DATATECH

CSD WEBINAIRE

DuckDB

23/06/2026

Kamel Gadouche

Bienvenue au webinaire DuckDB

(10h00-12h00)

- **Présentation de DuckDB**

  Objectif : présenter un aperçu des grandes caractéristiques de DuckDB

  Ce webinaire fait suite au webinaire Parquet/DuckDB (replay disponible)
  https://www.casd.eu/webinaire-casd-data-tech/#ong5
  Des sujets communs ne seront pas détaillés, comme par exemple : mode colonne, encodings, appels depuis R ou python.

- **Mais avant de commencer :**

  Si vous avez des questions : N'hésitez pas à utiliser le bouton Q&A.

  Le Webinaire est enregistré, il devrait être disponible prochainement sur le site du CASD.

CASD Web Data Tech

- Prochains webinaires CASD :
  - Spark en local ou cluster
  - Julia
  - Fonctionnalités avancées DuckDB : window function, update…
  - geoparquet et Sedona (données géolocalisées)
  - Modifications et validations de données VTL-Spark
  - LLM
  - ARROW

Let’s go

CSD WEBINAIRE

Je veux calculer le cumul des ventes par ville à partir d'un beau CSV

Nom	Modifié le	Type	Taille
Table_ventes400M.csv	21/06/2026 15:47	Fichier CSV Micro...	12 369 816 Ko

12 Go

id	ville	montant	jour
0	Lyon	1753	2024-01-01
1	Paris	1810	2024-01-02
2	Nice	710	2024-01-03
3	Lille	1415	2024-01-04

400 Millions de lignes

C:\Temp\DATA\Ventes400M

Je veux calculer le cumul des ventes par ville à partir d'un **plus** beau CSV

Nom	Modifié le	Type	Taille
ventes4B.csv	21/06/2026 12:36	Fichier CSV Microsoft Excel	127 604 426 Ko

127 Go

id	ville	montant	jour
0	Lyon	1753	2024-01-01
1	Paris	1810	2024-01-02
2	Nice	710	2024-01-03
3	Lille	1415	2024-01-04

4 Milliards de lignes

C:\Temp\DATA\Ventes4B

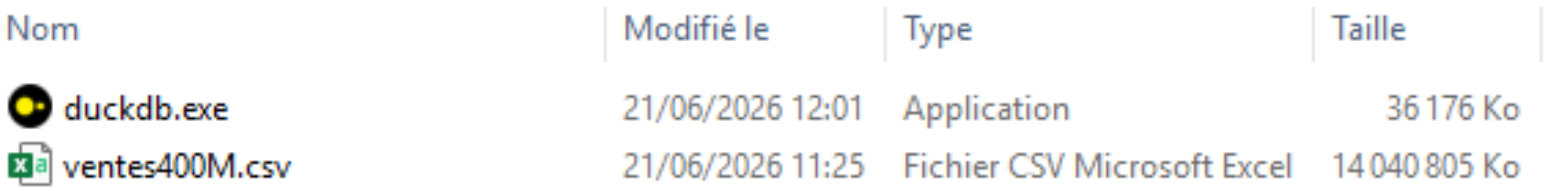
Pas de panique : 1 minute pour récupérer
DuckDB.exe

- **DuckDB a une version CLI (ligne de commande)**

  C'est un exécutable autonome

  Pour le télécharger : https://duckdb.org/install/?platform=windows&environment=cli

- **Je le mets par exemple pour la démo dans le même répertoire**



Conseil : ajouter le chemin C:\... AppData\duckDB-CLI\ dans la variable système PATH pour y avoir accès depuis tout répertoire

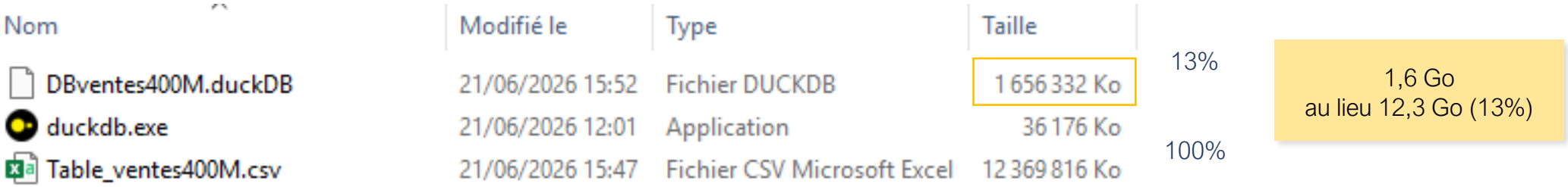
Faire un clic droit sur le répertoire puis choisir 'ouvrir un terminal'

1) Création d'un fichier de base de données DuckDB (penser à .timer on)
PS C:\Temp\DATA\Ventes400M> duckdb ventes400M.duckdb

```
PS C:\Temp\DATA\Ventes400M> duckdb ventes400M.duckdb
DuckDB v1.5.4 (Variegata)
```

2) Ingestion du fichier CSV dans une table ventes
.timer on
create table ventes400 as (from Table_ventes400M.csv);

```
DBventes400M D create table ventes400 as (from Table_ventes400M.csv);
 54% ?                          ?(~18 seconds remaining)                    Memory: 1.2 GB
```



```
.timer on
SELECT ville, AVG(montant)
FROM ventes400
GROUP BY ville;
```

```
DBventes400M D SELECT ville, AVG(montant)
               FROM ventes400
               GROUP BY ville;
```

ville varchar	avg(montant) double
Lyon	9999.9057629
Paris	9999.20096975
Nice	9999.51183082
Lille	9999.78312026

```
CD C:\Temp\DATA\Ventes4B>
duckdb ventes4B.duckdb

.table

.timer on

SELECT ville, AVG(montant)
FROM tventes4B
GROUP BY ville;
```

```
ventes4B D SELECT ville, AVG(montant) FROM tventes4B GROUP BY ville;
┌─────────┬──────────────────┐
│  ville  │ avg(montant)     │
│ varchar │ double           │
├─────────┼──────────────────┤
│ Nice    │ 9999.440145556   │
│ Lille   │ 9999.490115211   │
│ Lyon    │ 9999.550195214   │
│ Paris   │ 9999.430769448   │
└─────────┴──────────────────┘
Run Time (s): real 1.620 user 30.937500 sys 0.234375
```


Pourquoi c'est possible

C\SD WEBINAIRE

DuckDB en bref

In-process

SQL embarqué

Écrit en C++

open source

Mode Colonnes

+ exécution vectorisée
+ encoding

v1.5.x

LTS 1.4 « Andium »

- Système de gestion de base de données relationnel, analytique (OLAP), créé par DuckLabs (2019), 1.0 en juin 2024. Grand fan de sqlite depuis 20 ans : DuckDB est le pendant de sqlite pour l'analytics.
- S'utilise depuis Python, R, Java, C/C++, Node.js, Rust, le CLI… et même dans le navigateur (WASM).
- Conçu pour des requêtes complexes sur des tables de centaines de colonnes et des milliards de lignes.

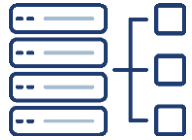
volume de données croissant

Quelques dizaines de To ou plus

Quelques dizaines de Go centaines de Go ou quelques To

Quelques Go

Cluster Spark ou autres clusters
sur plusieurs machines

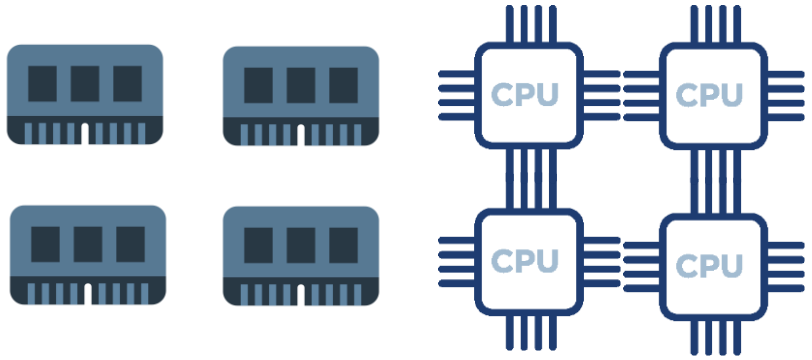


DuckDB

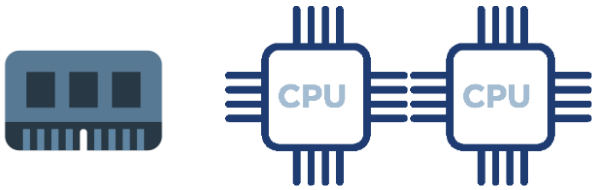
DuckDB comble le vide dataframe ↔ cluster
sur une seule machine

Petites données
Un dataframe en RAM suffit — R, pandas, Excel

DuckDB occupe le « gros secteur du milieu » : SQL embarqué (sans serveur) · mono-machine mais multi-cœur.

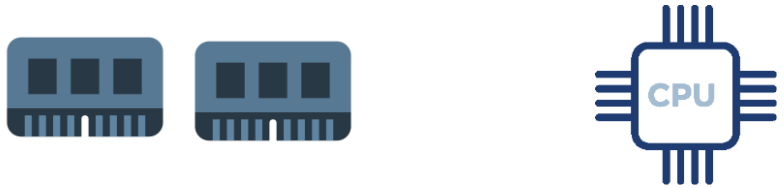


10 To de données :
avec 64 Cœurs et 512 Go a tourné



265 Go de données :
6 millliards de lignes
avec **2 Go de RAM** : Count(distinct)

DuckDB



30Go de Données :
1 millliard de lignes avec un
portable moderne standard

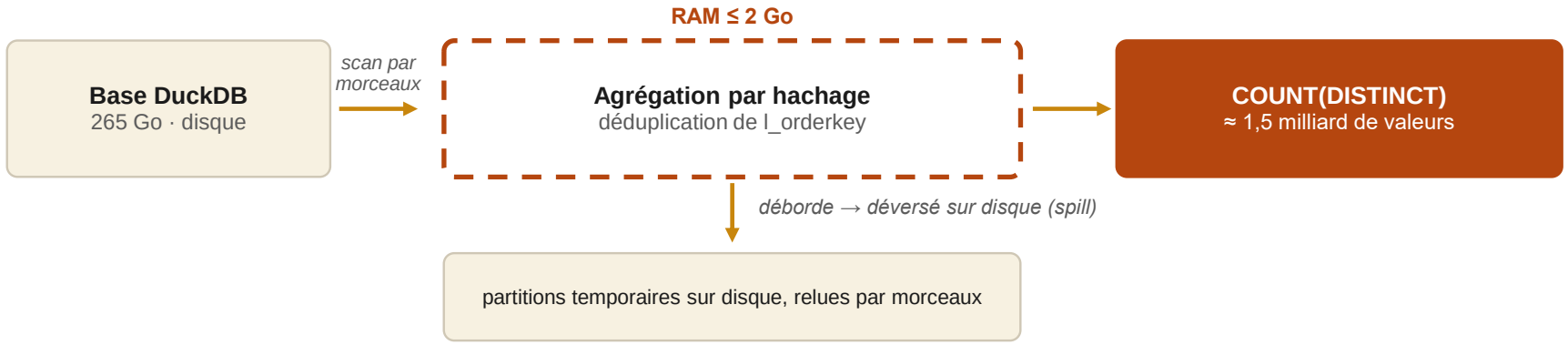
265 Go traités avec 2 Go de RAM

Démo de Hannes Mühleisen (keynote PyData Amsterdam 2025) : un COUNT(DISTINCT) sur 6 milliards de lignes, en mémoire bornée.

265 Go
données — 6 milliards de lignes (TPC-H SF1000)

2 Go
de RAM seulement (ratio ≈ 130:1)

46 s
pour un COUNT(DISTINCT) exact



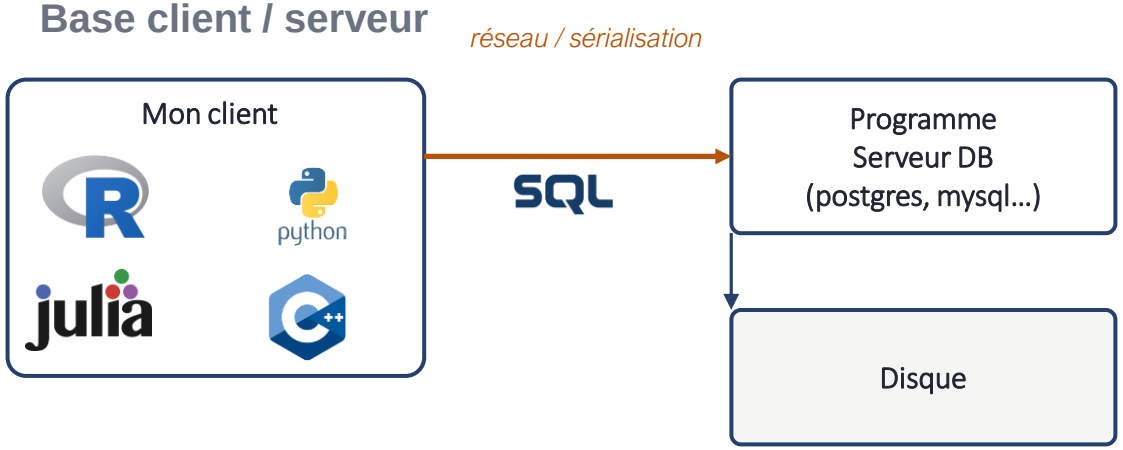
Mémoire bornée au lieu de planter (out-of-memory), DuckDB déverse les intermédiaires sur disque et continue. Une seule machine là où l'on croyait un cluster nécessaire.

Pourquoi DuckDB est rapide ?

Levier	Principe	Gain
Stockage colonnaire	On ne lit que les colonnes utiles ; min/max par bloc (rowgroups)	Moins d'I/O ; blocs entiers sautés
Exécution vectorisée	Blocs de ~2048 valeurs (chunk), boucles SIMD	Débit du C
Larger-than-memory	Déversement sur disque (spilling)	Tri / GROUP BY > RAM sans planter
Parallélisme	Multi-cœur automatique (morsels)	Passage à l'échelle, sans config
Optimisation stockage disque	Encoding (RLE, Bitpacking,…) efficace	Gros gains en stockage et accès disque

In-process : pas de serveur, pas de réseau

Base client / serveur

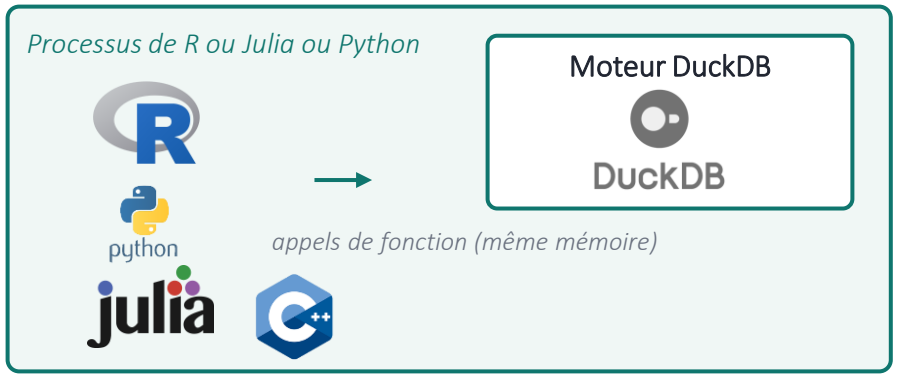


Le moteur de données s'exécute dans un autre processus, voire sur une autre machine.

Les données franchissent la frontière SQL ↔ Python/R/Julia via le réseau ou la communication inter-processus (souvent recopie).

Plus de configuration et un démarrage plus lent.

DuckDB (in-process) dans R ou Julia ou Python



Le moteur s'exécute dans la même mémoire :

aucune copie réseau, aucune sérialisation des résultats.

Les données franchissent la frontière SQL ↔ Python/R/Julia en mémoire partagée (zéro-copie avec Arrow).

Démarrage instantané, latence minimale — idéal pour l'exploration et les pipelines.

Démo SQL en DackDBX

- **DackDBX : Outil Interne CASD**

  Utilitaire C++ graphique compilant le source de DuckDB et de ses principales extensions.

  Intégration du code directement dans l'exécutable

- Tout ce qui sera montré peut être réalisé avec l'UI native de DuckDB ou la CLI.

  Il y a des outils comme Dbeaver (java) aussi qui permettent une expérience interactive ergonomique.

SQL

Comment calculer des rangs, somme cumulée, moyenne mobile, LAG...

C\SD WEBINAIRE

SQL

- SQL langage éprouvé pour consulter, stocker, modifier des données
- Agit comme un proxy entre un programme et les données
- Deux grands usages :
  - Modifications (saisie, acquisition, ingestion…) - OLTP
  - Analyse : analytics, analyser les données – OLAP
- Nous allons essentiellement parler de l'analytics

SQL en bref

- Peu de mot de départ, pour la consultation
  - Par quoi commencer
    - SELECT
  - Ensuite QUOI
    - FROM Liste des colonnes
  - A partir DE QUOI
    - Liste des tables concernées (possibilité de leur donner un alias)
  - Sur quels critères ?
    - WHERE, liste des conditions de ma sélection
  - Il est ensuite possible de faire des fusions de tables, des unions de tables, des agregats etc.

Exemple simple

```
SELECT  id, ville, montant, jour
FROM ventes
WHERE montant > 19990
```

id	ville	montant	jour
550	Nice	19998	2024-07-04
5864	Lyon	19991	2024-01-25
8756	Lyon	19992	2024-12-27
9909	Paris	19998	2024-02-24
17082	Nice	19996	2024-10-19

Exemple jointure

trajets

id_trajet	BIGINT
id_velo	BIGINT
station_depart	INTEGER
station_arrivee	INTEGER
depart	TIMESTAMP
arrivee	TIMESTAMP
duree_s	INTEGER
distance_m	INTEGER
type_velo	VARCHAR

→

stations

id_station	INTEGER
nom	VARCHAR
arrondissement	INTEGER
capacite	INTEGER
longitude	DOUBLE
latitude	DOUBLE

id_trajet	id_velo	station_depart	station_arrivee	depart	arrivee	duree_s	distance_m	type_velo	mois
2	13065	11	6	2026-01-08 17:33:03.000000	2026-01-08 17:45:19.000000	736	2650	électrique	2026-01
5	11776	4	12	2026-01-08 06:32:33.000000	2026-01-08 06:45:46.000000	793	3295	électrique	2026-01
9	7572	1	9	2026-01-19 17:16:08.000000	2026-01-19 17:41:37.000000	1529	4960	mécanique	2026-01
12	18327	8	22	2026-01-26 17:16:53.000000	2026-01-26 17:48:47.000000	1914	7497	mécanique	2026-01

```
SELECT t.id_trajet, t.id_velo, t.type_velo, t.depart, t.arrivee, t.duree_s, t.distance_m,
sd.nom AS station_depart,
sd.arrondissement AS arr_depart,
sa.nom AS station_arrivee,
sa.arrondissement AS arr_arrivee FROM trajets t
JOIN stations sd ON t.station_depart = sd.id_station
JOIN stations sa ON t.station_arrivee = sa.id_station;
```

id_trajet	id_ve...	type_velo	depart	arrivee	duree_s	distance_m	station_depart	arr_depart	station_arrivee	arr_arrivee
4941018	19561	mécanique	2026-01-27 18:42:34.000000	2026-01-27 19:30:39.000000	2885	9103	République	3	Porte de Versailles	15
4941020	17865	électrique	2026-01-30 18:56:16.000000	2026-01-30 19:07:42.000000	686	3077	Hôtel de Ville	4	Opéra	9
4941021	16270	électrique	2026-01-24 14:15:05.000000	2026-01-24 14:22:39.000000	454	1853	Gare de Lyon	12	Bibliothèque F. Mitterrand	13
4941031	5285	électrique	2026-01-31 12:47:04.000000	2026-01-31 12:57:53.000000	649	2815	Concorde	8	Tour Eiffel	7
4941035	155	mécanique	2026-01-21 17:01:26.000000	2026-01-21 17:30:24.000000	1738	6081	Denfert-Rochereau	14	République	3

Exemple agrégat

```
SELECT ville, AVG(montant)
FROM ventes400
GROUP BY ville;
```

ville	avg(montant)
Nice	9999.51
Lille	9999.78
Paris	9999.20
Lyon	9999.91

```sql
WITH trajets AS (
    SELECT *
    FROM trajet
    WHERE depart >= TIMESTAMP '2026-03-01'
      AND duree_s BETWEEN 60 AND 7200
),
enrichi AS (
    SELECT t.id_trajet, t.type_velo, t.duree_s, t.distance_m,
           sd.arrondissement AS arr_depart,
           t.distance_m / NULLIF(t.duree_s, 0) * 3.6 AS vitesse_kmh
    FROM trajets t
    JOIN stations sd ON sd.id_station = t.station_depart
    JOIN stations sa ON sa.id_station = t.station_arrivee
)
SELECT arr_depart,
       type_velo,
       count(*)                                                AS n_trajets,
       round(avg(duree_s) / 60.0, 1)                           AS duree_min_moy,
       round(avg(distance_m))                                  AS dist_moy_m,
       round(avg(vitesse_kmh), 1)                              AS vitesse_moy_kmh,
       round(100.0 * count(*)
             / sum(count(*)) OVER (PARTITION BY arr_depart), 1) AS pct_dans_arr
FROM enrichi
GROUP BY arr_depart, type_velo
HAVING count(*) > 1000
ORDER BY n_trajets DESC
LIMIT 50;
```

SYNTAXE

FROM d'abord (et SELECT facultatif)

On commence par dire OÙ l'on cherche — l'autocomplétion devient utile, et SELECT * est implicite.

```sql
-- SQL classique
SELECT *
FROM trajets
WHERE type_velo = 'électrique';

-- DuckDB : FROM d'abord
FROM trajets WHERE type_velo = 'électrique';

-- SELECT implicite : juste la table
FROM stations;
```

Pourquoi c'est utile

- L'éditeur connaît la table avant les colonnes → meilleure autocomplétion.
- Idéal pour explorer : « FROM trajets » et on voit tout de suite les données.
- Compatible avec le SQL standard : les deux écritures fonctionnent.

SYNTAXE

« SELECT négatif » : EXCLUDE et REPLACE

Sélectionner toutes les colonnes sauf certaines, ou en transformer une au passage — sans tout réécrire.

```sql
-- Toutes les colonnes SAUF deux
SELECT * EXCLUDE (id_trajet, id_velo)  FROM trajets;

-- Toutes les colonnes, mais on REMPLACE l'expression d'une seule
SELECT * REPLACE (duree_s / 60 AS duree_s)  FROM trajets;

-- Combinable avec COLUMNS(...) et les motifs
SELECT COLUMNS('.*_m') FROM trajets;   -- toutes les colonnes finissant par _m
```

Gain concret : plus besoin de lister 30 colonnes pour en retirer une. Le code reste lisible et robuste aux évolutions du schéma.

SYNTAXE

GROUP BY ALL et ORDER BY ALL

```sql
-- Sans répéter les colonnes non agrégées
SELECT station_depart, type_velo,
       count(*) AS n,
       avg(duree_s) AS duree_moy
FROM trajets
GROUP BY ALL          -- = station_depart, type_velo
ORDER BY ALL;         -- trie par toutes les colonnes
```

Moins d'erreurs

- GROUP BY ALL regroupe sur toutes les colonnes non agrégées : fini les listes à resynchroniser.
- On ajoute une dimension dans le SELECT → le GROUP BY suit automatiquement.
- ORDER BY ALL trie par l'ensemble des colonnes projetées.

SYNTAXE

Autres facilités du SQL DuckDB

Virgules finales

`SELECT a, b, c, FROM t` — la virgule traînante est tolérée.

COLUMNS() + lambda

`SELECT min(COLUMNS(*)) FROM t` — applique une fonction à toutes les colonnes.

Listes & structs

```
CREATE OR REPLACE TABLE deid AS
SELECT [1,2,3] AS l, {'x':1} AS s;  -- LIST et STRUCT / accès l[1]., s.x, UNNEST(l)
SELECT UNNEST(l), s.x from  deid;
```

Slicing de chaînes

`'Vélib'[1:3]`, `list_slice`, expressions de liste façon Python (caractère 1 à 3 de Vélib -> Vél) equivaut à `substring(chaîne, début, longueur)`.

QUALIFY

Filtrer directement sur une window function (voir partie 4).

PIVOT / UNPIVOT

Croiser/dé-croiser sans sous-requêtes verbeuses.

LIST & STRUCT — de la table au type complexe

LIST *agréger en gardant le détail des lignes*

TABLE ventes

client	produit
A	café
A	thé
B	sucre
A	café

→

REQUÊTE

```
SELECT client,
       list(produit) AS achats
FROM ventes
GROUP BY client;
```

→

RÉSULTAT

client	achats
A	['café','thé','café']
B	['sucre']

STRUCT *transporter un enregistrement entier dans une colonne*

TABLE ventes

ville	mont.	jour
Lyon	50	03-01
Lyon	80	03-02
Nice	30	03-01

→

REQUÊTE

```
SELECT ville,
  max_by({'m':mont,
          'j':jour}, mont)
  AS top
FROM ventes GROUP BY ville;
```

→

RÉSULTAT

ville	top
Lyon	{m:80, j:03-02}
Nice	{m:30, j:03-01}

accès : top.m → 80 · top.j → 03-02

LIST de STRUCT & MAP — cas composés

LIST de STRUCT *une relation un-à-plusieurs, sans table annexe*

TABLE lignes

cmd	produit	qté
1	café	2
1	thé	1
2	sucre	5

→

REQUÊTE

```
SELECT cmd,
  list({'prod':produit,
        'qte':qte}) AS lignes
FROM lignes GROUP BY cmd;
```

→

RÉSULTAT

cmd	lignes
1	[{café,2},{thé,1}]
2	[{sucre,5}]

MAP *des clés dynamiques, calculées à l'exécution*

TABLE ventes

ville
Lyon
Paris
Lyon
Nice

→

REQUÊTE

```
SELECT
  histogram(ville) AS compte
FROM ventes;
```

→

RÉSULTAT

compte
{Lyon:2, Paris:1, Nice:1}

les clés (villes) ne sont pas connues d'avance

Window functions

Comment calculer des rangs, somme cumulée, moyenne mobile, LAG...

C\SD WEBINAIRE

window function

```
fonction(...) OVER (PARTITION BY <groupes> ORDER BY <tri> <cadre>)
```

PARTITION BY

découpe en groupes (ex. par station)

ORDER BY

ordonne dans chaque groupe (ex. par date)

cadre (frame)

fenêtre de lignes : ROWS BETWEEN …

Table ventes (triée par jour)

jour	montant	
01/01	120	
02/01	90	
03/01	150	**cadre = 3 lignes**
04/01	200	
05/01	**170**	*ligne courante*
06/01	110	
07/01	140	

- La fonction calcule sur la fenêtre sans réduire le nombre de lignes (contrairement à GROUP BY).
- Cadre : ROWS BETWEEN 2 PRECEDING AND CURRENT ROW, UNBOUNDED PRECEDING, etc.
- Sans ORDER BY, le cadre par défaut est ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING → toute la partition
- Avec ORDER BY, le cadre par défaut est RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW → du début jusqu'à la ligne courante (le cumul).

Le cadre (frame) : une fenêtre qui glisse

Table ventes (triée par jour)

jour	montant
01/01	120
02/01	90
03/01	150
04/01	200
05/01	**170**
06/01	110
07/01	140

cadre = 3 lignes

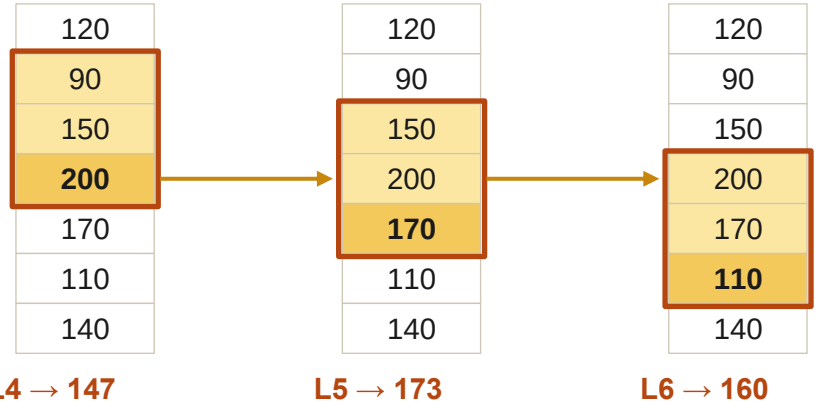
ligne courante

```
ROWS BETWEEN 2 PRECEDING AND CURRENT ROW
```

Fenêtre de 05/01 : **150 · 200 · 170**

moyenne glissante = 520 / 3 = 173

Le cadre glisse, ligne par ligne (moyenne glissante)



Le principe : OVER ne réduit pas les lignes

Contrairement à GROUP BY, une fonction fenêtrée garde toutes les lignes et ajoute une colonne calculée sur un groupe (la « fenêtre »), ici la moyenne des montants par ville.

```
SELECT ville, montant,
       AVG(montant) OVER (PARTITION BY ville) AS moy_ville
FROM Tventes4B
WHERE id < 8;
```

id	ville	montant	moy_ville
0	Lyon	5213	11920.50
1	Paris	1394	6527.50
2	Nice	17522	18221.00
3	Lille	11567	9261.50
4	Lyon	18628	11920.50
5	Paris	11661	6527.50
6	Nice	18920	18221.00
7	Lille	6956	9261.50

```
-- moyenne générale répétée sur chaque ligne
SELECT id, ville, montant,
AVG(montant) OVER () AS moyenne_generale
FROM ventes WHERE id < 8;
```

id	ville	montant	moyenne_generale
0	Lyon	5213	11482.62
1	Paris	1394	11482.62
2	Nice	17522	11482.62
3	Lille	11567	11482.62
4	Lyon	18628	11482.62
5	Paris	11661	11482.62
6	Nice	18920	11482.62
7	Lille	6956	11482.62

C:\Temp\DATA\WINDOW_FUNCTION_ventes4B.sql

Agrégats fenêtrés : la part de chaque ligne

PARTITION BY définit le groupe. On compare chaque vente au total de sa ville pour calculer sa contribution en %.

```
SELECT id, ville, montant,
       SUM(montant) OVER (PARTITION BY ville) AS total_ville,
       round(100.0 * montant
             / SUM(montant) OVER (PARTITION BY ville), 1) AS pct
FROM Tventes4B WHERE id < 8;
```

id	ville	montant	total_ville	pct
0	Lyon	5213	23841	21.90
1	Paris	1394	13055	10.70
2	Nice	17522	36442	48.10
3	Lille	11567	18523	62.40
4	Lyon	18628	23841	78.10
5	Paris	11661	13055	89.30
6	Nice	18920	36442	51.90
7	Lille	6956	18523	37.60

ROW_NUMBER : numéroter, faire du Top-N

Numérote les lignes de 1 à N dans chaque partition, selon l'ordre choisi : pour « la plus grosse vente de chaque ville ».

```
SELECT id, ville, montant,
      ROW_NUMBER() OVER (PARTITION BY ville
                  ORDER BY montant DESC) AS rang
FROM Tventes4B WHERE id < 12 AND ville IN ('Lyon','Paris')
ORDER BY ville, rang;
```

id	ville	montant	rang
1	Paris	1394	3
9	Paris	7191	2
5	Paris	11661	1
0	Lyon	5213	3
8	Lyon	16592	2
4	Lyon	18628	1

RANK vs DENSE_RANK (et ROW_NUMBER)

Sur les ex-aequo (ici, mêmes tranches de 5000 €), RANK laisse des trous, DENSE_RANK non, ROW_NUMBER reste toujours unique.

```
SELECT ville, montant,
       ROW_NUMBER() OVER w AS row_num,
       RANK() OVER w AS rank, DENSE_RANK() OVER w AS dense
FROM Tventes4B WHERE id < 6
WINDOW w AS (ORDER BY floor(montant/5000) DESC)
ORDER BY rank;
```

id	ville	montant	row_num	rank	dense
1	Paris	1394	6	6	4
0	Lyon	5213	5	5	3
3	Lille	11567	3	3	2
5	Paris	11661	4	3	2
2	Nice	17522	1	1	1
4	Lyon	18628	2	1	1

LAG / LEAD : comparer à la ligne voisine

LAG lit la valeur de la ligne précédente dans la partition — parfait pour une variation. La 1re ligne de chaque ville n'a pas de précédent (NULL).

```
SELECT id, jour, ville, montant,
       LAG(montant) OVER w AS prec,
       montant - LAG(montant) OVER w AS variation
FROM Tventes4B WHERE id < 12 AND ville IN ('Lyon','Nice')
WINDOW w AS (PARTITION BY ville ORDER BY jour)    --WINDOW est un raccourci d'écriture, après where
ORDER BY ville, jour;
```

id	jour	ville	montant	prec	variation
10	2024-01-11	Nice	12927		
6	2024-01-07	Nice	18920	12927	5993
2	2024-01-03	Nice	17522	18920	-1398
8	2024-01-09	Lyon	16592		
4	2024-01-05	Lyon	18628	16592	2036
0	2024-01-01	Lyon	5213	18628	-13415

Cumul : SUM avec ORDER BY

Ajouter ORDER BY à un agrégat le transforme en total cumulé : chaque ligne somme tout depuis le début de sa partition par ville jusqu'à elle.

```
SELECT jour, ville, montant,
       SUM(montant) OVER (PARTITION BY ville ORDER BY jour) AS cumul
FROM Tventes4B WHERE id < 12 AND ville IN ('Lyon','Nice')
ORDER BY ville, jour;
```

id	jour	ville	montant	cumul
10	2024-01-11	Nice	12927	12927
6	2024-01-07	Nice	18920	31847
2	2024-01-03	Nice	17522	49369
8	2024-01-09	Lyon	16592	16592
4	2024-01-05	Lyon	18628	35220
0	2024-01-01	Lyon	5213	40433

Moyenne glissante : le cadre ROWS BETWEEN

Le cadre (frame) borne la fenêtre à quelques lignes autour de la ligne courante. Ici, moyenne sur 2 points : la ligne et sa précédente.

```
SELECT id, jour, ville, montant,
      round(AVG(montant) OVER (PARTITION BY ville ORDER BY jour
            ROWS BETWEEN 1 PRECEDING AND CURRENT ROW)) AS moy_mobile
FROM Tventes4B WHERE id < 12 AND ville IN ('Lyon','Nice')
ORDER BY ville, jour;
```

id	jour	ville	montant	moy_mobile
10	2024-01-11	Nice	12927	12927.00
6	2024-01-07	Nice	18920	15924.00
2	2024-01-03	Nice	17522	18221.00
8	2024-01-09	Lyon	16592	16592.00
4	2024-01-05	Lyon	18628	17610.00
0	2024-01-01	Lyon	5213	11921.00

QUALIFY : filtrer sur une window function

Garder le Top-N par groupe directement, sans passer par une sous-requête.

```
SELECT ville, montant
FROM Tventes4B WHERE id < 12
QUALIFY ROW_NUMBER() OVER (
          PARTITION BY ville ORDER BY montant DESC) <= 2
ORDER BY ville, montant DESC;
```

id	ville	montant
5	Paris	11661
9	Paris	7191
6	Nice	18920
2	Nice	17522
4	Lyon	18628
8	Lyon	16592
11	Lille	17111
3	Lille	11567

La table Ny Taxi pour illustrer les window function

```
create table TableNycTaxi as (select * from 'nyc_taxi170M.parquet');
```

```
Base_nyctaxi170M D create table TableNycTaxi as (select * from 'nyc_taxi170M.parquet');
 51% ?                    ?                ?(~49 seconds remaining)                Memory: 5.9 GB
```

Nom	Modifié le	Type	Taille	
Base_nyctaxi170M	21/06/2026 18:23	Fichier	6 751 244 Ko	30%
nyc_taxi170M.parquet	27/01/2026 15:01	Fichier PARQUET	4 957 378 Ko	23%
NycTaxi170M.csv	21/06/2026 18:04	Fichier CSV Microsoft Excel	21 825 585 Ko	100%

La table Ny Taxi pour illustrer les window function

Calcul de la contribution de chaque course par vendor_id :
Pour chaque vendor id, cela calcul la somme des montants de toutes les courses
Pour chaque course, on fait le rapport du montant de la course avec cette somme

```
.timer on
SELECT  vendor_id,
  rowid AS num_course,
  10000000 * Total_amount / sum(Total_amount) OVER (PARTITION BY vendor_id) AS Part_Amount
 FROM TableNycTaxi LIMIT 10; ;
100% - Run Time (s): 170.90 million rows  3 columns. Run Time (s): real 2.913 user 10.265625 sys 13.828125
```

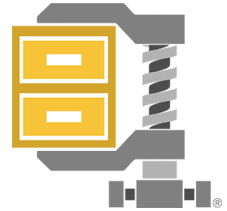
num_course	vendor_id	Part_Amount
122880	CMT	0.13
122881	CMT	0.04
122882	CMT	0.13
122883	CMT	0.32

Gestion Disque et RAM

Comment calculer des rangs, somme cumulée, moyenne mobile, LAG...

C\SD WEBINAIRE

Encodage & compression légère



Chaque colonne est compressée selon ses données, avec des schémas qui se décodent à la volée (peu de CPU).

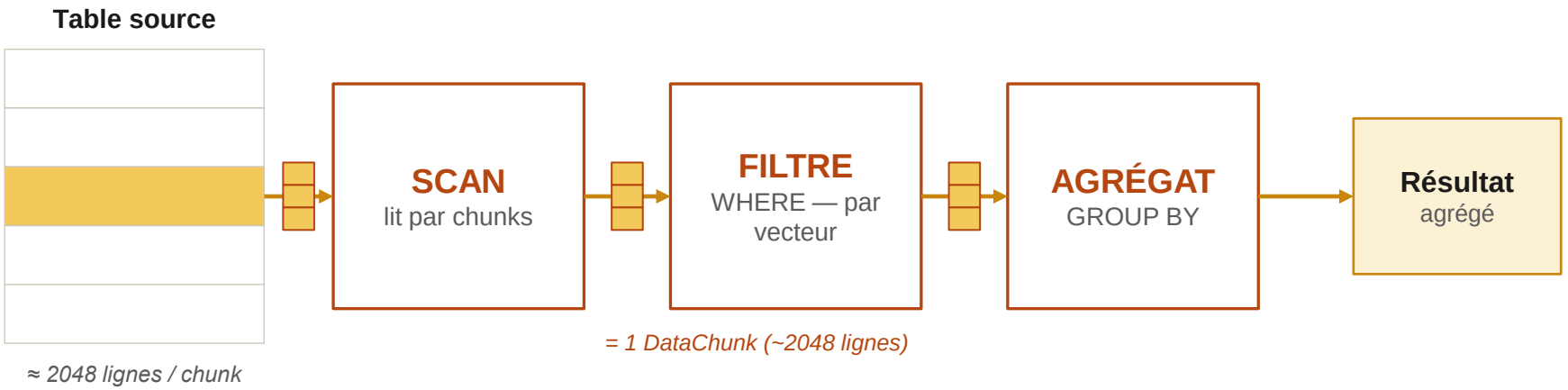
Encodage	Quand	Exemple
RLE (Run-Length Encoding)	Répétitions	Lyon, Lyon, Lyon, Lyon, Lyon, Paris, Paris, Paris **devient** (Lyon, ×5), (Paris, ×3)
Dictionnaire	Peu de valeurs distinctes	noms de villes, stations, arrondissements
Bitpacking	Entiers proches	une colonne age avec des valeurs de 0 à 120. En INTEGER, c'est 32 bits par ligne. En bit-packing sur 7 bits : divisez la taille par presque 5
FSST (Fast Static Symbol Table)	Chaînes courtes	https://, .com, www. ou @ipre.fr reviennent sans cesse : FSST les encode chacun sur un seul octet
Delta (codage par differences)	Suites croissants/décroissantes	horodatages, identifiants

Conséquence : le fichier sur disque est compact, et surtout on lit moins d'octets → les requêtes sont plus liées au CPU, moins au disque.
A noter que **contrairement à Parquet, le fichier est modifiable au champ (attention, le format DuckDB change souvent).**
DuckDB ne compresse que sur une base sur disque (jamais en mémoire), et seulement après écriture — donc il faut un fichier + un CHECKPOINT (COMMIT).

Nom	Modifié le	Type	Taille	
Table_ventes4B.csv	21/06/2026 15:18	Fichier CSV Microsoft Excel	127 604 362 Ko	100%
Table_ventes4B.parquet	21/06/2026 15:35	Fichier PARQUET	27 661 282 Ko	21%
ventes4B.duckdb	21/06/2026 15:04	Fichier DUCKDB	16 686 092 Ko	13%

Nom	Modifié le	Type	Taille	
Base_nyctaxi170M	21/06/2026 18:23	Fichier	6 751 244 Ko	30%
nyc_taxi170M.parquet	27/01/2026 15:01	Fichier PARQUET	4 957 378 Ko	23%
NycTaxi170M.csv	21/06/2026 18:04	Fichier CSV Microsoft Excel	21 825 585 Ko	100%

L'exécution vectorisée : le streaming des chunks



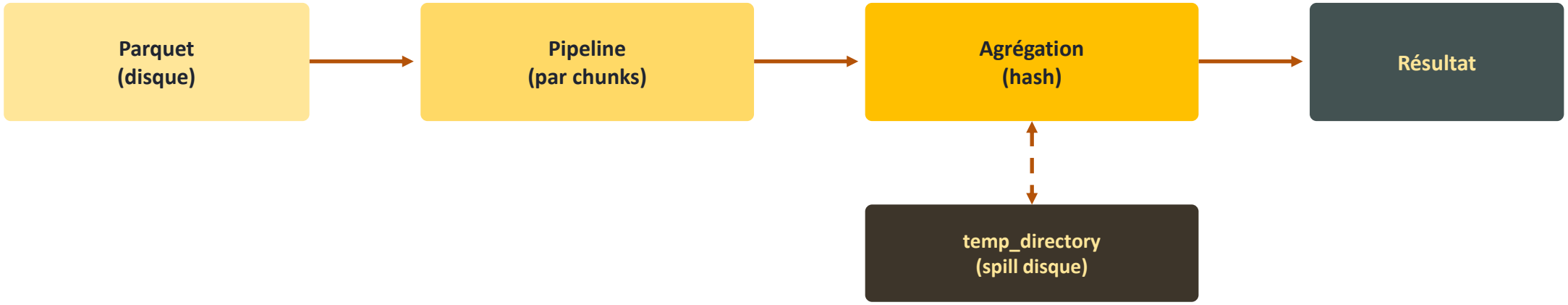
STREAMING
un chunk traverse tout le pipeline avant que le suivant n'entre

MÉMOIRE BORNÉE
la table entière n'est jamais matérialisée en RAM

VECTORISÉ · SIMD
chaque opérateur traite le vecteur (~2048) d'un coup

Streaming : plus grand que la mémoire

Les données circulent par lots dans le pipeline ; si un opérateur déborde, il déverse temporairement sur disque.



