Tabular Database Systems

②

Tabular Data in CSV Files

Summer 2025

Torsten Grust
Universität Tübingen, Germany

1 ¦ Most Data Lives Outside DBMSs

Reality check: The vast majority of data out there does *not* live in files and formats directly accessible for DBMSs:

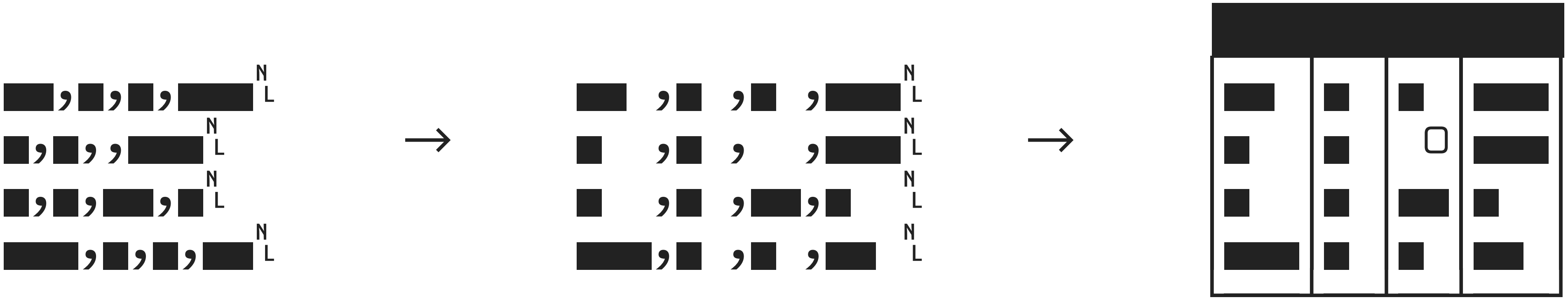
- Applications read/write their specific data formats which were probably devised before DBMS *X* became available.

- Users process their data using a variety of systems. Why lock-in to the propietary database format of DBMS *X* (or *Y* or 🦆)?
  - Rather build on a “lowest common denominator” format that allows **data interchange between systems.**

- Simplicity wins! DBMS-internal data formats are intricate, heavily optimized for access speed and space efficiency.
  - Rather use formats that admit **simple read/write routines.**
  - Let **humans read/write the data format,** ideally using no tools but their favorite **plain text** or spreadsheet editor.

2 ┆ Tabular Data in Comma-Separated Value (CSV) Files ❶

CSV files (*.csv) provide one such lowest common denominator DBMS-external dataformat. CSV encodes tabular data in **line-structured plain text** files:



CSV file
(plain text)

→

CSV file
(separators aligned)

→

intended
database table

1. Newline (␤) separates lines ≣, a **line** encodes one table **row**.
2. Delimiters (often comma ,) split lines into **columns**.
   - We expect each line to hold the same number of columns, leading to a rectangular grid of cells ▦.

Tabular Data in Comma-Separated Value (CSV) Files ❷

CSV file 007-vehicles.csv to define the column names and contents of table vehicles (see Chapter ①):

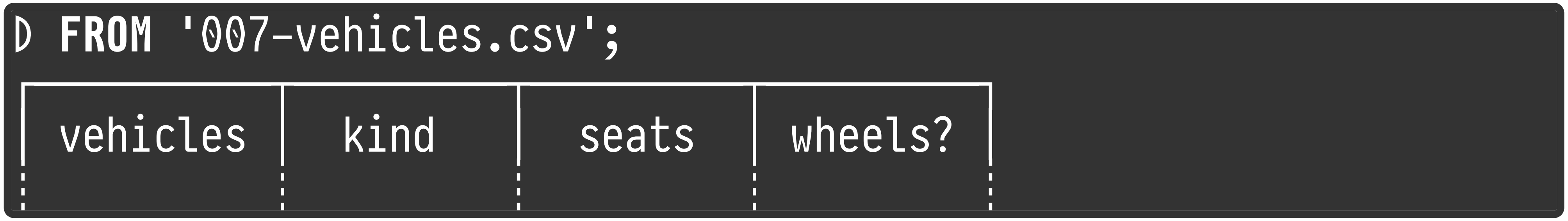
#007

```
vehicle, kind, seats, wheels?
🚗, car, 5, true
🚙, SUV, 3, true
🚌, bus, 42, true
🚐, bus, 7, true
🏍, bike, 1, true
🚜, tank,  , false
🏎, cabrio, 2, true
```

- **Notes:**
  - Column names held in first CSV line “by convention.”
    - Otherwise, **header and data rows are identical.**
  - Columns are **untyped** (wheels? “looks like” type boolean).
  - NULL represented by adjacent delimiters(, ,).

