Query plan freezing extension

design, issues and lessons learned

Belyalov D., Lepikhov A., Rybakina A.

Postgres Professional

2023

Self Introduction

- Specialist in Applied Mathematics graduated from Chelyabinsk State University in 2005
- Ph.D. in Computer Sciences (Distributed Databases) awarded at Moscow State University in 2008.
- Working in Postgres Professional as a Core Developer since 2017.
- Designed the Shardman project architecture on its earlier steps
- Worked on Multimaster project
- Working on various query optimization issues

Who we are

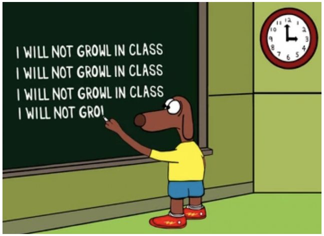
- Research team, part of Postgres Professional, dealing with optimization issues
- Caused by the idea of sustainable coding
- Design enterprise and core features to improve the planner effectiveness
- Projects: Self-Join Removal, Asymmetric JOIN, Optimized Group-by, AQO, sr_plan ...

Plan freezing?

Reason 1:
Don't optimize next time!

pgbench:

- With planning: $\approx$ 6000 tps
- Prepared statements: $\approx$ 7600 tps


I WILL NOT GROWL IN CLASS
I WILL NOT GROWL IN CLASS
I WILL NOT GROWL IN CLASS
I WILL NOT GRO

Plan freezing?

Reason 2:
Pin tweaked query plan into the plan cache:

- Predictable execution time
- No planning surprises after minor upgrade

Rationale

- Stale statistics
- Imperfection of cost estimation algorithms
- Implicit functional dependencies between columns

Functional Dependencies (overestimation)

```
CREATE TABLE people (
  name           text,
  occupation     text,
  sex            boolean,
  region         text,
  is_vaccinated  boolean
);
```

```
SELECT * FROM people t1
WHERE occupation = 'Tractor Driver' AND sex = 'female'
/*
Seq Scan on people  (rows=1584) (actual rows=1)
   Filter: (is_woman AND (occupation = 'Tractor Driver'))
   Rows Removed by Filter: 99999
 */
```

Functional Dependencies (underestimation)

```
CREATE TABLE people (
  name           text,
  occupation     text,
  sex            boolean,
  region         text,
  is_vaccinated  boolean
);
```

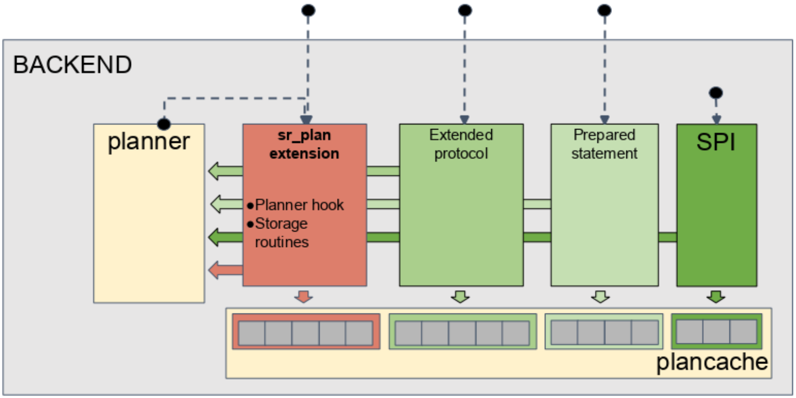
```
SELECT * FROM people
WHERE region = 'Chelyabinsk' AND is_vaccinated;
/*
 Seq Scan on people  (rows=114) (actual rows=907)
   Filter: (is_vaccinated AND (region = 'Chelyabinsk'))
   Rows Removed by Filter: 99094
 */
```

PostgresPro

Our conjecture

Unfortunate planning is inevitable [at least, for now]. Assuming someone or something could force the planner to generate a better plan, we should provide the tool to freeze the right solution for the subsequent executions.

Stick on the plan in the Plan Cache ?



The sr_plan extension

- Abbreviates **s**ave/**r**estore plan
- Introduced in Postgres Pro Enterprise 15 (dont́ mix up with the extension sr_plan existed up to PGPro Enterprise 13!)
- Freezes specific plan for a [parameterized] query

How it works

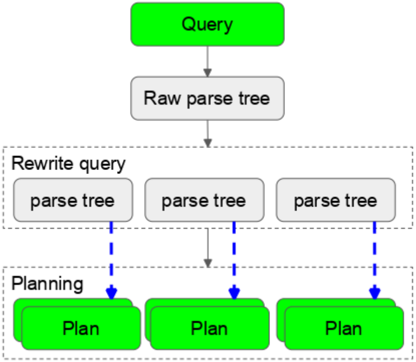
- sr_register_query('SELECT ... WHERE x = $1 AND y = 42', ...)
- sr_plan_freeze(srid)
- sr_plan_unfreeze(srid)

