# Rust MSSQL CRUD System

## Complete Setup & Build Guide

*For Developers*

---

| | |
|---|---|
| **Language:** | Rust |
| **Database:** | Microsoft SQL Server (MSSQL) |
| **GUI Framework:** | eframe / egui |
| **Version:** | 1.0.0 |

*February 2026*

# Overview

This guide walks developers through setting up, configuring, and building the Rust MSSQL CRUD System from source code. The system is a desktop application that provides full Create, Read, Update, and Delete (CRUD) operations against a Microsoft SQL Server database, with a modern graphical user interface built using the egui framework.

| Component | Technology | Version |
|---|---|---|
| Language | Rust | Latest Stable |
| GUI Framework | eframe / egui | 0.29 |
| Database Driver | Tiberius | 0.12 |
| Async Runtime | Tokio | 1.x |
| Database | Microsoft SQL Server | 2019 / 2022 |

# Project Structure

The project follows a clean modular architecture. Each file has a single responsibility:

```
rust_mssql_crud/
├── Cargo.toml          # Dependencies and project configuration
├── src/
│   ├── main.rs         # Application entry point
│   ├── app.rs          # GUI logic - all screens and forms
│   ├── database.rs     # CRUD operations for Users and Products
│   ├── db_config.rs    # Database connection configuration
│   └── models.rs       # Data structures (User, Product structs)
├── setup_database.sql  # SQL script to create database and tables
└── README.md           # Documentation
```

# Prerequisites

Before building the project, ensure the following software is installed on the developer machine:

## 1. Install Rust

Rust is required to compile the project. Install it using the official rustup installer.

> Windows: Download and run the installer from https://rustup.rs/

Linux / macOS: Run the following command in terminal:

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
source $HOME/.cargo/env
```

Verify the installation:

```
rustc --version
cargo --version
```

### 2. Install Microsoft SQL Server

The application requires a running SQL Server instance. SQL Server Express (free) is sufficient for development.

- Download SQL Server 2019 or 2022 Express from the Microsoft website
- Run the installer and choose 'Basic' installation type
- Note the instance name shown at the end of installation (e.g., SQLEXPRESS)
- Optionally install SQL Server Management Studio (SSMS) for database management

### 3. Install Linux Dependencies (Linux Only)

If building on Linux, install the required system libraries:

```
sudo apt-get update
sudo apt-get install -y libxcb-render0-dev libxcb-shape0-dev \
    libxcb-xfixes0-dev libxkbcommon-dev libssl-dev
```

## Database Setup

The application connects to an existing SQL Server database named SampleDB. Follow these steps to configure the database correctly.

### Step 1: Enable SQL Server Authentication (Mixed Mode)

1. Open SQL Server Management Studio (SSMS)
2. Connect using Windows Authentication
3. Right-click the server name in Object Explorer
4. Select Properties → Security
5. Set Server Authentication to 'SQL Server and Windows Authentication mode'
6. Click OK and restart the SQL Server service

## Step 2: Enable TCP/IP Protocol

7. Open SQL Server Configuration Manager
8. Expand SQL Server Network Configuration
9. Click Protocols for MSSQLSERVER (or SQLEXPRESS)
10. Right-click TCP/IP and select Enable
11. Right-click TCP/IP and select Properties
12. Click the IP Addresses tab and scroll to IPAll at the bottom
13. Set TCP Dynamic Ports to empty and TCP Port to 1433
14. Click OK and restart the SQL Server service

## Step 3: Create the SA Login Password

Connect to SQL Server using Windows Authentication, then run:

```
-- Connect first:
sqlcmd -S localhost -E -C

-- Then run these SQL commands:
ALTER LOGIN sa ENABLE;
GO
ALTER LOGIN sa WITH PASSWORD = 'YourPassword', CHECK_POLICY = OFF;
GO
quit
```

## Step 4: Create the Database and Tables

Run the provided setup_database.sql script, or execute the following SQL manually:

```
CREATE DATABASE SampleDB;
GO
USE SampleDB;
GO
CREATE TABLE Users (
    Id INT IDENTITY PRIMARY KEY,
    Username NVARCHAR(50) UNIQUE,
    Password NVARCHAR(255)
);

CREATE TABLE Products (
    Id INT IDENTITY PRIMARY KEY,
    Name NVARCHAR(100),
    Price DECIMAL(10,2)
);

-- Default login user for the application
INSERT INTO Users (Username, Password) VALUES ('admin', '1234');
GO
```

## Step 5: Verify the Connection

Test that sqlcmd can connect successfully before running the application:

```
sqlcmd -S localhost -U sa -P YourPassword -C
```

If you see the '1>' prompt, the connection is working correctly.

# Configuring the Application

Before building, update the default database connection settings inside the source code.

## Edit src/app.rs

Open the file src/app.rs and locate the Default::default() function (around line 50-60). Update these four lines to match your SQL Server configuration:

```
let server = "localhost".to_string();        // Your SQL Server address
let database = "SampleDB".to_string();       // Database name
let username = "sa".to_string();             // SQL login username
let password = "YourPassword".to_string();   // SQL login password
```

TIP: For a named instance like SQL Express, use double backslash:
let server = "localhost\\SQLEXPRESS".to_string();

TIP: You can also change these settings at runtime using the
'Database Settings' button on the login screen without rebuilding.

## Verify Cargo.toml

Ensure your Cargo.toml file contains the following exact dependencies. Using the wrong features (e.g. 'rustls' instead of 'native-tls') will cause compile errors:

```
[package]
name = "rust_mssql_crud"
version = "0.1.0"
edition = "2021"

[dependencies]
eframe = "0.29"
egui = "0.29"
egui_extras = { version = "0.29", features = ["default", "serde"] }
tiberius = { version = "0.12", features = ["sql-browser-tokio", "native-tls",
"chrono"] }
tokio = { version = "1", features = ["full"] }
tokio-util = { version = "0.7", features = ["compat"] }
async-trait = "0.1"
serde = { version = "1.0", features = ["derive"] }
serde_json = "1.0"
```

# Building the Application

## Option A: Release Build (Recommended)

Use the release build for the best performance. This is what end users should run.

