PERCONA

PimpMySchema: dénormaliser des tables en un index

Une requête complexe traduite en un index

Intro

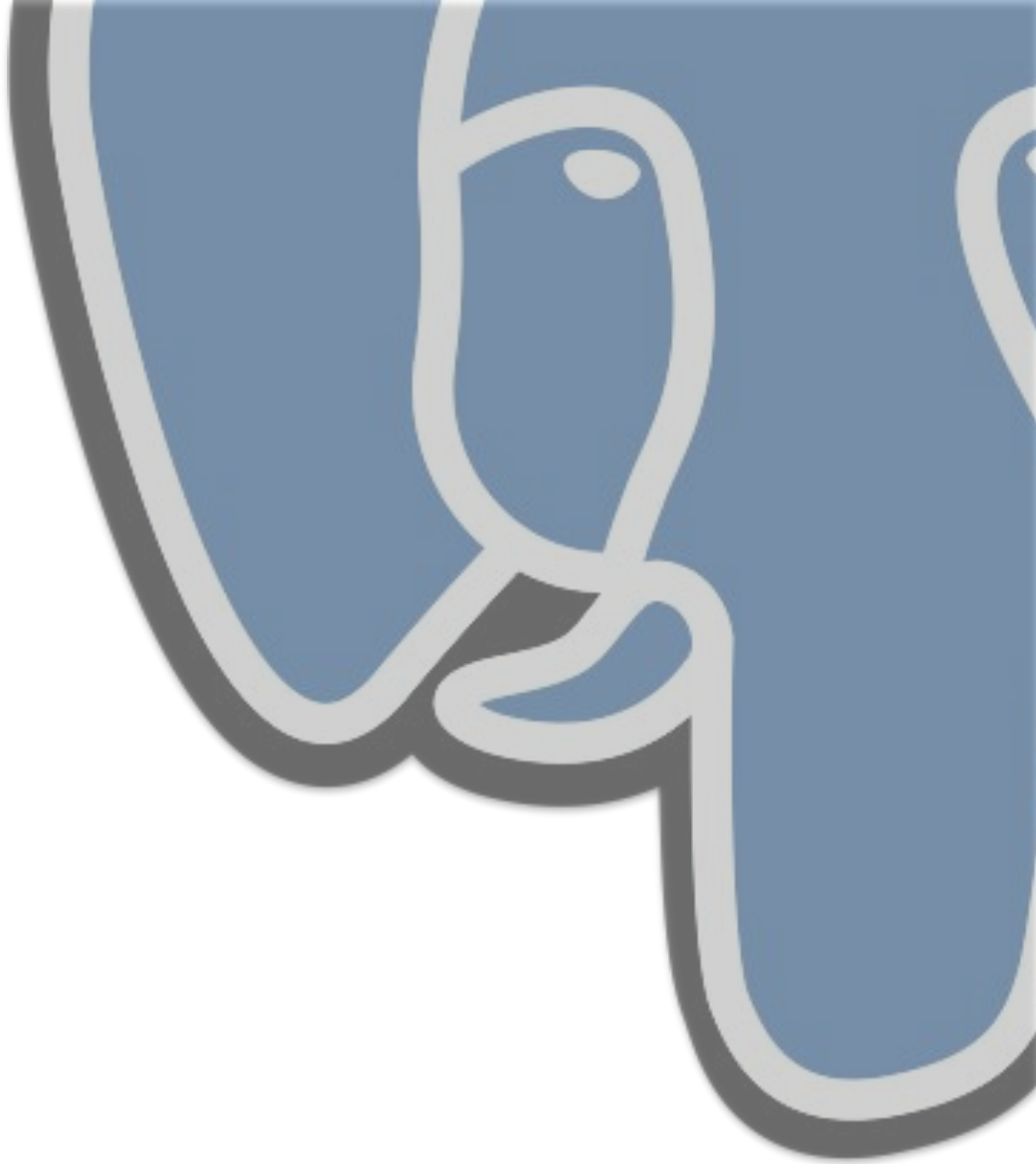
Yoann La Cancellera
Support engineer chez Percona, éditeur bdd open-source

