

# Apprendre les types de données avancés de manière inversée



ARRAY, LTREE, GIN et GiST



Paris, France 2024

**PERCONA**  
UNIVERSITY

Yoann La Cancellera

Senior support engineer at Percona



# Intro

**Le but:** vous encourager à tester et challenger vos choix de typage!

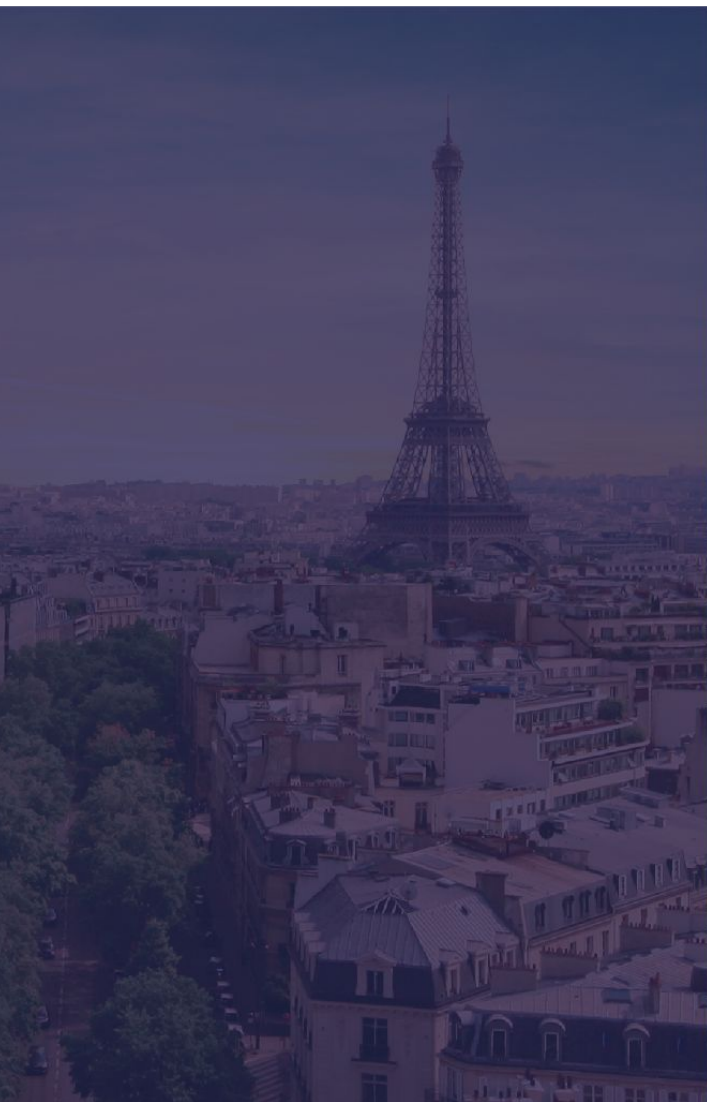
Tout est répétable, les commandes sont fournies:



: <https://github.com/ylacancellera/talks>

La démo se base sur des données publiques:

<https://github.com/credativ/omdb-postgresql>



## Situation initiale

# Le schema: **n-n** avec table de pivot

~15.000 categories

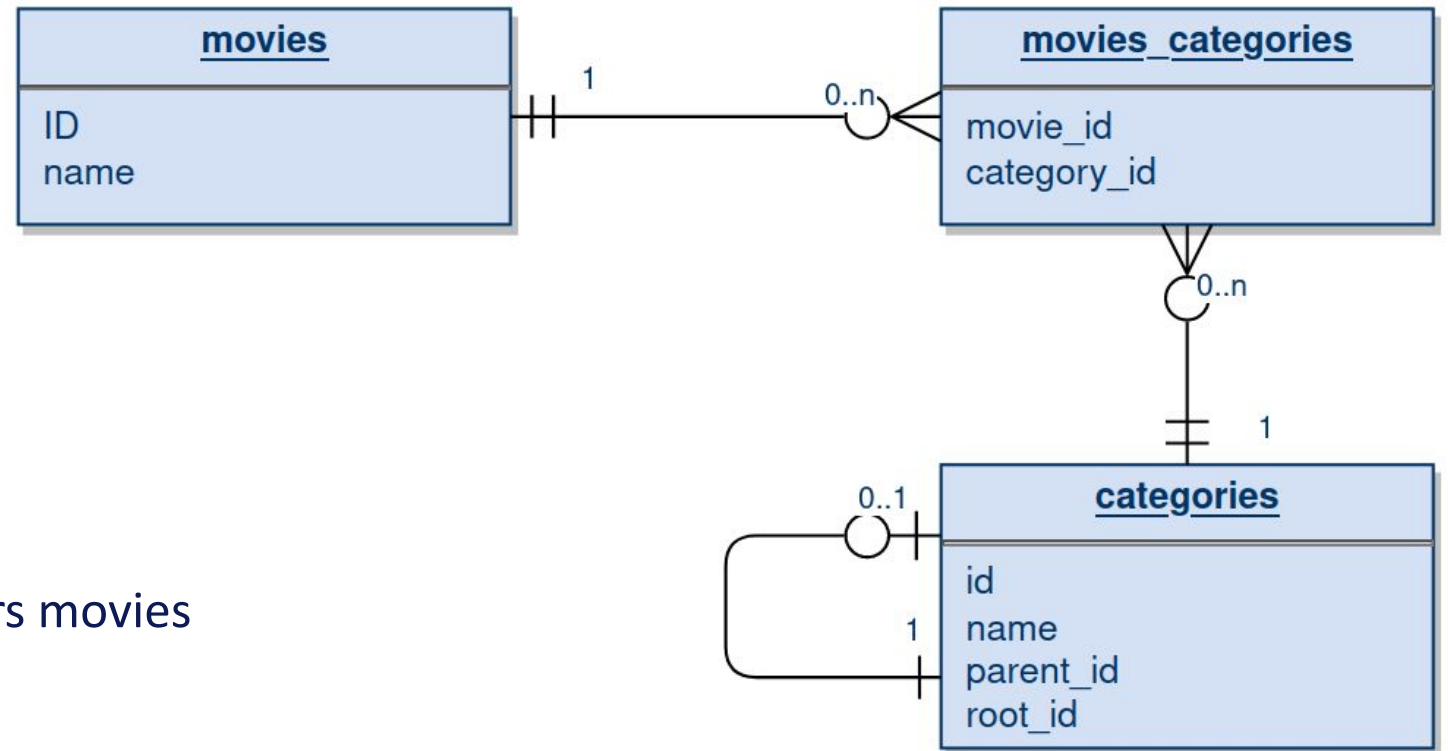
~200.000 movie\_categories

~190.000 movies

1 movie a plusieurs categories

1 categorie est partagé par plusieurs movies

Il y a des sous-catégories par catégories



# CTE ?

Comme une table temporaire

Limité pour cette requête

N'est jamais persisté

```
WITH cte_cs AS (  
    SELECT c.name, c.id, c.parent_id, c.root  
    FROM movie_categories mc  
    JOIN categories c  
    ON mc.category_id = c.id  
    WHERE movie_id = 77  
)  
SELECT name, id, parent_id  
FROM cte_cs  
ORDER BY id, parent_id;
```