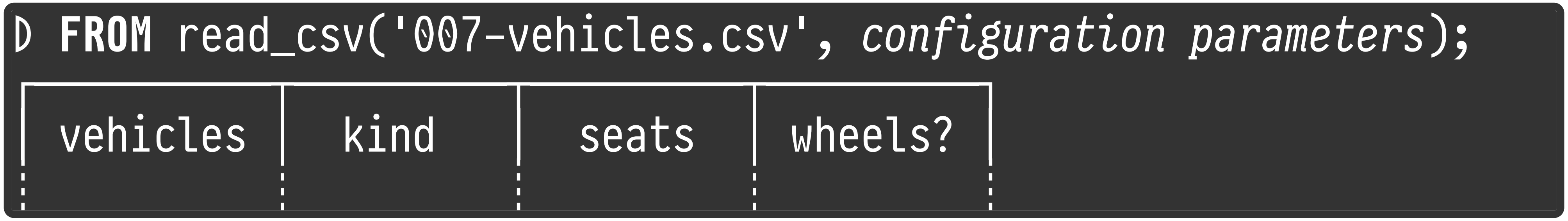
Reading CSV Files Like Database Tables

- In DuckDB, read a CSV file much like a database-internal table:



```
D FROM '007-vehicles.csv';
┌──────────┬───────┬───────┬─────────┐
│ vehicles │ kind  │ seats │ wheels? │
┆          ┆       ┆       ┆         ┆
```

- This is a shorthand for an explicit call to built-in function `read_csv()`:



```
D FROM read_csv('007-vehicles.csv', configuration parameters);
┌──────────┬───────┬───────┬─────────┐
│ vehicles │ kind  │ seats │ wheels? │
┆          ┆       ┆       ┆         ┆
```

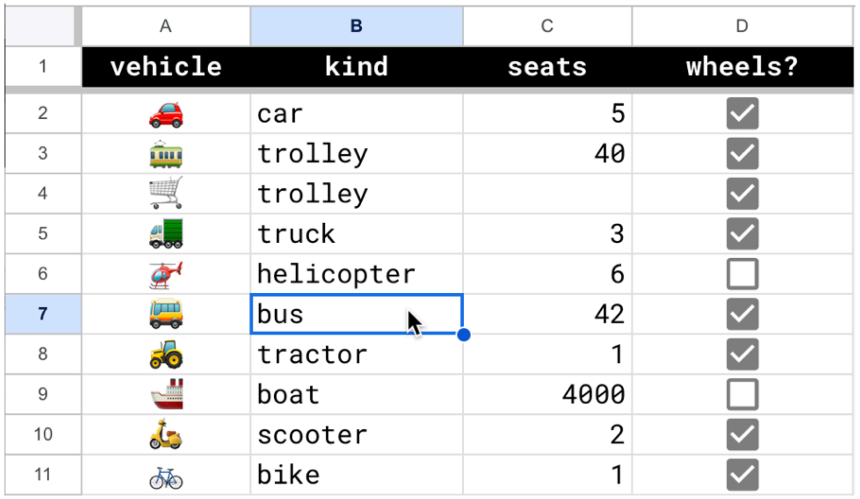
  - Non-default *configuration parameters* may be required to disambiguate CSV file contents regarding presence of headers, choice of delimiters, column types, …)

#008

3 ┆ CSV Enables Data Exchange

A variety of systems export data in CSV format. Paired with the DBMS's CSV import, this enables **CSV-based data exchange.**

- Example: Spreadsheets. On Google Sheets, use command File ▸ ⤓ Download ▸ Comma-Separated Values (.csv):



	A	B	C	D
1	vehicle	kind	seats	wheels?
2	🚗	car	5	☑
3	🚃	trolley	40	☑
4	🛒	trolley		☑
5	🚛	truck	3	☑
6	🚁	helicopter	6	☐
7	🚌	bus	42	☑
8	🚜	tractor	1	☑
9	🚢	boat	4000	☐
10	🛵	scooter	2	☑
11	🚲	bike	1	☑

Access this spreadsheet on Google Sheets

  - ○ **NB.:** On export, **data validation rules** (e.g., number ranges, value constraints ☑☐), formulæ, or cell links **are lost.**

#009

4 ¦ Importing Data Into Tables, Exporting Query Results Into Files

1. Save repeated imports, use COPY to **copy a file into a table:**

   ```
   COPY table
   FROM path_to_file (configuration parameters)
   ```

   ○ Can now query *table* like a regular database table.