- Beaucoup d'opérations (scan, filtre, agrégation, tri, jointure) fonctionnent en flux, sans tout charger.
- Si la RAM ne suffit pas, DuckDB déverse les données intermédiaires sur disque (out-of-core) et continue.
- Possible donc de traiter des jeux plus grands que la mémoire.

Garder la main sur la mémoire ! Surtout en serveur partagé

```
-- Plafonner la RAM utilisée par DuckDB
-- sinon prend 80% de la mémoire physique de la machine
SET memory_limit = '4GB';

-- Nombre de cœurs (attention en serveur partagé)
SET threads = 8;

-- Où déverser si ça déborde
SET temp_directory = '/tmp/duckdb_spill';

-- Diagnostic
PRAGMA database_size;
PRAGMA memory_limit;
```

Bonnes pratiques

- !!! **memory_limit borne la consommation** : utile en conteneur ou sur un serveur partagé.
- threads ajuste le parallélisme (par défaut : tous les cœurs).
- temp_directory choisit un disque rapide pour les déversements.
- EXPLAIN / EXPLAIN ANALYZE pour comprendre où va le temps et la mémoire.

Lecture directe & pushdown

DuckDB lit les fichiers sans import et ne charge que ce qui est nécessaire (projection et filtre « poussés » vers le fichier).

```sql
SELECT station_depart, count(*)
FROM 'trajets/*.parquet'
WHERE depart >= '2026-01-01'
GROUP BY ALL;

-- Lecture aussi possible : read_csv, read_json
-- glob de fichiers : 'data/**/*.parquet'
```

Ce que DuckDB lit vraiment

Projection : seulement station_depart et depart, pas les 9 colonnes.

Filtre : les groupes de lignes (row groups) hors période sont sautés via les statistiques min/max.

DuckDB et Dataframe R

C\SD WEBINAIRE

DuckDB vs dataframe R

DuckDB

Dimension	Dataframe R	DuckDB
Mémoire	Tout en RAM, copies fréquentes	Colonnaire, streaming, > RAM
Parallélisme	Souvent mono-thread	Multi-cœur par défaut
Modèle	Impératif : vous orchestrez	Déclaratif : un optimiseur planifie
Données > RAM	Erreur d'allocation	Déversement sur disque
Persistance / ACID	Non (objets en session)	Oui (fichier .duckdb)
Sources externes	Import explicite préalable	Lecture directe (fichiers, S3)

Atouts nets — et nuances côté R

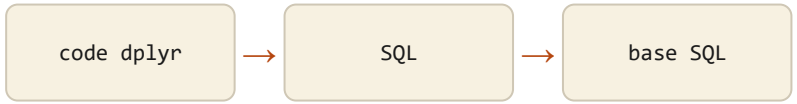
Atout de DuckDB	Nuance / force de R
Tient des volumes qui font exploser un dataframe	data.table : très rapide sur les charges en RAM
Parallélise sans effort	dplyr : expressivité + écosystème (ggplot, tidymodels)
Optimiseur (ordre des jointures, filtres poussés)	SQL n'a pas les modèles statistiques natifs
En pratique, ce n'est pas un choix binaire : avec le backend duckdb / dbplyr, on écrit du dplyr exécuté DANS DuckDB — syntaxe R, moteur DuckDB.	

duckplyr vs dbplyr — deux ponts vers DuckDB

Relier dplyr à DuckDB de deux façons : par traduction SQL, ou par l'API relationnelle.

dbplyr

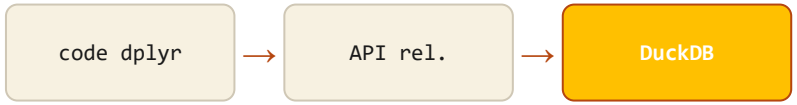
dplyr → SQL → base



- Traduit dplyr en SQL (visible via show_query)
- Toute base SQL : Postgres, SQLite, DuckDB…
- Sémantique proche de dplyr, mais pas identique
- Idéal si les données sont déjà en base

duckplyr

dplyr → API relationnelle → DuckDB



- Pas de SQL : API relationnelle de DuckDB
- Lié à DuckDB ; entrées/sorties = tibbles R
- Sémantique 100 % identique à dplyr (garantie)
- Repli auto sur dplyr + gère les données > RAM

En clair dbplyr = passerelle vers SQL · duckplyr = dplyr « turbo », mêmes résultats. Ils s'interfacent : `as_tbl()` ↔ `as_duckdb_tibble()`.

duckplyr vs dbplyr — deux ponts vers DuckDB

```
44  # creation d'une vue sur un fichier parquet
45  dbExecute(con, "CREATE VIEW vueNetflix AS SELECT * FROM read_parquet('D:/SHARE/COMMON/PRESENTATIONS/PRES20231130_PARQUET-DUCKDB_KG/datafiles/netflixTize.parquet');")
46  tbl(con, "vueNetflix") |>
47    group_by(Rank)  |>
48    summarise(bonscore = mean(Viewership_Score, na.rm = TRUE) ) |>
49    collect()
50
51
```

44:46 (Top Level) R Script

Console Terminal Background Jobs

R 4.2.2 · ~/

```
> # creation d'une vue sur un fichier parquet
> dbExecute(con, "CREATE VIEW vueNetflix AS SELECT * FROM read_parquet('D:/SHARE/COMMON/PRESENTATIONS/PRES20231130_PARQUET-DUCKDB_KG/datafiles/netflixTize.parquet');")
[1] 0
> tbl(con, "vueNetflix") |>
+   group_by(Rank)  |>
+   summarise(bonscore = mean(Viewership_Score, na.rm = TRUE) ) |>
+   collect()
# A tibble: 10 x 2
    Rank bonscore
   <dbl>    <dbl>
 1     4    103.
 2    10    128.
 3     2     92.2
 4     3     93.7
 5     9    148.
 6     1     90.8
 7     6    148.
 8     5    122.
 9     7    141.
10     8    162.
```

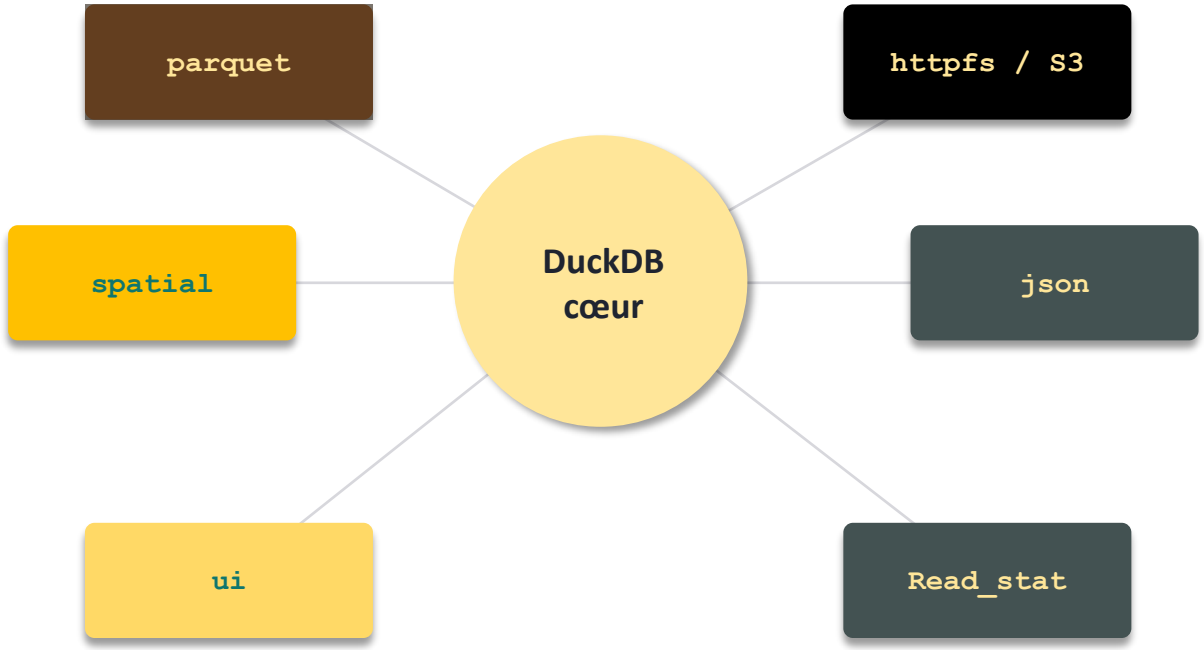
Extension et écosystème

C\SD WEBINAIRE

Un système d'extensions modulaire

```
INSTALL spatial;
LOAD spatial;

-- souvent auto-chargé :
FROM s3://bucket/
     trajets/*.parquet;
```



