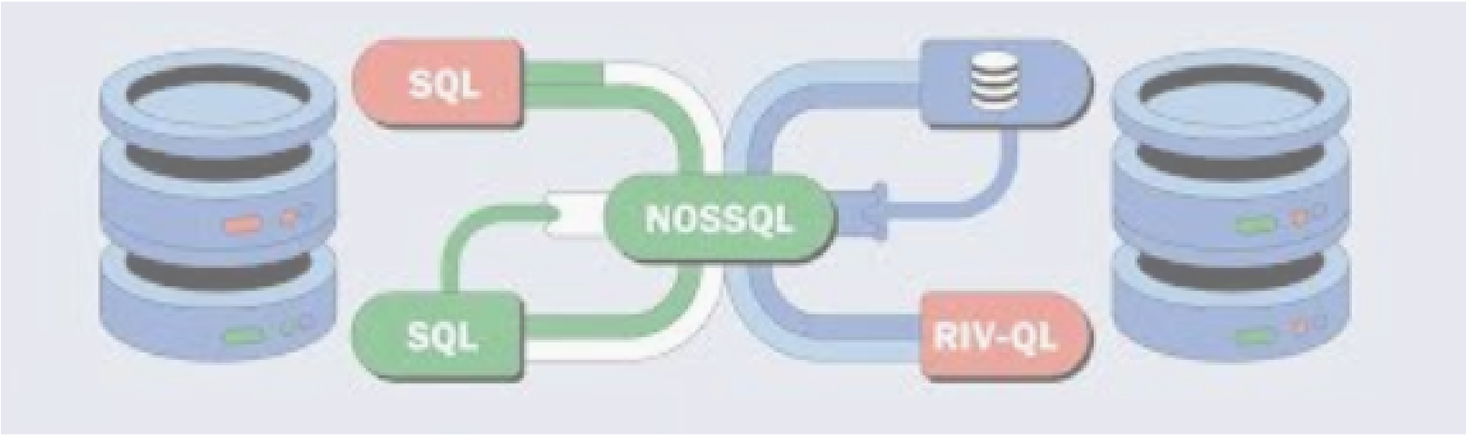
NEXA Digital School

20 JANVIER 2026

OPTIMISATION AVANCÉE SQL & NOSQL

MASTÈRE DATA & IA

CHEF DE PROJET



FORMATEUR : DIALLO ALIMOU

Objectifs Pédagogiques

- Mettre en œuvre un partitionnement de tables pour améliorer la gestion de gros volumes de données.
- Utiliser des agrégations, regroupements (GROUP BY), des CTE (Common Table Expressions) et des sous-requêtes pour optimiser les requêtes complexes.
- Réduire les lectures inutiles par des techniques de projection, de pagination, et de sélection ciblée de colonnes.
- Implémenter des vues matérialisées pour pré-calculer des résultats et améliorer les temps de réponse.
- Comparer les performances à l’aide de tests avant/après, en utilisant EXPLAIN (ANALYZE, BUFFERS) pour mesurer l’impact des optimisations.

Pourquoi les index ne suffisent plus

Quand les tables deviennent énormes, même indexées, certaines requêtes restent lentes

Problèmes :

- Index trop gros
- Trop de données scannées
- Maintenance lente

Solutions :

- Partitionnement
- Pré-calcul
- Réduction des lectures

Stratégies d’optimisation modernes

- Découper les données (partitionnement)
- Lire moins de colonnes (projection)
- Lire moins de lignes (filtres, pagination)
- Pré-calculer (vues matérialisées)
- Indexer intelligemment
- Changer de modèle (NoSQL)

Pourquoi partitionner ?

Le **partitionnement** comporte de nombreux avantages : les performances des requêtes peuvent être significativement améliorées dans certaines situations, particulièrement lorsque la plupart des lignes fortement accédées d'une table se trouvent sur une seule partition ou sur un petit nombre de partitions.

