credativ

Optimizer Hints in PostgreSQL

Michael Banck <michael.banck@credativ.de>

PGDay Lowlands 2026

credativ GmbH

- Founded 1999 in Mönchengladbach
- Strong ties to the Open-Source community
- 40 Open-Source specialists
- Consulting, Development, Trainings, Support (3rd-Level / 24x7)
- Open-Source infrastructure with Debian, Kubernetes and Proxmox
- Open-Source databases with PostgreSQL
- Open-Source DevOps with Ansible, Puppet, Terraform and others
- Independent owner-operated company again since 2025
- We are hiring: `https://www.credativ.de/karriere/`

Optimizer Hints
Agenda

- Introduction
- Planner Node Overrides
- External Extension
- Internal Plan Advisor

Introduction

Introduction
Planner Overview

credativ

- Postgres has a cost-based query-planner
- Startup and final costs are computed for each plan node
- Planner tries all possible combinations
- Plan with lowest overall costs is being executed
- This is not necessarily the plan with the shortest runtime
  - Table statistics can be bad or stale
  - Join results can be massively over or under estimated
  - Error can multiply for additional successive joins
  - Bad estimates can lead to choosing the wrong join method

Introduction
Optimizer Hints

credativ

- Optimizer hints allow DBAs to issue commands to the planner
  - “Use this join ordering with that join method!”
  - “Use this index for that table!”
- Many enterprise databases provide optimizer hints
  - Oracle Database
  - Microsoft SQL Server
  - IBM Db2
  - EDB Advanced Server
- Implementations differ

Introduction
Optimizer Hints and Postgres

credativ

- Postgres traditionally does not provide optimizer hints
- `https://wiki.postgresql.org/wiki/OptimizerHintsDiscussion`

```
We are not interested in implementing hints in the exact ways they are commonly
implemented on other databases. Proposals based on "because they've got them"
will not be welcomed. If you have an idea that avoids the problems that have
been observed with other hint systems, that could lead to valuable discussion
```

- Optimizer hints critique
  - Costing issues should be fixed in the code, not worked-around via hints
  - Administration is difficult and problematic to apply to application queries
  - Do not adapt to data changes
  - No advantages when behaviour improves after major version upgrades

Introduction
Optimizer Hints vs. Plan-Flips

credativ

- Optimizer hints might be problematic
- Plan-Flips can be catastrophic
  - Small changes to costs can lead to (massively) different plans
  - Execution time can change (increase) drastically
- Plan-flips after major upgrades a particular risk
  - Plans of important queries should be checked beforehand
- Optimizer hints can pin plans and stabilize queries
  - Better to leave 10-20% of performance on the table, than lose 1000%

Introduction
Test Database

credativ

- Open Media Database (OMDB) used for the following examples
  - `https://www.omdb.org`
  - Crowd-sourced IMDB alternative
  - Creative-Commons license
- Postgres schema: `https://github.com/credativ/omdb-postgresql`
  - Ca. 430 MB
  - Ca. 300k `movies`, 300k `people`, 1.25M `casts`

```
$ createdb omdb
$ pg_restore -d omdb -O omdb.dump
```

credativ

Planner Node Overrides

Planner Node Overrides
Overview

credativ

- Postgres users have always been able to deactivate specific planner node types
  - `enable_seqscan = on/off`
  - `enable_indexscan = on/off`
  - `enable_bitmapscan = on/off`
  - `enable_mergejoin = on/off`
  - `enable_nestloop = on/off`
  - `enable_hashjoin = on/off`
  - …
- Planner node get deactivated globally for the plan
  - `SET LOCAL enable_[...]` to set until transaction ends
- Normally only useful for debugging
  - What are the costs if no sequential scans would be used?
- Needs to be set/adjusted for each query

Planner Node Overrides
Example

credativ

```
omdb=# SET max_parallel_workers_per_gather = 0; SET jit TO off;
SET
omdb=# EXPLAIN SELECT name FROM movies WHERE id = 123;
                                QUERY PLAN
-------------------------------------------------------------------------------
 Index Scan using movies_pkey on movies  (cost=0.42..8.44 rows=1 width=17)
   Index Cond: (id = 123)

omdb=# SET enable_indexscan TO off; SET enable_bitmapscan TO off;
SET
omdb=# EXPLAIN SELECT name FROM movies WHERE id = 123;
                         QUERY PLAN
------------------------------------------------------------
 Seq Scan on movies  (cost=0.00..6902.76 rows=1 width=17)
   Filter: (id = 123)
```

