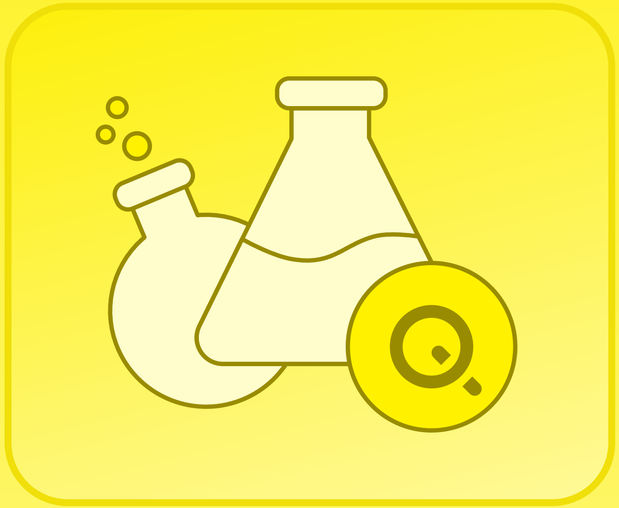
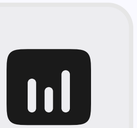


UBUNTU SUMMIT | 2026-05-28

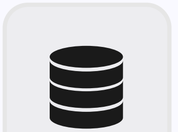
DuckDB: Not Quack Science

GÁBOR SZÁRNYAS
(DUCKLABS)

DuckDB at a glance



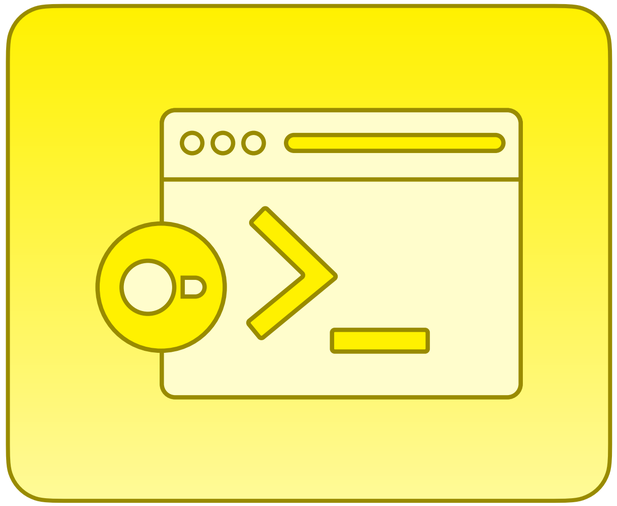
MIT license



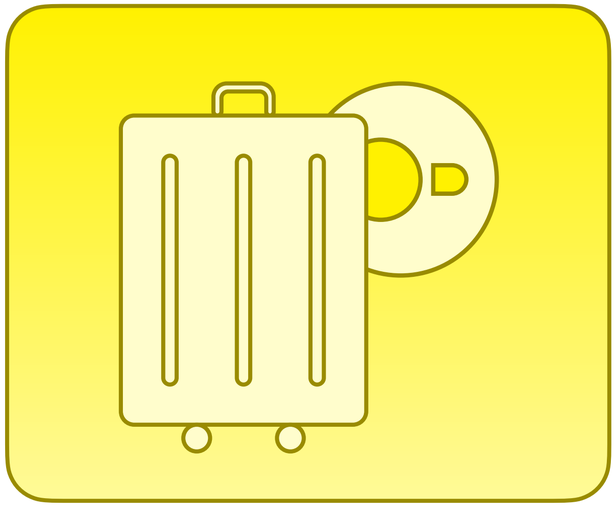
Relational DB with SQL

~1M LOC, no dependencies

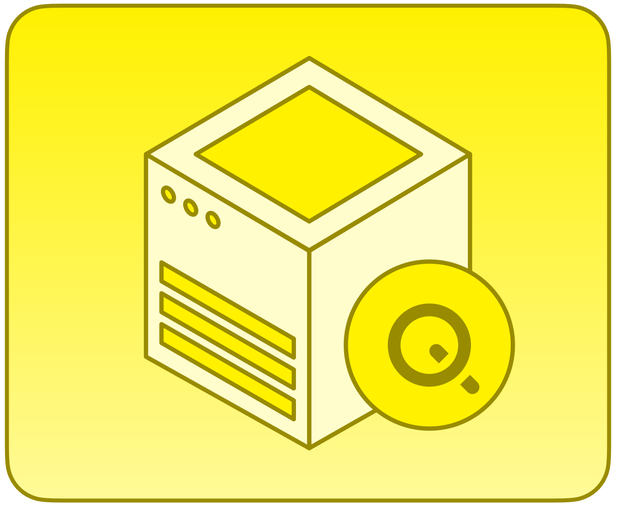
No sudo needed



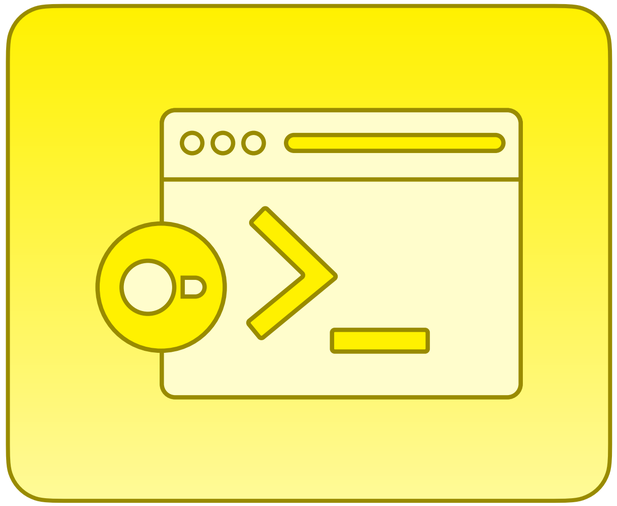
Command-line tool



Portable database



Database server



Command-line tool

```
$ curl https://install.duckdb.org | sh

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  4163  100  4163    0     0  20532      0 --:--:-- --:--:-- --:--:-- 20608

*** DuckDB Linux/MacOS installation script, version 1.5.3 ***


        .;odxdl,
       .xXXXXXXXXKc
       0XXXXXXXXXXXd  cooo:
      ,XXXXXXXXXXXXK  OXXXXd
       0XXXXXXXXXXXo  cooo:
       .xXXXXXXXXKc
         .;odxdl,
```

```
$ duckdb

DuckDB v1.5.3 (Variegata)
Enter ".help" for usage hints.
memory D
```

REPL

```
$ duckdb

DuckDB v1.5.3 (Variegata)
Enter ".help" for usage hints.
memory D SELECT sin(pi()/2) AS x;
```

x
double
1.0

```
memory D
```

```
$ duckdb

DuckDB v1.5.3 (Variegata)
Enter ".help" for usage hints.
memory D SELECT sin(pi()/2) AS x;
```

x
double
1.0

```
memory D SELECT today() - '2026-04-23'::DATE AS ubuntu_lts_out;
```

ubuntu_lts_out
int64
35

```
memory D
```

```
$ cat stations.tsv

station country
Amsterdam Centraal      NL
Brussels Midi   BE
Lille Europe    FR
London St Pancras       UK
Rotterdam Centraal      NL

$
```

```
$
```

CSV / text processing

```
$ cat stations.tsv

station↦country
Amsterdam Centraal⟼NL
Brussels Midi⟼BE
Lille Europe⟼FR
London St Pancras⟼UK
Rotterdam Centraal⟼NL

$
```

$

CSV / text processing

```
$ cat stations.tsv

station↦country
Amsterdam Centraal⟼NL
Brussels Midi⟼BE
Lille Europe⟼FR
London St Pancras⟼UK
Rotterdam Centraal⟼NL

$
```

```
$
```

CSV / text processing

List all countries!

```
$ cat stations.tsv

station↦country
Amsterdam Centraal⟼NL
Brussels Midi⟼BE
Lille Europe⟼FR
London St Pancras⟼UK
Rotterdam Centraal⟼NL

$
```

```
$ cat stations.tsv \
    | tail -n +2 \
    | cut -d $'\t' -f 2 \
    | sort -u
```

```
$ cat stations.tsv

station↦country
Amsterdam Centraal⟼NL
Brussels Midi⟼BE
Lille Europe⟼FR
London St Pancras⟼UK
Rotterdam Centraal⟼NL

$
```

```
$ cat stations.tsv \
    | tail -n +2 \
    | cut -d $'\t' -f 2 \
    | sort -u
```

```
$ cat stations.tsv

station↦country
Amsterdam Centraal ⟼ NL
Brussels Midi ⟼ BE
Lille Europe ⟼ FR
London St Pancras ⟼ UK
Rotterdam Centraal ⟼ NL

$
```

```
NL
BE
FR
UK
NL
```

```
$ cat stations.tsv \
    | tail -n +2 \
    | cut -d $'\t' -f 2 \
    | sort -u
```

```
$ cat stations.tsv

station↦country
Amsterdam Centraal ⟼ NL
Brussels Midi ⟼ BE
Lille Europe ⟼ FR
London St Pancras ⟼ UK
Rotterdam Centraal ⟼ NL

$
```

```
NL
BE
FR
UK
NL
```

```
BE
FR
NL
UK
```

```
$ cat stations.tsv \
    | tail -n +2 \
    | cut -d $'\t' -f 2 \
    | sort -u
```

```
$ cat stations.tsv

station↦country
Amsterdam Centraal ⟼ NL
Brussels Midi ⟼ BE
Lille Europe ⟼ FR
London St Pancras ⟼ UK
Rotterdam Centraal ⟼ NL

$
```

```
NL
BE
FR
UK
NL
```

```
BE
FR
NL
UK
```

```
$ cat stations.tsv \
    | tail -n +2 \
    | cut -d $'\t' -f 2 \
    | sort -u

BE
FR
NL
UK

$
```

```
$ cat stations.tsv

station↦country
Amsterdam Centraal⟼NL
Brussels Midi⟼BE
Lille Europe⟼FR
London St Pancras⟼UK
Rotterdam Centraal⟼NL

$
```

```
$ duckdb

DuckDB v1.5.3 (Variegata)
Enter ".help" for usage hints.
memory D SELECT DISTINCT country
         FROM 'stations.tsv';
```

```
$ cat stations.tsv

station↦country
Amsterdam Centraal⟼NL
Brussels Midi⟼BE
Lille Europe⟼FR
London St Pancras⟼UK
Rotterdam Centraal⟼NL

$
```

```
$ duckdb

DuckDB v1.5.3 (Variegata)
Enter ".help" for usage hints.
memory D SELECT DISTINCT country
         FROM 'stations.tsv';
```

```
$ cat stations.tsv

station↦country
Amsterdam Centraal⟼NL
Brussels Midi⟼BE
Lille Europe⟼FR
London St Pancras⟼UK
Rotterdam Centraal⟼NL

$
```

```
$ duckdb

DuckDB v1.5.3 (Variegata)
Enter ".help" for usage hints.
memory D SELECT DISTINCT country
         FROM 'stations.tsv';
```

```
$ cat stations.tsv

station↦country
Amsterdam Centraal ⟼ NL
Brussels Midi ⟼ BE
Lille Europe ⟼ FR
London St Pancras ⟼ UK
Rotterdam Centraal ⟼ NL

$
```

```
$ duckdb

DuckDB v1.5.3 (Variegata)
Enter ".help" for usage hints.
memory D SELECT DISTINCT country
         FROM 'stations.tsv';
```

country
varchar
FR
NL
BE
UK

```
memory D
```

```
$ cat stations.tsv

station↦country
Amsterdam Centraal⟼NL
Brussels Midi⟼BE
Lille Europe⟼FR
London St Pancras⟼UK
Rotterdam Centraal⟼NL

$
```

```
$ duckdb -c "SELECT DISTINCT country
             FROM 'stations.tsv'"
```

country
varchar
BE
UK
NL
FR

```
$
```

```
$ cat stations.tsv

station↦country
Amsterdam Centraal ⟼ NL
Brussels Midi ⟼ BE
Lille Europe ⟼ FR
London St Pancras ⟼ UK
Rotterdam Centraal ⟼ NL

$
```

```
$ duckdb -c "SELECT DISTINCT country
 FROM 'stations.tsv'"
```

country
varchar
BE
UK
NL
FR

```
$
```

```
$ cat stations.tsv

station↦country
Amsterdam Centraal ⟼ NL
Brussels Midi ⟼ BE
Lille Europe ⟼ FR
London St Pancras ⟼ UK
Rotterdam Centraal ⟼ NL

$
```

```
$ cat stations.tsv \
  | duckdb -c \
     "SELECT DISTINCT country
      FROM read_csv('/dev/stdin')"
```

country
varchar
FR
BE
UK
NL

```
$
```

```
$ cat stations.tsv

station↦country
Amsterdam Centraal ⟼ NL
Brussels Midi ⟼ BE
Lille Europe ⟼ FR
London St Pancras ⟼ UK
Rotterdam Centraal ⟼ NL

$
```

```
$ cat stations.tsv \
  | duckdb -c \
      "SELECT DISTINCT country
       FROM read_csv('/dev/stdin')"
```

country
varchar
FR
BE
UK
NL

```
$
```

```
$ cat trains.tsv

id       timestamp        station
ES9133   2026-05-22 10:40:00+02   Amsterdam Centraal
ES9133   2026-05-22 11:23:00+02   Rotterdam Centraal
ES9133   2026-05-22 12:38:00+02   Brussels Midi
ES9133   2026-05-22 13:30:00+02   Lille Europe
ES9133   2026-05-22 14:27:00+01   London St Pancras
ES9147   2026-05-22 14:40:00+02   Amsterdam Centraal
ES9147   2026-05-22 15:23:00+02   Rotterdam Centraal
ES9147   2026-05-22 16:38:00+02   Brussels Midi
ES9147   2026-05-22 18:17:00+01   London St Pancras

$
```

```
$ cat trains.tsv

id       timestamp           station
ES9133   2026-05-22 10:40:00+02   Amsterdam Centraal
ES9133   2026-05-22 11:23:00+02   Rotterdam Centraal
ES9133   2026-05-22 12:38:00+02   Brussels Midi
ES9133   2026-05-22 13:30:00+02   Lille Europe
ES9133   2026-05-22 14:27:00+01   London St Pancras
ES9147   2026-05-22 14:40:00+02   Amsterdam Centraal
ES9147   2026-05-22 15:23:00+02   Rotterdam Centraal
ES9147   2026-05-22 16:38:00+02   Brussels Midi
ES9147   2026-05-22 18:17:00+01   London St Pancras

$
```

```
$ duckdb

DuckDB v1.5.3 (Variegata)
Enter ".help" for usage hints.
memory D SELECT * FROM 'trains.tsv';
```

id varchar	timestamp timestamp with time zone	station varchar
ES9133	2026-05-22 09:40:00+01	Amsterdam Centraal
ES9133	2026-05-22 10:23:00+01	Rotterdam Centraal
ES9133	2026-05-22 11:38:00+01	Brussels Midi
ES9133	2026-05-22 12:30:00+01	Lille Europe
ES9133	2026-05-22 14:27:00+01	London St Pancras
ES9147	2026-05-22 13:40:00+01	Amsterdam Centraal
ES9147	2026-05-22 14:23:00+01	Rotterdam Centraal
ES9147	2026-05-22 15:38:00+01	Brussels Midi
ES9147	2026-05-22 18:17:00+01	London St Pancras

```
memory D
```

```
$ duckdb

DuckDB v1.5.3 (Variegata)
Enter ".help" for usage hints.
memory D SELECT * FROM 'trains.tsv';
```

id varchar	timestamp timestamp with time zone	station varchar
ES9133	2026-05-22 09:40:00+01	Amsterdam Centraal
ES9133	2026-05-22 10:23:00+01	Rotterdam Centraal
ES9133	2026-05-22 11:38:00+01	Brussels Midi
ES9133	2026-05-22 12:30:00+01	Lille Europe
ES9133	2026-05-22 14:27:00+01	London St Pancras
ES9147	2026-05-22 13:40:00+01	Amsterdam Centraal
ES9147	2026-05-22 14:23:00+01	Rotterdam Centraal
ES9147	2026-05-22 15:38:00+01	Brussels Midi
ES9147	2026-05-22 18:17:00+01	London St Pancras

```
memory D
```

```
$ duckdb

DuckDB v1.5.3 (Variegata)
Enter ".help" for usage hints.
memory D SELECT * FROM 'trains.tsv';
```

id varchar	timestamp timestamp with time zone	station varchar
ES9133	2026-05-22 09:40:00+01	Amsterdam Centraal
ES9133	2026-05-22 10:23:00+01	Rotterdam Centraal
ES9133	2026-05-22 11:38:00+01	Brussels Midi
ES9133	2026-05-22 12:30:00+01	Lille Europe
ES9133	2026-05-22 14:27:00+01	London St Pancras
ES9147	2026-05-22 13:40:00+01	Amsterdam Centraal
ES9147	2026-05-22 14:23:00+01	Rotterdam Centraal
ES9147	2026-05-22 15:38:00+01	Brussels Midi
ES9147	2026-05-22 18:17:00+01	London St Pancras

```
memory D
```

Let's add
the station's country!

```
$ duckdb

DuckDB v1.5.3 (Variegata)
Enter ".help" for usage hints.
memory D SELECT *
         FROM 'trains.tsv'
         JOIN 'stations.tsv' USING (station);
```

```
$ duckdb

DuckDB v1.5.3 (Variegata)
Enter ".help" for usage hints.
memory D SELECT *
         FROM 'trains.tsv'
         JOIN 'stations.tsv' USING (station);
```

```
$ duckdb

DuckDB v1.5.3 (Variegata)
Enter ".help" for usage hints.
memory D SELECT *
         FROM 'trains.tsv'
         JOIN 'stations.tsv' USING (station);
```

id varchar	timestamp timestamp with time zone	station varchar	country varchar
ES9133	2026-05-22 09:40:00+01	Amsterdam Centraal	NL
ES9133	2026-05-22 10:23:00+01	Rotterdam Centraal	NL
ES9133	2026-05-22 11:38:00+01	Brussels Midi	BE
ES9133	2026-05-22 12:30:00+01	Lille Europe	FR
ES9133	2026-05-22 14:27:00+01	London St Pancras	UK
ES9147	2026-05-22 13:40:00+01	Amsterdam Centraal	NL
ES9147	2026-05-22 14:23:00+01	Rotterdam Centraal	NL
ES9147	2026-05-22 15:38:00+01	Brussels Midi	BE
ES9147	2026-05-22 18:17:00+01	London St Pancras	UK

```
memory D
```

```
$ duckdb

DuckDB v1.5.3 (Variegata)
Enter ".help" for usage hints.
memory D SELECT *
         FROM 'trains.tsv'
         JOIN 'stations.tsv' USING (station);
```

id varchar	timestamp timestamp with time zone	station varchar	country varchar
ES9133	2026-05-22 09:40:00+01	Amsterdam Centraal	NL
ES9133	2026-05-22 10:23:00+01	Rotterdam Centraal	NL
ES9133	2026-05-22 11:38:00+01	Brussels Midi	BE
ES9133	2026-05-22 12:30:00+01	Lille Europe	FR
ES9133	2026-05-22 14:27:00+01	London St Pancras	UK
ES9147	2026-05-22 13:40:00+01	Amsterdam Centraal	NL
ES9147	2026-05-22 14:23:00+01	Rotterdam Centraal	NL
ES9147	2026-05-22 15:38:00+01	Brussels Midi	BE
ES9147	2026-05-22 18:17:00+01	London St Pancras	UK

```
memory D COPY _ TO 'trains-with-countries.tsv' (DELIMITER '\t');
```

```
$ echo "id\ttimestamp\tstation\tcountry" &&
  join -t $'\t' -1 3 -2 1 \
    <(tail -n +2 trains.tsv | sort -t $'\t' -k 3,3) \
    <(tail -n +2 stations.tsv) \
    | awk -F $'\t' '{ print $2 "\t" $3 "\t" $1 "\t" $4 }' \
    | sort -t $'\t' -k 2,2

id      timestamp        station country
ES9133  2026-05-22 10:40:00+02  Amsterdam Centraal      NL
ES9133  2026-05-22 11:23:00+02  Rotterdam Centraal      NL
ES9133  2026-05-22 12:38:00+02  Brussels Midi   BE
ES9133  2026-05-22 13:30:00+02  Lille Europe    FR
ES9133  2026-05-22 14:27:00+01  London St Pancras       UK
ES9147  2026-05-22 14:40:00+02  Amsterdam Centraal      NL
ES9147  2026-05-22 15:23:00+02  Rotterdam Centraal      NL
ES9147  2026-05-22 16:38:00+02  Brussels Midi   BE
ES9147  2026-05-22 18:17:00+01  London St Pancras       UK

$
```

```
$ echo "id\ttimestamp\tstation\tcountry" &&
  join -t $'\t' -1 3 -2 1 \
    <(tail -n +2 trains.tsv | sort -t $'\t' -k 3,3) \
    <(tail -n +2 stations.tsv) \
    | awk -F $'\t' '{ print $2 "\t" $3 "\t" $1 "\t" $4 }' \
    | sort -t $'\t' -k 2,2

id      timestamp        station country
ES9133  2026-05-22 10:40:00+02  Amsterdam Centraal      NL
ES9133  2026-05-22 11:23:00+02  Rotterdam Centraal      NL
ES9133  2026-05-22 12:38:00+02  Brussels Midi   BE
ES9133  2026-05-22 13:30:00+02  Lille Europe    FR
ES9133  2026-05-22 14:27:00+01  London St Pancras       UK
ES9147  2026-05-22 14:40:00+02  Amsterdam Centraal      NL
ES9147  2026-05-22 15:23:00+02  Rotterdam Centraal      NL
ES9147  2026-05-22 16:38:00+02  Brussels Midi   BE
ES9147  2026-05-22 18:17:00+01  London St Pancras       UK

$
```

drop header

pre-sort on join columns

join

awk to reorder

sort

add new header

Data wrangling in the CLI

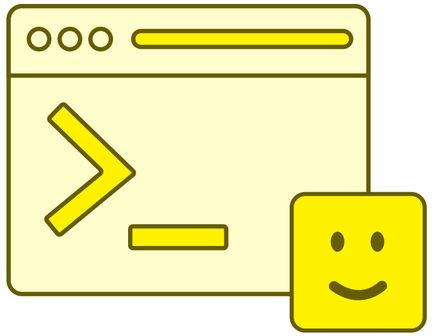
Common trajectory:

Start with a simple shell script

Evolve it into
- a complex shell script
- a small Python script (maybe with dataframes)

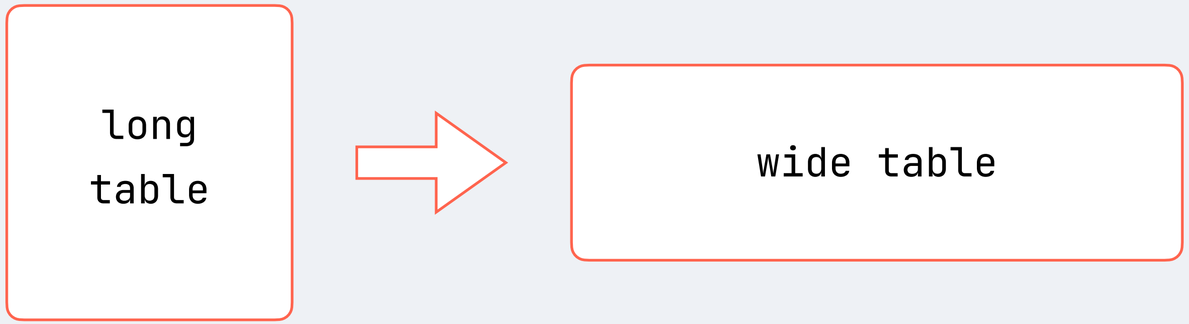
You can do most of these things in SQL!
It’s composable and it gets parallelized

Unix	SQL
`cut`	`SELECT`
`grep`	`WHERE`
`sort`	`ORDER BY`
`sort -u`	`DISTINCT`
`wc -l`	`count(*)`
`cat`	`UNION ALL`
`head`	`LIMIT`
`paste`	`POSITIONAL JOIN`
`comm -12`	`INTERSECT ALL`
`comm -23`	`EXCEPT ALL`

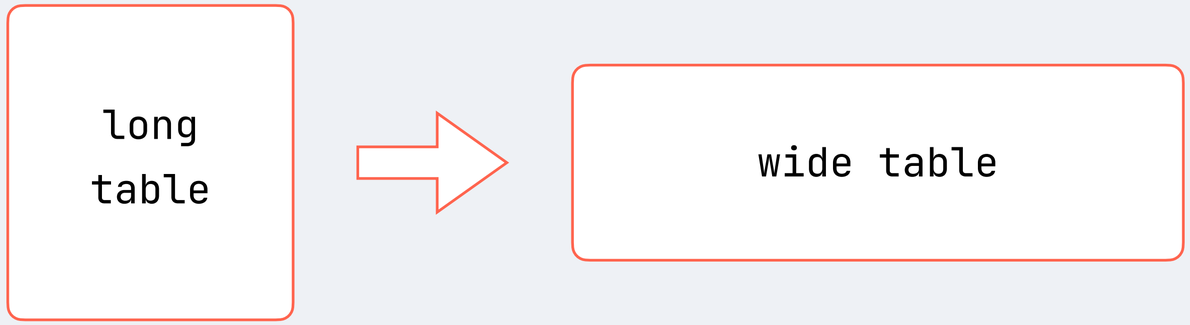


Powerful SQL:
Pivoting and other operations

pivoting



pivoting

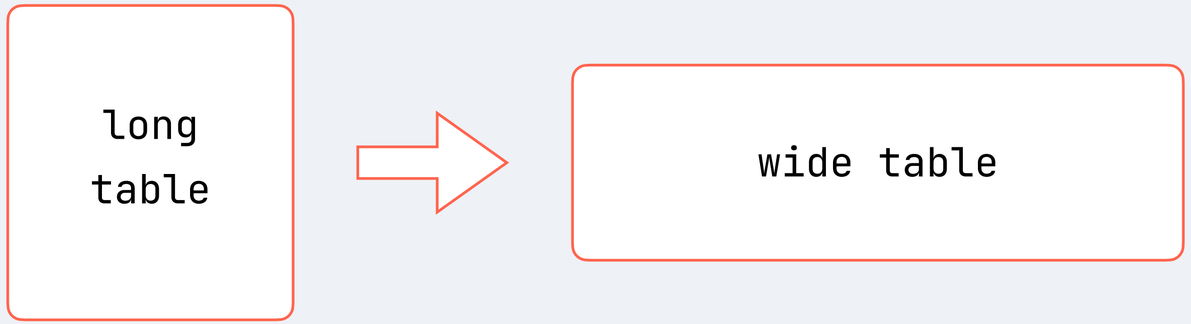



```
memory D CREATE TABLE dfl AS FROM 'distance-from-london.csv';
memory D FROM dfl;
```

station varchar	distance int64
London	0
Lille	260
Brussels	333
Rotterdam	460
Amsterdam	542

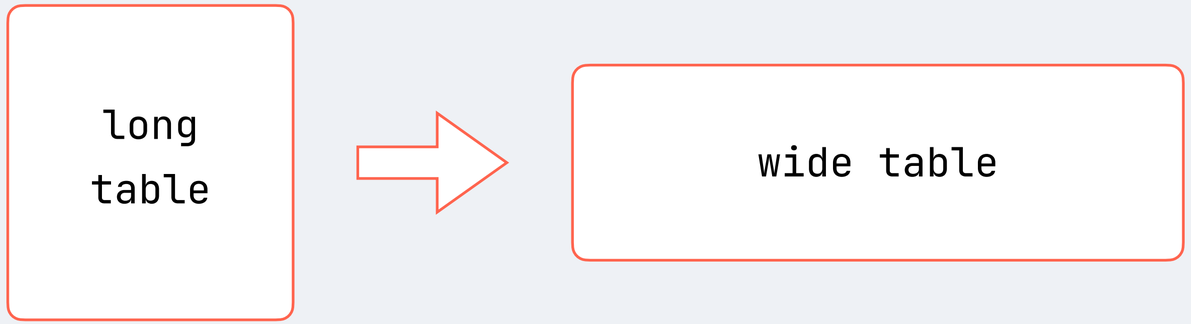
```
memory D
```

pivoting



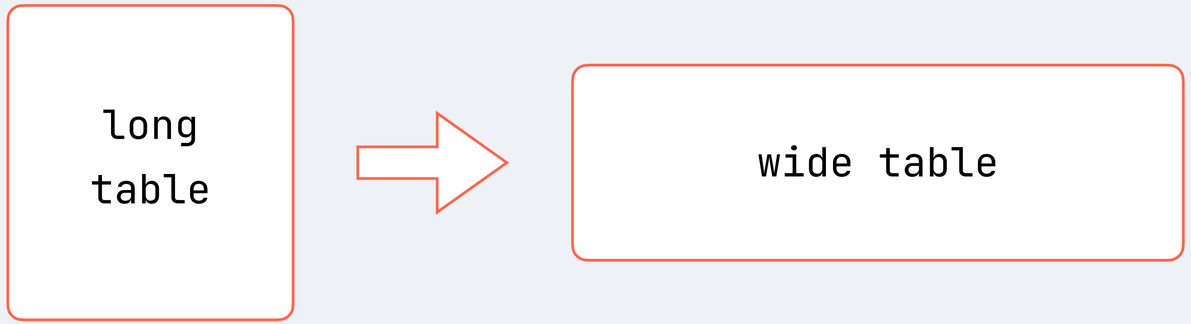

```
memory D SELECT
             dfl1.station AS station1,
             dfl2.station AS station2,
             abs(dfl1.distance - dfl2.distance) AS distance
         FROM dfl AS dfl1, dfl AS dfl2;
```

pivoting



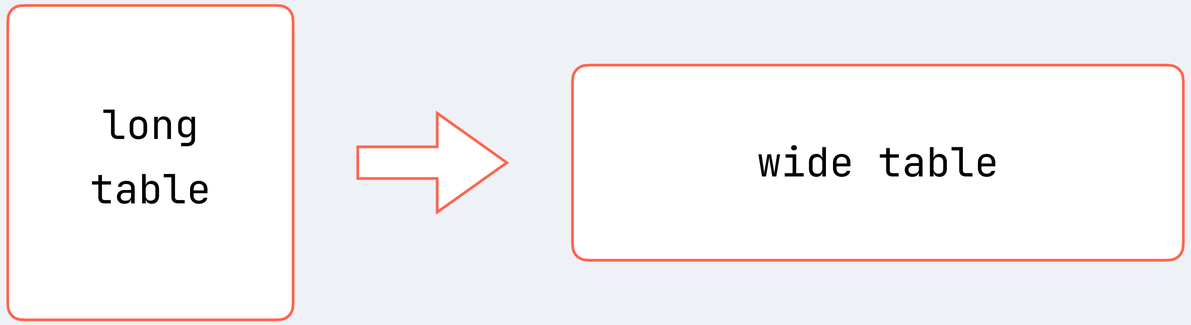

```
memory D SELECT
             dfl1.station AS station1,
             dfl2.station AS station2,
             abs(dfl1.distance - dfl2.distance) AS distance
         FROM dfl AS dfl1, dfl AS dfl2;
```

pivoting




```
memory D SELECT
             dfl1.station AS station1,
             dfl2.station AS station2,
             abs(dfl1.distance - dfl2.distance) AS distance
         FROM dfl AS dfl1, dfl AS dfl2;
```

pivoting

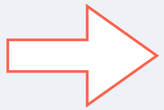



```
memory D SELECT
             dfl1.station AS station1,
             dfl2.station AS station2,
             abs(dfl1.distance - dfl2.distance) AS distance
         FROM dfl AS dfl1, dfl AS dfl2;
```

station1 varchar	station2 varchar	distance int64
London	London	0
Lille	London	260
Brussels	London	333
Rotterdam	London	460
Amsterdam	London	542
London	Lille	260
Lille	Lille	0

pivoting

station1 varchar	station2 varchar	distance int64
London	London	0
Lille	London	260
Brussels	London	333
Rotterdam	London	460
Amsterdam	London	542
London	Lille	260
Lille	Lille	0
Brussels	Lille	73
Rotterdam	Lille	200
Amsterdam	Lille	282
London	Brussels	333
Lille	Brussels	73
Brussels	Brussels	0
Rotterdam	Brussels	127

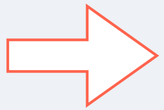


wide table

```
memory D PIVOT _
          ON station2
               USING first(distance)
               ORDER BY station1;
```

pivoting

station1 varchar	station2 varchar	distance int64
London	London	0
Lille	London	260
Brussels	London	333
Rotterdam	London	460
Amsterdam	London	542
London	Lille	260
Lille	Lille	0
Brussels	Lille	73
Rotterdam	Lille	200
Amsterdam	Lille	282
London	Brussels	333
Lille	Brussels	73
Brussels	Brussels	0
Rotterdam	Brussels	127

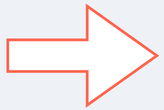


wide table

```
memory D PIVOT _
         ON station2
              USING first(distance)
              ORDER BY station1;
```

pivoting

station1 varchar	station2 varchar	distance int64
London	London	0
Lille	London	260
Brussels	London	333
Rotterdam	London	460
Amsterdam	London	542
London	Lille	260
Lille	Lille	0
Brussels	Lille	73
Rotterdam	Lille	200
Amsterdam	Lille	282
London	Brussels	333
Lille	Brussels	73
Brussels	Brussels	0
Rotterdam	Brussels	127

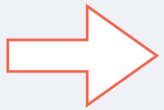


wide table

```
memory D PIVOT _
         ON station2
              USING first(distance)
              ORDER BY station1;
```

pivoting

station1 varchar	station2 varchar	distance int64
London	London	0
Lille	London	260
Brussels	London	333
Rotterdam	London	460
Amsterdam	London	542
London	Lille	260
Lille	Lille	0
Brussels	Lille	73
Rotterdam	Lille	200
Amsterdam	Lille	282
London	Brussels	333
Lille	Brussels	73
Brussels	Brussels	0
Rotterdam	Brussels	127



wide table

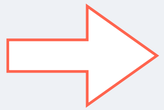
```
memory D PIVOT _
         ON station2
              USING first(distance)
              ORDER BY station1;
```

station1 varchar	Amsterdam int64	Brussels int64	Lille int64	London int64	Rotterdam int64
Amsterdam	0	209	282	542	82
Brussels	209	0	73	333	127
Lille	282	73	0	260	200
London	542	333	260	0	460
Rotterdam	82	127	200	460	0

```
memory D
```

pivoting

station1 varchar	station2 varchar	distance int64
London	London	0
Lille	London	260
Brussels	London	333
Rotterdam	London	460
Amsterdam	London	542
London	Lille	260
Lille	Lille	0
Brussels	Lille	73
Rotterdam	Lille	200
Amsterdam	Lille	282
London	Brussels	333
Lille	Brussels	73
Brussels	Brussels	0
Rotterdam	Brussels	127



wide table

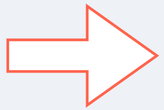
```
memory D PIVOT _
         ON station2
              USING first(distance)
              ORDER BY station1;
```

station1 varchar	Amsterdam int64	Brussels int64	Lille int64	London int64	Rotterdam int64
Amsterdam	0	209	282	542	82
Brussels	209	0	73	333	127
Lille	282	73	0	260	200
London	542	333	260	0	460
Rotterdam	82	127	200	460	0

```
memory D CREATE TABLE distance_matrix AS FROM _;
```

pivoting

station1 varchar	station2 varchar	distance int64
London	London	0
Lille	London	260
Brussels	London	333
Rotterdam	London	460
Amsterdam	London	542
London	Lille	260
Lille	Lille	0
Brussels	Lille	73
Rotterdam	Lille	200
Amsterdam	Lille	282
London	Brussels	333
Lille	Brussels	73
Brussels	Brussels	0
Rotterdam	Brussels	127



wide table

```
memory D PIVOT _
         ON station2
              USING first(distance)
              ORDER BY station1;
```

station1 varchar	Amsterdam int64	Brussels int64	Lille int64	London int64	Rotterdam int64
Amsterdam	0	209	282	542	82
Brussels	209	0	73	333	127
Lille	282	73	0	260	200
London	542	333	260	0	460
Rotterdam	82	127	200	460	0

```
memory D CREATE TABLE distance_matrix AS FROM _;
```

Convert kilometers to miles

Convert kilometers to miles

```
memory D SELECT
             station1,
             (Amsterdam / 1.61)::INT AS Amsterdam,
             (Brussels  / 1.61)::INT AS Brussels,
             (Lille     / 1.61)::INT AS Lille,
             (London    / 1.61)::INT AS London,
             (Rotterdam / 1.61)::INT AS Rotterdam
         FROM distance_matrix;
```

Convert kilometers to miles

```
memory D SELECT
             station1,
             (Amsterdam / 1.61)::INT AS Amsterdam,
             (Brussels  / 1.61)::INT AS Brussels,
             (Lille     / 1.61)::INT AS Lille,
             (London    / 1.61)::INT AS London,
             (Rotterdam / 1.61)::INT AS Rotterdam
         FROM distance_matrix;
```

station1 varchar	Amsterdam int32	Brussels int32	Lille int32	London int32	Rotterdam int32
Amsterdam	0	130	175	337	51
Brussels	130	0	45	207	79
Lille	175	45	0	161	124
London	337	207	161	0	286
Rotterdam	51	79	124	286	0

Convert kilometers to miles

```
memory D SELECT
           station1,
           (COLUMNS(* EXCLUDE station1) / 1.61)::INT
         FROM distance_matrix;
```

Convert kilometers to miles

```
memory D SELECT
           station1,
           (COLUMNS(* EXCLUDE station1) / 1.61)::INT
         FROM distance_matrix;
```

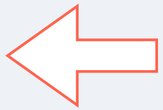
Convert kilometers to miles

```
memory D SELECT
           station1,
           (COLUMNS(* EXCLUDE station1) / 1.61)::INT
         FROM distance_matrix;
```

station1 varchar	Amsterdam int32	Brussels int32	Lille int32	London int32	Rotterdam int32
Amsterdam	0	130	175	337	51
Brussels	130	0	45	207	79
Lille	175	45	0	161	124
London	337	207	161	0	286
Rotterdam	51	79	124	286	0

unpivoting

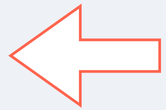
long table



station1 varchar	Amsterdam int64	Brussels int64	Lille int64	London int64	Rotterdam int64
Amsterdam	0	209	282	542	82
Brussels	209	0	73	333	127
Lille	282	73	0	260	200
London	542	333	260	0	460
Rotterdam	82	127	200	460	0

unpivoting

long table



station1 varchar	Amsterdam int64	Brussels int64	Lille int64	London int64	Rotterdam int64
Amsterdam	0	209	282	542	82
Brussels	209	0	73	333	127
Lille	282	73	0	260	200
London	542	333	260	0	460
Rotterdam	82	127	200	460	0

```
memory D UNPIVOT distance_matrix
            ON Amsterdam, Brussels, Lille, London, Rotterdam
            INTO
                NAME station2
                VALUE distance;
```

unpivoting

long table

station1 varchar	Amsterdam int64	Brussels int64	Lille int64	London int64	Rotterdam int64
Amsterdam	0	209	282	542	82
Brussels	209	0	73	333	127
Lille	282	73	0	260	200
London	542	333	260	0	460
Rotterdam	82	127	200	460	0

```
memory D UNPIVOT distance_matrix
         ON Amsterdam, Brussels, Lille, London, Rotterdam
         INTO
              NAME station2
              VALUE distance;
```

unpivoting

long table

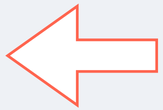
station1 varchar	Amsterdam int64	Brussels int64	Lille int64	London int64	Rotterdam int64
Amsterdam	0	209	282	542	82
Brussels	209	0	73	333	127
Lille	282	73	0	260	200
London	542	333	260	0	460
Rotterdam	82	127	200	460	0

```
memory D UNPIVOT distance_matrix
         ON Amsterdam, Brussels, Lille, London, Rotterdam
         INTO
             NAME station2
             VALUE distance;
```

station1 varchar	station2 varchar	distance int64
Amsterdam	Amsterdam	0
Amsterdam	Brussels	209
Amsterdam	Lille	282
Amsterdam	London	542
Amsterdam	Rotterdam	82
Brussels	Amsterdam	209
Brussels	Brussels	0

unpivoting

long table



station1 varchar	Amsterdam int64	Brussels int64	Lille int64	London int64	Rotterdam int64
Amsterdam	0	209	282	542	82
Brussels	209	0	73	333	127
Lille	282	73	0	260	200
London	542	333	260	0	460
Rotterdam	82	127	200	460	0

```
memory D UNPIVOT distance_matrix
            ON COLUMNS(* EXCLUDE station1)
            INTO
                NAME station2
                VALUE distance;
```

station1 varchar	station2 varchar	distance int64
Amsterdam	Amsterdam	0
Amsterdam	Brussels	209
Amsterdam	Lille	282
Amsterdam	London	542
Amsterdam	Rotterdam	82
Brussels	Amsterdam	209
Brussels	Brussels	0

unpivoting

long table

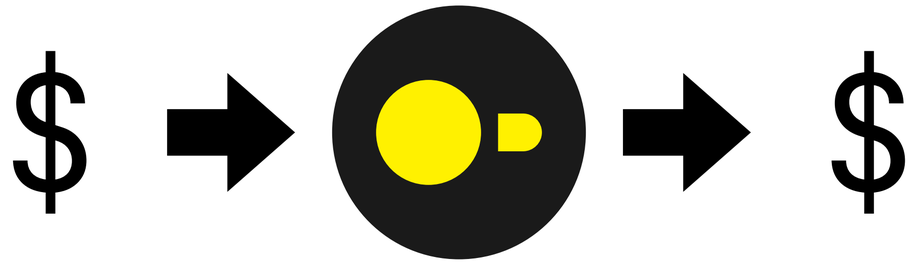
station1 varchar	Amsterdam int64	Brussels int64	Lille int64	London int64	Rotterdam int64
Amsterdam	0	209	282	542	82
Brussels	209	0	73	333	127
Lille	282	73	0	260	200
London	542	333	260	0	460
Rotterdam	82	127	200	460	0

```
memory D UNPIVOT distance_matrix
           ON COLUMNS(* EXCLUDE station1)
           INTO
               NAME station2
               VALUE distance;
```

station1 varchar	station2 varchar	distance int64
Amsterdam	Amsterdam	0
Amsterdam	Brussels	209
Amsterdam	Lille	282
Amsterdam	London	542
Amsterdam	Rotterdam	82
Brussels	Amsterdam	209
Brussels	Brussels	0

DuckDB in the CLI

You can use an analytical database in the shell



- works in the pipeline
- has error handling / type safety
- no storage required
- parallelism out-of-the-box
- handles larger-than-memory workloads

USING DUCKDB

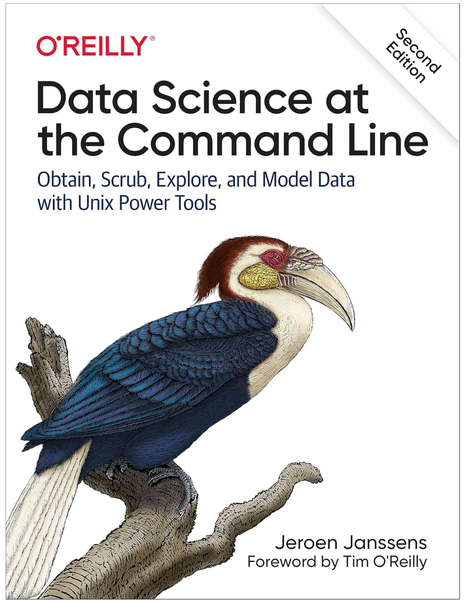
Command Line Data Processing: Using DuckDB as a Unix Tool

2024-06-20 Gábor Szárnyas

USING DUCKDB

Command Line Data Processing: Using DuckDB as a Unix Tool

2024-06-20 Gábor Szárnyas



DuckDB Solutions

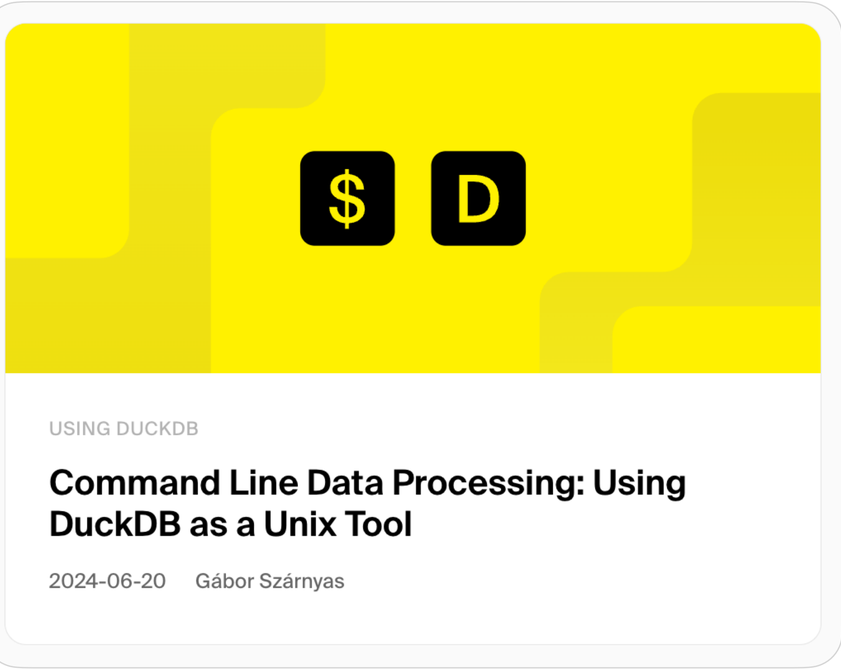
In the following, I give the DuckDB solutions for each exercise. If the command is part of a pipeline (e.g., when the input is read from the standard input), DuckDB is called as `... | duckdb -c "(SQL query)"`. Other times, the solution is presented simply as a SQL script.

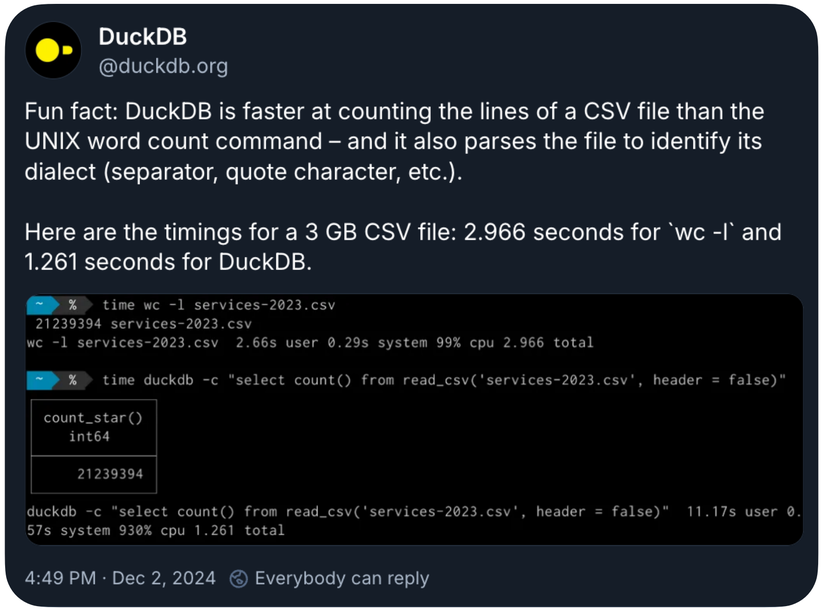
5.2 Transformations, Transformations Everywhere

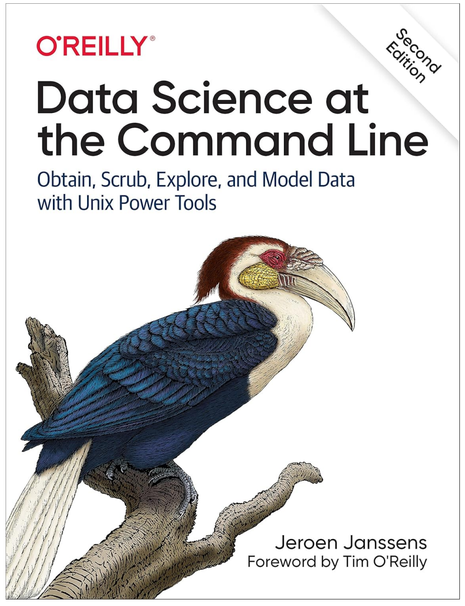
```
seq 100 |\
    python ../ch04/fizzbuzz.py |\
    tee fb.seq | trim
```

```
grep -E "fizz|buzz" fb.seq | # <1>\
sort | uniq -c | sort -nr > fb.cnt # <2>
bat -A fb.cnt
< fb.cnt awk 'BEGIN { print "value,count" } { print $2","$1 }' > fb.csv
```

```
create table fb as from read_csv('fb.seq', header = false);
create table counts as
    select column0 as value, count(*) as count
    from fb
    where column0 similar to '^(fizz|buzz).*'
    group by all
    order by count desc;
copy counts to 'fb.csv';
```







DuckDB Solutions

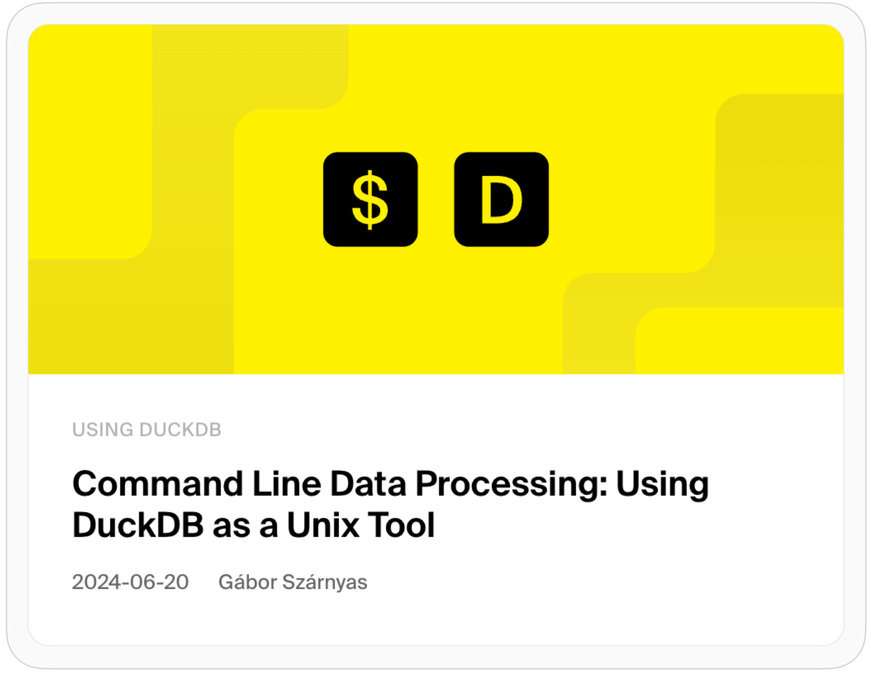
In the following, I give the DuckDB solutions for each exercise. If the command is part of a pipeline (e.g., when the input is read from the standard input), DuckDB is called as `... | duckdb -c "(SQL query)"`. Other times, the solution is presented simply as a SQL script.

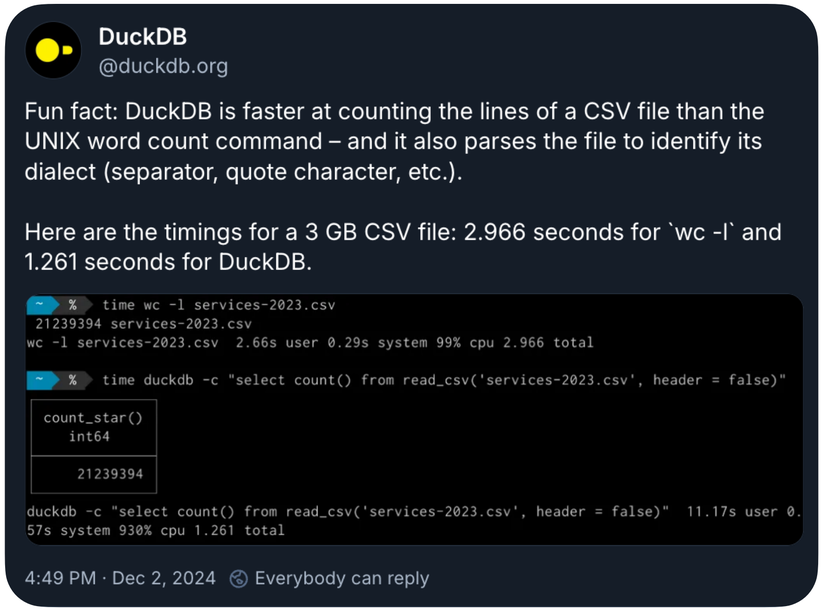
5.2 Transformations, Transformations Everywhere

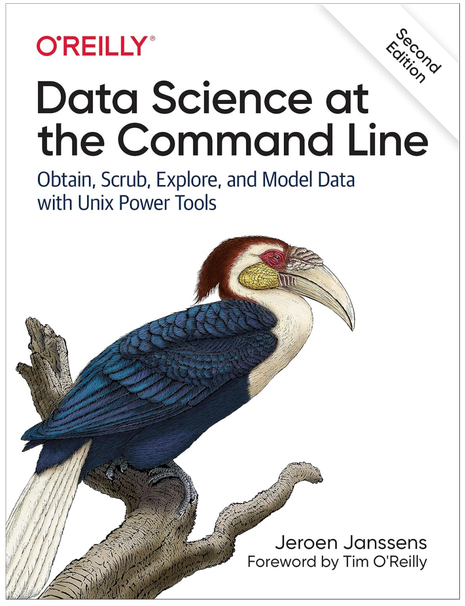
```
seq 100 |\
    python ../ch04/fizzbuzz.py |\
    tee fb.seq | trim
```

```
grep -E "fizz|buzz" fb.seq | # <1>\
sort | uniq -c | sort -nr > fb.cnt # <2>
bat -A fb.cnt
< fb.cnt awk 'BEGIN { print "value,count" } { print $2","$1 }' > fb.csv
```

```
create table fb as from read_csv('fb.seq', header = false);
create table counts as
    select column0 as value, count(*) as count
    from fb
    where column0 similar to '^(fizz|buzz).*'
    group by all
    order by count desc;
copy counts to 'fb.csv';
```







DuckDB Solutions

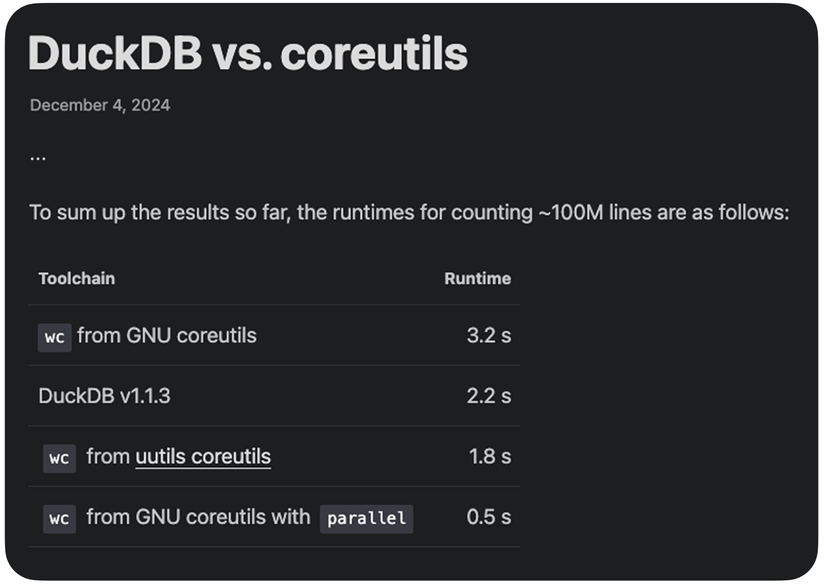
In the following, I give the DuckDB solutions for each exercise. If the command is part of a pipeline (e.g., when the input is read from the standard input), DuckDB is called as `... | duckdb -c "(SQL query)"` . Other times, the solution is presented simply as a SQL script.

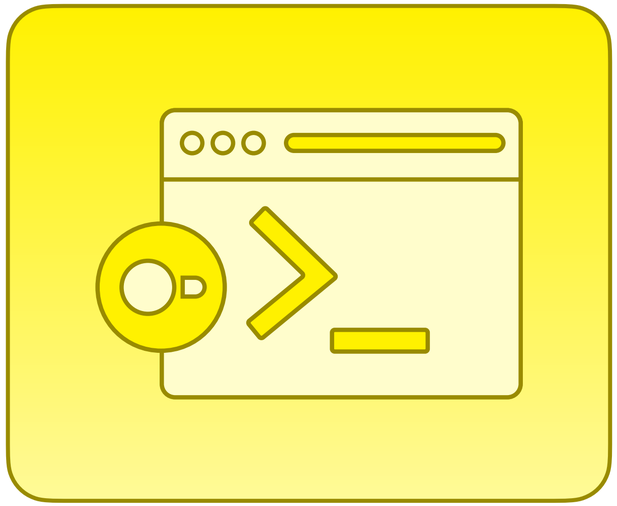
5.2 Transformations, Transformations Everywhere

```
seq 100 |\
    python ../ch04/fizzbuzz.py |\
    tee fb.seq | trim
```

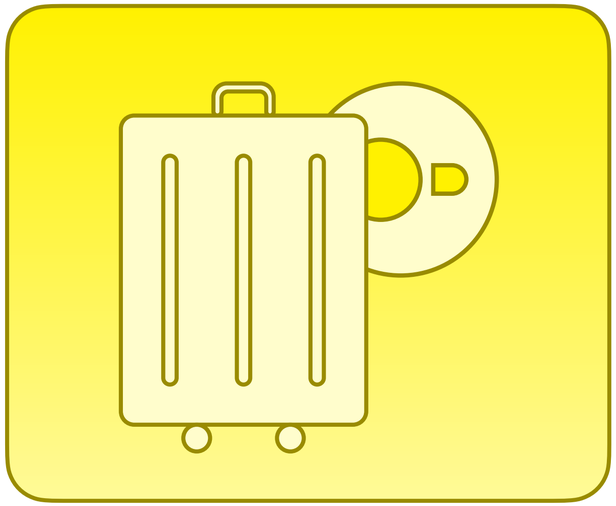
```
grep -E "fizz|buzz" fb.seq | # <1>\
sort | uniq -c | sort -nr > fb.cnt # <2>
bat -A fb.cnt
< fb.cnt awk 'BEGIN { print "value,count" } { print $2","$1 }' > fb.csv
```

```
create table fb as from read_csv('fb.seq', header = false);
create table counts as
    select column0 as value, count(*) as count
    from fb
    where column0 similar to '^(fizz|buzz).*'
    group by all
    order by count desc;
copy counts to 'fb.csv';
```

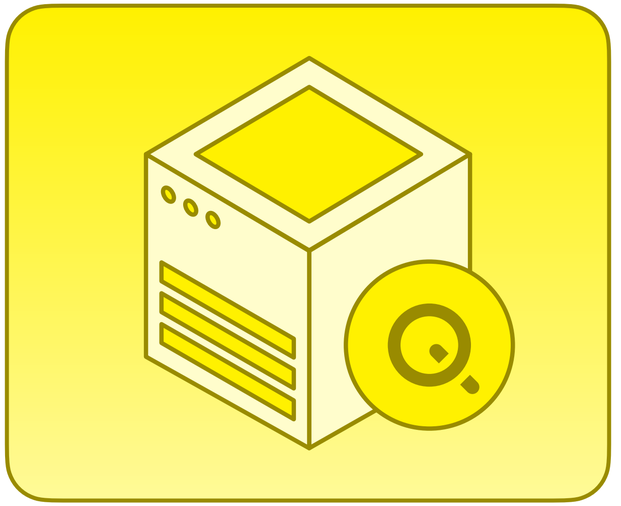




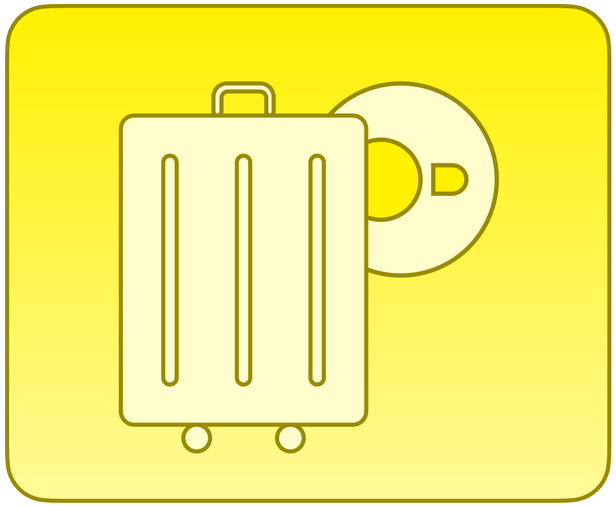
Command-line tool



Portable database



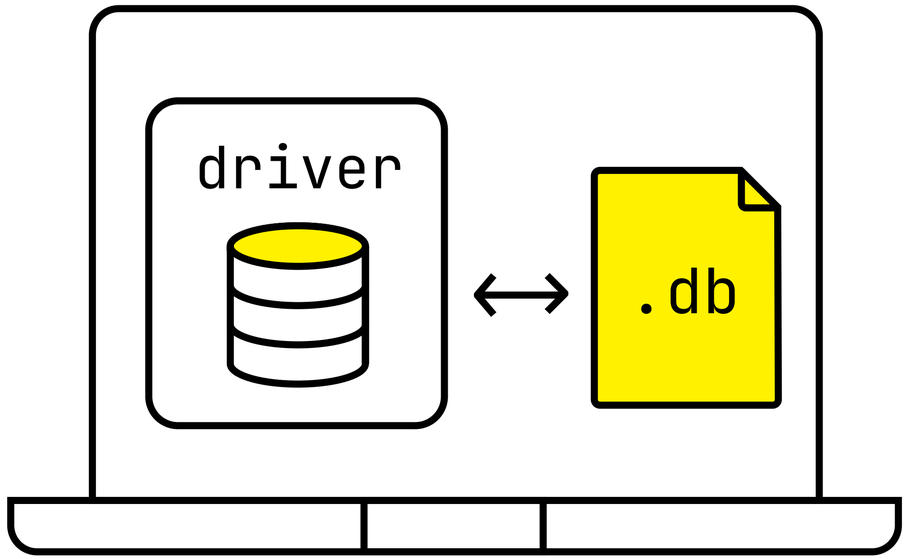
Database server



Portable database

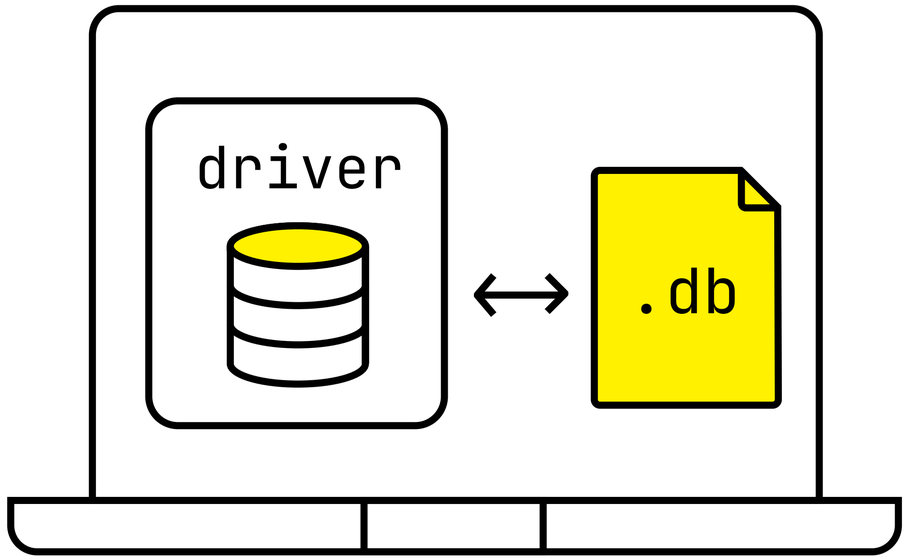
In-process architecture

DuckDB also works as a library



In-process architecture

DuckDB also works as a library



At a high level:

- No configuration
- No client-server protocol needed
- You can implement application logic with transactions, primary keys, constraints, etc.

DuckDB both:

- Integrates into the data ecosystem
- Works for small problems (e.g., compare results)

DuckDB drivers

```
pip install duckdb
```

```
go get github.com/duckdb/duckdb-go/v2
```

```
cargo add duckdb --features bundled
```

Open dataset: Railway services

NL trains 2019–

160 million records

21 GB as CSV files

Rijden de Treinen

Open dataset: Railway services

NL trains 2019–

160 million records

21 GB as CSV files

Rijden de Treinen

Analysis: Average delay per year

marimo

```python
import marimo as mo
import duckdb

duckdb_conn = duckdb.connect("trains.db")
```

```sql
CREATE TABLE services AS
FROM 'services-*.csv';
```

Output variable: _df | DuckDB (duckdb_conn) | Hide output | sql

```sql
SELECT count(*) FROM services;
```

Output variable: _df | DuckDB (duckdb_conn) | Hide output | sql

count_star()
i64
159,575,088

1 row, 1 column

Page 1 of 1

Download

```python
import marimo as mo
import duckdb

duckdb_conn = duckdb.connect("trains.db")
```

```sql
CREATE TABLE services AS
FROM 'services-*.csv';
```

Output variable: _df | DuckDB (duckdb_conn) | Hide output | sql

```sql
SELECT count(*) FROM services;
```

Output variable: _df | DuckDB (duckdb_conn) | Hide output | sql

count_star()
i64
159,575,088

1 row, 1 column

Page 1 of 1

Download

```python
import marimo as mo
import duckdb

duckdb_conn = duckdb.connect("trains.db")
```

```sql
CREATE TABLE services AS
FROM 'services-*.csv';
```

~1 GB/s

Output variable: _df | DuckDB (duckdb_conn) | Hide output | sql

```sql
SELECT count(*) FROM services;
```

Output variable: _df | DuckDB (duckdb_conn) | Hide output | sql

count_star() i64
159,575,088

1 row, 1 column

Page 1 of 1

Download

```python
import marimo as mo
import duckdb

duckdb_conn = duckdb.connect("trains.db")
```

```sql
CREATE TABLE services AS
FROM 'services-*.csv';
```

Output variable: _df | DuckDB (duckdb_conn) | Hide output | sql

```sql
SELECT count(*) FROM services;
```

Output variable: _df | DuckDB (duckdb_conn) | Hide output | sql

count_star()
i64
159,575,088

1 row, 1 column

Page 1 of 1

Download

```sql
SELECT
    year("Service:Date") AS year,
    (avg("Stop:Arrival delay") * 60)::DECIMAL(8, 2) AS delay
FROM services
GROUP BY year;
```

152ms

Output variable: delays | DuckDB (duckdb_conn) | Hide output | sql

year i64	delay decimal[8,2]
2,019	38.16
2,020	27.46
2,021	30.35
2,022	39.23
2,023	57.30
2,024	60.19
2,025	52.49
2,026	54.56

8 rows, 2 columns 10 / page Page 1 of 1 Download

```sql
SELECT
    year("Service:Date") AS year,
    (avg("Stop:Arrival delay") * 60)::DECIMAL(8, 2) AS delay
FROM services
GROUP BY year;
```

152ms

Output variable: delays | DuckDB (duckdb_conn) | Hide output | sql

year i64	delay decimal[8,2]
2,019	38.16
2,020	27.46
2,021	30.35
2,022	39.23
2,023	57.30
2,024	60.19
2,025	52.49
2,026	54.56

8 rows, 2 columns | 10 / page | Page 1 of 1 | Download

Table | Bar Chart × | +

Bar

Data | Style

⚠ Charts are not saved.

X-Axis

year

None

Data Type: Categorical

Sort: Ascending

Y-Axis

delay

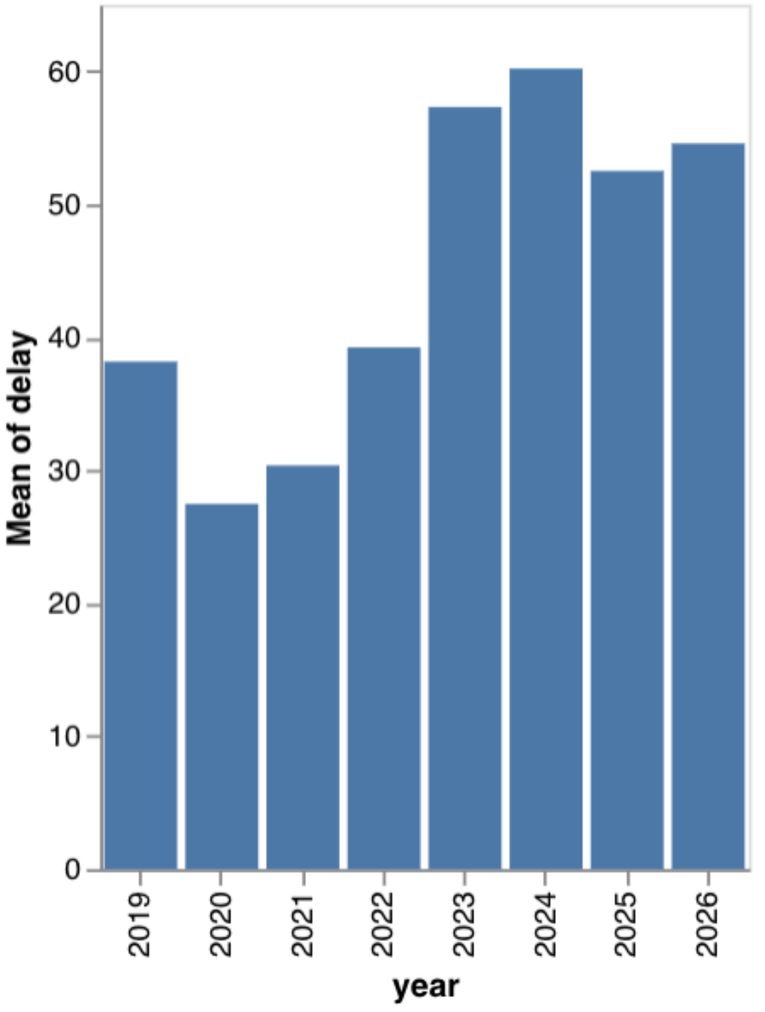
Data Type: # Number

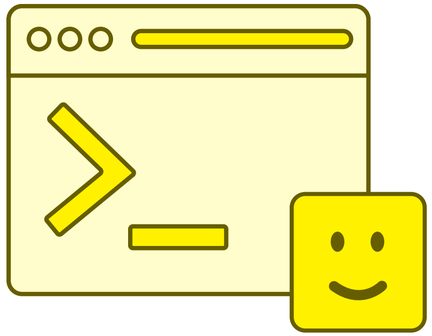
Invert axis

Color by

Select column

Chart | Python code





Friendly SQL
for interactive analysis

Friendly SQL features

```
SELECT
    date,
    station,
    avg(delay),
    max(delay),
FROM 'https://blobs.duckdb.org/data/services.csv.gz'
GROUP BY date, station;
```

Friendly SQL features

```
SELECT
    date,
    station,
    avg(delay),
    max(delay),
FROM 'https://blobs.duckdb.org/data/services.csv.gz'
GROUP BY date, station;
```

Replacement scan

Auto-detection features

Friendly SQL features

```
SELECT
    date,
    station,
    avg(delay),
    max(delay),
FROM 'https://blobs.duckdb.org/data/services.csv.gz'
GROUP BY date, station;
```

Replacement scan

Auto-detection features

Friendly SQL features

```
SELECT
    date,
    station,
    avg(delay),
    max(delay),
FROM 'https://blobs.duckdb.org/data/services.csv.gz'
GROUP BY date, station;
```

- Replacement scan
- Auto-detection features
- Trailing commas

Friendly SQL features

```
SELECT
    date,
    station,
    avg(delay),
--    max(delay),
FROM 'https://blobs.duckdb.org/data/services.csv.gz'
GROUP BY date, station;
```

- Replacement scan
- Auto-detection features
- Trailing commas

Friendly SQL features

```
SELECT
--    date,
    station,
    avg(delay),
--    max(delay),
FROM 'https://blobs.duckdb.org/data/services.csv.gz'
GROUP BY date, station;
```

- Replacement scan
- Auto-detection features
- Trailing commas

Friendly SQL features

```
SELECT
--    date,
    station,
    avg(delay),
--    max(delay),
FROM 'https://blobs.duckdb.org/data/services.csv.gz'
GROUP BY date, station;
```

- Replacement scan
- Auto-detection features
- Trailing commas

Friendly SQL features

```
SELECT
--    date,
    station,
    avg(delay),
--    max(delay),
FROM 'https://blobs.duckdb.org/data/services.csv.gz'
GROUP BY ALL;
```

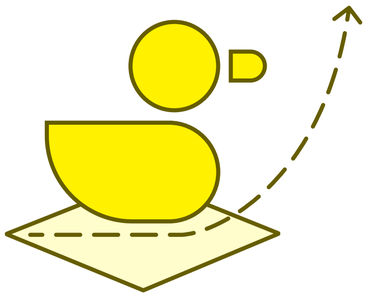
- Replacement scan
- Auto-detection features
- Trailing commas
- `GROUP BY ALL`

Friendly SQL features

```sql
SELECT
--    date,
    station,
    avg(delay),
--    max(delay),
FROM 'https://blobs.duckdb.org/data/services.csv.gz'
GROUP BY ALL;
```

- Replacement scan
- Auto-detection features
- Trailing commas
- `GROUP BY ALL`

DuckDB	databricks	snowflake	PostgreSQL	ISO SQL
2022	2023	2023	2026	2028 (?)

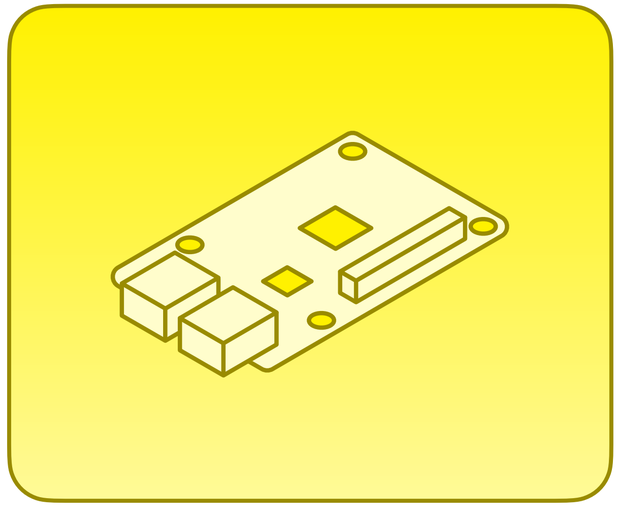


Performance & scalability

TPC-H read queries with DuckDB × Ubuntu

Raspberry Pi

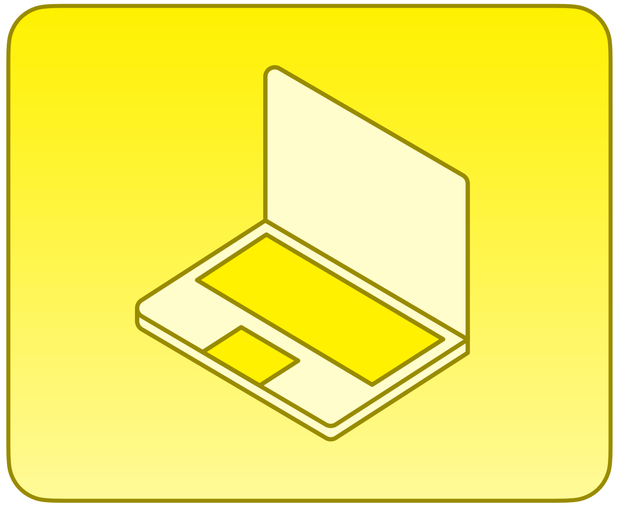
16 GB RAM



1,000 GB CSV

Ultrabook

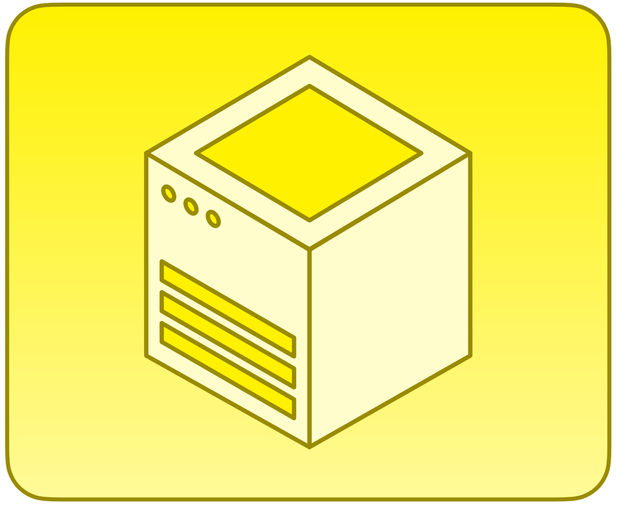
128 GB RAM



10,000 GB CSV

Server

1,536 GB RAM



100,000 GB CSV

Linux kernel performance: TPC-H

24.04 LTS

kernel v6

778k QphH

+9.8%

26.04 LTS beta

kernel v7

855k QphH

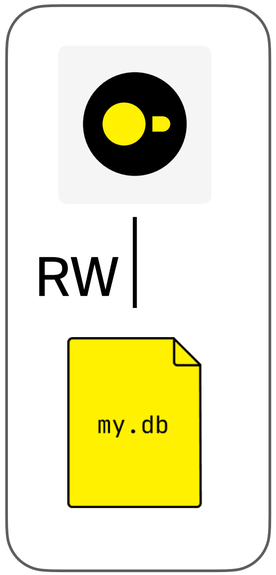
300 GB CSV data | 32 CPU cores, 256 GiB RAM, NVMe disk

In-process DuckDB: Limitations

Single-player mode: only a single process can attach in RW mode

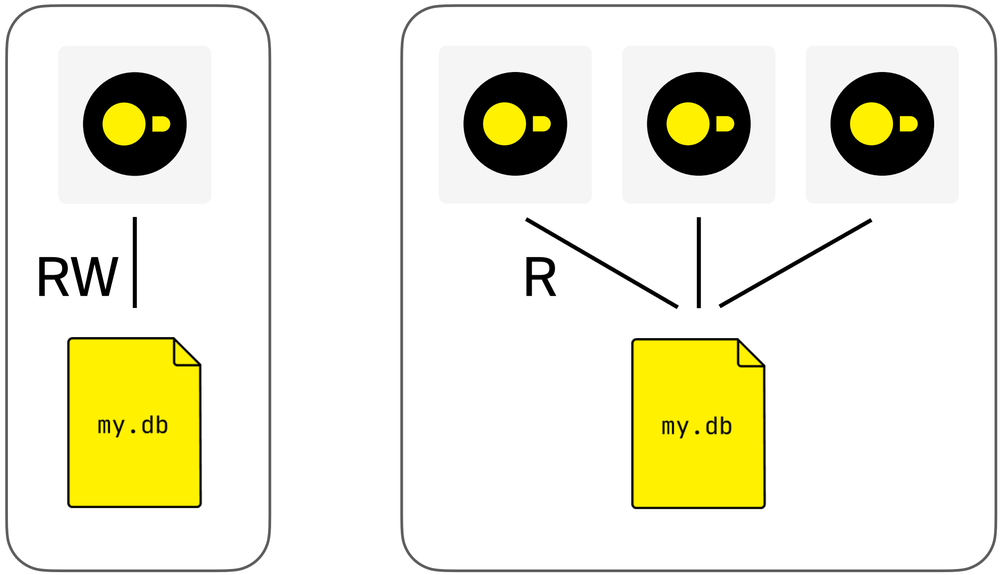
In-process DuckDB: Limitations

Single-player mode: only a single process can attach in RW mode



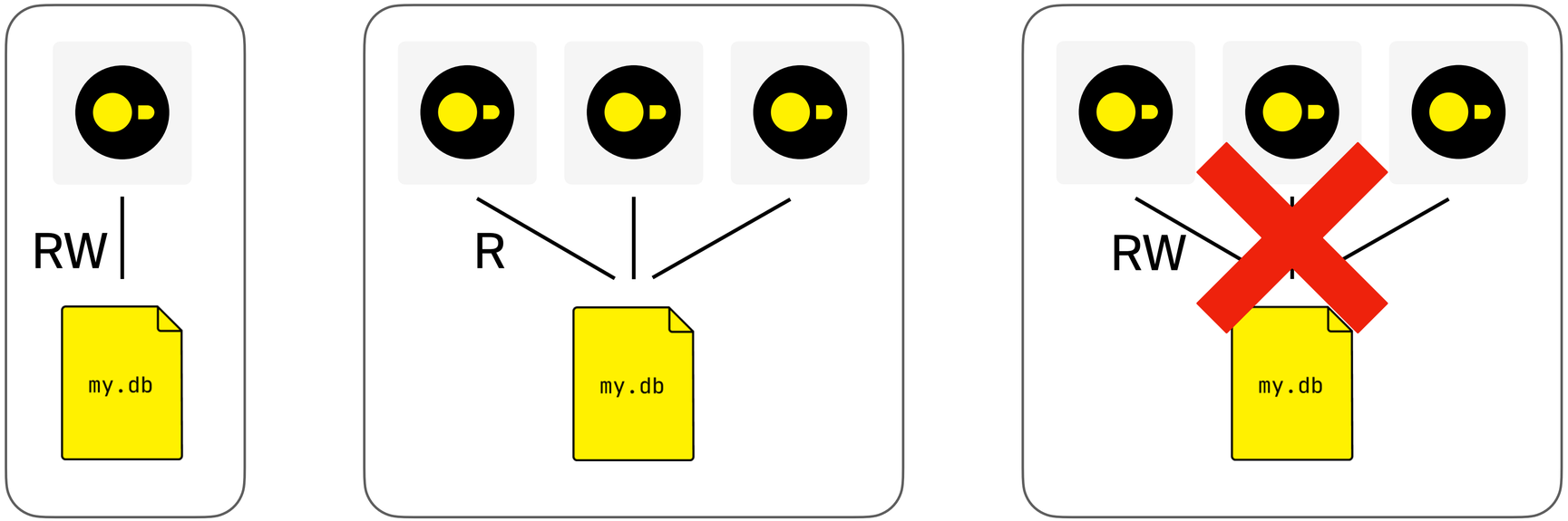
In-process DuckDB: Limitations

Single-player mode: only a single process can attach in RW mode



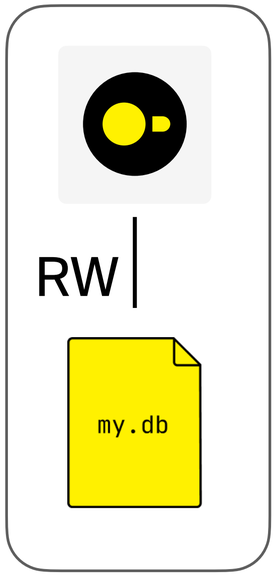
In-process DuckDB: Limitations

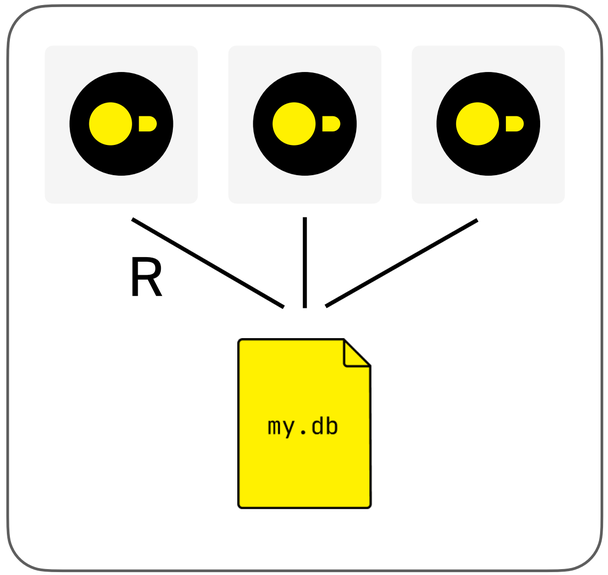
Single-player mode: only a single process can attach in RW mode

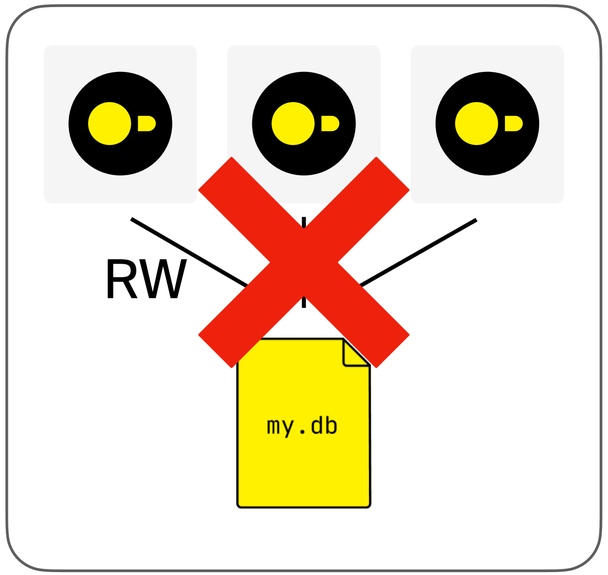


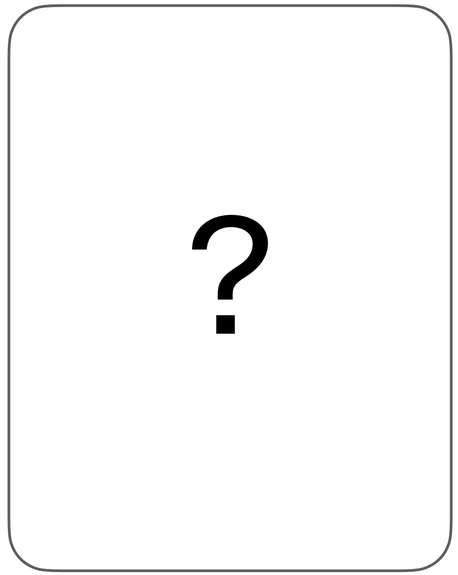
In-process DuckDB: Limitations

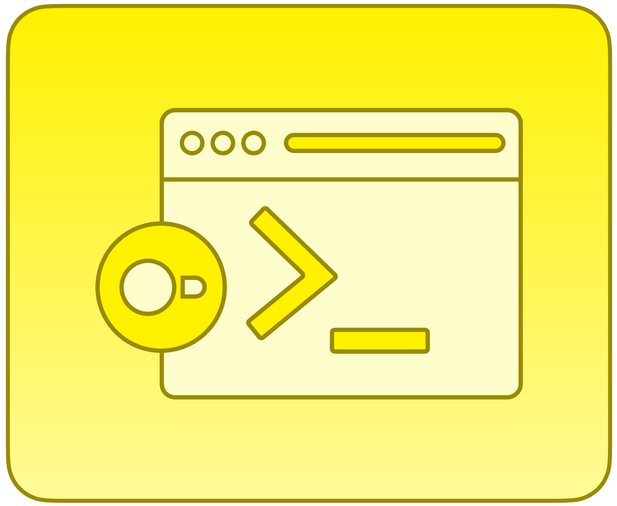
Single-player mode: only a single process can attach in RW mode



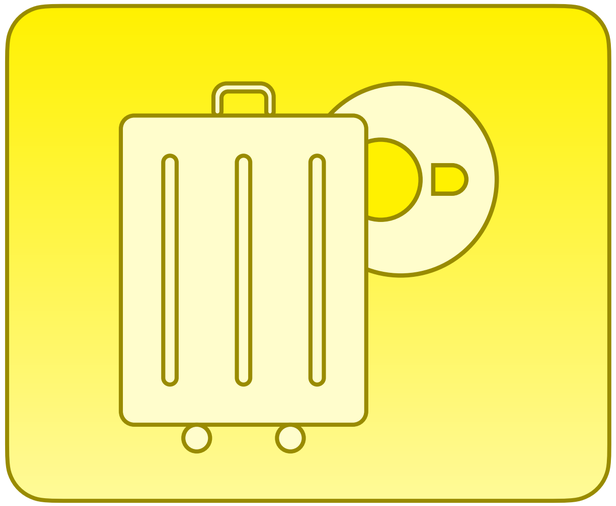




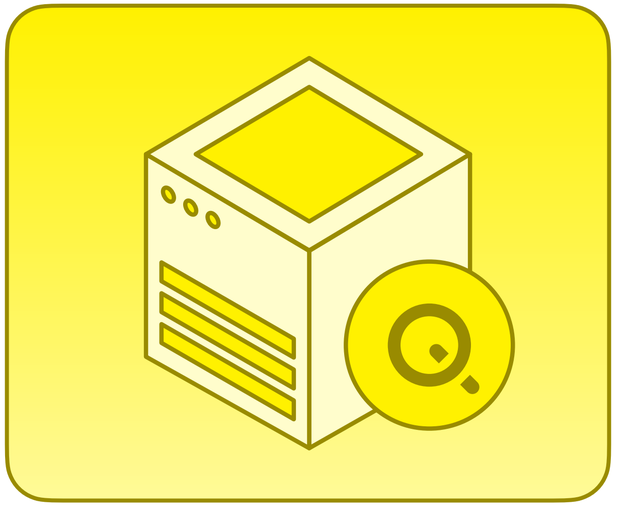




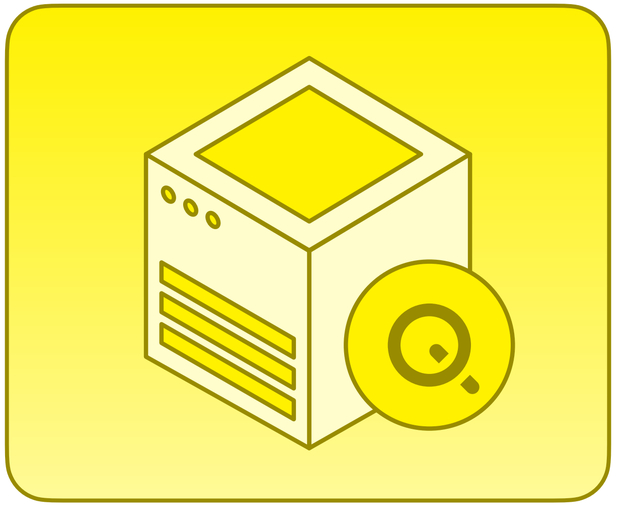
Command-line tool



Portable database

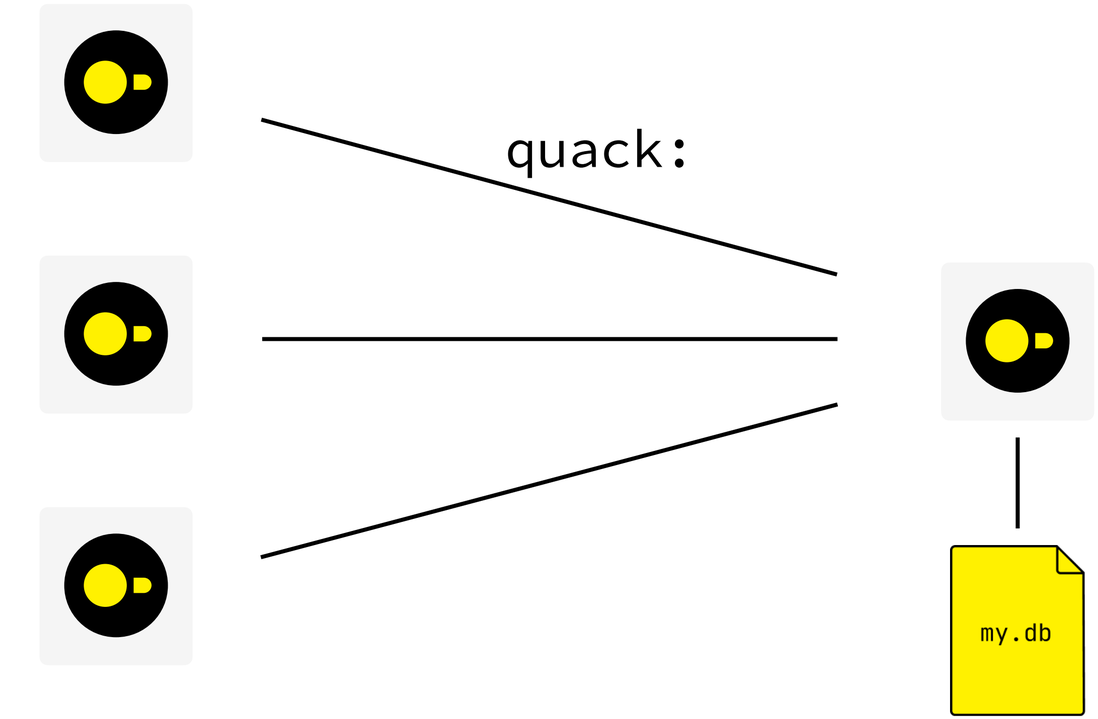


Database server



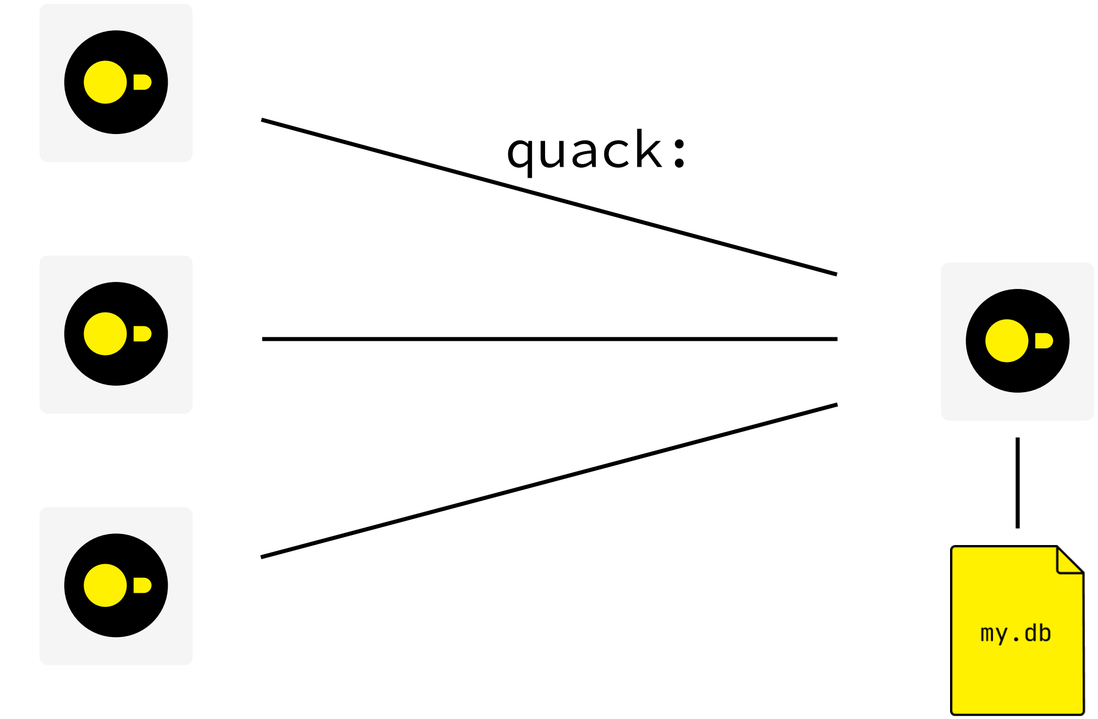
Database server

Client-server DuckDB with Quack



Multi-player mode!

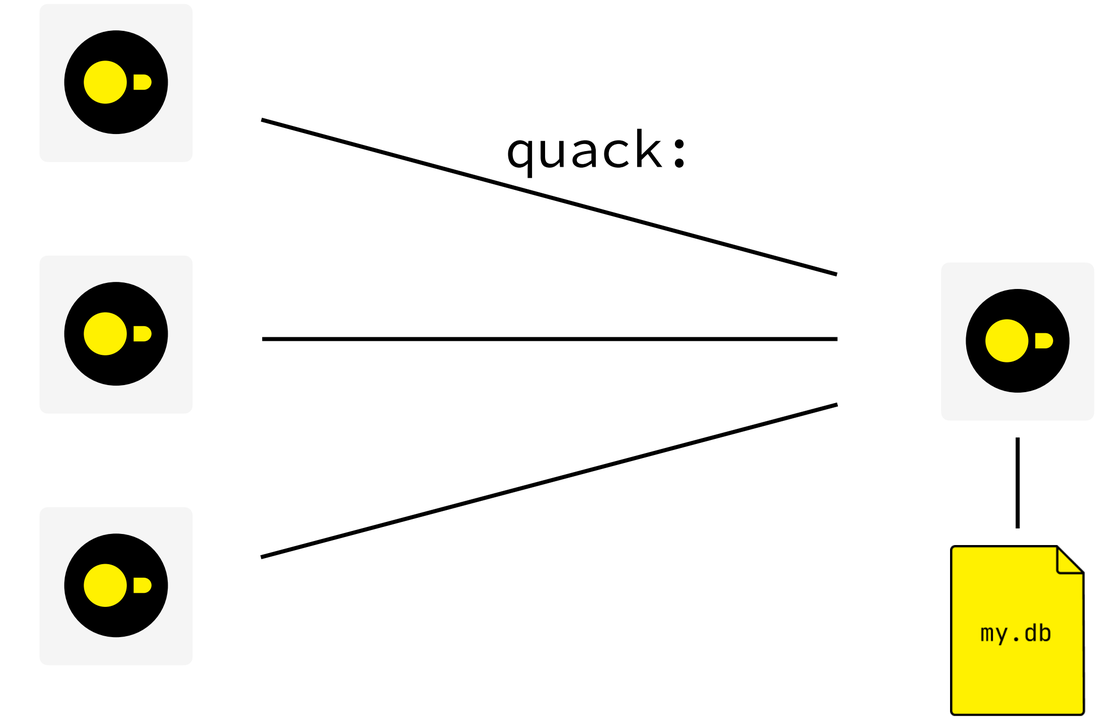
Client-server DuckDB with Quack



Multi-player mode!

- Concurrent RW access
- Clients are DuckDB instances

Client-server DuckDB with Quack



Multi-player mode!

- Concurrent RW access
- Clients are DuckDB instances

- Uses HTTP
- Authentication w/ tokens or BYO
- Authorization w/ callback
- Beta, released in May 2026

Client

Server

Client

Server

```
D CALL quack_serve('quack:localhost', token = 'asdf');
```

listen_uri varchar	listen_url varchar	auth_token varchar
quack:localhost	http://localhost:9494	asdf

Client

Server

```
D CALL quack_serve('quack:localhost', token = 'asdf');
```

listen_uri varchar	listen_url varchar	auth_token varchar
quack:localhost	http://localhost:9494	asdf

```
D CREATE TABLE hello AS
      FROM VALUES ('world') v(s);
```

Client

```
D CREATE SECRET (TYPE quack, TOKEN 'asdf');
```

Success boolean
true

Server

```
D CALL quack_serve('quack:localhost', token = 'asdf');
```

listen_uri varchar	listen_url varchar	auth_token varchar
quack:localhost	http://localhost:9494	asdf

```
D CREATE TABLE hello AS
      FROM VALUES ('world') v(s);
```

Client

```
D CREATE SECRET (TYPE quack, TOKEN 'asdf');
```

Success boolean
true

```
D ATTACH 'quack:localhost' AS remote;
```

Server

```
D CALL quack_serve('quack:localhost', token = 'asdf');
```

listen_uri varchar	listen_url varchar	auth_token varchar
quack:localhost	http://localhost:9494	asdf

```
D CREATE TABLE hello AS
      FROM VALUES ('world') v(s);
```

Client

```
D CREATE SECRET (TYPE quack, TOKEN 'asdf');
```

Success boolean
true

```
D ATTACH 'quack:localhost' AS remote;
D SELECT * FROM remote.hello;
```

s varchar
world

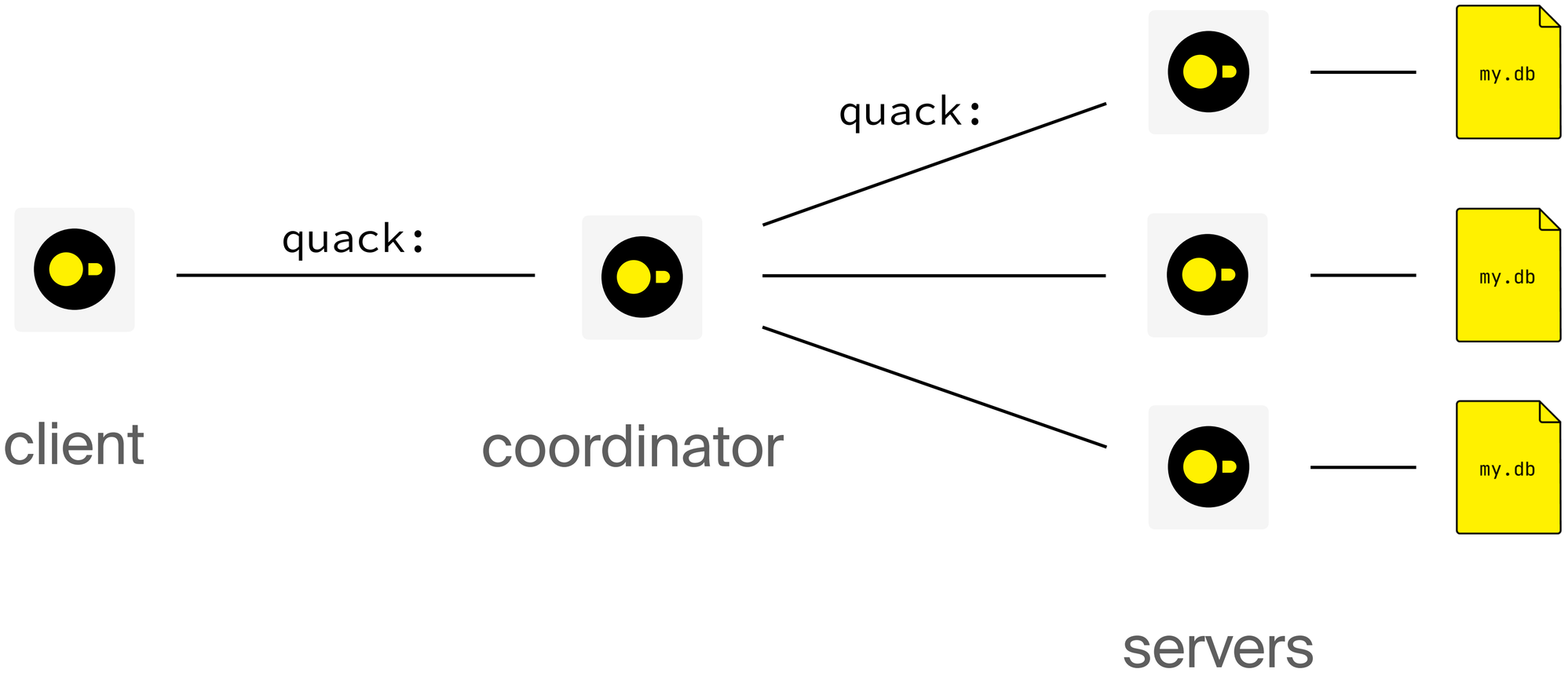
Server

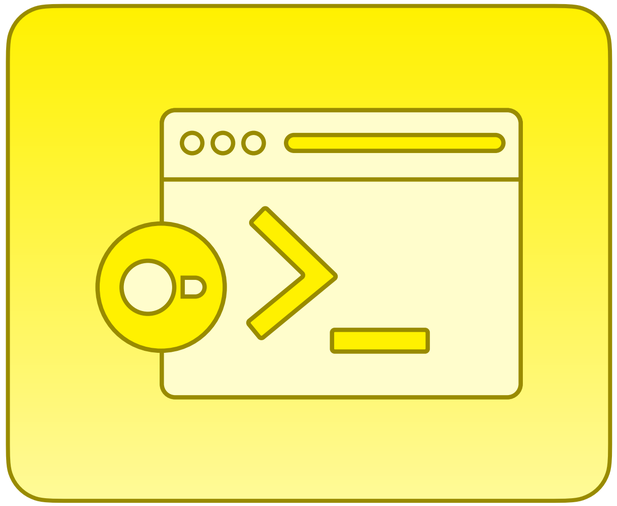
```
D CALL quack_serve('quack:localhost', token = 'asdf');
```

listen_uri varchar	listen_url varchar	auth_token varchar
quack:localhost	http://localhost:9494	asdf

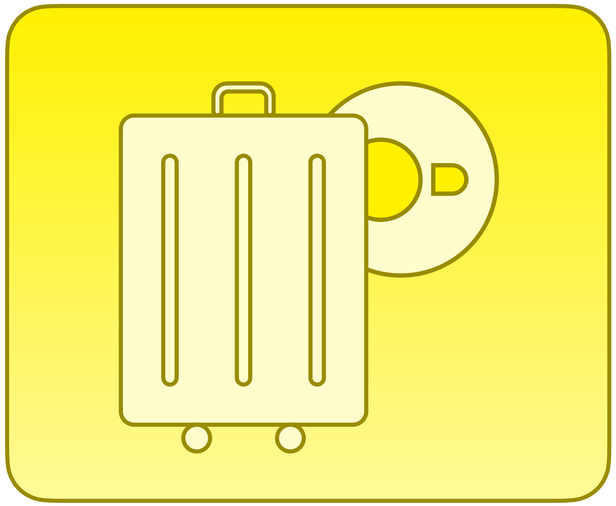
```
D CREATE TABLE hello AS
      FROM VALUES ('world') v(s);
```

