Proyecto:

Diseño de Bases de Datos Relacional y No Relacional Para el Restaurante La Terraza

Dairo Rocha Castellanos

Profesora: Tatiana Cabrera Vargas

Ingeniería de Datos
Escuela de Ciencias e Ingeniería
Colegio Mayor de Nuestra Señora del Rosario

Bogotá
2025

Índice

1. Introducción

Este documento presenta la documentación técnica de las implementaciones sugeridas sobre la base de datos relacional del Restaurante La Terraza por los grupos 1, 4, 5 y 7 de la asignatura Ingeniería de Datos liderada por la profesora Tatiana Cabrera de la Universidad del Rosario. Se han desarrollado cinco componentes fundamentales que permiten la gestión eficiente de información mediante vistas, funciones y procedimientos almacenados en MySQL.

Las implementaciones cubren aspectos críticos del negocio incluyendo:

- Gestión de clientes por tipo de servicio
- Consulta de ingredientes del menú
- Control de cuentas por cobrar
- Reportería de ventas diarias

Cada implementación ha sido diseñada siguiendo los requisitos funcionales establecidos en el documento de especificación del proyecto, garantizando una arquitectura robusta, eficiente y fácil de mantener.

2. VISTA RF026 - Listar Clientes por Tipo

2.1. Requisito Funcional

RF026: El sistema debe mostrar todos los clientes filtrados por tipo (Domicilio o Mensualidad).

Grado de necesidad: Medio

2.2. Implementación SQL

```sql
CREATE OR REPLACE VIEW vista_clientes_tipo AS
SELECT
    c.id,
    CONCAT(c.nombre, ' ', c.apellido) AS nombre_completo,
    c.apodo,
    c.direccion,
    c.telefono,
    tc.tipo AS tipo_cliente,
    c.tiene_deuda,
    c.id_tipo_cliente
FROM
    Clientes c
INNER JOIN
    TipoCliente tc ON c.id_tipo_cliente = tc.id
WHERE
    c.id_tipo_cliente IN (2, 3)
ORDER BY
    tc.tipo, c.apellido, c.nombre;
```

Listing 1: Código SQL de la Vista RF026

2.3. Descripción de la Acción

Esta vista materializa una consulta que combina información de las tablas `Clientes` y `TipoCliente` para presentar un listado organizado de todos los clientes que utilizan los servicios de domicilio o mensualidad del restaurante.

La vista realiza las siguientes operaciones:

1. **Unión de tablas:** Relaciona la tabla de clientes con la tabla de tipos mediante una clave foránea, permitiendo obtener el nombre descriptivo del tipo de cliente.
2. **Concatenación de nombres:** Combina los campos de nombre y apellido en un único campo llamado `nombre_completo`, facilitando la lectura de los resultados.
3. **Filtrado específico:** Aplica una condición WHERE que selecciona únicamente los clientes de tipo Domicilio (id=2) y Mensualidad (id=3), excluyendo los clientes de mesa.
4. **Ordenamiento lógico:** Organiza los resultados primero por tipo de cliente, luego alfabéticamente por apellido y nombre.

2.4. Resultados en Result Grid

Al ejecutar la consulta `SELECT * FROM vista_clientes_tipo;`, el sistema genera una tabla con las siguientes columnas:

- **id:** Identificador único del cliente (entero)
- **nombre_completo:** Nombre y apellido concatenados (texto)
- **apodo:** Sobrenombre del cliente (texto, puede ser NULL)
- **direccion:** Dirección de domicilio (texto)
- **telefono:** Número de contacto (texto)
- **tipo_cliente:** Categoría del cliente ("Domicilio" o "Mensualidad")
- **tiene_deuda:** Indicador booleano de deuda pendiente (0 o 1)
- **id_tipo_cliente:** ID numérico del tipo (2 o 3)

Ejemplo de filas resultantes:

- Primera sección: 20 clientes de tipo "Domicilio" ordenados alfabéticamente
- Segunda sección: 30 clientes de tipo "Mensualidad" ordenados alfabéticamente
- Total: 50 registros en el Result Grid

2.5. Consultas Adicionales Implementadas

2.5.1. Filtrar solo clientes de Domicilio

```
SELECT * FROM vista_clientes_tipo
WHERE tipo_cliente = 'Domicilio';
```

Resultado: 20 registros mostrando únicamente clientes que solicitan servicio a domicilio.

