Sistema Star Trek - 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 del universo Star Trek.

Bibliotecas Adicionales

- `vector`: Para el manejo de colecciones dinámicas

- `string`: Para el procesamiento de texto

- `sstream`: Para conversiones de tipos y manipulación de cadenas

- `limits`: Para acceder a valores límite de tipos de datos

- `algorithm`: Para operaciones algorítmicas comunes

Funciones de Utilidad

Funciones de Interfaz

`void limpiarPantalla()`

Esta función limpia la pantalla de la consola, adaptándose al sistema operativo en uso mediante directivas de preprocesador (#ifdef \_WIN32). Es crucial para mantener una interfaz limpia y profesional.

`void mostrarTitulo(const string& titulo)`

Una nueva adición que mejora la presentación visual del sistema, creando encabezados consistentes y bien formateados para cada sección. Utiliza caracteres '=' para crear líneas decorativas y centra el título automáticamente.

Funciones de Validación

El sistema implementa un robusto conjunto de funciones de validación:

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

Una implementación genérica que:

- Acepta cualquier tipo numérico

- Valida rangos mínimos y máximos

- Maneja errores de entrada

- Proporciona retroalimentación inmediata al usuario

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

Asegura que las entradas de texto:

- Cumplan con longitudes mínimas y máximas

- No estén vacías

- No contengan solo espacios en blanco

Estructura de la Base de Datos

El sistema implementa un modelo de datos complejo que representa el universo Star Trek:

Planetas (Planets)

- Identificación científica única

- Nombre común

- Población total

- Coordenadas galácticas (x, y, z)

- Montañas asociadas

Imperios (Empires)

- Código galáctico único

- Nombre

- Temperatura promedio

- Relación con planetas y flotas

Flotas (Fleets)

- Código galáctico único

- Destino

- Imperio al que pertenece

- Misiones asignadas

Razas (Races)

- Nombre científico único

- Habilidades especiales

- Distribución poblacional en planetas

Sistema de Naves y Capitanes

- Información detallada de naves

- Asignación de capitanes

- Maniobras disponibles

- Capacidades y energía

Funciones Principales del Sistema

Registro de Entidades

`void registrarPlaneta(connection &c)`

Gestiona el registro completo de nuevos planetas, incluyendo:

- Validación de coordenadas galácticas

- Control de población

- Nombres científicos únicos

`void registrarImperio(connection &c)`

Maneja la creación de nuevos imperios espaciales:

- Validación de códigos galácticos

- Control de temperaturas promedio

- Verificación de unicidad

`void registrarFlota(connection &c)`

Implementa el registro de flotas espaciales:

- Verificación de imperio existente

- Registro de múltiples misiones

- Manejo de transacciones anidadas

Consultas del Sistema

Todas las funciones de consulta implementan:

- Joins optimizados

- Agrupaciones efectivas

- Formatos de salida claros y organizados

Por ejemplo, `consultarFlotas()` muestra:

- Código galáctico

- Imperio propietario

- Misiones asignadas

- Cantidad de naves

Consideraciones de Diseño

El sistema implementa varias características de diseño moderno:

Manejo de Transacciones

Utiliza bloques de ámbito para garantizar el cierre apropiado de transacciones:

{

nontransaction N(c);

// Operaciones de consulta

} // Cierre automático de la transacción

Validación de Datos

Implementa múltiples capas de validación:

- A nivel de aplicación (funciones de validación)

- A nivel de base de datos (constraints)

- Durante la entrada de usuario

Interfaz de Usuario

Presenta una interfaz jerárquica con:

- Menú principal claro

- Submenús específicos

- Retroalimentación constante

- Manejo de errores amigable

Normalización de la Base de Datos

Primera Forma Normal (1NF)

El diseño cumple con 1NF porque:

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

- No hay grupos repetidos

- Todos los atributos son atómicos

Por ejemplo, en lugar de almacenar múltiples misiones en un solo campo de la tabla Fleets, se creó una tabla separada Fleet\_Missions para almacenar cada misión individualmente.