Même avec index :

- Une table de 100M lignes reste :
  - lourde à scanner
  - lourde à maintenir
  - lourde à indexer

Partitionner = diviser pour mieux agir

EXEMPLE

Imaginons que nous soyons en train de construire une base de données pour une grande société de crème glacée. La société mesure les pics de températures chaque jour, ainsi que les ventes de crème glacée dans chaque région.
Conceptuellement, nous voulons une table comme ceci :

```
CREATE TABLE mesure (
id_ville int not null,
date_trace date not null,
temperature int,
ventes int
);
```

Problème : après quelques années → table énorme

Après 10 ans :

~3650 jours

~1000 villes

≈ 3,6 millions de lignes

Conséquences :

Index énormes

Requêtes par période lentes

Archivage compliqué

CRÉATION DES PARTITIONS MENSUELLES

```
CREATE TABLE measurement_y2006m02 PARTITION OF measurement
FOR VALUES FROM ('2006-02-01') TO ('2006-03-01');
CREATE TABLE measurement_y2006m03 PARTITION OF measurement
FOR VALUES FROM ('2006-03-01') TO ('2006-04-01');
```

Plusieurs partitions = plusieurs fichiers

Chaque mois =

- une table physique
- ses propres index
- éventuellement son propre disque

Le partitionnement est une optimisation STRUCTURELLE basée sur le métier, pas un gadget technique.

MAINTENANCE DES PARTITIONS

Réalité en production

Une table partitionnée n'est jamais figée :

- On ajoute des données tous les jours
- On archive les anciennes
- On supprime des périodes entières

Automatisation indispensable

En production :

- Script SQL
- Script Python
- Job cron
- Migration automatique

Pour :

- Créer les partitions à l'avance
- Supprimer / détacher les anciennes

OPTIMISATION DES CALCULS ANALYTIQUES
GROUP BY, AGRÉGATIONS, VUES MATÉRIALISÉES

Problème général du reporting

En production

Ce ne sont pas les requêtes OLTP qui tuent les bases…

Ce sont les requêtes de reporting.

- Lentes
- Lourdes
- Lisent énormément de données
- Peuvent bloquer la prod

Le reporting, ce sont des requêtes SQL qui servent à :

- faire des statistiques
- faire des tableaux de bord
- analyser l’activité
- aider à la décision