Planner Node Overrides
Implementation

- `enable_* = off` parameter adds 10M to costs until PG17
  - In case there are no alternatives, planner node is still anyway
  - Problematic for plans with high costs

```
omdb=# EXPLAIN SELECT COUNT(*) FROM movies;
                              QUERY PLAN
-----------------------------------------------------------------------
 Aggregate  (cost=6902.76..6902.77 rows=1 width=8)
   ->  Seq Scan on movies  (cost=0.00..6195.81 rows=282781 width=0)
omdb=# SET enable_seqscan TO off; SET enable_indexscan TO off; SET enable_bitmapscan TO off;
SET
omdb=# EXPLAIN SELECT COUNT(*) FROM movies;
                                      QUERY PLAN
-------------------------------------------------------------------------------------------
 Aggregate  (cost=10000006902.76..10000006902.77 rows=1 width=8)
   ->  Seq Scan on movies  (cost=10000000000.00..10000006195.81 rows=282781 width=0)
```

Planner Node Overrides
Implementation

credativ

- Costs are no longer adjusted since version 18
- Planner counts the number of suppressed nodes
- Deactivated planner node types are initially fully ignored
- `Disabled: true` added to explain output if node is utilized anyway

Planner Node Overrides
Implementation

```
omdb=# SET enable_seqscan TO off;
SET
omdb=# EXPLAIN SELECT COUNT(*) FROM movies;
                                        QUERY PLAN
-------------------------------------------------------------------------------------------------
 Aggregate  (cost=8227.65..8227.66 rows=1 width=8)
   ->  Index Only Scan using movies_pkey on movies  (cost=0.42..7520.70 rows=282781 width=0)
omdb=# SET enable_indexonlyscan TO off;
SET
omdb=# EXPLAIN SELECT COUNT(*) FROM movies;
                             QUERY PLAN
--------------------------------------------------------------------
 Aggregate  (cost=6902.76..6902.77 rows=1 width=8)
   ->  Seq Scan on movies  (cost=0.00..6195.81 rows=282781 width=0)
         Disabled: true
```

Planner Node Overrides
Additional Parameters

credativ

- Cost parameter can be adjusted
  - `random_page_cost`
  - `seq_page_cost`
  - `cpu_index_tuple_cost`
  - `cpu_operator_cost`
  - `cpu_tuple_cost`
- Join-reordering an be switched off
  - `join_collapse_limit = 1`

Planner Node Overrides

More Complex Query

```
omdb=# EXPLAIN SELECT DISTINCT m.name FROM movies m JOIN casts c ON m.id = c.movie_id
JOIN people p ON p.id = c.person_id WHERE p.name = 'Quentin Tarantino';
                                        QUERY PLAN
-------------------------------------------------------------------------------------------------------
 Unique  (cost=32305.36..32305.38 rows=4 width=17)
   ->  Sort  (cost=32305.36..32305.37 rows=4 width=17)
         Sort Key: m.name
         ->  Nested Loop  (cost=5867.11..32305.32 rows=4 width=17)
               ->  Hash Join  (cost=5866.69..32303.51 rows=4 width=8)
                     Hash Cond: (c.person_id = p.id)
                     ->  Seq Scan on casts c  (cost=0.00..23137.33 rows=1256933 width=16)
                     ->  Hash  (cost=5866.68..5866.68 rows=1 width=8)
                           ->  Seq Scan on people p  (cost=0.00..5866.68 rows=1 width=8)
                                 Filter: (name = 'Quentin Tarantino'::text)
               ->  Index Scan using movies_pkey on movies m  (cost=0.42..0.45 rows=1 width=25)
                     Index Cond: (id = c.movie_id)
```

- ?

credativ

External Extension

pg_hint_plan
Introduction

credativ

- `https://github.com/ossc-db/pg_hint_plan`
- BSD license, maintained by Michael Paquier
- Available at all three of the leading managed Postgres providers
  - Amazon RDS
  - Microsoft Azure Flexible Server
  - Google Cloud SQL
- Included in PGDG community repositories (`{apt,yum,zypp}.postgresql.org`)
- Mature and established extension, duplicates large parts of planner code though
- Documentation: `https://pg-hint-plan.readthedocs.io/en/latest/`

pg_hint_plan
Introduction

credativ

- Allows optimizer hints in two ways:
  - Inline SQL comments with `/*+ <hint> */` markup
  - `hints` table with deposited hints

pg_hint_plan
Inline SQL Comment Example

```
omdb=# EXPLAIN SELECT name FROM movies WHERE id = 123;
                                QUERY PLAN
-------------------------------------------------------------------------------
 Index Scan using movies_pkey on movies  (cost=0.42..8.44 rows=1 width=17)
   Index Cond: (id = 123)

omdb=# LOAD 'pg_hint_plan';
LOAD
omdb=# EXPLAIN /*+ SeqScan(movies) */ SELECT name FROM movies WHERE id = 123;
                         QUERY PLAN
-------------------------------------------------------------
 Seq Scan on movies  (cost=0.00..6902.76 rows=1 width=17)
   Filter: (id = 123)
```

pg_hint_plan
Inline SQL Comment Example

credativ

```
omdb=# EXPLAIN SELECT COUNT(*) FROM movies;
                            QUERY PLAN
------------------------------------------------------------------
 Aggregate  (cost=6860.76..6860.77 rows=1 width=8)
   ->  Seq Scan on movies  (cost=0.00..6153.81 rows=282781 width=0)
(2 rows)

omdb=# EXPLAIN /*+ IndexOnlyScan(movies) */ SELECT COUNT(*) FROM movies;
                                      QUERY PLAN
---------------------------------------------------------------------------------------
 Aggregate  (cost=8061.09..8061.10 rows=1 width=8)
   ->  Index Only Scan using movies_pkey on movies  (cost=0.42..7354.14 rows=282781 width=0)
(2 rows)
```

pg_hint_plan
hints Table

credativ

- `hint_plan.hints` table
  - `pg_hint_plan` extensions needs to be created in database
- `queryid` since version 17, `norm_query_string` before
- Parameter `pg_hint_plan.enable_hint_table` must be activated

```
omdb=# CREATE EXTENSION pg_hint_plan;
CREATE EXTENSION
omdb=# SET pg_hint_plan.enable_hint_table TO on;
SET
omdb=# \d hint_plan.hints
                              Table "hint_plan.hints"
      Column      |  Type   | Collation | Nullable |             Default
------------------+---------+-----------+----------+---------------------------------
 id               | integer |           | not null | generated by default as identity
 query_id         | bigint  |           | not null |
 application_name | text    |           | not null |
 hints            | text    |           | not null |
```

pg_hint_plan
hints Table

credativ

- Query is identified via query ID, e.g. from
  - `pg_stat_statements`
  - `EXPLAIN (VERBOSE)`
  - `compute_query_id` might need to be set to `on` (instead of `auto`)

```
omdb=# EXPLAIN (VERBOSE, COSTS OFF) SELECT COUNT(*) FROM movies;
                              QUERY PLAN
-----------------------------------------------------------------------------
 Aggregate
   Output: count(*)
   ->  Seq Scan on public.movies
         Output: id, name, parent_id, date, series_id, kind, runtime, [...]
 Query Identifier: 2247493858800491864
```

pg_hint_plan
hints Table

credativ

- Query ID and hint are added to `hints` table
- Empty string for `application_name` implies arbitrary `application_name`

```
omdb=# INSERT INTO hint_plan.hints (query_id, application_name, hints)
VALUES (2247493858800491864, '', 'IndexOnlyScan(movies)');
INSERT 0 1
omdb=# EXPLAIN (COSTS OFF) SELECT COUNT(*) FROM movies;
                    QUERY PLAN
--------------------------------------------------
 Aggregate
   ->  Index Only Scan using movies_pkey on movies
```