```
cd rust_mssql_crud
cargo clean
cargo build --release
```

> NOTE: The first build downloads and compiles all dependencies.
> This can take 5-10 minutes. Subsequent builds are much faster (~10 seconds).

## Option B: Debug Build (Development)

Use the debug build during development. It compiles faster but runs slower.

```
cargo build
```

## Running the Application

```
# Release mode (recommended)
cargo run --release

# Or run the compiled executable directly:
# Windows
target\release\rust_mssql_crud.exe

# Linux / macOS
./target/release/rust_mssql_crud
```

# Using the Application

## Login Screen

When the application launches, a login screen appears. Enter your application credentials (stored in the Users table, not the SQL Server credentials):

| Field | Default Value | Notes |
|---|---|---|

| | | |
|---|---|---|
| Username | admin | Stored in Users table in SampleDB |
| Password | 1234 | Stored in Users table in SampleDB |

> IMPORTANT: The login credentials (admin / 1234) are the application-level user credentials stored in your SampleDB Users table.
> They are NOT the SQL Server credentials (sa / YourPassword).
> These are two separate credential sets.

The 'Database Settings' button opens a configuration dialog where you can change the SQL Server connection details at runtime without needing to recompile.

### Users Management

- View all users listed in a scrollable table
- Create a new user by filling in Username and Password, then clicking 'Create User'
- Edit an existing user by clicking the 'Edit' button on a row, modifying the fields, and clicking 'Update User'
- Delete a user by clicking the 'Delete' button on that row
- Click 'Refresh' to reload the latest data from the database

### Products Management

- View all products with their ID, name, and formatted price
- Create a product by entering a Name and Price (numeric), then clicking 'Create Product'
- Edit a product by clicking 'Edit', updating the fields, and clicking 'Update Product'
- Delete a product by clicking the 'Delete' button
- Click 'Refresh' to reload data from the database

## Troubleshooting

| Error | Cause | Solution |
|---|---|---|
| No connection could be made (error 10061) | SQL Server not running or TCP/IP disabled | Start SQL Server service; enable TCP/IP in Configuration Manager; set port to 1433 |
| Login failed for user 'sa' | Wrong password or SA disabled | Reset SA password using Windows Auth: ALTER LOGIN sa WITH PASSWORD = '...' |

| | | |
|---|---|---|
| SSL certificate not trusted | Missing trust_cert() in code | Ensure config.trust_cert() is present in src/db_config.rs |
| Numeric / DECIMAL conversion error | DECIMAL(10,2) cannot be cast directly to f64 | Use the Numeric-aware conversion in get_all_products() in database.rs |
| Cannot borrow self as mutable (E0502) | Rust borrow checker: editing self inside a loop over self | Collect edit/delete actions into Option variables, apply them after the loop |
| egui_extras feature 'all' not found | Invalid feature name | Change to features = ["default", "serde"] in Cargo.toml |
| tiberius TLS compile error | Conflict between rustls and native-tls features | Use features = ["native-tls"] instead of ["rustls"] in Cargo.toml |
| Process exited with code 101 | Runtime panic in the application | Run with RUST_BACKTRACE=1 to see the exact line causing the panic |

## Running with Full Backtrace

When the application crashes, enable the full backtrace to find the exact cause:

```
# Windows Command Prompt
set RUST_BACKTRACE=1
cargo run --release

# Windows PowerShell
$env:RUST_BACKTRACE=1
cargo run --release

# Linux / macOS
RUST_BACKTRACE=1 cargo run --release
```

## Testing the Database Connection Independently

Before running the application, verify the database connection works using sqlcmd:

```
# Connect with SQL Authentication
sqlcmd -S localhost -U sa -P YourPassword -C

# Once connected, verify the database and data:
USE SampleDB;
GO
SELECT * FROM Users;
GO
SELECT * FROM Products;
GO
quit
```

# Quick Reference Card

## Essential Commands

| Task | Command |
|---|---|
| First time build | cargo build --release |
| Run application | cargo run --release |
| Clean build cache | cargo clean |
| Clean and rebuild | cargo clean && cargo build --release |
| Test DB connection | sqlcmd -S localhost -U sa -P YourPassword -C |
| Connect with Windows Auth | sqlcmd -S localhost -E -C |
| Reset SA password | ALTER LOGIN sa WITH PASSWORD = 'newpass', CHECK_POLICY = OFF; |
| Enable SA login | ALTER LOGIN sa ENABLE; |
| Check SQL Server mode | SELECT SERVERPROPERTY('IsIntegratedSecurityOnly'); |

## Key Files to Edit

| File | What to Change |
|---|---|
| src/app.rs | Default database connection (server, database, username, password) |
| src/database.rs | Add new CRUD operations for additional tables |
| src/models.rs | Add new data structures for additional tables |
| src/db_config.rs | Connection configuration, SSL settings |
| Cargo.toml | Add or update dependencies |

## Default Login Credentials

| Type | Username | Password | Purpose |
|---|---|---|---|
| Application Login | admin | 1234 | Login screen of the CRUD app |
| SQL Server | sa | YourPassword | Connecting to SQL Server |

*End of Setup Guide — Rust MSSQL CRUD System*