Distributed processing with Quack

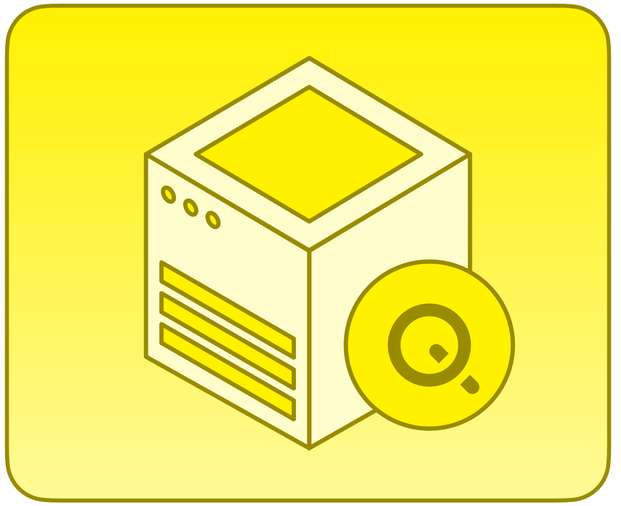




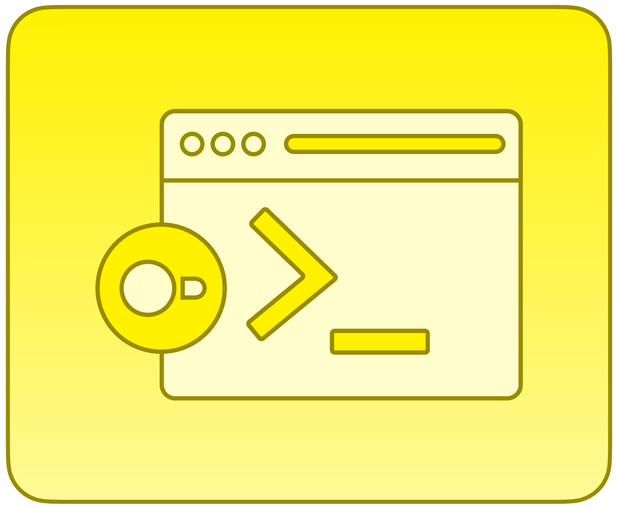
Command-line tool



Portable database

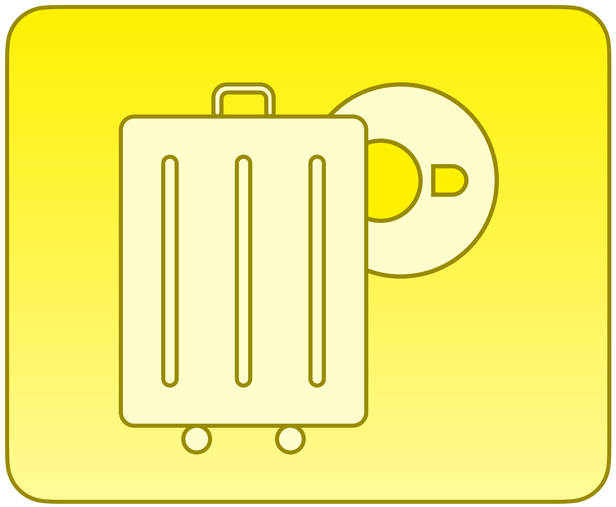


Database server

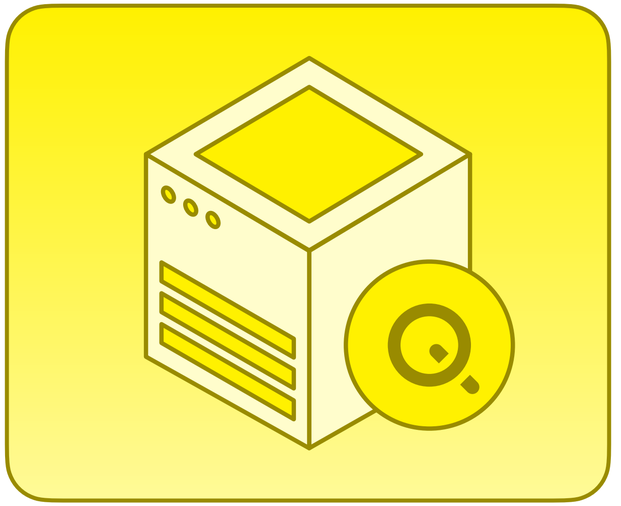


Command-line tool

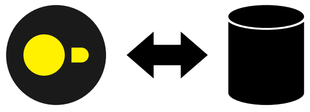
Unix tools and
dataframe libraries

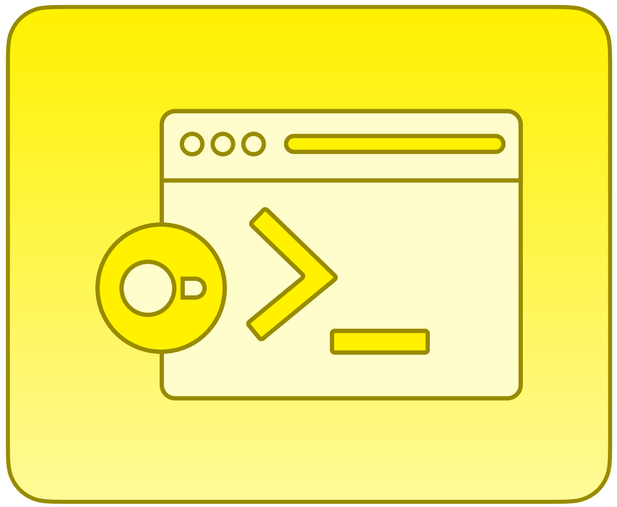


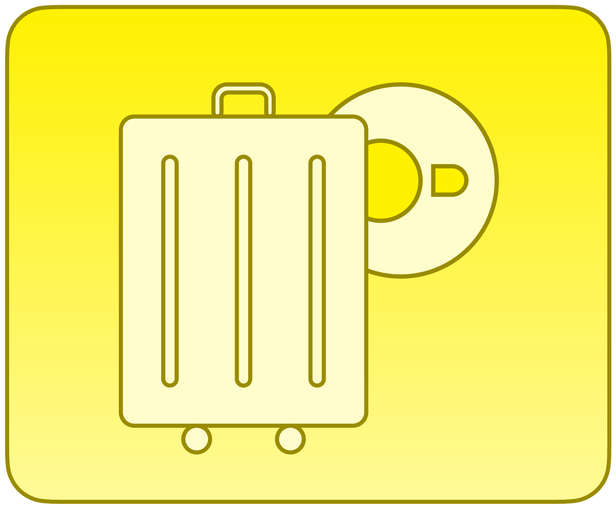
Portable database

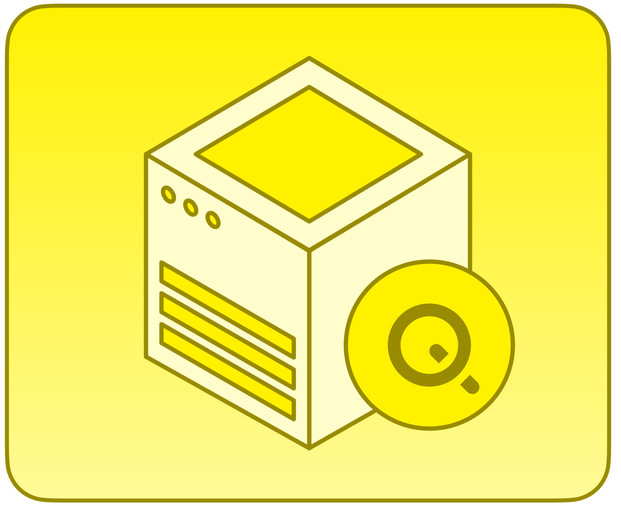


Database server









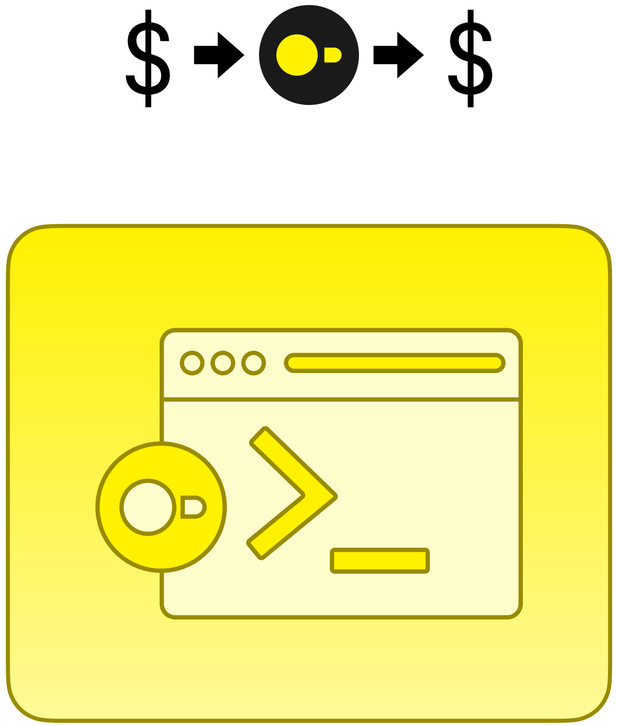
Command-line tool

Unix tools and
dataframe libraries

Portable database

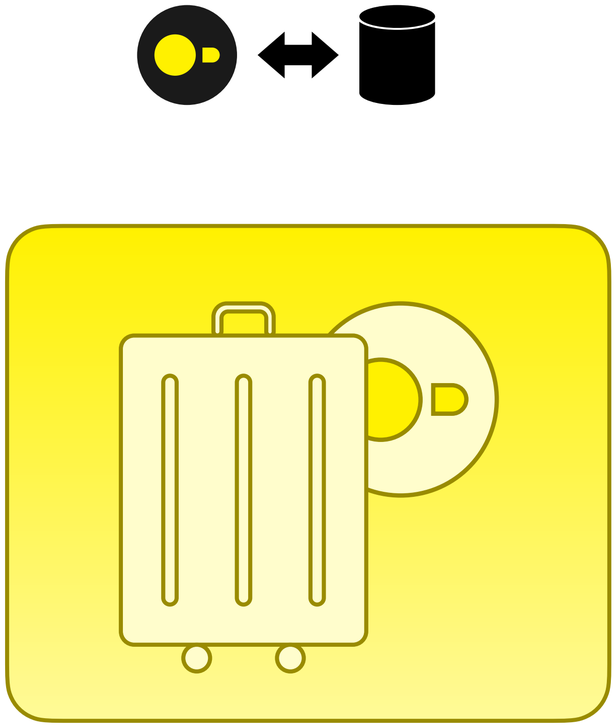
SQLite
for analytics

Database server



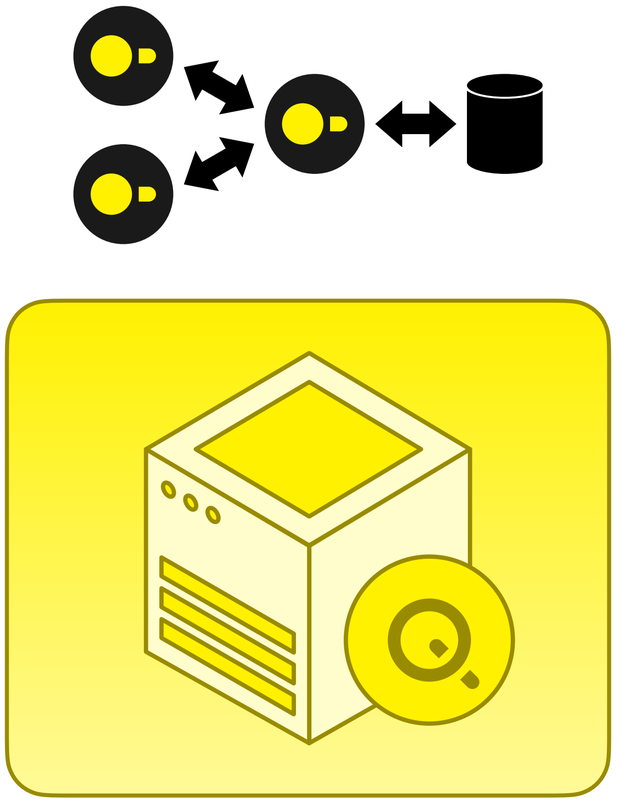
Command-line tool

Unix tools and
dataframe libraries



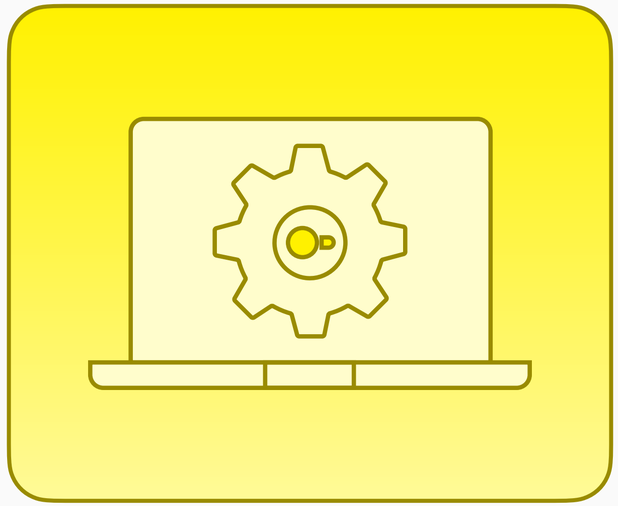
Portable database

SQLite
for analytics



Database server

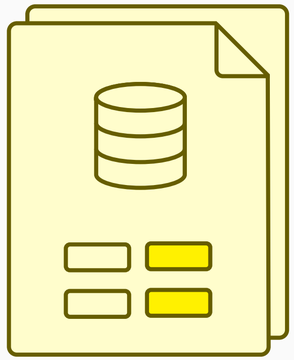
PostgreSQL
for analytics



Deep dive

Storage

Indexing



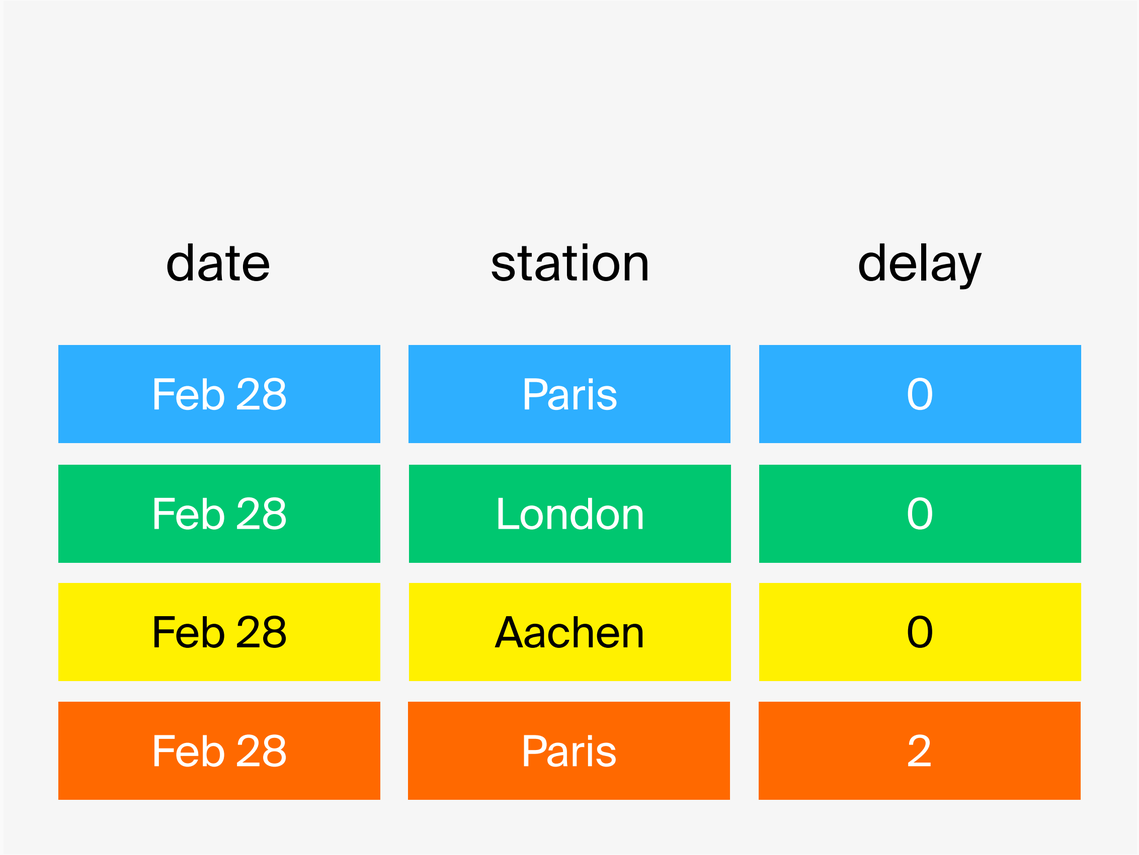
Tables on disk:

Row- or column-based storage?