pg_hint_plan
hints Table

credativ

- Hints in `hints` table have priority over hint comments

```
omdb=# SELECT * FROM hint_plan.hints;
 id |      query_id       | application_name |         hints
----+---------------------+------------------+-----------------------
  1 | 2247493858800491864 |                  | IndexOnlyScan(movies)
omdb=# EXPLAIN (COSTS OFF) /*+ SeqScan(movies) */ SELECT COUNT(*) FROM movies;
                    QUERY PLAN
---------------------------------------------------
 Aggregate
   ->  Index Only Scan using movies_pkey on movies
```

pg_hint_plan Hints
Force Sequential/Index Scan

credativ

- SeqScan(table)
- IndexScan(table [index])

```
omdb=# EXPLAIN SELECT name FROM movies WHERE date > '2000-01-01';
                           QUERY PLAN
-----------------------------------------------------------------
 Seq Scan on movies  (cost=0.00..6860.76 rows=190520 width=17)
   Filter: (date > '2000-01-01'::date)
omdb=# EXPLAIN /*+ IndexScan(movies movies_date_idx) */ SELECT name FROM movies
WHERE date > '2000-01-01';
                                    QUERY PLAN
-----------------------------------------------------------------------------------------
 Index Scan using movies_date_idx on movies  (cost=0.42..17440.06 rows=190520 width=17)
   Index Cond: (date > '2000-01-01'::date)
```

pg_hint_plan Hints
Suppress Sequential/Index Scan

credativ

- NoSeqScan(table)

```
omdb=# EXPLAIN /*+ NoSeqScan(movies) */ SELECT name FROM movies WHERE date > '2000-01-01';
                                      QUERY PLAN
-------------------------------------------------------------------------------------
 Bitmap Heap Scan on movies  (cost=2336.95..8044.45 rows=190520 width=17)
   Recheck Cond: (date > '2000-01-01'::date)
   ->  Bitmap Index Scan on movies_date_idx  (cost=0.00..2289.32 rows=190520 width=0)
         Index Cond: (date > '2000-01-01'::date)
```

- DisableIndex(table index) # PG18+

```
omdb=# EXPLAIN /*+ DisableIndex(movies movies_pkey) */ SELECT name FROM movies WHERE id = 123;
                        QUERY PLAN
-----------------------------------------------------------
 Seq Scan on movies  (cost=0.00..6902.76 rows=1 width=17)
   Filter: (id = 123)
```

pg_hint_plan Hints
Force Join Type

credativ

- `NestLoop(table table [table])`

```
omdb=# EXPLAIN /*+ NestLoop(p c) */ SELECT DISTINCT m.name FROM casts c JOIN movies m
ON m.id = c.movie_id JOIN people p ON p.id = c.person_id WHERE p.name = 'Quentin Tarantino';
                                        QUERY PLAN
-------------------------------------------------------------------------------------------------------
 Unique  (cost=44717.52..44717.54 rows=4 width=17)
   ->  Sort  (cost=44717.52..44717.53 rows=4 width=17)
         Sort Key: m.name
         ->  Nested Loop  (cost=0.42..44717.48 rows=4 width=17)
               ->  Nested Loop  (cost=0.00..44715.67 rows=4 width=8)
                     Join Filter: (c.person_id = p.id)
                     ->  Seq Scan on people p  (cost=0.00..5866.68 rows=1 width=8)
                           Filter: (name = 'Quentin Tarantino'::text)
                     ->  Seq Scan on casts c  (cost=0.00..23137.33 rows=1256933 width=16)
               ->  Index Scan using movies_pkey on movies m  (cost=0.42..0.45 rows=1 width=25)
                     Index Cond: (id = c.movie_id)
```

pg_hint_plan Hints
Force Join Type

credativ

- `NestLoop(table table [table])`
- `HashJoin(table table [table])`
- `MergeJoin(table table [table])`
- `NoNestLoop(table table [table])`
- `NoHashJoin(table table [table])`
- `NoMergeJoin(table table [table])`

