Working With Data

From database concepts to a mini SQL demo

Workshop context:
Coding Café @ Built Environment
Instructor: Özgün Balaban

becodingcafe.com

What is Coding Cafe?

Originating in places like Netherlands eScience Center, Programming Cafés are low-threshold, peer-driven sessions where researchers:

- 👩‍💻 Present a short demo of a programming method or tool
- 🛠️ Ask or offer help with code or workflows
- Create networking between people working on same issues / software
- Co-work informally, with coffee or pizza
- Focus on community over curriculum

Outline

- Working with data: spreadsheets → CSV/JSON → databases
- Why & when databases matter
- Relational model basics (tables, keys, relationships)
- SQL essentials: SELECT, WHERE, ORDER BY, JOIN
- Mini SQL demo with real dataset

How We Usually Start Working With Data

Common formats we all use:

Spreadsheets

- Small datasets, quick editing
- Easy for personal work

Example

name	age	city
Anna	34	Rotterdam
Eli	28	Eindhoven

! What are the potential issues?

Pain points

- What if there are multiple entries for each person like say the books that they write

name	age	city	books
Anna	34	Rotterdam	Anna’s Book 1
Anna	34	Rotterdam	Anna’s Book 2
Anna	34	Rotterdam	Anna’s Book 3
Eli	28	Eindhoven	Eli’s Book 1
Eli	28	Eindhoven	Eli’s Book 2

Pain point

- What if they have a book that they created together

name	age	city	books
Anna	34	Rotterdam	Anna’s Book 1
Anna	34	Rotterdam	Anna & Eli’s Book 1
Anna	34	Rotterdam	Anna’s Book 2
Eli	28	Eindhoven	Eli’s Book 1
Eli	28	Eindhoven	Anna & Eli’s Book 1

What's Really Happening Here?

- Repeated information everywhere
- Hard to update consistently
- No single source of truth
- Difficult to ask questions like:
  - "Which authors wrote books together?"
  - "How many books has each person written?"
- Can be proprietry format, xls, xlsx

CSV file format

Comma-Separated Values

Instead of xls, csv can be used

```
name,age,city,books
Anna,34,Rotterdam,book1
Eli,28,Eindhoven,book2
```

Pros

- Simple, universal, works in every tool
- Easy to generate and read

> ! What are the potential issues?

Limitations

- No nested or relational structure
- No data types (everything is text)
- Hard to manage multiple linked tables
- Breaks when there are , in the data

JSON – Flexible & Nested Structure

JavaScript Object Notation – stores richer objects

```
{
  "name": "Anna",
  "age": 34,
  "location": {
    "city": "Rotterdam",
    "postcode": "3051"
  },
  "books": ["Book A", "Book B"]
}
```

JSON – Flexible & Nested Structure

Pros

- Great for flexible & complex objects
- API-friendly
- Represents nested relationships

Limitations

- Hard to query and analyze at scale
- Can become inconsistent across files
- No shared structure enforcement

Other File Formats for Storing Data

Examples

YAML

```yaml
person:
  name: Anna
  city: Rotterdam
  books:
    - Book A
    - Book B
```

Other File Formats for Storing Data

Examples

XML

```
<person>
  <name>Anna</name>
  <city>Rotterdam</city>
  <book>Book A</book>
</person>
```

Common Issues With Storing Data in Files

- Hard to combine multiple files or datasets
- No enforced structure or validation (easy to break data)
- Difficult to collaborate, overwriting & version chaos
- No relationships between entities (joins become manual)
- Difficult to run complex queries or analytics
- Performance breaks with large files
- Extra tools needed to process and clean data

> 💡 **Files are great for exchange, but not for managing and querying complex data.**

From Files to Databases

When data, collaboration, or complexity grows, file-based storage stops being enough.

> 💡 **Databases provide structure, reliability, performance, and query power.**

What is a Database?

A system for **storing, organizing, and querying** data efficiently.

