Tabular Database Systems

①

Tabular Data and Database Systems

March 17, 2026

Torsten Grust
Universität Tübingen, Germany

1 Welcome!

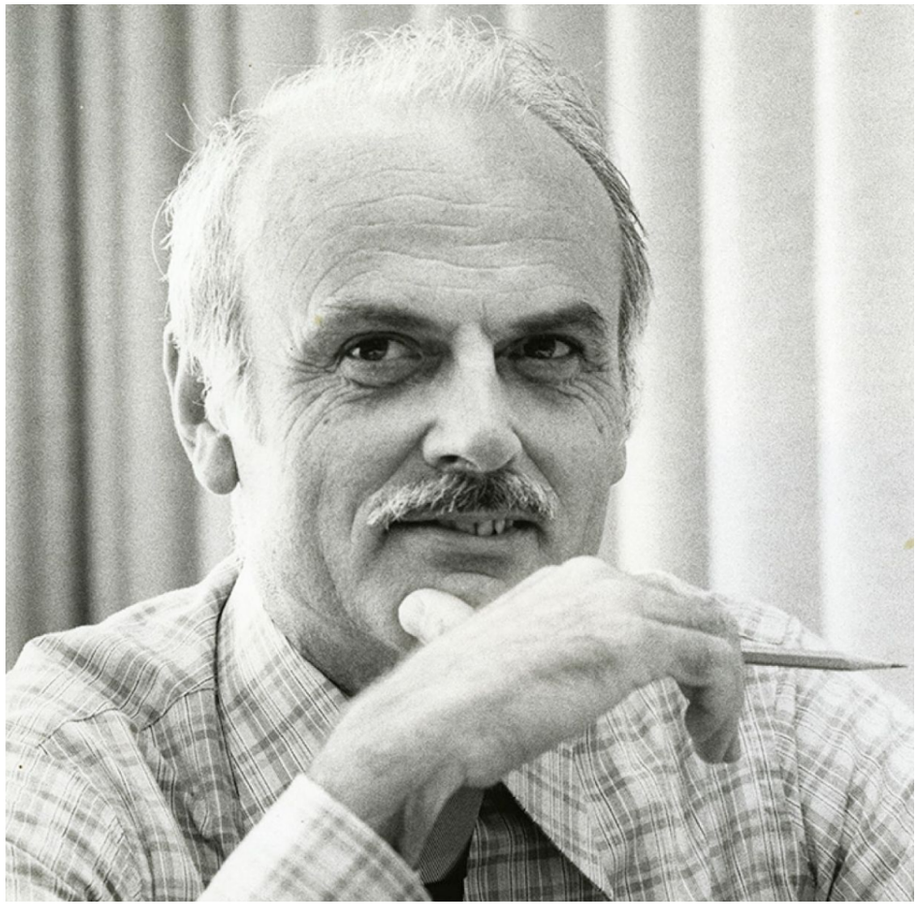
Welcome to the rectangular world of **tabular database systems.**

Reshaping all kinds of data to fit into rows + columns may appear restrictive and arcane (it certainly did to me).

This course will investigate how **tables**...

- ... are a **versatile and flexible data representation** (also for non-tabular data),
- lead to **compact data storage** (in main memory as well as secondary memory), and
- admit **super-efficient data processing.**

