Sistema Gestor del Mundial Brasil 2014 - Documentación Técnica

Bibliotecas Utilizadas

1. iostream

- Proporciona funcionalidades de entrada/salida estándar

- Permite interacción con usuario mediante consola

- Utilizado para mostrar información (`cout`) y recibir datos (`cin`)

2. pqxx/pqxx

- Biblioteca oficial de C++ para PostgreSQL

- Interfaz orientada a objetos para interacción con bases de datos

- Permite conexiones, ejecución de consultas y manejo de transacciones

3. cstdlib

- Funciones generales de utilidad

- Utilizado para comandos de sistema (`system()`)

4. regex

- Manejo de expresiones regulares

- Usado para validaciones de entrada de datos

5. vector

- Contenedor de la biblioteca estándar

- Utilizado para almacenar colecciones dinámicas

Funciones de Utilidad

1. Funciones de Interfaz

void limpiarTerminal() {

system("clear");

}

- Limpia la pantalla de la consola

- Mantiene una interfaz limpia y organizada

void limpiarEntrada() {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

}

- Limpia el buffer de entrada

- Evita problemas en lecturas secuenciales de datos

void pausarSistema() {

cout << "\nPresione Enter para continuar...";

cin.get();

}

- Pausa la ejecución del programa

- Permite al usuario leer mensajes antes de continuar

2. Funciones de Validación

bool validarDocumentoIdentidad(const string& documento)

- Valida documentos de identidad

- Verifica longitud (6-15 caracteres)

- Asegura solo caracteres alfanuméricos

bool validarNombre(const string& nombre)

- Valida nombres y textos

- Permite letras con acentos

- Verifica longitud entre 2-100 caracteres

bool validarFechaPartido(const string& fecha\_str)

- Valida fechas de partidos del Mundial

- Verifica formato YYYY-MM-DD

- Restringe fechas al rango del Mundial (junio-julio 2014)

bool validarDomicilio(const string& domicilio)

- Valida direcciones y domicilios

- Permite letras, números y caracteres de dirección

- Verifica longitud entre 5-300 caracteres

3. Funciones de Base de Datos

connection iniciarConexionBaseDatos()

- Establece conexión con base de datos PostgreSQL

- Configura parámetros de conexión

- Maneja errores de conexión

void configurarEstructuraBD(connection &c)

- Crea estructura de base de datos

- Tablas:

1. Equipos

2. Hinchas

3. Hoteles

4. Hospedaje

5. Packs de Cotillón

6. Estadios

7. Partidos

8. Árbitros

9. Fun Fests

10. Consumo de Cerveza

Funciones de Registro y Consulta

1. Registro de Entidades

void registrarHincha(connection &c)

- Registra nuevos hinchas

- Valida información personal

- Permite registro con invitación

void registrarHotel(connection &c)

- Registra hoteles del Mundial

- Valida información del hotel

- Verifica unicidad de ID

void registrarEquipo(connection &c)

- Registra equipos participantes

- Valida información del equipo

- Verifica mundiales ganados

void registrarPartido(connection &c)

- Registra partidos del Mundial

- Valida información del partido

- Gestiona equipos, estadios y árbitros

- Controla cantidad de partidos por equipo

2. Consultas

void consultarHinchas(connection &c)

void consultarPartidos(connection &c)

void consultarEquiposYPartidos(connection &c)

void consultarConsumoCerveza(connection &c)

void consultarFunFests(connection &c)

void consultarHotelesHospedaje(connection &c)

- Muestran información detallada

- Utilizan consultas SQL complejas

- Presentan información organizada

Menú Principal

void menuPrincipal(connection &c)

- Interfaz principal del sistema

- Opciones de registro y consulta

- Validación robusta de entrada

- Manejo de errores

Función Principal

int main()

- Inicialización del sistema

- Establece conexión de base de datos

- Configura estructura de tablas

- Inicia menú principal

- Manejo de errores globales

Consideraciones de Diseño

1. Modularidad

- Funciones específicas para cada tarea

- Código reutilizable

- Separación clara de responsabilidades

2. Validación de Datos

- Validaciones exhaustivas

- Protección contra entradas incorrectas

- Feedback al usuario

3. Manejo de Errores

- Uso consistente de try-catch

- Mensajes descriptivos

- Recuperación graciosa ante fallos

4. Seguridad de Datos

- Transacciones en operaciones críticas

- Validación de entrada

- Protección de integridad de datos

5. Experiencia de Usuario

- Interfaz intuitiva

- Mensajes claros

- Confirmaciones en operaciones importantes

Realizaré un análisis detallado de las consultas SQL y validaciones utilizadas en cada función del sistema:

1. Consulta de Hinchas (`consultarHinchas`)

SELECT h1.nombre, h1.dni, h1.nacionalidad,

h2.nombre AS invitado\_por

FROM Hinchas h1

LEFT JOIN Hinchas h2 ON h1.invitado\_por = h2.hincha\_id;

Características de la consulta:

- Utiliza LEFT JOIN para incluir hinchas sin invitador

- Combina la tabla Hinchas consigo misma

- Recupera información personal y de invitación

- Maneja casos de hinchas sin invitador mediante LEFT JOIN

Validaciones en el registro de hinchas:

bool validarNombre(const string& nombre) {

regex patron("^[A-Za-zÁÉÍÓÚáéíóúÑñ ]+$");

return regex\_match(nombre, patron) &&

nombre.length() >= 2 &&

nombre.length() <= 100;

}

bool validarDocumentoIdentidad(const string& documento) {

return (documento.length() >= 6 && documento.length() <= 15) &&

all\_of(documento.begin(), documento.end(), [](char c) {

return isalnum(c);

});

}

- Valida nombres con letras y acentos

- Verifica longitud del documento (6-15 caracteres)

- Asegura solo caracteres alfanuméricos

2. Consulta de Partidos (`consultarPartidos`)

SELECT p.partido\_id,

e1.pais AS equipo1,

e2.pais AS equipo2,

p.fecha,

est.nombre AS estadio,

p.espectadores,

array\_agg(DISTINCT a.nombre) AS arbitros

FROM Partidos p

JOIN Equipos e1 ON p.equipo1\_id = e1.equipo\_id

JOIN Equipos e2 ON p.equipo2\_id = e2.equipo\_id

JOIN Estadios est ON p.estadio\_id = est.estadio\_id

LEFT JOIN ArbitrosPartidos ap ON p.partido\_id = ap.partido\_id

LEFT JOIN Arbitros a ON ap.arbitro\_id = a.arbitro\_id

GROUP BY p.partido\_id, e1.pais, e2.pais, p.fecha, est.nombre;

Validaciones en el registro de partidos:

bool validarFechaPartido(const string& fecha\_str) {

regex patron("^(2014)-(0[1-7])-(0[1-9]|[12][0-9]|3[01])$");

if (!regex\_match(fecha\_str, patron)) return false;

int anio, mes, dia;

sscanf(fecha\_str.c\_str(), "%d-%d-%d", &anio, &mes, &dia);

return (anio == 2014 && mes >= 6 && mes <= 7 && dia >= 1 && dia <= 31);

}

bool validarCantidadPartidosPorEquipo(connection &c, int equipo\_id) {

nontransaction N(c);

result R = N.exec\_params(

"SELECT mundiales\_ganados, "

"(SELECT COUNT(\*) FROM Partidos WHERE equipo1\_id = $1 OR equipo2\_id = $1) AS partidos\_jugados "

"FROM Equipos WHERE equipo\_id = $1",

equipo\_id

);

int mundiales = R[0][0].as<int>();

int partidos = R[0][1].as<int>();

int max\_partidos;

switch(mundiales) {

case 0: max\_partidos = 3; break;

case 1: max\_partidos = 4; break;

case 2: max\_partidos = 5; break;

case 3: max\_partidos = 6; break;

default: max\_partidos = 7; break;

}

return partidos < max\_partidos;

}

Características de las validaciones:

- Restringe fechas al Mundial 2014

- Limita número de partidos según historial del equipo

- Valida formato de fecha

- Controla cantidad de partidos por equipo

3. Consulta de Equipos y Partidos (`consultarEquiposYPartidos`)

SELECT e.pais, e.mundiales\_ganados,

COUNT(p.partido\_id) AS partidos\_jugados,

SUM(CASE WHEN p.equipo1\_id = e.equipo\_id OR p.equipo2\_id = e.equipo\_id THEN 1 ELSE 0 END) AS total\_partidos

FROM Equipos e

LEFT JOIN Partidos p ON (p.equipo1\_id = e.equipo\_id OR p.equipo2\_id = e.equipo\_id)

GROUP BY e.pais, e.mundiales\_ganados

ORDER BY total\_partidos DESC;

Validaciones en registro de equipos:

bool validarNombre(const string& nombre) {

regex patron("^[A-Za-zÁÉÍÓÚáéíóúÑñ ]+$");

return regex\_match(nombre, patron) &&

nombre.length() >= 2 &&

nombre.length() <= 100;

}

bool validarMundiales(int mundiales) {

return mundiales >= 0 && mundiales <= 5;

}

Características de las validaciones:

- Valida nombres de países

- Limita número de mundiales ganados (0-5)

4. Consulta de Consumo de Cerveza (`consultarConsumoCerveza`)

SELECT p.partido\_id,

e1.pais AS equipo1,

e2.pais AS equipo2,

SUM(cc.cantidad\_cervezas) AS total\_cervezas,

COUNT(DISTINCT h.hincha\_id) AS total\_hinchas

FROM Partidos p

JOIN Equipos e1 ON p.equipo1\_id = e1.equipo\_id

JOIN Equipos e2 ON p.equipo2\_id = e2.equipo\_id

LEFT JOIN ConsumoCerveza cc ON p.partido\_id = cc.partido\_id

LEFT JOIN Hinchas h ON cc.hincha\_id = h.hincha\_id

GROUP BY p.partido\_id, e1.pais, e2.pais

ORDER BY total\_cervezas DESC;

