliente)

- **Tipo:** ObjectId referenciando customers.\_id
- **Rango:** 67000000000000000000000001 a 67000000000000000000000020
- **Función:** Vinculación 1:1 con cliente
- **Relación:** Cada cliente tiene exactamente un carrito activo

### items (Productos en carrito)

- **Tipo:** Array de objetos (carrito puede tener múltiples items)
- **Estructura por ítem:**
  - productId: ObjectId único generado con ObjectId()
  - sku: Código único del producto (ej: "AUD-001", "MON-002")

- name: Nombre descriptivo del producto
- price: Precio unitario como Double() para precisión monetaria
- quantity: Cantidad en carrito (1-5 unidades)

#### **status (Estado del carrito)**

- **Tipo:** String
- **Valor:** Siempre "active"
- **Función:** Indica carritos pendientes de checkout
- **Futuros valores:** "abandoned", "converted", "expired"

#### **updatedAt (Última actualización)**

- **Tipo:** Date
- **Valor:** new Date() (actual al momento de inserción)
- **Función:** Track de actividad del carrito

#### **Catálogo de Productos Incluidos**

##### **Categorías de Productos**

- **Audio:** Auriculares Pro, Micrófono Condensador
- **Pantallas:** Monitor Curvo 27", Webcam HD
- **Componentes:** Disco SSD, Memoria RAM, Tarjeta Gráfica, Procesador, Motherboard
- **Periféricos:** Teclado Mecánico, Mouse Inalámbrico, Mousepad XL
- **Muebles:** Silla Gamer, Escritorio Eléctrico
- **Accesorios:** Gabinete, Fuente de Poder, Cooler, Luces LED, Cables, Hub USB

##### **Rangos de Precios**

- **Económico:** \$150 - \$450 (Mousepad, Cables, Luces LED)
- **Medio:** \$450 - \$900 (Auriculares, Teclado, Webcam, SSD)
- **Alto:** \$900 - \$1800 (Motherboard, Gabinete, Micrófono)
- **Premium:** \$1800+ (Monitor, Tarjeta Gráfica, Procesador, Silla, Escritorio)

#### **Funcionamiento del Script**

##### **Relación con Clientes**

- **Mapeo Directo:** customerId del carrito = \_id del cliente
- **Consistencia:** Todos los clientes tienen su carrito correspondiente
- **Integridad:** No hay carritos sin cliente ni clientes sin carrito

## **Gestión de Inventario**

- **SKUs Únicos:** Cada producto tiene código identificador
- **Precios Precisos:** Uso de Double() para valores monetarios
- **Cantidades Variables:** Simula diferentes patrones de compra

## **Orders.js**

**Propósito:** Script para crear órdenes de compra históricas y actuales del sistema e-commerce.

## **Estructura del Documento**

```
{
  orderNumber: "A-10251",
  customerId: ObjectId("67000000000000000000000001"),
  items: [
    {
      productId: ObjectId(),
      name: "Teclado Mecánico",
      price: Double(750),
      quantity: 1
    }
  ],
  shippingDetails: {
    address: "Calle Falsa 123",
    city: "Ciudad Capital"
  },
  total: Double(850),
  status: "Entregado",
  createdAt: new Date("2025-08-15T10:00:00Z")
}
```

### Campos Explicados

#### orderNumber (Número de orden legible)

- **Tipo:** String
- **Formato:** "A-XXXXXX" (A-10251 a A-10270)

- **Función:** Identificador amigable para clientes
- **Secuencia:** Incremental por fecha de creación

**customerId (Cliente que realizó la orden)**

- **Tipo:** ObjectId referenciando customers.\_id
- **Función:** Relación con cliente
- **Cardinalidad:** Algunos clientes tienen múltiples órdenes

**items (Productos comprados)**

- **Tipo:** Array de objetos
- **Estructura:**
  - productId: ObjectId único
  - name: Nombre del producto
  - price: Precio unitario al momento de la compra
  - quantity: Cantidad comprada (1-3 unidades)

**shippingDetails (Información de envío)**

- **Tipo:** Objeto embebido
- **Campos:**
  - address: Dirección completa (coincide con customers.addresses.street)
  - city: Ciudad de destino

