BASES DE DATOS:

Índice

Introducción a las bases de datos en C++
Configuración del entorno
Vinculación con una base de datos
Cargar datos
Modificar datos
Eliminar datos
Ejemplo completo
Consideraciones adicionales

Introducción a las Bases de Datos en C++

En C++, puedes interactuar con bases de datos utilizando bibliotecas y controladores que facilitan la conexión y la manipulación de datos. Entre las opciones más comunes se encuentran:

ODBC (Open Database Connectivity): Una interfaz estándar para acceder a bases de datos.
MySQL Connector/C++: Un conector específico para bases de datos MySQL.
SQLite: Una biblioteca que proporciona una base de datos SQL ligera y sin servidor.

Configuración del Entorno

Para trabajar con bases de datos en C++, primero necesitas instalar y configurar el entorno adecuado. Aquí te muestro cómo hacerlo con dos opciones populares: SQLite y MySQL Connector/C++.

SQLite

Descargar SQLite: Ve al sitio web oficial de SQLite **(https://www.sqlite.org/download.html)** y descarga la biblioteca precompilada para tu sistema operativo.

Incluir SQLite en tu proyecto:

Añade el archivo de cabecera **sqlite3.h** y el archivo de implementación **sqlite3.c** a tu proyecto.
Compila el archivo sqlite3.c con tu proyecto para vincularlo.

MySQL Connector/C++

Descargar MySQL Connector/C++: Visita el sitio oficial de **MySQL (https://dev.mysql.com/downloads/connector/cpp/)** y descarga el conector adecuado.

Instalar el conector:

Sigue las instrucciones de instalación para tu sistema operativo.
Asegúrate de tener las bibliotecas y los archivos de cabecera necesarios en tu proyecto.

Vinculación con una Base de Datos

SQLite

Aquí tienes un ejemplo de cómo conectar tu programa C++ a una base de datos SQLite:

```
#include <iostream>
#include <sqlite3.h>

// Callback para imprimir los resultados de la consulta

static int callback(void* data, int argc, char** argv, char** azColName) {
        for (int i = 0; i < argc; i++) {
                std::cout << azColName[i] << " = " << (argv[i] ? argv[i] : "NULL") << std::endl;
        }
        std::cout << std::endl;
        return 0;
}

int main() {
        sqlite3* db;
        char* errorMessage = 0;
```

```
        // Abrir una base de datos (o crearla si no existe)
        if (sqlite3_open("test.db", &db)) {
                  std::cerr << "No se puede abrir la base de datos: " << sqlite3_errmsg(db) <<
std::endl;
                return 1;
        }

        // Ejecutar una consulta SQL para crear una tabla

const char* sqlCreateTable = "CREATE TABLE IF NOT EXISTS persona (id INTEGER PRIMARY KEY, nombre
TEXT, edad INTEGER);";
        if (sqlite3_exec(db, sqlCreateTable, 0, 0, &errorMessage) != SQLITE_OK) {
                std::cerr << "Error al crear la tabla: " << errorMessage << std::endl;
                sqlite3_free(errorMessage);
                return 1;
        }

        // Cerrar la base de datos

sqlite3_close(db);
        return 0;
}
```

MySQL Connector/C++

Aquí tienes un ejemplo de cómo conectar tu programa C++ a una base de datos MySQL:

```
#include <mysql_driver.h>
#include <mysql_connection.h>
#include <iostream>

int main() {
        sql::mysql::MySQL_Driver* driver;
        sql::Connection* con;

        // Obtener el controlador de MySQL

 driver = sql::mysql::get_mysql_driver_instance();

        // Conectar a la base de datos

con = driver->connect("tcp://127.0.0.1:3306", "user", "password");

        // Crear una base de datos y usarla

con->setSchema("test");

        // Ejecutar una consulta SQL para crear una tabla

sql::Statement* stmt = con->createStatement();
stmt->execute("CREATE TABLE IF NOT EXISTS persona (id INT AUTO_INCREMENT PRIMARY KEY, nombre
VARCHAR(255), edad INT);");

        // Limpiar
        delete stmt;
        delete con;
        return 0;
}
```

Cargar Datos

SQLite

Para insertar datos en una tabla en SQLite, puedes usar una consulta SQL INSERT:

```
const char* sqlInsert = "INSERT INTO persona (nombre, edad) VALUES ('Juan Pérez', 30);";
if (sqlite3_exec(db, sqlInsert, 0, 0, &errorMessage) != SQLITE_OK) {
        std::cerr << "Error al insertar datos: " << errorMessage << std::endl;
        sqlite3_free(errorMessage); }
```

