

Tabular Database Systems

①

Tabular Data and Database Systems

Summer 2025















Torsten Grust
Universität Tübingen, Germany

1 : Welcome!

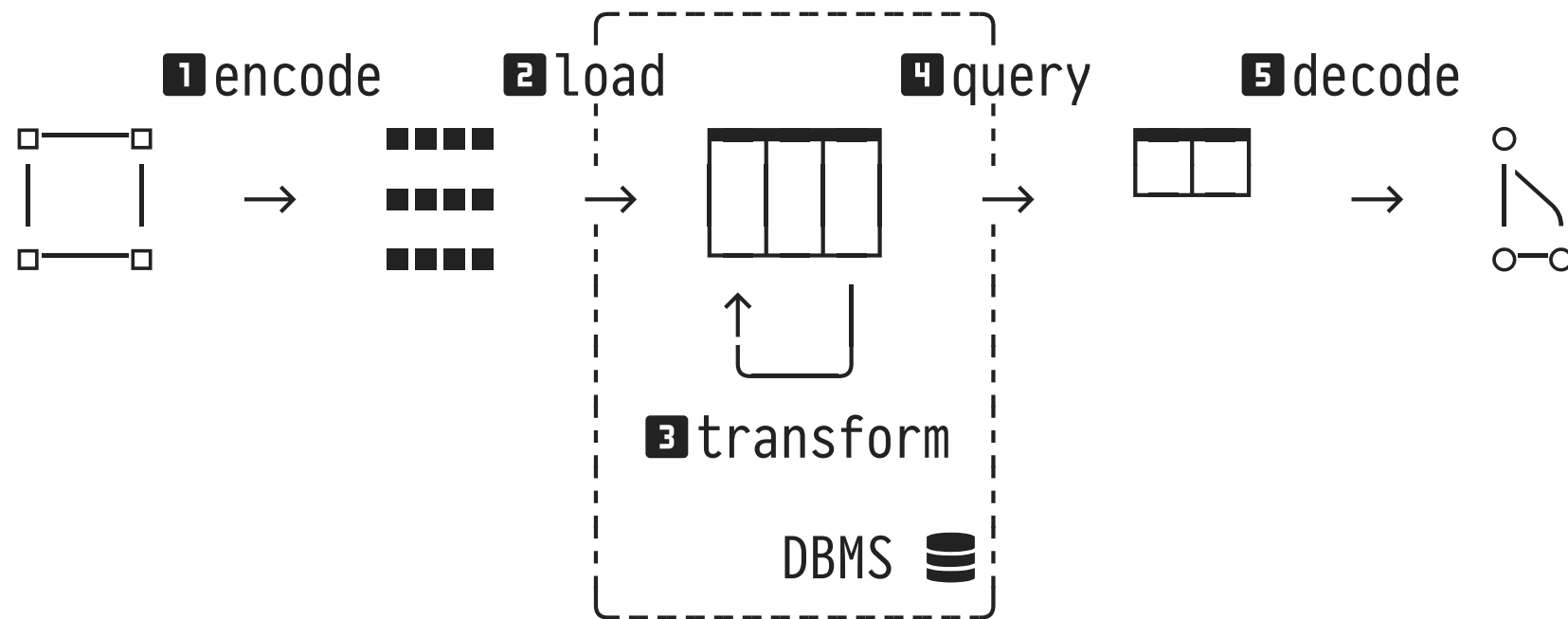
Welcome to the rectangular world  of **tabular database systems**.

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**.

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 **1** (encode), **2** (load), and **5** (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 **3**+**4** 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.


Torsten Grust?

Time Frame	Affiliation/Position
1989–1994	Diploma in Computer Science, TU Clausthal
1994–1999	Promotion (PhD), U Konstanz
2000	<i>Visiting Researcher</i> , 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>
- Office: WSI, Sand 13, Room B318
- Bluesky 🦋: [@tegggy.org](https://bsky.app/profile/tegggy.org)
- Best bet is to catch me on the [DB group's Discord server](#) 

Administrivia





Weekday/Time	Slot	Room
Monday, 10:15–11:45 ☹	Lecture	N14 (Morgenstelle)
Wednesday, 12:15–13:45 ☹	Tutorial	A301 (Sand 13)

-  **No lecture/tutorials on**
 - Mon, April 21 (Easter Monday)
 - Mon, June 9 (Whitsun Break)
 - Wed, June 11
 - Wed, June 25 (Conference SIGMOD 2025)
- **End-Term Exam (6 ECTS)**
 - Written exam on Mon, July 28, 10:00–12:00 (Room *tbd*)
 - Score $\geq \frac{2}{3}$ of the overall assignment points to be admitted to the exam.