- INSTALL télécharge, LOAD active ; beaucoup d'extensions sont auto-chargées au besoin.
- ~27 extensions « cœur » officielles + 150+ communautaires (httpfs, spatial, json, iceberg, delta, postgres, mysql, h3…).

Parquet (et CSV, JSON)

```sql
-- Lire un dossier de fichiers Parquet
FROM 'trajets/year=2026/*.parquet';

-- Écrire, partitionné par mois
COPY (FROM trajets) TO 'export/'
  (FORMAT parquet, PARTITION_BY (mois));

-- Convertir un gros CSV en Parquet compressé
COPY (FROM read_csv('trajets.csv')) TO 'trajets.parquet' (FORMAT parquet);
```

Parquet = format colonnaire compressé. DuckDB lit/écrit nativement, exploite les statistiques (pushdown) et gère le partitionnement Hive.

Données distantes : httpfs & S3

```
INSTALL httpfs; LOAD httpfs;

SET s3_region = 'eu-west-3';
SET s3_access_key_id = '...';
SET s3_secret_access_key = '...';

-- Requête directe sur l'objet S3, sans téléchargement
SELECT type_velo, count(*)
FROM 's3://velib-data/trajets/*.parquet'
GROUP BY ALL;
```

```
SELECT * FROM
's3://us-prd-motherduck-open-datasets/netflix/netflix_daily_top_10.parquet'
limit 100;
```

Lac de données local

- Interroge Parquet/CSV sur S3, GCS, Azure ou HTTP comme des tables.
- Grâce au pushdown, seuls les fragments utiles sont téléchargés.
- DuckDB devient un moteur de requête sur votre lac de données — sans cluster.

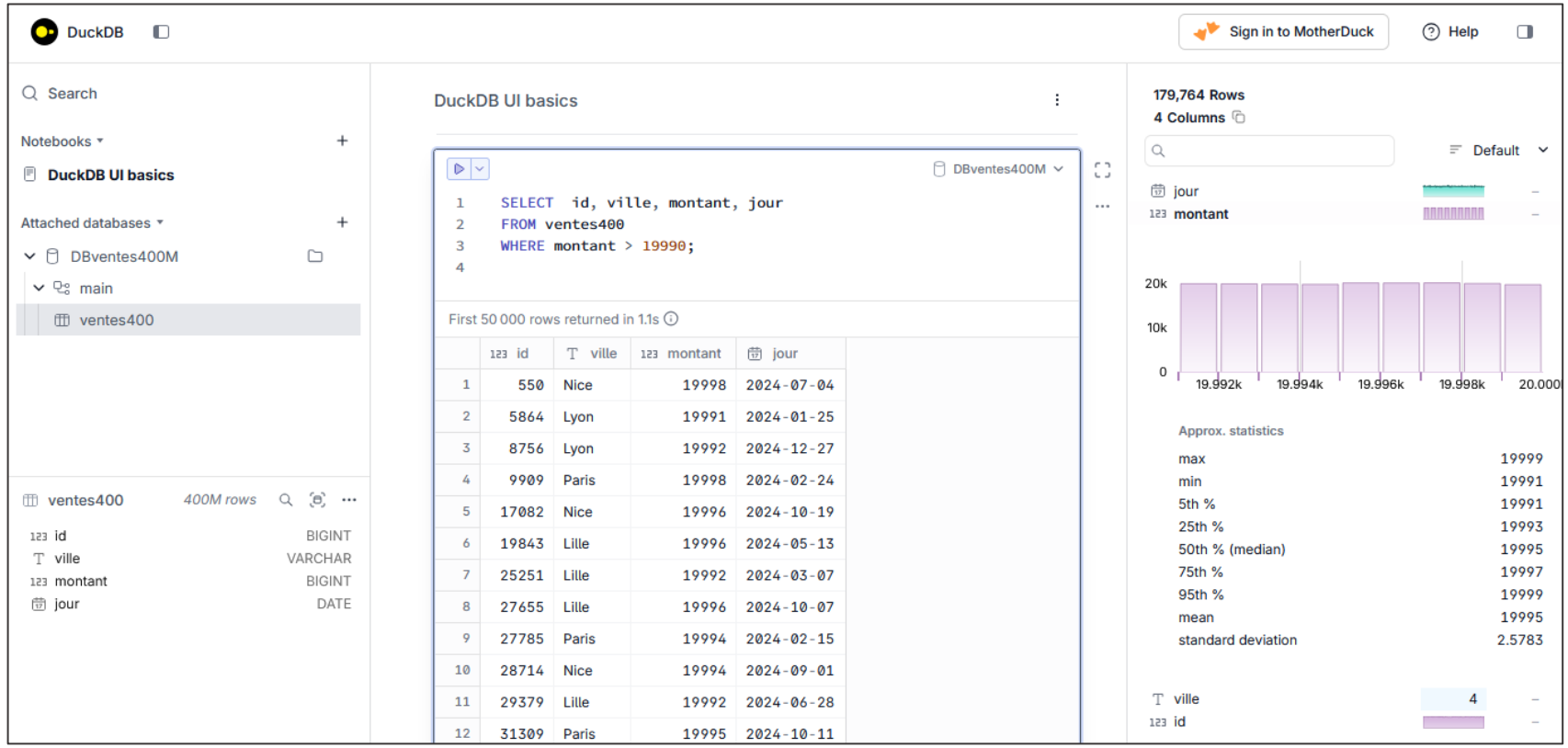
Spatial : géométries et jointures géographiques

```
INSTALL spatial; LOAD spatial;

-- Distance entre station de départ et d'arrivée (mètres)
SELECT t.id_trajet,
  ST_Distance(ST_Point(d.longitude, d.latitude)::GEOMETRY,
              ST_Point(a.longitude, a.latitude)::GEOMETRY) AS d_m
FROM trajets t
JOIN stations d ON d.id_station = t.station_depart
JOIN stations a ON a.id_station = t.station_arrivee;
```

Fonctions ST_*, lecture GeoParquet/Shapefile/GeoJSON, et jointures spatiales accélérées par index R-tree (≈ 58× plus rapides). Parfait pour relier trajets et géographie urbaine.

L'interface web locale (extension ui)



- Intégrée depuis la 1.2.1 : on lance avec duckdb -ui ou

  CALL start_ui();

