COMP1314 Data Management

Coursework: Gold Price Tracker (Documentation)

Tutor:

Dr Fairuz Safwan Mahad

F.S.Mahad@soton.ac.uk

University of Southampton Malaysia

Faculty of Electronics and Computer Science

Prepared by:

Ong Hui Min

hmo1e25@soton.ac.uk

University of Southampton Malaysia

BCs Computer Science Part 1

Faculty of Electronics and Computer Science

15 December 2025

Table of Contents

1.0 INTRODUCTION

This coursework implements a **Unix-based automated gold price tracking system** that collects live gold price data from an external website, Kitco at https://www.kitco.com/charts/gold, processes and converts the data into multiple currencies (AUD, CNY, EUR, GBP, USD) and weights units (ounce, gram, kilo, pennyweight, tola and tael) that stores the data in a relational MySQL database and visualises historical trends using automated plots.

The system was designed and implemented using **Bash scripting**, **MySQL**, and **Gnuplot**, running inside a **Linux (WSL) environment**. Automation is achieved through **crontab scheduling**, enabling the system to execute without manual intervention at fixed time intervals.

The key objectives of this project are:

- To demonstrate the ability to **collect and parse real-world web data**
- To perform **data cleaning, manipulation, and currency conversion**
- To design and implement a **normalized relational database**
- To insert and retrieve data programmatically using Unix scripts
- To generate meaningful **data visualizations**
- To automate the entire pipeline using **crontab**
- To maintain proper **version control** using Git and GitHub

Ong Hui Min hmo1e25

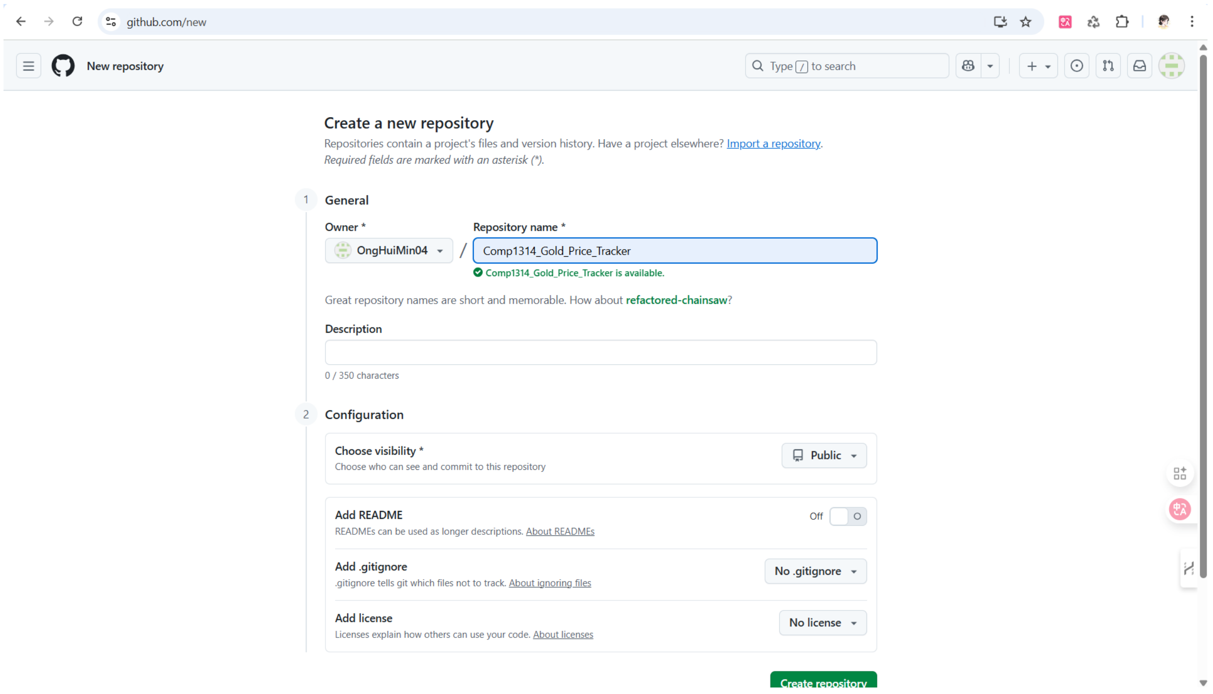
2.0 GITHUB SETUP

Before setting up the development environment, a GitHub repository is created to store all scripts, SQL files, and documentation for this coursework. This ensures proper version control, organisation, and ease of submission.

Step 1: Create the GitHub Repository

1. Log in to **GitHub**.
2. Click **New Repository**.
3. Enter the repository name: Comp1314_Gold_Price_Tracker
4. The repository visibility was set to **Public**.
5. The repository was initialised without additional files (no. gitignore or license at this stage).

This repository serves as the central location for all Bash scripts, SQL schemas, ERD diagrams, logs, plots, and documentation related to the coursework.



github.com/new

New repository

Type / to search

Create a new repository

Repositories contain a project's files and version history. Have a project elsewhere? Import a repository.

Required fields are marked with an asterisk (*).

1 General

Owner *: OngHuiMin04

Repository name *: Comp1314_Gold_Price_Tracker

Comp1314_Gold_Price_Tracker is available.

Great repository names are short and memorable. How about refactored-chainsaw?

Description

0 / 350 characters

2 Configuration

Choose visibility *: Public
Choose who can see and commit to this repository

Add README: Off
READMEs can be used as longer descriptions. About READMEs

Add .gitignore: No .gitignore
.gitignore tells git which files not to track. About ignoring files

Add license: No license
Licenses explain how others can use your code. About licenses

Create repository

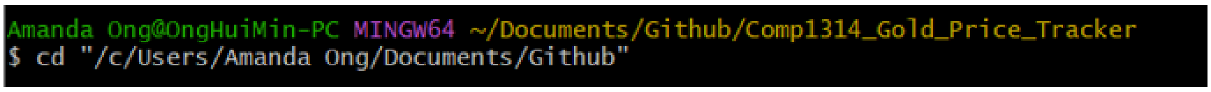
Figure 1: Creating a repository name at GitHub.

Ong Hui Min hmo1e25

Step 2: Initialising the Local Git Repository (Windows)

After creating the remote repository on GitHub, a corresponding local repository was set up on the Windows system using **Git Bash**.

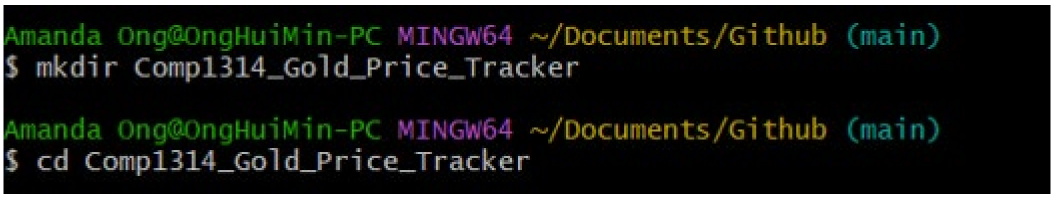
1. Navigate to the local GitHub workspace directory:



```
Amanda Ong@OngHuiMin-PC MINGW64 ~/Documents/Github/Comp1314_Gold_Price_Tracker
$ cd "/c/Users/Amanda Ong/Documents/Github"
```

Figure 2: Bash Comment that was used to process this repository at the Git Bash terminal.

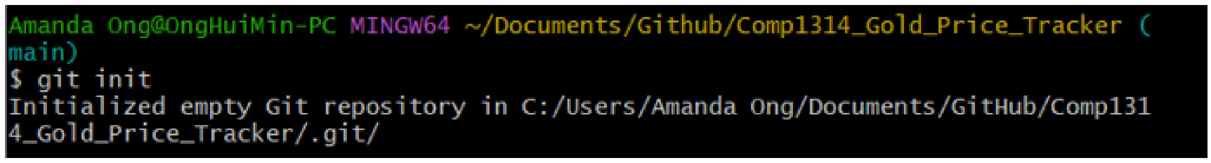
2. Create a new project folder matching the repository name:



```
Amanda Ong@OngHuiMin-PC MINGW64 ~/Documents/Github (main)
$ mkdir Comp1314_Gold_Price_Tracker

Amanda Ong@OngHuiMin-PC MINGW64 ~/Documents/Github (main)
$ cd Comp1314_Gold_Price_Tracker
```

Figure 3: Bash Comment that was used to process this repository at the Git Bash terminal.

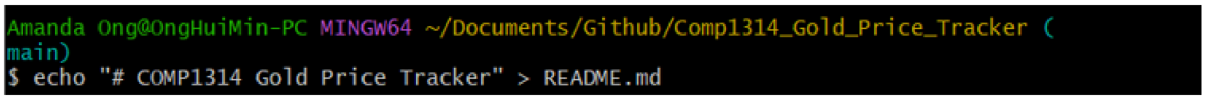
3. Initialise the Git repository:



```
Amanda Ong@OngHuiMin-PC MINGW64 ~/Documents/Github/Comp1314_Gold_Price_Tracker (
main)
$ git init
Initialized empty Git repository in C:/Users/Amanda Ong/Documents/GitHub/Comp131
4_Gold_Price_Tracker/.git/
```

Figure 4: Bash Comment that was used to process this repository at the Git Bash terminal.

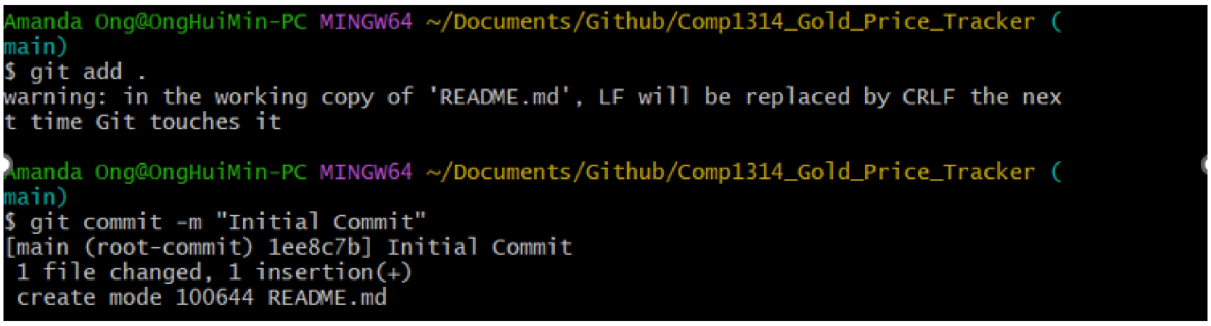
4. Create the initial README file:



```
Amanda Ong@OngHuiMin-PC MINGW64 ~/Documents/Github/Comp1314_Gold_Price_Tracker (
main)
$ echo "# COMP1314 Gold Price Tracker" > README.md
```

Figure 5: Bash Comment that was used to process this repository at the Git Bash terminal.

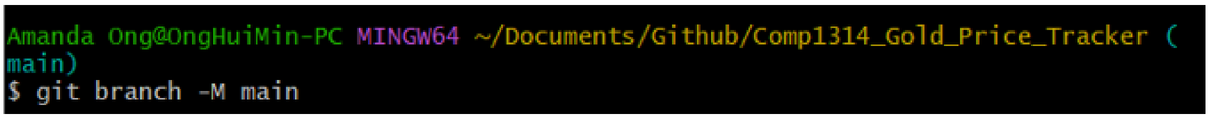
5. Stage and commit the initial files:



```
Amanda Ong@OngHuiMin-PC MINGW64 ~/Documents/Github/Comp1314_Gold_Price_Tracker (
main)
$ git add .
warning: in the working copy of 'README.md', LF will be replaced by CRLF the nex
t time Git touches it

Amanda Ong@OngHuiMin-PC MINGW64 ~/Documents/Github/Comp1314_Gold_Price_Tracker (
main)
$ git commit -m "Initial Commit"
[main (root-commit) 1ee8c7b] Initial Commit
 1 file changed, 1 insertion(+)
 create mode 100644 README.md
```

Figure 6: Bash Comment that was used to process this repository at the Git Bash terminal.

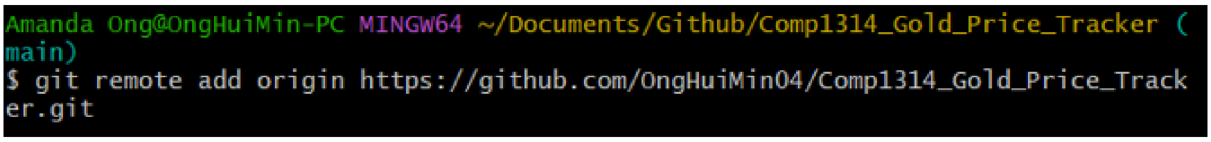
6. Rename the default branch to main:



```
Amanda Ong@OngHuiMin-PC MINGW64 ~/Documents/Github/Comp1314_Gold_Price_Tracker (
main)
$ git branch -M main
```

Figure 7: Bash Comment that was used to process this repository at the Git Bash terminal.

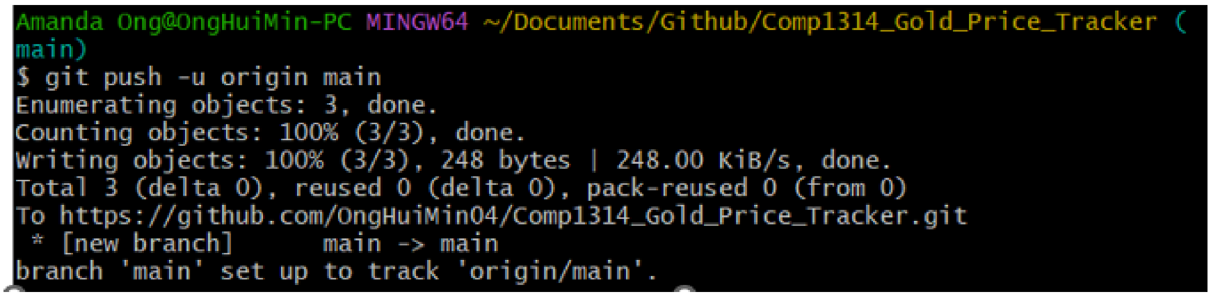
7. Link the local repository to the remote GitHub repository:



```
Amanda Ong@OngHuiMin-PC MINGW64 ~/Documents/Github/Comp1314_Gold_Price_Tracker (
main)
$ git remote add origin https://github.com/OngHuiMin04/Comp1314_Gold_Price_Track
er.git
```

Figure 8: Bash Comment that was used to process this repository at the Git Bash terminal.

8. Push the initial commit to GitHub:



```
Amanda Ong@OngHuiMin-PC MINGW64 ~/Documents/Github/Comp1314_Gold_Price_Tracker (
main)
$ git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 248 bytes | 248.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/OngHuiMin04/Comp1314_Gold_Price_Tracker.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```

Figure 9: Bash Comment that was used to process this repository at the Git Bash terminal.

Once the GitHub repository and local version control were successfully configured, the next step was to install **Windows Subsystem for Linux (WSL)**. This ensures that all Bash scripts, crontab jobs, and Unix-based tools used in the project can execute correctly in a Linux-compatible environment.

3.0 ENVIRONMENT SETUP (WINDOWS + WSL)

The project was developed in a **Windows operating system environment** using the **Windows Subsystem for Linux (WSL)** to provide a fully functional Linux-based development platform. WSL enables the execution of Bash scripts, crontab jobs and Unix command-line utilities directly on a Windows laptop without requiring a separate virtual machine or dual-boot setup.

Using **Ubuntu on WSL** ensures compatibility with Linux-native tools such as curl, awk, cron, mysql and gnuplot, which are essential for web scraping, database interaction, automation and data visualisation in this project. This environment closely mirrors a standard Linux server setup, making the system behaviour predictable and consistent during script execution and scheduled tasks.

All project scripts, SQL files and output directories are organised within a single GitHub repository to maintain portability, reproducibility and ease of assessment. The environment setup described in the following sections ensures that the system can be deployed and executed reliably on any Windows machine that supports WSL.

3.1 Installing WSL (Windows)

Instead of manually enabling WSL through PowerShell, Ubuntu was installed directly via the Microsoft Store. This method automatically enables Windows Subsystem for Linux (WSL) and configures the required Linux environment.

Steps Performed

1. Open the **Microsoft Store** on Windows.
2. Search for **Ubuntu**.
3. Download and install **Ubuntu (latest LTS version)**.
4. Launch **Ubuntu** from the Start menu.
5. Complete the first-time setup by creating a Linux **username and password**.
6. After setup, the Ubuntu terminal opens and is ready for use.

During installation, Windows automatically enables WSL in the background. No additional manual configuration was required.

Verification

To confirm that WSL is installed correctly, run the following command in PowerShell or Ubuntu: wsl –status. This proves that WSL is active and that Ubuntu is registered as a Linux distribution.

3.2 Installing Required Packages in WSL

Once Ubuntu was installed and configured, the required packages were installed using the Ubuntu package manager.

```
sudo apt update
```

```
sudo apt install -y curl grep gawk bc mysql-client gnuplot
```

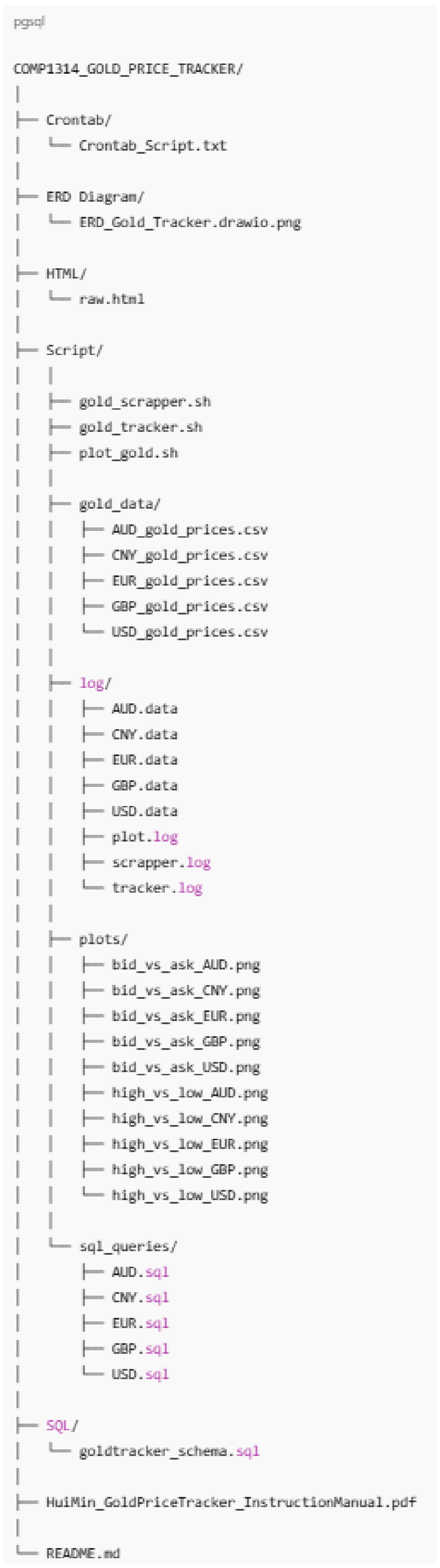
Explanation of Key Tools

- **curl**
  Used to download the HTML page containing live gold price data.
- **grep / awk**

  Used to extract specific values from the downloaded HTML and perform text processing.
- **gnuplot**
  Generates historical gold price plots automatically from stored data.

3.3 Project Folder and Directory Structure

All files are organised into a single GitHub project folder. Scripts are placed inside a dedicated Script directory, and output folders are created automatically by the scripts to ensure compatibility with crontab jobs.