**total (Monto total de la orden)**

- **Tipo:** Double
- **Observación:** No siempre igual a price \* quantity (incluye envío/impuestos)
- **Rango:** \$240 - \$5,200

**status (Estado del ciclo de vida)**

- **Tipo:** String
- **Valores:**
  - "Entregado": Completado exitosamente (6 órdenes)
  - "Enviado": En tránsito (5 órdenes)
  - "Pagado": Confirmado, preparando envío (6 órdenes)
  - "Cancelado": No completado (2 órdenes)

**createdAt (Fecha de creación)**

- **Tipo:** Date con timestamp específico
- **Rango Temporal:** Agosto 2025 - Octubre 2025
- **Formato:** new Date("YYYY-MM-DDTHH:mm:ssZ")

## **Análisis de Datos de Órdenes**

### **Distribución Temporal**

- **Agosto 2025:** 2 órdenes (10%)
- **Septiembre 2025:** 5 órdenes (25%)
- **Octubre 2025:** 13 órdenes (65%)

### **Estados de Órdenes**

- **Completadas:** 12 órdenes (60% - Entregado + Pagado)
- **En Proceso:** 5 órdenes (25% - Enviado)
- **Fallidas:** 2 órdenes (10% - Cancelado)

### **Patrones de Compra**

- **Clientes Recurrentes:**
  - Cliente 001: 2 órdenes
  - Cliente 002: 2 órdenes
- **Órdenes Múltiples Items:** Solo orden A-10264 tiene 2 productos
- **Valores Totales:** Incluyen costos adicionales (envío/impuestos)

## **Funcionamiento del Script**

### **Relaciones con Otras Colecciones**

- **Con Customers:** customerId referencia directa
- **Con Carts:** Relación lógica (carrito → orden)
- **Independencia:** No hay referencias directas a productos específicos

### **Diseño para Reporting**

- **Timestamps Específicos:** Permite análisis temporal preciso
- **Estados Completos:** Cubre todo el ciclo de vida de órdenes
- **Totales Reales:** Incluye costos adicionales para análisis financiero

### **Escenarios de Negocio Cubiertos**

- **Ventas Exitosas:** Órdenes entregadas y pagadas
- **Logística en Proceso:** Órdenes enviadas



- **Problemas Operativos:** Órdenes canceladas
- **Clientes Recurrentes:** Múltiples compras por cliente

## **Resumen de Relaciones y Consistencia**

### **Mapeo de Relaciones**

Customers (20)  $\longleftrightarrow$  Carts (20)  $\longleftrightarrow$  Orders (20)

1:1            1:1            1:N (algunos clientes)

### **Consistencia de Datos**

- **IDs Predecibles:** Facilitan testing y debugging
- **Relaciones Válidas:** Todos los foreign keys existen
- **Datos Realistas:** Precios, cantidades y estados coherentes
- **Coverage Completo:** Todos los escenarios de negocio representados

### **Uso para Desarrollo**

- **Testing:** Datos consistentes para pruebas automatizadas
- **Demo:** Sistema completo funcional para demostraciones
- **Performance:** Volumen adecuado para pruebas de rendimiento
- **Analytics:** Datos suficientes para análisis y reportes

## Validacion\_orders.js

**Propósito:** Define la estructura y reglas de validación para documentos en la colección orders, asegurando la integridad de los datos de pedidos en el sistema e-commerce.

### Estructura del Esquema

#### Campos Requeridos (Obligatorios)

```
required: [  
  'orderNumber',    // Identificador único del pedido  
  'customerId',     // Relación con el cliente  
  'items',          // Productos del pedido  
  'shippingDetails', // Información de envío  
  'total',          // Monto total  
  'status',         // Estado del pedido  
  'createdAt'       // Fecha de creación  
]
```

### Validaciones Detalladas por Campo

#### orderNumber

- **Tipo:** string
- **Función:** Identificador único legible para clientes y sistema
- **Ejemplo:** "A-10251", "A-10252"
- **Uso:** Referencia en comunicaciones con clientes

#### customerId

- **Tipo:** objectId
- **Función:** Vinculación con el cliente que realizó el pedido
- **Integridad:** Garantiza que cada pedido pertenece a un cliente existente
- **Relación:** Referencia a la colección customers

#### items - Array de Productos

- **Tipo:** array

- **Mínimo:** minItems: 1 (el pedido debe tener al menos un producto)
- **Estructura de cada item:**
  - **Campos requeridos:** productId, name, price, quantity
  - productId: objectId (referencia única al producto)
  - name: string (nombre descriptivo del producto)
  - price: double (precisión decimal para valores monetarios)
  - quantity: int con minimum: 1 (cantidad debe ser positiva)

#### shippingDetails - Detalles de Envío

- **Tipo:** object
- **Campos requeridos:** address, city
- **Campos opcionales:** recipientName (nombre del destinatario)
- **Función:** Información completa de destino del pedido
- **Ejemplo:**

```
shippingDetails: {  
  address: "Calle Falsa 123",  
  city: "Ciudad Capital"  
}
```

#### total

- **Tipo:** double
- **Función:** Monto total del pedido con precisión monetaria
- **Consideración:** Debe incluir subtotal, impuestos y costos de envío
- **Precisión:** Uso de double para evitar errores de redondeo

## status

- **Tipo:** enum con valores predefinidos
- **Valores permitidos:**
  - "Pagado" - Pago confirmado, preparando envío
  - "Enviado" - En proceso de entrega
  - "Entregado" - Completado exitosamente
  - "Cancelado" - Pedido cancelado por cliente o sistema
- **Función:** Control del ciclo de vida completo del pedido

## createdAt

- **Tipo:** date
- **Función:** Timestamp de creación para auditoría, reporting y análisis

## Casos de Validación Prácticos

### Documento Válido

```
{
  orderNumber: "A-10251",
  customerId: ObjectId("67000000000000000000000001"),
  items: [{
    productId: ObjectId("507f1f77bcf86cd799439011"),
    name: "Teclado Mecánico",
    price: 750.0,
    quantity: 1
  }],
  shippingDetails: {
    address: "Calle Falsa 123",
    city: "Ciudad Capital"
  },
  total: 850.0,
  status: "Entregado",
  createdAt: new Date("2025-08-15T10:00:00Z")
}
```

### Documentos Inválidos y Razones

- **Falta** orderNumber: Violación de campo requerido
- items: []: Array vacío viola minItems: 1
- quantity: 0: Viola minimum: 1
- status: "Pendiente": No está en la lista de enum
- price: "750": String en lugar de double
- shippingDetails **sin** city: Campo requerido faltante

### Impacto en el Sistema de Órdenes

- **Integridad de Datos:** Previene pedidos incompletos
- **Consistencia de Estados:** Garantiza ciclo de vida coherente
- **Relaciones Válidas:** Asegura referencias a clientes existentes
- **Auditoría Completa:** Timestamps obligatorios para tracking

## Validacion\_carts.js

**Propósito:** Define la estructura de validación para documentos en la colección carts, asegurando la integridad de los carritos de compras.

### Estructura del Esquema Corregido

#### Campos Requeridos

```
required: [  
  'customerId',    // Dueño del carrito  
  'items',         // Productos en el carrito  
  'status',        // Estado del carrito  
  'updatedAt'      // Última actualización  
]
```

### Validaciones Detalladas por Campo

#### customerId

- **Tipo:** objectId
- **Función:** Vinculación 1:1 con el cliente dueño del carrito
- **Relación:** Referencia a la colección customers
- **Cardinalidad:** Un carrito activo por cliente

#### items - Array de Productos en Carrito

- **Tipo:** array
- **Máximo:** maxItems: 100 (límite práctico para evitar carritos enormes)
- **Estructura de cada item:**
  - **Campos requeridos:** productId, sku, name, price, quantity
  - productId: objectId (referencia única al producto)
  - sku: string (Stock Keeping Unit - identificador único del producto)
  - name: string (nombre descriptivo del producto)
  - price: double (precio unitario con precisión decimal)

- quantity: int con minimum: 1 (cantidad mínima de 1 unidad)

#### status

- **Tipo:** enum con valores predefinidos
- **Valores permitidos:**
  - "active" - Carrito en uso activo
  - "abandoned" - Carrito abandonado por el cliente
  - "converted" - Carrito convertido en orden
- **Función:** Gestión del ciclo de vida del carrito

#### updatedAt

- **Tipo:** date
- **Función:** Timestamp de última modificación para limpieza y analytics