- Notebooks SQL, exploration du schéma, graphiques — tout en local, vos données ne sortent pas.
- Et au-delà : MotherDuck (cloud), DuckLake (lakehouse), Iceberg/Delta, scanners Postgres/MySQL…

vss	loaded	core	Adds indexing support to accelerate Vector Similarity Search
vortex	available	core	Adds support for reading and writing files using the Vortex file format
ui	loaded	core	Adds local UI for DuckDB
tpch	loaded	core	Adds TPC-H data generation and query support
tpcds	loaded	core	Adds TPC-DS data generation and query support
sqlite_scanner	available	core	Adds support for reading and writing SQLite database files
spatial	installed	core	Geospatial extension that adds support for working with spatial data and functions
read_stat	installed	core	Read data sets from SAS, Stata, and SPSS with ReadStat
rdf	installed	core	A DuckDB extension to read and write RDF
quack	available	core	The DuckDB 'Quack' Client/Server Protocol
postgres_scanner	available	core	Adds support for connecting to a Postgres database
parquet	loaded	core	Adds support for reading and writing parquet files
odbc_scanner	available	core	Adds support for connecting to remote databases over ODBC
mysql_scanner	available	core	Adds support for connecting to a MySQL database
motherduck	available	core	Enables motherduck integration with the system
lance	available	core	Adds support for querying Lance datasets
json	loaded	core	Adds support for JSON operations
inet	loaded	core	Adds support for IP-related data types and functions
icu	loaded	core	Adds support for time zones and collations using the ICU library
iceberg	available	core	Adds support for Apache Iceberg
httpfs	loaded	core	Adds support for reading and writing files over a HTTP(S) connection
fts	loaded	core	Adds support for Full-Text Search Indexes
excel	loaded	core	Adds support for Excel-like format strings
encodings	loaded	core	All unicode encodings to UTF-8
ducklake	available	core	Adds support for DuckLake, SQL as a Lakehouse Format
duck_geoarrow	installed	core	DuckDB extension for converting GEOMETRY/WKB to and from GeoArrow native encodings, powered by geoarrow-c. Built against DuckDB v1.5.2
delta	available	core	Adds support for Delta Lake
core_functions	loaded	core	Core function library
azure	available	core	Adds a filesystem abstraction for Azure blob storage to DuckDB
aws	installed	core	Provides features that depend on the AWS SDK
autocomplete	loaded	core	Adds support for autocomplete in the shell
zipfs	available	community	Read files within zip archives
zim	available	community	Read .zim (Kiwix / openZIM) archives directly in DuckDB via libzim, from local files or remote S3/HTTP — offline Wikipedia, WikiMed, Stack Exchange, iFixit,
zeek	available	community	Read Zeek network security monitor log files with automatic schema detection and type-aware parsing
yardstick	available	community	Measure-aware SQL implementing Julian Hyde's 'Measures in SQL' paper
yaml	available	community	Read YAML files into DuckDB with native YAML type support, comprehensive extraction functions, and seamless JSON interoperability

Extensions

Readstat pour lire des fichiers SAS (retro-ingienerie donc pas full garanti mais efficace), STATA et SPSS;

fts full-texte search index ;

VSS vector similarity search (cosine est maintenant intégrée dans duckDB mais pas HSNW) ;

Core functions permet notamment d'avoir des lists ;

duckdb-raquet extension pour le format Raquet (raster stocké dans Parquet avec l'indexation spatiale QUADBIN). Elle s'appuie sur les types GEOMETRY natifs de DuckDB 1.5+, offre une API façon PostGIS — ST_RasterValue, ST_RasterSummaryStats, ST_RegionStats, ST_Clip mais n'est pas signee

Etc.

Les extensions community n'offrent pas les garanties de celles prises en charge par DuckLabs en matière de perennité (exemple encoding, cold).

DuckDB
and the rest of the world

C\SD WEBINAIRE

DuckDB face aux moteurs SQL — benchmarks 2025

Sur ClickBench (oct. 2025, exécutions « hot »), DuckDB est le moteur open-source **n°1** ; au coude-à-coude avec ClickHouse et Polars sur une seule machine.

Moteur	Type	Sur une seule machine (analytique)
DuckDB	Embarqué · vectorisé · OSS	#1 open-source en « hot runs » (ClickBench 10/2025)
Umbra	Prototype de recherche (non public)	Tête de la catégorie hot — référence académique
ClickHouse	Serveur colonne · OSS	Au coude-à-coude avec DuckDB
Polars	DataFrame · Rust · OSS	À égalité ; meilleur au tri/CSV, plus gourmand en RAM
PostgreSQL	Transactionnel (OLTP) · lignes	100 à 1000× plus lent sur l'OLAP
Spark	Distribué · cluster	Pertinent au-delà d'une seule machine

#1

moteur open-source sur ClickBench (exécutions « hot »)

100–1000×

plus rapide que PostgreSQL / SQLite sur l'OLAP

3,3 %

des développeurs interrogés utilisent DuckDB (Stack Overflow 2025) — contre 1,4 % en 2024

Sources : ClickBench (single-node, hot runs, oct. 2025) ; Stack Overflow Developer Survey 2025. Le « 3,3 % » est un taux d'usage déclaré (un répondant peut citer plusieurs bases), pas une part de marché. Classements variables selon le workload, le matériel et la version ; Umbra est un prototype de recherche non public.

L'écosystème autour de DuckDB

Un moteur compact (sous licence MIT) entouré de clients, formats, extensions, cloud et outils du data stack.

CLIENTS & LANGAGES

Python · R · Java · Node.js
Go · Rust · **C / C++** · ODBC
CLI · **WASM (dans le navigateur)**

FORMATS & STOCKAGE

Parquet · CSV · JSON · Arrow
Excel · Avro · géospatial
Iceberg · Delta · DuckLake

EXTENSIONS

httpfs (S3) · spatial · json
vss (vecteurs) · fts (texte)
scanners Postgres / MySQL / SQLite
150+ extensions communautaires

CLOUD & LAKEHOUSE

MotherDuck — DuckDB serverless en cloud (requêtes hybrides)
DuckLake — lakehouse en SQL : ACID, time-travel

PIPELINES & DATA STACK

dbt · dlt · Meltano · Fivetran
Dagster · Airflow
Ibis · pandas · Polars · Arrow

BI & IA

Evidence · Rill · Metabase
Superset · Lightdash · Hex
LangChain · **serveur MCP · RAG**

Cœur DuckDB et extensions sous licence MIT · plus de 17 M de téléchargements d'extensions par mois. Aperçu 2026, non exhaustif.

DuckDB or not DuckDB

C\SD WEBINAIRE

Quand NE PAS utiliser DuckDB

Situation	Pourquoi	Préférer
Charges transactionnelles (OLTP)	Beaucoup d'écritures / MAJ ligne à ligne	SQLite, PostgreSQL, MySQL
Multi-utilisateurs en écriture	Modèle « un seul writer »	Base serveur (PostgreSQL, MySQL) ou DuckDB-Quack
Très petites données, geste simple	SQL ajoute une syntaxe inutile	Dataframe direct
ML avancés, visualisation	Hors du rôle d'un moteur SQL	R / Python
Au-delà d'une machine	Pas de calcul distribué tolérant aux pannes	Spark, clusters

Guide de choix

Quand utiliser DuckDB — et quand ne pas l'utiliser

DuckDB excelle quand…	Limites à connaître
Données locales ou S3 Lecture directe sans serveur ni ETL	**Multi-utilisateurs > 10** → StarRocks, ClickHouse
≤ quelques TB, single-node Spilling automatique jusqu'au TB	**Cluster distribué natif** → Apache Spark, Trino
Requêtes analytiques Agrégations, jointures complexes, window functions	**OLTP (writes fréquents)** → SQLite, PostgreSQL
Intégration Python / R / Julia API natives, zero-copy Arrow/Pandas/Polars	**Peta-byte scale distribué** → Apache Spark
1–2 utilisateurs simultanés Analytique interactive, notebooks	**High Availability / réplication** → ClickHouse
Zéro infra, zéro config Un seul fichier binaire, aucun daemon	**Log analytics temps réel** → ClickHouse

Conclusion

C\SD WEBINAIRE

DuckDB — à retenir

CE QUE C'EST

Moteur SQL analytique in-process — « le SQLite de l'analytique ». Pas de dépendance, zéro administration. Lit Parquet, CSV, S3 sans étape d'import.

SA FORCE

Vectorisé, encodings, multi-cœur, déversement sur disque. Parmi les plus rapides en single-node. Des charges « cluster » peuvent tenir désormais sur un simple laptop ou serveur.

SES LIMITES

Pas un transactionnel · un seul writer à la fois · non distribué. Forte concurrence en écriture ou volumes > un autre SGBD.

QUAND LE CHOISIR

Analytique, exploration, prototypage, tests locaux, embarqué dans une appli. Atout en environnement confiné / bulle sécurisée : pas d'infra complexe à sécuriser.

DuckDB ne remplace pas les clusters— il est complémentaire pour une grande partie des cas.

Conclusion

- **DuckDB central pour les data**

  Comble efficacement le secteur du milieu (vous vous rappelez ?)

  Potentiels importants en termes de performances si optimisé (RAM, disque, stockage)

  DuckDB a déplacé la frontière entre « ça tient sur mon serveur » et « il me faut un cluster »

  Ces propriétés — une seule dépendance, pas de serveur à administrer, exécution entièrement locale — en font un candidat particulièrement adapté aux bulles sécurisées du CASD par exemple

- **DuckDB en évolution rapide**

  La version de cette semaine 1.5.4 n'est pas la même que celle de la semaine dernière, 1.5.3

  Beaucoup de développements autour de DuckDB : Quack, MCP…

MERCI DE VOTRE ATTENTION

C\SD WEBINAIRE