Sistema de Gestión de Pedidos - Documentación Técnica

Bibliotecas Utilizadas

El sistema utiliza un conjunto esencial de bibliotecas de C++ para proporcionar su funcionalidad:

iostream

Proporciona la funcionalidad básica de entrada/salida para la interacción con el usuario a través de la consola. Es fundamental para mostrar información y recibir datos del usuario.

pqxx/pqxx

Es la biblioteca oficial de C++ para PostgreSQL, que permite una interacción robusta y orientada a objetos con la base de datos. En este sistema, se utiliza para gestionar todas las operaciones relacionadas con el almacenamiento y recuperación de datos de pedidos, clientes y productos.

Bibliotecas Adicionales

El sistema hace uso de varias bibliotecas estándar de C++ para mejorar su funcionalidad:

- `vector`: Para el manejo de colecciones dinámicas, especialmente en la gestión de productos en cajones

- `string`: Para el procesamiento de texto y validaciones

- `sstream`: Para conversiones de tipos y formateo de fechas

- `limits`: Para acceder a valores límite en validaciones numéricas

- `algorithm`: Para operaciones de validación como `all\_of`

- `set`: Para manejar conjuntos únicos, como teléfonos registrados

- `iomanip`: Para formateo de números decimales y fechas

- `ctime`: Para el manejo y validación de fechas

Funciones de Utilidad

Funciones de Interfaz

`void limpiarConsola()`

Limpia la pantalla de la consola para mantener una interfaz ordenada. Utiliza el comando "clear" para sistemas Unix/Linux.

`void pausarPrograma()`

Implementa una pausa en la ejecución, esperando que el usuario presione Enter. Mejora la experiencia de usuario permitiendo leer los mensajes antes de continuar.

`void mostrarTitulo(const string& titulo)`

Mejora la presentación visual del sistema, creando encabezados bien formateados con bordes decorativos usando caracteres Unicode.

Funciones de Validación

El sistema implementa un robusto conjunto de funciones de validación:

`bool esCIValido(const string& ci)`

Valida que el número de documento:

- Tenga exactamente 11 dígitos

- Contenga solo caracteres numéricos

`bool esNumeroTelefono(const string& str)`

Asegura que los números telefónicos:

- No estén vacíos

- Contengan solo dígitos

`bool validarFecha(const string& fecha)`

Implementa una validación completa de fechas que:

- Verifica el formato YYYY-MM-DD

- Valida rangos de años (2024-2100)

- Considera meses y días válidos

- Maneja años bisiestos

`bool validarNumeroCuenta(const string& cuenta)`

Valida cuentas bancarias asegurando que:

- Tengan entre 10 y 50 caracteres

- Contengan solo números, letras, guiones y espacios

Estructura de la Base de Datos

El sistema implementa un modelo de datos que representa el negocio de venta de cajones de productos:

Clientes

- ID único autogenerado

- Número de documento (11 dígitos)

- Nombre y apellidos

- Teléfonos asociados (relación uno a muchos)

Proveedores

- ID único autogenerado

- Nombre

- Domicilio

- Días de entrega

Productos

- ID único autogenerado

- Nombre

- Precio por kilogramo

- Proveedor asociado (clave foránea)

- Restricción de unicidad: nombre+proveedor

Cajones

- ID único autogenerado

- Nombre único

- Descripción

- Productos asociados con cantidades

Pedidos

- ID único autogenerado

- Cliente asociado

- Fecha de pedido (automática)

- Fecha de entrega

- Forma de pago (efectivo/transferencia)

- Cajones seleccionados

Normalización de la Base de Datos

Primera Forma Normal (1NF)

El diseño cumple con 1NF porque:

- Cada tabla tiene una clave primaria única (SERIAL PRIMARY KEY)

- Los teléfonos de clientes están normalizados en una tabla separada

- Los productos de cada cajón están en una tabla separada

- Los cajones de cada pedido están en una tabla separada

Segunda Forma Normal (2NF)

El diseño cumple con 2NF porque:

- Cumple con 1NF

- Los datos dependen completamente de sus claves primarias

- Las relaciones entre entidades se manejan mediante claves foráneas

Tercera Forma Normal (3NF)

El diseño cumple con 3NF porque:

- Cumple con 2NF

- No hay dependencias transitivas

- Los datos de transferencia están separados en Detalle\_Pago

- Los productos y proveedores están separados

- Los cajones y sus productos están separados

Funciones Principales del Sistema

Registro de Entidades

`void registrarCliente(connection &c)`

Gestiona el registro completo de nuevos clientes:

- Validación de documento de identidad

- Control de unicidad de documento

- Registro de múltiples teléfonos

- Manejo de transacciones para mantener la integridad

`void registrarCajon(connection &c)`

Implementa el registro de cajones de productos:

- Validación de nombre único

- Control de peso total (máximo 100kg)

