Practice 5: A Dashboard

Python and Streamlit. You need the database from Practice 3.

Where we are

Last week you wrote four kinds of query: slice, dice, roll-up and drill-down. A dashboard is those same queries with controls on top, and several views of one slice on one page. Nothing new to learn about the warehouse — only how to put a front end on it.

Streamlit turns a Python script into a web page. You write ordinary Python; it draws the widgets.

```
pip install streamlit        # already done on the lab machines
streamlit run dashboard.py   # opens in your browser
```

Part 1 — One query, cached

Everything on the page comes from one query. Load it once and keep it in memory, so moving a control does not hit the database again:

```
import sqlite3
import pandas as pd
import streamlit as st

DB = "warehouse/world_economy_multi.sqlite"


@st.cache_data
def load(indicator_cd):
    sql = """
        SELECT e.name          AS country,
               e.region        AS region,
               y.year          AS year,
               y.is_projection AS projected,
               f.value         AS value
        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.indicator_cd  = ?
          AND e.entity_type   = 'country'
          AND e.region IS NOT NULL
    """
    with sqlite3.connect(DB) as db:
        return pd.read_sql_query(sql, db, params=[indicator_cd])
```

Why cache? Without `@st.cache_data` Streamlit re-runs the whole script every time you touch a control, so the query runs again and again. With it, the query runs once per indicator.

Part 2 — The controls are the slice and dice

```
INDICATORS = {
    "Unemployment (%)":     "UNEMPLOYMENT",
    "Inflation (%)":        "INFLATION_CPI",
    "GDP per capita (PPP)": "GDP_PCAP_PPP",
}

st.title("World Economy Dashboard")

c1, c2, c3 = st.columns(3)
label = c1.selectbox("Indicator", list(INDICATORS))
df = load(INDICATORS[label])

# 2026 onwards are IMF projections and cover half as many
# countries, so keep them out of the picker.
actual = sorted(df[df.projected == 0].year.unique())
year = c2.selectbox("Year", actual, index=len(actual) - 1)

regions = ["All regions"] + sorted(df.region.unique())
region = c3.selectbox("Region", regions)

view = df[df.year == year]                  # slice
if region != "All regions":
    view = view[view.region == region]      # dice
```

Those two lines are the slice and the dice from last week. The dropdowns just supply the values.

Part 3 — Four views of the same slice

A dashboard is not one chart. It is several views that answer different questions about the same selection, and they all change together.

Numbers

```
k1, k2, k3, k4 = st.columns(4)
by_region = view.groupby("region").value.mean().sort_values(ascending=False)

k1.metric("Average",        f"{view.value.mean():.1f}")
k2.metric("Highest region", f"{by_region.iloc[0]:.1f}",  by_region.index[0])
k3.metric("Lowest region",  f"{by_region.iloc[-1]:.1f}", by_region.index[-1])
k4.metric("Countries",      len(view))
```

Two charts

```
left, right = st.columns(2)

with left:
    st.subheader("By region")
    st.bar_chart(by_region, horizontal=True)     # roll-up

with right:
    st.subheader("Over time")
    scope = df[df.projected == 0]
    if region != "All regions":
        scope = scope[scope.region == region]
    st.line_chart(scope.groupby("year").value.mean())
```

A table

```
st.subheader("Countries")
if region == "All regions":
    detail = view.nlargest(10, "value")[["country", "region", "value"]]
else:
    detail = view.sort_values("value", ascending=False)[["country", "value"]]

st.dataframe(detail, hide_index=True, use_container_width=True)
```

Pick a region and watch all four change at once. The bar chart is a roll-up; choosing a region is a drill-down. Same operations as last week, now with a mouse.

Part 4 — Make it yours

Task 4.1 — Add an indicator

Add government debt to the `INDICATORS` dictionary. How much code did you have to change?

Task 4.2 — Add a widget

Add a fifth view: the three countries that changed most since 2015. You will need a second query, or a groupby on the data you already have.

Task 4.3 — Break the cache

Comment out `@st.cache_data` and change the year a few times. Does it feel slower? With 5,000 rows it probably does not — which is the point of the next section.

When this stops working

Your fact table has about 57,000 rows, so every query is instant and the dashboard feels fine. A real warehouse has millions. At that size the roll-up you run on every click becomes the slow part, and the answer is to compute it once and store it — a summary table.

That is a real technique, but it solves a problem you do not have yet. Build the dashboard first; add the summary table when it is slow.

Next class

Summary tables and performance. Making a slow warehouse fast, and measuring whether it worked.

The finished dashboard and earlier sheets: dwhdemo.vedara.net