Proyecto Estructura II

Documentacion

Nombre del estudiante:
Jeffrey Torres
Allison Villamil
Ana Reyes
Ruth Reyes

Número de cuenta:
22311308
22251016
22311203
22311294

Sede de estudio:
UNITEC San Pedro Sula

Docente:
Lucas Antonio Cueva Ramos

Sección:
25

Fecha de entrega:
30/2/24

Proyecto Gestión de Tienda

Introducción

Este proyecto es un sistema de gestión para una tienda que permite administrar clientes, productos, pedidos, empleados y ventas, utilizando una tabla de dispersión para el manejo eficiente de datos. A continuación, se detalla la estructura y funcionalidad de todos los archivos del proyecto.

Estructura del Proyecto

El proyecto está organizado en varios archivos de código fuente y cabeceras, cada uno con una responsabilidad específica

1. **archivo binario:** Maneja la lectura y escritura de datos en un archivo binario.
2. **cliente:** Define la clase cliente y sus métodos para gestionar la información de los clientes.
3. **empleado:** Define la clase empleado y sus métodos para gestionar la información de los empleados.
4. **main:** Contiene la función principal del programa y el menú de gestión.
5. **pedido:** Define la clase pedido y sus métodos para gestionar los pedidos.
6. **producto:** Define la clase producto y sus métodos para gestionar los productos.
7. **tabla dispersión:** Implementa una tabla de dispersión (hash table) para el manejo eficiente de datos.
8. **Sistema de gestión:** este contendrá toda la sintaxis para manejar las gestiones de ventas empleados
9. **venta.cpp:** Define la clase venta y sus métodos para gestionar las transacciones de venta.

Estructura del cliente

- idCliente
- nombre
- telefono
- correo
- saldo
- historialCompras

Estructura de Empleado

- id
- nombre
- departamento;
- puesto
- salario
- estado

Estructura de Pedido

- idPedido;
- idCliente;
- productosSolicitados;
- fechaEntrega;
- estado;

Estructura de Producto

- idProducto;
- nombre;
- categoria;
- precio;
- cantidadStock;
- estado

Clases y Funcionalidades

Clase archivobinario (**Allison Villamil**): En esta clase se maneja la apertura, escritura y cierre de un archivo binario.
Métodos:

- archivobinario(const string &nombre): Constructor que abre el archivo con el nombre especificado.
- aggcliente(cliente c): Escribe los datos de un cliente en el archivo.
- Buscarcliente(cliente c): Busca los datos de un cliente en el archivo.
- Aggempleado(empleado e): Agrega los datos de los empleados en el archivo.
- Buscarempleado(empleado e): Busca los datos de los empleados.
- DesactivarEmpleado(empleado e): Pone en estado inactivo a los empleados despedidos.
- Guardarpedido(pedido p): Almacena los pedidos en el archivo binario.
- Gurdarproducto(producto p): almacena los productos en el archivo binario.

- ~archivobinario(): Destructor que cierra el archivo.

Clase cliente (**Ana Reyes y Allison Villamil**): Representar y gestionar la información de los clientes.
Métodos:

- cliente(): Constructor por defecto.
- cliente(int id, string nombre, string telefono, string correo, double saldo, string historial): Constructor con parámetros.
- Setters y Getters para los atributos: idCliente, nombre, telefono, correo, saldo, historialCompras.

Clase empleado (**Ruth Reyes**): implementada para gestionar la información de los empleados.
Métodos:

- empleado(): Constructor por defecto.
- empleado(int id, string nombre, string departamento, string puesto, double salario, string estado): Constructor con parámetros.
- Setters y Getters para los atributos: id, nombre, departamento, puesto, salario, estado.

Clase pedido (**Jeffrey Torres**): creada para gestionar la información de los pedidos.
Métodos:

- pedido (): Constructor por defecto.
- pedido(int idPedido, int idCliente, std::vector<int> productosSolicitados, std::string fechaEntrega, std::string estado): Constructor con parámetros.
- Setters y Getters para los atributos: idPedido, idCliente, productosSolicitados, fechaEntrega, estado.

Clase producto **(Allison Villamil)**: gestiona la información de los productos.
Métodos:

- producto(): Constructor por defecto.
- producto(int id, string nombre, string categoria, double precio, int cantidad, string estado): Constructor con parámetros.
- Setters y Getters para los atributos: idProducto, nombre, categoria, precio, cantidadStock, estado.

Clase tabladispersion **(Ruth Reyes):** Implementa una tabla de dispersión para ser exactos tabla hash para almacenar y gestionar pares clave-valor.
Métodos:

- tabladispersion(): Constructor que inicializa la tabla con un tamaño predefinido.
- int hashFuncion(int clave) const: Calcula el índice en la tabla para una clave dada.
- void insertar(int clave, const std::string& valor): Inserta un par clave-valor en la tabla.
- std::string buscar(int clave) const: Busca un valor en la tabla utilizando la clave.
- void eliminar (int clave): Elimina un par clave-valor de la tabla.