OPTIMISATION DES CALCULS ANALYTIQUES
GROUP BY, AGRÉGATIONS, VUES MATÉRIALISÉES

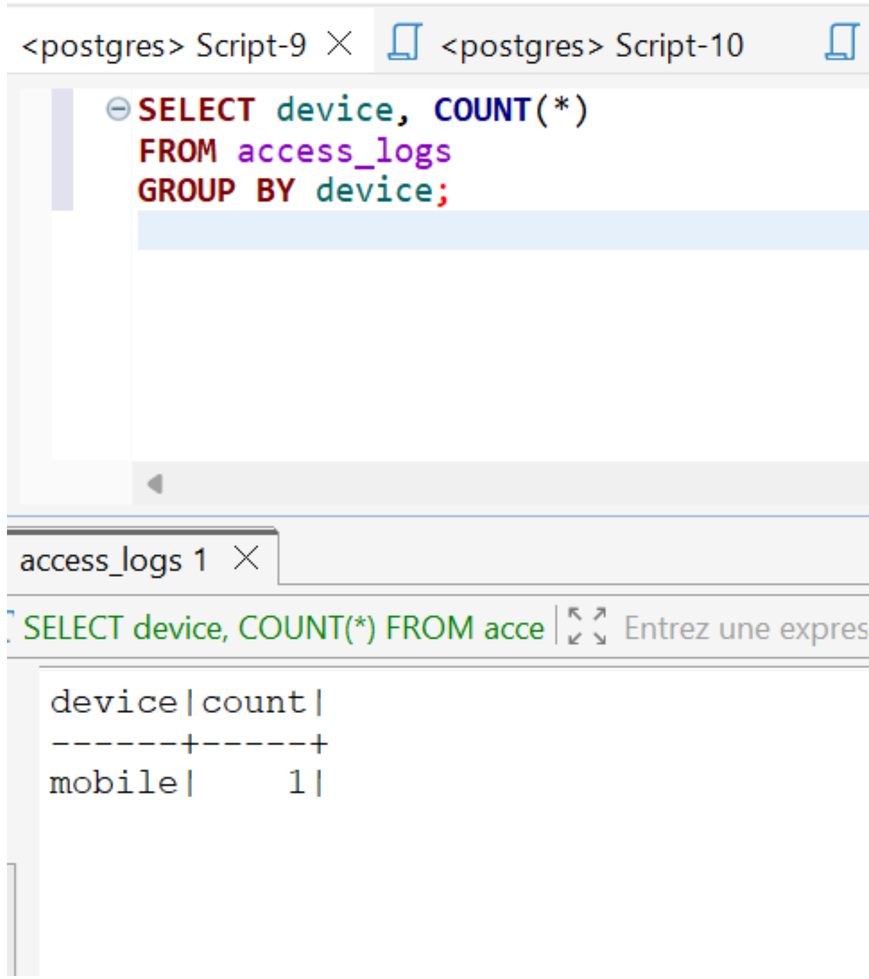
Ce que PostgreSQL doit faire

Pour répondre :

1. Lire toutes les lignes
2. Regrouper par device
3. Compter chaque groupe

Même si :

Il n'y a que 3 lignes en sortie



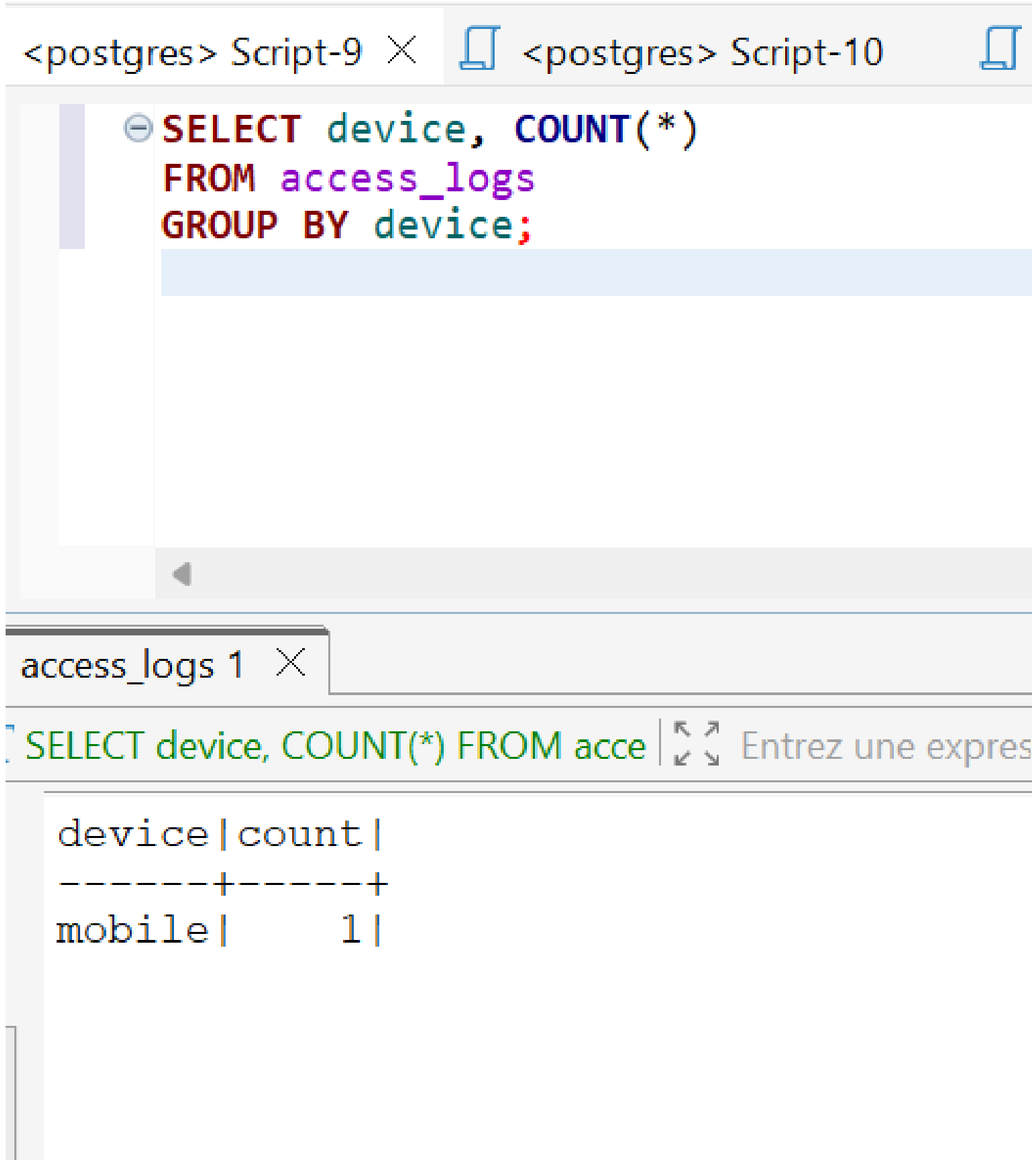
Combien de connexions par type d'appareil ?"

