PERCONA

Learning advanced data types

The reverse way

Intro

Yoann La Cancellera

Senior support engineer at Percona

When coming into the field, I was frustrated that talks on datatypes were starting from ideal data.

Mine was far from it, like any other legacy system.

So this talk aims to bridge this gap!

This talk can be repeated ! Every commands used are detailed

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

This demo is based on public data: https://github.com/credativ/omdb-postgresql

Plan

1. Initial situation:
    - Starting table schema
    - Showcasing some data and access
    - **Problem:** it quickly require recursive queries

2. Simplifying using **ARRAYS** and GIN

3. Simplifying using **LTREE** and GIST

Initial situation

and complex queries

The schema: n-n relationship with pivot table

~15.000 categories

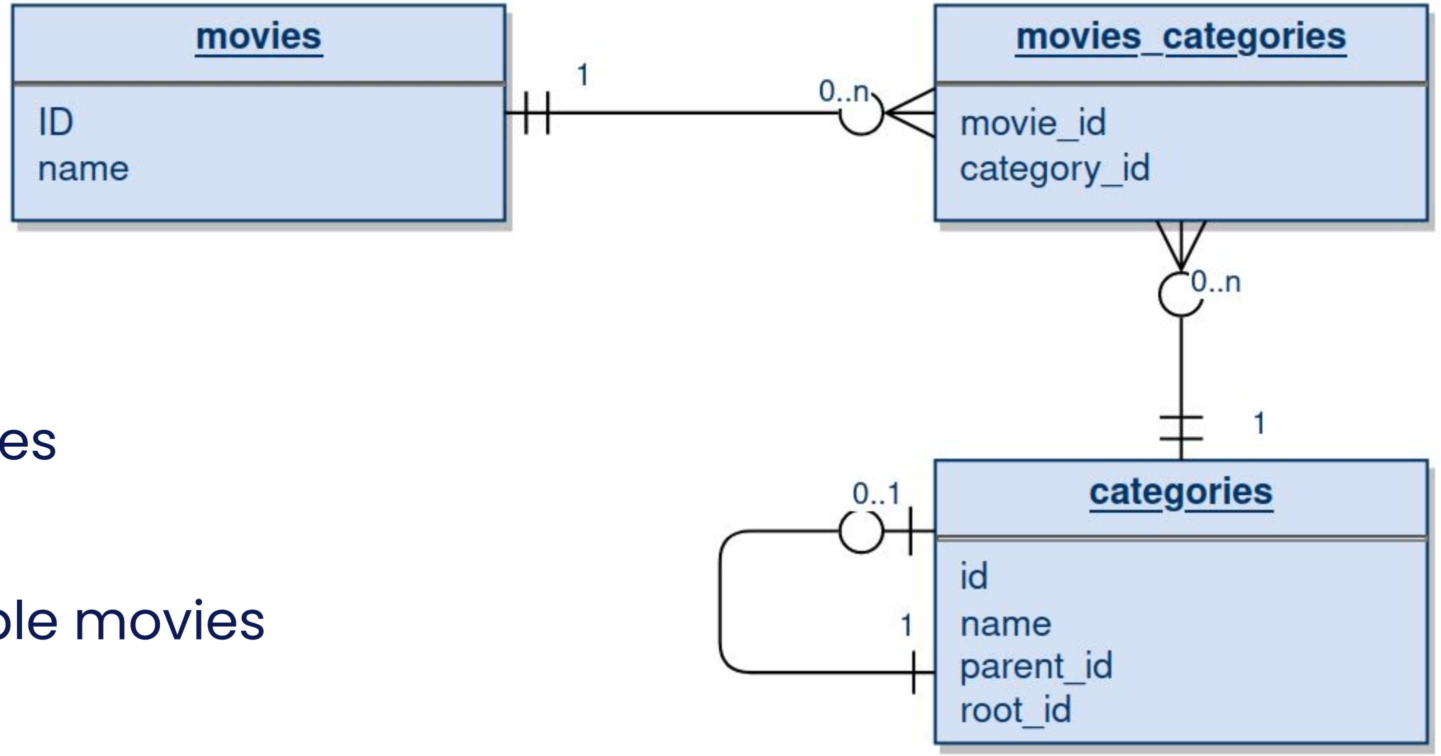
~200.000 movie_categories

~190.000 movies

1 movie has multiple categories

1 category is shared by multiple movies

There are subcategories of categories



CTE ? Recursive ?

