

Working with Data

Session 3 — Can I trust the answer?

Block 5 — Did every record arrive?

Knowledge check 3 — 10 minutes, closed book, 3 questions

- Covers Session 2 and Homework 2: a join's rows, one row per order first, the rows with no partner
- Laptops closed, phones away. On the desk: the paper, a pen, and your one handwritten A4 sheet.
- One scenario, three questions. Each question stands on its own.
- Worth 5% of the final grade
- **This block's lecture is 35 minutes**, not 45: the check comes out of the lecture, never out of the lab
- Hand it in when the timer goes. Then I say the answers, once.

Knowledge check 3 — the answers, said once

Listen, and mark your own memory of what you wrote.

The written key is on Moodle next Monday, with the marking by clause. Your marked paper is at the TA office from the same day.

One question first

Which economy was the largest in 2024?

GDP at purchasing-power parity: each economy counted at what its money buys where it is spent.

Write your answer down. Keep the paper: the World Bank's own file answers it before the break.

One record, one row

An **API** answers a web address with data, in **JSON**: `{...}` is a record of named values, `[...]` a list. We asked the World Bank once and saved every page. One record from page 1:

```
{ "indicator": {"id": "NY.GDP.MKTP.PP.KD", "value": "GDP, PPP (constant 2021 international $)"},  
  "country": {"id": "ZH", "value": "Africa Eastern and Southern"},  
  "countryiso3code": "AFE", "date": "2024", "value": 3153966875412.08,  
  "unit": "", "obs_status": "", "decimal": 0 }
```

The same record, as one row of a table:

iso3	country	year	gdp
AFE	Africa Eastern and Southern	2024	3,153,966,875,412.08

What the setup does

Any **JSON file** is records and lists, and DuckDB can read any of them.

This API puts two things on each page, its metadata and up to 1,000 records, and sends a long answer as several pages.

Our setup reads these pages and turns them into two tables (each a **view**: a saved query you read like a table):

Table	One row is
gdp	one record: a place or a group of places, in one year, with its GDP
gdp_metadata	one page, and what it says about itself: the pages, the records per page, the total

You never write the setup. Your work is to ask the tables a question, and to check the answer.

The table it made: **gdp**

Cell 3: its first five rows.

cell 3

```
con.sql("SELECT * FROM gdp LIMIT 5").df()
```

	iso3	country	year	gdp
0	AFE	Africa Eastern and Southern	2024	3,153,966,875,412.08
1	AFE	Africa Eastern and Southern	2023	3,059,171,608,102.83
2	AFE	Africa Eastern and Southern	2022	2,999,096,418,677.03
3	AFE	Africa Eastern and Southern	2021	2,887,996,518,425.35
4	AFE	Africa Eastern and Southern	2020	2,762,599,295,409.12

Predict: the first row

Cell 4: the five largest in 2024, divided by a trillion and rounded, so they read in trillions.

```
SELECT country, ROUND(gdp / 1e12, 2) AS trillions
FROM gdp
WHERE year = 2024
ORDER BY gdp DESC
LIMIT 5
```

Predict: which name is in the first row?

The first row — what it printed

cell 4

```
con.sql("""  
    SELECT country, ROUND(gdp / 1e12, 2) AS trillions  
    FROM gdp  
    WHERE year = 2024  
    ORDER BY gdp DESC  
    LIMIT 5  
""").df()
```

	country	trillions
0	World	174.16
1	IDA & IBRD total	101.20
2	Low & middle income	91.52
3	IBRD only	91.32
4	Middle income	89.79

World. Then four groups of places. One row of `gdp` is one place *or one group of places*, in one year.

The question from the start: places only

The World Bank's list of countries gives each one a region; a group's region is **Aggregates**. The setup made **places** from that list: 217 rows, one place each. Cell 5: the same query, joined to it. A row of **gdp** stays only if its **iso3** is in **places**: **CHN** is; **WLD**, World, is a group, so it drops out.

```
cell 5 con.sql("""
SELECT g.country, ROUND(g.gdp / 1e12, 2) AS trillions
FROM gdp AS g
JOIN places AS pl ON pl.iso3 = g.iso3
WHERE g.year = 2024
ORDER BY g.gdp DESC
LIMIT 5
""").df()
```

	country	trillions
0	China	33.59
1	United States	25.74
2	India	13.66
3	Russian Federation	6.12
4	Japan	5.89

What the pages say they hold

Cell 6: `gdp_metadata`, one row per page.

cell 6 `con.sql("SELECT * FROM gdp_metadata ORDER BY page").df()`

	page	pages	per_page	total
0	1	4	1000	3975
1	2	4	1000	3975
2	3	4	1000	3975
3	4	4	1000	3975

Every page says the same thing: 4 pages, up to 1,000 records each, **3,975 records in all**.

Predict: how many rows?

Cell 7:

```
SELECT COUNT(*) AS rows FROM gdp
```

Predict: which number from the last slide should the count match, `per_page` or `total` ? And why?