OPTIMISATION DES CALCULS ANALYTIQUES
GROUP BY, AGRÉGATIONS, VUES MATÉRIALISÉES

Le coût dépend du volume d'entrée, pas du volume de sortie.

Exemple :

- 5 000 000 lignes lues
- 3 lignes retournées



STRATÉGIES INTERNES POSTGRESQL

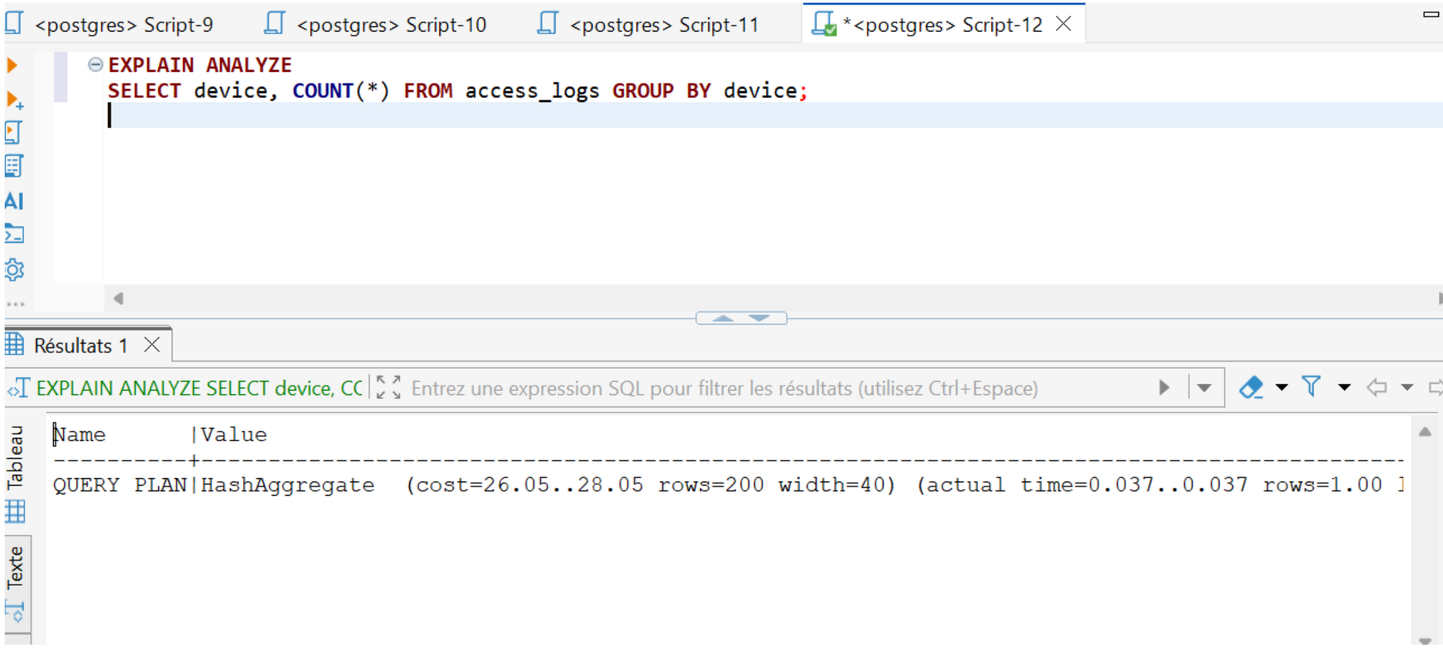
PostgreSQL peut utiliser :

