Practice 3: Transformation

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

Where we are

Practice 1 cleaned each of the seven sources separately, then stacked them into **one** file:

`build/consolidated/consolidated_long.csv`

It has one row per source per observation, with these columns:

Column	Example	Meaning
`source_id`	`S1_worldbank_api`	which source this row came from
`iso3`	`USA`	country code
`indicator_cd`	`GDP_CUR_USD`	which measure
`year`	`2024`	year
`value`	`29298000000000`	the number
`unit`	`USD`	unit of the number
`flag`	`FORECAST`	empty, or why the value is special

Today you turn that one long table into a database you can query: a few small description tables and one big table of numbers.

Your file has rows from several sources. Pick **one source per indicator** and keep only those rows — one line of code. Which source is a judgement call: the World Bank for development series, the IMF for fiscal ones. Write down what you chose.

```
KEEP = {
    "GDP_CUR_USD":   "S1_worldbank_api",
    "GDP_GROWTH":    "S2_imf_datamapper",
    "INFLATION_CPI": "S2_imf_datamapper",
    "UNEMPLOYMENT":  "S1_worldbank_api",
    "POPULATION":    "S1_worldbank_api",
    "EXPORTS_GDP":   "S1_worldbank_api",
    "GOVT_DEBT_GDP": "S2_imf_datamapper",
    "GDP_PCAP_PPP":  "S3_worldbank_csv",
    "GDP_PCAP_EUR":  "S4_eurostat_tsv",
    "POLICY_RATE":   "S5_wiki_scrape",
}

rows = [r for r in rows
        if r["source_id"] == KEEP[r["indicator_cd"]]]

print(len(rows), "rows")
```

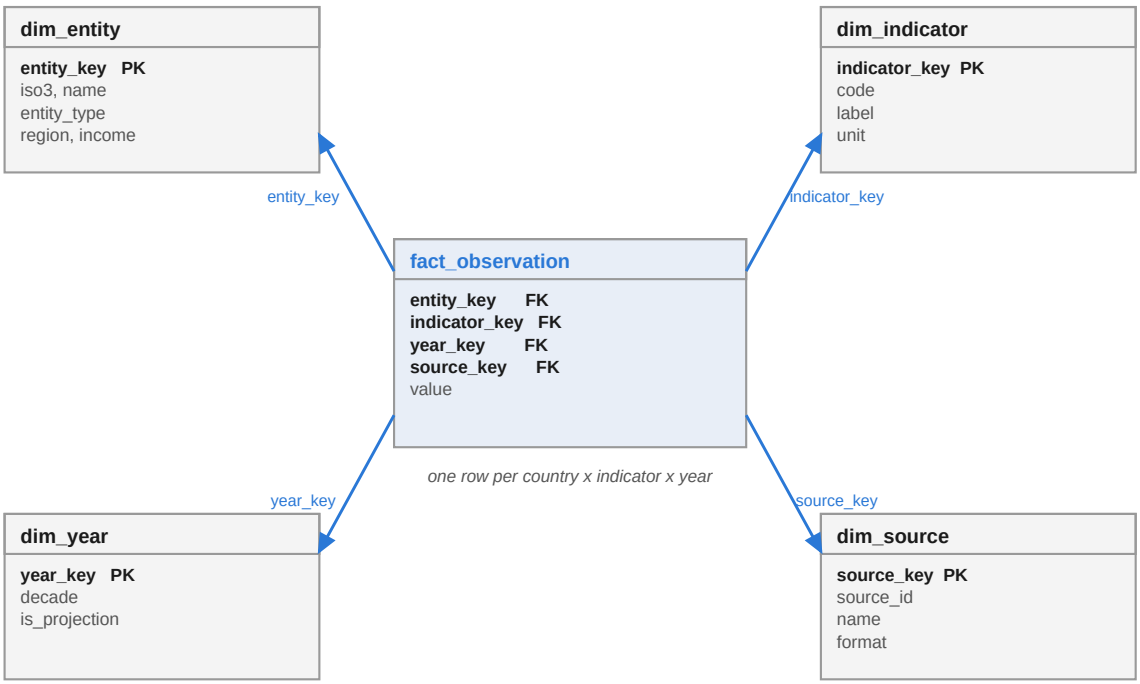
Part 1 — The star schema

One long table is fine for cleaning, but slow and repetitive to query. Split it into small description tables (dimensions) and one big number table (facts).