pg_hint_plan Hints
Force Join Ordering

credativ

- Leading(table table [table])

```
omdb=# EXPLAIN /*+ Leading(m c p) */ SELECT DISTINCT m.name FROM casts c JOIN movies m
ON m.id = c.movie_id JOIN people p ON p.id = c.person_id WHERE p.name = 'Quentin Tarantino';
                                        QUERY PLAN
-------------------------------------------------------------------------------------------------------
 Unique  (cost=61477.62..61477.64 rows=4 width=17)
   ->  Sort  (cost=61477.62..61477.63 rows=4 width=17)
         Sort Key: m.name
         ->  Hash Join  (cost=17531.26..61477.58 rows=4 width=17)
               Hash Cond: (c.person_id = p.id)
               ->  Hash Join  (cost=11664.57..52311.40 rows=1256933 width=25)
                     Hash Cond: (c.movie_id = m.id)
                     ->  Seq Scan on casts c  (cost=0.00..23137.33 rows=1256933 width=16)
                     ->  Hash  (cost=6195.81..6195.81 rows=282781 width=25)
                           ->  Seq Scan on movies m  (cost=0.00..6195.81 rows=282781 width=25)
               ->  Hash  (cost=5866.68..5866.68 rows=1 width=8)
                     ->  Seq Scan on people p  (cost=0.00..5866.68 rows=1 width=8)
                           Filter: (name = 'Quentin Tarantino'::text)
```

pg_hint_plan Hints
Correct Join Result

credativ

- Rows(table table correction)

```
omdb=# EXPLAIN /*+ Leading(m c p) Rows(c p *10) */ SELECT DISTINCT m.name FROM casts c JOIN movies m
ON m.id = c.movie_id JOIN people p ON p.id = c.person_id WHERE p.name = 'Quentin Tarantino';
                                        QUERY PLAN
-------------------------------------------------------------------------------------------------------
 Unique  (cost=61478.64..61478.84 rows=40 width=17)
   ->  Sort  (cost=61478.64..61478.74 rows=40 width=17)
         Sort Key: m.name
         ->  Hash Join  (cost=17531.26..61477.58 rows=40 width=17)
               Hash Cond: (c.person_id = p.id)
               ->  Hash Join  (cost=11664.57..52311.40 rows=1256933 width=25)
                     Hash Cond: (c.movie_id = m.id)
                     ->  Seq Scan on casts c  (cost=0.00..23137.33 rows=1256933 width=16)
                     ->  Hash  (cost=6195.81..6195.81 rows=282781 width=25)
                           ->  Seq Scan on movies m  (cost=0.00..6195.81 rows=282781 width=25)
               ->  Hash  (cost=5866.68..5866.68 rows=1 width=8)
                     ->  Seq Scan on people p  (cost=0.00..5866.68 rows=1 width=8)
                           Filter: (name = 'Quentin Tarantino'::text)
```

credativ

Internal Plan Advisor

Internal Plan Advisor
Introduction

credativ

- PostgreSQL 19 will (probably) include two new extensions
  - `pg_plan_advice`
  - `pg_stash_advice`
- Mainly targeted at plan stabilization
  - Can be (ab)used as optimizer hints as well
- Implemented by removal of unwanted plan nodes
  - Based on new `enable_[...] = off` implementation in PG18
  - Can deactivate each individual plan node, not just globally
- Desired (piece of) plan is kept
  - Via deactivation of all other possible planner node options/combinations

pg_plan_advice
Example

credativ

- Two components
  - `EXPLAIN` option that outputs plan advice
  - Advice tags similar to `pg_hint_plan` syntax
- `EXPLAIN (PLAN_ADVICE)` option outputs generated plan advice

```
omdb=# LOAD 'pg_plan_advice';
LOAD
omdb=# EXPLAIN (COSTS OFF, PLAN_ADVICE) SELECT name FROM movies WHERE id = 123;
               QUERY PLAN
----------------------------------------
 Index Scan using movies_pkey on movies
   Index Cond: (id = 123)
 Generated Plan Advice:
   INDEX_SCAN(movies public.movies_pkey)
   NO_GATHER(movies)
```