HashAggregate

- Construit une table de hachage en mémoire
- Rapide si ça tient en RAM
- Peut déborder sur disque

GroupAggregate + Sort

- Trie toutes les lignes
- Puis regroupe
- Très coûteux en I/O



PREMIÈRE OPTIMISATION : FILTRER AVANT D'AGRÉGER

On agrège moins de lignes – beaucoup plus rapide

```sql
SELECT device, COUNT(*)
FROM access_logs
WHERE accessed_at >= '2025-01-01'
GROUP BY device;
```

CTE : DÉCOUPER LOGIQUEMENT LE PROBLÈME

CTE (Common Table Expression)

- Une table temporaire définie dans la requête
- **Sert à :**
  - Découper une requête complexe
  - Rendre le SQL plus lisible
  - Structurer un raisonnement

```sql
WITH recent_logs AS (
  SELECT *
  FROM access_logs
  WHERE accessed_at >= '2025-01-01'
)
SELECT device, COUNT(*)
FROM recent_logs
GROUP BY device;
```

Avantages :

- Lisibilité
- Débogage
- Structuration du raisonnement

CTE : DÉCOUPER LOGIQUEMENT LE PROBLÈME

PostgreSQL peut :

- matérialiser le CTE
- empêcher certaines optimisations

Toujours vérifier avec : EXPLAIN ANALYZE

Solution : Vue matérialisée

Une vue matérialisée =

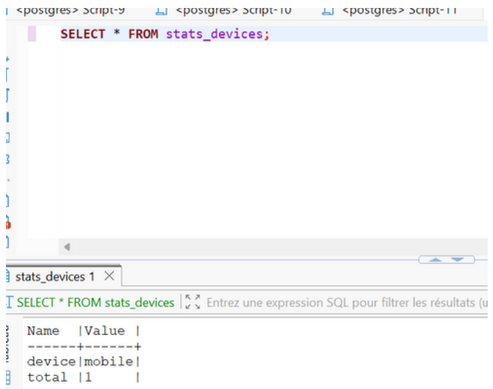
Une table cachée qui contient le résultat pré-calculé.

Utilisation

SELECT * FROM stats_devices;

<postgres> Script-9 <postgres> Script-10 <postgres> Script-11

```
CREATE MATERIALIZED VIEW stats_devices AS
SELECT device, COUNT(*) AS total
FROM access_logs
GROUP BY device;
```



TESTS COMPARÉS

Méthode professionnelle

On ne dit jamais “c’est plus rapide” sans chiffres.

EXPLAIN ANALYZE

EXPLAIN (ANALYZE, BUFFERS)

MONGODB

MongoDB est une base de données

NoSQL orientée documents :

- Les données sont stockées en documents JSON
- Regroupés dans des collections
- Sans schéma strict imposé

MONGODB

MongoDB = mêmes problèmes, autre moteur

Idée clé

MongoDB n’est pas “**magiquement rapide**”.