J'investigue des problèmes de performance SQL
(et des bugs mysql le reste du temps)

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

Ce talk se base sur https://github.com/credativ/omdb-postgresql

Sommaire



1. Situation de base:
   - Présentation schéma
   - Quelques requêtes visuelles
   - **Problème:** Requête avancée récursive
2. Simplification par **ARRAYS** et GIN
3. Simplification par **LTREE** et GIST

Situation de base

et requête complexe

Le schéma: relation n-n avec table pivot

~15.000 categories

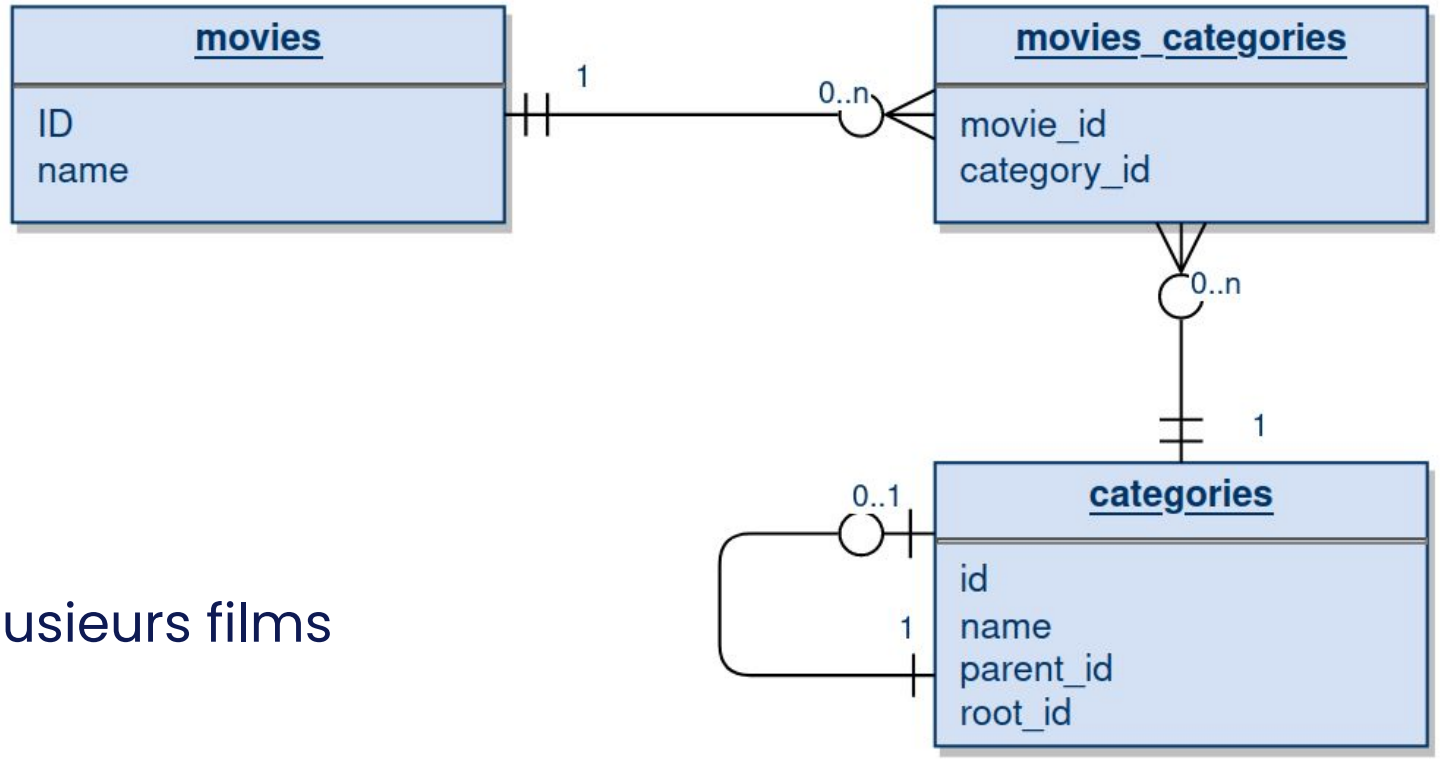
~200.000 movie_categories

~190.000 movies

1 film a plusieurs catégories

1 catégorie est partagé par plusieurs films

Il y a des sous-catégories de catégories



Une CTE ? Récursive ?