- Verificación de productos existentes

- Cálculo de precios

`void registrarPedido(connection &c)`

Maneja la creación de nuevos pedidos:

- Validación de cliente existente

- Control de fechas de entrega

- Manejo de formas de pago

- Cálculo de totales

- Verificación de disponibilidad de cajones

Consultas del Sistema

Todas las funciones de consulta implementan:

- Joins optimizados

- Agrupaciones efectivas

- Formatos de salida claros y organizados

- Manejo de casos nulos

Por ejemplo, `consultarPedidos()` muestra:

- Número de pedido

- Datos del cliente

- Fechas de pedido y entrega

- Forma de pago y cuenta (si aplica)

- Cajones seleccionados

Consideraciones de Diseño

Manejo de Transacciones

El sistema implementa un manejo robusto de transacciones usando bloques de ámbito:

{

work W(c);

try {

// Operaciones de escritura

W.commit();

} catch (...) {

W.abort();

throw;

}

}

Validación de Datos

Implementa múltiples capas de validación:

- Validación de entrada de usuario

- Validación de reglas de negocio

- Restricciones de base de datos

- Manejo de unicidad y referencias

Interfaz de Usuario

Presenta una interfaz profesional con:

- Menús jerárquicos claros

- Bordes y decoraciones consistentes

- Retroalimentación inmediata

- Manejo amigable de errores

- Formato consistente para valores monetarios

El sistema está diseñado para ser robusto, mantenible y fácil de usar, priorizando la integridad de los datos y la experiencia del usuario final.  
  
 Análisis Detallado de Consultas SQL

Inicialización de Tablas

La función `inicializarTablas` establece la estructura completa de la base de datos. Vamos a analizar cada tabla y sus relaciones:

Tabla Clientes

CREATE TABLE IF NOT EXISTS Clientes (

cliente\_id SERIAL PRIMARY KEY,

numero\_documento VARCHAR(11) UNIQUE NOT NULL CHECK (LENGTH(numero\_documento) = 11),

nombre VARCHAR(100) NOT NULL,

apellidos VARCHAR(100) NOT NULL

);

Esta definición incorpora varias características importantes:

- Uso de SERIAL para auto-incremento del ID

- Restricción UNIQUE en número de documento para evitar duplicados

- CHECK para validar la longitud exacta del documento

- Campos obligatorios para garantizar datos completos

Tabla Teléfonos

CREATE TABLE IF NOT EXISTS Telefonos (

cliente\_id INTEGER REFERENCES Clientes(cliente\_id),

numero VARCHAR(20) NOT NULL CHECK (numero ~ '^[0-9]+$'),

PRIMARY KEY (cliente\_id, numero)

);

Aspectos destacables:

- Clave primaria compuesta para evitar números duplicados por cliente

- Expresión regular para validar que el número solo contenga dígitos

- Referencia a Clientes para mantener integridad referencial

Funciones de Registro

Registro de Clientes

La función `registrarCliente` utiliza varias consultas SQL:

1. Verificación de cliente existente:

SELECT cliente\_id

FROM Clientes

WHERE numero\_documento = $1

Esta consulta verifica la existencia previa del documento. Es simple pero crucial para mantener la unicidad.

2. Inserción del cliente:

INSERT INTO Clientes (numero\_documento, nombre, apellidos)

VALUES ($1, $2, $3)

La inserción utiliza parámetros para prevenir inyección SQL.

3. Registro de teléfonos:

INSERT INTO Telefonos (cliente\_id, numero)

SELECT currval('clientes\_cliente\_id\_seq'), $1

Esta consulta es interesante porque:

- Usa currval para obtener el ID del cliente recién insertado

- Mantiene la atomicidad de la transacción

- Permite múltiples inserciones en un bucle manteniendo la relación

Registro de Cajones

La función `registrarCajon` implementa consultas más complejas:

1. Consulta de productos disponibles:

SELECT p.producto\_id, p.nombre, pr.nombre as proveedor,

p.precio\_kg, pr.proveedor\_id

FROM Productos p

JOIN Proveedores pr ON p.proveedor\_id = pr.proveedor\_id

ORDER BY p.nombre, pr.nombre

Esta consulta:

- Une productos con sus proveedores

- Ordena los resultados para mejor visualización

- Obtiene toda la información necesaria en una sola operación

2. Verificación de nombre único:

SELECT 1

FROM Cajones

WHERE nombre = $1

Una consulta simple pero efectiva para verificar duplicados.

3. Inserción del cajón con retorno:

INSERT INTO Cajones (nombre, descripcion)

VALUES ($1, $2)

RETURNING cajon\_id

La cláusula RETURNING es crucial para obtener el ID del nuevo cajón.