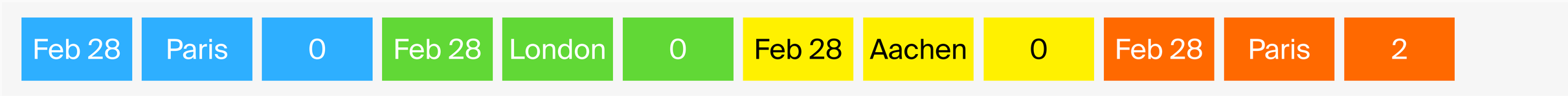
Row-based storage

date	station	delay
Feb 28	Paris	0
Feb 28	London	0
Feb 28	Aachen	0
Feb 28	Paris	2

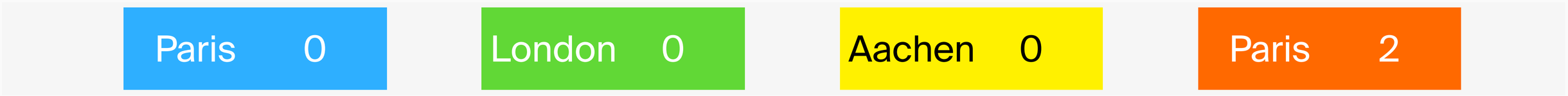
Row-based storage



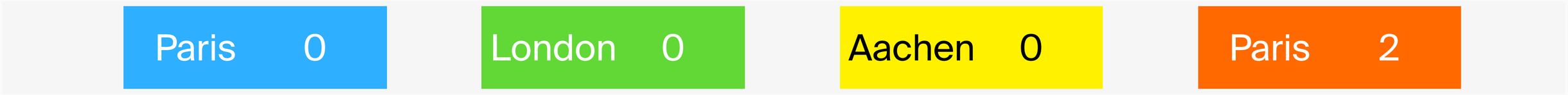
Row-based storage



Row-based storage



Row-based storage



transactional databases, e.g., PostgreSQL

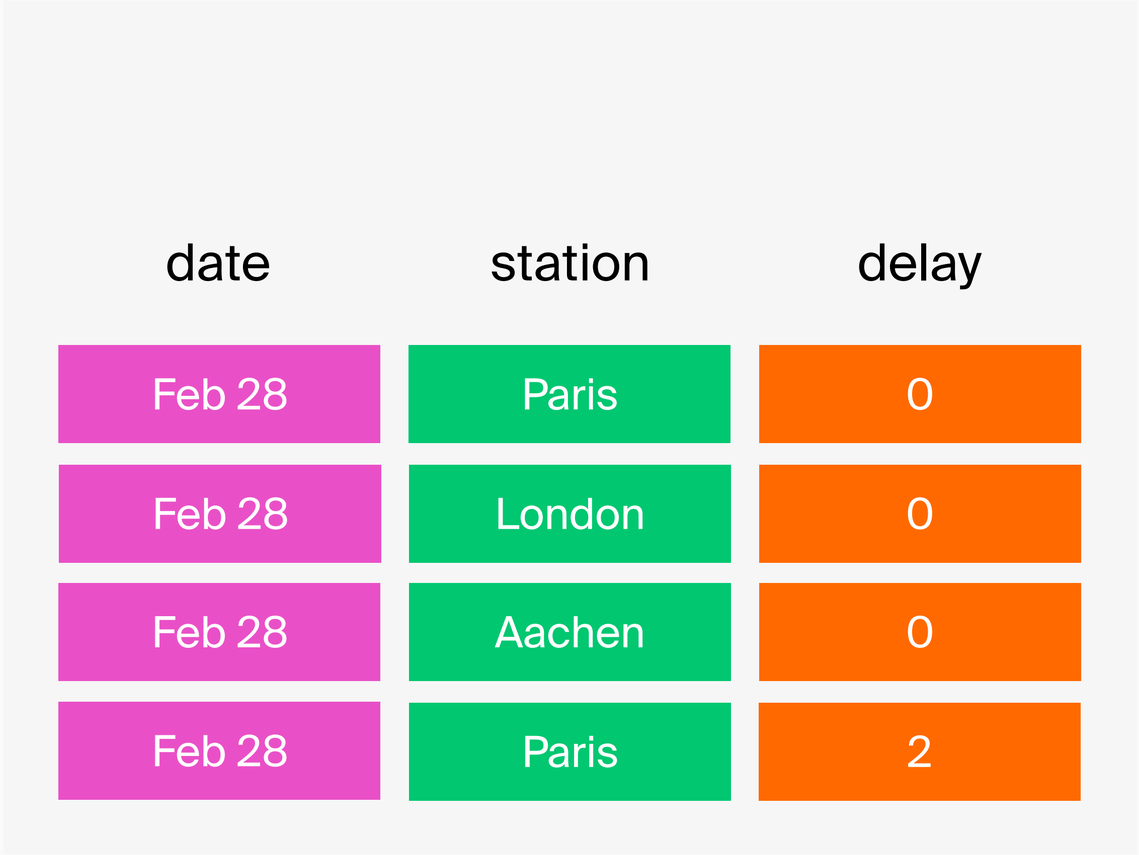
CSV format

Avro format

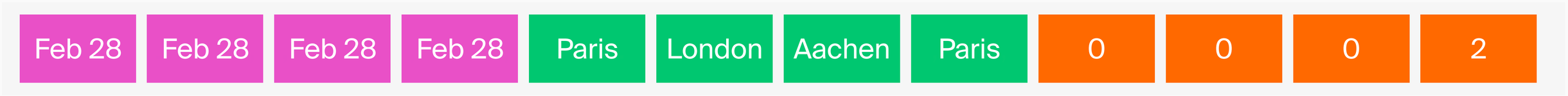
Column-based storage

date	station	delay
Feb 28	Paris	0
Feb 28	London	0
Feb 28	Aachen	0
Feb 28	Paris	2

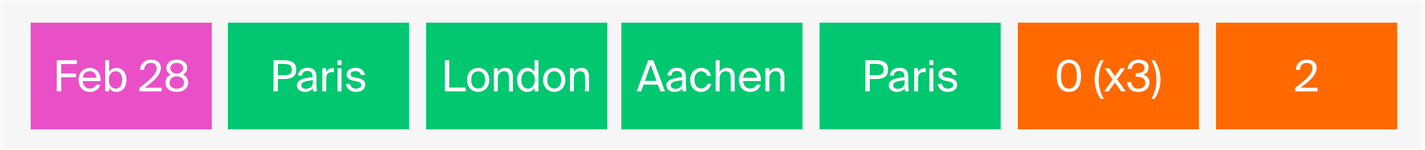
Column-based storage



Column-based storage



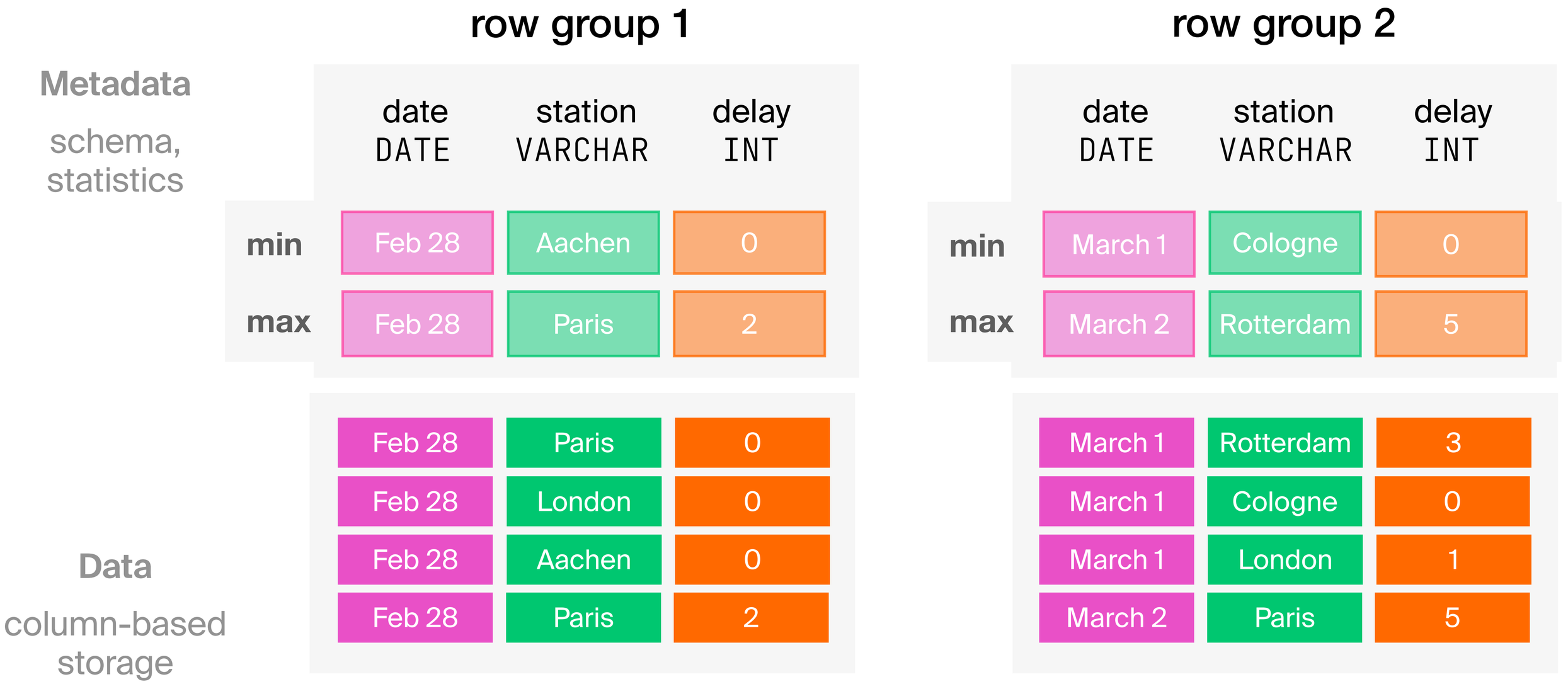
Column-based storage



analytical databases, e.g., DuckDB

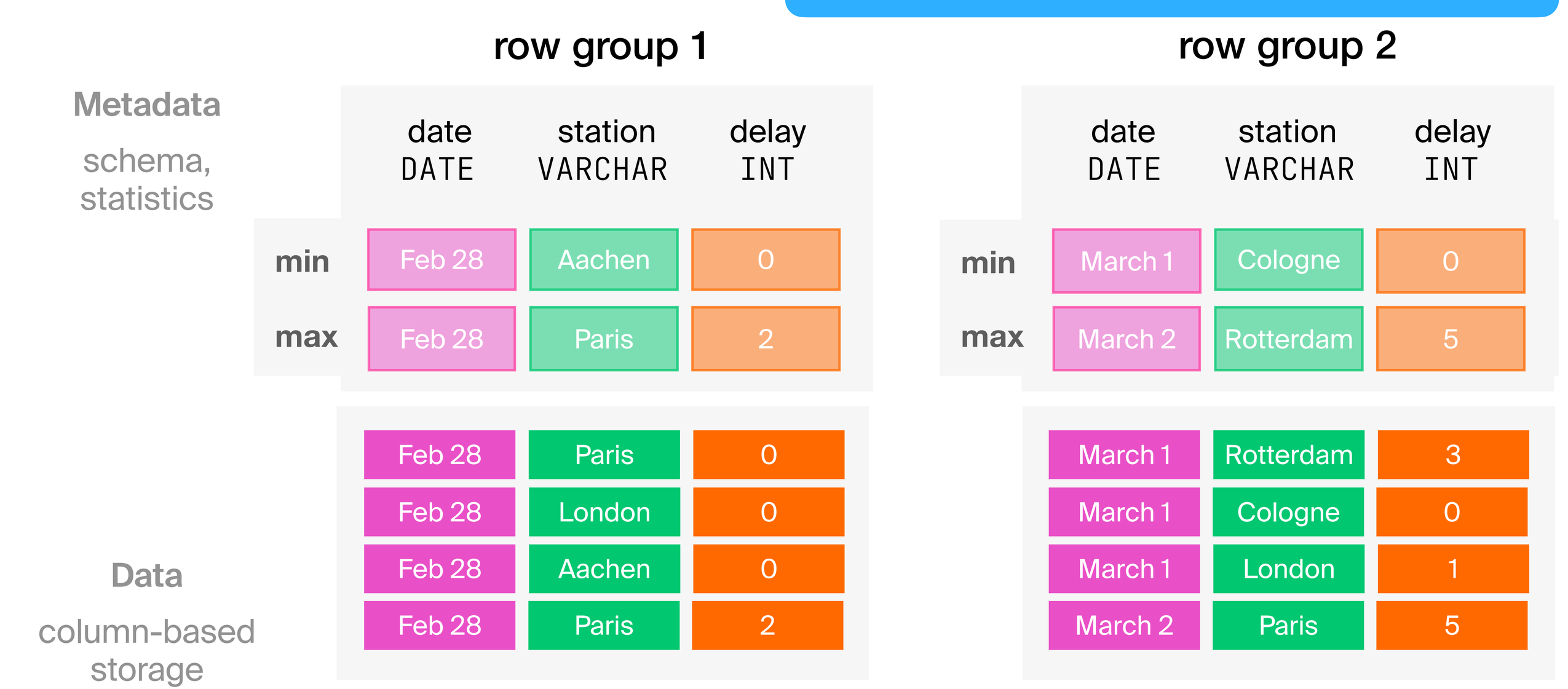
Parquet format

Parquet format



Parquet format

Which train stations had the worst delays on the weekends of Feb 2025?



Parquet format

Which train stations had the worst delays on the weekends of Feb 2025?

Metadata

schema, statistics

Data

column-based storage

Built-in min-max indexes

row group 1

	date DATE	station VARCHAR	delay INT
min	Feb 28	Aachen	0
max	Feb 28	Paris	2

date DATE	station VARCHAR	delay INT
Feb 28	Paris	0
Feb 28	London	0
Feb 28	Aachen	0
Feb 28	Paris	2

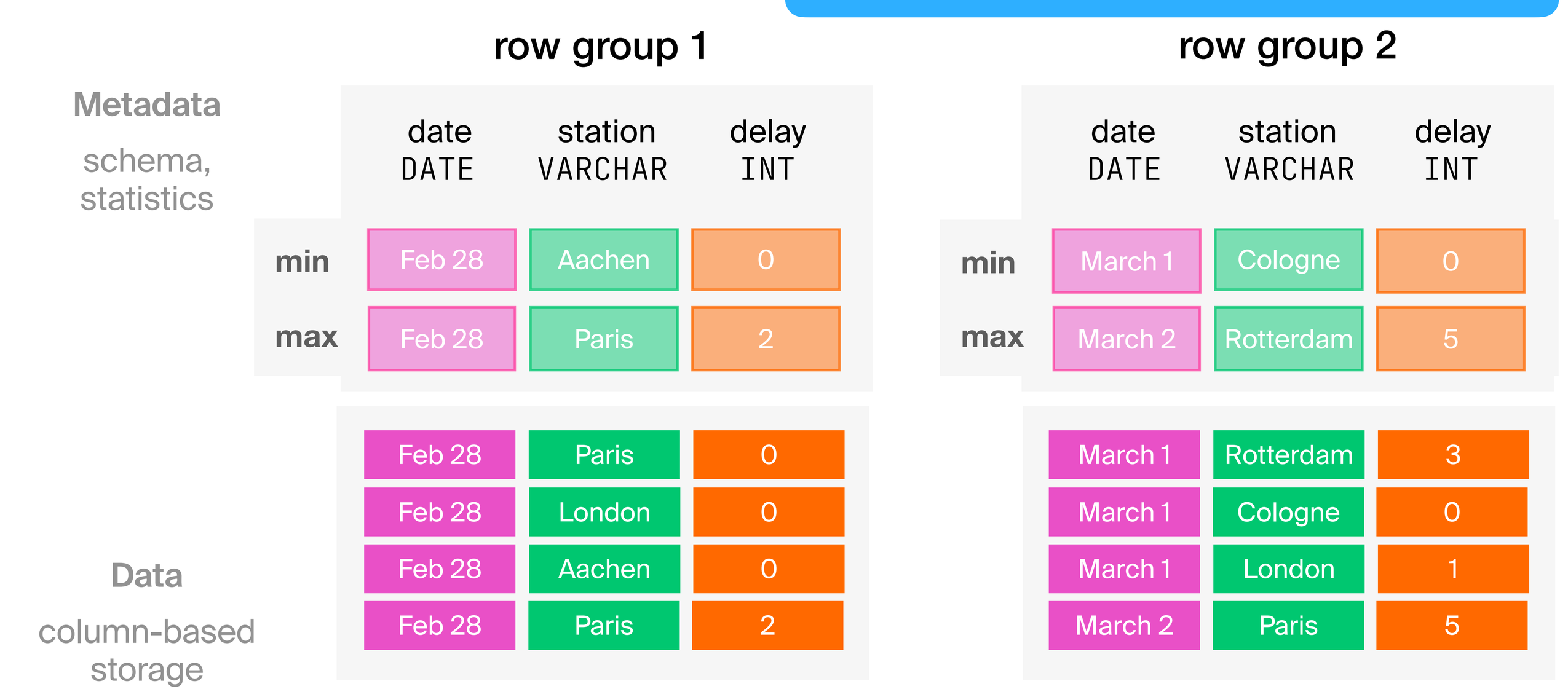
row group 2

	date DATE	station VARCHAR	delay INT
min	March 1	Cologne	0
max	March 2	Rotterdam	5

date DATE	station VARCHAR	delay INT
March 1	Rotterdam	3
March 1	Cologne	0
March 1	London	1
March 2	Paris	5

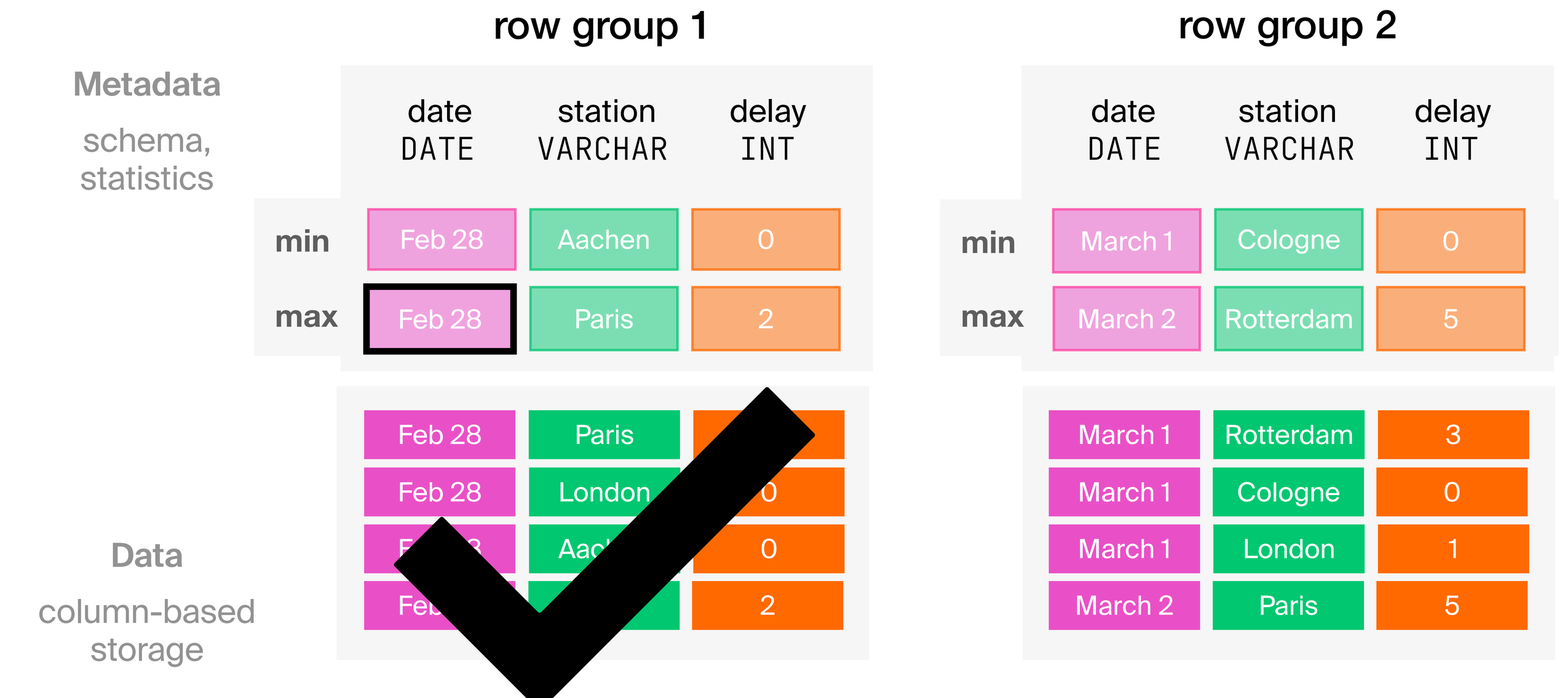
Parquet format

Which train stations had the worst delays on the weekends of Feb 2025?



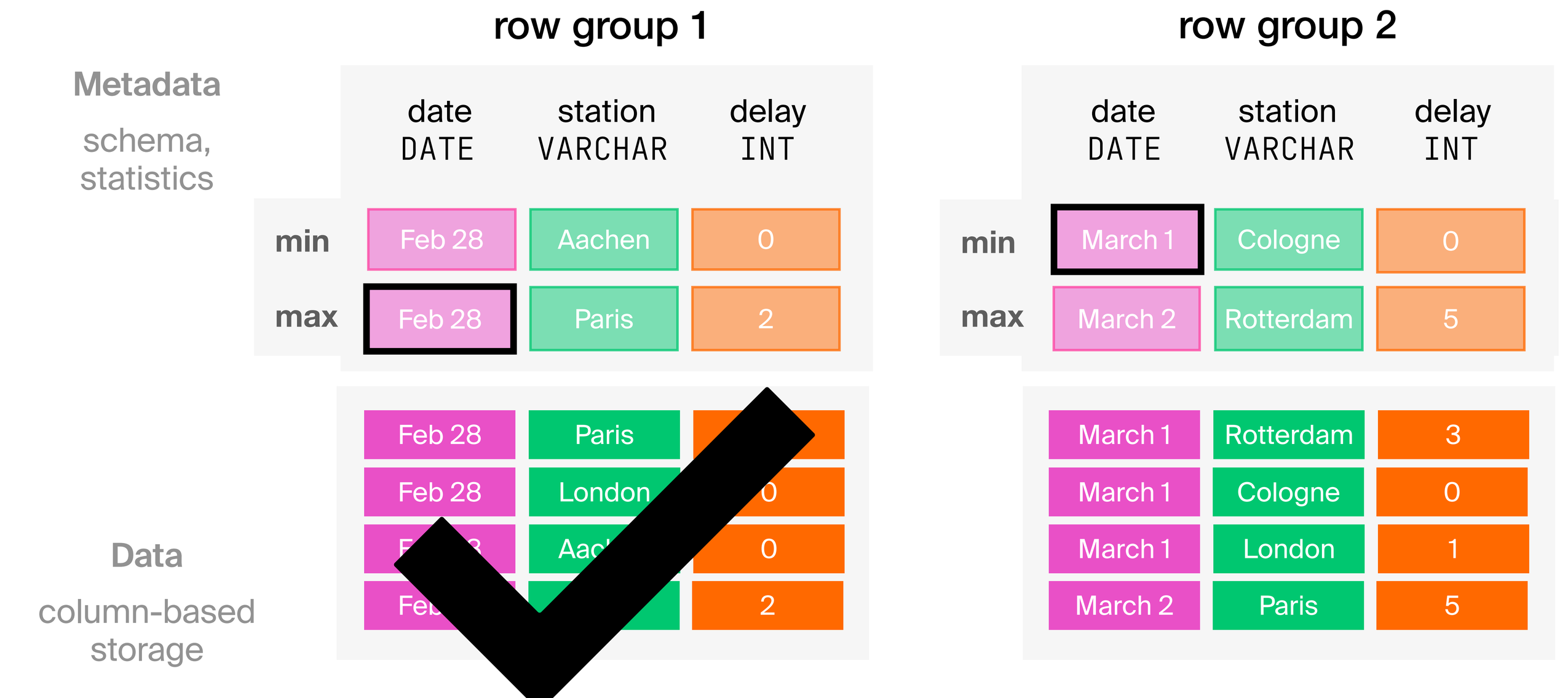
Parquet format

Which train stations had the worst delays on the weekends of Feb 2025?