MySQL Connector/C++

Para insertar datos en una tabla en MySQL:

```
sql::Statement* stmt = con->createStatement();
stmt->execute("INSERT INTO persona (nombre, edad) VALUES ('Ana Gómez', 25);");
delete stmt;
```

Modificar Datos

SQLite

Para actualizar datos, usa una consulta SQL UPDATE:

```
const char* sqlUpdate = "UPDATE persona SET edad = 31 WHERE nombre = 'Juan Pérez';";
if (sqlite3_exec(db, sqlUpdate, 0, 0, &errorMessage) != SQLITE_OK) {
        std::cerr << "Error al actualizar datos: " << errorMessage << std::endl;
        sqlite3_free(errorMessage);
}
```

MySQL Connector/C++

Para actualizar datos:

```
sql::Statement* stmt = con->createStatement();
stmt->execute("UPDATE persona SET edad = 26 WHERE nombre = 'Ana Gómez';");
delete stmt;
```

Eliminar Datos

SQLite

Para eliminar datos, usa una consulta SQL DELETE:

```
const char* sqlDelete = "DELETE FROM persona WHERE nombre = 'Juan Pérez';";
if (sqlite3_exec(db, sqlDelete, 0, 0, &errorMessage) != SQLITE_OK) {
        std::cerr << "Error al eliminar datos: " << errorMessage << std::endl;
        sqlite3_free(errorMessage);
}
```

MySQL Connector/C++

Para eliminar datos:

```
sql::Statement* stmt = con->createStatement();
stmt->execute("DELETE FROM persona WHERE nombre = 'Ana Gómez';");
delete stmt;
```

Ejemplo Completo SQLite

```
#include <iostream>
#include <sqlite3.h>

static int callback(void* data, int argc, char** argv, char** azColName) {
        for (int i = 0; i < argc; i++) {
                std::cout << azColName[i] << " = " << (argv[i] ? argv[i] : "NULL") << std::endl;
        }
        std::cout << std::endl;
        return 0;
}

int main() {
        sqlite3* db;
        char* errorMessage = 0;

        if (sqlite3_open("test.db", &db)) {
                std::cerr << "No se puede abrir la base de datos: " << sqlite3_errmsg(db) <<
std::endl;
                return 1;
        }
```

```
        const char* sqlCreateTable = "CREATE TABLE IF NOT EXISTS persona (id INTEGER PRIMARY KEY,
nombre TEXT, edad INTEGER);";
        if (sqlite3_exec(db, sqlCreateTable, 0, 0, &errorMessage) != SQLITE_OK) {
                std::cerr << "Error al crear la tabla: " << errorMessage << std::endl;
                sqlite3_free(errorMessage);
                return 1;
        }

        const char* sqlInsert = "INSERT INTO persona (nombre, edad) VALUES ('Juan Pérez', 30);";
        if (sqlite3_exec(db, sqlInsert, 0, 0, &errorMessage) != SQLITE_OK) {
                std::cerr << "Error al insertar datos: " << errorMessage << std::endl;
                sqlite3_free(errorMessage);
        }

        const char* sqlSelect = "SELECT * FROM persona;";
        if (sqlite3_exec(db, sqlSelect, callback, 0, &errorMessage) != SQLITE_OK) {
                std::cerr << "Error al seleccionar datos: " << errorMessage << std::endl;
                sqlite3_free(errorMessage);
        }

        sqlite3_close(db);
        return 0;
}
```

MySQL Connector/C++

```
#include <mysql_driver.h>
#include <mysql_connection.h>
#include <iostream>

int main() {
        sql::mysql::MySQL_Driver* driver;
        sql::Connection* con;

        driver = sql::mysql::get_mysql_driver_instance();
        con = driver->connect("tcp://127.0.0.1:3306", "user", "password");
        con->setSchema("test");

        sql::Statement* stmt = con->createStatement();
        stmt->execute("CREATE TABLE IF NOT EXISTS persona (id INT AUTO_INCREMENT PRIMARY KEY, nombre
VARCHAR(255), edad INT);");
        stmt->execute("INSERT INTO persona (nombre, edad) VALUES ('Ana Gómez', 25);");

        sql::ResultSet* res = stmt->executeQuery("SELECT * FROM persona;");
        while (res->next()) {
                std::cout << "ID: " << res->getInt("id") << ", Nombre: " << res->getString("nombre")
<< ", Edad: " << res->getInt("edad") << std::endl;
        }

        delete res;
        delete stmt;
        delete con;
        return 0;
}
```

Consideraciones Adicionales

Manejo de Errores: Asegúrate de manejar errores adecuadamente, tanto en la conexión a la base de datos como en la ejecución de consultas.

Seguridad: Evita inyecciones SQL utilizando consultas preparadas si estás trabajando con datos proporcionados por el usuario.