4. Verificación e inserción de productos:

SELECT 1

FROM Productos

WHERE producto\_id = $1 AND proveedor\_id = $2

Seguida por:

INSERT INTO Productos\_Cajon (cajon\_id, producto\_id, cantidad\_kg)

VALUES ($1, $2, $3)

Este par de consultas asegura la integridad de los datos antes de la inserción.

Registro de Pedidos

La función `registrarPedido` utiliza las consultas más complejas:

1. Obtención de cajones disponibles:

SELECT c.cajon\_id, c.nombre, c.descripcion,

SUM(pc.cantidad\_kg \* p.precio\_kg) as precio\_total

FROM Cajones c

LEFT JOIN Productos\_Cajon pc ON c.cajon\_id = pc.cajon\_id

LEFT JOIN Productos p ON pc.producto\_id = p.producto\_id

GROUP BY c.cajon\_id, c.nombre, c.descripcion

ORDER BY c.nombre

Esta consulta es compleja porque:

- Calcula precios totales usando datos de múltiples tablas

- Agrupa resultados por cajón

- Usa LEFT JOIN para incluir cajones sin productos

- Ordena los resultados para mejor visualización

2. Búsqueda de cliente:

SELECT cliente\_id, nombre, apellidos

FROM Clientes

WHERE numero\_documento = $1

3. Inserción del pedido:

INSERT INTO Pedidos (cliente\_id, fecha\_entrega, forma\_pago)

VALUES ($1, $2, $3)

4. Registro de detalles de pago (si es transferencia):

INSERT INTO Detalle\_Pago (pedido\_id, cuenta\_origen)

VALUES (currval('pedidos\_pedido\_id\_seq'), $1)

5. Registro de cajones seleccionados:

INSERT INTO Pedidos\_Cajones (pedido\_id, cajon\_id)

VALUES (currval('pedidos\_pedido\_id\_seq'), $1)

Funciones de Consulta

Consulta de Clientes

SELECT c.numero\_documento, c.nombre, c.apellidos,

string\_agg(t.numero, ', ') as telefonos,

COUNT(DISTINCT p.pedido\_id) as total\_pedidos

FROM Clientes c

LEFT JOIN Telefonos t ON c.cliente\_id = t.cliente\_id

LEFT JOIN Pedidos p ON c.cliente\_id = p.cliente\_id

GROUP BY c.cliente\_id, c.numero\_documento, c.nombre, c.apellidos

ORDER BY c.apellidos, c.nombre

Esta consulta demuestra técnicas avanzadas:

- Uso de string\_agg para concatenar teléfonos

- Conteo de pedidos distintos

- Joins múltiples para relacionar toda la información

- Ordenamiento por apellidos y nombre

Consulta de Cajones

SELECT c.cajon\_id, c.nombre, c.descripcion,

COUNT(DISTINCT pc.producto\_id) as total\_productos,

SUM(pc.cantidad\_kg \* p.precio\_kg) as precio\_total

FROM Cajones c

LEFT JOIN Productos\_Cajon pc ON c.cajon\_id = pc.cajon\_id

LEFT JOIN Productos p ON pc.producto\_id = p.producto\_id

GROUP BY c.cajon\_id, c.nombre, c.descripcion

ORDER BY c.nombre

Características destacables:

- Cálculo de totales y cantidades

- Uso de DISTINCT para evitar duplicados

- Múltiples joins para obtener información completa

- Agrupación para estadísticas por cajón

Consulta de Pedidos

SELECT p.pedido\_id, c.numero\_documento, c.nombre, c.apellidos,

p.fecha\_pedido, p.fecha\_entrega, p.forma\_pago,

COUNT(DISTINCT pc.cajon\_id) as total\_cajones

FROM Pedidos p

JOIN Clientes c ON p.cliente\_id = c.cliente\_id

LEFT JOIN Pedidos\_Cajones pc ON p.pedido\_id = pc.pedido\_id

GROUP BY p.pedido\_id, c.numero\_documento, c.nombre, c.apellidos,

p.fecha\_pedido, p.fecha\_entrega, p.forma\_pago

ORDER BY p.fecha\_pedido DESC, p.pedido\_id DESC

Esta consulta muestra:

- Joins múltiples para relacionar toda la información

- Ordenamiento por fecha descendente para mostrar los más recientes primero

- Conteo de cajones por pedido

- Agrupación compleja para mantener la integridad de los datos

Consideraciones de Rendimiento

Todas las consultas han sido diseñadas considerando:

1. Uso apropiado de índices (PK/FK automáticos)

2. Minimización de joins innecesarios

3. Agrupaciones eficientes

4. Selección específica de columnas necesarias

5. Ordenamiento significativo para el usuario final

Seguridad

El sistema implementa varias medidas de seguridad en sus consultas:

1. Uso consistente de consultas parametrizadas

2. Validaciones antes de operaciones de escritura

3. Manejo de transacciones para mantener la integridad

4. Restricciones a nivel de base de datos

5. Verificaciones de existencia antes de inserciones relacionadas