The five tables

Table	One row per	Holds
`dim_entity`	country or aggregate	name, ISO3, region, income group, type
`dim_indicator`	indicator	code, label, unit
`dim_year`	year	year, decade, is it a projection
`dim_source`	source	name, format, endpoint
`fact_observation`	country x indicator x year	the value, plus keys to the four dimensions

How they fit together



Every arrow is a join. The fact table stores four numbers that point at the four dimensions; the dimensions store everything you would want to filter or group by.

Task 1.1 — Create the tables

Note what dim_entity does: countries and aggregates live in the same table, told apart by one column.

```
CREATE TABLE dim_entity (
    entity_key   INTEGER PRIMARY KEY,
    iso3         TEXT UNIQUE NOT NULL,
    name         TEXT NOT NULL,
    entity_type  TEXT NOT NULL,   -- 'country' or 'aggregate'
    region       TEXT,
    income_level TEXT
);

CREATE TABLE dim_indicator (
    indicator_key INTEGER PRIMARY KEY,
    code  TEXT UNIQUE NOT NULL,
    label TEXT NOT NULL,
    unit  TEXT NOT NULL
);

CREATE TABLE dim_year (
    year_key      INTEGER PRIMARY KEY,
    year          INTEGER UNIQUE NOT NULL,
    is_projection INTEGER NOT NULL DEFAULT 0
);

CREATE TABLE dim_source (
    source_key INTEGER PRIMARY KEY,
    source_id  TEXT UNIQUE NOT NULL,
    name       TEXT NOT NULL,
    format     TEXT NOT NULL
);

CREATE TABLE fact_observation (
    entity_key    INTEGER NOT NULL REFERENCES dim_entity,
    indicator_key INTEGER NOT NULL REFERENCES dim_indicator,
    year_key      INTEGER NOT NULL REFERENCES dim_year,
    source_key    INTEGER NOT NULL REFERENCES dim_source,
    value         REAL NOT NULL,
    PRIMARY KEY (entity_key, indicator_key, year_key)
);
```

The primary key on the fact table is the grain: one row per country, indicator and year. If you try to insert a second row for the same three, the database stops you. That is the point.

Task 1.2 — Fill the dimensions first

Facts point at dimensions, so dimensions must exist first. Build a lookup from the natural key to the surrogate key as you go:

```
entity_keys = {}
for i, iso3 in enumerate(sorted(all_countries), start=1):
    entity_keys[iso3] = i
    db.execute("INSERT INTO dim_entity VALUES (?,?,?,?,?,?)",
               (i, iso3, name[iso3], kind[iso3], region[iso3], income[iso3]))

# then facts, translating each natural key into its number
db.execute("INSERT INTO fact_observation VALUES (?,?,?,?,?)",
           (entity_keys[iso3], indicator_keys[code],
            year, source_keys[src], value))
```

Why numbers and not names? The fact table has tens of thousands of rows. An integer is smaller and faster to join than a text name, and if a country is renamed you change one row in dim_entity, not thousands of facts.

Task 1.3 — Ask a question

Now the payoff. Top ten economies in 2024 — note the entity_type filter:

```
SELECT e.name, f.value / 1e12 AS gdp_trillions
FROM fact_observation f
JOIN dim_entity    e ON e.entity_key    = f.entity_key
JOIN dim_indicator i ON i.indicator_key = f.indicator_key
JOIN dim_year      y ON y.year_key      = f.year_key
WHERE i.code = 'GDP_CUR_USD'
  AND y.year = 2024
  AND e.entity_type = 'country'
ORDER BY f.value DESC
LIMIT 10;
```

Task 1.4 — Now remove one line

Delete `AND e.entity_type = 'country'` and run it again. What comes top now? How many of the ten are real countries?

This is the mistake the whole design exists to prevent. "World" is a row in the same table as "Germany". Without that one condition, your league table of economies is mostly not economies.

Part 2 — Check your work

Task 2.1 — Compare against the live warehouse

Open **dwhdemo.vedara.net**, pick a country, and compare a few numbers against your own database.

Where you differ, it is almost always because you kept a different source. That is a choice, not a bug — but you should be able to say which one you kept and why.

Next class

Load. Running the build twice must not create duplicates or half-written tables. We will look at building into a fresh copy, checking it, and only then replacing the live one.

A finished version of this warehouse is at dwhdemo.vedara.net. Practice 2: dwhdemo.vedara.net/docs/practice2_cleaning_model.pdf