CTE ?

Some sort of temporary table

Only lives for this query scope

Is not persisted at any point

```
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;
```

Regular part we queried initially

Recursive part

```
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 ?

Cost: a value without unit to estimate the cost of execution

For example:

"**1**" represents loading a page in memory

"**0.0025**" for any common operator usage (+, -, *, /)

"**0.01**" to handle a row

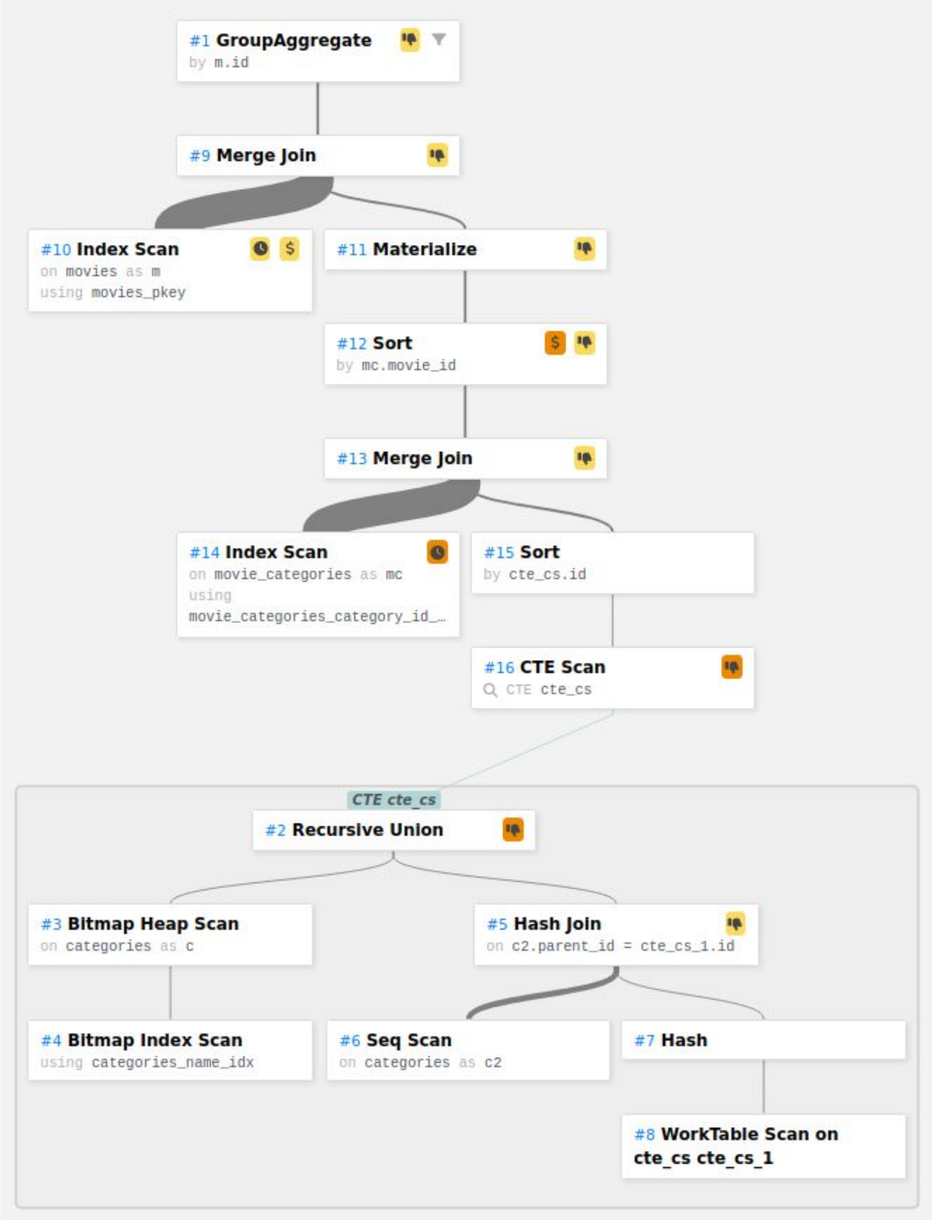
Lower cost = faster
(mostly, at least that's the theory behind it)

COST, plan ?

Plan:

This is the chosen paths (order of tables) and methods(loops, hash, …) to execute a query.

Postgres will estimate the cost of different possible plan, **and will choose the least costly one**



ARRAY with its GIN

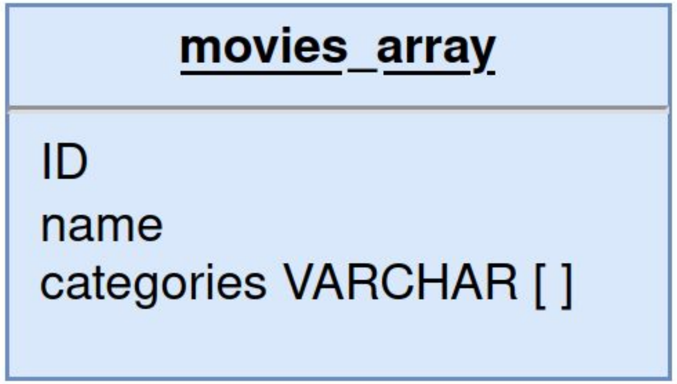
GIN = Generalized Inverted Index

Goal:

Storing every categories and subcategories that is describing a movie, into said movie's row

Then, use it to filter movies directly without joins

This would be a denormalization



@> operator ?

“array_1 **includes** array_2”

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

Results of using an array indexed with GIN

The array type enable to skip both categories table AND the pivot table

Pros

- Cost divided by 10 000
- Single table access
- Way more readable queries
- A sensible feeling of success

Cons

- We lose data validation
- Data is duplicated (=> denormalized), so it can get out of sync
- We lose the notion of subcategories

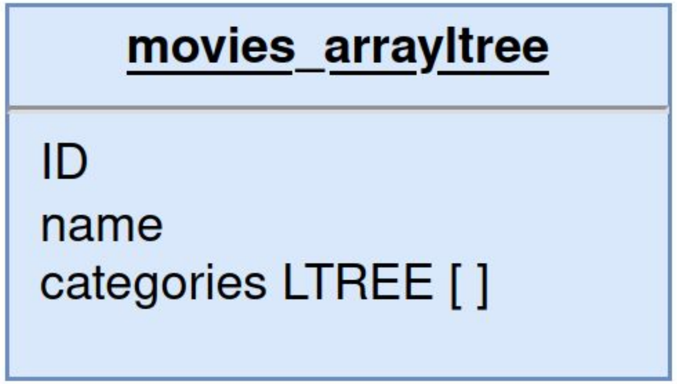
  (=> we had to link “Memento” to the broad category “Genre”, which does not make sense on its own)

LTREE with GiST

GiST = Generalized Search Tree

Goal:

Store every categories describing movies, while keeping their hierarchies



PERCONA

@> operator again, but also @

"ltree_1 **includes** ltree_2"

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

"ltree **has** item"

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


PERCONA

Ltree and @

```
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)
```

Results of using an LTREE indexed with GiST

We reinforce the ARRAY type, while keeping data hierarchical links

Pros

- ARRAY's advantages
- Even more search features through the LQUERY language
- Subcategories hierarchies are maintained

Cons

- We still lose the validation that a category do exists in categories table
- We still duplicate data
- We had to adapt data: replacing symbols and white spaces by underscores

Conclusion

Not magical solutions

Writing to GIN has a cost: updates are delayed until the next vacuum or analyze

ARRAYs and LTREE may not be supported by your ORM

The important one:

Most of the time, performance issue is because of badly normalized data in the first place

That’s it!

Happy to get feedbacks from it !

(even mean ones, but be creative)

Be sure to test our product!

➔ Percona distribution for Postgres
➔ Percona Operator for PostgreSQL
➔ Percona Monitoring Management

We have a TDE solution looking for testers!

➔ **github.com/Percona-Lab/postgresql-tde**

Extras

COST, plan ?

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

"Node"

"current" cost so far .. total cost

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

=> EXPLAIN

"current" time so far .. total time

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

=> ANALYZE

Cost of the step = total cost - current cost

Current cost = Sum of every costs from earlier steps

Thank You!