- Manages structured information
- Maintains relationships between data
- Enables fast searching, filtering, and analytics
- Supports multiple users safely

Different Types of Databases

Type	Best For	Examples
Relational (SQL)	Structured, consistent relationships	PostgreSQL, MySQL, SQLite
Document (NoSQL)	Flexible, evolving data	MongoDB, Firestore, CouchDB
Graph	Complex relationships & networks	Neo4j, JanusGraph
Time-Series	Sensor/IoT, timestamped data	InfluxDB, TimescaleDB
Columnar / Analytical	BI & big data analytics	BigQuery, Redshift, DuckDB

Choosing the Right Database

If you need…	Then consider…
Clean, stable tables with relationships	**Relational / SQL**
Flexible or nested JSON-like structure	**Document / NoSQL**
Network or connection reasoning	**Graph DB**
High-frequency data (sensors, logs)	**Time-series**
Large-scale analytics / dashboards	**Columnar DB**

> 💡 **When in doubt: Start with relational SQL – it teaches core concepts used everywhere.**

Why Not Use a Database for Everything?

- Adds infrastructure complexity
- Requires learning & planning upfront
- Overkill for small or short-lived datasets

> 🎯 **Use databases when structure, consistency, collaboration, or performance matters.**

Relational Model

Data is structured into **tables** with relationships.

Table	What it stores
`Person`	People collecting data
`Site`	Locations
`Visited`	Who visited which site, when
`Survey`	Measurements

Schema Example (Survey Dataset)

```
Person(id, personal, family)
Site(name, lat, long)
Visited(visit_id, site, date, person)
Survey(taken, person, quant, reading)
```

- Each row is a record
- Each column is a field
- Relationships link tables using shared keys (e.g., person)

What is SQL

Structured Query Language – a standard language for asking questions about data.

- Retrieve specific data
- Filter and clean
- Aggregate results
- Join related tables

Basic SQL Query Structure

```
SELECT columns
FROM table
WHERE condition;
```

Example:

```
SELECT * FROM Person;
```

This returns all rows and columns from Person.

Filtering Data: WHERE

```
SELECT family, personal
FROM Person
WHERE family = 'Dyer';
```

Sorting & Removing Duplicates

```
SELECT DISTINCT quant
FROM Survey
ORDER BY quant;
```

Joining Tables

We combine related tables using JOIN.

```
SELECT
  *
FROM
  Site
  JOIN Visited ON Site.name = Visited.site
```

```
SELECT
  *
FROM
  Site
  JOIN Visited ON Site.name = Visited.site
  WHERE name="DR-1"
```

Now we can answer: Who visited site DR-1 and when?

Your Questions – Time for exercise

Beyond Traditional Databases

When data becomes too large or too fast for a single database instance, we use **analytics engines** and **distributed processing** tools.

Tool / Framework	Best For	Examples / Notes
Databricks / Apache Spark	Distributed big-data processing, ML pipelines	Works across clusters, integrates notebooks & PySpark
Polars	Lightning-fast dataframe operations	Rust-based engine, 10–100x faster than Pandas
DuckDB	In-process SQL analytics	“SQLite for analytics”, perfect for medium data
BigQuery / Redshift	Cloud data warehouses	SQL at massive scale
Dask / Ray	Parallel computing	Scale Python workloads

Why These Tools Matter Today

- ⚡ Speed: optimized computation (vectorized, parallel, columnar)
- 📈 Scale: handle billions of rows beyond SQL server limits
- 🤝 Interoperability: integrates with Python, ML tools, BI dashboards
- 🏗️ Hybrid workflows: combine SQL + code + big data pipelines

> 💡 **SQL still matters – it's the language that powers nearly all of these tools.**

When to Move Beyond a Database

Need	Best Option
Analytics on large datasets	DuckDB, Databricks, BigQuery
Fast dataframe transformations	Polars
Cluster-scale ML pipelines	Spark / Databricks
SQL analytics without infrastructure	DuckDB
Complex BI dashboards	Data warehouse (BigQuery, Redshift, Snowflake)

🎯 **Databases organize data — analytics engines unlock insights from it.**