```
pgsql

COMP1314_GOLD_PRICE_TRACKER/
│
├── Crontab/
│   └── Crontab_Script.txt
│
├── ERD Diagram/
│   └── ERD_Gold_Tracker.drawio.png
│
├── HTML/
│   └── raw.html
│
├── Script/
│   │
│   ├── gold_scrapper.sh
│   ├── gold_tracker.sh
│   ├── plot_gold.sh
│   │
│   ├── gold_data/
│   │   ├── AUD_gold_prices.csv
│   │   ├── CNY_gold_prices.csv
│   │   ├── EUR_gold_prices.csv
│   │   ├── GBP_gold_prices.csv
│   │   └── USD_gold_prices.csv
│   │
│   ├── log/
│   │   ├── AUD.data
│   │   ├── CNY.data
│   │   ├── EUR.data
│   │   ├── GBP.data
│   │   ├── USD.data
│   │   ├── plot.log
│   │   ├── scrapper.log
│   │   └── tracker.log
│   │
│   ├── plots/
│   │   ├── bid_vs_ask_AUD.png
│   │   ├── bid_vs_ask_CNY.png
│   │   ├── bid_vs_ask_EUR.png
│   │   ├── bid_vs_ask_GBP.png
│   │   ├── bid_vs_ask_USD.png
│   │   ├── high_vs_low_AUD.png
│   │   ├── high_vs_low_CNY.png
│   │   ├── high_vs_low_EUR.png
│   │   ├── high_vs_low_GBP.png
│   │   └── high_vs_low_USD.png
│   │
│   └── sql_queries/
│       ├── AUD.sql
│       ├── CNY.sql
│       ├── EUR.sql
│       ├── GBP.sql
│       └── USD.sql
│
├── SQL/
│   └── goldtracker_schema.sql
│
├── HuiMin_GoldPriceTracker_InstructionManual.pdf
│
└── README.md
```

Figure 10: File Directory Structure

3.3.1 Script

The directory contains all Bash scripts that implement the Gold Price Tracker System**.**

- **gold_scrapper.sh**
  Scrapes live gold price data, performs unit and currency conversion, and inserts records into the MySQL database.
- **gold_tracker.sh**
  Acts as the main controller script, coordinating scraping, database insertion, logging, and plotting.
- **plot_gold.sh**
  Queries historical data and generates visual plots using Gnuplot.

3.3.2 gold_data/

It stores CSV files generated from database queries. Each file represents historical gold price data for a specific currency.

3.3.3 log/

It contains log and debug output produced during script execution and crontab automation. These logs are used to verify correct execution and troubleshoot errors.

3.3.4 plots/

It stores all PNG graphs generated by Gnuplot, including:

- Bid vs Ask price comparisons
- High vs Low price comparisons for each supported currency.

3.3.5 sql_queries/

It contains SQL query files automatically generated by the system. These queries are used to extract data from the database for plotting and reporting.

3.3.6 SQL/

It contains the database schema files:

- goldtracker_schema.sql
  Defines all tables, keys, and relationships based on the ERD design.

3.3.7 Crontab/

It stores the crontab configuration file used to automate the execution of the tracker at fixed intervals.

3.3.8 ERD Diagram/

It contains the ERD used during database design and normalisation.

3.3.9 Documentation

- Instruction Manual: Full step-by-step setup and usage guide
- READ.md: The project overview stated in the GitHub Profile
- Documentation: It’s a step-by-step explanation of how the project was built from start to finish.

4.0 UNIT SCRIPT FOR DATA COLLECTION

This section describes the implementation of the **unit script responsible for collecting gold price data** from the Kitco website. The script is implemented using **Bash,** and it performs data collection by **downloading the HTML page once and extracting the required values directly from the saved file**. The design ensures repeatability, reduces network dependency during parsing, and allows the script to be safely scheduled using crontab.

4.1 Downloading the HTML File, Set up Working Directory and Generating Timestamps

4.1.1 HTML Download Process

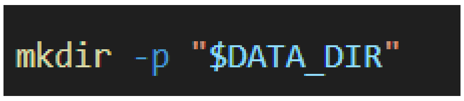
The first step of the data collection process is to download the Kitco gold price webpage and store it locally as a raw HTML file. This is done using the curl command with a browser-compatible user agent to ensure the correct page content is returned.

```
PAGE= “https://www.kitco.com/charts/gold”
RAW_FILE= “raw.html”
curl -s -H “User-Agent: Mozilla/5.0” “$PAGE” -o “$RAW_FILE”
```

The downloaded file (raw.html) serves as the **single source of truth** for all subsequent data extraction operations. All parsing operations are performed **offline on this file**, ensuring consistency across different script executions and preventing partial data extraction caused by network latency. The structure of the downloaded HTML confirms that the gold price data is embedded directly in the page content, including bid, ask, unit prices, and currency conversion metadata.

4.1.2 Set up working directory and output folders

At the start of the script, the working directory is set to the project’s Script folder. Required directories such as gold_data, log, plots, and sql_queries are created if they do not already exist. This prevents file and path errors during execution.



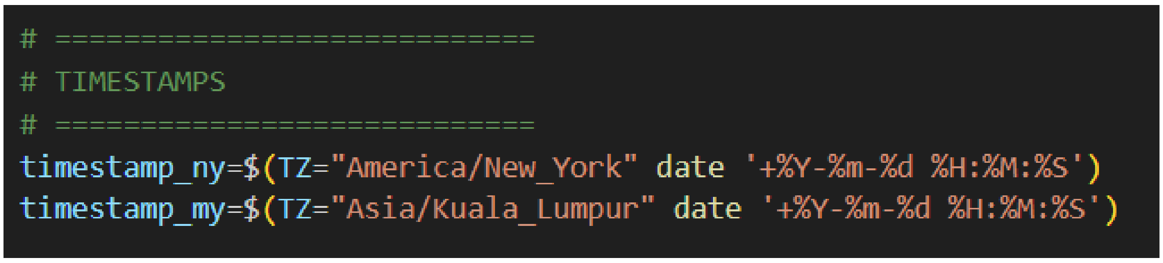
```
mkdir -p "$DATA_DIR"
```

Figure 11: Code for generating working directory and output folders.

4.1.3 Timestamp Generation

Two timestamps are generated to reflect different time zones:

- **New York time** – represents the gold market data timestamp
- **Malaysia time** – represents the script execution timestamp



```
# ============================
# TIMESTAMPS
# ============================
timestamp_ny=$(TZ="America/New_York" date '+%Y-%m-%d %H:%M:%S')
timestamp_my=$(TZ="Asia/Kuala_Lumpur" date '+%Y-%m-%d %H:%M:%S')
```

Figure 12: Code for generating different timestamps.

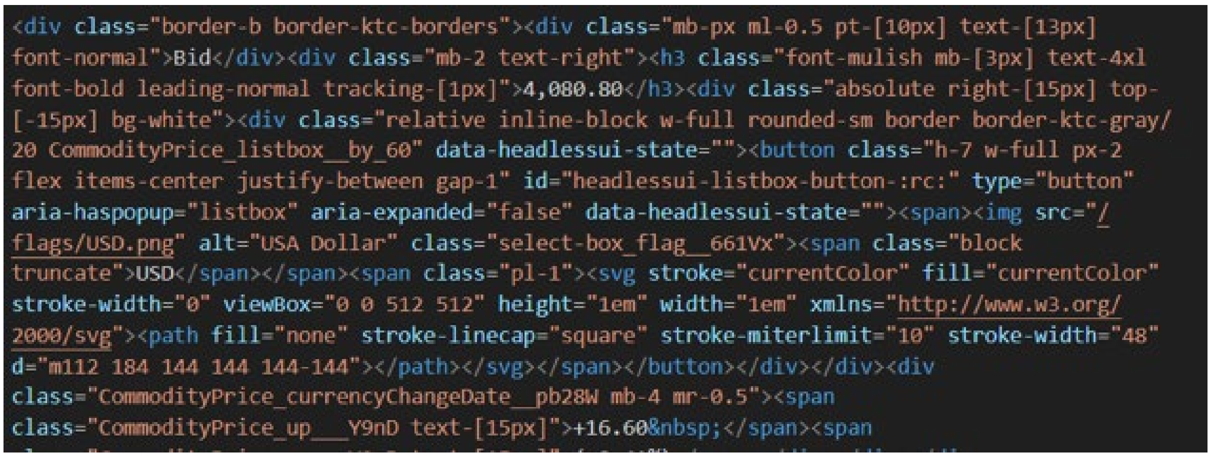
4.2 Extracting Required Data from the HTML File

After the HTML file is downloaded, the tracker extracts only the required gold price values using **pattern-based parsing** (grep -P). No external APIs or JavaScript execution is involved.

4.2.1 Bid and Ask Price Extraction

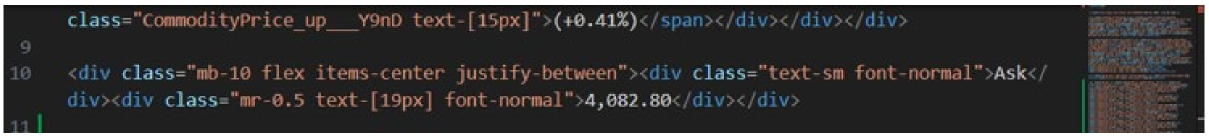
From the HTML file, the bid price is located inside an <h3> element, while the **ask price** is displayed in a <div> styled with a specific font class.

HTML Source



```
<div class="border-b border-ktc-borders"><div class="mb-px ml-0.5 pt-[10px] text-[13px]
font-normal">Bid</div><div class="mb-2 text-right"><h3 class="font-mulish mb-[3px] text-4xl
font-bold leading-normal tracking-[1px]">4,080.80</h3><div class="absolute right-[15px] top-
[-15px] bg-white"><div class="relative inline-block w-full rounded-sm border border-ktc-gray/
20 CommodityPrice_listbox__by_60" data-headlessui-state=""><button class="h-7 w-full px-2
flex items-center justify-between gap-1" id="headlessui-listbox-button-:rc:" type="button"
aria-haspopup="listbox" aria-expanded="false" data-headlessui-state=""><span><img src="/
flags/USD.png" alt="USA Dollar" class="select-box_flag__661Vx"><span class="block
truncate">USD</span></span><span class="pl-1"><svg stroke="currentColor" fill="currentColor"
stroke-width="0" viewBox="0 0 512 512" height="1em" width="1em" xmlns="http://www.w3.org/
2000/svg"><path fill="none" stroke-linecap="square" stroke-miterlimit="10" stroke-width="48"
d="m112 184 144 144 144-144"></path></svg></span></button></div></div><div
class="CommodityPrice_currencyChangeDate__pb28W mb-4 mr-0.5"><span
class="CommodityPrice_up__Y9nD text-[15px]">+16.60&nbsp;</span><span
```

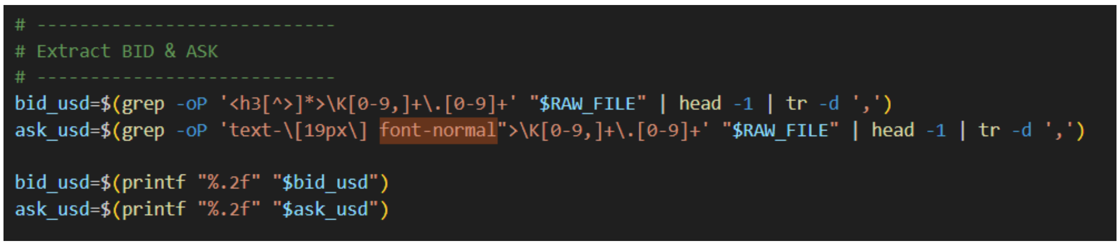
Figure 13: HTML for bid price.



```
class="CommodityPrice_up__Y9nD text-[15px]">(+0.41%)</span></div></div></div>

<div class="mb-10 flex items-center justify-between"><div class="text-sm font-normal">Ask</
div><div class="mr-0.5 text-[19px] font-normal">4,082.80</div></div>
```

Figure 14: HTML for ask price.

Bash Extraction Logic



```
# ---------------------------
# Extract BID & ASK
# ---------------------------
bid_usd=$(grep -oP '<h3[^>]*>\K[0-9,]+\.[0-9]+' "$RAW_FILE" | head -1 | tr -d ',')
ask_usd=$(grep -oP 'text-\[19px\] font-normal">\K[0-9,]+\.[0-9]+' "$RAW_FILE" | head -1 | tr -d ',')

bid_usd=$(printf "%.2f" "$bid_usd")
ask_usd=$(printf "%.2f" "$ask_usd")
```

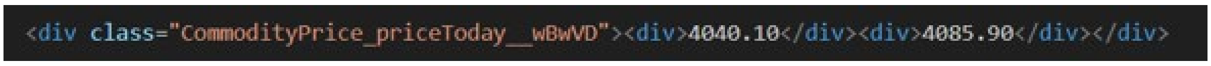
Figure 15: Bash Extraction Logic for bid and ask price.

These values are then formatted to two decimal places to ensure numerical consistency.

4.2.2 High and Low Price Extraction

The daily high and low prices are embedded within a container identified by the CommodityPrice_priceToday__ class.

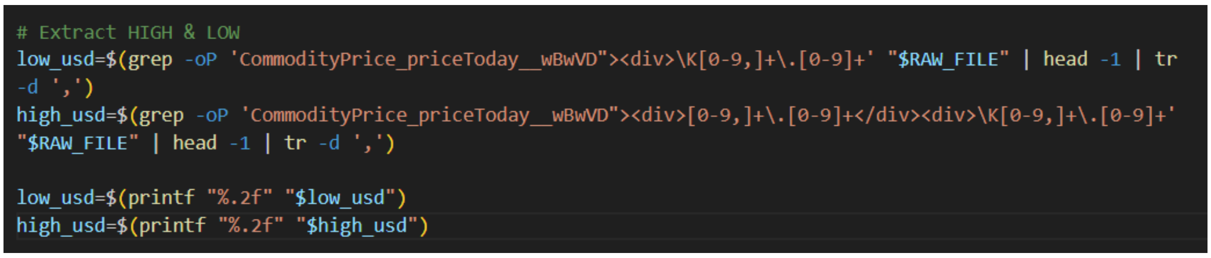
HTML Source



```
<div class="CommodityPrice_priceToday__wBwVD"><div>4040.10</div><div>4085.90</div></div>
```

Figure 16: HTML for high and low prices.

Bash Extraction Logic



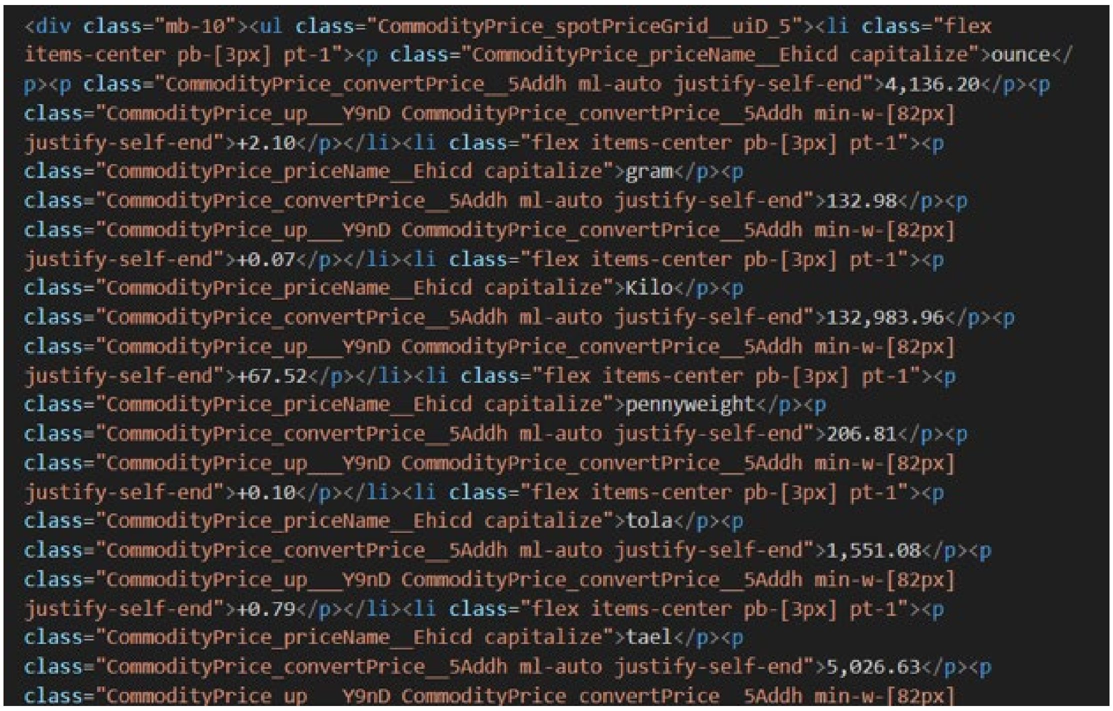
```
# Extract HIGH & LOW
low_usd=$(grep -oP 'CommodityPrice_priceToday__wBwVD"><div>\K[0-9,]+\.[0-9]+' "$RAW_FILE" | head -1 | tr
-d ',')
high_usd=$(grep -oP 'CommodityPrice_priceToday__wBwVD"><div>[0-9,]+\.[0-9]+</div><div>\K[0-9,]+\.[0-9]+'
"$RAW_FILE" | head -1 | tr -d ',')

low_usd=$(printf "%.2f" "$low_usd")
high_usd=$(printf "%.2f" "$high_usd")
```

Figure 17: Bash Extraction Logic for high and low prices.

4.2.3 Unit Price Extraction (Ounce, Gram, Kilo, Pennyweight, Tola, Tael)

Gold prices per unit (ounce, gram, kilo, pennyweight, tola, tael) are displayed in a structured list format.

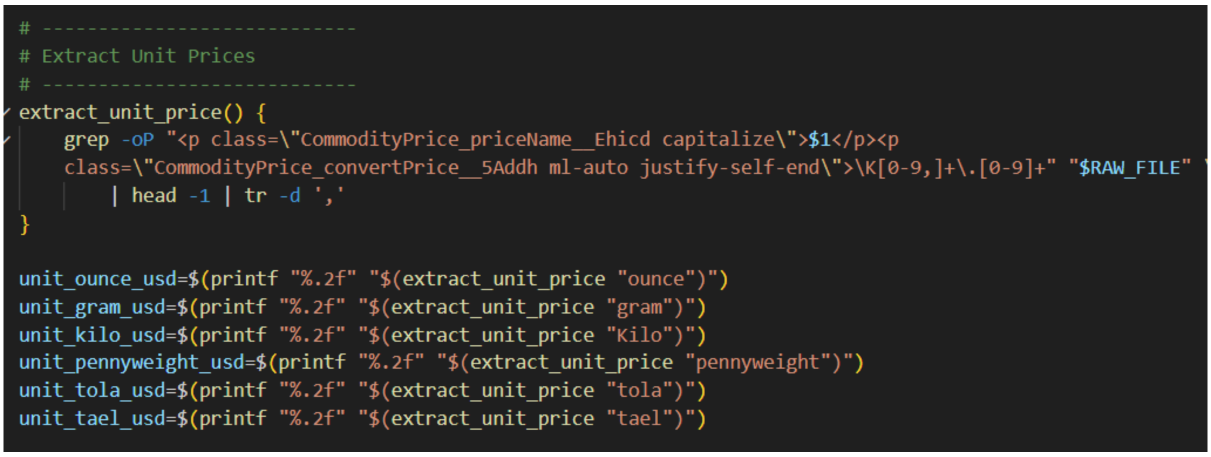
HTML Source



```
<div class="mb-10"><ul class="CommodityPrice_spotPriceGrid__uiD_5"><li class="flex
items-center pb-[3px] pt-1"><p class="CommodityPrice_priceName__Ehicd capitalize">ounce</
p><p class="CommodityPrice_convertPrice__5Addh ml-auto justify-self-end">4,136.20</p><p
class="CommodityPrice_up___Y9nD CommodityPrice_convertPrice__5Addh min-w-[82px]
justify-self-end">+2.10</p></li><li class="flex items-center pb-[3px] pt-1"><p
class="CommodityPrice_priceName__Ehicd capitalize">gram</p><p
class="CommodityPrice_convertPrice__5Addh ml-auto justify-self-end">132.98</p><p
class="CommodityPrice_up___Y9nD CommodityPrice_convertPrice__5Addh min-w-[82px]
justify-self-end">+0.07</p></li><li class="flex items-center pb-[3px] pt-1"><p
class="CommodityPrice_priceName__Ehicd capitalize">Kilo</p><p
class="CommodityPrice_convertPrice__5Addh ml-auto justify-self-end">132,983.96</p><p
class="CommodityPrice_up___Y9nD CommodityPrice_convertPrice__5Addh min-w-[82px]
justify-self-end">+67.52</p></li><li class="flex items-center pb-[3px] pt-1"><p
class="CommodityPrice_priceName__Ehicd capitalize">pennyweight</p><p
class="CommodityPrice_convertPrice__5Addh ml-auto justify-self-end">206.81</p><p
class="CommodityPrice_up___Y9nD CommodityPrice_convertPrice__5Addh min-w-[82px]
justify-self-end">+0.10</p></li><li class="flex items-center pb-[3px] pt-1"><p
class="CommodityPrice_priceName__Ehicd capitalize">tola</p><p
class="CommodityPrice_convertPrice__5Addh ml-auto justify-self-end">1,551.08</p><p
class="CommodityPrice_up___Y9nD CommodityPrice_convertPrice__5Addh min-w-[82px]
justify-self-end">+0.79</p></li><li class="flex items-center pb-[3px] pt-1"><p
class="CommodityPrice_priceName__Ehicd capitalize">tael</p><p
class="CommodityPrice_convertPrice__5Addh ml-auto justify-self-end">5,026.63</p><p
class="CommodityPrice_up___Y9nD CommodityPrice_convertPrice__5Addh min-w-[82px]
```

Figure 18: HTML for unit prices.

Bash Extraction Logic

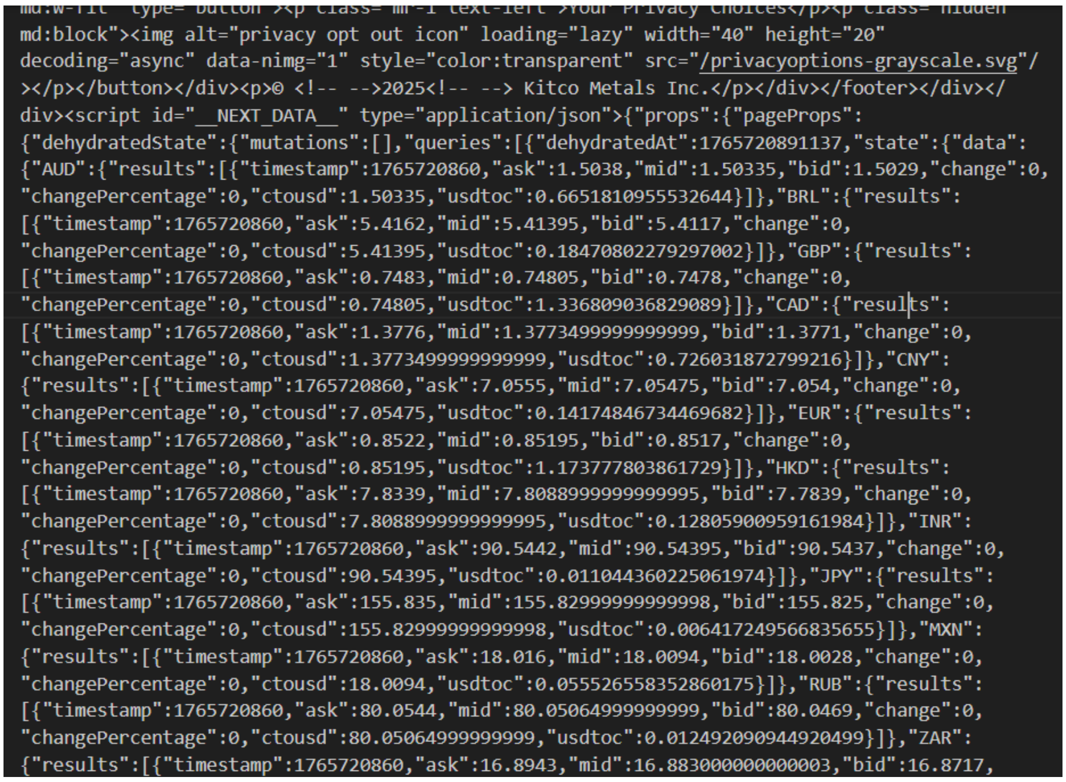


```
# ---------------------------
# Extract Unit Prices
# ---------------------------
extract_unit_price() {
    grep -oP "<p class=\"CommodityPrice_priceName__Ehicd capitalize\">$1</p><p
    class=\"CommodityPrice_convertPrice__5Addh ml-auto justify-self-end\">\K[0-9,]+\.[0-9]+" "$RAW_FILE"
        | head -1 | tr -d ','
}

unit_ounce_usd=$(printf "%.2f" "$(extract_unit_price "ounce")")
unit_gram_usd=$(printf "%.2f" "$(extract_unit_price "gram")")
unit_kilo_usd=$(printf "%.2f" "$(extract_unit_price "Kilo")")
unit_pennyweight_usd=$(printf "%.2f" "$(extract_unit_price "pennyweight")")
unit_tola_usd=$(printf "%.2f" "$(extract_unit_price "tola")")
unit_tael_usd=$(printf "%.2f" "$(extract_unit_price "tael")")
```

Figure 19: Bash Extraction Logic for unit prices.

4.2.4 Currency Conversion Metadata Extraction

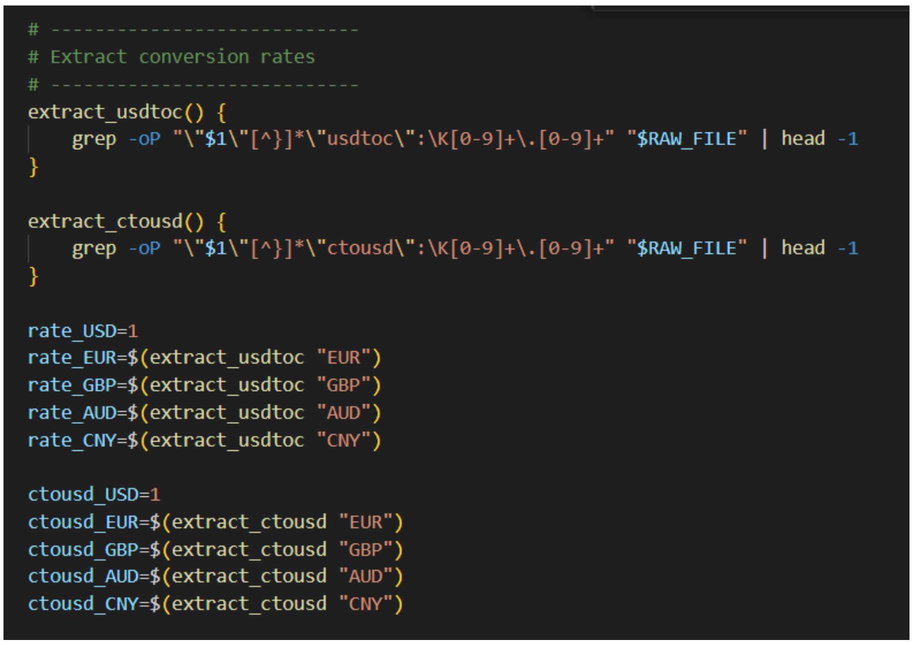
HTML Source



```
md:w-fit" type="button"><p class="mr-1 text-left">Your Privacy Choices</p><p class="hidden
md:block"><img alt="privacy opt out icon" loading="lazy" width="40" height="20"
decoding="async" data-nimg="1" style="color:transparent" src="/privacyoptions-grayscale.svg"/
></p></button></div><p>© <!-- -->2025<!-- --> Kitco Metals Inc.</p></div></footer></div></
div><script id="__NEXT_DATA__" type="application/json">{"props":{"pageProps":
{"dehydratedState":{"mutations":[],"queries":[{"dehydratedAt":1765720891137,"state":{"data":
{"AUD":{"results":[{"timestamp":1765720860,"ask":1.5038,"mid":1.50335,"bid":1.5029,"change":0,
"changePercentage":0,"ctousd":1.50335,"usdtoc":0.6651810955532644}]},"BRL":{"results":
[{"timestamp":1765720860,"ask":5.4162,"mid":5.41395,"bid":5.4117,"change":0,
"changePercentage":0,"ctousd":5.41395,"usdtoc":0.18470802279297002}]},"GBP":{"results":
[{"timestamp":1765720860,"ask":0.7483,"mid":0.74805,"bid":0.7478,"change":0,
"changePercentage":0,"ctousd":0.74805,"usdtoc":1.336809036829089}]},"CAD":{"results":
[{"timestamp":1765720860,"ask":1.3776,"mid":1.3773499999999999,"bid":1.3771,"change":0,
"changePercentage":0,"ctousd":1.3773499999999999,"usdtoc":0.726031872799216}]},"CNY":
{"results":[{"timestamp":1765720860,"ask":7.0555,"mid":7.05475,"bid":7.054,"change":0,
"changePercentage":0,"ctousd":7.05475,"usdtoc":0.14174846734469682}]},"EUR":{"results":
[{"timestamp":1765720860,"ask":0.8522,"mid":0.85195,"bid":0.8517,"change":0,
"changePercentage":0,"ctousd":0.85195,"usdtoc":1.173777803861729}]},"HKD":{"results":
[{"timestamp":1765720860,"ask":7.8339,"mid":7.8088999999999995,"bid":7.7839,"change":0,
"changePercentage":0,"ctousd":7.8088999999999995,"usdtoc":0.12805900959161984}]},"INR":
{"results":[{"timestamp":1765720860,"ask":90.5442,"mid":90.54395,"bid":90.5437,"change":0,
"changePercentage":0,"ctousd":90.54395,"usdtoc":0.011044360225061974}]},"JPY":{"results":
[{"timestamp":1765720860,"ask":155.835,"mid":155.82999999999998,"bid":155.825,"change":0,
"changePercentage":0,"ctousd":155.82999999999998,"usdtoc":0.006417249566835655}]},"MXN":
{"results":[{"timestamp":1765720860,"ask":18.016,"mid":18.0094,"bid":18.0028,"change":0,
"changePercentage":0,"ctousd":18.0094,"usdtoc":0.055526558352860175}]},"RUB":{"results":
[{"timestamp":1765720860,"ask":80.0544,"mid":80.05064999999999,"bid":80.0469,"change":0,
"changePercentage":0,"ctousd":80.05064999999999,"usdtoc":0.012492090944920499}]},"ZAR":
{"results":[{"timestamp":1765720860,"ask":16.8943,"mid":16.883000000000003,"bid":16.8717,
```

Figure 20: HTML for currencies conversion.

Bash Extraction Logic

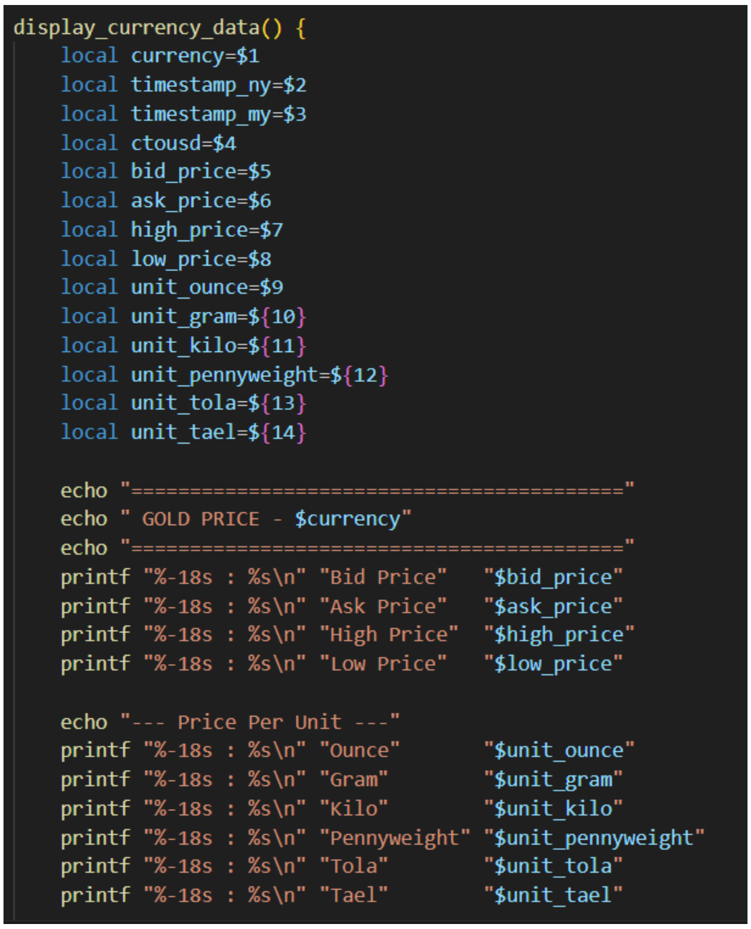


```bash
# ----------------------------
# Extract conversion rates
# ----------------------------
extract_usdtoc() {
    grep -oP "\"$1\"[^}]*\"usdtoc\":\K[0-9]+\.[0-9]+" "$RAW_FILE" | head -1
}

extract_ctousd() {
    grep -oP "\"$1\"[^}]*\"ctousd\":\K[0-9]+\.[0-9]+" "$RAW_FILE" | head -1
}

rate_USD=1
rate_EUR=$(extract_usdtoc "EUR")
rate_GBP=$(extract_usdtoc "GBP")
rate_AUD=$(extract_usdtoc "AUD")
rate_CNY=$(extract_usdtoc "CNY")

ctousd_USD=1
ctousd_EUR=$(extract_ctousd "EUR")
ctousd_GBP=$(extract_ctousd "GBP")
ctousd_AUD=$(extract_ctousd "AUD")
ctousd_CNY=$(extract_ctousd "CNY")
```

Figure 21: Bash Extraction Logic for currencies conversion.

4.2.5 Display values for debugging and clarity

All extracted and converted values are printed in a formatted table. This improves readability during testing and helps detect parsing errors quickly.

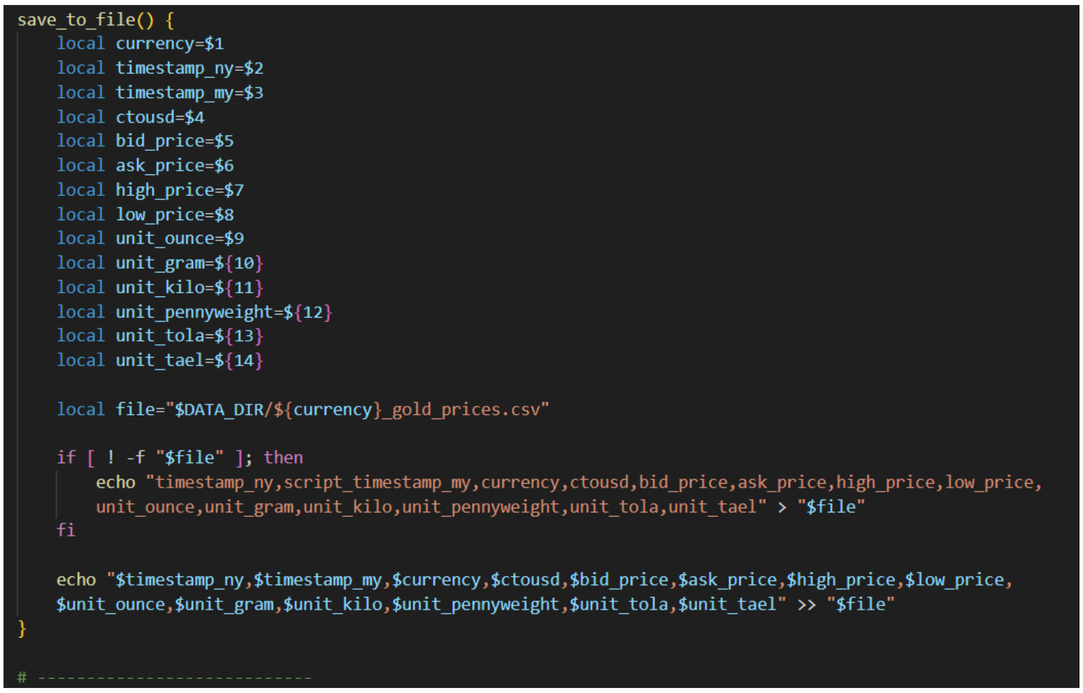


```
display_currency_data() {
    local currency=$1
    local timestamp_ny=$2
    local timestamp_my=$3
    local ctousd=$4
    local bid_price=$5
    local ask_price=$6
    local high_price=$7
    local low_price=$8
    local unit_ounce=$9
    local unit_gram=${10}
    local unit_kilo=${11}
    local unit_pennyweight=${12}
    local unit_tola=${13}
    local unit_tael=${14}

    echo "=========================================="
    echo " GOLD PRICE - $currency"
    echo "=========================================="
    printf "%-18s : %s\n" "Bid Price"   "$bid_price"
    printf "%-18s : %s\n" "Ask Price"   "$ask_price"
    printf "%-18s : %s\n" "High Price"  "$high_price"
    printf "%-18s : %s\n" "Low Price"   "$low_price"

    echo "--- Price Per Unit ---"
    printf "%-18s : %s\n" "Ounce"       "$unit_ounce"
    printf "%-18s : %s\n" "Gram"        "$unit_gram"
    printf "%-18s : %s\n" "Kilo"        "$unit_kilo"
    printf "%-18s : %s\n" "Pennyweight" "$unit_pennyweight"
    printf "%-18s : %s\n" "Tola"        "$unit_tola"
    printf "%-18s : %s\n" "Tael"        "$unit_tael"
```

Figure 22: Code to display values.

4.2.6 Save data into CSV files

Each script run appends a new row into a currency-specific CSV file. If the file does not exist, a header row is written first. These CSV files provide historical data for plotting and act as a backup.

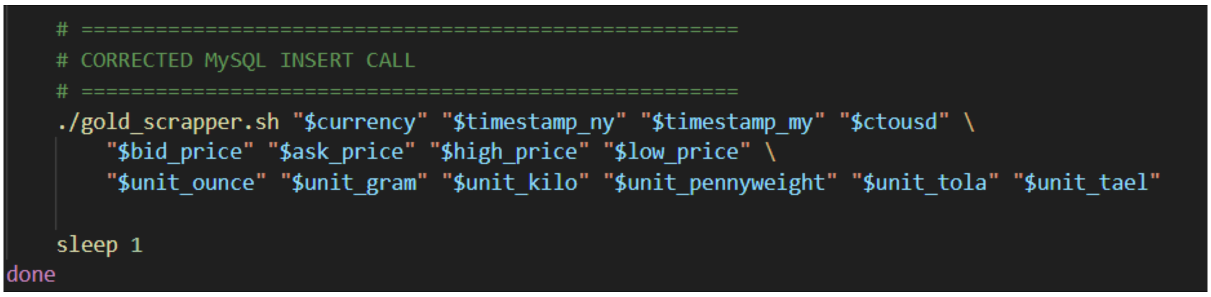


```
save_to_file() {
    local currency=$1
    local timestamp_ny=$2
    local timestamp_my=$3
    local ctousd=$4
    local bid_price=$5
    local ask_price=$6
    local high_price=$7
    local low_price=$8
    local unit_ounce=$9
    local unit_gram=${10}
    local unit_kilo=${11}
    local unit_pennyweight=${12}
    local unit_tola=${13}
    local unit_tael=${14}

    local file="$DATA_DIR/${currency}_gold_prices.csv"

    if [ ! -f "$file" ]; then
        echo "timestamp_ny,script_timestamp_my,currency,ctousd,bid_price,ask_price,high_price,low_price,
        unit_ounce,unit_gram,unit_kilo,unit_pennyweight,unit_tola,unit_tael" > "$file"
    fi

    echo "$timestamp_ny,$timestamp_my,$currency,$ctousd,$bid_price,$ask_price,$high_price,$low_price,
    $unit_ounce,$unit_gram,$unit_kilo,$unit_pennyweight,$unit_tola,$unit_tael" >> "$file"
}

# ---------------------------
```

Figure 23: Code to save data into CSV files.

4.2.7 Insert data into MySQL

After validation checks, the cleaned data is passed to gold_scrapper.sh as arguments. This script inserts the data into the database using the predefined schema.



```
    # ====================================================
    # CORRECTED MySQL INSERT CALL
    # ====================================================
    ./gold_scrapper.sh "$currency" "$timestamp_ny" "$timestamp_my" "$ctousd" \
        "$bid_price" "$ask_price" "$high_price" "$low_price" \
        "$unit_ounce" "$unit_gram" "$unit_kilo" "$unit_pennyweight" "$unit_tola" "$unit_tael"

    sleep 1
done
```

Figure 24: Code to insert data into MySQL.

4.2.8 Repeat for each currency

The script loops through all supported currencies (USD, EUR, GBP, AUD, and CNY). A short delay (sleep 1) is added between iterations to avoid overloading the source website.

5.0 DATABASE DESIGN, ERD EXPLANATION AND SQL SCHEMA

A relational database was designed to support the storage, organisation and retrieval of historical gold price data collected by the tracking system. The database design follows standard ERD and is fully normalised to minimise data redundancy, maintain data integrity and clearly represent relationships between entities. It allows gold price records, currency information, unit conversions, and system logs to be stored in a structured and consistent manner. All tables and relationships are designed to directly reflect the data generated by the Bash tracking and scraping scripts, ensuring seamless integration between the application logic and the database schema.

5.1 ERD and relationships

The ERD shows the core structure of the Gold Price Tracker database and how data flows between entities. The design separates currency details, gold price records, unit conversion values and system logs into distinct tables to ensure clear responsibilities and reduce data duplication. One currency can be associated with multiple gold price records; each gold price record can generate a single set of unit conversion values and system logs can reference both the currency and the related gold price entry. Primary and foreign key relationships are used throughout the design to enforce referential integrity and maintain consistent historical records.

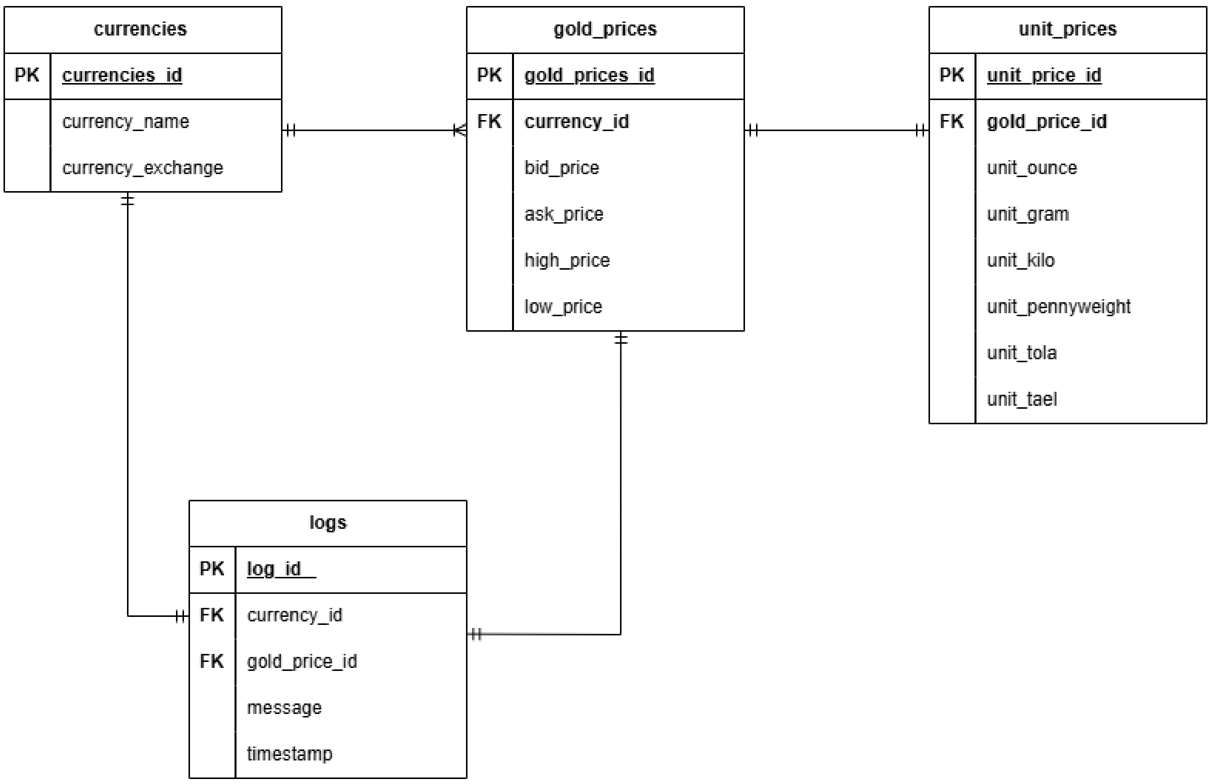


Figure 25: ERD Diagram for the Gold Price Tracker System.

5.2 Table Description

This section describes each table in the database and explains its purpose, key attributes and relationships within the system. Each table is designed with a specific responsibility to support accurate storage of gold price data, currency information, unit conversions, and system logs. These tables implement the ERD structure and ensure data consistency, integrity, and efficient querying for historical analysis.

5.2.1 Table: currencies

Purpose: Stores supported currencies and their exchange rates relative to USD.

Key Fields:

- **currencies_id (Primary Key):** A unique identifier for each currency record.
- **currency_name:** The currency code (e.g. USD, EUR, GBP, AUD, CNY).
- **currency_exchange:** The exchange rate used to convert gold prices from USD to the target currency.

Relationship:

- One currency can be associated with many gold price records.
- This forms a one-to-many (1:M) relationship between currencies and gold_prices.

5.2.2 Table: gold_prices

Purpose: Stores gold price snapshots for each currency at a specific run

Key Fields:

- **gold_prices_id (Primary Key):** Unique identifier for each gold price record.
- **currency_id (Foreign Key → currencies):** Links the gold price record to its corresponding currency.
- **price_timestamp_ny:** The market timestamp based on New York trading time.
- **script_timestamp_my:** The timestamp representing when the script was executed in Malaysia.
- **bid_price, ask_price, high_price, low_price:** Market price values collected from the source.

Relationship:

- Each gold price record belongs to one currency.
- Each gold price record can be linked to one set of unit prices.

5.2.3 Table: unit_prices

Purpose: Stores unit conversion prices (ounce, gram, kilo, pennyweight, tola, tael) for a specific gold_prices record

Key Fields:

- **unit_prices_id (Primary Key)**
- **gold_price_id (Foreign Key → gold_prices)**
- **unit_ounce, unit_gram, unit_kilo**
- **unit_pennyweight, unit_tola, unit_tael**

Relationship:

- Each unit price record corresponds to exactly one gold price record.
- This forms a one-to-one (1:1) relationship with the gold_prices table.

5.2.3 Table: logs

Purpose: Provides centralised system logging for debugging and audit trails

Key Fields:

- **log_id (Primary Key)**
- **currency_id (Foreign Key, nullable)**
- **gold_price_id (Foreign Key, nullable)**
- **message:** A descriptive log message indicating system actions or errors.
- **timestamp:** Automatically generated log timestamp.

Relationship:

- Log entries may reference both currency and gold price records.
- Foreign keys are nullable to ensure logs are preserved even if related data is deleted.

5.3 Normalisation

To ensure efficient data storage, reduce redundancy, and maintain data integrity, the database used in the **Gold Price Tracker System** was designed according to standard database normalisation principles. The final database structure is normalised up to **Third Normal Form (3NF)**.

The database consists of four main tables:

- currencies
- gold_prices
- unit_prices
- logs

Each table has a clear and distinct responsibility, and relationships between tables are defined using primary and foreign keys.

5.3.1 First Normal Form (1NF)

First Normal Form requires that:

- All table attributes contain values
- There are **no repeating groups or multi-valued attributes**
- Each record can be uniquely identified by a **primary key**

In this database:

- Each table has a primary key (currencies_id, gold_prices_id, unit_prices_id, log_id)
- All attributes store single, indivisible values (e.g., bid_price, ask_price, unit_gram)
- Repeating data, such as gold price unit conversions, is stored in a separate table (unit_prices) rather than repeated within the gold_prices table

Therefore, all tables satisfy the requirements of **First Normal Form (1NF)**.

5.3.2 Second Normal Form (2NF)

Second Normal Form requires that:

- The database is already in **1NF**
- All non-key attributes depend on the **entire primary key**
- There are **no partial dependencies**

In this design:

- All tables use a **single-column primary key**
- There are no composite primary keys
- Every non-key attribute is fully dependent on its table's primary key

For example:

- In the gold_prices table, price values (bid_price, ask_price, high_price, low_price) depend entirely on gold_prices _id
- In the unit_prices table, all unit conversion values depend entirely on unit_prices_id

As a result, the database fully satisfies **Second Normal Form (2NF)**.

5.3.3 Third Normal Form (3NF)

Third Normal Form requires that:

- The database is in **2NF**
- There are **no transitive dependencies**
- Non-key attributes do not depend on other non-key attributes

This requirement is satisfied through the separation of concerns across tables:

- Currency-related information (currency_name, currency_exchange) is stored only in the currencies table
- Gold price snapshot data is stored in the gold_prices table and references currencies using a foreign key (currency_id)
- Unit conversion values are stored in the unit_prices table and reference a specific gold price record using gold_price_id

- Logging information is stored separately in the logs table to prevent duplication of log messages within operational tables

No non-key attribute depends on another non-key attribute, and all dependencies are based solely on primary keys.

Hence, the database is normalised to **Third Normal Form (3NF)**.

5.4 Schema Implementation

This section explains **how I turned my ERD diagram into real MySQL tables**, and how I enforced relationships and data integrity using **primary keys, foreign keys and constraints**.

Step 1: Create a clean database

Before creating any tables, I reset the database so my testing would always start from a clean state (no old tables/data causing errors). I did this using:

- DROP DATABASE IF EXISTS goldtracker;
- CREATE DATABASE goldtracker;
- USE goldtracker;

This ensures the schema is always reproducible from scratch.

Step 2: Create tables in the correct order (to avoid foreign key errors)

Because **foreign keys depend on parent tables**, I created the tables in this order:

1. currencies
2. gold_prices (depends on currencies)
3. unit_prices (depends on gold_prices)
4. logs (depends on currencies and gold_prices)

If I created currencies tables first, MySQL would throw an error because the referenced table does not exist yet.

Step 3: Implement primary keys (unique identity for every row)

For each table, I added an **AUTO_INCREMENT primary key** so every record has a unique identifier:

- currencies.currencies_id

- gold_prices.gold_prices_id
- unit_prices.unit_prices.id
- logs.log_id

Step 4: Implement a unique constraint for currencies (prevents duplicates)

In currencies, I set:

- currency_name VARCHAR(20) NOT NULL UNIQUE

This guarantees there is only one USD, one EUR and others.

Why I did this:

- My tracker runs repeatedly crontab, so without UNIQUE, I could accidentally insert the same currency many times.
- This also enables ON DUPLICATE KEY UPDATE to work properly (because it triggers on the UNIQUE key).

Step 5: Implement foreign keys (enforce ERD relationships)

To enforce the relationships exactly as in the ERD, I used foreign keys:

(A) gold_prices → currencies (1 currency to many gold snapshots)

- currency_id references currencies(currencies_id)

This ensures every gold price record **must belong to a valid currency** and cannot insert a gold price row with a currency that doesn't exist

(B) unit_prices → gold_prices (1 gold snapshot to 1 unit conversion row)

- gold_price_id references gold_prices(gold_prices_id)

This ensures the unit conversions are always linked to a real gold price snapshot.

Step 6: Add referential integrity rules for logs

For the logs table, I allowed the IDs to be nullable and set safe rules:

- ON UPDATE CASCADE (if ID changes, logs update automatically)
- ON DELETE SET NULL (if row is deleted, logs keep the message but remove the broken reference)

Step 7: Use ON DUPLICATE KEY UPDATE in my scripts (prevents duplicate currencies)

After the schema was ready, I applied this design in my **scraper/tracker insertion logic**:

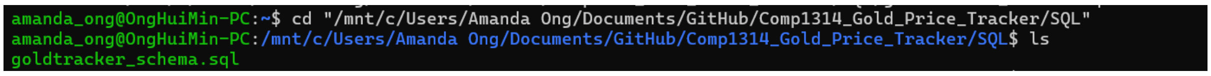
- When inserting into currencies, I used ON DUPLICATE KEY UPDATE so:
    - If the currency already exists (same currency_name), it won't create a duplicate row
    - Instead, it updates fields like currency_exchange (if needed)

This works **because** currency_name is UNIQUE.

5.5 MySQL Installation, Schema Execution and Data Verification

After completing the database design and SQL schema, MySQL was installed and used in the WSL Ubuntu environment to implement the database physically. This step ensures that the design schema can be executed correctly and supports real data insertion from the tracking system.

First, MySQL was installed and accessed through the terminal. Once MySQL was running, I navigated to the project's SQL directory to locate the schema file that contains all table definitions and constraints.

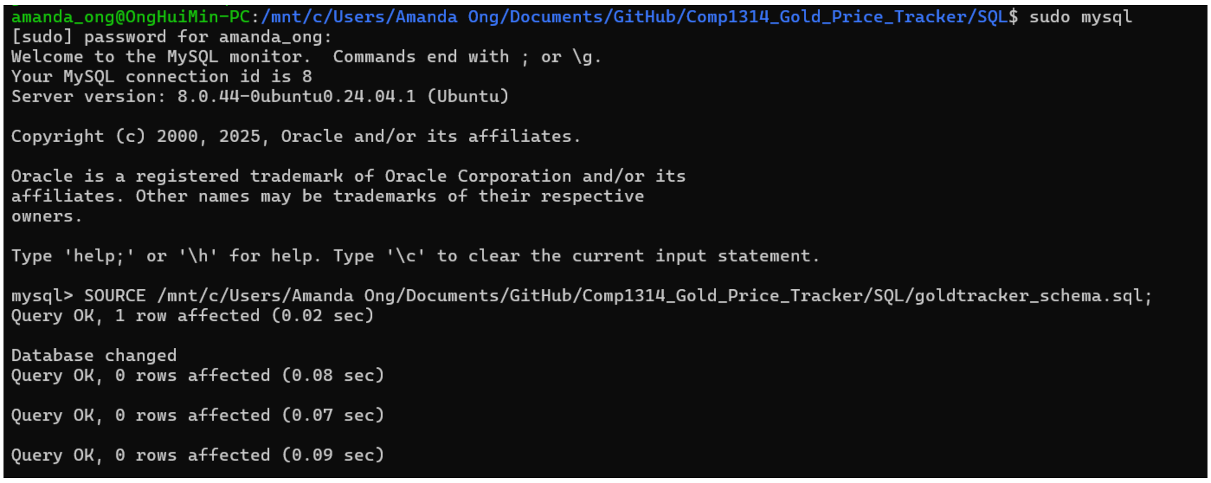


```
amanda_ong@OngHuiMin-PC:~$ cd "/mnt/c/Users/Amanda Ong/Documents/GitHub/Comp1314_Gold_Price_Tracker/SQL"
amanda_ong@OngHuiMin-PC:/mnt/c/Users/Amanda Ong/Documents/GitHub/Comp1314_Gold_Price_Tracker/SQL$ ls
goldtracker_schema.sql
```

Figure 26: Navigating to the SQL directory and locating the schema file.

This confirmed that the goldtracker_schema.sql file was present in the directory.

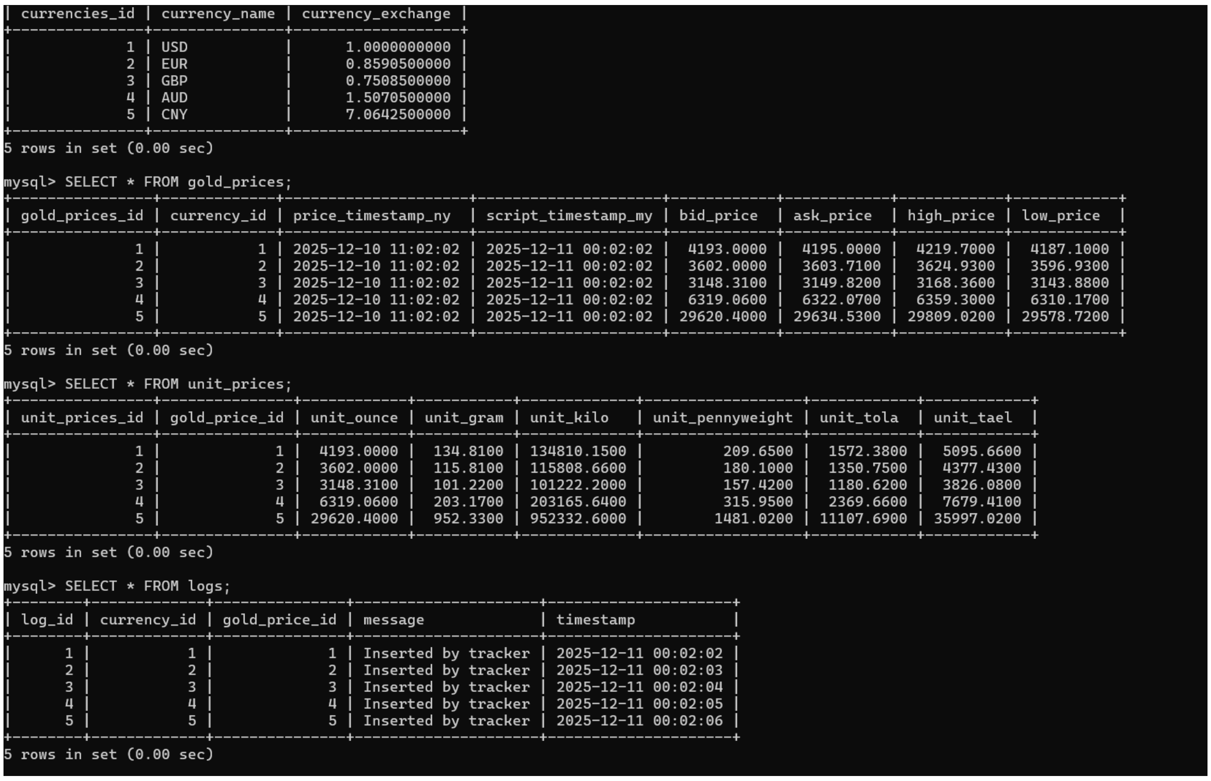
Next, I logged into the MySQL command-line interface using administrator privileges:



```
amanda_ong@OngHuiMin-PC:/mnt/c/Users/Amanda Ong/Documents/GitHub/Comp1314_Gold_Price_Tracker/SQL$ sudo mysql
[sudo] password for amanda_ong:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.44-0ubuntu0.24.04.1 (Ubuntu)

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SOURCE /mnt/c/Users/Amanda Ong/Documents/GitHub/Comp1314_Gold_Price_Tracker/SQL/goldtracker_schema.sql;
Query OK, 1 row affected (0.02 sec)

Database changed
Query OK, 0 rows affected (0.08 sec)

Query OK, 0 rows affected (0.07 sec)

Query OK, 0 rows affected (0.09 sec)
```

Figure 27: Executing the SQL schema file using the MySQL SOURCE command.

The system returned multiple Query OK messages, confirming that the database and tables were created successfully without errors. After the schema was applied, I verified that all tables were created correctly and that data was being inserted as expected. This was done using SELECT queries on each table.



```
| currencies_id | currency_name | currency_exchange |
+---------------+---------------+-------------------+
|             1 | USD           |      1.0000000000 |
|             2 | EUR           |      0.8590500000 |
|             3 | GBP           |      0.7508500000 |
|             4 | AUD           |      1.5070500000 |
|             5 | CNY           |      7.0642500000 |
+---------------+---------------+-------------------+
5 rows in set (0.00 sec)

mysql> SELECT * FROM gold_prices;
+----------------+-------------+---------------------+---------------------+------------+------------+------------+------------+
| gold_prices_id | currency_id | price_timestamp_ny  | script_timestamp_my | bid_price  | ask_price  | high_price | low_price  |
+----------------+-------------+---------------------+---------------------+------------+------------+------------+------------+
|              1 |           1 | 2025-12-10 11:02:02 | 2025-12-11 00:02:02 |  4193.0000 |  4195.0000 |  4219.7000 |  4187.1000 |
|              2 |           2 | 2025-12-10 11:02:02 | 2025-12-11 00:02:02 |  3602.0000 |  3603.7100 |  3624.9300 |  3596.9300 |
|              3 |           3 | 2025-12-10 11:02:02 | 2025-12-11 00:02:02 |  3148.3100 |  3149.8200 |  3168.3600 |  3143.8800 |
|              4 |           4 | 2025-12-10 11:02:02 | 2025-12-11 00:02:02 |  6319.0600 |  6322.0700 |  6359.3000 |  6310.1700 |
|              5 |           5 | 2025-12-10 11:02:02 | 2025-12-11 00:02:02 | 29620.4000 | 29634.5300 | 29809.0200 | 29578.7200 |
+----------------+-------------+---------------------+---------------------+------------+------------+------------+------------+
5 rows in set (0.00 sec)

mysql> SELECT * FROM unit_prices;
+----------------+---------------+------------+-----------+-------------+------------------+------------+------------+
| unit_prices_id | gold_price_id | unit_ounce | unit_gram | unit_kilo   | unit_pennyweight | unit_tola  | unit_tael  |
+----------------+---------------+------------+-----------+-------------+------------------+------------+------------+
|              1 |             1 |  4193.0000 |  134.8100 | 134810.1500 |         209.6500 |  1572.3800 |  5095.6600 |
|              2 |             2 |  3602.0000 |  115.8100 | 115808.6600 |         180.1000 |  1350.7500 |  4377.4300 |
|              3 |             3 |  3148.3100 |  101.2200 | 101222.2000 |         157.4200 |  1180.6200 |  3826.0800 |
|              4 |             4 |  6319.0600 |  203.1700 | 203165.6400 |         315.9500 |  2369.6600 |  7679.4100 |
|              5 |             5 | 29620.4000 |  952.3300 | 952332.6000 |        1481.0200 | 11107.6900 | 35997.0200 |
+----------------+---------------+------------+-----------+-------------+------------------+------------+------------+
5 rows in set (0.00 sec)

mysql> SELECT * FROM logs;
+--------+-------------+---------------+---------------------+---------------------+
| log_id | currency_id | gold_price_id | message             | timestamp           |
+--------+-------------+---------------+---------------------+---------------------+
|      1 |           1 |             1 | Inserted by tracker | 2025-12-11 00:02:02 |
|      2 |           2 |             2 | Inserted by tracker | 2025-12-11 00:02:03 |
|      3 |           3 |             3 | Inserted by tracker | 2025-12-11 00:02:04 |
|      4 |           4 |             4 | Inserted by tracker | 2025-12-11 00:02:05 |
|      5 |           5 |             5 | Inserted by tracker | 2025-12-11 00:02:06 |
+--------+-------------+---------------+---------------------+---------------------+
5 rows in set (0.00 sec)
```

Figure 28: Verifying data insertion in all database tables.

The query results showed valid records in all tables, including currency information, gold price snapshots, unit conversion values, and system log entries. This confirms that the schema implementation works correctly and integrates seamlessly with the Bash tracking and scraping scripts.

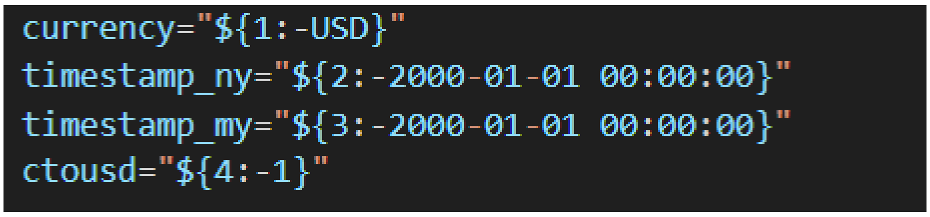
6.0 UNIX SCRIPT FOR SCRAPPER

This section explains the MySQL insertion script used in the Gold Price Tracker system. The gold_scrapper.sh script receives cleaned and validated values from gold_tracker.sh as command-line arguments. Its main responsibility is to safely insert data into the MySQL database while maintaining correct foreign key relationships and protecting data integrity. The script ensures that data is inserted in the correct order so that all relational constraints are satisfied.

6.1 Step-by-Step Insertion Sequence

Step 1: Read input arguments safely

The script reads all required values (currency, timestamps, conversion rate, prices, and unit values) from command-line arguments. Default values are provided to prevent script crashes if any argument is missing.

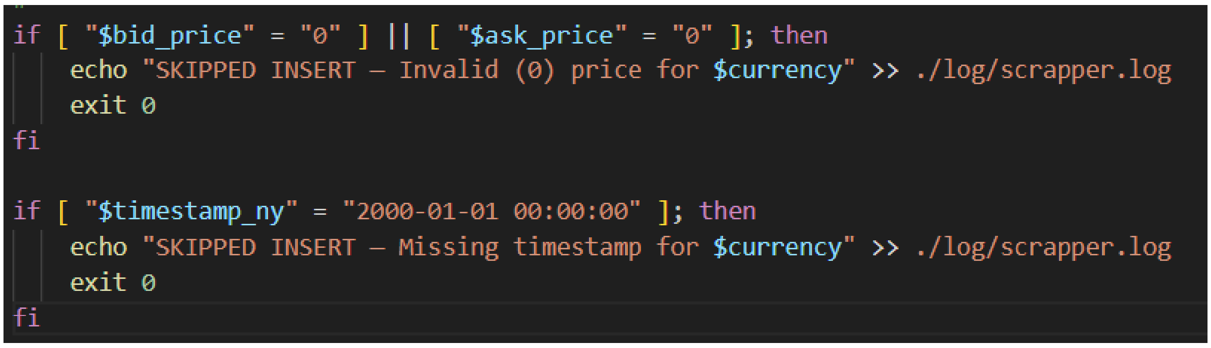


```
currency="${1:-USD}"
timestamp_ny="${2:-2000-01-01 00:00:00}"
timestamp_my="${3:-2000-01-01 00:00:00}"
ctousd="${4:-1}"
```

Figure 29: Code for reading input.

Step 2: Validate key values before insertion

Before inserting any data, the script checks for invalid values such as missing timestamps or bid/ask prices equal to 0. If invalid data is detected, the insertion is skipped, and a log message is written.

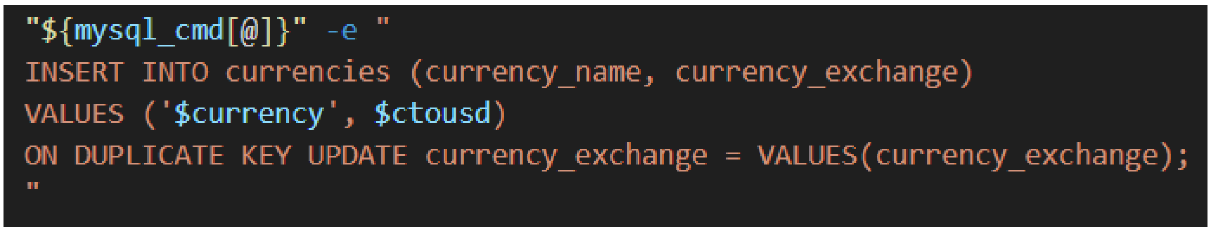


```
if [ "$bid_price" = "0" ] || [ "$ask_price" = "0" ]; then
    echo "SKIPPED INSERT – Invalid (0) price for $currency" >> ./log/scrapper.log
    exit 0
fi

if [ "$timestamp_ny" = "2000-01-01 00:00:00" ]; then
    echo "SKIPPED INSERT – Missing timestamp for $currency" >> ./log/scrapper.log
    exit 0
fi
```

Figure 30: Code for validating the key before insertion.

Step 3: Insert or update currency information

The script inserts the currency into the currencies table. If the currency already exists, the exchange rate is updated using ON DUPLICATE KEY UPDATE. This prevents duplicate currency records.

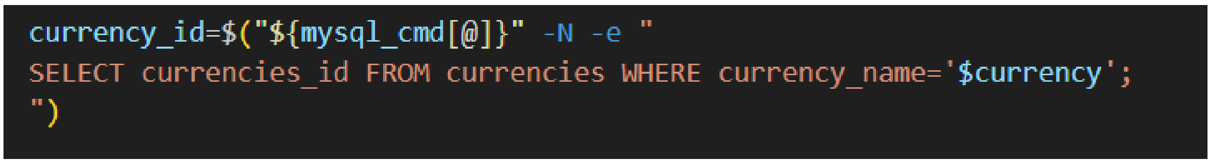


```
"${mysql_cmd[@]}" -e "
INSERT INTO currencies (currency_name, currency_exchange)
VALUES ('$currency', $ctousd)
ON DUPLICATE KEY UPDATE currency_exchange = VALUES(currency_exchange);
"
```

Figure 31: Code for inserting/updating currency information.

Step 4: Retrieve the currency_id

After inserting or updating the currency, the script queries the database to obtain the corresponding currency_id. This ID is required for foreign key relationships.

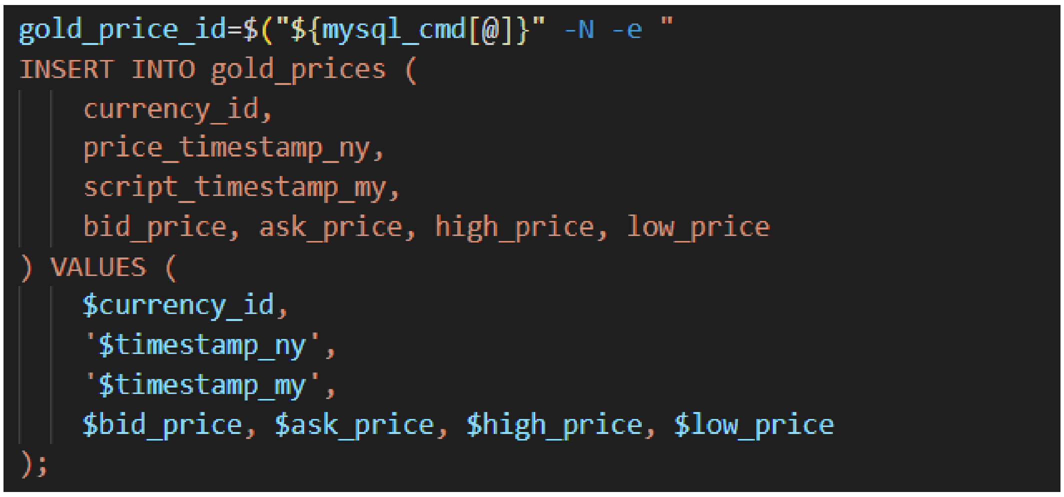


```
currency_id=$("${mysql_cmd[@]}" -N -e "
SELECT currencies_id FROM currencies WHERE currency_name='$currency';
")
```

Figure 32: Code for retrieving currency information.

Step 5: Insert a new gold price record

A new row is inserted into the gold_prices table using the retrieved currency_id and the main price values.

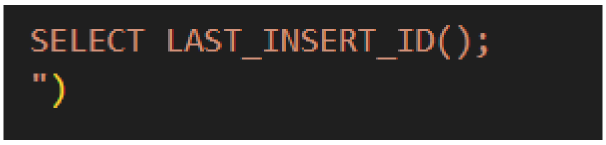


```
gold_price_id=$("${mysql_cmd[@]}" -N -e "
INSERT INTO gold_prices (
    currency_id,
    price_timestamp_ny,
    script_timestamp_my,
    bid_price, ask_price, high_price, low_price
) VALUES (
    $currency_id,
    '$timestamp_ny',
    '$timestamp_my',
    $bid_price, $ask_price, $high_price, $low_price
);
```

Figure 31: Code for inserting new gold price record.

Step 6: Capture the generated gold_prices_id

The script uses LAST_INSERT_ID() to capture the newly created gold_prices_id. This ID is required to link unit prices correctly.

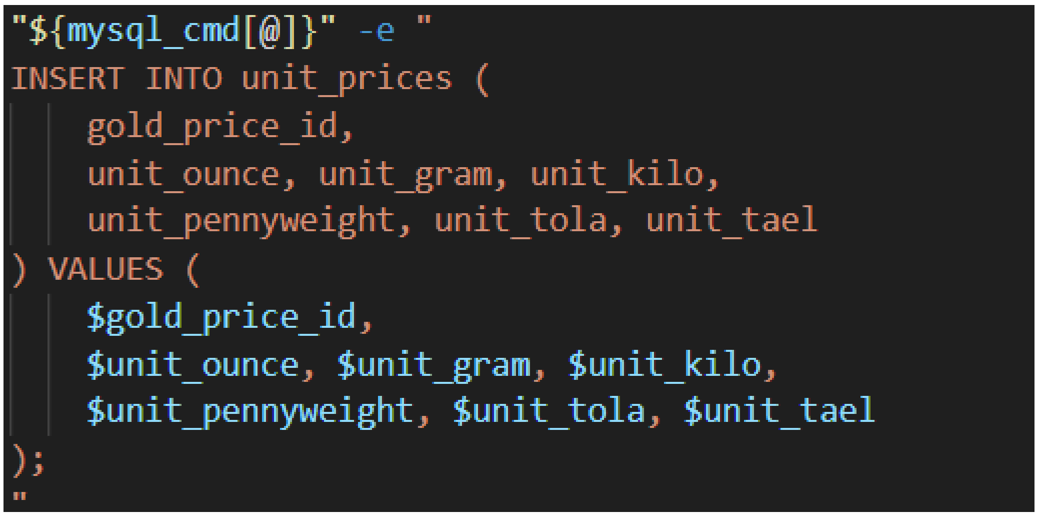


```
SELECT LAST_INSERT_ID();
")
```

Figure 32: Code for capturing the generated gold_prices_id.

Step 7: Insert unit conversion values

The unit conversion values are inserted into the unit_prices table using the captured gold_prices_id as a foreign key.

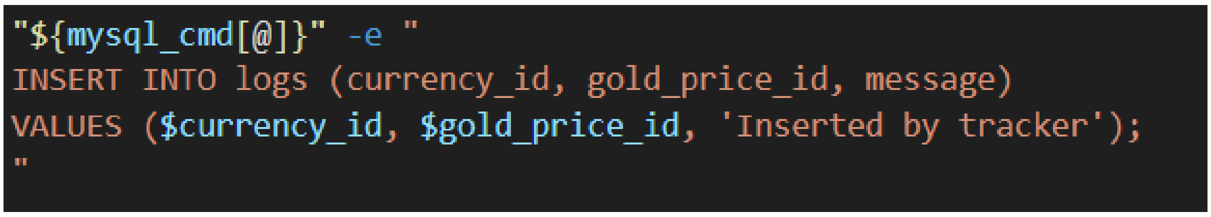


```
"${mysql_cmd[@]}" -e "
INSERT INTO unit_prices (
    gold_price_id,
    unit_ounce, unit_gram, unit_kilo,
    unit_pennyweight, unit_tola, unit_tael
) VALUES (
    $gold_price_id,
    $unit_ounce, $unit_gram, $unit_kilo,
    $unit_pennyweight, $unit_tola, $unit_tael
);
"
```

Figure 33: Code for inserting unit conversion values.

Step 8: Insert a system log record

Finally, a record is inserted into the logs table to confirm that the insertion was successful. This is useful for debugging and cron verification.



```
"${mysql_cmd[@]}" -e "
INSERT INTO logs (currency_id, gold_price_id, message)
VALUES ($currency_id, $gold_price_id, 'Inserted by tracker');
"
```

Figure 34: Code for inserting a system log record.

7.0 UNIX SCRIPT FOR PLOTTING

This section describes how UNIX scripting and GNUPlot were used to visualise historical gold price data. After data has been collected and stored in the MySQL database through multiple runs, the plotting script extracts the data and generates time-series graphs. These help users to understand price trends over time.

The plotting process is fully automated using a Bash script (plot_gold.sh). The script dynamically generates SQL queries for each currency, retrieves historical data from the database, and passes the results to GNUPlot. GNUPlot then produces labelled PNG graphs with timestamps on the x-axis and price values on the y-axis. All plots are saved automatically without manual intervention.

7.1 Installation of GNUPlot

Before generating visual plots, GNUPlot was installed in the WSL Ubuntu environment. GNUPlot is required to generate time-series graphs from the extracted gold price data. The installation was done using the system package manager.

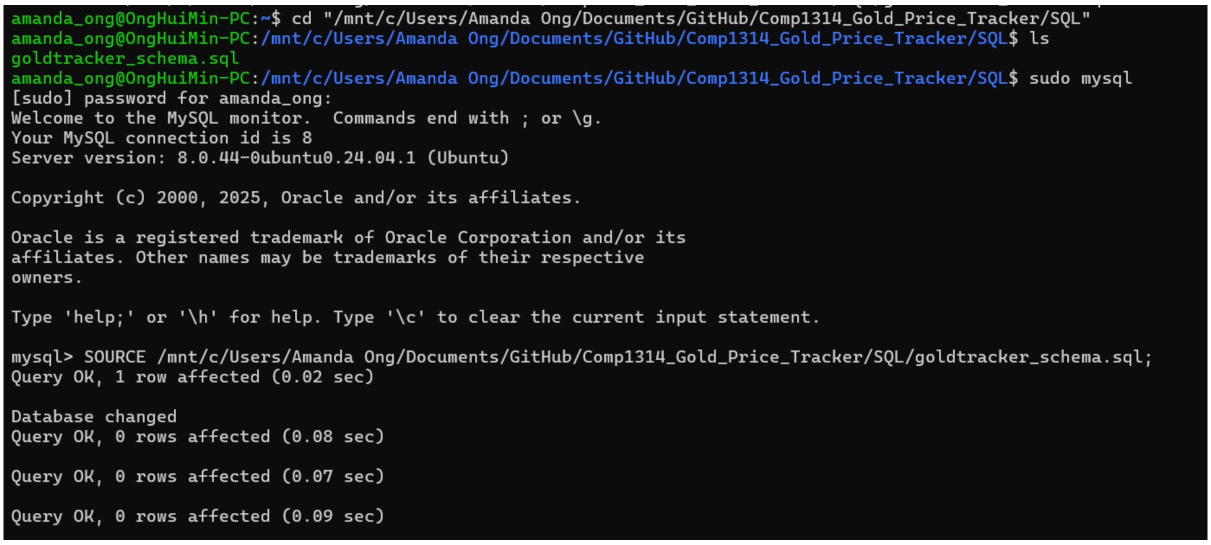
First, the package list was updated to ensure the latest software versions were available:

- sudo apt update

Next, GNUPlot was installed using the following command:

- sudo apt install gnuplot -y

The -y flag was used to automatically confirm the installation without manual input. After installation, GNUPlot became available for use by the plotting script (plot_gold.sh).



```
amanda_ong@OngHuiMin-PC:~$ cd "/mnt/c/Users/Amanda Ong/Documents/GitHub/Comp1314_Gold_Price_Tracker/SQL"
amanda_ong@OngHuiMin-PC:/mnt/c/Users/Amanda Ong/Documents/GitHub/Comp1314_Gold_Price_Tracker/SQL$ ls
goldtracker_schema.sql
amanda_ong@OngHuiMin-PC:/mnt/c/Users/Amanda Ong/Documents/GitHub/Comp1314_Gold_Price_Tracker/SQL$ sudo mysql
[sudo] password for amanda_ong:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.44-0ubuntu0.24.04.1 (Ubuntu)

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SOURCE /mnt/c/Users/Amanda Ong/Documents/GitHub/Comp1314_Gold_Price_Tracker/SQL/goldtracker_schema.sql;
Query OK, 1 row affected (0.02 sec)

Database changed
Query OK, 0 rows affected (0.08 sec)

Query OK, 0 rows affected (0.07 sec)

Query OK, 0 rows affected (0.09 sec)
```

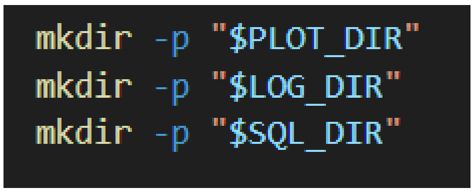
Figure 35: Installing GNUPlot using the Ubuntu system package manager.

Once GNUPlot was installed, the plotting script was able to generate PNG graphs automatically from the database query results. This completed the setup required for UNIX-based data visualisation.

7.2 Step-by-Step Plotting Workflow

Step 1: Ensure required directories exist

The script first creates the plots/, log/, and sql_queries/ directories if they do not already exist. This prevents file-not-found errors.

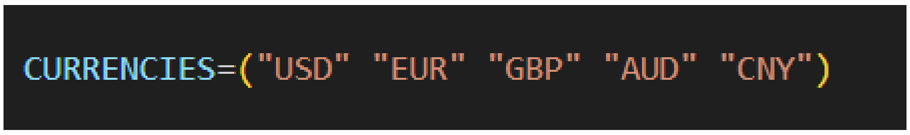


```
mkdir -p "$PLOT_DIR"
mkdir -p "$LOG_DIR"
mkdir -p "$SQL_DIR"
```

Figure 36: Code for the directory.

Step 2: Define supported currencies

A list of supported currencies is defined in an array. This allows the script to loop through each currency automatically.

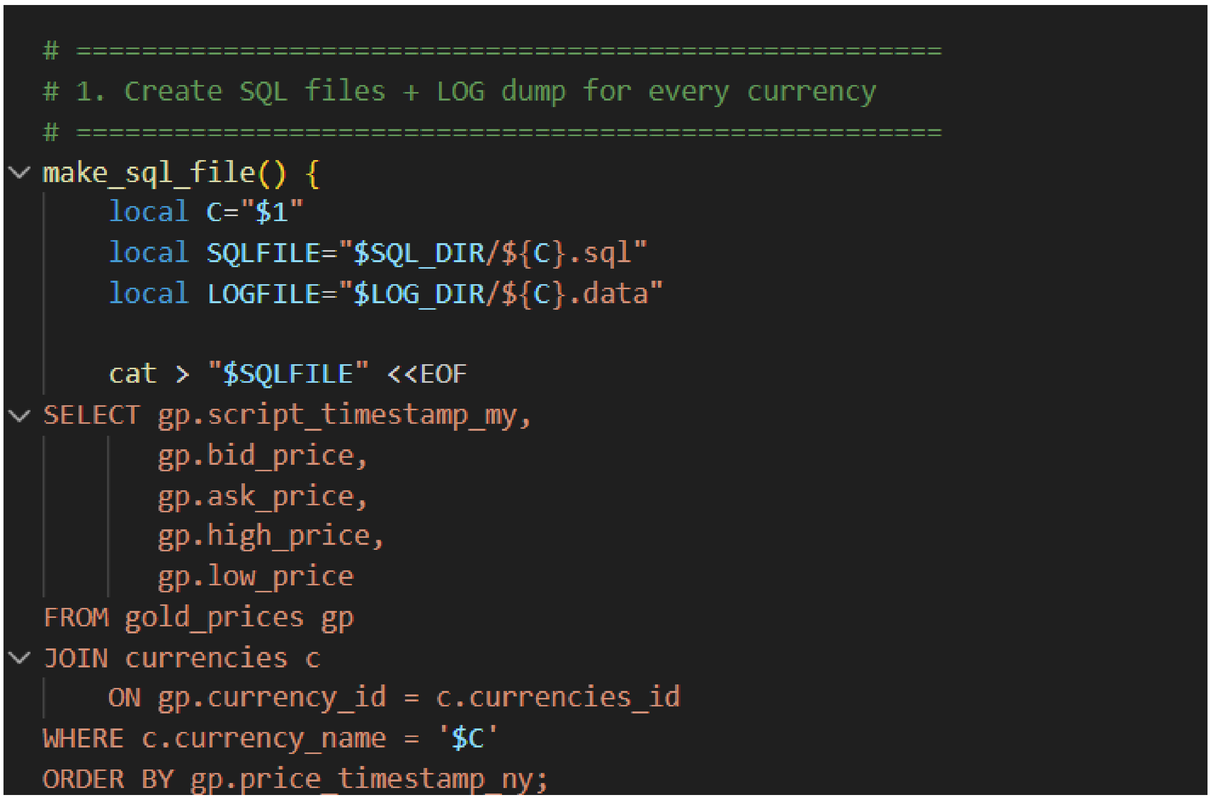


```
CURRENCIES=("USD" "EUR" "GBP" "AUD" "CNY")
```

Figure 37: Code for define supported currencies.

Step 3: Generate SQL query files for each currency

For each currency, a SQL file is generated to select historical price data from the database. This includes timestamp, bid, ask, high, and low prices.

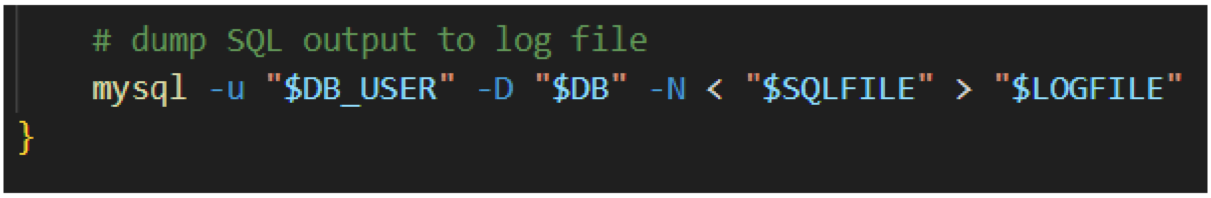


```
# ======================================================
# 1. Create SQL files + LOG dump for every currency
# ======================================================
make_sql_file() {
    local C="$1"
    local SQLFILE="$SQL_DIR/${C}.sql"
    local LOGFILE="$LOG_DIR/${C}.data"

    cat > "$SQLFILE" <<EOF
SELECT gp.script_timestamp_my,
       gp.bid_price,
       gp.ask_price,
       gp.high_price,
       gp.low_price
FROM gold_prices gp
JOIN currencies c
    ON gp.currency_id = c.currencies_id
WHERE c.currency_name = '$C'
ORDER BY gp.price_timestamp_ny;
```

Figure 38: Code for generating SQL query files for each currency.

Step 4: Dump query output into data files

The SQL query output is saved into a .data file. This file acts as the input source for GNUPlot.



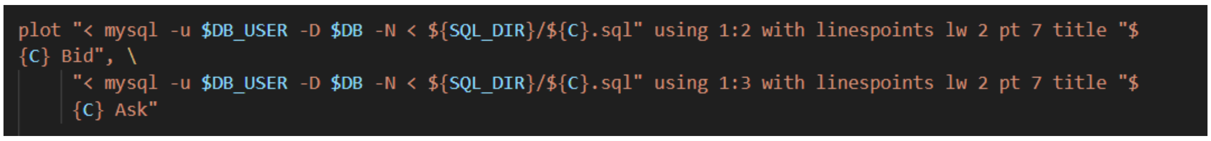
```
    # dump SQL output to log file
    mysql -u "$DB_USER" -D "$DB" -N < "$SQLFILE" > "$LOGFILE"
}
```

Figure 39: Code for Dump query output into data files.

Step 5: Generate plots using GNUPlot

GNUPlot reads the query output and generates PNG graphs. Time is displayed on the x-axis and prices on the y-axis. Two plots are created per currency.

Example (Bid vs Ask):



```
plot "< mysql -u $DB_USER -D $DB -N < ${SQL_DIR}/${C}.sql" using 1:2 with linespoints lw 2 pt 7 title "$
{C} Bid", \
    "< mysql -u $DB_USER -D $DB -N < ${SQL_DIR}/${C}.sql" using 1:3 with linespoints lw 2 pt 7 title "$
    {C} Ask"
```

Figure 40: Code for generating plots using GNUPlot.

Step 6: Repeat for all currencies

The script loops through all currencies and repeats the process automatically. This results in at least ten plots being generated in total.

7.3 Handling Flat Price Data During Market Closure (Weekends)

During the analysis of the generated plots, it was observed that in some time periods the gold price lines appear almost completely flat over an extended duration. This behaviour is not caused by an error in the data collection or plotting scripts, but instead reflects **real-world market conditions**.

Gold markets are typically **closed during weekends (Saturday and Sunday)**. When the market is closed, no new price updates are published by the data source. As a result, repeated executions of the tracker script during these periods record **identical price values** across multiple timestamps. When plotted, these identical values produce a flat horizontal line on the graph.

This phenomenon is expected and indicates that:

- The cron job is running correctly at the scheduled intervals
- The data extraction and insertion scripts are functioning as intended
- The plotted data accurately reflects market availability rather than system failure

7.3.1 Detection of Flat Data Patterns in Plots

Flat patterns in the plots can be identified by the following characteristics:

- Bid, ask, high, and low prices remain unchanged for several consecutive timestamps
- The flat region usually spans from late Friday through Sunday
- The flat section ends once the market reopens on Monday

This behaviour can be clearly observed in the generated time-series plots, particularly for currencies such as USD and EUR.

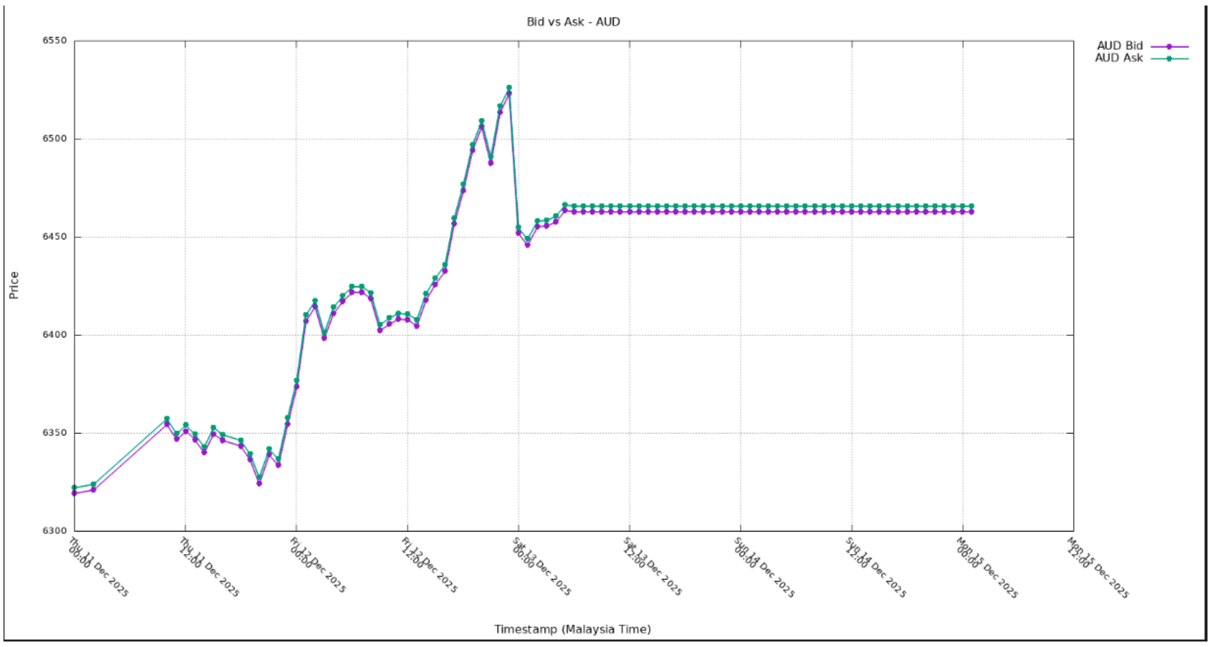


Figure 41: Example plot showing flat weekend section.

7.3.2 Design Consideration and Interpretation

The plotting script intentionally does **not remove or modify weekend data**, as doing so would distort the true historical timeline. Instead, all recorded data points are plotted to preserve data integrity and ensure transparency.

By retaining flat weekend data:

- The plots remain chronologically accurate
- Users can visually distinguish between active trading periods and market closures
- The system demonstrates correct handling of real-world data constraints

This design choice aligns with good data management practices, where collected data is presented as-is and interpreted correctly rather than artificially altered.

8.0 CRONTAB AUTOMATION

Crontab was used to automate the Gold Price Tracker system so that it can run without manual execution. Automation allows the system to collect data, process it and generate plots automatically every hour. This ensures consistent data collection and reduces human error.

8.1 Installation and Enabling Cron (WSL Ubuntu)

First, crontab was installed and enabled in the WSL Ubuntu environment. The package list was updated, and the crontab was installed using the system package manager.

Step 1: Update the package list

Updated the package list to make sure the latest versions can be installed.

- sudo apt update

Step 2: Install cron

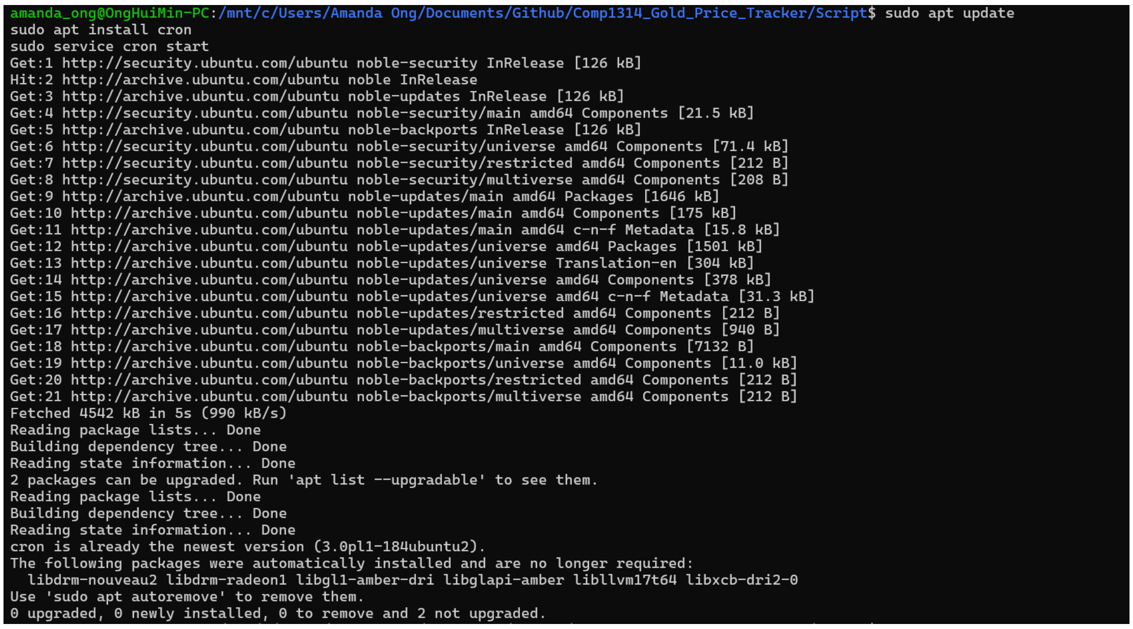
Then I installed the cron service.

- sudo apt install cron

Step 3: Start the cron service

After installation, I started cron so it can run scheduled jobs.

- sudo service cron start