### Casos de Validación Prácticos

#### Documento Válido

```
{
  customerId: ObjectId("670000000000000000000001"),
  items: [{
    productId: ObjectId("507f1f77bcf86cd799439011"),
    sku: "AUD-001",
    name: "Auriculares Pro",
    price: 450.0,
    quantity: 1
  }],
  status: "active",
  updatedAt: new Date()
}
```

#### Documentos Inválidos y Razones

- **Falta** customerId: Carrito sin dueño
- items **con más de 100 productos**: Viola maxItems: 100
- **Producto sin sku**: Campo requerido faltante
- quantity: 0: Viola minimum: 1
- status: "pending": No está en la lista de enum
- price: 450: Entero en lugar de double (aunque MongoDB lo convertiría)

#### **Diferencias Clave con la Versión Anterior**

- **CORREGIDO**: Eliminados campos incorrectos (name, email, addresses)
- **CORREGIDO**: Agregados campos correctos (customerId, items, status)
- **MEJORADO**: Límite de maxItems: 100 para control de tamaño
- **MEJORADO**: Estructura específica para ítems de carrito con sku

#### **Impacto en el Sistema de Carritos**

- **Rendimiento**: Límite de 100 items previene carritos demasiado grandes
- **Integridad Relacional**: Garantiza que cada carrito tenga un cliente válido
- **Gestión de Estado**: Permite limpieza automática de carritos abandonados
- **Consistencia de Datos**: SKU obligatorio para integración con inventario



## Validacion\_costumers.js

**Propósito:** Define la estructura de validación para documentos en la colección customers, asegurando la integridad de los datos de clientes.

### Estructura del Esquema

#### Campos Requeridos

```
required: [  
  'name',      // Nombre completo del cliente  
  'email',     // Email único para login  
  'createdAt'  // Fecha de registro  
]
```

### Validaciones Detalladas por Campo

#### name

- **Tipo:** string
- **Descripción:** "debe ser una cadena y es obligatorio"
- **Función:** Identificación humana del cliente
- **Ejemplos:** "Ana Torres", "Luis Garcia"

#### email

- **Tipo:** string con validación de patrón regex
- **Patrón:** `^.+@.+\\..+$`
- **Descripción:** "debe ser un email válido y es obligatorio"
- **Cobertura del Patrón:**
  - usuario@dominio.com
  - nombre.apellido@empresa.com.mx
  - u@d.c (mínimo técnicamente válido)
  - usuario@ (dominio incompleto)
  - usuario.dominio.com (falta @)

- @dominio.com (falta usuario)

#### **addresses - Array de Direcciones (Opcional)**

- **Tipo:** array
- **Descripción:** "debe ser un array de direcciones"
- **Estructura de cada dirección:**
  - **Campos requeridos:** alias, street, city
  - alias: string (identificador amigable: "Casa", "Trabajo", "Oficina")
  - street: string (dirección física completa)
  - city: string (ciudad de destino)

#### **createdAt**

- **Tipo:** date
- **Descripción:** "debe ser una fecha y es obligatorio"
- **Función:** Timestamp de registro para auditoría y análisis de antigüedad

#### **Casos de Validación Prácticos**

```
// Cliente con dirección
{
  name: "Ana Torres",
  email: "ana.torres@example.com",
  addresses: [{
    alias: "Casa",
    street: "Calle Falsa 123",
    city: "Ciudad Capital"
  }],
  createdAt: new Date()
}

// Cliente sin direcciones (válido porque addresses es opcional)
{
  name: "Cliente Nuevo",
  email: "cliente@ejemplo.com",
  createdAt: new Date()
}
```

### Documentos Inválidos y Razones

- name: 123: Número en lugar de string
- email: "invalid-email": No cumple patrón regex
- email: null: Campo requerido faltante
- **Dirección con city faltante:** Campo requerido en subdocumento
- createdAt: "2025-01-01": String en lugar de Date object

### Impacto en el Sistema de Clientes

- **Unicidad:** El email sirve como identificador único para login
- **Flexibilidad:** Direcciones opcionales permiten registro rápido
- **Auditoría:** createdAt obligatorio para análisis de crecimiento
- **Calidad de Datos:** Validación estricta previene datos corruptos

