**1.** schema.sql

**-- Table to store product information**  
CREATE TABLE IF NOT EXISTS products (  
 product\_id INTEGER PRIMARY KEY AUTOINCREMENT,  
 name TEXT NOT NULL,  
 category TEXT,  
 brand TEXT  
);  
  
-- Table to store retailer information  
CREATE TABLE IF NOT EXISTS retailers (  
 retailer\_id INTEGER PRIMARY KEY AUTOINCREMENT,  
 name TEXT NOT NULL,  
 website TEXT  
);  
  
-- Table to store prices  
CREATE TABLE IF NOT EXISTS prices (  
 price\_id INTEGER PRIMARY KEY AUTOINCREMENT,  
 product\_id INTEGER,  
 retailer\_id INTEGER,  
 price REAL,  
 currency TEXT,  
 date\_collected TEXT,  
 FOREIGN KEY (product\_id) REFERENCES products(product\_id),  
 FOREIGN KEY (retailer\_id) REFERENCES retailers(retailer\_id)  
);  
  
-- ------------------------------------------------------------------  
-- Indexes for faster search and joins  
-- ------------------------------------------------------------------  
CREATE INDEX IF NOT EXISTS idx\_prices\_product\_id ON prices (product\_id);  
CREATE INDEX IF NOT EXISTS idx\_prices\_retailer\_id ON prices (retailer\_id);  
CREATE INDEX IF NOT EXISTS idx\_products\_name ON products (name);

**2.** database.py

**# database.py**  
# ---------------------------------------------------------------------------------------------------------  
# Project Name : Product Price Intelligence System  
# Author : Deepa Ponnusamy  
# Email : deepa.ponnusamy@calbrightcollege.org  
# GitHub : https://github.com/python-sql09/Python-SQL/tree/main/myprojects/product\_price\_tracker  
# Date : June 03, 2025  
# Description : A price comparison tool that builds a structured SQLite database  
# to collect, store, and retrieve pricing data from online retailers.  
# Supports efficient product lookups, integrates with future  
# application infrastructure, and demonstrates database schema design,  
# data insertion, and querying logic.  
# ----------------------------------------------------------------------------------------------------------  
import sqlite3  
  
def create\_connection(db\_file="pricing\_db.db"):  
 return sqlite3.connect(db\_file)  
  
def initialize\_database():  
 conn = create\_connection()  
 with open("schema.sql", "r") as schema\_file:  
 conn.executescript(schema\_file.read())  
 conn.commit()  
 conn.close()

**3.** insert\_data.py

**# insert\_data.py**  
# -----------------------------------------------------------------------------------------------------  
# Project Name : Product Price Intelligence System  
# Author : Deepa Ponnusamy  
# Email : deepa.ponnusamy@calbrightcollege.org  
# GitHub : https://github.com/python-sql09/Python-SQL/tree/main/myprojects/product\_price\_tracker  
# Date : June 18, 2025  
# Description : Inserts sample data into a clean SQLite database, avoiding duplicates.  
# ----------------------------------------------------------------------------------------------------------  
from database import create\_connection  
def insert\_sample\_data():  
 conn = create\_connection()  
 cursor = conn.cursor()  
  
 # Clear existing records  
 cursor.execute("DELETE FROM prices")  
 cursor.execute("DELETE FROM products")  
 cursor.execute("DELETE FROM retailers")  
 conn.commit()  
  
 # Insert products  
 products = [  
 ('Wireless Mouse', 'Electronics', 'Logitech'),  
 ('Noise Cancelling Headphones', 'Audio', 'Sony'),  
 ('Portable Charger', 'Accessories', 'Anker'),  
 ('Bluetooth Speaker', 'Audio', 'JBL'),  
 ('Webcam HD', 'Electronics', 'Logitech'),  
 ('Smart Fitness Tracker', 'Wearables', 'Fitbit')  
 ]  
 cursor.executemany("INSERT INTO products (name, category, brand) VALUES (?, ?, ?)", products)  
 conn.commit()  
  
 # Get product\_id mapping by name  
 cursor.execute("SELECT product\_id, name FROM products")  
 product\_map = {name: pid for pid, name in cursor.fetchall()}  
  
 # Insert retailers  
 retailers = [  
 ('Amazon', 'https://www.amazon.com'),  
 ('Best Buy', 'https://www.bestbuy.com'),  
 ('Walmart', 'https://www.walmart.com'),  
 ('Target', 'https://www.target.com'),  
 ('Newegg', 'https://www.newegg.com'),  
 ('eBay', 'https://www.ebay.com'),  
 ('Demo Store', 'http://demo.local')  
 ]  
 cursor.executemany("INSERT INTO retailers (name, website) VALUES (?, ?)", retailers)  
 conn.commit()  
  
 # Get retailer\_id mapping by name  
 cursor.execute("SELECT retailer\_id, name FROM retailers")  
 retailer\_map = {name: rid for rid, name in cursor.fetchall()}  
  
 from datetime import date  
 today = str(date.today())  
  
 # Now use mapped IDs to insert prices  
 prices = [  
 (product\_map['Wireless Mouse'], retailer\_map['Amazon'], 25.99, 'USD', today),  
 (product\_map['Wireless Mouse'], retailer\_map['Best Buy'], 27.49, 'USD', today),  
 (product\_map['Wireless Mouse'], retailer\_map['Target'], 26.89, 'USD', today),  
  
 (product\_map['Noise Cancelling Headphones'], retailer\_map['Amazon'], 199.99, 'USD', today),  
 (product\_map['Noise Cancelling Headphones'], retailer\_map['Newegg'], 189.49, 'USD', today),  
  
 (product\_map['Portable Charger'], retailer\_map['Walmart'], 39.99, 'USD', today),  
 (product\_map['Portable Charger'], retailer\_map['eBay'], 35.99, 'USD', today),  
  
 (product\_map['Bluetooth Speaker'], retailer\_map['Amazon'], 79.99, 'USD', today),  
 (product\_map['Bluetooth Speaker'], retailer\_map['Best Buy'], 76.49, 'USD', today),  
 (product\_map['Bluetooth Speaker'], retailer\_map['Newegg'], 74.99, 'USD', today),  
  
 (product\_map['Webcam HD'], retailer\_map['Amazon'], 49.99, 'USD', today),  
 (product\_map['Webcam HD'], retailer\_map['Best Buy'], 47.99, 'USD', today),  
 (product\_map['Webcam HD'], retailer\_map['Target'], 46.50, 'USD', today),  
  
 (product\_map['Smart Fitness Tracker'], retailer\_map['Amazon'], 99.99, 'USD', today),  
 (product\_map['Smart Fitness Tracker'], retailer\_map['Target'], 95.50, 'USD', today),  
 (product\_map['Smart Fitness Tracker'], retailer\_map['eBay'], 92.99, 'USD', today),  
 ]  
  
 cursor.executemany("""  
 INSERT INTO prices (product\_id, retailer\_id, price, currency, date\_collected)  
 VALUES (?, ?, ?, ?, ?)  
 """, prices)  
  
 conn.commit()  
 conn.close()

**4.** main.py

**# main.py**  
# ---------------------------------------------------------------------------------------------------------  
# Project Name : Product Price Intelligence System  
# Author : Deepa Ponnusamy  
# Email : deepa.ponnusamy@calbrightcollege.org  
# GitHub : https://github.com/python-sql09/Python-SQL/tree/main/myprojects/product\_price\_tracker  
# Date : June 21, 2025  
# Description : A price comparison tool that builds a structured SQLite database  
# to collect, store, and retrieve pricing data from online retailers.  
# Supports efficient product lookups, integrates with future  
# application infrastructure, and demonstrates database schema design,  
# data insertion, and querying logic.  
# ---------------------------------------------------------------------------------------------------------  
  
from database import initialize\_database  
from insert\_data import insert\_sample\_data  
import sqlite3  
import csv  
  
def get\_best\_prices():  
 conn = sqlite3.connect("pricing\_db.db")  
 cursor = conn.cursor()  
 query = """  
 SELECT p.name, MIN(pr.price) as best\_price  
 FROM prices pr  
 JOIN products p ON p.product\_id = pr.product\_id  
 GROUP BY p.name  
 """  
 cursor.execute(query)  
 results = cursor.fetchall()  
 conn.close()  
 return results  
  
def print\_best\_prices(results):  
 print("\nBest Prices:\n")  
 for row in results:  
 print(f"Product: {row[0]} | Best Price: ${row[1]:.2f}")  
  
def search\_product(product\_name):  
 conn = sqlite3.connect("pricing\_db.db")  
 cursor = conn.cursor()  
 query = """  
 SELECT p.name, r.name, pr.price, pr.currency, pr.date\_collected  
 FROM prices pr  
 JOIN products p ON p.product\_id = pr.product\_id  
 JOIN retailers r ON r.retailer\_id = pr.retailer\_id  
 WHERE LOWER(p.name) LIKE LOWER(?)  
 """  
 cursor.execute(query, (f"%{product\_name}%",))  
 results = cursor.fetchall()  
 conn.close()  
  
 if results:  
 print(f"\nSearch Results for '{product\_name}':\n")  
 for row in results:  
 print(f"Product: {row[0]} | Retailer: {row[1]} | Price: ${row[2]:.2f} {row[3]} | Date: {row[4]}")  
 else:  
 print(f"\nNo results found for '{product\_name}'.")  
  
def export\_best\_prices\_to\_csv(results, filename="best\_prices.csv"):  
 with open(filename, mode="w", newline="") as file:  
 writer = csv.writer(file)  
 writer.writerow(["Product", "Best Price"])  
 writer.writerows(results)  
 print(f"\n\U0001F4C1 Exported best prices to '{filename}'")  
  
def print\_sample\_data():  
 conn = sqlite3.connect("pricing\_db.db")  
 cursor = conn.cursor()  
 print("\nSample products:")  
 for row in cursor.execute("SELECT product\_id, name FROM products LIMIT 5"):  
 print(row)  
 print("\nSample retailers:")  
 for row in cursor.execute("SELECT retailer\_id, name FROM retailers LIMIT 5"):  
 print(row)  
 print("\nSample prices:")  
 for row in cursor.execute("SELECT product\_id, retailer\_id, price FROM prices LIMIT 5"):  
 print(row)  
 conn.close()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 initialize\_database()  
 insert\_sample\_data()  
 print\_sample\_data() # Verify inserted data  
 best\_prices = get\_best\_prices()  
 print\_best\_prices(best\_prices)  
 search\_product("Mouse")  
 export\_best\_prices\_to\_csv(best\_prices)

**5.** scraper.py

**# scraper.py**  
# ------------------------------------------------------------------  
# Description: Simulates fetching product name and price from a local HTML file  
# using BeautifulSoup for demonstration purposes.  
# ------------------------------------------------------------------  
  
from bs4 import BeautifulSoup  
  
def fetch\_price\_from\_demo\_file(filepath):  
 try:  
 with open(filepath, 'r', encoding='utf-8') as file:  
 content = file.read()  
 soup = BeautifulSoup(content, 'html.parser')  
  
 product = soup.find('h1', {'class': 'product-name'}).text.strip()  
 price\_text = soup.find('span', {'class': 'price'}).text.strip()  
 price = float(price\_text.replace('$', ''))  
  
 print(f"✅ Scraped from local file: {product} | Price: ${price:.2f}")  
 return product, price  
  
 except Exception as e:  
 print(f"❌ Error scraping product: {e}")  
 return None

6. demo\_scrape\_test.py

# demo\_scraper\_test.py  
# ------------------------------------------------------------------  
# Test script for scraper.py + database insertion  
# ------------------------------------------------------------------  
  
from scraper import fetch\_price\_from\_demo\_file  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 filepath = "demo\_product.html" # Make sure this file is present in the project directory  
  
 result = fetch\_price\_from\_demo\_file(filepath)  
  
 if result:  
 product, price = result  
 print(f"🎯 Final Result -> Product: {product} | Price: ${price}")  
 else:  
 print("❗ No product information extracted.")

7. price\_dashboard.py

# price\_dashboard.py  
# ------------------------------------------------------------------  
# Streamlit Dashboard to View and Search Product Prices  
# ------------------------------------------------------------------  
  
import streamlit as st  
import sqlite3  
import pandas as pd  
  
def get\_connection():  
 return sqlite3.connect("pricing\_db.db")  
  
def get\_best\_prices():  
 conn = get\_connection()  
 df = pd.read\_sql\_query("""  
 SELECT p.name AS Product, MIN(pr.price) AS Best\_Price  
 FROM prices pr  
 JOIN products p ON p.product\_id = pr.product\_id  
 GROUP BY p.name  
 """, conn)  
 conn.close()  
 return df  
  
def get\_all\_products():  
 conn = get\_connection()  
 df = pd.read\_sql\_query("SELECT name FROM products", conn)  
 conn.close()  
 return df["name"].tolist()  
  
def get\_all\_retailers():  
 conn = get\_connection()  
 df = pd.read\_sql\_query("SELECT name FROM retailers", conn)  
 conn.close()  
 return df["name"].tolist()  
  