```
amanda_ong@OngHuiMin-PC:/mnt/c/Users/Amanda Ong/Documents/Github/Comp1314_Gold_Price_Tracker/Script$ sudo apt update
sudo apt install cron
sudo service cron start
Get:1 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Hit:2 http://archive.ubuntu.com/ubuntu noble InRelease
Get:3 http://archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Get:4 http://security.ubuntu.com/ubuntu noble-security/main amd64 Components [21.5 kB]
Get:5 http://archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
Get:6 http://security.ubuntu.com/ubuntu noble-security/universe amd64 Components [71.4 kB]
Get:7 http://security.ubuntu.com/ubuntu noble-security/restricted amd64 Components [212 B]
Get:8 http://security.ubuntu.com/ubuntu noble-security/multiverse amd64 Components [208 B]
Get:9 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 Packages [1646 kB]
Get:10 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 Components [175 kB]
Get:11 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 c-n-f Metadata [15.8 kB]
Get:12 http://archive.ubuntu.com/ubuntu noble-updates/universe amd64 Packages [1501 kB]
Get:13 http://archive.ubuntu.com/ubuntu noble-updates/universe Translation-en [304 kB]
Get:14 http://archive.ubuntu.com/ubuntu noble-updates/universe amd64 Components [378 kB]
Get:15 http://archive.ubuntu.com/ubuntu noble-updates/universe amd64 c-n-f Metadata [31.3 kB]
Get:16 http://archive.ubuntu.com/ubuntu noble-updates/restricted amd64 Components [212 B]
Get:17 http://archive.ubuntu.com/ubuntu noble-updates/multiverse amd64 Components [940 B]
Get:18 http://archive.ubuntu.com/ubuntu noble-backports/main amd64 Components [7132 B]
Get:19 http://archive.ubuntu.com/ubuntu noble-backports/universe amd64 Components [11.0 kB]
Get:20 http://archive.ubuntu.com/ubuntu noble-backports/restricted amd64 Components [212 B]
Get:21 http://archive.ubuntu.com/ubuntu noble-backports/multiverse amd64 Components [212 B]
Fetched 4542 kB in 5s (990 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
2 packages can be upgraded. Run 'apt list --upgradable' to see them.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
cron is already the newest version (3.0pl1-184ubuntu2).
The following packages were automatically installed and are no longer required:
  libdrm-nouveau2 libdrm-radeon1 libgl1-amber-dri libglapi-amber libllvm17t64 libxcb-dri2-0
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 2 not upgraded.
```

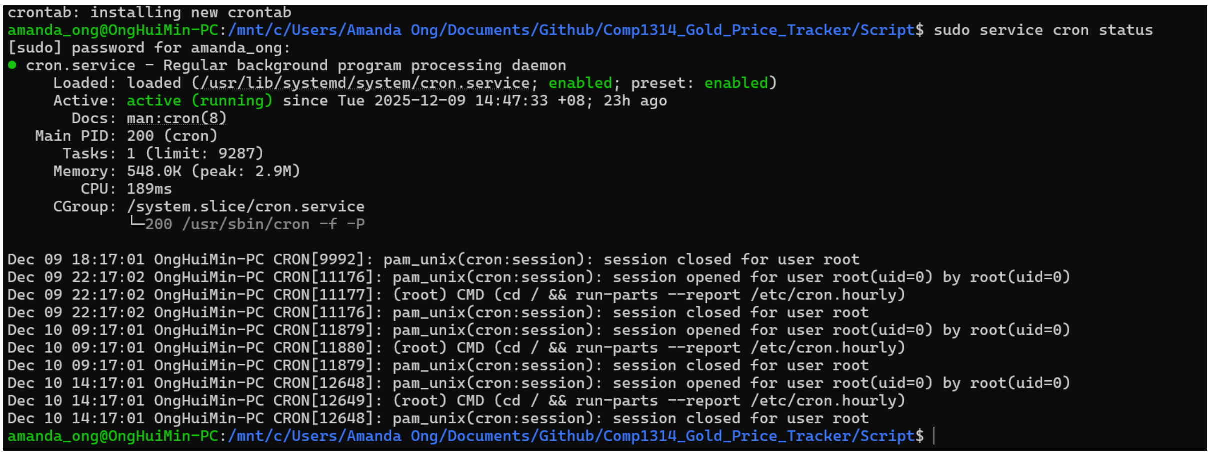
Figure 42: Downloading the Crontab in the system package manager.

Step 4: Confirm cron is running

I checked the status to confirm the service is active in the background.

- sudo service cron status

If the status shows **active (running)**, it means cron is ready to execute tasks automatically.



```
crontab: installing new crontab
amanda_ong@OngHuiMin-PC:/mnt/c/Users/Amanda Ong/Documents/Github/Comp1314_Gold_Price_Tracker/Script$ sudo service cron status
[sudo] password for amanda_ong:
● cron.service - Regular background program processing daemon
     Loaded: loaded (/usr/lib/systemd/system/cron.service; enabled; preset: enabled)
     Active: active (running) since Tue 2025-12-09 14:47:33 +08; 23h ago
       Docs: man:cron(8)
   Main PID: 200 (cron)
      Tasks: 1 (limit: 9287)
     Memory: 548.0K (peak: 2.9M)
        CPU: 189ms
     CGroup: /system.slice/cron.service
             └─200 /usr/sbin/cron -f -P

Dec 09 18:17:01 OngHuiMin-PC CRON[9992]: pam_unix(cron:session): session closed for user root
Dec 09 22:17:02 OngHuiMin-PC CRON[11176]: pam_unix(cron:session): session opened for user root(uid=0) by root(uid=0)
Dec 09 22:17:02 OngHuiMin-PC CRON[11177]: (root) CMD (cd / && run-parts --report /etc/cron.hourly)
Dec 09 22:17:02 OngHuiMin-PC CRON[11176]: pam_unix(cron:session): session closed for user root
Dec 10 09:17:01 OngHuiMin-PC CRON[11879]: pam_unix(cron:session): session opened for user root(uid=0) by root(uid=0)
Dec 10 09:17:01 OngHuiMin-PC CRON[11880]: (root) CMD (cd / && run-parts --report /etc/cron.hourly)
Dec 10 09:17:01 OngHuiMin-PC CRON[11879]: pam_unix(cron:session): session closed for user root
Dec 10 14:17:01 OngHuiMin-PC CRON[12648]: pam_unix(cron:session): session opened for user root(uid=0) by root(uid=0)
Dec 10 14:17:01 OngHuiMin-PC CRON[12649]: (root) CMD (cd / && run-parts --report /etc/cron.hourly)
Dec 10 14:17:01 OngHuiMin-PC CRON[12648]: pam_unix(cron:session): session closed for user root
amanda_ong@OngHuiMin-PC:/mnt/c/Users/Amanda Ong/Documents/Github/Comp1314_Gold_Price_Tracker/Script$
```

Figure 43: Checking cron service status.

8.2 Creating the User Crontab File

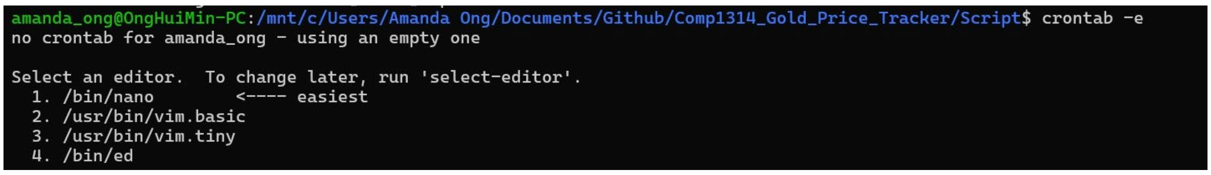
Step 1: Open the crontab editor

To add scheduled jobs, use:

- crontab -e

Step 2: Select an editor (first-time setup)

Since this was my first time creating a crontab, the editor selection menu appeared. I selected **/bin/nano** because it is simple to use.



```
amanda_ong@OngHuiMin-PC:/mnt/c/Users/Amanda Ong/Documents/Github/Comp1314_Gold_Price_Tracker/Script$ crontab -e
no crontab for amanda_ong - using an empty one

Select an editor.  To change later, run 'select-editor'.
  1. /bin/nano        <---- easiest
  2. /usr/bin/vim.basic
  3. /usr/bin/vim.tiny
  4. /bin/ed
```

Figure 44: Interface when first time creating Crontab.

Step 3: Save and exit

After typing the cron jobs, I saved and exited nano:

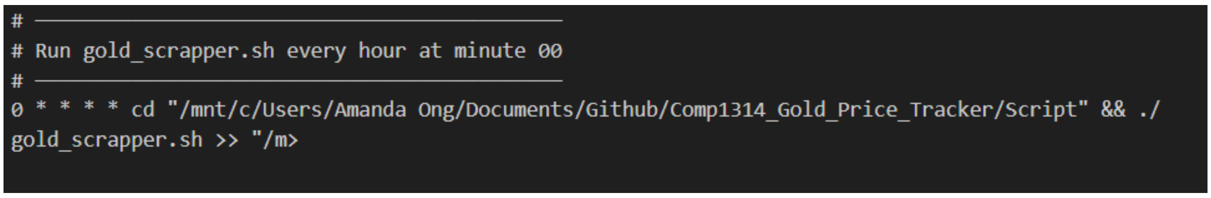
- Press Ctrl + O to save

- Press Enter to confirm
- Press Ctrl + X to exit

8.3 Crontab Commands Used

To avoid path error, each cron job first changes to the script folder using cd, then runs the script. Output and errors are saved into a log file (cron.log) for checking.

1. **Run scraper at minute 00**

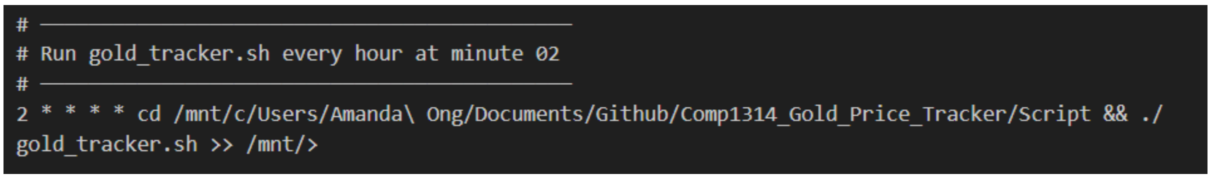


```
# ——————————————————————————————————————
# Run gold_scrapper.sh every hour at minute 00
# ——————————————————————————————————————
0 * * * * cd "/mnt/c/Users/Amanda Ong/Documents/Github/Comp1314_Gold_Price_Tracker/Script" && ./
gold_scrapper.sh >> "/m>
```

Figure 45: Cron Command Used.

Explanation:
This command runs the scraper at the start of every hour. It collects the latest gold price data and inserts it into the database. Output and errors are saved in cron.log.

2. **Run tracker at minute 02**

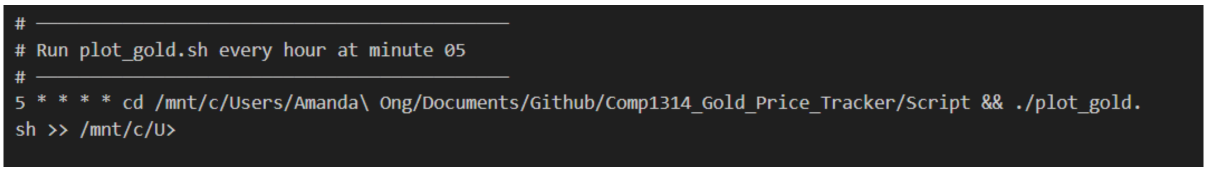


```
# ——————————————————————————————————————
# Run gold_tracker.sh every hour at minute 02
# ——————————————————————————————————————
2 * * * * cd /mnt/c/Users/Amanda\ Ong/Documents/Github/Comp1314_Gold_Price_Tracker/Script && ./
gold_tracker.sh >> /mnt/>
```

Figure 46: Cron Command Used.

Explanation:
This command runs two minutes after the scraper. It processes the collected data. The delay ensures the scraper finishes first.

3. **Run plotter at minute 05**



```
# ——————————————————————————————————————
# Run plot_gold.sh every hour at minute 05
# ——————————————————————————————————————
5 * * * * cd /mnt/c/Users/Amanda\ Ong/Documents/Github/Comp1314_Gold_Price_Tracker/Script && ./plot_gold.
sh >> /mnt/c/U>
```

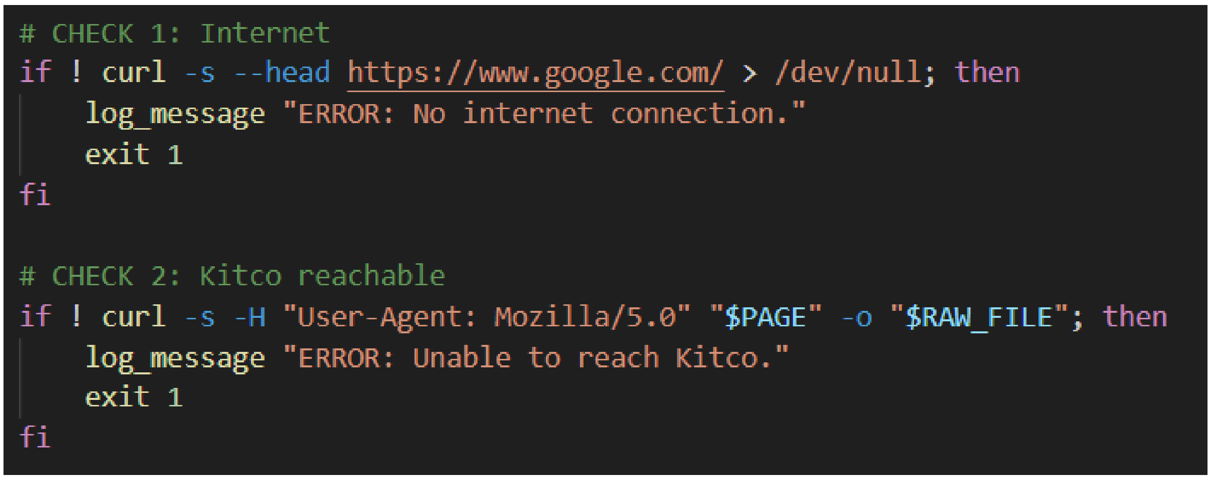
Figure 47: Cron Command Used.

Explanation:

This command generates updated plots using the latest database data. Running it last ensures the graphs are always up to date.

9.0 ERROR HANDLING

9.1 Example 1: Network and Source Validation (gold_tracker.sh)

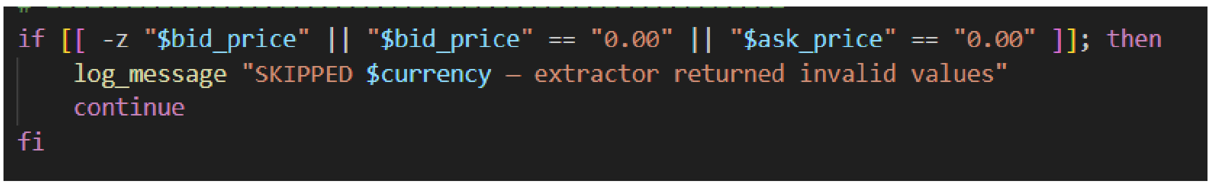


```
# CHECK 1: Internet
if ! curl -s --head https://www.google.com/ > /dev/null; then
    log_message "ERROR: No internet connection."
    exit 1
fi

# CHECK 2: Kitco reachable
if ! curl -s -H "User-Agent: Mozilla/5.0" "$PAGE" -o "$RAW_FILE"; then
    log_message "ERROR: Unable to reach Kitco."
    exit 1
fi
```

Figure 48: Example code of Network and Source Validation.

Before any data extraction is performed, the script verifies that an active internet connection is available and that the target website (Kitco) can be reached successfully. If either check fails, the script terminates immediately and records an error message in the log file. This prevents the system from attempting to parse missing or incomplete HTML data and ensures that only valid data sources are processed.

9.2 Example 2: Preventing Invalid Data Insertion (gold_tracker.sh)

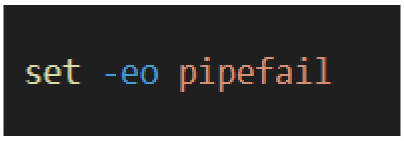


```
if [[ -z "$bid_price" || "$bid_price" == "0.00" || "$ask_price" == "0.00" ]]; then
    log_message "SKIPPED $currency — extractor returned invalid values"
    continue
fi
```

Figure 49: Example code of preventing Invalid Data Insertion.

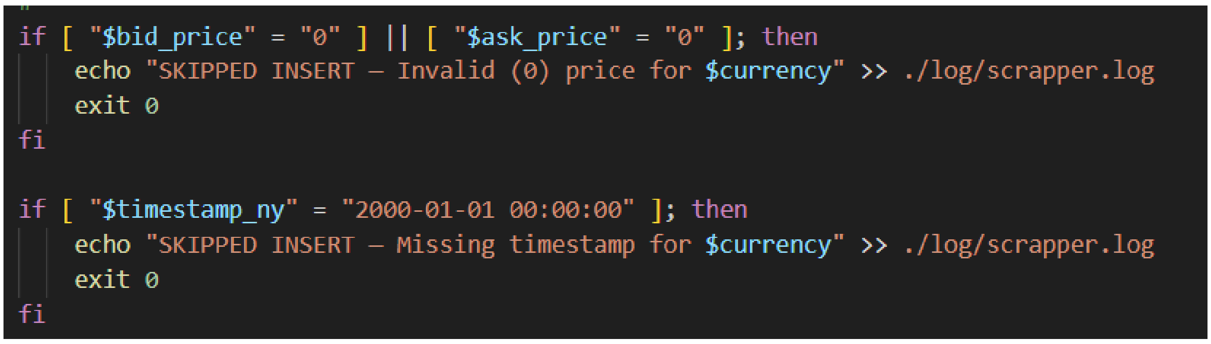
After extracting price values from the HTML file, the script validates critical fields such as bid and ask prices. If the extracted values are missing or equal to zero, the insertion process for that currency is skipped. This validation step ensures that incomplete or incorrect price data is not passed to the database insertion script.

9.3 Example 3: Safe Argument Handling and Insert Blocking (gold_scrapper.sh)



```
set -eo pipefail
```

Figure 50: Example code for Safe Argument Handling and Insert Blocking.



