Sistema de Gestión de Taller Mecánico - Documentación Técnica

Bibliotecas Utilizadas

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

Bibliotecas Principales

iostream

Proporciona la funcionalidad básica de entrada/salida para la interacción con el usuario a través de la consola. Se utiliza extensivamente en la interfaz de usuario para mostrar menús, recibir datos y presentar información.

pqxx/pqxx

Es la biblioteca oficial de C++ para PostgreSQL, que permite una interacción robusta con la base de datos. En este sistema, se utiliza para gestionar todas las operaciones relacionadas con clientes, vehículos, reparaciones, mecánicos y repuestos.

Bibliotecas Adicionales

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

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

- `sstream`: Para conversiones de tipos y manejo de entrada/salida en memoria

- `iomanip`: Para el formateo de números decimales en facturas y presentación de datos

- `vector`: Para el manejo de colecciones dinámicas, especialmente en menús y selección de elementos

Funciones de Utilidad

Funciones de Interfaz

`void limpiarPantalla()`

Mantiene una interfaz limpia y profesional, utilizando el comando "clear" para sistemas Unix/Linux.

`void esperarEntrada()`

Implementa una pausa en la ejecución, mejorando la experiencia de usuario al permitir la lectura de mensajes antes de continuar.

Funciones de Validación

El sistema implementa un robusto conjunto de funciones de validación:

`bool validarDNI(const string& str)`

Asegura que los números de identificación:

- Tengan exactamente 9 caracteres

- Contengan 8 dígitos seguidos de una letra

- Sirve para validar la identificación única de clientes

`bool validarMatricula(const string& str)`

Valida las matrículas de vehículos asegurando que:

- Tengan exactamente 7 caracteres

- Los primeros 4 caracteres sean dígitos

- Los últimos 3 caracteres sean letras

Funciones de Entrada de Datos

`template <typename T> T obtenerNumero(const string& mensaje, T min, T max)`

Función genérica que:

- Solicita un número al usuario

- Valida que esté dentro de un rango específico

- Maneja errores de entrada de manera robusta

`string obtenerTexto(const string& mensaje, size\_t minLen, size\_t maxLen)`

Función que:

- Solicita texto al usuario

- Valida la longitud del texto

- Asegura que cumpla con los requisitos mínimos y máximos

Estructura de la Base de Datos

Tablas Principales

Clientes

- DNI (clave primaria)

- Nombre

- Apellidos

- Dirección

- Teléfono

Vehículos

- Matrícula (clave primaria)

- Modelo

- Color

- DNI del cliente (clave foránea)

Mecánicos

- ID de mecánico (clave primaria)

- Nombre

- Apellidos

- Disponibilidad

Reparaciones

- ID de reparación (clave primaria)

- Matrícula de vehículo (clave foránea)

- Fecha de entrada

- Hora de entrada

- Mecánico principal (clave foránea)

- Mano de obra

- Estado

- Fecha de finalización

Repuestos

- ID de repuesto (clave primaria)

- Nombre

- Precio por unidad

Relaciones Entre Tablas

El sistema utiliza tablas de relación para manejar conexiones complejas:

- `Mecanicos\_Reparacion`: Permite asignar múltiples mecánicos a una reparación

- `Repuestos\_Reparacion`: Permite registrar repuestos utilizados en cada reparación

Funciones Principales del Sistema

Registro de Entidades

`void registrarClienteVehiculo(connection &c)`

Gestiona el registro completo de clientes y sus vehículos:

- Valida DNI y matrícula

- Verifica que no existan duplicados

- Registra cliente y vehículo en una transacción única

`void asignarMecanicos(connection &c)`

Implementa la asignación de mecánicos a reparaciones:

- Permite seleccionar un mecánico principal

- Opcional: Asignar mecánicos adicionales

- Actualiza el estado de disponibilidad de los mecánicos

Consultas del Sistema

`void consultarReparacionesCliente(connection &c)`

Ofrece un historial detallado de reparaciones:

- Muestra todas las reparaciones de un cliente

- Incluye detalles como vehículo, mecánicos, fechas y estado

`void consultarEstadoMecanicos(connection &c)`

Proporciona un resumen del estado actual de los mecánicos:

- Lista de mecánicos

- Su disponibilidad

- Reparaciones activas

Funciones de Generación de Documentos

`void generarFactura(connection &c)`

Crea facturas detalladas para reparaciones finalizadas:

- Información del cliente y vehículo

- Detalle de repuestos utilizados

- Cálculo de mano de obra

- Cálculo de IVA

- Presentación en euros y pesetas

Consideraciones de Diseño

Manejo de Transacciones

El sistema utiliza la clase `work` de pqxx para manejar transacciones de manera segura:

work W(c);

try {

// Operaciones de escritura

W.commit();

} catch (const exception &e) {

// Manejo de errores

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 referencias y relaciones

Interfaz de Usuario

Presenta una interfaz profesional con:

- Menús jerárquicos

- Retroalimentación clara

- Formato consistente para valores

- Manejo amigable de errores

Consideraciones de Seguridad y Rendimiento

- Uso de consultas parametrizadas

- Validaciones antes de operaciones de escritura

- Manejo de transacciones para mantener la integridad

- Restricciones a nivel de base de datos

- Verificaciones de existencia antes de inserciones relacionadas

Normalización de la Base de Datos

Principios de Normalización

En el diseño de la base de datos del sistema de gestión de taller mecánico, se han aplicado principios de normalización para garantizar la integridad, eficiencia y reducción de la redundancia de datos.

Primera Forma Normal (1NF)

La base de datos cumple con la Primera Forma Normal porque:

1. Cada tabla tiene una clave primaria única:

- `Clientes`: DNI como clave primaria

- `Vehiculos`: Matrícula como clave primaria

- `Mecanicos`: ID de mecánico como clave primaria

- `Reparaciones`: ID de reparación como clave primaria

- `Repuestos`: ID de repuesto como clave primaria

2. Cada columna contiene valores atómicos (indivisibles):

- Los nombres están separados en campos distintos

- Las direcciones no contienen información anidada

- Los datos están en su forma más simple y no repetitiva

Ejemplo de atomicidad en la tabla `Clientes`:

CREATE TABLE Clientes (

dni VARCHAR(10) PRIMARY KEY,

nombre VARCHAR(50) NOT NULL,

apellidos VARCHAR(100) NOT NULL,

direccion VARCHAR(200) NOT NULL,

telefono VARCHAR(15) NOT NULL

)

Segunda Forma Normal (2NF)

La base de datos cumple con la Segunda Forma Normal porque:

1. Ya cumple con la Primera Forma Normal

2. Todos los atributos no clave dependen completamente de la clave primaria

Ejemplo en la tabla `Reparaciones`:

- La clave primaria es `id\_reparacion`

- `matricula\_vehiculo`, `fecha\_entrada`, `hora\_entrada`, `mecanico\_principal`, `estado` dependen completamente de `id\_reparacion`

Tercera Forma Normal (3NF)

La base de datos cumple con la Tercera Forma Normal porque:

1. Ya cumple con la Segunda Forma Normal

2. No hay dependencias transitivas

3. Cada atributo no clave depende directamente de la clave primaria

Ejemplo de eliminación de dependencias transitivas:

- En lugar de almacenar información del vehículo directamente en `Reparaciones`, se usa una referencia a `Vehiculos`

- La información del mecánico se maneja mediante una referencia a `Mecanicos`

Tablas de Relación y Cardinalidad

Tablas de Relación Muchos a Muchos

Mecanicos\_Reparacion

CREATE TABLE Mecanicos\_Reparacion (

id\_reparacion INTEGER REFERENCES Reparaciones(id\_reparacion),

id\_mecanico INTEGER REFERENCES Mecanicos(id\_mecanico),

PRIMARY KEY (id\_reparacion, id\_mecanico)

)

Función: Permite asignar múltiples mecánicos a una reparación

Repuestos\_Reparacion

CREATE TABLE Repuestos\_Reparacion (

id\_reparacion INTEGER REFERENCES Reparaciones(id\_reparacion),

id\_repuesto INTEGER REFERENCES Repuestos(id\_repuesto),

cantidad INTEGER NOT NULL,

PRIMARY KEY (id\_reparacion, id\_repuesto)

