Tabular Database Systems

⑧

SQL: Grouping + Aggregation and Functional Dependencies

April 7, 2026

Torsten Grust
Universität Tübingen, Germany

1 ¦ SQL: Grouping (Clause GROUP BY)

- Up to now, SQL queries operate *row-by-row*:
  - **FROM** *t* binds *individual rows* of *t* to row variables,
  - **WHERE** *p* drops row var bindings that do not satisfy *p*,
  - **SELECT** *e*,...,*e* evaluates *e*,...,*e* for each row var binding.

Clause **GROUP BY** e_1,...,e_n identifies **subtables (or: groups) of rows that yield the same value for expressions (criteria)** e_1,...,e_n:

vehicles

vehicle	kind	seats	wheels?	driver	groups
🏍	bike	1	true	p_2	
🚙	cabrio	2	true	p_4	true
🚜	SUV	3	true	p_4	
🚗	car	5	true	p_4	
🚌	bus	42	true	▢	false
🚐	bus	7	true	▢	
🚜	tank	▢	false	p_3	▢

```
SELECT   ···
FROM     vehicles AS v
GROUP BY v.seats < 5
         └──────────┘
       grouping criterion
```

∀ rows v in this group:
v.seats < 5 ≡ false

SQL: Grouping Requires Aggregation

💡 After grouping, use **aggregation** to represent group contents:

1 ⊞ **vehicles** (grouped)

vehicle	⋯	**seats**	⋯	**driver**
🏍		1		p_2
🚗	⋯	2	⋯	p_4
🚙		3		p_4
🚘		5		p_4
🚌	⋯	42	⋯	▢
🚐		7		▢
🚜	⋯	▢	⋯	p_3

2 ⊞ **vehicles** (grouped + aggregated)

seats<5	**count**(*)	**max**(**seats**)	**list**(**driver**)
true	3	3	$[p_2, p_4, p_4]$
false	3	42	$[p_4, ▢, ▢]$
▢	1	▢	$[p_3]$

- Tables are in 1NF. Cells cannot contain groups/subtables. Thus:
  - SQL never provides access to the grouped table **1** itself.
  - **Aggregation after grouping 2 is mandatory.**

01 #047

GROUP BY Reduces Granularity (Row-by-Row to Group-by-Group)

- The **FROM** 1 and **WHERE** 2 clauses operate row-by-row.
- After **GROUP BY** 3, the **SELECT** 4 clause operates group-by-group and yields **one row per group:**

```
SELECT   expr₁,...,exprₙ,
         agg,...,agg
FROM     t
WHERE    p
GROUP BY expr₁,...,exprₙ
```

```
3 } g
4 }
1 n (= |t|)
2 m (⩽ n)
3 g (⩽ m)
  # rows
```

#048

Non-aggregate $expr_i$ in **SELECT** is OK only if (the DBMS can statically deduce that) $expr_i$ is *constant within each group*.

- **Grouping Quiz:** What can you infer if you observe the following?
  Ⓐ $g = 1$. Ⓑ $g = m$. Ⓒ $g = 0$.
  Ⓓ Adding a new $expr_j$ to the **GROUP BY** clause does not change g.

Controlling Granularity: Derived Grouping Criteria

```
SELECT t.c,agg,...,agg                                  #049
FROM   t
GROUP BY t.c -- criterion c leads to too many groups... ¯\_(ツ)_/¯
```

1. **Division.** Group numeric c into buckets of **equal width** $1/N$:
   - If c is integral (int): **GROUP BY** c // N
   - Otherwise: **GROUP BY** round(c / N)
2. **Modulus.** Put integral c into one of N buckets (**order lost**):
   - **GROUP BY** c % N
   - If $N = 2^n$: **GROUP BY** c & $(1 << n) - 1$ (based on n low bits)
3. **Grading.**[1] Place ordered c into buckets with **given borders** b_i:

[1] *Grading* is derived from APL's ⍋ ("grade up"), list_grade_up() in DuckDB: list_grade_up([30,10,20]) = [2,3,1] are the list indices that would rearrange [30,10,20] in ascending order.