Frozen plan have a special node in explain

```
                         QUERY PLAN
----------------------------------------------------
 Custom Scan (SRScan) (actual rows=1 loops=1)
   Frozen plan ID: 1
   ->  Aggregate (actual rows=1 loops=1)
      ->  Seq Scan on a (actual rows=10 loops=1)
            Filter: ((x = \$1) AND (y = 42))
```

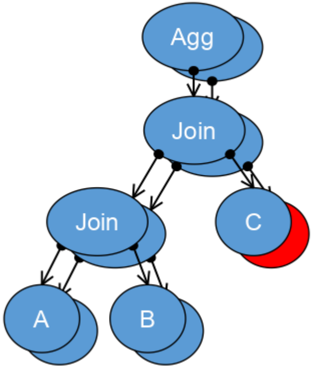
Lesson 1

In DBMS, the way from a query text to the plan is not straightforward.



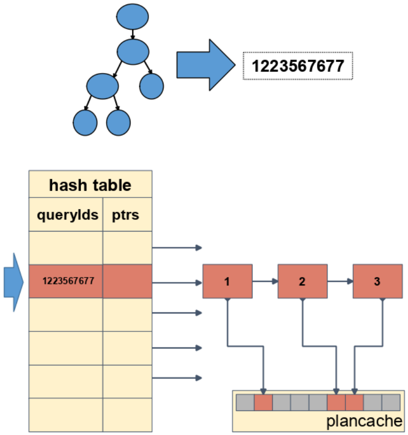
Lesson 2

Only one way to prove applicability of the plan to the given query is to compare stored and incoming parse trees



Lesson 3

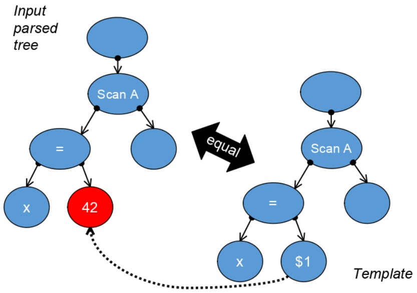
To make overhead admissible, we should have kind of parse tree signature - queryId



Lesson 4

To apply plan freezing for parameterized queries we should '*generalize*' the parse tree

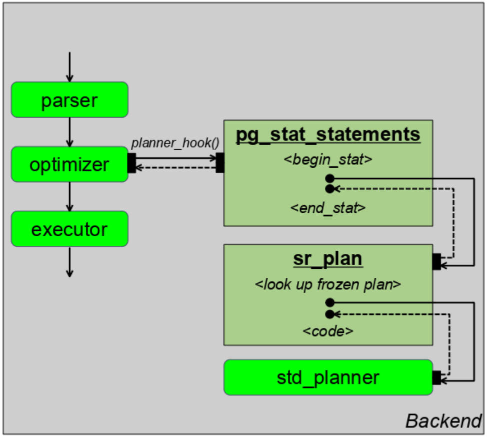
sr_register_query('SELECT ... FROM A, ... WHERE x = $1...')



PostgresPro

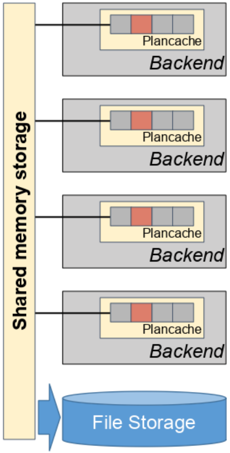
Lesson 5

The extension loading order matters



Points of overhead

- Parse tree comparison
- Plan invalidation
    - Per-backend cache invalidation
    - Disc storage sync
    - Transactional issues



Freeze pgbench

Freezing procedure

```
SELECT srid FROM sr_register_query('
  SELECT abalance FROM pgbench_accounts
  WHERE aid = $1', 'int')  \gset
SELECT abalance FROM pgbench_accounts WHERE aid = 1;
SELECT sr_plan_freeze(:srid);
```

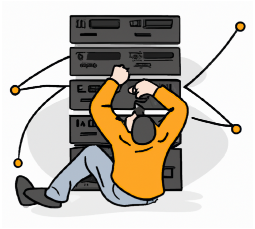
```
UPDATE pgbench_accounts ...
UPDATE pgbench_tellers ...
UPDATE pgbench_branches ...
INSERT INTO pgbench_history ...
```

- With planning: $\approx$ 6000 tps
- Frozen statements: $\approx$ 6500 tps

PostgresPro

The future

- Detect bad plan and try something different
- Plan transfer procedure
- Global prepared statements



Questions ?