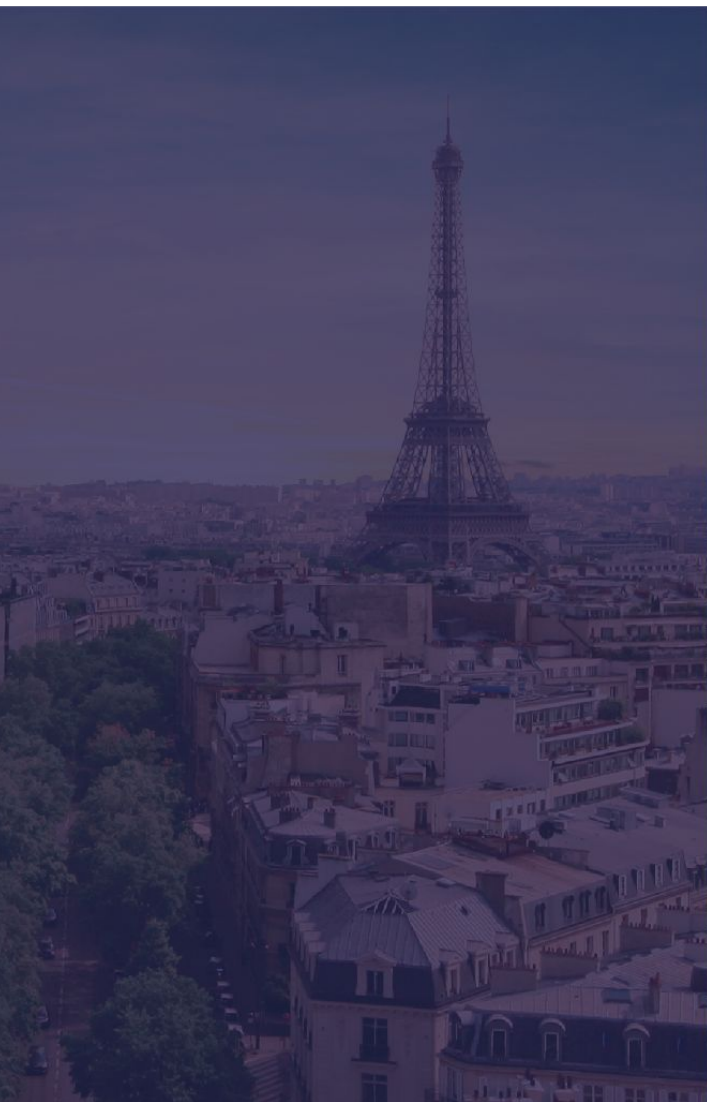


Requête simple  
et directe

```
WITH RECURSIVE cte_cs AS (  
    SELECT c.name, c.id, c.parent_id, c.root  
    FROM movie_categories mc  
    JOIN categories c  
    ON mc.category_id = c.id  
    WHERE movie_id = 77
```

Partie récursive

```
    UNION  
    SELECT c2.name, c2.id, c2.parent_id, c2.root_id  
    FROM categories c2  
    JOIN cte_cs  
    ON cte_cs.parent_id = c2.id  
)  
SELECT name, id, parent_id  
FROM cte_cs  
ORDER BY id, parent_id;
```



## 1ère étape: ARRAY et GIN

Stocker chaque categories et sous-categories qui décrit un movie dans la même ligne

Puis, l'utiliser directement pour filtrer la table

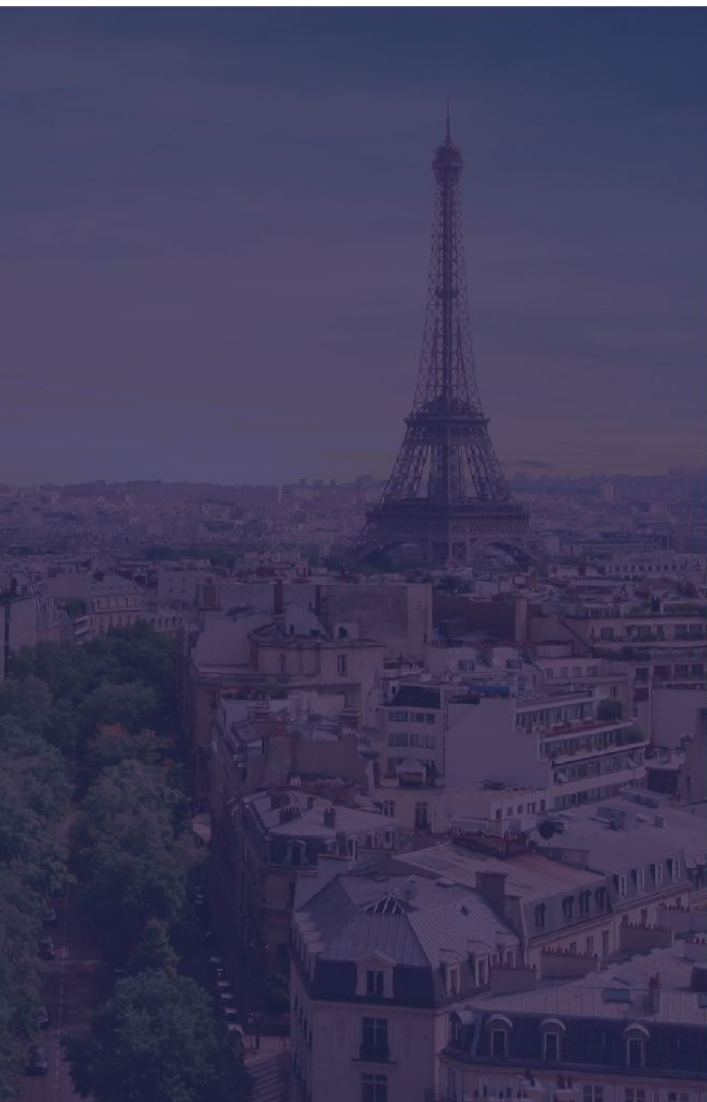
=> dénormalisation

| <u>movies_array</u>                  |
|--------------------------------------|
| ID<br>name<br>categories VARCHAR [ ] |

# opérateur @> ?

“array\_1”                      inclut                      array\_2”

```
omdb=# select '{1, 2, 3, 4, 5}'::int[] @> '{2, 5}'::int[];  
?column?  
-----  
t => TRUE  
(1 row)
```



## 2e étape: LTREE et GiST

Stocker chaque catégories dans la ligne, tout en gardant leur hierarchies

### movies\_arrayltree

ID  
name  
categories LTREE [ ]

# Opérateur @>, mais aussi @

“ltree\_1”                      inclut                      “ltree\_2”

```
omdb=# SELECT 'percona.university.paris'::ltree @> 'percona.university.paris.postgres';
?column?
```

