Crear servicios web Restfull

Antonio Espín Herranz

Contenidos

- Diseño de una API REST desde cero.
- Configuración del servidor y manejo de peticiones GET, POST, PUT, DELETE.
- Gestión de rutas y parámetros en APIs REST.
- Integración con bases de datos (SQLite, PostgreSQL, MySQL).
- Demostración práctica: Creación de un microservicio en C++.

Diseño API Rest desde cero

Operaciones: GET, POST,PUT, DELETE

Gestión de rutas y parámetros

Bases de datos: SQLite3, MySQL y PostGreSQL

Librerías

- Para trabajar con estas bases de datos tenemos librerías en el gestor de paquetes: vcpkg

- Para instalar:
  - Sqlite3 **vcpkg install slite3**
  - MySQL **vcpkg install libmysql**
  - PostgreSQL **vcpkg install libpq**

- Para integrar en Visual Studio:
  - **vcpkg integrate install**

Problemas con libmysql

- Ir a la carpeta de vcpkg
  - git pull
  - vcpkg update
  - vcpkg upgrade
    - Puede dar un warning y habrá que ejecutar con:
    - vcpkg upgrade --no-dry-run

- Después de actualizar:
  - vcpkg remove libmysql
  - vcpkg remove --outdated
  - vcpkg install libmysql
  - vcpkg integrate install

Librería sqlite3

Prepared Statement

- **Ventajas de usar prepared statements:**
  - **Seguridad**: Evita inyecciones SQL.
  - **Rendimiento**: Puedes reutilizar la sentencia preparada para múltiples inserciones.
  - **Flexibilidad**: Puedes vincular diferentes tipos de datos (texto, enteros, blobs, etc.).

Ejemplo de prepared statement

- **Abrir conexión:**

```
sqlite3* db;
sqlite3_stmt* stmt;
int rc;

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

Ejemplo de prepared statement

- SQL y preparar la sentencia:

```
// Sentencia SQL con parámetros
const char* sql = "INSERT INTO usuarios(nombre, edad, activo, fecha_registro) VALUES (?, ?, ?, ?);";

rc = sqlite3_prepare_v2(db, sql, -1, &stmt, nullptr);
if (rc != SQLITE_OK) {
    std::cerr << "Error al preparar la sentencia: " << sqlite3_errmsg(db) << std::endl;
    sqlite3_close(db);
    return rc;
}
```

Ejemplo de prepared statement

- Varios tipos de datos a insertar:

```
// Datos a insertar
  const char* nombre = “Ana";
  int edad = 35;
  bool activo = true; // SQLite no tiene tipo booleano, se usa 0/1
  const char* fecha = "2025-09-03";

  // Vincular parámetros
  sqlite3_bind_text(stmt, 1, nombre, -1, SQLITE_STATIC);      // nombre (TEXT)
  sqlite3_bind_int(stmt, 2, edad);                            // edad (INTEGER)
  sqlite3_bind_int(stmt, 3, activo ? 1 : 0);            // activo (BOOLEAN como INTEGER)
  sqlite3_bind_text(stmt, 4, fecha, -1, SQLITE_STATIC);     // fecha_registro (TEXT)
```

Ejemplo prepared statement

- Ejecutar sentencia y liberar recursos:

```
rc = sqlite3_step(stmt);
if (rc != SQLITE_DONE) {
    std::cerr << "Error al insertar la fila: " << sqlite3_errmsg(db) << std::endl;
} else {
    std::cout << "Fila insertada correctamente." << std::endl;
}

// Liberar recursos
sqlite3_finalize(stmt);
sqlite3_close(db);
```

Ejemplo de prepared statement

- BOOLEAN en SQLite se representa como INTEGER (0 o 1).

- Fechas se suelen guardar como texto en formato ISO (YYYY-MM-DD), aunque también puedes usar INTEGER como timestamp Unix.
- Puedes usar SQLITE_TRANSIENT en lugar de SQLITE_STATIC si los datos no viven más allá del sqlite3_bind_text.
  - ***Ojo con datos en punteros ...***

- Para el tipo REAL de sqlite3: sqlite3_bind_double

Filas afectadas

- Cuando ejecutamos sentencias de insert / delete / update podemos obtener el número de filas afectadas:

- int filasAfectadas = **sqlite3_changes**(conexión)

- Si estamos dentro de una transacción y hemos realizado varias operaciones:
- int filas = **sqlite3_total_changes**(conexion);

Transacciones

- Se envían las instrucciones de SQL:
- Begin transaction
- Commit
- Rollback

- Con la función: sqlite3_exec

Ejemplo 1 de 2

```
// Iniciar transacción
  rc = sqlite3_exec(db, "BEGIN TRANSACTION;", nullptr, nullptr, nullptr);
  if (rc != SQLITE_OK) {
     std::cerr << "Error al iniciar transacción: " << sqlite3_errmsg(db) << std::endl;
     return false;
  }

  // Primera operación
  const char* sql1 = "UPDATE empleados SET activo = 0 WHERE cargo = 'Intern';";
  rc = sqlite3_exec(db, sql1, nullptr, nullptr, nullptr);
  if (rc != SQLITE_OK) {
     std::cerr << "Error en la primera operación: " << sqlite3_errmsg(db) << std::endl;
     sqlite3_exec(db, "ROLLBACK;", nullptr, nullptr, nullptr);
     return false;
  }
```

Ejemplo 2 de 2

```
// Segunda operación
  const char* sql2 = "DELETE FROM empleados WHERE edad > 65;";
  rc = sqlite3_exec(db, sql2, nullptr, nullptr, nullptr);
  if (rc != SQLITE_OK) {
     std::cerr << "Error en la segunda operación: " << sqlite3_errmsg(db) << std::endl;
     sqlite3_exec(db, "ROLLBACK;", nullptr, nullptr, nullptr);
     return false;
  }

  // Confirmar transacción
  rc = sqlite3_exec(db, "COMMIT;", nullptr, nullptr, nullptr);
  if (rc != SQLITE_OK) {
     std::cerr << "Error al confirmar transacción: " << sqlite3_errmsg(db) << std::endl;
     return false;
  }

  std::cout << "Transacción completada con éxito." << std::endl;
```

MySQL

libmysql

Librería libmysql

- Con mysql: diferencias con sqlite3

`sqlite3_prepare_v2`	`mysql_stmt_prepare`
`sqlite3_bind_*`	`mysql_stmt_bind_param` con `MYSQL_BIND`
`sqlite3_step`	`mysql_stmt_execute`
`sqlite3_finalize`	`mysql_stmt_close`
`sqlite3_changes`	`mysql_stmt_affected_rows`
`sqlite3_exec` con callback	`mysql_query` + `mysql_store_result` + `mysql_fetch_row`

Librería libpq