CTE ?

Une sorte de table temporaire

Mais uniquement utilisable pour une requête

```
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 normal

Partie récursive

```
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
    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;
```

COST, plan ?

Le cost: une valeur sans unité qui estime le coût d'exécution

Par exemple:

- "**1**" représente le chargement en mémoire d'une page
- "**0.0025**" pour une utilisation d'opérateur (+, -, *, /)
- "**0.01**" pour traiter une ligne

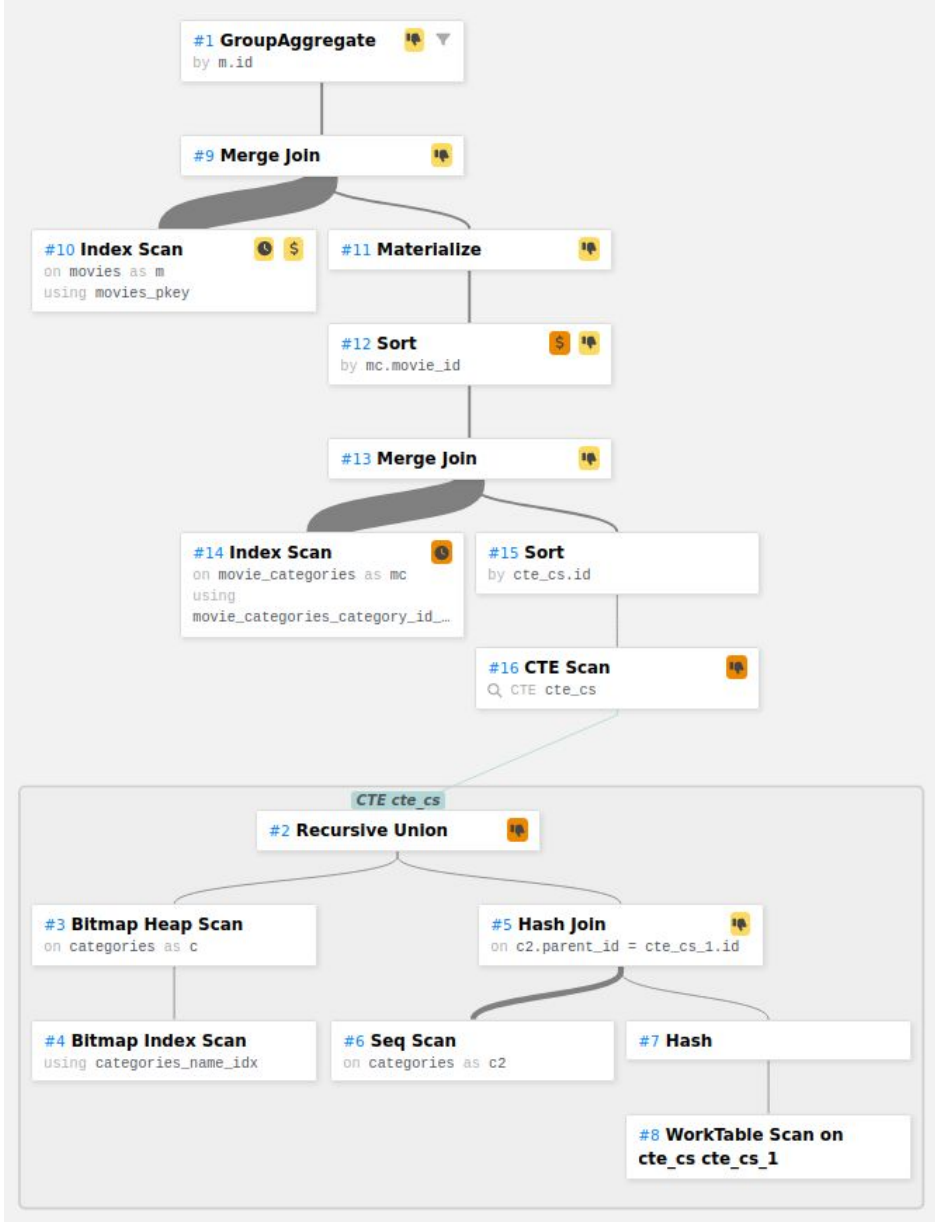
Cost plus bas = plus rapide
(généralement)

COST, plan ?

Le plan:

le chemin (ordre des tables) et la méthode (loops, hash, ...) pour exécuter une requête.

Postgres calcule le coût de chaque plan possible, **le moins coûteux sera utilisé**



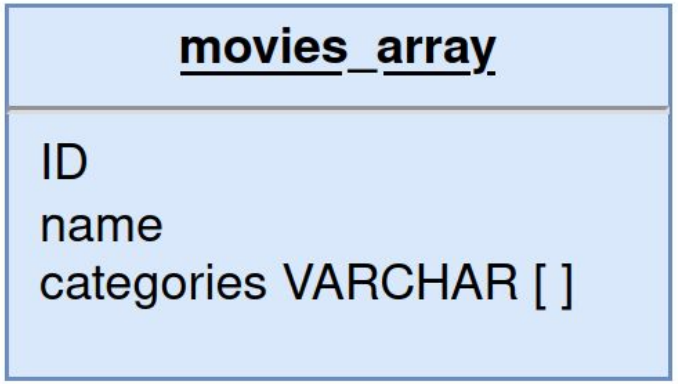
ARRAY et son GIN