Tables are Versatile

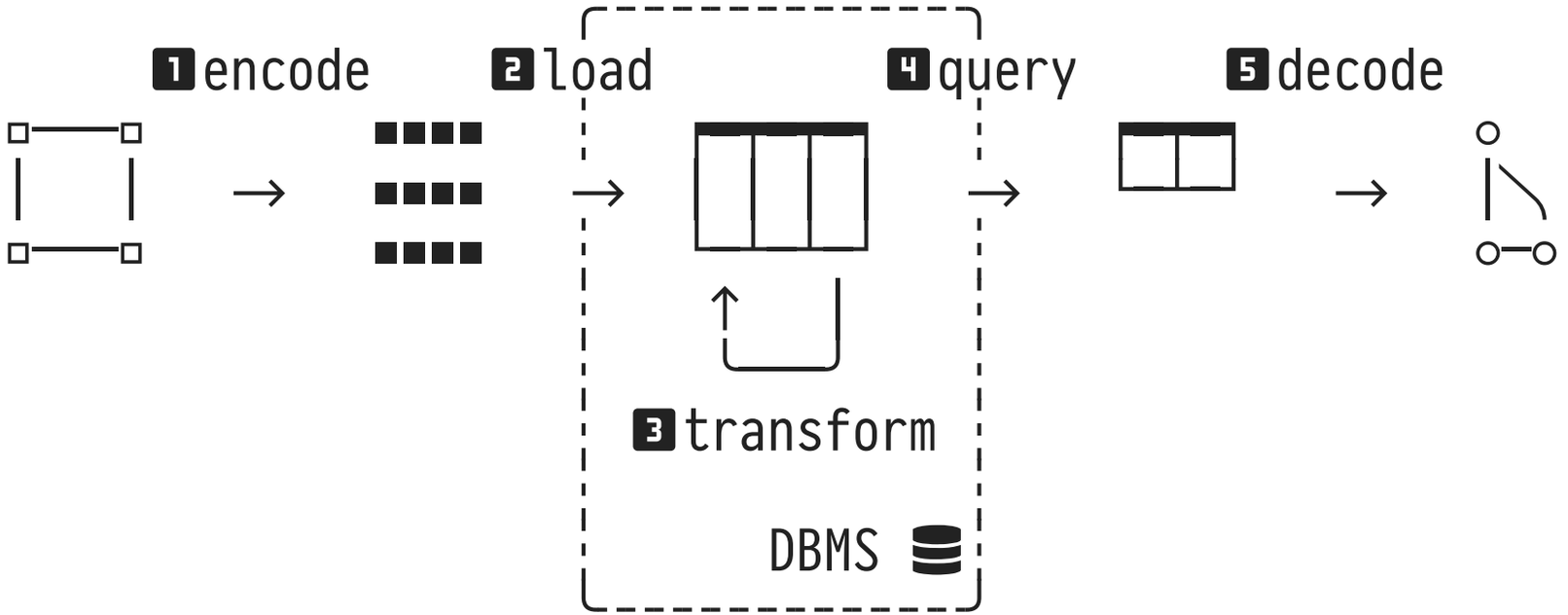


Edgar F. Codd (© IBM)

“Twelve distinct ways to represent data at the logical level are eleven too many.”

— Edgar F. Codd, ACM Turing Laureate

Processing Data Using Tabular Database Systems



- Step 1 reshapes data structures to fit rows + columns ▪▪▪▪.
- Data format ▪▪▪▪ still is external to the **database management system** (DBMS ⬚🛢). Use the DBMS's *load/copy* facilities 2 to turn this external format into in-database tables ◫.
- At 3 and 4 use **SQL** as a language to process tabular data.
- Users or apps may expect the original data structure again 5.

Tabular Database Systems: The Payoff is Substantial

Q: “But are 1️⃣ (encode), 2️⃣ (load), and 5️⃣ (decode) worth it?”
A: “*You bet!*”

- The DBMS-internal data format is **compact** (compression).
- **SQL is a declarative language** whose primary data type are tables. SQL queries 3️⃣+4️⃣ tend to be concise, often elegant.
- DBMS internals rely on regular table structures: **SQL is very efficient,** typically *way* faster than handcrafted programs.
- DBMS **coordinates concurrent access** to its tables—many users may operate on the same tables using well-defined semantics.
- DBMS **safely persists data** under its control, preventing data loss, e.g., through bugs in apps or system outages.

2 ¦ This Course (Tabular Database Systems, short: TaDa or 🎉)

- We will focus on **table-centric data representation and processing.** There are other kinds of DBMSs (e.g., graph-based DBMS: ⚂→¦⚂↻ ⚃¦→⚃), but we will ignore those.
- Whenever we can approach the theory or **pragmatics** of a concept, we typically choose the latter. (Yet, tabular database systems are rooted in a rich and elegant mathematical foundation.)
- We will get our hands dirty using the tabular DBMS **DuckDB** 🦆 and its extensive **SQL** dialect.
- We will explore selected aspects of **DuckDB's internals** (implementation techniques used inside the ⬚).
- We will draw from a variety of data sources and have **fun** along the way! 😎

Torsten Grust?

Time Frame	Affiliation/Position
1989–1994	Diploma in Computer Science, TU Clausthal
1994–1999	Promotion (PhD), U Konstanz
2000	*Visiting Researcher*, IBM (USA)
2000–2004	Habilitation, U Konstanz
2004–2005	Professor Database Systems, TU Clausthal
2005–2008	Professor Database Systems, TU München
since 2008	Professor Database Systems, U Tübingen

- Web: https://db.cs.uni-tuebingen.de/grust
- Bluesky 🦋: @teggy.org
- E-mail: torsten.grust@uni-tuebingen.de

- Feel free to reach out with criticism, bug reports, suggestions for improvement, pats on the back, or simply to say “*Hi!*” 👋

Slides and Further Lecture Material

These **slides** (PDF), **code fragments** (SQL, Python, C), and **sample data** will be uploaded to a GitHub repository:

github.com/DBatUTuebingen/TaDa

- Slides point to relevant code files or extra material using tags like `#001`:
  - Refers to a file named `001-*` on the GitHub repository (e.g., `001-create-table.sql`).