Clase venta **(Jeffrey Torres)**: gestiona la información de las ventas.
Métodos:

- venta (): Constructor por defecto.
- venta (int idVenta, int idCliente, std::vector<int> productosVendidos, std::vector<int> cantidades, double total, std::string fecha): Constructor con parámetros.
- Setters y Getters para los atributos: idVenta, idCliente, productosVendidos, cantidades, total, fecha.

Clase ArbolesB **(Ruth Reyes)**: gestiona el recorrido por el árbol manejando los empleados y los clientes

- recorrer(NodoB* nodo) Recorrer el árbol en orden
- buscar(NodoB* nodo, int clave) Busca la clave en el árbol
- eliminarNodo(NodoB* nodo, int clave) se encarga de eliminar una clave del árbol también está la función eliminar(int clave) que se encargar de eliminar públicamente

Menú Principal main (Ana Reyes)

El menú principal ofrece las siguientes opciones:

1. Gestión de Clientes:
    - Agregar clientes.
    - Almacenar la información de los clientes en un archivo binario.
2. Gestión de Productos:

- Agregar productos.
- Buscar productos.
- Eliminar productos.

3. Gestión de Pedidos:
    - Realizar pedidos.
    - Buscar pedidos.
    - Eliminar pedidos.
4. Gestión de Empleados:
    - Agregar empleados.
    - Eliminar empleados.
5. Gestión de Ventas:
    - Realizar ventas.
    - Registrar los detalles de las ventas, incluyendo productos vendidos, cantidades, total y fecha.
6. Salir: Finaliza la ejecución del programa.

Análisis Empresarial:

Nuestro Proyecto permite la generación de reportes detallados que facilitan la toma de decisiones estrategias dentro de la empresa. Estos reportes permiten evaluar el rendimiento de ventas, identificar tendencias de consumo y analizar la gestión de empleados.

1.Reporte de Ventas Totales por Periodo:

Uso de Árbol B:

-Las ventas se almacenan en un Árbol B, lo que permite realizar consultas eficientes sobre ventas en distintos periodos de tiempo.

2.Reporte de clientes con mayor número de compras

Uso de tablas de dispersión:

-Los clientes y su historial de compras se almacenan en una Tabla Hash, lo que permite acceder de manera eficientes al número de compras realizadas por cada cliente.

-Esta información se usa para identificar a los clientes más recurrentes y aplicar estrategias de fidelización.

3.Reporte de producto más vendidos

Uso de tablas de dispersión:

-se utiliza una tabla hash para registrar y acceder rápidamente a la cantidad de veces que un producto ha sido vendido.

-esta estructura permite generar un ranking de productos más vendidos sin necesidad de recorrer grandes volúmenes de datos.

4.Analisis de Empleados por departamento salario

Uso de Árbol B:

-Los empleados se almacenan en un Árbol B, lo que permite generar reportes ordenados por departamento y rango salarial.

Concurrencia y Sincronización:

Con estas técnicas manejaremos de mejor manera los procesos; por ejemplo que varios empleados podrían estar haciendo compras/ventas al mismo tiempo, o que varios clientes podrían estar realizando varios pedidos, para poder manejar todas estas situaciones de manera eficiente necesitaremos utilizar técnicas como los semáforos, el mutex, y la sincronización.

Semáforos: En nuestro sistema utilizaremos semáforos para dar prioridad a aquellas ventas o procesos principales, como cuando se agregue un nuevo producto o se actualice el inventario.

Mutex: se utilizarán para proteger el acceso a los recursos compartidos en el sistema, como las tablas de dispersión, el árbol B y los archivos binarios. Esto garantiza que solo un hilo a la vez pueda acceder a estos recursos, evitando condiciones de carrera y asegurando la consistencia de los datos. En la clase de ventas, se utilizará un mutex para garantizar que solo un hilo a la vez pueda realizar una venta. Esto es especialmente importante cuando se actualiza el inventario. Cuando se actualiza el inventario (por ejemplo, al agregar o eliminar productos), se utilizará un mutex para proteger el acceso a la tabla de dispersión que almacena los productos.

Sincronización; aquí utilizaremos también los semáforos y técnicas de sincronización para que todos los hilos del proyecto se coordinen de la mejor manera. Utilizaremos “variables de sincronización” que prácticamente paran a un hilo hasta que se cumple una situación/condición específica, como al momento de realizar una venta y tenemos que ver si el producto está disponible o no. También utilizaremos barreras para cuando varios hilos estén procesando diferentes partes de una tarea y se necesite que todos terminen para continuar

Compresión y Recuperación de Datos

Comprensión:

En nuestro sistema, la compresión de daros será especialmente útil para optimizar el almacenamiento de archivos binarios que contienen información de clientes, ventas y productos. Utilizaremos una técnica de comprensión de algoritmos para manejarlo como el algoritmo LZ4 ya que es muy eficiente, en nuestro proyecto los datos de clientes, ventas y productos se comprimirán antes de ser almacenados en archivos binarios. Esto reducirá el espacio de almacenamiento requerido y mejorará el rendimiento al leer y escribir datos.