A Grouping Playground: The New York City Taxi Database

The *Yellow Cab* database[2] published by the NYC Taxi & Limousine Commission (TLC) provides a fun playground for **GROUP BY** queries.

A DuckDB database file holds the dataset for year 2024: #051

rides	main table, one row per taxi trip
zones	division of New York City into 265 zones (“boroughs”), rides start/stop here
central_park_weather	temperature/wind/precipitation data for Central Park, one row per day

Find plain + embedded SQL queries over this data in #050 #052

[2] Published monthly on the TLC Trip Record Data page, (formerly in CSV, since 2022 in Parquet format).

2 ¦ Group Preservation

```sql
-- How many peeps can each driver give a lift?
SELECT p.pid AS driver, ▨▨▨▨▨▨▨▨,
       max(v.seats) - 1 AS "can lift"
FROM   vehicles AS v NATURAL JOIN peeps AS p
GROUP BY p.pid, ▨▨▨▨▨▨▨▨;
```

#053

vehicles ⋈ peeps

vehicle	kind	seats	wheels?	pid	pic	name	born
	bike	1	true	2		Bert	1968
	tank	▢	false	3		Drew	▢
	car	5	true	4		Alex	2002
	SUV	3	true	4		Alex	2002
	cabrio	2	true	4		Alex	2002

- Adding p.name (pic, born) to GROUP BY ▨ **preserves groups:** rows that agree on pid, also agree on name: pid → name.

Functional Dependencies (FDs)

Given table $t(c_1,..,x,..,y,..,c_n)$, the **functional dependency (FD)** $x \to y$ ("*column x determines column y*") holds in t iff

$$\forall \text{ rows } r_1,r_2 \in t\text{: } r_1.x = r_2.x \implies r_1.y = r_2.y.$$

- **Notes on FDs:**
  - FD $x \to y$ indicates that t **embeds a lookup table** $t^f(\underline{x},y)$ defining a function $f(x) = y$.[3] Embedded table t^f has key x.
  - Generalization: $x_1\ x_2\ ..\ x_n \to y$ defines an n-ary function.
  - If $x \to y$ and $y \to z$, then $x \to z$ (transitivity, like $g \circ f$).
  - Shorthand notation: $x \to y$ and $x \to z \equiv x \to y\ z$.
  - A key 🔑 k for table $t(\underline{k},c_1,..,c_n)$ is like a special, powerful FD: $k \to c_1\ ..\ c_n$ (key k determines all columns).
  - Trivial FD (holds for any column x in any table t): $x \to x$.
  - $x \to y$ does *not* generally imply $y \to x$.

01 #053

[3] Example: For the FD pid → pic in vehicles ⋈ peeps, we have $f(2)$ = 😳, $f(3)$ = 😎, $f(4)$ = 🥵 (3×).

SQL: HAVING (and a Test for Functional Dependencies)

- SQL clause **HAVING** is evaluated *after* groups have been formed. Predicate q thus may use aggregates to **filter entire groups:**

```
SELECT   expr,...,expr,          ┐ 3 ┐
         agg,...,agg             │ 4 ┘ h
FROM     t                       │ 1   n (= |t|)
WHERE    p                       │ 2   m (⩽ n)
GROUP BY expr,...,expr           │ 3   g (⩽ m)
HAVING   q                       │ 4   h (⩽ g)
                                 └──   # rows
```

- A **HAVING**-based test: Does FD $x \to y$ hold in table t? #054

```
SELECT DISTINCT 'x ↛ y' AS "FD violated?"  ┐
FROM   t                                   │ empty result
GROUP BY x                                 │ indicates
HAVING count(DISTINCT y) > 1               ┘ that x → y does hold
```

