

Documentación técnica

Análisis de Ventas de Librería

1.	Objetivos del análisis	2
2.	Impacto en el negocio	2
3.	Modelo de datos	2
4.	Arquitectura de consultas	3
5.	Características del Código	8
6.	Resultados esperados	9
7.	Optimización y performance	9
8.	Best practices implementadas	10
9.	Información técnica	10
10.	Notas de mantenimiento	11

1. Objetivos del análisis

Problema de Negocio: Identificar autores con ventas significativas y clientes recurrentes para optimizar estrategias comerciales.

Preguntas Clave:

- Autores con ventas superiores a \$500 en categorías específicas
- Clientes VIP con patrones de compra recurrentes por ciudad
- Métricas generales de desempeño del último año

Alcance: Análisis de ventas del último año en categorías 'Ficción' y 'Misterio', clientes con al menos 3 pedidos.

2. Impacto en el negocio

Decisión empresarial: Estrategias de adquisición y retención de clientes, optimización del catálogo.

Métrica afectada: Ventas por categoría, retención de clientes.

3. Modelo de datos

Fuentes de datos: Sistema transaccional de librería (ficticio)

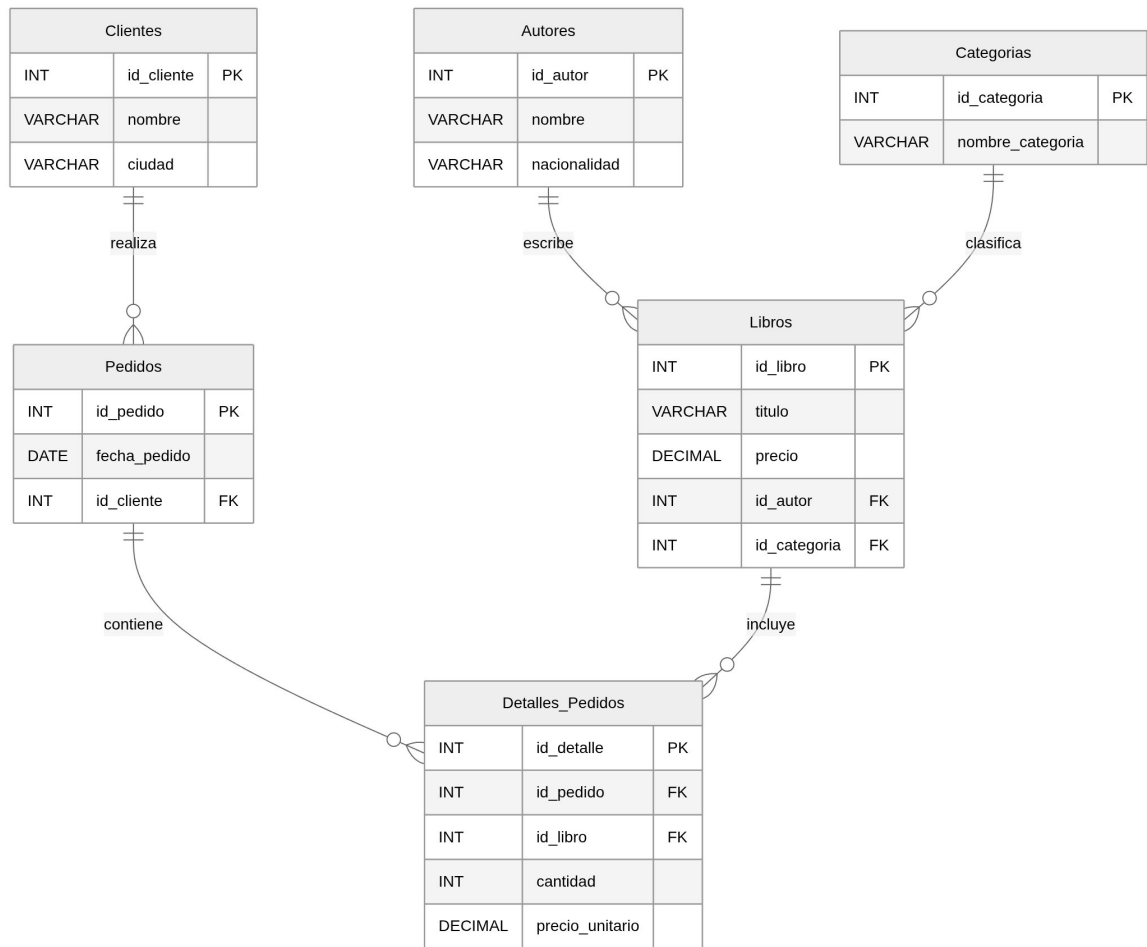
Tablas principales:

- Libros - Autores - Categorías
- Pedidos - Detalles_Pedidos - Clientes

Relaciones clave:

- Libros → Autores (id_autor)
- Libros → Categorías (id_categoria)
- Pedidos → Clientes (id_cliente)
- Detalles_Pedidos → Libros + Pedidos

Esquema de ER



4. Arquitectura de consultas

Consulta 1 - ventas por autor

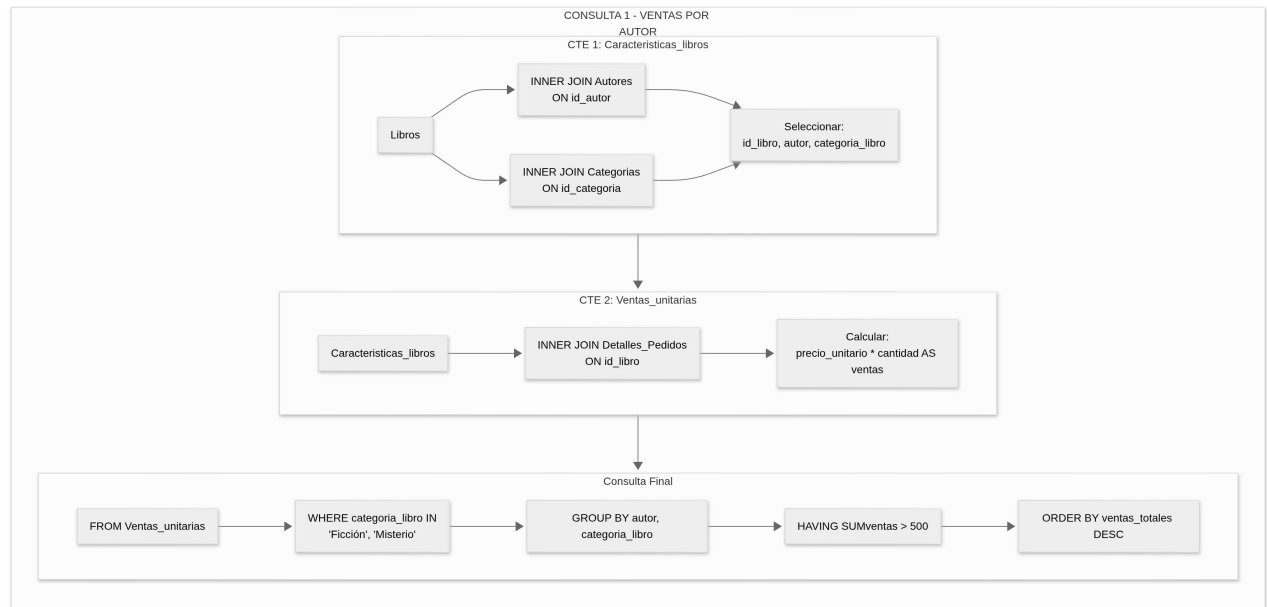
- CTE 1: Caracteristicas_libros
 - Unir libros con autores y categorías
 - WITH
 - Caracteristicas_libros AS (
 - SELECT
 - a.id_libro,
 - b.nombre AS autor,
 - c.nombre_categoria AS categoria_libro
 - FROM Libros a
 - INNER JOIN Autores b ON a.id_autor = b.id_autor
 - INNER JOIN Categorías c ON a.id_categoria = c.id_categoria
 -),