Configuración: Asegúrate de que el servidor de la base de datos esté en funcionamiento y configurado correctamente.

en Linux.

Índice

Introducción a las Bases de Datos en C++
Configuración del Entorno en Linux
Vinculación con una Base de Datos
Cargar Datos
Modificar Datos
Eliminar Datos
Ejemplo Completo en Linux
Consideraciones Adicionales

Introducción a las Bases de Datos en C++

En C++, puedes interactuar con bases de datos utilizando bibliotecas que facilitan la conexión y manipulación de datos. *Dos opciones comunes en Linux son SQLite y MySQL*.

SQLite

SQLite es una base de datos ligera que se almacena en un solo archivo. Es ideal para aplicaciones locales y de pequeña escala.

MySQL

MySQL es un sistema de gestión de bases de datos relacional más robusto, adecuado para aplicaciones más grandes y entornos de producción.

Configuración del Entorno en Linux

SQLite

Instalar SQLite: En la mayoría de las distribuciones de Linux, puedes instalar SQLite utilizando el gestor de paquetes. Abre una terminal y ejecuta:

bash

```
sudo apt-get install sqlite3 libsqlite3-dev
```

Incluir SQLite en tu proyecto: Cuando compiles tu programa en C++, asegúrate de incluir las bibliotecas de SQLite. Puedes hacerlo con el siguiente comando:

bash

```
g++ -o mi_programa mi_programa.cpp -lsqlite3
```

MySQL

Instalar MySQL y el conector C++: Puedes instalar MySQL y el conector utilizando el gestor de paquetes:

bash

```
sudo apt-get install mysql-server libmysqlcppconn-dev
```

Incluir MySQL Connector/C++ en tu proyecto: Asegúrate de incluir las bibliotecas de MySQL en tu proyecto. Compila tu programa con el siguiente comando:

bash

```
g++ -o mi_programa mi_programa.cpp -lmysqlcppconn
```

Vinculación con una Base de Datos

SQLite

Para conectar con una base de datos SQLite en C++:

```
#include <iostream>
#include <sqlite3.h>

// Callback para imprimir los resultados de la consulta

static int callback(void* data, int argc, char** argv, char** azColName) {
        for (int i = 0; i < argc; i++) {
                std::cout << azColName[i] << " = " << (argv[i] ? argv[i] : "NULL") << std::endl;
        }
        std::cout << std::endl;
        return 0;
}

int main() {
        sqlite3* db;
        char* errorMessage = 0;

        // Abrir una base de datos (o crearla si no existe)
        if (sqlite3_open("test.db", &db)) {
                std::cerr << "No se puede abrir la base de datos: " << sqlite3_errmsg(db) <<
std::endl;
                return 1;
        }

        // Ejecutar una consulta SQL para crear una tabla

const char* sqlCreateTable = "CREATE TABLE IF NOT EXISTS persona (id INTEGER PRIMARY KEY, nombre
TEXT, edad INTEGER);";
        if (sqlite3_exec(db, sqlCreateTable, 0, 0, &errorMessage) != SQLITE_OK) {
                std::cerr << "Error al crear la tabla: " << errorMessage << std::endl;
                sqlite3_free(errorMessage);
                return 1;
        }

        // Cerrar la base de datos

sqlite3_close(db);
        return 0;
}
```

MySQL

Para conectar con una base de datos MySQL en C++:

```
#include <mysql_driver.h>
#include <mysql_connection.h>
#include <iostream>

int main() {
        sql::mysql::MySQL_Driver* driver;
        sql::Connection* con;

        driver = sql::mysql::get_mysql_driver_instance();
        con = driver->connect("tcp://127.0.0.1:3306", "user", "password");
        con->setSchema("test");

        // Crear una base de datos y usarla
sql::Statement* stmt = con->createStatement();
stmt->execute("CREATE DATABASE IF NOT EXISTS test");
stmt->execute("USE test");
stmt->execute("CREATE TABLE IF NOT EXISTS persona (id INT AUTO_INCREMENT PRIMARY KEY, nombre
VARCHAR(255), edad INT);");

        // Limpiar
 delete stmt;
```

```
        delete con;
        return 0;
}
```

Cargar Datos

SQLite

Para insertar datos en una tabla en SQLite:

```
const char* sqlInsert = "INSERT INTO persona (nombre, edad) VALUES ('Juan Pérez', 30);";
if (sqlite3_exec(db, sqlInsert, 0, 0, &errorMessage) != SQLITE_OK) {
        std::cerr << "Error al insertar datos: " << errorMessage << std::endl;
        sqlite3_free(errorMessage);
}
```

MySQL

Para insertar datos en una tabla en MySQL:

```
sql::Statement* stmt = con->createStatement();
stmt->execute("INSERT INTO persona (nombre, edad) VALUES ('Ana Gómez', 25);");
delete stmt;
```