3 ¦ Apps Define FDs …

trips

trip	weekday	driver	vehicle	from	to	dist	via	hop#
t_1	Mon	4	🚙	Alton	Corby	150	Alton	1
t_1	Mon	4	🚙	Alton	Corby	150	Luton	2
t_1	Mon	4	🚙	Alton	Corby	150	Corby	3
t_2	Tue	2	🏍	Derby	Eaton	17	Derby	1
t_2	Tue	2	🏍	Derby	Eaton	17	Eaton	2
t_3	Wed	2	🏍	Derby	Crich	23	Derby	1
t_3	Wed	2	🏍	Derby	Crich	23	Crich	2
t_4	Thu	4	🚙	Alton	Corby	150	Alton	1
t_4	Thu	4	🚙	Alton	Corby	150	Luton	2
t_4	Thu	4	🚙	Alton	Corby	150	Corby	3

Schedule of multi-hop road trips taken (with drivers and vehicles in use).

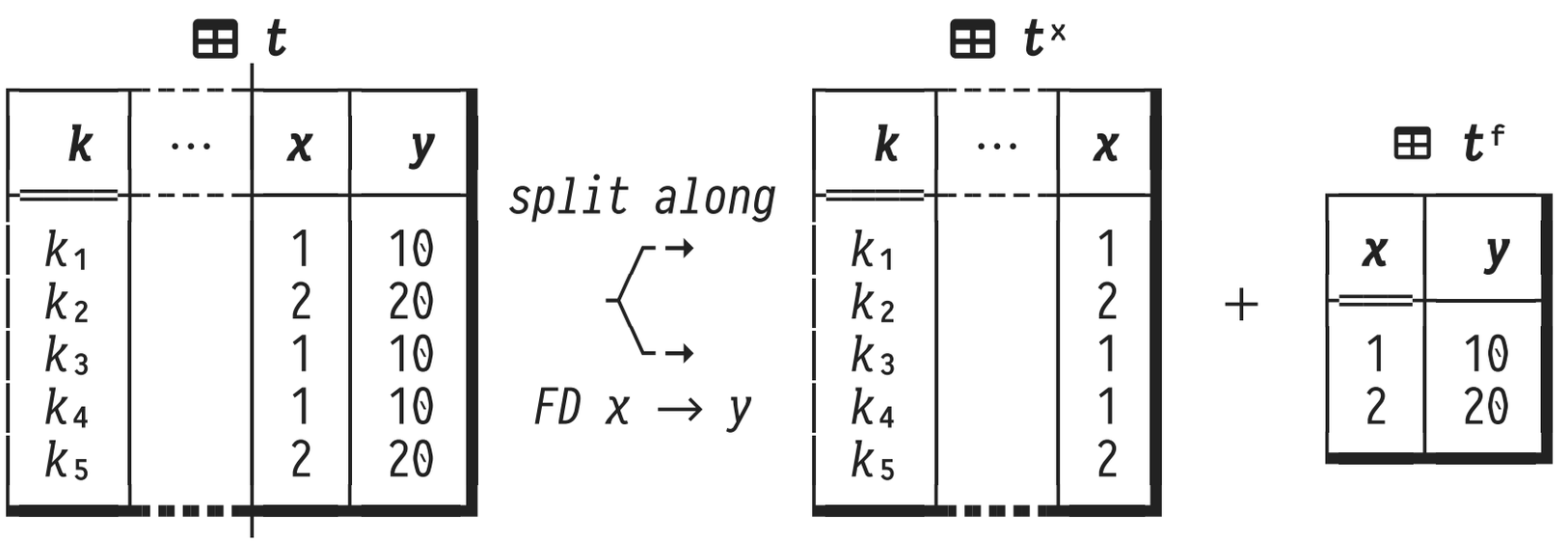
- Here are some FDs that hold in table trips. Verbalize what they mean in a DB application that manages road trips:

  Ⓐ vehicle → driver Ⓑ trip → weekday from to Ⓒ weekday → driver
  Ⓓ from to → dist driver Ⓔ from to hop# → via vehicle

… FDs Suggest Good Table Designs

1. The only FDs actually enforced by DBMSs are primary keys 🔑: recall the PRIMARY KEY constraint.
2. Non-🔑 FDs (embedded lookup tables) indicate data redundancy.