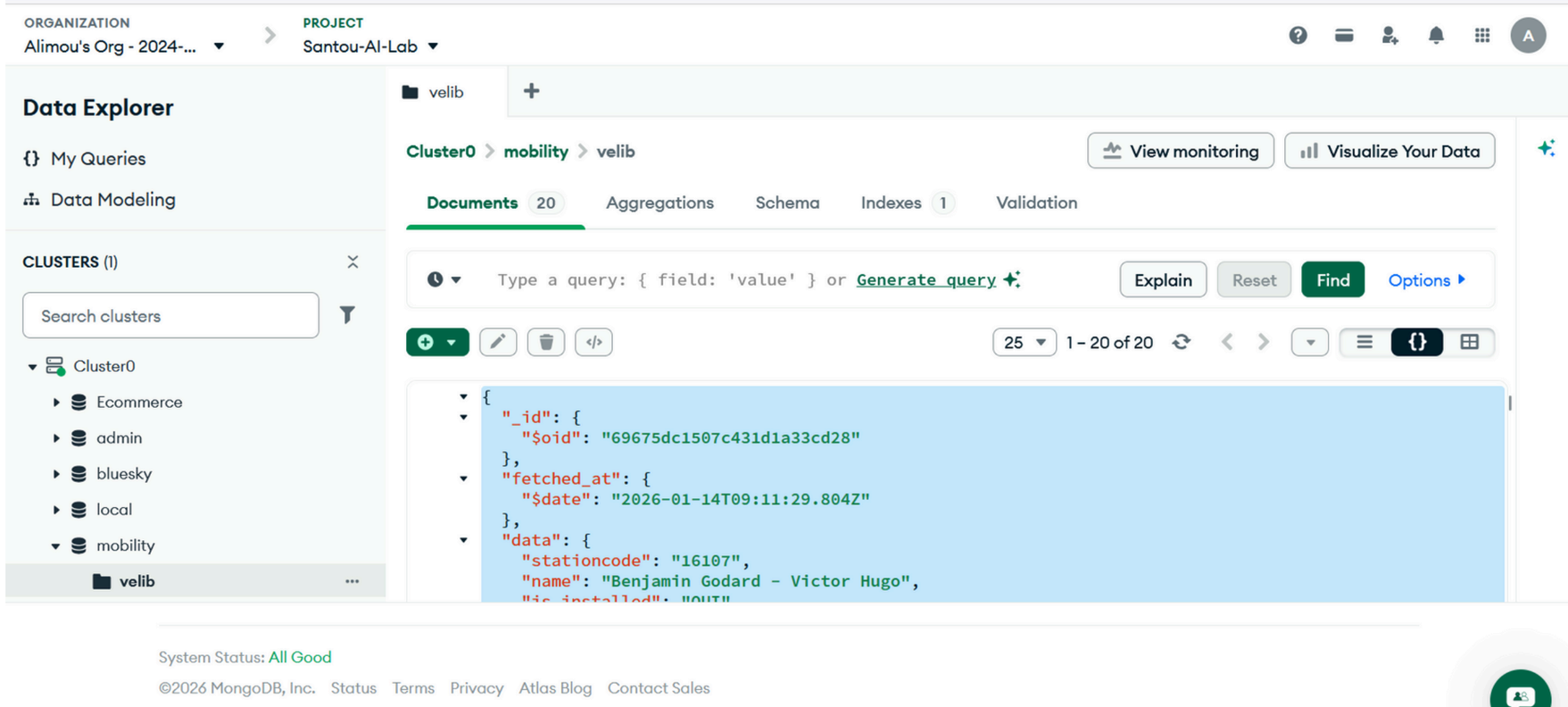
Comme PostgreSQL :

- **Sans index** → scan complet
- **Avec index** → recherche dans une structure arborescente (B-tree)

COMMENT MONGODB STOCKE LES DONNÉES

Les données sont stockées sous forme de documents JSON

Regroupés dans des collections



État actuel d'une station :

Que se passe-t-il sans index

```
db.velib.find({ "data.stationcode": "16107" })
```

COMMENT MONGODB STOCKE LES DONNÉES

Analyse réelle

b.velib.find({ "data.stationcode": "16107" }).explain("executionStats")

MongoDB lit toute la collection pour trouver 1 document.

stage: COLLSCAN

totalDocsExamined: 20

nReturned: 1

>_MONGOSH


```
          '$eq': '16107'
        }
      },
      direction: 'forward'
    },
    rejectedPlans: []
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 1,
    executionTimeMillis: 0,
    totalKeysExamined: 0,
    totalDocsExamined: 20,
    executionStages: {
      isCached: false,
      stage: 'COLLSCAN',
      filter: {
        'data.stationcode': {
          '$eq': '16107'
        }
      },
```

CRÉATION DE L'INDEX MÉTIER

appliquons la même requete

```
db.velib.find({ "data.stationcode": "16107" })
        .sort({ fetched_at: -1 })
        .limit(1)
        .explain("executionStats")
```

```
nReturned: 1,
executionTimeMillisEstimate: 0,
works: 1,
advanced: 1,
needTime: 0,
needYield: 0,
saveState: 0,
restoreState: 0,
isEOF: 0,
docsExamined: 1,
alreadyHasObj: 0,
inputStage: {
  stage: 'IXSCAN',
  nReturned: 1,
  executionTimeMillisEstimate: 0,
  works: 1,
  advanced: 1,
  needTime: 0,
  needYield: 0,
```