-----  
t => TRUE  
(1 row)

“ltree”                      possède                      “élément”

```
omdb=# select 'percona.university.paris.postgres'::ltree @ 'paris';
?column?
```

-----  
t => TRUE  
(1 row)



# Resultats

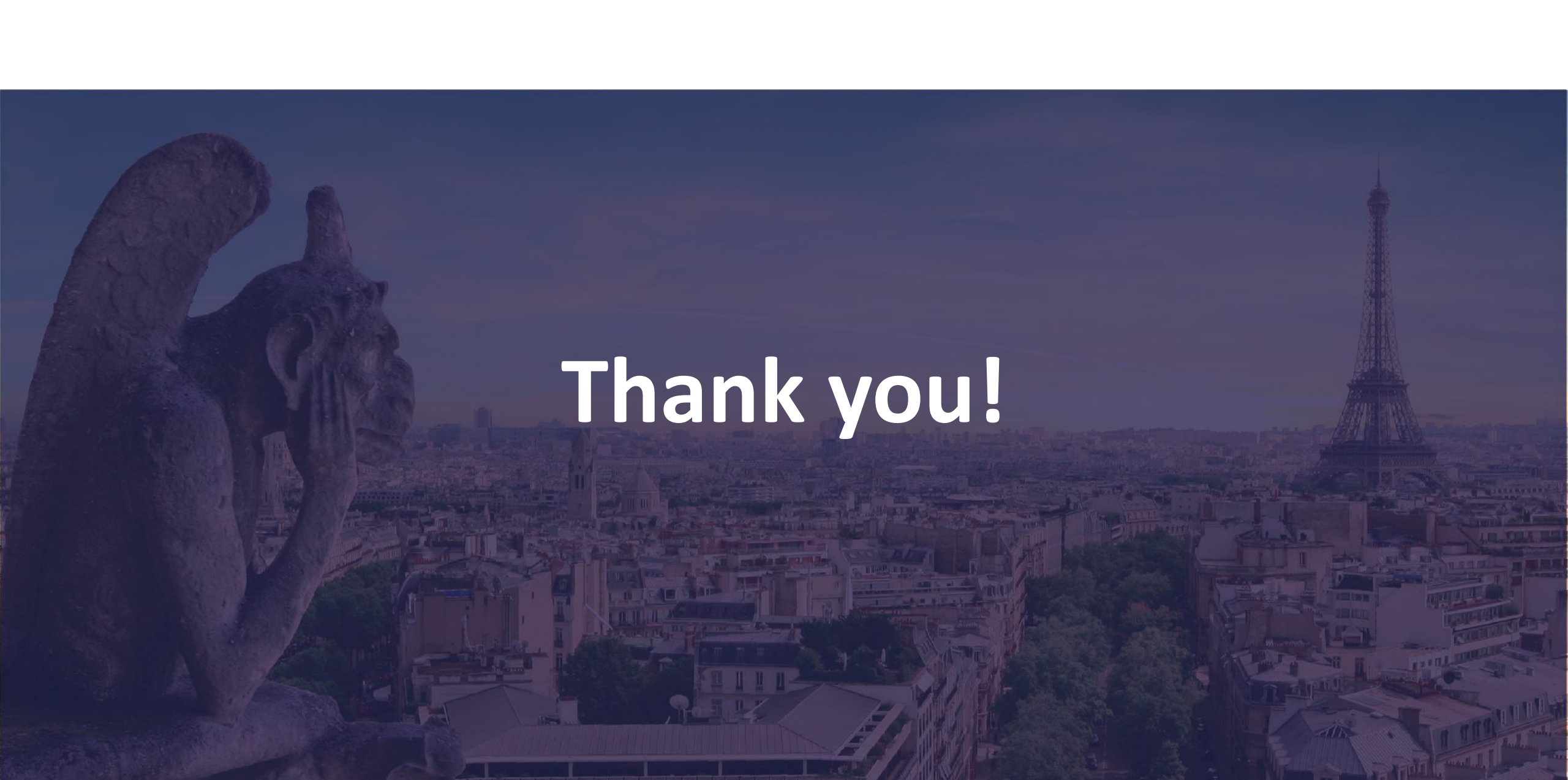
## Recherche accélérée sans perdre le sens de la donnée

### Avantages

- Complexité d'exécution divisée
- Possibilité de recherches avancées via LQUERY
- Lisibilité et simplicité de la requête
- Modification des liens hiérarchiques des catégories simplifiée

### Inconvénients

- Duplication de la donnée
- La donnée a dû être adapté: les symboles et espaces remplacés par des tirets
- L'opérateur n'est pas intuitif sans connaissances



Thank you!