- **NB.** Code and extra material provide essential content (e.g., details on SQL syntax and semantics).
  - + = **:** Only slides + code provide a complete picture.

Material

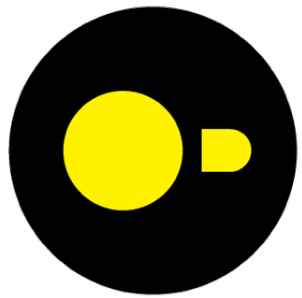
This course is *not* based on a single textbook. Rather, we build on

- a variety of scientific papers,
- textbook excerpts (few),
- the DuckDB documentation at https://duckdb.org/docs/,
- Python/C/C++ code snippets (our own and from inside the),
- blog posts from a range of authors,
- SQL references/standards,
- experience, and best practices.

There is a plethora of books on tabular DBMSs (both usage and internals), sample SQL snippets (quizzes, puzzles, and idioms), or performance tweaks. If we will use such sources, we will provide pointers.

Get Your Hands Dirty: Install DuckDB!

The tabular DBMS **DuckDB** will be the primary tool in this course:



DuckDB

https://duckdb.org, version 1.5 (March 2026: 1.5.0)

- Implements an extensive SQL dialect, is highly performant, open to contributions, and generally awesome.
- Straightforward to install and use on macOS , Windows , Linux (x86 + ARM).

  No DuckDB CLI (SQL prompt/REPL) on iOS or Android.[1]

[1] Run the DuckDB CLI in the web browser: https://shell.duckdb.org. Suffices for quick SQL experiments.

The **tabular data model** arranges all information in a rigorous grid-like fashion. Consider table vehicles below:

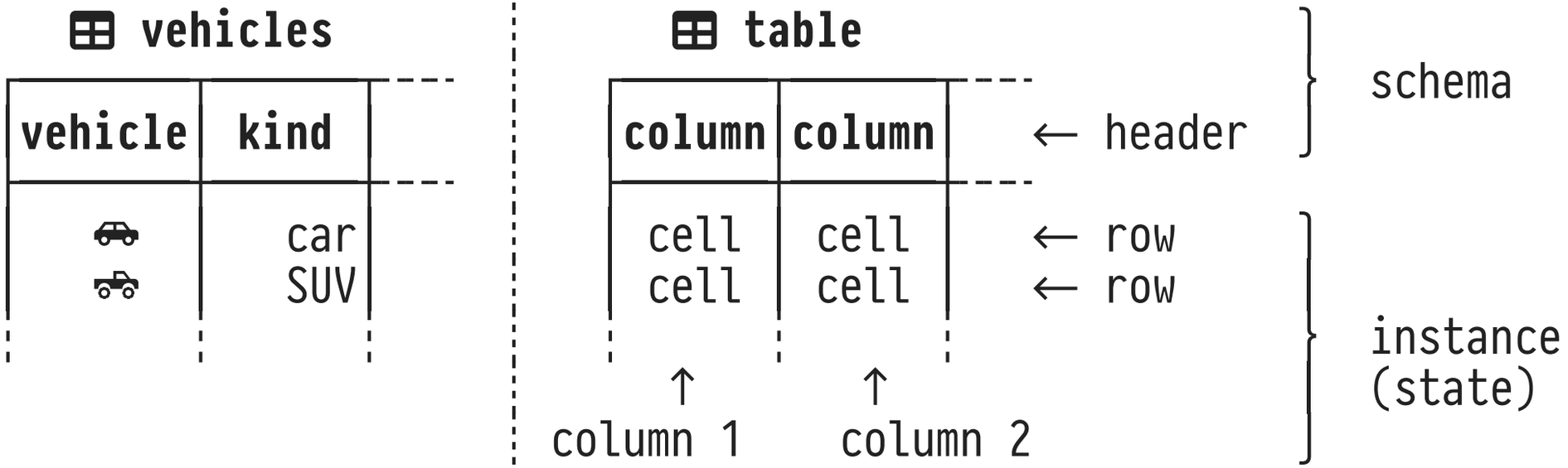
vehicles

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

- This **table** has four **columns**, seven **rows**.
- Table and columns are **named**, rows are not.
- Each column holds **cell** values of a single **type**.
- Cell value **NULL** (▢) signals absence of information.