)

Función: Registra los repuestos utilizados en cada reparación con su cantidad

Análisis Detallado de Consultas SQL

1. Registro de Cliente y Vehículo

INSERT INTO Clientes (dni, nombre, apellidos, direccion, telefono)

VALUES ($1, $2, $3, $4, $5)

INSERT INTO Vehiculos (matricula, modelo, color, dni\_cliente)

VALUES ($1, $2, $3, $4)

Objetivos:

- Registrar un nuevo cliente

- Asociar un vehículo al cliente

- Usar consultas parametrizadas para prevenir inyección SQL

- Mantener la integridad referencial entre clientes y vehículos

2. Asignación de Mecánicos a Reparación

INSERT INTO Reparaciones

(matricula\_vehiculo, fecha\_entrada, hora\_entrada, mecanico\_principal, estado)

VALUES ($1, CURRENT\_DATE, CURRENT\_TIME, $2, 'En evaluación')

INSERT INTO Mecanicos\_Reparacion (id\_reparacion, id\_mecanico)

VALUES ($1, $2)

UPDATE Mecanicos SET disponible = false WHERE id\_mecanico = $1

Objetivos:

- Crear un registro de reparación

- Asignar mecánico principal

- Permitir asignación de mecánicos adicionales

- Actualizar el estado de disponibilidad de los mecánicos

3. Consulta de Reparaciones por Cliente

SELECT c.nombre, c.apellidos,

v.matricula, v.modelo,

r.id\_reparacion, r.fecha\_entrada, r.fecha\_fin, r.estado, r.mano\_obra,

m.nombre as mec\_nombre, m.apellidos as mec\_apellidos

FROM Clientes c

JOIN Vehiculos v ON c.dni = v.dni\_cliente

JOIN Reparaciones r ON v.matricula = r.matricula\_vehiculo

JOIN Mecanicos m ON r.mecanico\_principal = m.id\_mecanico

WHERE c.dni = $1

ORDER BY r.fecha\_entrada DESC

Objetivos:

- Recuperar todas las reparaciones de un cliente específico

- Incluir detalles de vehículo, reparación y mecánico principal

- Ordenar por fecha de entrada más reciente

- Usar JOINs para relacionar información de múltiples tablas

4. Generación de Factura

SELECT

r.id\_reparacion, r.fecha\_entrada, r.mano\_obra,

c.dni, c.nombre, c.apellidos, c.direccion,

v.modelo, v.color,

m.nombre as mec\_nombre, m.apellidos as mec\_apellidos

FROM Reparaciones r

JOIN Vehiculos v ON r.matricula\_vehiculo = v.matricula

JOIN Clientes c ON v.dni\_cliente = c.dni

JOIN Mecanicos m ON r.mecanico\_principal = m.id\_mecanico

WHERE v.matricula = $1 AND r.estado = 'Finalizada'

Objetivos:

- Recuperar información completa para generar una factura

- Incluir detalles de cliente, vehículo, reparación y mecánico

- Filtrar solo reparaciones finalizadas

5. Consulta de Estado de Mecánicos

SELECT

m.id\_mecanico, m.nombre, m.apellidos, m.disponible,

COUNT(r.id\_reparacion) as reparaciones\_activas

FROM Mecanicos m

LEFT JOIN Reparaciones r ON m.id\_mecanico = r.mecanico\_principal

AND r.estado != 'Finalizada'

GROUP BY m.id\_mecanico

ORDER BY m.id\_mecanico

Objetivos:

- Obtener el estado actual de todos los mecánicos

- Contar reparaciones activas por mecánico

- Usar LEFT JOIN para incluir mecánicos sin reparaciones

- Agrupar por mecánico para obtener resumen

Consideraciones Finales

El diseño de la base de datos y las consultas SQL del sistema de gestión de taller mecánico se caracterizan por:

1. Normalización rigurosa

2. Integridad referencial

3. Flexibilidad para manejar escenarios complejos

4. Prevención de redundancia de datos

5. Uso de consultas parametrizadas para seguridad

La estructura permite un seguimiento detallado de clientes, vehículos, reparaciones, mecánicos y repuestos, facilitando una gestión eficiente del taller mecánico.