Segunda Forma Normal (2NF)

El diseño cumple con 2NF porque:

- Cumple con 1NF

- Todos los atributos no clave dependen completamente de la clave primaria

Por ejemplo, en la tabla Ships:

- ship\_id es la clave primaria

- fleet\_code, max\_speed, y energy dependen completamente de ship\_id

- Las relaciones con flotas y capitanes se manejan mediante claves foráneas

Tercera Forma Normal (3NF)

El diseño cumple con 3NF porque:

- Cumple con 2NF

- No hay dependencias transitivas

Por ejemplo:

- Las razas y sus habilidades están separadas (Races y Race\_Skills)

- Los planetas y sus montañas están separados (Planets y Mountains)

- Los imperios y sus flotas están separados (Empires y Fleets)

Análisis de Consultas SQL

Consulta de Planetas

SELECT p.scientific\_name, p.common\_name, p.population,

e.name as empire\_name,

string\_agg(DISTINCT m.name || ' (' || m.height || 'm)', ', ') as mountains

FROM Planets p

LEFT JOIN Planet\_Empires pe ON p.planet\_id = pe.planet\_id

LEFT JOIN Empires e ON pe.empire\_id = e.empire\_id

LEFT JOIN Mountains m ON p.planet\_id = m.planet\_id

GROUP BY p.planet\_id, p.scientific\_name, p.common\_name, p.population, e.name

ORDER BY p.scientific\_name;

Esta consulta:

- Usa LEFT JOIN para incluir planetas sin imperio o montañas

- Agrupa las montañas de cada planeta usando string\_agg

- Formatea la altura de las montañas dentro de la misma consulta

- Ordena los resultados por nombre científico para una presentación consistente

Consulta de Flotas

SELECT f.fleet\_id, f.galactic\_code, f.destination,

e.name as empire\_name,

string\_agg(DISTINCT fm.mission\_type, ', ') as missions,

COUNT(DISTINCT s.ship\_id) as ship\_count

FROM Fleets f

JOIN Empires e ON f.empire\_id = e.empire\_id

LEFT JOIN Fleet\_Missions fm ON f.fleet\_id = fm.fleet\_id

LEFT JOIN Ships s ON f.fleet\_id = s.fleet\_id

GROUP BY f.fleet\_id, f.galactic\_code, f.destination, e.name

ORDER BY e.name, f.galactic\_code;

Esta consulta:

- Usa JOIN con Empires porque toda flota debe tener un imperio

- Usa LEFT JOIN con misiones y naves porque podrían no tener

- Cuenta las naves asignadas a cada flota

- Concatena las misiones en una sola columna

- Ordena por imperio y código galáctico para una vista jerárquica

Consulta de Imperios

SELECT e.empire\_id, e.galactic\_code, e.name, e.avg\_temperature,

COUNT(DISTINCT p.planet\_id) as planet\_count,

COUNT(DISTINCT f.fleet\_id) as fleet\_count

FROM Empires e

LEFT JOIN Planet\_Empires pe ON e.empire\_id = pe.empire\_id

LEFT JOIN Planets p ON pe.planet\_id = p.planet\_id

LEFT JOIN Fleets f ON e.empire\_id = f.empire\_id

GROUP BY e.empire\_id, e.galactic\_code, e.name, e.avg\_temperature

ORDER BY e.name;

Esta consulta:

- Calcula estadísticas agregadas para cada imperio

- Cuenta planetas y flotas usando DISTINCT para evitar duplicados

- Usa LEFT JOIN para incluir imperios sin planetas o flotas

- Mantiene la integridad de los datos durante la agregación

Inserción de Datos

Las inserciones utilizan consultas parametrizadas para prevenir SQL injection:

-- Ejemplo de inserción de planeta

INSERT INTO Planets

(scientific\_name, common\_name, population, galactic\_x, galactic\_y, galactic\_z)

VALUES ($1, $2, $3, $4, $5, $6)

Características de seguridad:

- Uso de exec\_params en lugar de exec directo

- Validación de datos antes de la inserción

- Manejo de transacciones para mantener la integridad

Gestión de Relaciones Muchos a Muchos

Para relaciones complejas, como las misiones de las flotas:

INSERT INTO Fleet\_Missions (fleet\_id, mission\_type)

SELECT currval('fleets\_fleet\_id\_seq'), $1

Esta aproximación:

- Utiliza currval para obtener el ID recién creado

- Mantiene la atomicidad de la transacción

- Asegura la integridad referencial

El diseño general de las consultas prioriza:

1. Integridad de datos

2. Rendimiento

3. Facilidad de mantenimiento

4. Claridad en los resultados

5. Seguridad contra inyección SQL

Cada consulta está diseñada para manejar casos nulos y proporcionar resultados significativos incluso cuando faltan datos relacionados, lo que es crucial en un sistema que modela un universo complejo como Star Trek.