pg_plan_advice
Example

credativ

```
omdb=# EXPLAIN (PLAN_ADVICE, COSTS OFF) SELECT DISTINCT m.name FROM casts c JOIN movies m
ON m.id = c.movie_id JOIN people p ON p.id = c.person_id WHERE p.name = 'Quentin Tarantino';
                                QUERY PLAN
--------------------------------------------------------------------------------
 Unique
   ->  Sort
         Sort Key: m.name
         ->  Nested Loop
               ->  Hash Join
                     Hash Cond: (c.person_id = p.id)
                     ->  Seq Scan on casts c
                     ->  Hash
                           ->  Seq Scan on people p
                                 Filter: (name = 'Quentin Tarantino'::text)
               ->  Index Scan using movies_pkey on movies m
                     Index Cond: (id = c.movie_id)
 Generated Plan Advice:
   JOIN_ORDER(c p m) NESTED_LOOP_PLAIN(m) HASH_JOIN(p) SEQ_SCAN(c p)
   INDEX_SCAN(m public.movies_pkey) NO_GATHER(c m p)
```

pg_plan_advice
Example

credativ

- Plan advice can be set via `pg_plan_advice.advice` parameter

```
omdb=# SET pg_plan_advice.advice = 'SEQ_SCAN(movies)';
SET
omdb=# EXPLAIN (COSTS OFF) SELECT name FROM movies WHERE id = 123;
            QUERY PLAN
----------------------------------
 Seq Scan on movies
   Filter: (id = 123)
 Supplied Plan Advice:
   SEQ_SCAN(movies) /* matched */
```

pg_plan_advice
Advice Feedback

credativ

- `EXPLAIN` shows the used advice
  - `/* matched */` - advice was applied
  - `/* partially matched */` - advice was partially applied
  - `/* not matched */` - advice cannot be applied to query
- Other error messages
  - `/* inapplicable */` - advice cannot be applied to query
  - `/* failed */` - advice cannot be applied to query
  - `/* conflicting */` - conflicting advice, this advice was ignored

Advice Tags
Force Scan-Method

- SEQ_SCAN(table)
- INDEX_SCAN(table index)
- INDEX_ONLY_SCAN(table index)
- BITMAP_HEAP_SCAN(table index)

```
omdb=# SET pg_plan_advice.advice = 'INDEX_SCAN(movies movies_date_idx)';
SET
omdb=# EXPLAIN (COSTS OFF) SELECT name FROM movies WHERE date > '2000-01-01';
                       QUERY PLAN
--------------------------------------------------------
 Index Scan using movies_date_idx on movies
   Index Cond: (date > '2000-01-01'::date)
 Supplied Plan Advice:
   INDEX_SCAN(movies movies_date_idx) /* matched */
```

Advice Tags
Force Join Type

credativ

- `NESTED_LOOP_PLAIN(table table)`
- `NESTED_LOOP_MATERIALIZE(table table)`
- `NESTED_LOOP_MEMOIZE(table table)`
- `HASH_JOIN(table table)`
- `MERGE_JOIN_PLAIN(table table)`
- `MERGE_JOIN_MATERIALIZE(table table)`

Advice Tags
Force Join Type

credativ

```
omdb=# SET pg_plan_advice.advice = 'NESTED_LOOP_PLAIN(p c)';
omdb=# EXPLAIN (COSTS OFF) SELECT DISTINCT m.name FROM casts c JOIN movies m
ON m.id = c.movie_id JOIN people p ON p.id = c.person_id WHERE p.name = 'Quentin Tarantino';
                              QUERY PLAN
-----------------------------------------------------------------------
 Unique
   ->  Sort
         Sort Key: m.name
         ->  Nested Loop
               ->  Nested Loop
                     ->  Seq Scan on movies m
                     ->  Index Only Scan using casts_pkey on casts c
                           Index Cond: (movie_id = m.id)
               ->  Index Scan using people_pkey on people p
                     Index Cond: (id = c.person_id)
                     Filter: (name = 'Quentin Tarantino'::text)
 Supplied Plan Advice:
   NESTED_LOOP_PLAIN(p) /* matched */
   NESTED_LOOP_PLAIN(c) /* matched */
```