- CTE 2: Ventas_unitarias


```
-- Calcular ventas individuales por libro
Ventas_unitarias AS (
    SELECT
        a.autor AS autor,
        a.categoria_libro AS categoria_libro,
        b.precio_unitario * b.cantidad AS ventas
    FROM Caracteristicas_libros a
    INNER JOIN Detalles_Pedidos b ON a.id_libro = b.id_libro
)
```
- Consulta Final:


```
-- Agrupar y filtrar resultados
SELECT
    autor,
    categoria_libro,
    SUM(ventas) AS ventas_totales
FROM Ventas_unitarias
WHERE categoria_libro IN ('Ficción', 'Misterio') -- Optimizado: IN
en lugar de OR
GROUP BY
    autor,
    categoria_libro
HAVING SUM(ventas) > 500 -- Filtro: solo autores con ventas
significativas
ORDER BY ventas_totales DESC; -- Ordenar por ventas descendentes
```

- Diagrama de bloques:



Consulta 2 - Clientes recurrentes

- CTE 1: Pedidos_recientes


```

-- Filtrar pedidos del último año
WITH
pedidos_recientes AS (
    SELECT
        id_cliente,
        id_pedido,
        fecha_pedido
    FROM Pedidos
    WHERE fecha_pedido >= DATE('now', '-1 year')
    -- FILTRO TEMPORAL: ÚLTIMO AÑO
    -- SQLite/MySQL: DATE('now', '-1 year')
    -- SQL Server: DATEADD(YEAR, -1, GETDATE())
    -- PostgreSQL: CURRENT_DATE - INTERVAL '1 year'
),

```

- CTE 2: clientes_recurrentes
-- Identificar clientes recurrentes
clientes_recurrentes AS (
 SELECT
 id_cliente,
 COUNT(id_pedido) AS total_pedidos
 -- Agregado: Contar pedidos para referencia
 FROM pedidos_recientes
 GROUP BY id_cliente
 HAVING COUNT(id_pedido) >= 3
 -- Criterio de recurrencia: ≥3 pedidos
),
- CTE 3: valor_pedidos
-- Calcular valor total de cada pedido
valor_pedidos AS (
 SELECT
 a.id_cliente,
 a.id_pedido,
 SUM(b.cantidad * b.precio_unitario) AS valor_total_pedido
 FROM Pedidos a
 JOIN Detalles_Pedidos b ON a.id_pedido = b.id_pedido
 WHERE a.fecha_pedido >= DATE('now', '-1 year')
 -- Consistencia: mismo filtro temporal
 GROUP BY
 a.id_pedido,
 a.id_cliente
),

- CTE 4: promedio_por_cliente

```
-- Calcular promedio de pedidos por cliente recurrente
promedio_por_cliente AS (
    SELECT
        c.id_cliente,
        c.total_pedidos, -- Incluir métrica de recurrencia
        AVG(d.valor_total_pedido) AS promedio_valor_pedido,
        SUM(d.valor_total_pedido) AS valor_total_cliente
        -- Agregado: valor total del cliente
    FROM clientes_recurrentes c
    JOIN valor_pedidos d ON c.id_cliente = d.id_cliente
    GROUP BY c.id_cliente, c.total_pedidos
)
```

- Consulta Final:

```
-- Unir con datos demográficos y ordenar
SELECT
    e.ciudad,
    e.nombre AS nombre_cliente,
    p.total_pedidos,
    ROUND(p.promedio_valor_pedido, 2) AS valor_promedio_pedidos,
    ROUND(p.valor_total_cliente, 2) AS valor_total_cliente,
    -- Segmentación basada en el valor promedio
    CASE
        WHEN p.promedio_valor_pedido > 150 THEN 'VIP'
        WHEN p.promedio_valor_pedido > 100 THEN 'Recurrente'
        ELSE 'Activo'
    END AS segmento_cliente
FROM promedio_por_cliente p
JOIN Clientes e ON p.id_cliente = e.id_cliente
ORDER BY
    p.promedio_valor_pedido DESC,
    p.total_pedidos DESC;
```

- Consulta Final:



5. Características del Código

Enfoque técnico: Common Table Expressions (CTEs) para modularidad.