2.5.2. Filtrar solo clientes de Mensualidad

```
SELECT * FROM vista_clientes_tipo
WHERE tipo_cliente = 'Mensualidad';
```

Resultado: 30 registros mostrando únicamente clientes que pagan mensualmente.

2.5.3. Filtrar clientes con deuda

```
SELECT * FROM vista_clientes_tipo
WHERE tiene_deuda = TRUE;
```

Resultado: Lista variable de clientes que tienen deudas pendientes, mostrando tanto clientes de domicilio como de mensualidad.

2.5.4. Contar clientes por tipo

```
SELECT tipo_cliente, COUNT(*) AS total_clientes
FROM vista_clientes_tipo
GROUP BY tipo_cliente;
```

Resultado (Result Grid con 2 filas):

- Domicilio: 20 clientes
- Mensualidad: 30 clientes

3. VISTA RF004 - Consultar Ingredientes

3.1. Requisito Funcional

RF004: El sistema debe mostrar los ingredientes de un plato específico del menú.
Entrada: ID del producto del menú
Grado de necesidad: Medio

3.2. Implementación SQL

```sql
CREATE OR REPLACE VIEW vista_ingredientes_plato AS
SELECT
    m.id AS id_plato,
    m.nombre AS plato,
    m.precio,
    c.categoria,
    i.id AS id_ingrediente,
    i.nombre AS ingrediente
FROM
    Menu m
INNER JOIN
    menu_ingredientes mi ON m.id = mi.id_menu
INNER JOIN
    Ingredientes i ON mi.id_ingrediente = i.id
INNER JOIN
    Categorias c ON m.id_categoria = c.id
ORDER BY
    m.nombre, i.nombre;
```

Listing 2: Código SQL de la Vista RF004

3.3. Descripción de la Acción

Esta vista implementa una consulta compleja que resuelve la relación muchos-a-muchos entre platos e ingredientes, permitiendo visualizar la composición detallada de cada producto del menú del restaurante.

La vista ejecuta las siguientes operaciones:

1. **Unión múltiple de tablas:** Conecta cuatro tablas diferentes:
   - `Menu`: Contiene los platos disponibles
   - `menu_ingredientes`: Tabla puente de la relación N:M
   - `Ingredientes`: Lista maestra de ingredientes
   - `Categorias`: Clasificación de platos (Plato, Bebida, Postre)
2. **Desnormalización controlada:** Aunque crea redundancia en los datos del plato (un plato aparece múltiples veces si tiene múltiples ingredientes), esta estructura facilita las consultas y reportes.

3. **Enriquecimiento de información:** Además de los ingredientes, incluye precio y categoría del plato para proporcionar contexto completo.

4. **Ordenamiento doble:** Organiza primero por nombre de plato y luego alfabéticamente por ingrediente, facilitando la lectura.

3.4. Resultados en Result Grid

Al ejecutar `SELECT * FROM vista_ingredientes_plato;`, se obtiene una tabla donde cada fila representa la asociación entre un plato y uno de sus ingredientes.

Estructura de columnas:

- **id_plato:** ID numérico del plato en el menú
- **plato:** Nombre del plato (ej: "Sopa de Cuchuco", "Ajiaco")
- **precio:** Precio en pesos colombianos (formato decimal)
- **categoria:** Tipo de producto ("Plato", "Bebida", "Postre")
- **id_ingrediente:** ID del ingrediente
- **ingrediente:** Nombre del ingrediente (ej: "Res", "Papa", "Pollo")

Ejemplo de registros:

- *Ajiaco - Papa*
- *Ajiaco - Pollo*
- *Ajiaco - Verduras*
- *Bandeja con Alitas - Arroz*
- *Bandeja con Alitas - Ensalada*
- *Bandeja con Alitas - Lenteja*
- *Bandeja con Alitas - Pollo*

Total de registros: Varía según los platos que tengan ingredientes registrados en la tabla puente.

3.5. Consultas Adicionales Implementadas

3.5.1. Consultar ingredientes por ID de plato

```sql
SELECT plato, precio, ingrediente
FROM vista_ingredientes_plato
WHERE id_plato = 1;
```

Resultado (Sopa de Cuchuco - id=1):

- Papa
- Res
- Verduras

3 filas mostrando los tres ingredientes de este plato específico.