def search\_product(product\_name="", retailer\_name=""):  
 conn = get\_connection()  
 query = """  
 SELECT p.name AS Product, r.name AS Retailer, pr.price AS Price,  
 pr.currency AS Currency, pr.date\_collected AS Date  
 FROM prices pr  
 JOIN products p ON p.product\_id = pr.product\_id  
 JOIN retailers r ON r.retailer\_id = pr.retailer\_id  
 WHERE p.name LIKE ? AND r.name LIKE ?  
 """  
 df = pd.read\_sql\_query(query, conn, params=(f"%{product\_name}%", f"%{retailer\_name}%"))  
 conn.close()  
 return df  
  
st.set\_page\_config(page\_title="💲 Price Intelligence", layout="wide")  
st.title("🛍️ Product Price Intelligence Dashboard")  
  
tab1, tab2, tab3 = st.tabs(["📌 Best Prices", "🔍 Filter & Search", "📊 Price Chart"])  
  
with tab1:  
 st.subheader("📉 Best Prices by Product")  
 best\_prices\_df = get\_best\_prices()  
 st.dataframe(best\_prices\_df, use\_container\_width=True)  
  
 csv = best\_prices\_df.to\_csv(index=False).encode("utf-8")  
 st.download\_button("⬇️ Download as CSV", csv, "best\_prices.csv", "text/csv")  
  
with tab2:  
 st.subheader("🔍 Filter Products & Retailers")  
 col1, col2 = st.columns(2)  
  
 with col1:  
 selected\_product = st.selectbox("Select Product", [""] + get\_all\_products())  
 with col2:  
 selected\_retailer = st.selectbox("Select Retailer", [""] + get\_all\_retailers())  
  
 if selected\_product or selected\_retailer:  
 filtered\_df = search\_product(selected\_product, selected\_retailer)  
 if not filtered\_df.empty:  
 st.dataframe(filtered\_df, use\_container\_width=True)  
 else:  
 st.warning("⚠️ No results for your selection.")  
  
with tab3:  
 st.subheader("📊 Visualize Best Prices")  
 chart\_df = get\_best\_prices()  
 st.bar\_chart(data=chart\_df.set\_index("Product"), use\_container\_width=True)

8. demo\_product.html (sample HTML file for scraping demo)

<!-- demo\_product.html -->  
<html>  
 <body>  
 <div class="product">  
 <span class="product-title">Wireless Mouse</span>  
 <span class="price">$25.99</span>  
 </div>  
 </body>  
</html>

9.Readme.md

**#** Product Price Intelligence System  
  
This is a SQLite-based system that stores and retrieves product pricing data from multiple online retailers. It supports price comparisons and can be extended for integration with real-world web scraping tools or e-commerce APIs.  
  
### Features  
- Structured SQLite schema  
- Insert and retrieve product data  
- Price comparison report (best price per product)  
- Modular code with clean separation  
  
### Tools Used  
- Python 3  
- SQLite  
- PyCharm IDE  
  
### How to Run  
1. Clone or download this project.  
2. Open it in PyCharm.  
3. Make sure you have required Python packages installed:  
 \* pip install streamlit pandas beautifulsoup4  
4. Run main.py once to initialize DB and insert sample data:  
 \* python main.py  
5. Run Streamlit dashboard:  
 \* streamlit run price\_dashboard.py  
6. Run scraper test if you want to test scraping local HTML:  
 \* python demo\_scrape\_test.py  
   
*### Author*Deepa Ponnusamy   
[GitHub Portfolio](https://github.com/python-sql09/tree/main/myprojects/product\_price\_tracker)

# How to Run

1. Make sure you have required Python packages installed:

pip install streamlit pandas beautifulsoup4

1. Run main.py once to initialize DB and insert sample data:

python main.py

1. Run Streamlit dashboard:

streamlit run price\_dashboard.py

1. Run scraper test if you want to test scraping local HTML:

python demo\_scrape\_test.py

## 🧩 ****Project Title:****

**"Product Price Intelligence System using SQLite"**

## ✅ ****Project Goal:****

To build a professional, efficient, and maintainable database system that:

* Collects product pricing data from multiple retailers.
* Stores it in a structured format (SQLite).
* Retrieves the best price per product.
* Can be visualized and eventually integrated into a price-scouting application.