```
if [ "$bid_price" = "0" ] || [ "$ask_price" = "0" ]; then
    echo "SKIPPED INSERT – Invalid (0) price for $currency" >> ./log/scrapper.log
    exit 0
fi

if [ "$timestamp_ny" = "2000-01-01 00:00:00" ]; then
    echo "SKIPPED INSERT – Missing timestamp for $currency" >> ./log/scrapper.log
    exit 0
fi
```

Figure 51: Example code for Safe Argument Handling and Insert Blocking.

The database insertion script enforces additional validation before executing SQL statements. Default values are assigned to missing arguments to prevent script crashes, and critical fields such as timestamps and price values are checked explicitly. If invalid data is detected, the insertion is skipped and a log entry is recorded instead of allowing incorrect records to enter the database. The use of set -eo pipefail further ensures that the script exits immediately if any command fails, preventing partial or inconsistent database updates.

10.0 DATABASE EXPORT FROM WSL

This section describes the procedure used to **export and verify the MySQL database contents** generated by the Gold Price Tracker system. Since the database is hosted locally within the **WSL (Windows Subsystem for Linux)** environment, direct access by the lecturer is not possible. Therefore, all relevant database contents were exported and included as assessment evidence.

10.1 Crontab Commands Used

Running locally on the author's computer, the MySQL database holds historical gold price data that is automatically gathered using cron jobs.

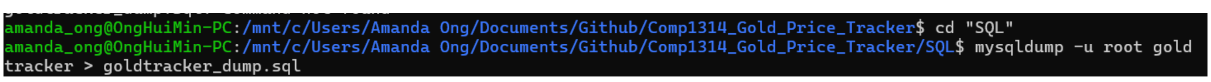
As this database is not externally accessible, exporting the database serves the following purposes:

- To provide **verifiable evidence** that data was successfully collected
- To demonstrate **correct database schema implementation** based on the ERD
- To preserve **cron-generated historical records**
- To allow the lecturer to **inspect table structures and data values**

10.2 Full Database Export

A complete database export was performed using the **mysqldump** utility. This export contains all tables, relationships and data records.

9.2.1 Export Command



```
amanda_ong@OngHuiMin-PC:/mnt/c/Users/Amanda Ong/Documents/Github/Comp1314_Gold_Price_Tracker$ cd "SQL"
amanda_ong@OngHuiMin-PC:/mnt/c/Users/Amanda Ong/Documents/Github/Comp1314_Gold_Price_Tracker/SQL$ mysqldump -u root gold
tracker > goldtracker_dump.sql
```

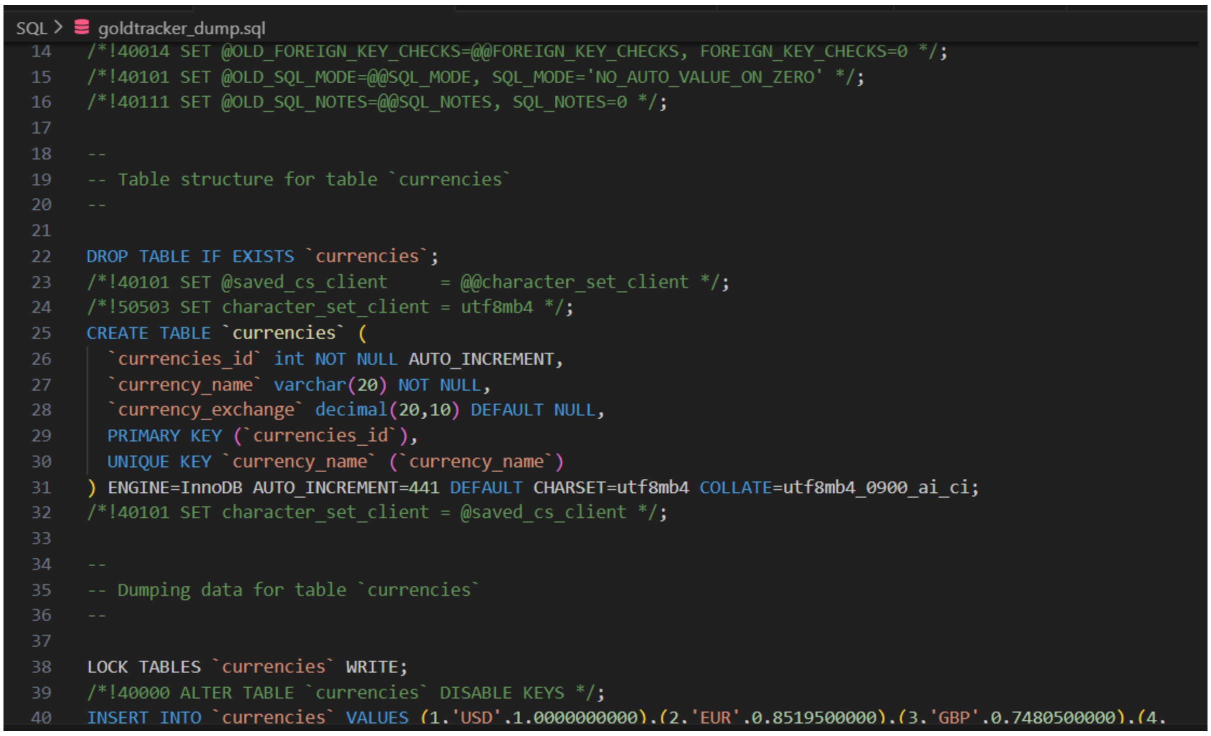
Figure 52: Export Command Used.

10.2.2 Exported Contents

The generated file goldtracker_dump.sql includes:

- Database creation statements
- Table definitions (currencies, gold_prices, unit_prices, logs)
- Foreign key relationships
- All historical data inserted by scheduled cron executions

This file allows the database to be **fully reconstructed** on another MySQL system if required.



SQL > goldtracker_dump.sql

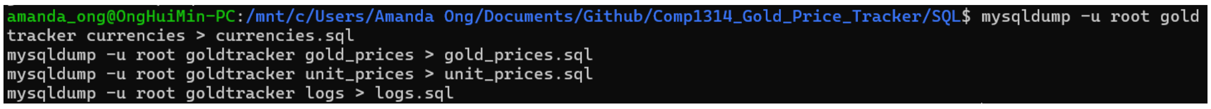
```sql
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

--
-- Table structure for table `currencies`
--

DROP TABLE IF EXISTS `currencies`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `currencies` (
  `currencies_id` int NOT NULL AUTO_INCREMENT,
  `currency_name` varchar(20) NOT NULL,
  `currency_exchange` decimal(20,10) DEFAULT NULL,
  PRIMARY KEY (`currencies_id`),
  UNIQUE KEY `currency_name` (`currency_name`)
) ENGINE=InnoDB AUTO_INCREMENT=441 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `currencies`
--

LOCK TABLES `currencies` WRITE;
/*!40000 ALTER TABLE `currencies` DISABLE KEYS */;
INSERT INTO `currencies` VALUES (1,'USD',1.0000000000),(2,'EUR',0.8519500000),(3,'GBP',0.7480500000),(4,
```

Figure 53: Example of Full Database Export.

10.3 Individual Table Exports

In addition to the full database export, each table was exported individually to provide **granular inspection and verification**.

10.3.1 Export Command



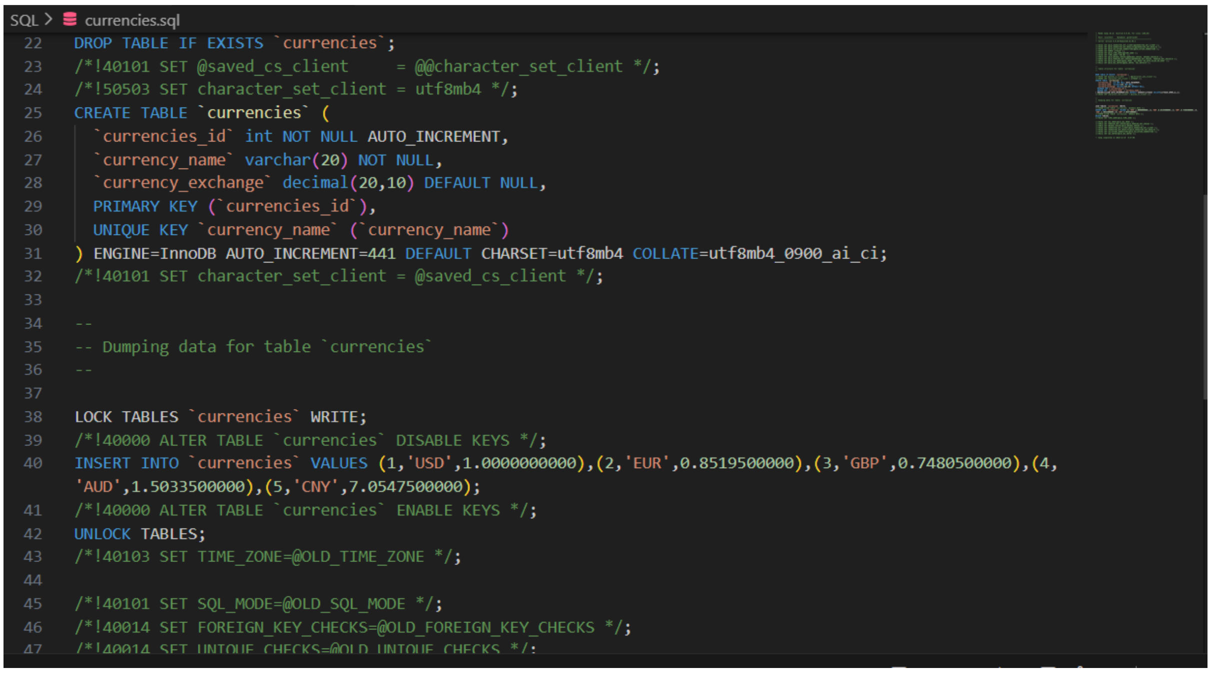
```
amanda_ong@OngHuiMin-PC:/mnt/c/Users/Amanda Ong/Documents/Github/Comp1314_Gold_Price_Tracker/SQL$ mysqldump -u root gold
tracker currencies > currencies.sql
mysqldump -u root goldtracker gold_prices > gold_prices.sql
mysqldump -u root goldtracker unit_prices > unit_prices.sql
mysqldump -u root goldtracker logs > logs.sql
```

Figure 54: Export Command Used.

10.3.2 Exported Contents

Exporting tables separately provides the following advantages:

- Easier verification of **data normalization**
- Clear inspection of **table-specific content**
- Independent validation of **foreign key relationships**
- Faster review without loading the entire database



SQL > currencies.sql

```sql
DROP TABLE IF EXISTS `currencies`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `currencies` (
  `currencies_id` int NOT NULL AUTO_INCREMENT,
  `currency_name` varchar(20) NOT NULL,
  `currency_exchange` decimal(20,10) DEFAULT NULL,
  PRIMARY KEY (`currencies_id`),
  UNIQUE KEY `currency_name` (`currency_name`)
) ENGINE=InnoDB AUTO_INCREMENT=441 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `currencies`
--

LOCK TABLES `currencies` WRITE;
/*!40000 ALTER TABLE `currencies` DISABLE KEYS */;
INSERT INTO `currencies` VALUES (1,'USD',1.0000000000),(2,'EUR',0.8519500000),(3,'GBP',0.7480500000),(4,
'AUD',1.5033500000),(5,'CNY',7.0547500000);
/*!40000 ALTER TABLE `currencies` ENABLE KEYS */;
UNLOCK TABLES;
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
```

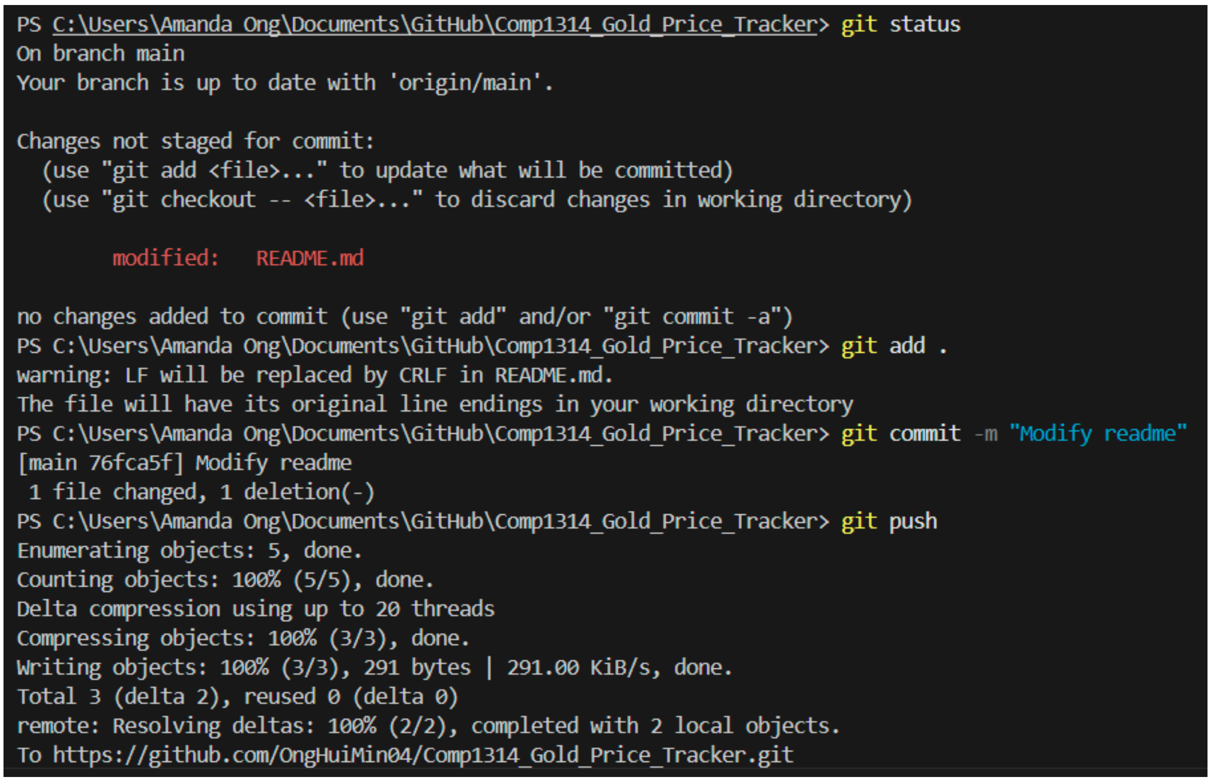
Figure 55: Example of certain table export.

11.0 VERSION CONTROL USING GIT

In this project, Git was used throughout this coursework to manage source code, track progress and maintain a clear history of development. All project files, including Bash scripts, SQL schema files, and plotting scripts, were stored in a GitHub repository. This ensured that the project was properly backed up and that changes could be reviewed and managed over time. By using Git, it also reflects real-world software development practices and supports good control discipline.

The version control process followed a clear workflow. After making changes to scripts or database files, the first command used was **git status**. This command allowed me to check which file have been modified, added or deleted. Once the changes were verified, the relevant files were added to the staging area using **git add .** . This step ensured that only the wanted files were prepared for submission, preventing accidental commits of unwanted or temporary files.

After staging the files, changes were saved using **git commit -m “”** with some commit message. Each commit message described what was changed, such as fixing a script error, adding a new SQL query, or updating the cron data. Commits were made when there are anything changes, demonstrating continuous development rather than a single final submission. Finally, the committed changes were uploaded to the remote GitHub repository using **git push**. This helps to ensure that the latest version of the project is safely stored online and available to view. Overall, Git provided a structured and reliable method for submitting and managing the project’ code and data.



```
PS C:\Users\Amanda Ong\Documents\GitHub\Comp1314_Gold_Price_Tracker> git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
PS C:\Users\Amanda Ong\Documents\GitHub\Comp1314_Gold_Price_Tracker> git add .
warning: LF will be replaced by CRLF in README.md.
The file will have its original line endings in your working directory
PS C:\Users\Amanda Ong\Documents\GitHub\Comp1314_Gold_Price_Tracker> git commit -m "Modify readme"
[main 76fca5f] Modify readme
 1 file changed, 1 deletion(-)
PS C:\Users\Amanda Ong\Documents\GitHub\Comp1314_Gold_Price_Tracker> git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 20 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 291 bytes | 291.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/OngHuiMin04/Comp1314_Gold_Price_Tracker.git
```

Figure 56: Example of Version Control using Git when submitting code to GitHub.

12.0 LIMITATIONS AND FUTURE IMPROVEMENTS

12.1 Limitations

Although the Gold Price Tracker system functions very well and is reliable, but there are several limitations were found during the development process. The first limitation is the **dependency on the structure of the target HTML webpage**. The data scraping process relies on specific HTML tags and patterns to extract gold price values. If the current website changes its layout, class names or table structure, the current regular expressions and parsing algorithms might not function properly. In these situations, the scraping script would need to be manually updated to conform to the new HTML format.

Besides, the **use of crontab for automation** is another limitation. The system must stay switched on and operational to complete the planned duties. Crontab jobs won't run on time if the laptop goes into hibernation or sleep mode. This may restrict long-term unattended data collecting because the system is not entirely autonomous and still depends on the user to keep the environment awake.

Lastly, the system currently **focuses on basic historical data storage and visualisation**. While price trends over time can be observed through generated plots, more advanced analysis, such as trend prediction, volatility measurement, or comparative analytics is not included. This limits the breadth of insights that can be generated from the information gathered.

12.2 Future Improvements

Several improvements can be made to enhance the system in the future. One major improvement would be **replacing HTML scraping if a structured JSON or REST API is available**. API-based data sources are more stable and less affected by layout changes, which would significantly improve system reliability.

Another improvement would be **adding error-handling and retry mechanisms for network failures**. Currently, data gathering may fail due to temporary internet problems. The system would be more resilient and fault-tolerant if timeout, retries and fallback logging were implemented.

Finally, the system could be **extended to include additional statistical plots** such as moving averages or percentage changes. These improvements would offer a deeper understanding of the dynamics of gold prices.

13.0 CONCLUSION

This project successfully shows the development of a complete UNIX-based data management system that automates the collection, storage and visualisation of gold price data. The project brings several key technical components, including Bash scripting, web data extraction, relational database design, and graphical data representation to form a fully functioning and reliable solution. By working with a Linux environment using WSL, the system reflects real-world server-based workflows, allowing scripts to run consistently without relying on manual execution. Each stage of the system, from data scraping to plot generation, has been carefully designed to work together as a single, integrated pipeline.

Data integrity and structured database design were prioritised throughout the development. The MySQL schema was developed directly from an Entity Relationship Diagram (ERD) and normalised to reduce redundancy and improve scalability. Primary keys, foreign keys, and constraints were used to enforce relationships between tables and ensure accurate data storage. Duplicate records are avoided and the tracker can be executed repeatedly thanks to automated insertion logic, which includes the usage of ON DUPLICATE KEY UPDATE. Because of this design, the system may operate continuously over time, gathering historical data without sacrificing performance or consistency.

Overall, this project provided hands-on experience in building and understanding automated data management system using UNIX tools. It highlights the importance of careful planning, testing and clear separation of responsibilities between scripts, database and visualisation tools. The use of Gnuplot visualisation and crontab-based automation shows how raw data may be converted into insightful knowledge with no human involvement.