2. **Copy SQL query results into a file** external to the DBMS:

   ```
   COPY (query)
   TO   path_to_file (configuration parameters)
   ```

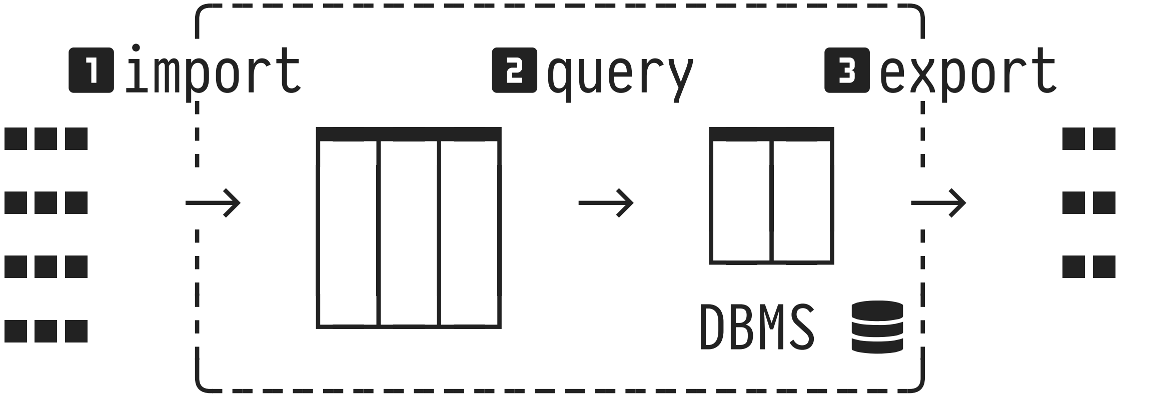
The *configuration parameters* determine file format (e.g., CSV), delimiters, header output, compression, overwrite/append mode, …

Using DuckDB as a CSV Processor

DBMS may act as a processing engine for database-external data:

+ SQL is versatile, expressive, and executed efficiently.
— Need to pay for data import and export.

#010



①

```
D COPY (SELECT   ...
        FROM     input_file
        WHERE    ...
        ORDER BY ...)
  TO   output_file;
```

②

```
D COPY table
  FROM input_file;
D COPY (SELECT   ...
        FROM     table
        WHERE    ...
        ORDER BY ...)
  TO   output_file;
```

5 ¦ Is CSV Too Simple? Ubiquitous, But Fragile

The CSV format is documented in RFC 4180. Still, CSV files in daily practice deviate in a variety of ways[1], including

- choice of **delimiters** (frequent: `,` `;` `|` (pipe) ␉ (tab)),
- **quoting** (enclose columns with significant spaces ␣ in `"` or `'`),
- character **escape** sequences (what if `,` `"` ␤ are in columns?),
- absence/presence of **headers or preamble text** above data rows,
- choice of representation of **missing values** (e.g., `…,,…`), or
- a **mixture of data types** in a column.

DuckDBs aims to auto-adapt to these CSV dialects using a “multi-hypothesis sniffing” algorithm[2].

[1] DuckDB's CSV reader comes with 25+ configuration options. To get an impression of the variety of CSV dialects out there, see the paper Characteristics of Open Data CSV Files (2016).

[2] See the blog post DuckDB's CSV Sniffer: Automatic Detection of Types and Dialects and the paper Multi-Hypothesis CSV Parsing (2017) by Hannes Mühleisen *et al*.

DuckDB's CSV “Sniffer”

1. **Sample 20480 lines** from all over the file (unless we need to read top to bottom, e.g., if input is stdin or compressed).
2. Try **CSV dialects** based on 24 selected combinations of the options below. Choose the CSV dialect options that lead to a *consistent and maximum number of columns* per row:

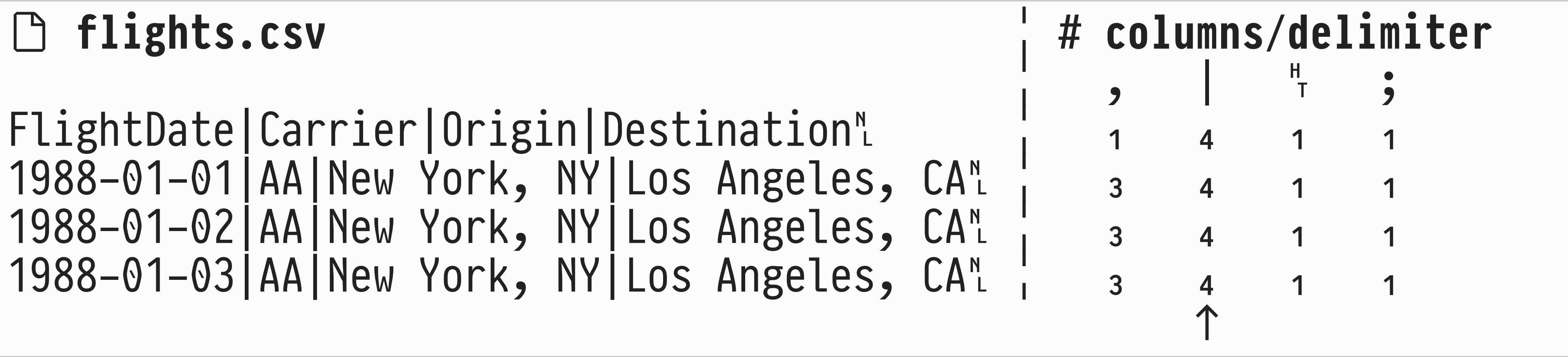
parameter	options
delim	, | ␉ ;
quote	" ' (empty)
escape	" ' \ (empty)

3. **Detect types.** Try to cast values to candidate types *in order*: [(NULL,) boolean, int, double, time, date, timestamp, text].
4. **Header detection:** Do columns in first row match detected types (*is data*, thus generate own header) or not (*is header*)?

DuckDB's CSV “Sniffer”: Dialect and Type Detection

Dialect detection:

#011



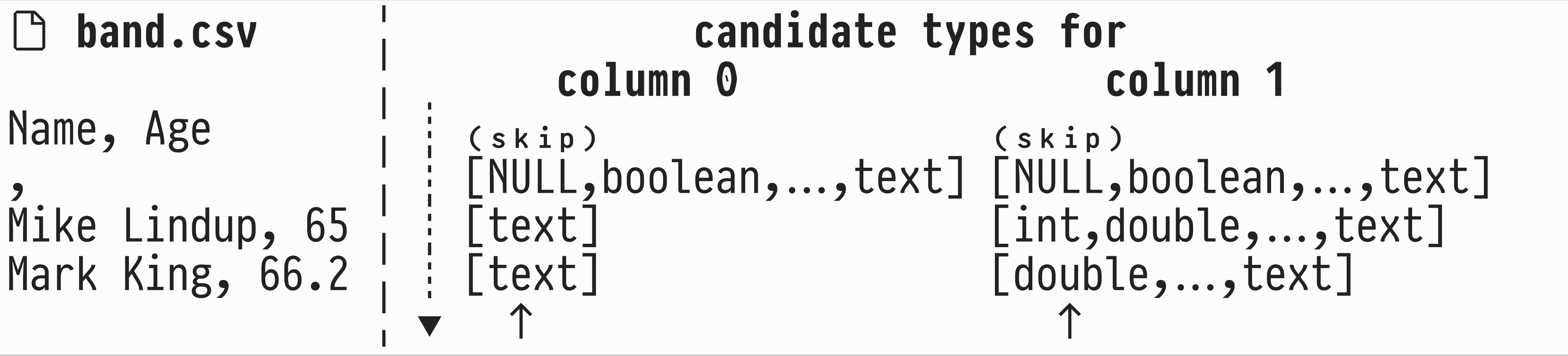
flights.csv

```
FlightDate|Carrier|Origin|Destination␤
1988-01-01|AA|New York, NY|Los Angeles, CA␤
1988-01-02|AA|New York, NY|Los Angeles, CA␤
1988-01-03|AA|New York, NY|Los Angeles, CA␤
```

# columns/delimiter

,	|	␉	;
1	4	1	1
3	4	1	1
3	4	1	1
3	4	1	1
	↑		

Type detection (file reading order ⋯▶):



band.csv

```
Name, Age
,
Mike Lindup, 65
Mark King, 66.2
```

candidate types for

column 0	column 1
(skip)	(skip)
[NULL,boolean,…,text]	[NULL,boolean,…,text]
[text]	[int,double,…,text]
[text]	[double,…,text]
↑	↑