>_MONGOSH

```
> db.velib.createIndex({
    "data.stationcode": 1,
    "fetched_at": -1
  })
< data.stationcode_1_fetched_at_-1
Atlas atlas-5pqsgt-shard-0 [primary] mobility>
```

Accès direct, ultra rapide

CRÉATION DE L'INDEX MÉTIER

appliquons la même requete

```
db.velib.find({ "data.stationcode": "16107" })
        .sort({ fetched_at: -1 })
        .limit(1)
        .explain("executionStats")
```

```
nReturned: 1,
executionTimeMillisEstimate: 0,
works: 1,
advanced: 1,
needTime: 0,
needYield: 0,
saveState: 0,
restoreState: 0,
isEOF: 0,
docsExamined: 1,
alreadyHasObj: 0,
inputStage: {
  stage: 'IXSCAN',
  nReturned: 1,
  executionTimeMillisEstimate: 0,
  works: 1,
  advanced: 1,
  needTime: 0,
  needYield: 0,
```

>_MONGOSH

```
> db.velib.createIndex({
    "data.stationcode": 1,
    "fetched_at": -1
  })
< data.stationcode_1_fetched_at_-1
Atlas atlas-5pqsgt-shard-0 [primary] mobility>
```

Accès direct, ultra rapide

AUTRE REQUÊTE MÉTIER

Stations presque vides :

```
db.velib.find({
  "data.numbikesavailable": { $lt: 3 }
})
```

Index secondaire

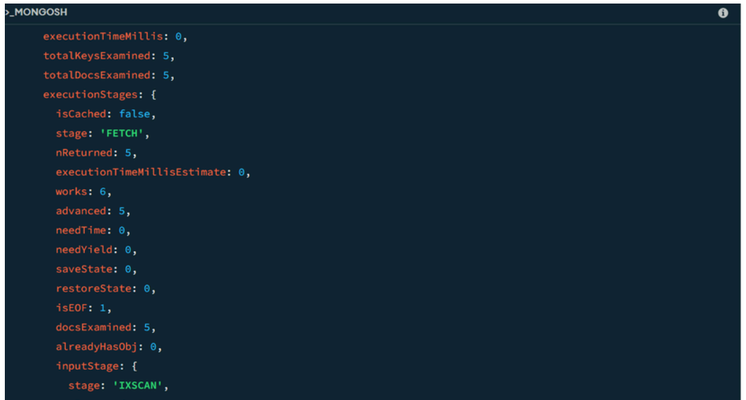
```
db.velib.createIndex({
  "data.numbikesavailable": 1
})
```

```
>_MONGOSH

> db.velib.createIndex({
    "data.numbikesavailable": 1
  })
< data.numbikesavailable_1

Atlas atlas-5pqsgt-shard-0 [primary] mobility>
```

Vérification avec explain



COMPARAISON AVEC POSTGRESQL

Stations presque vides :

PostgreSQL	MongoDB
Seq Scan	COLLSCAN
Index Scan	IXSCAN
EXPLAIN ANALYZE	explain("executionStats")

TP 2 — OPTIMISATION AVANCÉE SQL & NOSQL

Nowledgeable Mon espace étudiant Mo

1. TP2 — Optimisation avancée S...

1. TP2 — Optimisation avancée SQL & NoSQL

Objectif général

Mettre en place et optimiser une plateforme d'analyse de données issues d'une API publique réelle afin de :

- structurer efficacement les données
- optimiser les requêtes analytiques
- comparer les approches SQL et NoSQL
- mesurer objectivement les gains de performance

PHASE 1 — Mise en place des données (SQL + MongoDB)

Objectif : construire une base exploitable et volumineuse.

- Écrire un script Python de collecte depuis une API publique
- Stocker :
  - les données brutes dans MongoDB
  - les données structurées dans PostgreSQL
- Concevoir le modèle SQL (table de mesures + tables de dimensions)
- Justifier les choix de :
  - types de données
  - clés
  - structure
- Accumuler au moins :
  - 500 000 à 1 000 000 de mesures
- Mesurer :
  - le temps d'ingestion
  - la taille occupée sur disque

PHASE 2 — Diagnostic des performances (SQL)

- Définir des requêtes métier :