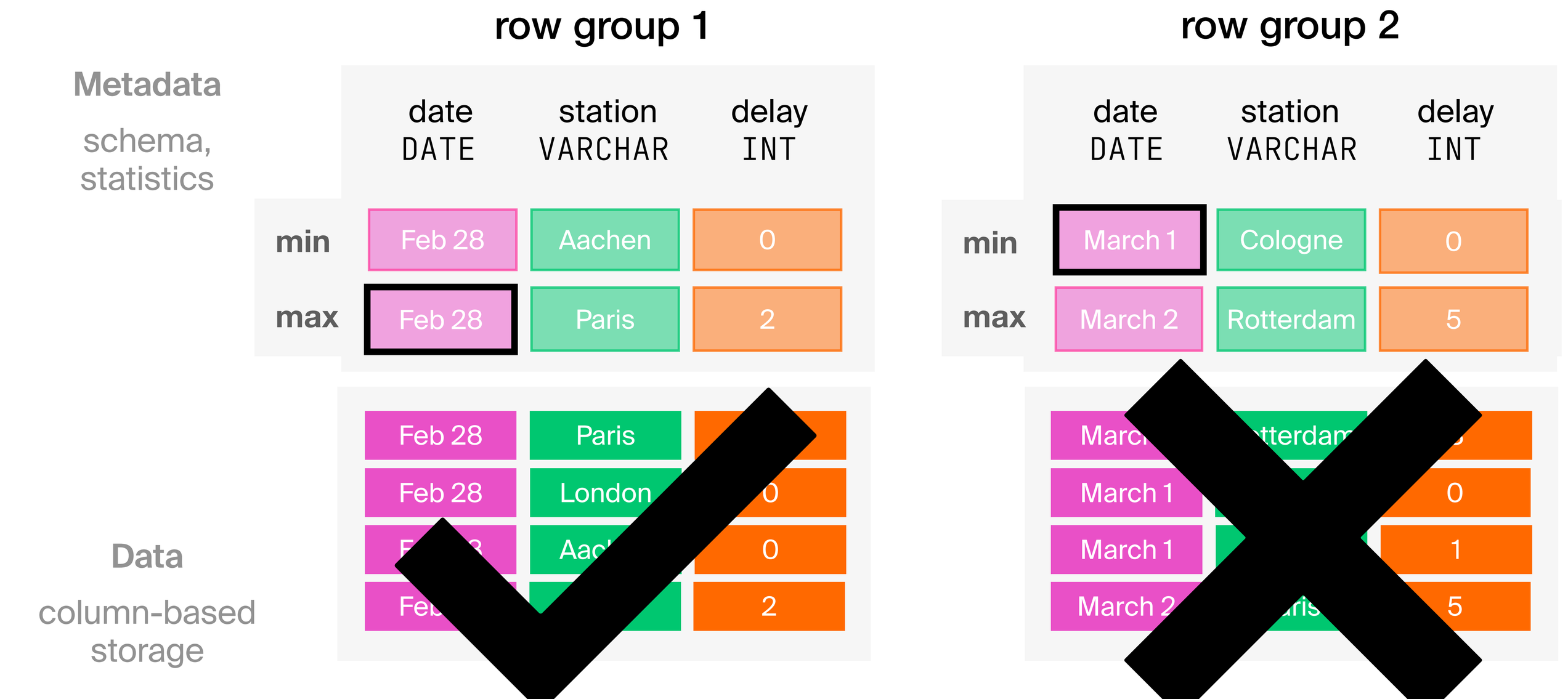
Parquet format

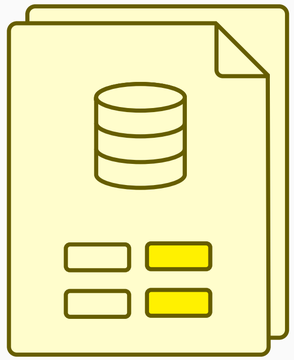
Which train stations had the worst delays on the weekends of Feb 2025?



Parquet format

Which train stations had the worst delays on the weekends of Feb 2025?





DuckDB:

Columnar storage in a single file

```
$ duckdb trains.db

D CREATE TABLE services AS
  FROM 'services.csv';
```

20 s

The min-max indexes are created automatically for each row group

```
$ duckdb trains.db
```

Which train stations had the worst delays on the weekends of Feb 2025?

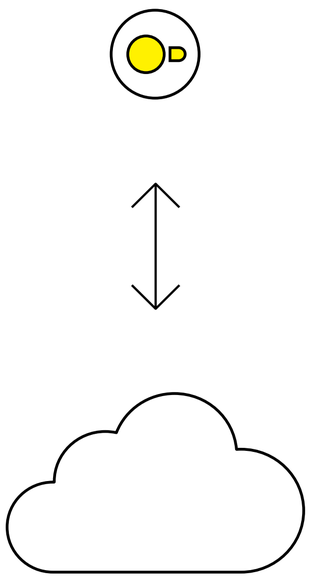
```
$ duckdb trains.db

D SELECT
    station,
    avg(delay) AS avg_delay
  FROM services
  WHERE date BETWEEN '2025-02-01' AND '2025-02-28'
    AND extract(isodow FROM date) IN (6, 7)
  GROUP BY station
  ORDER BY avg_delay DESC
  LIMIT 3;
```

station	avg_delay
Siegburg/Bonn	5.58
Emmerich-Elten	3.97
Brussel-Zuid	3.86

Which train stations had the worst delays on the weekends of Feb 2025?

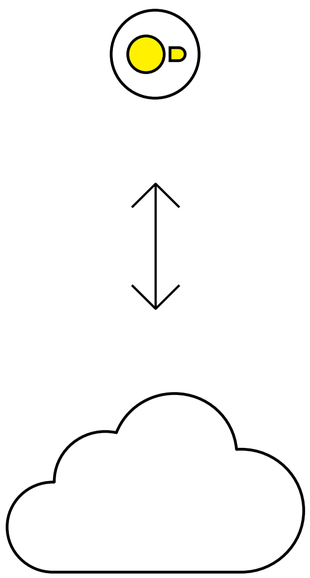
0.03 s

Partial reading with remote endpoints



Which train stations had the worst delays on the weekends of Feb 2025?

Partial reading with remote endpoints



Which train stations had the worst delays on the weekends of Feb 2025?

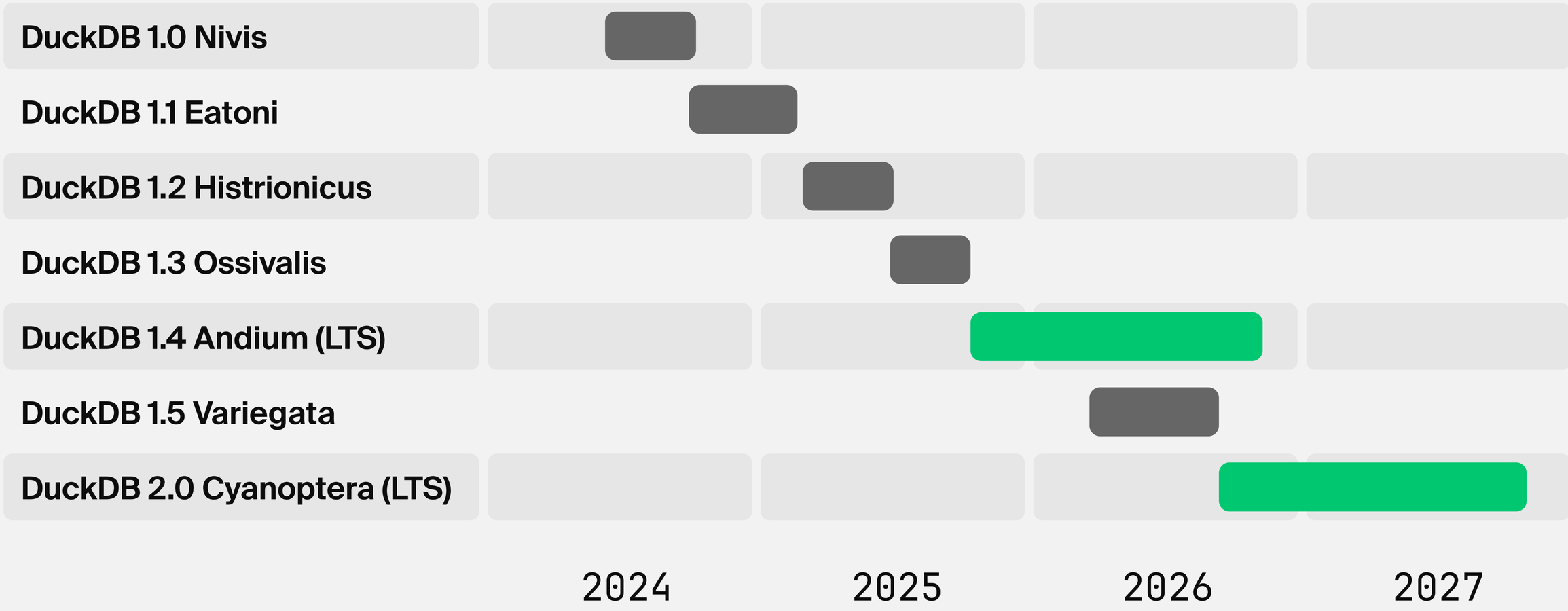
- Column projection and predicate pushdown both work
- Implemented with HTTP range requests
- Works with DuckDB's format, Parquet, etc.

DuckDB in practice

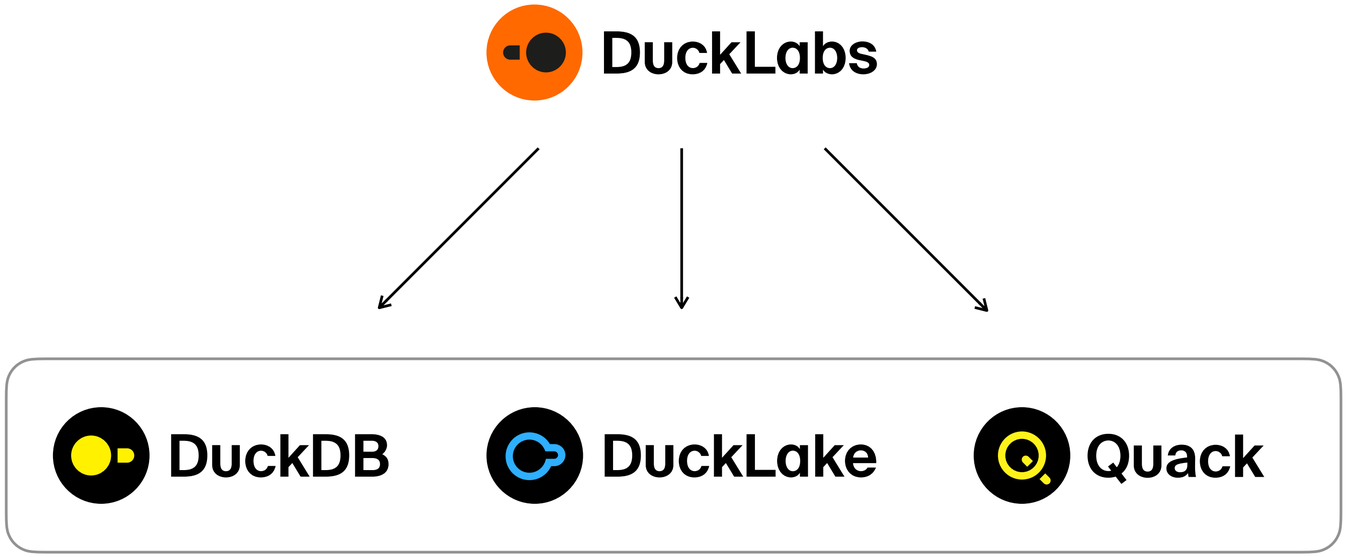
38.5k GitHub stars

100k (human?) website visits / week

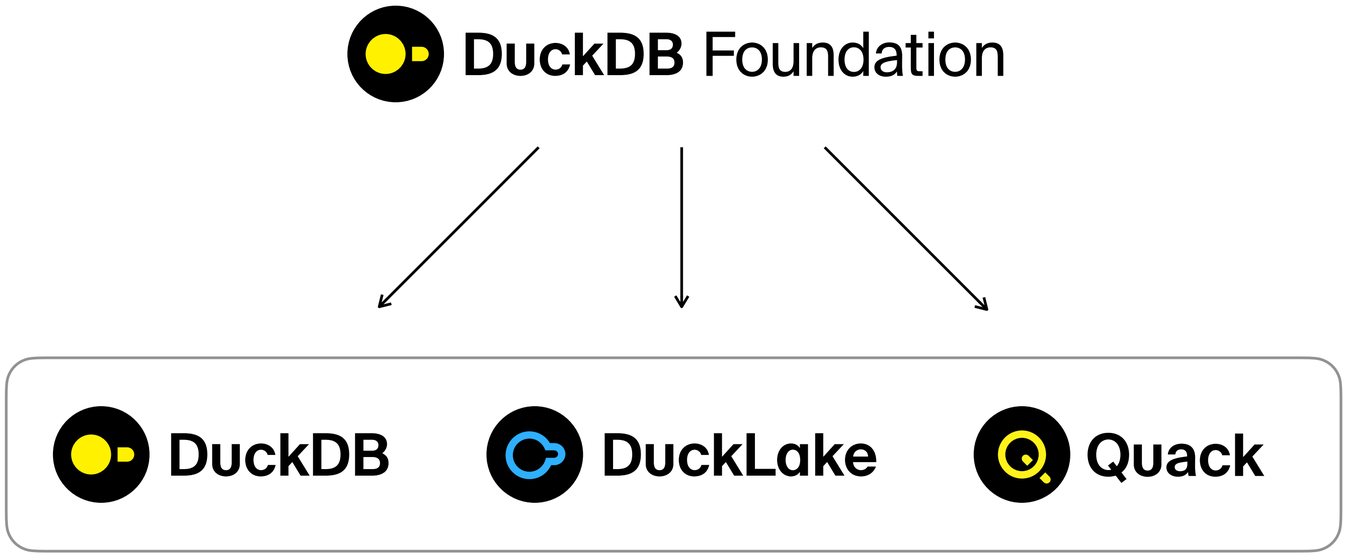
10M+ downloads from PyPI / week



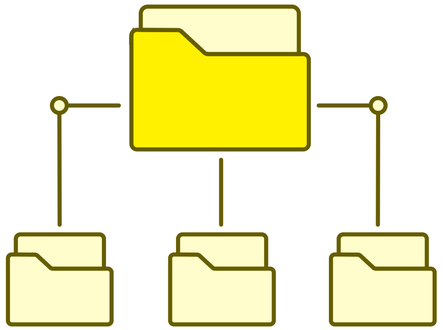
Funding and governance



- Founded in 2021 as “DuckDB Labs”
- Bootstrapped company based in Amsterdam
- Services around the DuckDB, DuckLake, and Quack

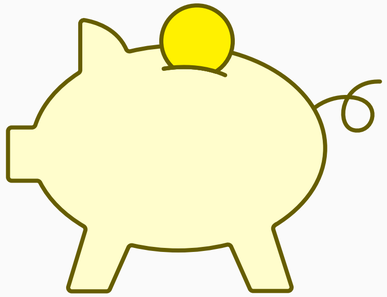


- Owns the intellectual property of the projects
- Ensures the project stays open-source under MIT
- Collects donations



Use cases

Cost saving



Fast local development

save time and network

Replace proprietary systems

save on license cost

Replace distributed systems

save on hardware cost

Last mile analytics

1 PB log →
pre-process with Spark

Process the 200 GB
aggregate DuckDB

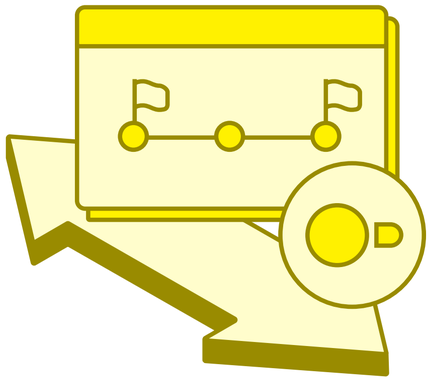
Build fast dashboards
with DuckDB-Wasm

Education

Easy to install

Open-source

No DBA needed



Learning DuckDB

Documentation

Connecting to DuckDB

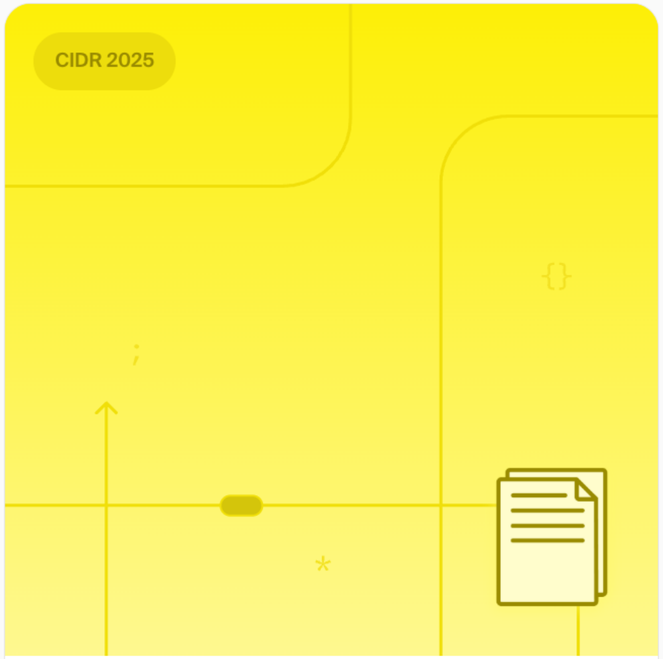
- DuckDB connection overview
- Quack remote protocol

Client APIs

- C
- C++
- CLI (command line interface)
- Go
- Java (JDBC)
- Node.js

DuckDB Library

DuckDB Documentation Resources GitHub 38.4k Support



PAPER

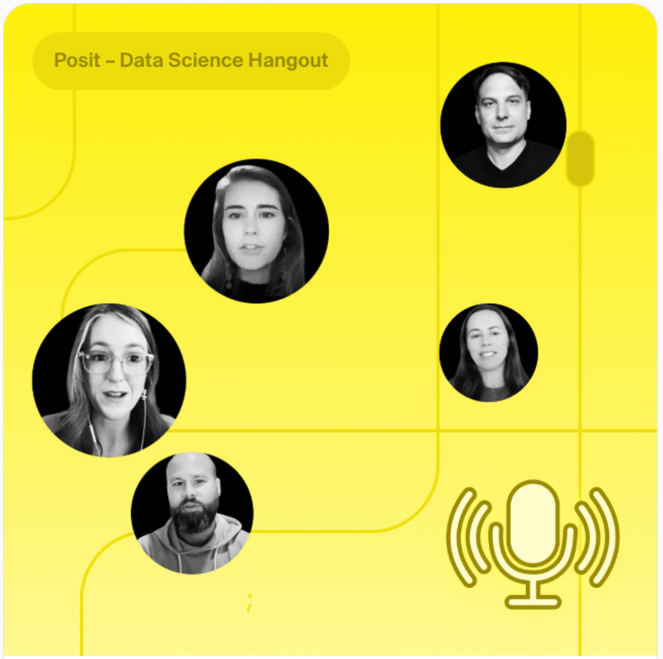
Runtime-Extensible Parsers

2025-01-19 Hannes Mühleisen, Mark Raas...

PAPER

Adaptive Factorization Using Linear-Chained Hash Tables

2025-01-19 Paul Groß, Daniel ten Wolde, P...



PODCAST 60 min

DuckDB and the Future of Databases

2025-01-09 Hannes Mühleisen

Learn SQL and DuckDB internals from U. Tübingen

Tabular Database Systems

Inner Join Follows Foreign Keys

vehicles

vid	vehicle	kind	seats	wheels?	pid
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

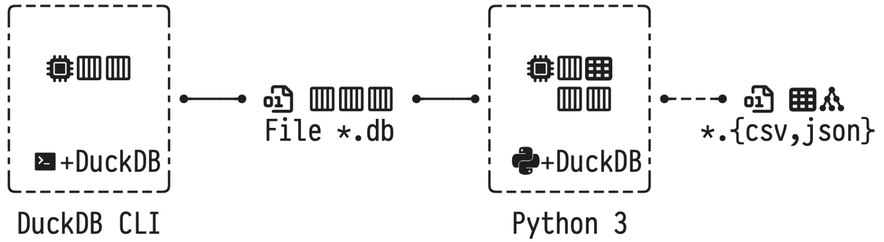
pid	pic	name	born
p_1		Cleo	2013
p_2		Bert	1968
p_3		Drew	▢
p_4		Alex	2002

```
SELECT v.vehicle, p.name AS driver, p.pic
FROM   vehicles AS v INNER JOIN peeps AS p ON (v.pid = p.pid)
                                              "equi-join" ↗
```

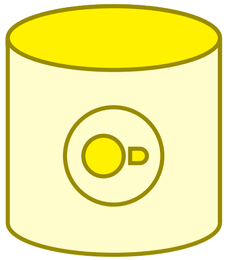
- Some rows in table peeps find multiple *join partners* in vehicles (like p_4 🧔), some find none (p_1 👶).
- General: A join between t_1 and t_2 may yield 0…$|t_1|\times|t_2|$ rows.

Design and Implementation of DuckDB Internals

DuckDB: A Tabular DBMS Inside Your Own Process



- DuckDB kernel and client share a **single process** ⬚.
  - Python: `import duckdb`, C/C++: link with `libduckdb`.
- Table data resides in a **single database file** 🗄 `*.db`, native DuckDB data formats ▥ in file and in RAM are similar.
- DuckDB sees in-process client data and can **directly read/write client data structures** ▦ using SQL (**"zero copy"**, replacement scans).
- DuckDB allocates sizable RAM buffers (but can use disk 💾 for temporary storage if required).



Summary

UBUNTU SUMMIT | 2026-05-28

DuckDB: Not Quack Science

Databases: Not Rocket Science

GÁBOR SZÁRNYAS
(DUCKLABS)