Características:

- Filtros temporales: DATE('now', '-1 year')
- Agregaciones con HAVING
- Segmentación con CASE:

CASE

```

WHEN promedio_valor_pedido > 150 THEN 'VIP'
WHEN promedio_valor_pedido > 100 THEN 'Recurrente'
ELSE 'Activo'

```

END

Compatibilidad: SQLite

Para otros motores, modificar las funciones de tiempo

SQL Server: `DATEADD(YEAR, -1, GETDATE())` → '2023-10-31'

MySQL: `DATE_SUB(NOW(), INTERVAL 1 YEAR)` → '2023-10-31'

6. Resultados esperados

Consulta 1 - Ejemplo de resultados

autor	categoria_libro	ventas_totales
J.K. Rowling	Ficción	1250.00
Stephen King	Misterio	890.50
Agatha Christie	Misterio	745.75

Consulta 2 - Ejemplo de resultados

ciudad	nombre_cliente	total_pedidos	valor_promedio	segmento_cliente
Madrid	María González	5	185.00	VIP
Barcelona	Carlos Rodríguez	4	162.50	VIP
Valencia	Ana Martínez	3	145.75	Recurrente

Métricas resumen

total_clientes	total_autores	total_libros	ventas_totales_anuales	ticket_promedio
150	45	320	25,840.50	172.27

7. Optimización y performance

Índices recomendados

```
CREATE INDEX idx_pedidos_fecha ON Pedidos(fecha_pedido);  
CREATE INDEX idx_detalle_pedido_id ON Detalles_Pedidos(id_pedido);  
CREATE INDEX idx_libros_autor ON Libros(id_autor);  
CREATE INDEX idx_libros_categoria ON Libros(id_categoria);  
CREATE INDEX idx_clientes_ciudad ON Clientes(ciudad);
```

Métricas de performance

- Consulta 1: < 100ms con índices
- Consulta 2: < 200ms (hasta 100K registros)
- Escalabilidad: Optimizado para crecimiento

8. Best practices implementadas

Calidad de código

- Documentación clara con comentarios
- Consultas modulares con CTEs
- Filtros optimizados con IN
- Manejo consistente de tipos de datos

Estrategia de datos

- Validación de integridad referencial
- Filtros temporales para relevancia
- Segmentación basada en métricas de negocio
- Escalabilidad considerada en diseño

9. Información técnica

Stack tecnológico:

- Base de datos: SQLite
- Herramientas: SQL estándar
- Entorno: Desarrollo

Metadatos:

- Autor: Alfonso Droguett
- Fecha: octubre 2025
- Versión: 1.0
- Web: adroguett-portfolio.cl/SQL/SQL-libreria
- Repositorio:
[https://github.com/adroguettth/SQL_Database_Analytics/tree/Analisis de Ventas de Libreria](https://github.com/adroguettth/SQL_Database_Analytics/tree/Analisis%20de%20Ventas%20de%20Libreria)

10. Notas de mantenimiento

Última actualización: [2025-10-31]

Cambios realizados:

- Corrección de sintaxis en JOIN de Autores
- Optimización de filtros con IN
- Adición de métricas adicionales
- Mejora en documentación

Próximas mejoras:

- Funciones de ventana para rankings
- Análisis de tendencias temporales
- Vistas materializadas para reporting

Documentación generada a partir del archivo SQL original - Proyecto de análisis ficticio