Modificar Datos

SQLite

Para actualizar datos en SQLite:

```
const char* sqlUpdate = "UPDATE persona SET edad = 31 WHERE nombre = 'Juan Pérez';";
if (sqlite3_exec(db, sqlUpdate, 0, 0, &errorMessage) != SQLITE_OK) {
        std::cerr << "Error al actualizar datos: " << errorMessage << std::endl;
        sqlite3_free(errorMessage);
}
```

MySQL

Para actualizar datos en MySQL:

```
sql::Statement* stmt = con->createStatement();
stmt->execute("UPDATE persona SET edad = 26 WHERE nombre = 'Ana Gómez';");
delete stmt;
```

Eliminar Datos

SQLite

Para eliminar datos en SQLite:

```
const char* sqlDelete = "DELETE FROM persona WHERE nombre = 'Juan Pérez';";
if (sqlite3_exec(db, sqlDelete, 0, 0, &errorMessage) != SQLITE_OK) {
        std::cerr << "Error al eliminar datos: " << errorMessage << std::endl;
        sqlite3_free(errorMessage);
}
```

MySQL

Para eliminar datos en MySQL:

```
sql::Statement* stmt = con->createStatement();
stmt->execute("DELETE FROM persona WHERE nombre = 'Ana Gómez';");
delete stmt;
```

Ejemplo Completo en Linux

SQLite

```
#include <iostream>
#include <sqlite3.h>

static int callback(void* data, int argc, char** argv, char** azColName) {
        for (int i = 0; i < argc; i++) {
                std::cout << azColName[i] << " = " << (argv[i] ? argv[i] : "NULL") << std::endl;
        }
        std::cout << std::endl;
        return 0;
}

int main() {
        sqlite3* db;
        char* errorMessage = 0;

        if (sqlite3_open("test.db", &db)) {
                  std::cerr << "No se puede abrir la base de datos: " << sqlite3_errmsg(db) <<
std::endl;
                return 1;
        }

         const char* sqlCreateTable = "CREATE TABLE IF NOT EXISTS persona (id INTEGER PRIMARY KEY,
nombre TEXT, edad INTEGER);";
        if (sqlite3_exec(db, sqlCreateTable, 0, 0, &errorMessage) != SQLITE_OK) {
                std::cerr << "Error al crear la tabla: " << errorMessage << std::endl;
                sqlite3_free(errorMessage);
                return 1;
        }

        const char* sqlInsert = "INSERT INTO persona (nombre, edad) VALUES ('Juan Pérez', 30);";
        if (sqlite3_exec(db, sqlInsert, 0, 0, &errorMessage) != SQLITE_OK) {
                std::cerr << "Error al insertar datos: " << errorMessage << std::endl;
                sqlite3_free(errorMessage);
        }

        const char* sqlSelect = "SELECT * FROM persona;";
        if (sqlite3_exec(db, sqlSelect, callback, 0, &errorMessage) != SQLITE_OK) {
                std::cerr << "Error al seleccionar datos: " << errorMessage << std::endl;
                sqlite3_free(errorMessage);
        }

        sqlite3_close(db);
        return 0;
}
```

MySQL

```
#include <mysql_driver.h>
#include <mysql_connection.h>
#include <iostream>

int main() {
        sql::mysql::MySQL_Driver* driver;
        sql::Connection* con;

        driver = sql::mysql::get_mysql_driver_instance();
        con = driver->connect("tcp://127.0.0.1:3306", "user", "password");
        con->setSchema("test");

        sql::Statement* stmt = con->createStatement();
        stmt->execute("CREATE TABLE IF NOT EXISTS persona (id INT AUTO_INCREMENT PRIMARY KEY, nombre
VARCHAR(255), edad INT);");
        stmt->execute("INSERT INTO persona (nombre, edad) VALUES ('Ana Gómez', 25);");

        sql::ResultSet* res = stmt->executeQuery("SELECT * FROM persona;");
        while (res->next()) {
```

```
            std::cout << "ID: " << res->getInt("id") << ", Nombre: " << res->getString("nombre")
<< ", Edad: " << res->getInt("edad") << std::endl;
        }

        delete res;
        delete stmt;
        delete con;
        return 0;
}
```

Consideraciones Adicionales

Manejo de Errores: Asegúrate de manejar los errores de conexión y consulta adecuadamente para evitar problemas en la ejecución del programa.

Seguridad: Utiliza consultas preparadas para prevenir inyecciones SQL, especialmente si estás trabajando con entradas del usuario.

Configuración: Verifica que el servidor de la base de datos (si corresponde) esté funcionando y configurado correctamente en tu entorno de Linux.