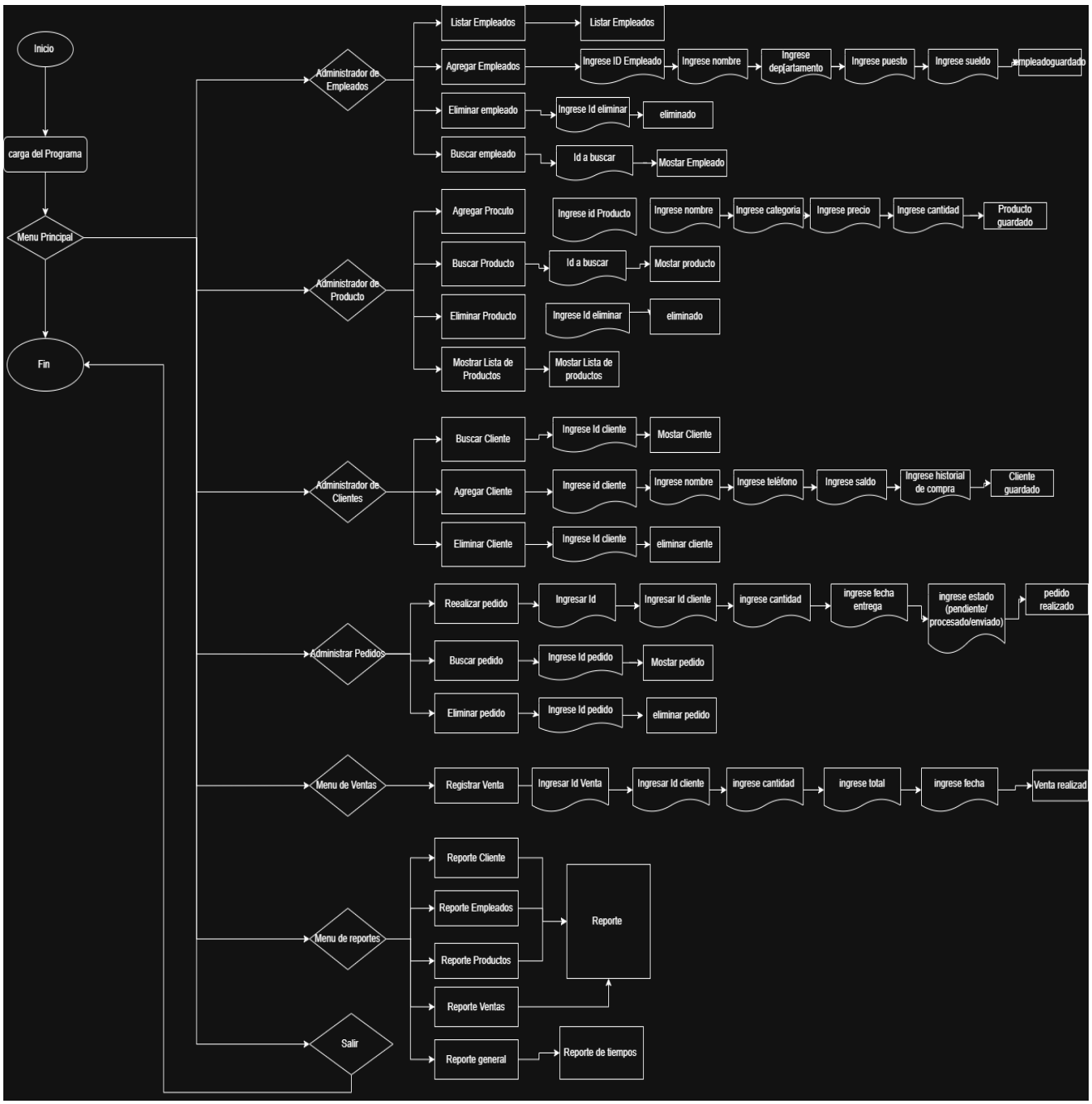
Recuperación: Utilizaremos un algoritmo de checksum, como CRC32 , para calcular un valor único a partir de los datos. Este valor se almacenará junto con los datos para su posterior verificación. Antes de utilizar los datos, se calculará el checksum actual y se comparará con el checksum almacenado. Si los valores no coinciden, significa que los datos están corruptos.

Cálculos de Tiempo

En nuestro proyecto, es esencial calcular los tiempos necesarios para las operaciones críticas, como la compresión, descompresión y análisis de archivos, con el fin de garantizar que el sistema esté lo más optimizado posible. Para lograr esto, utilizaremos la biblioteca de C++ llamada <chrono>, que nos permite medir el tiempo de ejecución de manera precisa.

Diagrama

Diagrama y documentación (Jeffrey Torres)



Conclusión

Cada clase está diseñada para ser modular y fácil de extender, lo que facilita la incorporación de nuevas funcionalidades en el futuro.