3.5.2. Consultar ingredientes por nombre de plato

```
SELECT plato, precio, ingrediente
FROM vista_ingredientes_plato
WHERE plato = 'Ajiaco';
```

Resultado:

- Papa
- Pollo
- Verduras

3 filas con los ingredientes del Ajiaco, junto con su precio (9000.00).

3.5.3. Ver ingredientes agrupados

```
SELECT
    id_plato, plato, precio, categoria,
    GROUP_CONCAT(ingrediente ORDER BY ingrediente
        SEPARATOR ', ') AS ingredientes
FROM vista_ingredientes_plato
GROUP BY id_plato, plato, precio, categoria
ORDER BY plato;
```

Resultado: Una fila por plato con todos sus ingredientes concatenados en una sola celda separados por comas. Ejemplo:
Bandeja Paisa (18000.00): "Arroz, Cerdo, Frijol, Huevo, Plátano, Res"

3.5.4. Buscar platos con ingrediente específico

```
SELECT DISTINCT plato, precio, categoria
FROM vista_ingredientes_plato
WHERE ingrediente = 'Pollo';
```

Resultado: Lista de todos los platos que contienen pollo:

- Ajiaco (9000.00) - Plato
- Bandeja con Alitas (12000.00) - Plato
- Bandeja con Pollo (11000.00) - Plato

4. FUNCIÓN RF037 - Listar Deudores

4.1. Requisito Funcional

RF037: El sistema debe mostrar todos los clientes de mensualidad que tienen deuda pendiente.

Grado de necesidad: Medio

4.2. Implementación SQL

```
DELIMITER //

CREATE FUNCTION contar_deudores()
RETURNS INT
DETERMINISTIC
READS SQL DATA
BEGIN
    DECLARE total_deudores INT;

    SELECT COUNT(*) INTO total_deudores
    FROM Clientes
    WHERE id_tipo_cliente = 3 AND tiene_deuda = TRUE;

    RETURN total_deudores;
END//

DELIMITER ;
```

Listing 3: Código SQL de la Función RF037

4.3. Descripción de la Acción

Esta función almacenada encapsula una lógica de negocio específica para el control financiero del restaurante. Su propósito es contabilizar rápidamente cuántos clientes del tipo "Mensualidad" mantienen deudas pendientes.

El proceso que ejecuta la función es el siguiente:

1. **Declaración de variable local:** Se crea una variable temporal `total_deudores` de tipo entero para almacenar el resultado del conteo.
2. **Consulta SELECT con condiciones múltiples:** Ejecuta un conteo (`COUNT(*)`) sobre la tabla Clientes aplicando dos filtros simultáneos:
   - `id_tipo_cliente = 3`: Selecciona solo clientes de mensualidad
   - `tiene_deuda = TRUE`: Filtra únicamente aquellos con deuda activa
3. **Asignación a variable:** Mediante la cláusula `INTO`, el resultado del conteo se almacena en la variable declarada.

4. **Retorno del valor:** La función devuelve el número entero calculado.

Características técnicas:

- **DETERMINISTIC:** Indica que la función siempre retorna el mismo resultado para el mismo estado de la base de datos.
- **READS SQL DATA:** Especifica que la función solo lee datos, no los modifica.
- **RETURNS INT:** Define el tipo de dato del valor de retorno.

4.4. Resultados en Result Grid

4.4.1. Ejecutar la función directamente

```
SELECT contar_deudores() AS total_deudores;
```

Resultado (Result Grid con 1 fila, 1 columna):

total_deudores
21

Este valor indica que hay 21 clientes de mensualidad con deuda pendiente en el sistema.

4.4.2. Ver listado completo de deudores

```
SELECT
    CONCAT(c.nombre, ' ', c.apellido) AS cliente,
    c.apodo,
    c.telefono,
    cm.deuda_total,
    cm.fecha_corte
FROM Clientes c
INNER JOIN CuentasMensualidad cm ON c.id = cm.id_cliente
WHERE c.id_tipo_cliente = 3 AND c.tiene_deuda = TRUE
ORDER BY cm.deuda_total DESC;
```

Resultado (Result Grid con 21 filas):
Tabla con las siguientes columnas:

- **cliente:** Nombre completo del deudor
- **apodo:** Sobrenombre
- **telefono:** Número de contacto
- **deuda_total:** Monto adeudado (de mayor a menor)
- **fecha_corte:** Fecha límite de pago

Los registros están ordenados de mayor a menor deuda, facilitando la identificación de las cuentas más críticas. Los montos varían típicamente entre 50,000 y 250,000 pesos colombianos según la configuración inicial de datos.