## Script\_final.js

### Propósito

Script completo de configuración e inicialización para un sistema e-commerce en MongoDB que realiza tres funciones principales:

1. Creación de colecciones con validación de esquema
2. Configuración de índices optimizados
3. Población con datos de prueba
4. Validación del funcionamiento del sistema

### Análisis Detallado por Secciones

#### 1. Configuración Inicial y Limpieza

##### Conexión y Preparación

```
use("ecommerceTareaTaller");  
db.customers.drop();  
db.carts.drop();  
db.orders.drop();
```

##### Funcionamiento:

- use(): Selecciona/crea la base de datos "ecommerceTareaTaller"
- drop(): Elimina colecciones existentes para re-inicialización limpia
- **Propósito:** Garantiza un estado conocido antes de la configuración

#### 2. Creación de Colecciones con Validación JSON Schema

##### 2.1 Colección customers

##### Esquema de Validación:

```

{
  bsonType: "object",
  required: ["name", "email", "createdAt"],
  properties: {
    name: { bsonType: "string" },
    email: {
      bsonType: "string",
      pattern: "^.+@.+\\.\\.+$" // Validación regex para email
    },
    addresses: {
      bsonType: "array",
      items: {
        bsonType: "object",
        required: ["alias", "street", "city"],
        properties: {
          alias: { bsonType: "string" },
          street: { bsonType: "string" },
          city: { bsonType: "string" }
        }
      }
    },
    createdAt: { bsonType: "date" }
  }
}

```

#### Decisiones de Diseño:

- **Embedding de Direcciones:** Las direcciones se incrustan como subdocumentos
- **Justificación:** Optimiza lectura del perfil completo con una sola consulta
- **Ventaja:** Elimina necesidad de joins para datos frecuentemente accedidos

#### 2.2 Colección carts

#### Esquema de Validación:

```
{
  bsonType: "object",
  required: ["customerId", "items", "status", "updatedAt"],
  properties: {
    customerId: { bsonType: "objectId" },
    items: {
      bsonType: "array",
      maxItems: 100, // Límite para prevenir arrays ilimitados
      items: {
        bsonType: "object",
        required: ["productId", "sku", "price", "quantity"],
        properties: {
          productId: { bsonType: "objectId" },
          sku: { bsonType: "string" },
          price: { bsonType: "double" },
          quantity: { bsonType: "int", minimum: 1 }
        }
      }
    },
    status: { enum: ["active", "abandoned", "converted"] },
    updatedAt: { bsonType: "date" }
  }
}
```

#### Decisiones de Diseño:

- **Referencia a Cliente:** customerId como ObjectId para relación
- **Límite de Items:** maxItems: 100 previene el anti-patrón de arrays ilimitados
- **Precisión Monetaria:** double para precios evita errores de redondeo

#### 2.3 Colección orders

## Esquema de Validación:

```
{
  bsonType: "object",
  required: ["orderId", "customerId", "items", "shippingDetails", "total", "status", "createdAt"],
  properties: {
    orderId: { bsonType: "string" },
    customerId: { bsonType: "objectId" },
    items: {
      bsonType: "array",
      minItems: 1, // Pedido debe tener al menos un producto
      items: {
        bsonType: "object",
        required: ["productId", "name", "price", "quantity"],
        properties: {
          productId: { bsonType: "objectId" },
          name: { bsonType: "string" },
          price: { bsonType: "double" },
          quantity: { bsonType: "int", minimum: 1 }
        }
      }
    },
    shippingDetails: {
      bsonType: "object",
      required: ["address", "city"],
      properties: {
        recipientName: { bsonType: "string" },
        address: { bsonType: "string" },
        city: { bsonType: "string" }
      }
    },
    total: { bsonType: "double" },
    status: { enum: ["Pagado", "Enviado", "Entregado", "Cancelado"] },
    createdAt: { bsonType: "date" }
  }
}
```

## Decisiones de Diseño:

- **Denormalización Estratégica:** Se copian datos críticos (precios, direcciones)
- **Inmutabilidad:** Los pedidos son "fotografías" que no cambian
- **Integridad Histórica:** Los cambios en productos no afectan pedidos pasados