Validaciones en registro de consumo de cerveza:

void registrarConsumoCerveza(connection &c) {

// Validaciones de entrada

if (cantidad\_cervezas < 0 || cantidad\_cervezas > 50) {

cout << "Cantidad de cervezas inválida (0-50)." << endl;

return;

}

// Verificación de hincha y partido existentes

result R\_hincha = W.exec\_params("SELECT hincha\_id FROM Hinchas WHERE dni = $1", dni\_hincha);

result R\_partido = W.exec\_params("SELECT 1 FROM Partidos WHERE partido\_id = $1", partido\_id);

if (R\_hincha.empty() || R\_partido.empty()) {

cout << "Hincha o partido no encontrado." << endl;

return;

}

}

Características de las validaciones:

- Limita cantidad de cervezas

- Verifica existencia de hincha y partido

- Previene registros inválidos

5. Consulta de Fun Fests (`consultarFunFests`)

SELECT ff.funfest\_id, ff.ciudad, ff.duracion,

e1.pais AS equipo1, e2.pais AS equipo2

FROM FunFests ff

JOIN Partidos p ON ff.partido\_id = p.partido\_id

JOIN Equipos e1 ON p.equipo1\_id = e1.equipo\_id

JOIN Equipos e2 ON p.equipo2\_id = e2.equipo\_id;

Validaciones en registro de Fun Fests:

void registrarFunFest(connection &c) {

// Validación de ciudad

do {

cout << "Ciudad: ";

getline(cin, ciudad);

if (!validarNombre(ciudad)) {

cout << "Ciudad inválida. Use solo letras." << endl;

}

} while (!validarNombre(ciudad));

// Validación de duración

regex duracion\_patron("^\\d{2}:\\d{2}:\\d{2}$");

if (!regex\_match(duracion\_str, duracion\_patron)) {

cout << "Formato de duración inválido. Use HH:MM:SS" << endl;

return;

}

}

Características de las validaciones:

- Valida nombre de ciudad

- Verifica formato de duración

- Asegura coherencia de datos

6. Consulta de Hoteles y Hospedaje (`consultarHotelesHospedaje`)

SELECT h.hotel\_id, h.nombre, h.direccion, h.estrellas,

COUNT(DISTINCT ho.hincha\_id) AS hinchas\_hospedados

FROM Hoteles h

LEFT JOIN Hospedaje ho ON h.hotel\_id = ho.hotel\_id

GROUP BY h.hotel\_id, h.nombre, h.direccion, h.estrellas

ORDER BY hinchas\_hospedados DESC;

Validaciones en registro de hoteles:

bool validarDomicilio(const string& domicilio) {

regex patron("^[A-Za-zÁÉÍÓÚáéíóúÑñ0-9 ,.-]+$");

return regex\_match(domicilio, patron) &&

domicilio.length() >= 5 &&

domicilio.length() <= 300;

}

void registrarHotel(connection &c) {

// Validación de estrellas

do {

cout << "Número de estrellas (1-5): ";

cin >> estrellas;

limpiarEntrada();

} while (estrellas < 1 || estrellas > 5);

}

Características de las validaciones:

- Valida direcciones de hoteles

- Limita número de estrellas (1-5)

- Verifica unicidad de ID de hotel

7. Consulta de Packs de Cotillón (`generarInformePacks`)

SELECT h.nombre AS hincha, h.nacionalidad,

COUNT(DISTINCT ph.pack\_id) AS packs\_obtenidos,

array\_agg(DISTINCT pc.nombre) AS nombres\_packs

FROM Hinchas h

LEFT JOIN PacksHinchas ph ON h.hincha\_id = ph.hincha\_id

LEFT JOIN PacksCotillon pc ON ph.pack\_id = pc.pack\_id

GROUP BY h.hincha\_id, h.nombre, h.nacionalidad

ORDER BY packs\_obtenidos DESC;

Validaciones en registro de packs:

bool verificarLimitePacks(connection &c, int hincha\_id) {

nontransaction N(c);

string query = "SELECT COUNT(DISTINCT pack\_id) FROM PacksHinchas WHERE hincha\_id = $1";

result R = N.exec\_params(query, hincha\_id);

int total\_packs = R[0][0].as<int>();

return total\_packs < 3; // Límite de 3 packs por hincha

}

Características de las validaciones:

- Limita packs por hincha a 3

- Previene duplicación de packs con mismo nombre

Consideraciones Generales de Validación:

1. Uso extensivo de expresiones regulares

2. Validaciones de longitud

3. Restricciones de rango

4. Verificación de existencia de registros

5. Prevención de datos inconsistentes

6. Feedback al usuario sobre errores

Técnicas de Validación:

- Regex para formatos complejos

- Funciones de validación reutilizables

- Validación en tiempo de entrada

- Validación en base de datos

- Control de integridad referencial