Advice Tags
Force Join Ordering

credativ

- JOIN_ORDER(table (table table) table)

```
omdb=# SET pg_plan_advice.advice = 'JOIN_ORDER(m c p)';
omdb=# EXPLAIN (COSTS OFF) SELECT DISTINCT m.name FROM casts c JOIN movies m
ON m.id = c.movie_id JOIN people p ON p.id = c.person_id WHERE p.name = 'Quentin Tarantino';
                              QUERY PLAN
------------------------------------------------------------------------
 Unique
   ->  Sort
         ->  Hash Join
               Hash Cond: (c.person_id = p.id)
               ->  Merge Join
                     ->  Index Scan using movies_pkey on movies m
                     ->  Index Only Scan using casts_pkey on casts c
               ->  Hash
                     ->  Seq Scan on people p
 Supplied Plan Advice:
   JOIN_ORDER(m c p) /* matched */
```

pg_stash_advice
Overview

credativ

- Central storage of plan advice for queries
- Several stashes possible in parallel, stashed are identified by their names
- Persistence of plan advice through
  - Client reconnects
    - `LOAD 'pg_stash_advice'` required
  - Server restarts
    - Addition to `shared_preload_libraries` required
    - Background worker persists stashes every 30s

pg_stash_advice
Setup

- Stash gets initialized via `pg_create_advice_stash(<stash>)` function
- Parameter `pg_stash_advice.stash_name` selects stash

```
omdb=# RESET pg_plan_advice.advice;
RESET
omdb=# CREATE EXTENSION pg_stash_advice;
CREATE EXTENSION
omdb=# SELECT pg_create_advice_stash('stash');
  pg_create_advice_stash
 ------------------------
omdb=# SET pg_stash_advice.stash_name = 'stash';
SET
```

pg_stash_advice
Setting Plan Stashes

credativ

- Query gets identified query ID
- `pg_set_stashed_advice(<stash>, <queryid>, <plan_advice>)` sets plan advice

```
omdb=# EXPLAIN (VERBOSE, COSTS OFF) SELECT name FROM movies WHERE id = 123;
                    QUERY PLAN
-------------------------------------------------
 Index Scan using movies_pkey on public.movies
   Output: name
   Index Cond: (movies.id = 123)
 Query Identifier: 194340523991197898
omdb=# SELECT pg_set_stashed_advice('stash', '194340523991197898', 'SEQ_SCAN(movies)');
  pg_set_stashed_advice
 -----------------------
```

pg_stash_advice
Plan-Stash Usage

credativ

```
omdb=# LOAD 'pg_stash_advice';
LOAD
omdb=# SET pg_stash_advice.stash_name = 'stash';
SET
omdb=# EXPLAIN (VERBOSE, COSTS OFF) SELECT name FROM movies WHERE id = 123;
              QUERY PLAN
---------------------------------------
 Seq Scan on public.movies
   Output: name
   Filter: (movies.id = 123)
 Query Identifier: 194340523991197898
 Supplied Plan Advice:
   SEQ_SCAN(movies) /* matched */
```

pg_stash_advice
Plan-Stash Usage

credativ

- Plan stash usable for application roles via per-role/database settings

```
omdb=# CREATE USER app WITH PASSWORD 'app';
CREATE ROLE
omdb=# ALTER USER app SET pg_stash_advice.stash_name = 'stash';
ALTER ROLE
omdb=# \c "user=app password=app dbname=omdb"
You are now connected to database "omdb" as user "app".
omdb=> EXPLAIN (COSTS OFF) SELECT name FROM movies WHERE id = 123;
            QUERY PLAN
----------------------------------
 Seq Scan on movies
   Filter: (id = 123)
 Supplied Plan Advice:
   SEQ_SCAN(movies) /* matched */
```

Summary

Summary

credativ

- Postgres 19 will prove native plan pinning/optimizer hints for the fist time
  - `pg_plan_advice`/`pg_stash_plan` extensions
- External `pg_hint_plan` extension available in earlier versions
  - Infrastructure changes in PG19 are helpful for `pg_hint_plan` as well
  - ossc-db/pg_hint_plan#234 (`+1.577, −3.982` code lines)
- Optimizer plans can get very complicated
- `EXPLAIN (PLAN_ADVICE)` enables output of plan tags and provides advice feedback

credativ

Questions?