GIN = Generalized Inverted Index

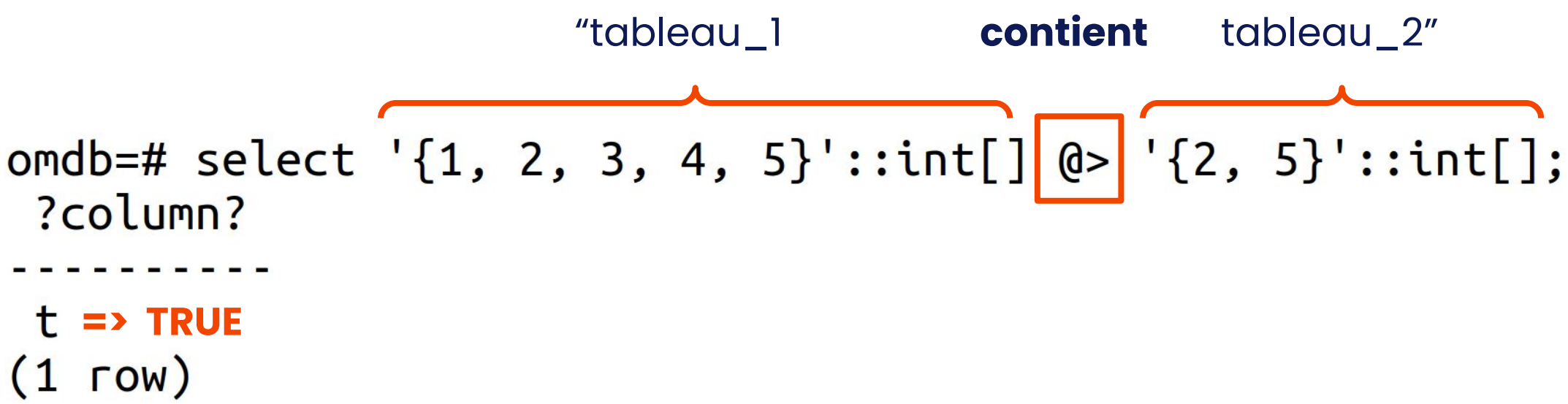
Le but:

Stocker toutes les catégories et sous catégories qui caractérise le film

Filtrer via ce tableau directement



Opérateur @> ?



Résultat de l'array avec index GIN

Le type tableau permet d'éviter la table catégories ET la table pivot

Avantages

- Coût divisé par 10 000
- accès qu'à une seule table
- requête grandement simplifiée à la lecture
- un sentiment d'accomplissement

Inconvénients

- on perd la validation qu'une catégorie existe bien
- on duplique de la donnée (=> dénormalisation)
- on perd la notion de sous-catégories

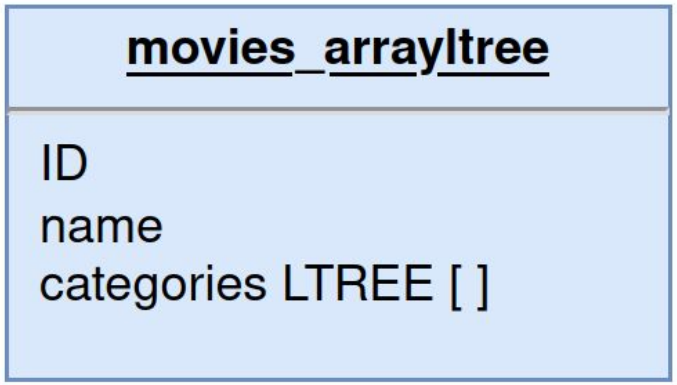
  (=> On a dû associer memento à "Genre", ce qui perd de sens)

LTREE et son GiST

GiST = Generalized Search Tree

Le but:

Stocker toutes les hiérarchies de catégories associées à un film



Opérateur @> encore, mais également @