## 3. Configuración de Índices

### Índices para customers

```
db.customers.createIndex({ email: 1 }, { unique: true });
```

#### Funcionamiento:

- **Índice Único:** Garantiza no duplicados en emails
- **Rendimiento:** Búsquedas  $O(\log n)$  en lugar de  $O(n)$
- **Caso de Uso:** Login y recuperación de perfil

### Índices para carts

```
db.carts.createIndex({ customerId: 1 }, { unique: true });  
db.carts.createIndex({ updatedAt: 1 }, { expireAfterSeconds: 172800 }); // 48 horas
```

#### Funcionamiento:

- **Índice Único:** Un carrito por cliente
- **TTL Index:** Borra automáticamente carritos abandonados después de 48 horas
- **Ventaja:** Elimina necesidad de jobs de limpieza manual

### Índices para orders

```
db.orders.createIndex({ customerId: 1, createdAt: -1 });  
db.orders.createIndex({ status: 1 });
```

#### Funcionamiento:

- **Índice Compuesto:** Optimiza consultas por cliente + ordenamiento por fecha
- **Covering Index:** Evita operaciones SORT en memoria
- **Índice Simple:** Acelera filtrados por estado en dashboards

## 4. Inserción de Datos de Prueba

#### Estrategia de Inserción

```
db.collection.insertMany([...]); // Inserción masiva eficiente
```



### Ventajas:

- **Single Round Trip:** Una sola comunicación con la base de datos
- **Atomicidad:** Todas las inserciones se ejecutan como una unidad
- **Rendimiento:** Mucho más rápido que múltiples insertOne

### Tipos de Datos Específicos

```
Double(450)      // Precisión decimal para valores monetarios
ObjectId()       // Identificadores únicos
new Date()       // Timestamps actuales
new Date("ISO")  // Fechas históricas específicas
```

## 5. Prueba de Validación de Schema

### Caso de Prueba Controlado

```
const invalidCustomer = {
  name: "Usuario De Prueba Sin Email",
  addresses: [],
  createdAt: new Date()
  // Campo 'email' requerido deliberadamente omitido
};

try {
  db.customers.insertOne(invalidCustomer);
} catch (error) {
  // Captura y muestra el error de validación
  printjson(error);
}
```

### Funcionamiento:

- **Error Inducido:** Se viola intencionalmente una regla de validación

- **Try-Catch:** Captura el MongoServerError esperado
- **Verificación:** Confirma que los validadores están activos y funcionando

## Patrones de Diseño Implementados

### 1. Embedding vs Referencing

- **Embedding:** Direcciones en customers, items en carts/orders
- **Referencing:** customerId en carts y orders para relaciones

### 2. Prevención de Anti-patrones

- **Unbounded Arrays:** maxItems: 100 en carritos
- **Data Integrity:** Validación estricta con JSON Schema
- **Performance:** Índices estratégicos para consultas frecuentes

### 3. Manejo de Datos Temporales

- **TTL Index:** Limpieza automática de carritos abandonados
- **Timestamps:** createdAt y updatedAt para auditoría

## Flujo de Ejecución Completo

1. **Preparación** → Limpieza de colecciones existentes
2. **Configuración** → Creación de colecciones con validadores
3. **Optimización** → Configuración de índices
4. **Población** → Inserción de datos de prueba
5. **Validación** → Prueba de funcionamiento de validadores
6. **Confirmación** → Mensaje de éxito final

## Mensajes de Salida y Logging

```
print("Creando colección 'customers' con validación...");
print("\n>>> ¡ÉXITO! La base de datos ha sido creada...");
printjson(error); // Para mostrar errores de forma legible
```

## Propósito:

- **Feedback en Tiempo Real:** Usuario ve el progreso del script
- **Debugging:** Información detallada en caso de errores
- **Confirmación:** Indicadores claros de éxito/failure

## **Consideraciones de Producción**

### **Escalabilidad**

- Los índices compuestos soportan crecimiento de datos
- Los límites de arrays previenen documentos demasiado grandes
- TTL index maneja automáticamente datos temporales

### **Mantenibilidad**

- Validadores documentan la estructura esperada
- Índices nombrados automáticamente por MongoDB
- Script idempotente (puede ejecutarse múltiples veces)

### **Seguridad de Datos**

- Validación a nivel de base de datos
- Tipado estricto previene datos corruptos
- Relaciones referenciales garantizan consistencia