4.5. Aplicaciones Prácticas

Esta función es útil para:

- Generar métricas rápidas en dashboards administrativos
- Automatizar alertas cuando el número de deudores supera umbrales
- Calcular tasas de morosidad (% deudores sobre total de clientes mensualidad)
- Alimentar reportes financieros mensuales

5. PROCEDIMIENTO RF004 - Consultar Ingredientes

5.1. Requisito Funcional

RF004: El sistema debe mostrar los ingredientes de un plato específico del menú.
Entrada: ID del producto del menú
Grado de necesidad: Medio

5.2. Implementación SQL

```sql
DELIMITER //

CREATE PROCEDURE consultar_ingredientes(IN p_id_plato INT)
BEGIN
    SELECT
        m.id AS id_plato,
        m.nombre AS plato,
        m.precio,
        c.categoria,
        i.nombre AS ingrediente
    FROM
        Menu m
    INNER JOIN
        menu_ingredientes mi ON m.id = mi.id_menu
    INNER JOIN
        Ingredientes i ON mi.id_ingrediente = i.id
    INNER JOIN
        Categorias c ON m.id_categoria = c.id
    WHERE
        m.id = p_id_plato
    ORDER BY
        i.nombre;
END//

DELIMITER ;
```

Listing 4: Código SQL del Procedimiento RF004

5.3. Descripción de la Acción

Este procedimiento almacenado implementa el mismo requisito funcional que la vista RF004, pero con una diferencia arquitectónica clave: recibe un parámetro de entrada que especifica qué plato consultar, haciendo la búsqueda más eficiente y directa.

El procedimiento realiza las siguientes operaciones:

1. **Recepción de parámetro:** Acepta un valor entero (`p_id_plato`) que representa el identificador del plato a consultar.

2. **Unión de cuatro tablas:** Similar a la vista RF004, relaciona:
   - Menu
   - menu_ingredientes (tabla puente)
   - Ingredientes
   - Categorias
3. **Filtrado específico:** A diferencia de la vista que muestra todos los platos, este procedimiento aplica una condición `WHERE m.id = p_id_plato` que limita los resultados a un único plato.
4. **Ordenamiento alfabético:** Los ingredientes se presentan ordenados alfabéticamente para facilitar la lectura.
5. **Ejecución y retorno:** Al ser un procedimiento, ejecuta el SELECT y retorna el resultado directamente al cliente sin necesidad de almacenamiento intermedio.

5.4. Resultados en Result Grid

5.4.1. Ejemplo 1: Sopa de Cuchuco (ID = 1)

```
CALL consultar_ingredientes(1);
```

Resultado (Result Grid con 3 filas):

id_plato	plato	precio	categoria	ingrediente
1	Sopa de Cuchuco	8000.00	Plato	Papa
1	Sopa de Cuchuco	8000.00	Plato	Res
1	Sopa de Cuchuco	8000.00	Plato	Verduras

La acción ejecutada es: consultar y mostrar los tres ingredientes principales que componen la Sopa de Cuchuco, presentando además el precio del plato (8000 pesos) y su categoría.

5.4.2. Ejemplo 2: Ajiaco (ID = 2)

```
CALL consultar_ingredientes(2);
```

Resultado (Result Grid con 3 filas):

id_plato	plato	precio	categoria	ingrediente
2	Ajiaco	9000.00	Plato	Papa
2	Ajiaco	9000.00	Plato	Pollo
2	Ajiaco	9000.00	Plato	Verduras

Se obtiene la composición del Ajiaco, plato típico colombiano, mostrando sus tres ingredientes principales y su precio de 9000 pesos.

5.4.3. Ejemplo 3: Bandeja Paisa (ID = 13)

```
CALL consultar_ingredientes(13);
```

Resultado (Result Grid con 6 filas):

id_plato	plato	precio	categoria	ingrediente
13	Bandeja Paisa	18000.00	Plato	Arroz
13	Bandeja Paisa	18000.00	Plato	Cerdo
13	Bandeja Paisa	18000.00	Plato	Frijol
13	Bandeja Paisa	18000.00	Plato	Huevo
13	Bandeja Paisa	18000.00	Plato	Plátano
13	Bandeja Paisa	18000.00	Plato	Res

Para el plato más complejo (Bandeja Paisa), se visualizan los seis ingredientes que lo componen, justificando su precio mayor de 18000 pesos.