## 🛠️ ****Step 1: Project Setup****

### 1. Create a new folder/project in PyCharm

Name it: product\_price\_tracker

### 2. Inside that project, create the following files:

pgsql

CopyEdit

product\_price\_tracker/

├── main.py

├── database.py

├── schema.sql

├── insert\_data.py

├── README.md

## 📄 ****Step 2: Create**** schema.sql

This file defines your database structure (tables).

sql

CopyEdit

-- schema.sql

CREATE TABLE IF NOT EXISTS products (

product\_id INTEGER PRIMARY KEY AUTOINCREMENT,

name TEXT NOT NULL,

category TEXT,

brand TEXT

);

CREATE TABLE IF NOT EXISTS retailers (

retailer\_id INTEGER PRIMARY KEY AUTOINCREMENT,

name TEXT NOT NULL,

website TEXT

);

CREATE TABLE IF NOT EXISTS prices (

price\_id INTEGER PRIMARY KEY AUTOINCREMENT,

product\_id INTEGER,

retailer\_id INTEGER,

price REAL,

currency TEXT,

date\_collected TEXT,

FOREIGN KEY (product\_id) REFERENCES products(product\_id),

FOREIGN KEY (retailer\_id) REFERENCES retailers(retailer\_id)

);

## 🧠 ****Step 3: Create**** database.py

This script connects to the database and sets it up using the schema.

python

CopyEdit

# database.py

import sqlite3

def create\_connection(db\_file="pricing\_db.db"):

return sqlite3.connect(db\_file)

def initialize\_database():

conn = create\_connection()

with open("schema.sql", "r") as schema\_file:

conn.executescript(schema\_file.read())

conn.commit()

conn.close()

## 📥 ****Step 4: Create**** insert\_data.py

This script inserts some initial test data.

python

CopyEdit

# insert\_data.py

from database import create\_connection

from datetime import date

def insert\_sample\_data():

conn = create\_connection()

cursor = conn.cursor()

# Insert products

products = [

('Wireless Mouse', 'Electronics', 'Logitech'),

('Noise Cancelling Headphones', 'Audio', 'Sony'),

('Portable Charger', 'Accessories', 'Anker')

]

cursor.executemany("INSERT INTO products (name, category, brand) VALUES (?, ?, ?)", products)

# Insert retailers

retailers = [

('Amazon', 'https://www.amazon.com'),

('Best Buy', 'https://www.bestbuy.com'),

('Walmart', 'https://www.walmart.com')

]

cursor.executemany("INSERT INTO retailers (name, website) VALUES (?, ?)", retailers)

# Insert prices

prices = [

(1, 1, 25.99, 'USD', str(date.today())),

(1, 2, 27.49, 'USD', str(date.today())),

(2, 1, 199.99, 'USD', str(date.today())),

(3, 3, 39.99, 'USD', str(date.today()))

]

cursor.executemany(

"INSERT INTO prices (product\_id, retailer\_id, price, currency, date\_collected) VALUES (?, ?, ?, ?, ?)",

prices

)

conn.commit()

conn.close()

## 🚀 ****Step 5: Create**** main.py

This script ties everything together and queries the best prices.

python

CopyEdit

# main.py

from database import initialize\_database

from insert\_data import insert\_sample\_data

import sqlite3

def get\_best\_prices():

conn = sqlite3.connect("pricing\_db.db")

cursor = conn.cursor()

query = """

SELECT p.name, MIN(pr.price) as best\_price

FROM prices pr

JOIN products p ON p.product\_id = pr.product\_id

GROUP BY p.name

"""

cursor.execute(query)

results = cursor.fetchall()

print("\nBest Prices:\n")

for row in results:

print(f"Product: {row[0]} | Best Price: ${row[1]:.2f}")

conn.close()

if \_\_name\_\_ == "\_\_main\_\_":

initialize\_database()

insert\_sample\_data()

get\_best\_prices()

## 📝 ****README.md (Professional Portfolio Touch)****

markdown

CopyEdit

# Product Price Intelligence System

This is a SQLite-based system that stores and retrieves product pricing data from multiple online retailers. It supports price comparisons and can be extended for integration with real-world web scraping tools or e-commerce APIs.

### Features

- Structured SQLite schema

- Insert and retrieve product data

- Price comparison report (best price per product)

- Modular code with clean separation

### Tools Used

- Python 3

- SQLite

- PyCharm IDE

### How to Run

1. Clone or download this project.

2. Open it in PyCharm.

3. Run `main.py`.

### Author

Deepa Ponnusamy

[GitHub Portfolio](https://github.com/python-sql09)

## ✅ Next Steps:

* Let me know when you’ve created the folder and files.
* I’ll help you test, improve, and then optionally expand this into:
  + JSON/web scraping input
  + Power BI/Excel export
  + Visual dashboards using Python or Tableau

Ready to build this like a pro? Let’s go step by step — just say **“ready for step 1”** and we’ll begin.