#055



- After the **split** ▦ ⟨ ▦+⊞ along FD $x \to y$:[4]
  - x is key in t^f: the DBMS **can now enforce the FD** $x \to y$. 👍
  - **No redundancy** in t^f. Function $f(x) = y$ is now explicit. 👍

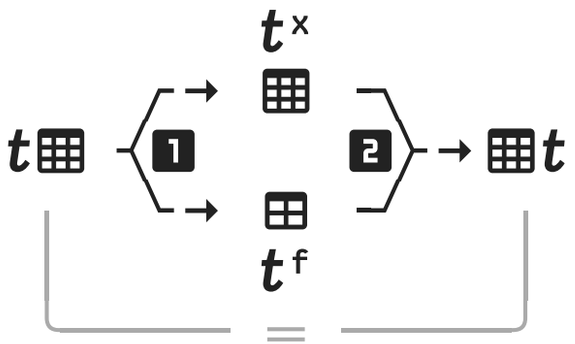
[4] Advise: When splitting along FD $x \to y$, move *all* columns functionally determined by x into t^f (including transitively determined columns). If t^f contains non-🔑 FDs, split t^f again.

Splits Can Be Undone (Virtually)

FD splits improve constraint checking and reduce redundancy. But users or apps may require/expect the original non-splitted table…

- An inner join ⋈ can **faithfully reconstruct** the original t.
  Table state is preserved (rows are neither lost nor added):[5]

FD split 1 2 Inner join ⋈ #056



- A **table view** can render this reconstruction of t transparent.

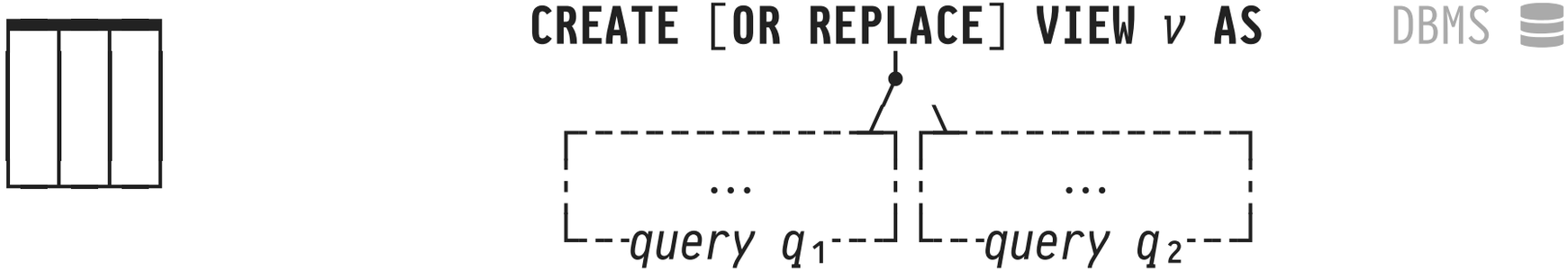
[5] FD splits are **lossless splits.** ⚠️ But: arbitrary vertical table splits may lose information (▦ ≠ 目+目+目).

4 ¦ Views (Logical Data Independence)

A SQL **view** v represents a *virtual* table whose schema and state is **defined by a query** q (instead of extensionally by a bag of rows).

- Views provide **logical data independence.** The actual data sources read by q remain hidden (q may change ↯ ↔ ↯):

Table ⊞ t **View** ⊞ v app



- Whenever v is referenced by a query, the DBMS re-evaluates q.
- Views are read-only and cannot be updated.[6]

[6] In general, it is ambigious how an “update” of v should affect the underlying source tables read by q.

The End.

Since you've got this far... A companion course on the design and implementation of selected DuckDB internals is available at https://github.com/DBatUTuebingen/DiDi.[7]

[7] DiDi: **D**esign and **I**mplementation of **D**uckDB **I**nternals 🩺🦆