- Tables are vertical: feature **few columns** ($\leqslant 20$ is typical), but may contain **large numbers of rows** (10–10^6 rows are typical).

Tabular Data Model: Terminology



- Each **column** $i \in \{1,2,...\}$ is assigned a **name** c_i and its **type** τ_i.
- **Types** τ_i are simple (text, int, boolean, float, int[], ...): **first normal form** or **1NF** (columns may *not* hold nested tables).
  - *All* cell values in column i are of type τ_i (NULL if allowed).
- The **schema** determines the structure of a table (say t):
  $t(c_1\ \tau_1,...,\ c_n\ \tau_n)$, in short: $t(c_1,...,c_n)$. Typically *fixed*.
  Ex: vehicles(vehicle text, kind text, seats int, "wheels?" boolean).
- The **instance** is the bag of rows held in the table. *Dynamic*.

SQL: Creating Tables

- The SQL statement CREATE TABLE $t(c_1\ \tau_1,\ldots,c_n\ \tau_n)$ creates
  - a table with given **schema** $t(c_1\ \tau_1,\ldots,\ c_n\ \tau_n)$ and
  - an **empty instance** (no rows yet).

- In the DuckDB CLI:

optional — table name

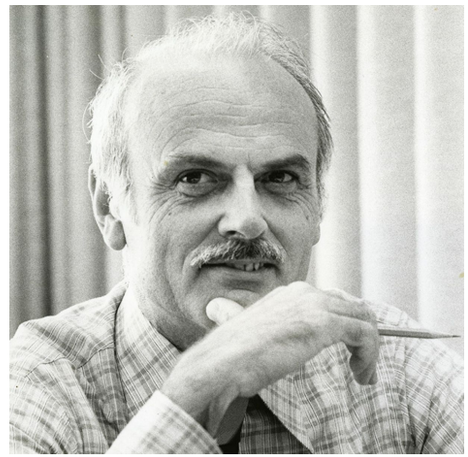
```
D CREATE OR REPLACE TABLE vehicles (
    vehicle   text     NOT NULL,
    kind      text     NOT NULL,
    seats     int                ,
    "wheels?" boolean  NOT NULL
  );
```

columns — types — constraints [optional]

refers to code/extra material in a file named 001- →* #001

Tables vs. Relations ❶

Origin and foundation of the tabular data model is the **relational model** (Edgar F. Codd, 1970).



Edgar F. Codd (© IBM)

Tables and **relations** (subsets of the Cartesian product of domains) are closely related.

- Relation $<$ (*less than*) on integers:
  $< \equiv \{ (x,y) \mid x \in \mathbb{Z}, y \in \mathbb{Z}, \exists n \in \mathbb{N}, n \neq 0 : x+n = y \} \subseteq \mathbb{Z} \times \mathbb{Z}$
  - Notation: $(x,y) \in <$ or $\underline{<}(\underline{x},\underline{y})$ or $x < y$, e.g., $<(0,42)$

- Can define relations **intensionally** (as above) or **extensionally.**
  Game *Rock*, *Paper*, *Scissors* (finite domain $\mathbb{RPS} = \{✊,✋,✌\}$, with extensional "*beats*" relation: $x < y \Leftrightarrow y$ *beats* x):

$<$ ≡

x	*y*
✌	✊
✋	✌
✊	✋

extent of $<$ (a subset of $\mathbb{RPS} \times \mathbb{RPS}$):
lists the set of all ordered pairs included in $<$

Tables vs. Relations ❷

relation <

{ (✌,👎),
(✋,✌), ← tuple (pair)
(👎,✋) }

set of tuples
(⊆ ℝℙ𝕊 × ℝℙ𝕊)

- ✌ ∈ ℝℙ𝕊, 👎 ∈ ℝℙ𝕊, ✋ ∈ ℝℙ𝕊
- tuple components ordered
- no order among tuples
- no duplicate tuples

beats

lose	win
✌	👎
✋	✌
👎	✋

← row

bag (multiset) of rows

- **lose** :: text, **win** :: text
- columns ordered
- no order among rows
- possibly duplicate rows

- Read symbol :: as “*has type*”.