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-Teaching/tada-ss25 

- 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.

Weekly Assignments & Tutorial Sessions

- We will distribute, collect, and grade **weekly assignments** (Wednesday→Wednesday) via GitHub .
- You work on these in **teams of two**. Hand-in again via GitHub .

Organized and run by **Denis Hirn**:

- Web: <https://db.cs.uni-tuebingen.de/team/members/denis-hirn/>
- E-Mail: db-lehre@cs.uni-tuebingen.de
- Office: WSI, Sand 13, Rooms B314
- Find us on the [Discord server](#) 

Assignments start once we have collected the first batch of interesting material, probably on Wednesday, April 23.

Discord

During this summer semester, the **TaDa Discord** is *the* course hub:

<https://db.cs.uni-tuebingen.de/discord> 

- ⚠ **Registration** (do it!): use `/verify` with your e-mail address
- ❓ Questions and answers (do *no* post complete solutions)
- ↓ Download additional code examples (e.g., SQL snippets)
- 💬 General discussion
- ⌚ Quick turnaround (responses often within minutes)

TaDa's Course Homepage

<https://db.cs.uni-tuebingen.de/teaching/ss25/tada/> 

- **Organizational matters**



Curriculum. General announcements regarding the lecture, exams, rooms, or dates. (Less important this summer.)

- **Contact information**

Turn to Discord first. But feel free to send e-mail if you seek specific help/need to discuss personal issues with us.

Material

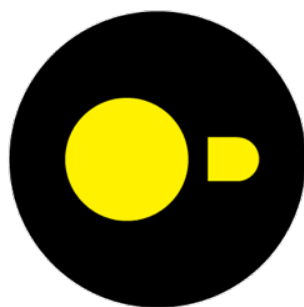
This course is *not* based on a single textbook. Rather, we build on

- 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 1.2.x (April 2025: 1.2.1)

- 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.

3 | The Tabular Data Model

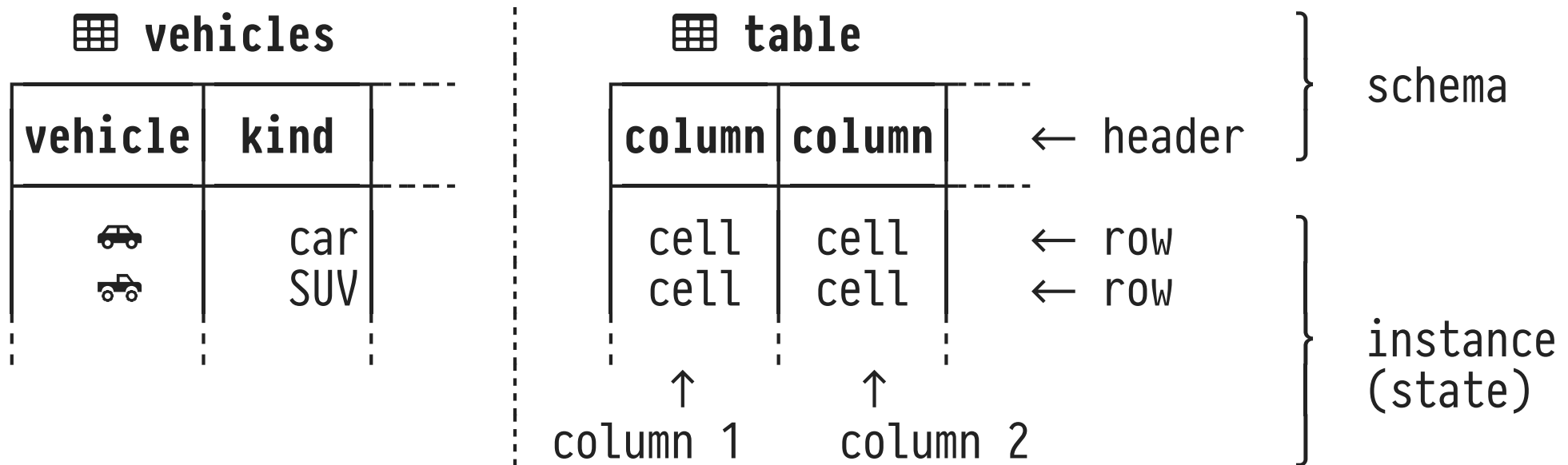
The **tabular data model** arranges all information in a rigorous grid-like fashion. Consider table `vehicles` below:

 **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** (≤ 20 is typical), but may contain **large numbers of rows** (10 – 10^6 rows are typical).