5.5. Ventajas del Procedimiento vs Vista

- **Eficiencia:** Solo consulta los datos necesarios para un plato específico
- **Flexibilidad:** Puede extenderse fácilmente para aceptar parámetros adicionales
- **Rendimiento:** Reduce el volumen de datos transferidos al cliente
- **Reutilización:** Puede llamarse desde aplicaciones externas fácilmente

6. PROCEDIMIENTO RF021 - Generar Reporte Diario

6.1. Requisito Funcional

RF021: El sistema debe generar un reporte con todas las ventas realizadas en un día específico.

Entrada: Fecha del reporte

Grado de necesidad: Alto

6.2. Implementación SQL

```
DELIMITER //

CREATE PROCEDURE generar_reporte_diario(IN p_fecha DATE)
BEGIN
    -- Detalle de ventas del d a
    SELECT
        v.folio,
        v.id_orden,
        v.fecha AS fecha_venta,
        o.referencia,
        v.metodo_pago,
        v.referencia_pago,
        v.total,
        CONCAT(p.nombre, ' ', p.apellido) AS atendido_por
    FROM
        Ventas v
    INNER JOIN
        Ordenes o ON v.id_orden = o.id
    LEFT JOIN
        Personal p ON o.id_personal = p.id
    WHERE
        DATE(v.fecha) = p_fecha
    ORDER BY
        v.fecha;

    -- Resumen estad stico del d a
    SELECT
        COUNT(*) AS total_ventas,
        SUM(total) AS total_ingresos,
        AVG(total) AS promedio_venta,
        MIN(total) AS venta_minima,
        MAX(total) AS venta_maxima
    FROM
        Ventas
    WHERE
        DATE(fecha) = p_fecha;
```

```
37 END//
38
39 DELIMITER ;
```

Listing 5: Código SQL del Procedimiento RF021

6.3. Descripción de la Acción

Este procedimiento almacenado es la implementación más compleja de las cinco presentadas, diseñado para generar reportes financieros completos de las operaciones diarias del restaurante. Su importancia radica en que satisface un requisito funcional de alta prioridad para el control administrativo.

El procedimiento ejecuta dos consultas secuenciales que se complementan:

6.3.1. Primera Consulta - Detalle de Ventas

1. **Unión de tres tablas:**
   - `Ventas`: Tabla principal con las transacciones de pago
   - `Ordenes`: Información del pedido asociado
   - `Personal`: Datos del empleado que atendió (JOIN opcional con LEFT JOIN)
2. **Filtrado temporal:** Usa la función `DATE()` para extraer solo la fecha de los timestamps y compararla con el parámetro `p_fecha`.
3. **Información detallada por venta:**
   - Folio de venta (identificador único)
   - ID de la orden relacionada
   - Timestamp exacto de la transacción
   - Referencia (mesa o cliente)
   - Método de pago (Efectivo, Nequi, Daviplata)
   - Referencia de pago (número de teléfono si es digital)
   - Monto total
   - Nombre del empleado que atendió
4. **Ordenamiento cronológico:** Presenta las ventas en orden temporal de ocurrencia.

6.3.2. Segunda Consulta - Resumen Estadístico

Calcula cinco métricas clave del desempeño diario:

1. **Total de ventas:** Número de transacciones realizadas (`COUNT(*)`)
2. **Total de ingresos:** Suma de todos los montos vendidos (`SUM(total)`)
3. **Promedio de venta:** Ticket promedio por transacción (`AVG(total)`)

4. **Venta mínima:** Transacción de menor valor (`MIN(total)`)
5. **Venta máxima:** Transacción de mayor valor (`MAX(total)`)

6.4. Resultados en Result Grid

El procedimiento genera **dos pestañas de resultados** en MySQL Workbench:

6.4.1. Pestaña 1 - Detalle de Ventas

Al ejecutar `CALL generar_reporte_diario(CURDATE());`, suponiendo que existe una venta registrada:

folio	id_orden	fecha_venta	referencia	metodo_pago	total
1	1	2025-10-24 12:30:00	Mesa 5	Efectivo	30000.00

(Continuación de columnas)

referencia_pago	atendido_por
NULL	Andrea Gómez

Descripción de la acción: El sistema recupera y presenta cada transacción de venta realizada en la fecha especificada, mostrando información completa que permite rastrear quién atendió, qué se vendió (mediante la referencia), cómo se pagó y cuánto se cobró. Si el día tiene múltiples ventas, aparecen múltiples filas, cada una representando una transacción independiente.

