**# Jenny\_Wu\_F24\_MP7 & IP2**

[![Rust CI/CD](https://github.com/nogibjj/Jenny\_Wu\_F24\_IP2/actions/workflows/rust\_actions.yml/badge.svg)](https://github.com/nogibjj/Jenny\_Wu\_F24\_IP2/actions/workflows/rust\_actions.yml)

## Project #2: Rust CLI Binary with SQLite

\*\*\*

### Purpose:

The purpose of this project is to take an existing python project, convert it to Rust, and compare the speed and efficiency of both scripts. This projects demonstrates how to perform CRUD operations in both Rust and Python, using a CLI (Command Line Integration) Tool.

### Project Guidelines:

Rust source code: The code should comprehensively understand Rust's syntax and unique features.

Use of LLM: In your README, explain how you utilized an LLM in your coding process.

SQLite Database: Include a SQLite database and demonstrate CRUD (Create, Read, Update, Delete) operations.

Optimized Rust Binary: Include a process that generates an optimized Rust binary as a Gitlab Actions artifact that can be downloaded.

README.md: A file that clearly explains what the project does, its dependencies, how to run the program, and how Gitlab Copilot was used.

Github/Gitlab Actions: A workflow file that tests, builds, and lints your Rust code.

Video Demo: A YouTube link in README.md showing a clear, concise walkthrough and demonstration of your CLI binary.

\*\*\*

### Binary Artifact: https://github.com/nogibjj/Jenny\_Wu\_F24\_IP2/actions/runs/11717299977/artifacts/2156028161

### LLM Usage: To help me complete the project, I utilized Codeium to supplement my numerous variable inputs and ChatGPT to check my code. I found that using Codeium was helpful in their suggestions as to how to move forward with my code once I started type a couple of lines to demonstrate what I wanted to achieve. As such, when inputting the list of variables and their respective variable formats, it was quite easy. I used ChatGPT to check my code when I encountered errors and for it to walk me through some of the Rust schemas. I thought that having a bot support helped me learn what I was missing and get me acquainted and recognize common errors I was consistently running into.

\*\*\*

### Rust Dependencies

To begin, we will need clap, rusqlite, and csv for dependencies. This can be set up by running the following commands in the terminal:

- [x] cargo add clap --features derive

- [x] cargo add rusqlite

- [x] cargo add csv

After running these commands, the dependencies will appear in the Cargo.toml file.

### Workflow Structure

Each time a change is made to the code, it will need to be saved and compiled, in order for the program to update itself. This can be done by running:

- [x] Cargo check

- [x] Cargo build

- [x] Cargo build --release

> This cargo build --release command generates a binary file within the project/target/release folder. This file ultimately is what gets returned as a binary artifact when running the GitHub actions. [Binary Artifact] (<https://github.com/nogibjj/Jenny_Wu_F24_IP2/actions/runs/11717299977/artifacts/2156028161>)

### CLI:

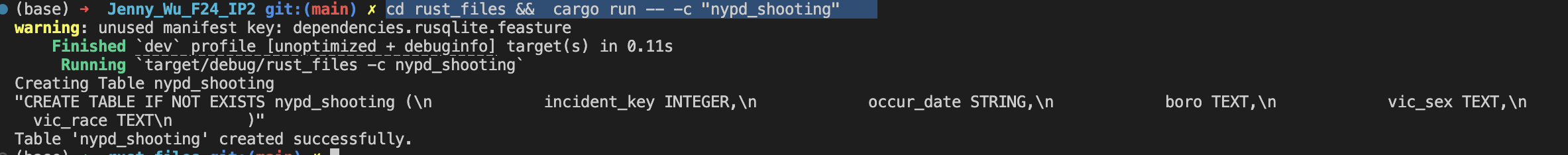
To run through the creation, loading, and querying of the table, please following the following steps.

Using the created CLI tool to create a table, run

```

cd rust\_files -- -c "nypd\_shooting"

```



Now that the directory is in the “rust\_files”, we can run all cargo commands as the CLI and Makefile is within this directory.

```

cargo run -- -l "nypd\_shooting" "data/nypd\_shooting.csv"

```

A screenshot of a computer screen

Description automatically generated

```

cargo run -- -q "SELECT \* FROM nypd\_shooting LIMIT 5"

A screen shot of a computer

Description automatically generated

```

A screen shot of a computer code

Description automatically generated

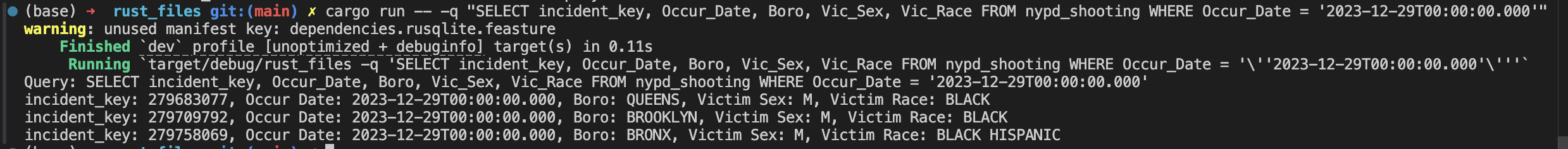
A screen shot of a computer

Description automatically generated

```

cargo run -- -q "SELECT incident\_key, Occur\_Date, Boro, Vic\_Sex, Vic\_Race FROM nypd\_shooting WHERE Occur\_Date = '2023-12-29T00:00:00.000'"

```



\*\*\*

## Results:

### Rust vs Python Speed

In addition to rewriting the python script, this project also provides a comparison of Rust and Python's speed and resource usage advantages and disadvantages. Rust is a compiled (or binary deployment) language, meaning that the code compiles into a single executable file. This method is faster and more efficient for users, but has limited flexibility, as binaries are platform-specific.

In comparison, Python is slower, but more user friendly. Packaging deployment is common in Python and involves distributing code as packages. These packages require an interpreter (Python) to run. Unlike Rust, this method is not platform-specific, but is much slower for a couple of reasons, including having to manage more dependencies.

To compare the two, I compared the speed of the Rust and Python code in loading and reading the table. In this project, you can run this by running the following steps: