Practice 2: Finishing the Cleaning, and the Model

Python and SQL. You need the file Practice 1 produced.

Where we are

Practice 1 gave you one file with 72,437 rows:

`build/consolidated/consolidated_long.csv`

It looked finished. It is not. In Part 1 you will find real mistakes in it. In Part 2 you clean the descriptions, not the numbers. In Part 3 we name the pieces of the model you will build next week.

Get the scripts first. The ready-made ETL scripts live at `dwhdemo.vedara.net/docs/etl/` — extract.py, extract_multi.py, clean_consolidate.py, make_source_fixtures.py. Download them into a folder called `etl/` next to your data.

Part 1 — What is still wrong

Task 1.1 — Look for the same thing twice

A key is one country, one indicator, one year. The same **source** should never give two values for one key. Count how often it does:

```
import csv
from collections import Counter

rows = list(csv.DictReader(open("build/consolidated/consolidated_long.csv")))

seen = Counter((r["source_id"], r["iso3"], r["indicator_cd"], r["year"])
               for r in rows)

dupes = [k for k, n in seen.items() if n > 1]
print(len(dupes), "keys appear twice from the same source")
print(Counter(k[0] for k in dupes))
```

You should find **909**, all from Eurostat.

Task 1.2 — Find out why

Print both values for one of them:

```
k = dupes[0]
for r in rows:
    if (r["source_id"], r["iso3"], r["indicator_cd"], r["year"]) == k:
        print(r["value"], r["unit"], r["flag"])
```

For Albania in 2000 you get **2120.0** and **8.2**. One is euros per person, the other is a percentage change. They are two different measures that our cleaner put under one name.

Now look at the raw file `staging_multi/s4_eurostat.tsv`. The first column holds four values joined by commas. The second one is the unit:

```
A,CLV20_EUR_HAB,B1GQ,AL      <- euros per person
A,CLV_PCH_PRE_HAB,B1GQ,AL    <- percent change
```

The Practice 1 cleaner only read the country code and ignored the unit. **Fix it yourself:** in `clean_s4_eurostat`, skip any row whose unit is not `CLV20_EUR_HAB`. Re-run and check the 909 are gone; Eurostat should drop from 1,854 rows to 923, and the total from 72,437 to 71,506.

A corrected copy is published as `docs/etl/clean_consolidate_fixed.py` — compare it with yours after you have tried, and use it from Practice 3 on.

This is normal. You do not find every problem on the first pass. You find them when you ask a question the data cannot answer — here, “is this key unique?”

Task 1.3 — Look for impossible numbers

Check each indicator against what it can physically be:

```
for r in rows:
    v = float(r["value"])
    if r["indicator_cd"] == "UNEMPLOYMENT" and not 0 <= v <= 100:
        print("impossible unemployment:", r)
    if r["indicator_cd"] in ("GDP_CUR_USD", "POPULATION") and v < 0:
        print("negative:", r)
```

You should find none. That is a useful result: it means the number parsing in Practice 1 worked.

Task 1.4 — Look at the extremes

Print the ten largest inflation values:

```
infl = [r for r in rows if r["indicator_cd"] == "INFLATION_CPI"]
infl.sort(key=lambda r: -float(r["value"]))

for r in infl[:10]:
    print(r["iso3"], r["year"], float(r["value"]))
```

The top one is Venezuela in 2018 at **65,374%**. Zimbabwe 2024 is 736%. Your first instinct is that these are broken. They are not — that is what hyperinflation looks like.

An outlier is not an error. Deleting values because they look extreme is how you lose the most interesting part of a dataset. Check the value before you touch it, and write down the rule you used.

Part 2 — Cleaning the descriptions

So far you cleaned numbers. The names, codes and groupings that go *with* the numbers are also dirty, and nobody has looked at them yet.

Task 2.1 — One country, three names

Each source writes country names its own way:

Code	World Bank	Excel workbook	SQL agency
USA	United States	United States of America	USA
GBR	United Kingdom	United Kingdom of Great Britain	GBR
DEU	Germany	Germany (Fed. Rep.)	DEU
KOR	Korea, Rep.	Korea, Rep. of	KOR

Pick one name per country and say why. This is the same kind of decision as choosing between two numbers, except nobody can check you with arithmetic. **Do not join on the name** — join on the code, which all three agree on.

Task 2.2 — Facts with no country

Every fact should belong to a country you have a description for. Check:

```
import json

# the country list Practice 1 downloaded
countries = json.load(open("staging/countries.json"))
known = {c["id"] for c in countries}

orphans = {r["iso3"] for r in rows} - known
print(len(orphans), "codes have facts but no description")
print(sorted(orphans))
```

You will find **25** codes carrying **2,466** facts, all from the IMF. They include **TWN** (Taiwan), **UVK** (Kosovo) and **WBG** (West Bank and Gaza) — places the World Bank list does not cover.

You have to choose, and either answer is defensible:

Option	What it costs
Drop the facts	You lose 2,466 real observations and Taiwan disappears from your warehouse.
Add the missing rows by hand	You are now a publisher of country data, and must keep it up to date.

Whichever you pick, write it down. Someone will ask why Taiwan is missing.

Task 2.3 — Describe the sources themselves

Nothing yet records where a number came from. Build a small table by hand with one row per source: its id, its name, and the format you read it from (JSON, CSV, TSV, HTML, Excel, SQL). Seven rows. You will need it next week.

Task 2.4 — Countries are not groups

The entity list holds **217 countries** and **78 aggregates** — things like “World”, “Euro area” and “High income”. They sit in the same list because the World Bank publishes them together.

Add up the GDP of everything in the list for 2024, then compare it with the row called World. What do you get, and why?

> Adding “World” to “Germany” counts Germany twice. Any table mixing the two is wrong, and it will look perfectly reasonable. Keep a column that says which kind each row is.

Part 3 — Naming the pieces

Next week you build the database. These are the four words you need first. No code this week — just be able to point at each one in your own data.

Fact

A measurement. “Germany, GDP, 2024, 4.69 trillion dollars.” Facts are numbers you add up, average, or rank. You have about 57,000 of them.

Dimension

A description you filter or group by. “Germany is in Europe and is a high-income country.” Dimensions are text, they are few, and they change rarely. Your four are: country, indicator, year, source.

The test: if you would put it after `WHERE` or `GROUP BY`, it is a dimension. If you would put it inside `SUM()`, it is a fact.

Grain

What one row means. Ours is **one country, one indicator, one year**. Decide this before anything else: it fixes what questions you can answer. If you had kept months, the grain would be finer and the table much bigger.

Two rows with the same grain values are a contradiction, not extra information. That is exactly the bug you found in Task 1.1.

Surrogate key

A plain number standing in for a real-world code. Instead of writing `"DEU"` in 57,000 fact rows, you write `57`, and keep one row elsewhere saying 57 means Germany. It is smaller, faster to join, and when a country is renamed you change one row instead of thousands.

Task 3.1 — Sort your own columns

Take the seven columns of your clean file and put each one in a box: fact, dimension, or neither. Then write one sentence saying what the grain of your file is. Bring both next week.

Next class

Transformation. Building the star schema from the pieces you just named: the dimension tables, the fact table, and the queries they make easy.

A finished version of this warehouse is at dwhdemo.vedara.net. Practice 1: dwhdemo.vedara.net/docs/practice_gathering_cleaning.pdf