#002

Identifying Rows in Tables: Keys 🔑

Tables are unordered: cannot refer to rows by position (~~1st/2nd/… row~~). Instead, use **cell values that uniquely identify its row.**

vehicles

vehicle	kind	seats	wheels?
🚗	car	5	true
🚙	SUV	3	true
🚌	bus	42	true
🚐	bus	7	true
🚲	bike	1	true
🚜	tank	▢	false
🏎	cabrio	2	true
↑ unique	↑ ~~unique~~	↑ ⁇	↑ ~~unique~~

- A **key** is a column (combination) whose values identify rows:

  *"Return the row with a **vehicle** value of 🚐."*

- **vehicle** is a key for **vehicles.** Columns **kind** or **wheels**? are not.

⁇ Column **seats** is not:
1. column contains **NULL** and
2. will values remain unique once more vehicles are added?

- A table *t* may contain multiple candidate key columns. Choose one of these candidates to be **the PRIMARY KEY for** *t*.

#003

Identifying Rows in Tables: Good Keys

Choosing good primary keys is vital in **database schema design:**

1. Key values identify rows uniquely in **any table state**
   (no matter how many rows will be added in the future).
   - Key values should be immutable during row lifetime.

2. Key columns will be used in `WHERE` filter predicates:
   - Aim for **narrow keys** (ideally: single-column).
   - Prefer key columns whose values can be **compared efficiently**
     (e.g., prefer type `int` over `text`, `date`, or arrays).

3. Consider **introducing additional/artificial key columns** if the
   application domain does offer natural/narrow/efficient keys.
   - Examples: book ISBNs, product EANs, social security IDs.

Foreign Keys: Identifying (or: Pointing to) Rows in *Other* Tables

Splitting data between tables helps

- to **avoid redundancy** (and thus data integrity issues), and
- to design tables focused on specific **domain concepts.**

vehicles

vid	vehicle	kind	seats	wheels?	driver
v_1		car	5	true	p_4
v_2		SUV	3	true	p_4
v_3		bus	42	true	▢
v_4		bus	7	true	▢
v_5		bike	1	true	p_2
v_6		tank	▢	false	p_3
v_7		cabrio	2	true	p_4

peeps

pid	pic	name	born
p_1		Cleo	2013
p_2		Bert	1968
p_3		Drew	▢
p_4		Alex	2002

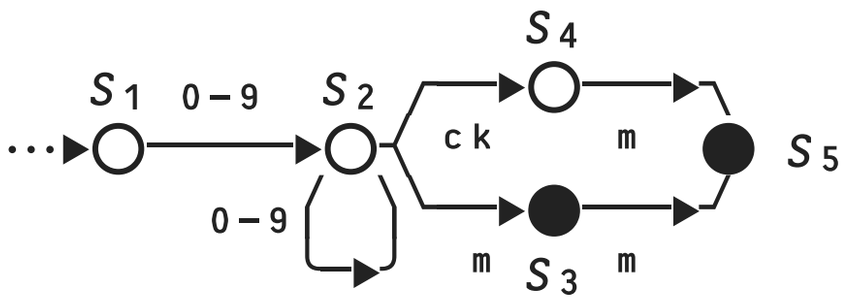
↑

foreign key (references rows in **peeps**, ⊆ peeps(pid))

- Notation: vehicles(vid, vehicle, ..., driver → peeps).

#005

Example: Two Tables Represent a Finite State Machine (FSM)



- FSM accepts metric lengths: '2mm', '135km', '3cm', '42m'.
- Character labels define deterministic transitions.

states

state	start?	final?
s_1	true	false
s_2	false	false
s_3	false	true
s_4	false	false
s_5	false	true

transitions

from	to	labels
s_1	s_2	[0,...,9]
s_2	s_2	[0,...,9]
s_2	s_3	[m]
s_2	s_4	[c,k]
s_3	s_5	[m]
s_4	s_5	[m]

Q: How would you define the **keys**?

Q: Identify **foreign keys** (if any).

Q: Provide SQL **CREATE TABLE** statements for both tables.

4 ┆ *Duck*DB? 🦆?

In case you were wondering:

DuckDB has been named after *Wilbur, the Duck*, which has been living as a pet with Hannes Mühleisen[2]—co-inventor of DuckDB with Mark Raasveldt—on Hannes' houseboat in Amsterdam.

Hannes (CEO) and Mark (CTO) run DuckDB Labs, a company that provides support and consultancy services around DuckDB. The labs are located in Amsterdam, The Netherlands.

Hannes and Wilbur (© Hannes Mühleisen)

[2] Hannes originally is from the Stuttgart area. Back then his car had the license plate S QL 1337.