6.4.2. Pestaña 2 - Resumen del Día

total_ventas	total_ingresos	promedio_venta
1	30000.00	30000.00

(Continuación)

venta_minima	venta_maxima
30000.00	30000.00

Descripción de la acción: El sistema agrega todas las ventas del día en una única fila de métricas consolidadas. Esta vista panorámica permite al administrador evaluar rápidamente el desempeño del restaurante: número de clientes atendidos, ingresos totales, ticket promedio y rango de precios de las ventas.

6.5. Ejemplos de Ejecución

6.5.1. Reporte del día actual

```
CALL generar_reporte_diario(CURDATE());
```

Genera el reporte de ventas del día en curso, útil para monitoreo en tiempo real.

6.5.2. Reporte de fecha específica

```
CALL generar_reporte_diario('2025-10-24');
```

Permite consultar ventas históricas de cualquier día específico para análisis retrospectivo.

6.5.3. Reporte del día anterior

```
CALL generar_reporte_diario(DATE_SUB(CURDATE(), INTERVAL 1 DAY));
```

Genera automáticamente el reporte de ayer, útil para revisiones matutinas del desempeño del día anterior.

6.6. Casos de Uso Administrativos

Este procedimiento es fundamental para:

- **Cierre de caja:** Conciliación diaria de ingresos
- **Control de empleados:** Verificar quién atendió cada venta
- **Análisis de métodos de pago:** Identificar preferencias de los clientes
- **Planificación de inventario:** Correlacionar ventas con consumo
- **Detección de anomalías:** Identificar ventas inusualmente altas o bajas
- **Reportes gerenciales:** Alimentar dashboards de indicadores de negocio

6.7. Consideraciones de Performance

Si un día tiene múltiples ventas, el resultado puede ser extenso. Se recomienda:

- Usar paginación en la aplicación cliente
- Implementar índices en `Ventas.fecha`
- Considerar exportación a CSV para análisis en hojas de cálculo

7. Conclusiones

Las cinco implementaciones presentadas en este documento constituyen un conjunto robusto de herramientas de consulta y análisis para la gestión operativa del Restaurante La Terraza. Cada componente ha sido diseñado con criterios de eficiencia, mantenibilidad y facilidad de uso.

7.1. Logros Principales

1. **Cobertura de requisitos funcionales:** Se implementaron exitosamente los requisitos RF004, RF021, RF026 y RF037, todos clasificados entre prioridad media y alta.

2. **Diversidad de técnicas SQL:** El proyecto demuestra dominio de vistas, funciones y procedimientos almacenados, cada uno aplicado en el contexto más apropiado.

3. **Arquitectura escalable:** Las implementaciones pueden extenderse fácilmente para agregar nueva funcionalidad sin afectar el código existente.

4. **Optimización de consultas:** Uso apropiado de JOINs, índices implícitos y filtrado eficiente.

7.2. Recomendaciones Futuras

Para evolucionar el sistema, se sugiere:

- Implementar vistas materializadas para reportes frecuentes
- Crear triggers para auditoría automática de cambios
- Desarrollar funciones adicionales para cálculos financieros complejos
- Agregar procedimientos para automatización de tareas periódicas
- Implementar índices adicionales en campos frecuentemente consultados

7.3. Impacto en el Negocio

Estas implementaciones permiten:

- Reducción del tiempo de consulta de información crítica
- Mejora en la toma de decisiones basada en datos
- Control financiero más riguroso
- Mejor experiencia de usuario para el personal administrativo
- Base sólida para futura integración con aplicaciones web o móviles

7.4. Cumplimiento de Objetivos

El proyecto ha alcanzado exitosamente los objetivos planteados:

Diseño e implementación de estructuras avanzadas de consulta

Documentación técnica completa y detallada

Validación funcional mediante casos de prueba

Alineación con requisitos funcionales del negocio

Código listo para producción y fácil de mantener