```
omdb=# SELECT 'meetups.postgres.lille'::ltree <@ 'meetups.postgres';
 ?column?
----------
 t => TRUE
(1 row)

omdb=# SELECT 'meetups.postgres.lille'::ltree @ 'lille';
 ?column?
----------
 t => TRUE
(1 row)
```

Ltree et @

```
omdb=# SELECT 'meetups.postgres.lille'::ltree @ 'lille & mysql';
 ?column?
----------
 f   => FALSE
(1 row)

omdb=# SELECT 'meetups.postgres.lille'::ltree @ 'lille & (mysql | postgres)';
 ?column?
----------
 t   => TRUE
(1 row)
```

Résultat de l'array de LTREE avec index GIST

On renforce le type ARRAY en conservant les relations hierarchiques

Avantages

- avantages de l'ARRAY
- encore plus de fonctionnalités de recherches avec le langage LQUERY
- maintien de la hiérarchie de catégories

Inconvénients

- on perd encore la validation qu'une catégorie existe bien
- on duplique toujours de la donnée
- On a dû nettoyer les données: remplacer les symboles et espaces par un underscore

Conclusion

Conclusion

Les types avancés peuvent libérer de l'aspect stricte du SQL relationnel.

EXPLAIN, test, EXPLAIN ANALYZE

Mais pas de solutions magiques:

- Attention au biais de "la solution qui cherche un problème"
- 95% du temps, la réponse est de mieux normaliser

Extras

COST, plan ?

```
GroupAggregate  (cost=102474.55..139061.44 rows=987 width=57) (actual time=50.242..95.575 rows=13 loops=1)
```

“Node”

Cost courant .. cost total

```
GroupAggregate  (cost=102474.55..139061.44 rows=987 width=57)
```

=> EXPLAIN

Temps courant .. temps total

```
(actual time=50.242..95.575 rows=13 loops=1)
```

=> ANALYZE

Cost de l’étape = Cost total - Cost courant

Cost courant = Somme des COST des étapes sous-jacentes

Un peu de lectures

https://www.postgresql.org/docs/current/gin-implementation.html

https://www.postgresql.org/docs/12/functions-array.html

Limites de GIN:
https://gitlab.com/gitlab-com/gl-infra/production/-/issues/4725#note_596146675

https://www.postgresql.org/docs/current/ltree.html

Thank You!