Tabular Data Model: Terminology




- Each **column** $i \in \{1, 2, \dots\}$ 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):

$t(c_1 \ \tau_1, \dots, c_n \ \tau_n)$, in short: $t(c_1, \dots, c_n)$. 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 $t(c_1 \tau_1, \dots, c_n \tau_n)$` creates
 - a table with given **schema** $t(c_1 \tau_1, \dots, c_n \tau_n)$ and
 - an **empty instance** (no rows yet).
- In the DuckDB  CLI:

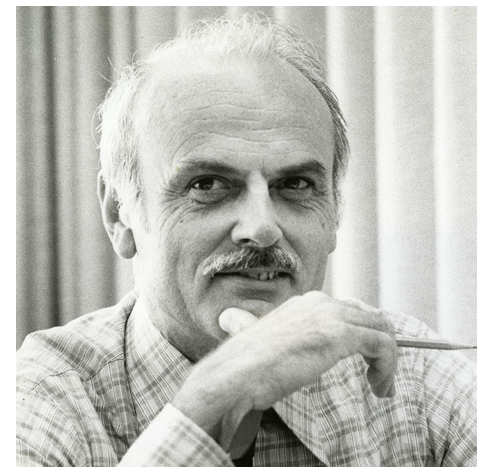
	<u>optional</u>		table name
			↓
<pre>D CREATE OR REPLACE TABLE vehicles (</pre>			
	vehicle	text	NOT NULL,
	kind	text	NOT NULL,
	seats	int	,
	"wheels?"	boolean	NOT NULL
<pre>);</pre>			
↑	↑	↑	
columns	types	constraints [optional]	

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

Tables vs. Relations ①

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

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



Edgar F. Codd (© IBM)

- 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 $\underline{\leq(x,y)}$ or $x < y$, e.g., $<(0,42)$

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




$$< \equiv \left. \begin{array}{c|c} \mathbf{x} & \mathbf{y} \\ \text{✊} & \text{✊} \\ \text{✋} & \text{✂} \\ \text{✂} & \text{✋} \end{array} \right\} \begin{array}{l} \text{extent of } < \text{ (a subset of } RPS \times RPS\text{):} \\ \text{lists the set of all ordered pairs included in } < \end{array}$$

Tables vs. Relations ②

relation <



$\{$
,
,
,
,
,

 $\}$
 ← tuple (pair)

{
 set of tuples
 (\subseteq RPS \times RPS)

-  \in RPS,  \in RPS,  \in RPS
- tuple components ordered
- no order among tuples
- no duplicate tuples

- Read symbol **::** as “has type”.

 **beats**

lose	win
	
	
	

← row








{
 bag (multiset) of rows

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

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	0	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*.

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 to 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	\square
v_4		bus	7	true	\square
v_5		bike	1	true	p_2
v_6		tank	\square	false	p_3
v_7		cabrio	2	true	p_4

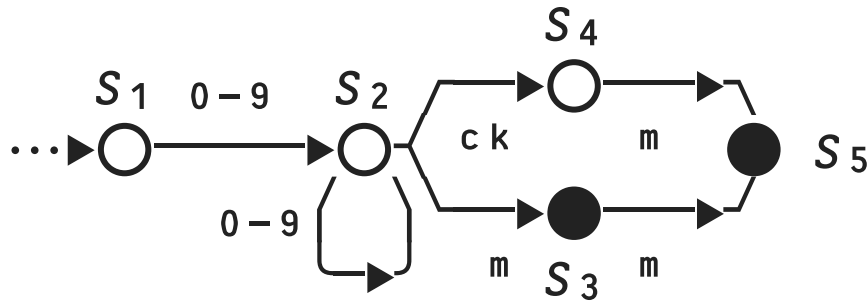
peeps

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

↑
foreign key (references rows in peeps, \subseteq peeps(pid))

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

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 : DuckDB? 🦆?

In case you were wondering:

DuckDB has been named after *Wilbur*, the *Duck*, which has been living as a pet with Hannes Mühleisen²—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 he used the license plate .