How many rows — what it printed

cell 7 `con.sql("SELECT COUNT(*) AS rows FROM gdp").df()`

	rows
0	3975

3,975: the same as the total. The setup read every page, because of one character in its file name:

`NY.GDP.MKTP.PP.KD_page*.json`

A `*` in a file name matches any text: `page1` , `page2` , `page3` , `page4` . **The file name to remember.**

Predict: page 1 alone

Two tables, both made by the setup: `gdp` from `NY.GDP.MKTP.PP.KD_page*.json`, and `gdp_page1` from `NY.GDP.MKTP.PP.KD_page1.json`. Cell 8: your question again, on `gdp_page1`.

```
make_table("gdp_page1",
           "data/raw/api/worldbank/NY.GDP.MKTP.PP.KD_page1.json",
           "gdp", "DOUBLE")
con.sql("""
SELECT g.country, ROUND(g.gdp / 1e12, 2) AS trillions
FROM gdp_page1 AS g
JOIN places AS pl ON pl.iso3 = g.iso3
WHERE g.year = 2024
ORDER BY g.gdp DESC
LIMIT 5
""").df()
```

Predict: must China still come first? And what would you check before you trusted the answer?

Page 1 alone — what it printed

```
cell 8 make_table("gdp_page1",  
                  "data/raw/api/worldbank/NY.GDP.MKTP.PP.KD_page1.json",  
                  "gdp", "DOUBLE")  
con.sql("""  
    SELECT g.country, ROUND(g.gdp / 1e12, 2) AS trillions  
    FROM gdp_page1 AS g  
    JOIN places AS pl ON pl.iso3 = g.iso3  
    WHERE g.year = 2024  
    ORDER BY g.gdp DESC  
    LIMIT 5  
""").df()
```

	country	trillions
0	Australia	1.64
1	Bangladesh	1.47
2	Argentina	1.22
3	Belgium	0.75
4	Algeria	0.73

Australia, 1.64 trillion. No error, no warning. The same question, on a table that held less.

On your own: did page 1's table get everything?

In your notes, alone: two queries, `SELECT` to the end.

1. The rows page 1's table actually read.
2. The total its page promised.

View	Its columns
<code>gdp_page1</code>	<code>iso3</code> , <code>country</code> , <code>year</code> , <code>gdp</code>
<code>gdp_page1_metadata</code>	<code>page</code> , <code>pages</code> , <code>per_page</code> , <code>total</code>

Page 1's table, counted — what it printed

cell 9

```
display(con.sql("SELECT COUNT(*) AS rows FROM gdp_page1").df())  
con.sql("SELECT page, pages, per_page, total FROM gdp_page1_metadata").df()
```

	rows
0	1000

	page	pages	per_page	total
0	1	4	1000	3975

The page says **3,975 records in all**. The table holds **1,000**. That count is what tells you a page is missing.

Did every record arrive?

1. The setup turns the pages into tables. **Look at what it made**, and say what one row is.
2. The source says how much it sent. **Count your table against it**: 3,975 and 3,975.
3. A table that is missing a page **runs without an error**, and answers the question wrong.

Your team today

Teams of three, as last week, on one work laptop: **keyboard**, **AI** (your own device; nothing from the assistant for the first five minutes), **checker** (says before each query which number must stay the same, rise or fall).

- Teams are the people sitting next to you: three in a row. The same team all day.
- **After the break, everyone takes a role they have not had today.**
- Decide now who is keyboard: that person clones at the break.

5-minute stand-and-stretch — start the Lab 5 clone first

The keyboard person starts the clone and `uv sync`. Then everyone takes the break: it downloads while you are away.

In your terminal (Git Bash on Windows, Terminal on macOS), in the folder where you keep course work:

```
git clone https://github.com/earino/ecbs5294-lab05-json-pages.git  
cd ecbs5294-lab05-json-pages  
uv sync
```

Then leave the laptop. Back in five.

Lab 5 — World population, from the World Bank's pages

Where this lab goes: your team asks the World Bank's population table a real question, and checks that the table holds everything. Then, in section D, a colleague's report for the board says the World Bank has a 2024 population for **19 places**, holding **422,186,027 people**. Nothing errors.

First minute: the clone finished? The keyboard opens the folder in VS Code, opens `notebooks/report.ipynb`, picks the `.venv` kernel, types your names in the *Your team* cell, and runs the three setup cells. Then top to bottom.

Every cell marked *yours* starts empty: your team writes it. A, B, C: your own question and your own check. D: the colleague's report. E: the check, supplied.

The syntax is on one page: `REFERENCE-sql.md`. The team next to you at minute 33. The checkpoint is the notebook, on Moodle.

The last ten minutes — explain it to the team next to you

Two teams, one laptop at a time. At your laptop, finished or not, start with one result you found interesting; then the last step you reached; if you repaired the report, why it was wrong and what the check printed. About two minutes.

The other team asks one question. If you reached the report: **why was it wrong, not just where?** If not: **what is one row of what you showed?** Then swap laptops.

Then the what-if, both teams together: **Belgium's records are on page 1. Does that mean a report built on page 1 alone can give Belgium's change in population since 2010? Which count would tell you?** Commit to an answer before anyone runs anything. Unsure, or the teams disagree? **Hands up.**

The lab ends here, finished or not. Before you leave, the keyboard uploads the notebook as it is to Moodle; everyone else pastes the *Your team* line.