credativ

Optimizer Hints in PostgreSQL

Michael Banck <michael.banck@credativ.de>

Swiss PGDay 2026

credativ GmbH

Open Source for Business

- Gegründet 1999 in Mönchengladbach
- Enge Verbindung zur Open-Source Community
- 40 Open-Source Spezialisten
- Beratung, Entwicklung, Schulung, Support (3rd-Level / 24x7)
- Open-Source Infrastruktur mit Debian, Kubernetes und Proxmox
- Open-Source Datenbanken mit PostgreSQL
- Open-Source DevOps mit Ansible, Puppet, Terraform und anderen
- Seit 2025 wieder unabhängiges, Inhaber-geführtes Unternehmen

Optimizer Hints
Agenda

- Einführung
- Planknoten-Overrides
- Externe Erweiterung
- Interner Plan Advisor

Einführung

Einführung
Planer-Übersicht

credativ

- Postgres hat einen Kosten-basierten Query-Planer
- Für Planknoten werden Anfangs- und Endkosten berechnet
- Der Planer probiert alle verschiedenen Möglichkeiten aus
- Der Plan mit den geringsten Gesamtkosten wird ausgeführt
- Das ist nicht unbedingt der Plan mit der geringsten Laufzeit
  - Tabellen-Statistiken können schlecht oder veraltet sein
  - Join-Ergebnisse können massiv über- oder unterschätzt werden
  - Bei mehreren Joins kann sich der Fehler multiplizieren
  - Schlechte Schätzungen können zur Wahl einer falschen Join-Methode führen

Einführung
Optimizer Hints

credativ

- Optimizer Hints erlauben dem DBA dem Planer Befehle zu erteilen
  - “Verwende diese Join-Reihenfolge mit dieser Join-Methode!”
  - “Verwende für diese Tabelle diesen Index!”
- Viele Enterprise-Datenbanken bieten Optimizer Hints
  - Oracle Database
  - Microsoft SQL Server
  - IBM Db2
  - EDB Advanced Server
- Implementierung unterschiedlich

Einführung
Optimizer Hints und Postgres

credativ

- Postgres bietet traditionell keine 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
```

- Kritik an Optimizer Hints
  - Kosten-Probleme sollten im Code behoben werden, nicht via Hints umgangen
  - Kompliziert zu verwalten und problematisch bei Anwendungs-Queries
  - Passen sich nicht Datenänderungen an
  - Kein Vorteil wenn sich das Verhalten nach Major-Upgrades verbesesrt

Einführung
Optimizer Hints vs. Plan-Flips

credativ

- Optimizer Hints sind vielleicht problematisch
- Aber Plan-Flips können katastrophal sein
  - Kleine Änderungen an den Kosten führen zu (massiv) anderen Plänen
  - Ausführungszeit kann stark (nach oben) abweichen
- Plan-Flips vor allem auch bei Major Upgrades möglich
  - Pläne von wichtigen Queries sollten getestet werden
- Optimizer Hints können den Plan einer Query Pinnen und Stabilisieren
  - Besser immer 10-20% weniger Performance als manchmal 1000% weniger

Einführung
Testdatenbank

credativ

- Für die folgenden Beispiele wird die Open Media Database (OMDB) verwendet
  - `https://www.omdb.org`
  - Crowdsourced IMDB Alternative
  - Creative-Commons Lizenz
- 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

Planknoten-Overrides

Planknoten-Overrides
Überblick

credativ

- Postgres-Nutzer konnten schon immer einzelne Planknoten-Typen deaktivieren
  - `enable_seqscan = on/off`
  - `enable_indexscan = on/off`
  - `enable_bitmapscan = on/off`
  - `enable_mergejoin = on/off`
  - `enable_nestloop = on/off`
  - `enable_hashjoin = on/off`
  - …
- Planknoten werden global für den Plan deaktiviert
  - `SET LOCAL enable_[...]` für Einstellungen bis zum Transaktions-Ende
- Normalerweise nur zu Debugging-Zwecken sinnvoll
  - Wie hoch sind die Kosten wenn kein Sequential Scan verwendet werden würde?
- Muss potenziell für jede Query angepasst werden

Planknoten-Overrides

Beispiel

```
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)
```

Planknoten-Overrides
Implementierung

credativ

- `enable_* = off` Parameter fügen bis PG17 10M zu Kosten hinzu
  - Wenn es nicht anders geht wird Planknoten trotzdem verwendet
  - Problematisch bei Plänen mit hohen Kosten

```
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)
```

Planknoten-Overrides
Implementierung

credativ

- Seit Version 18 werden die Kosten nicht mehr verändert
- Planer merkt sich die Anzahl der unterdrückten Planknoten
- Plankonten die deaktiviert sind werden zunächst ignoriert
- `Disabled: true` wird zur Explain-Ausgabe hinzugefügt, wenn Plankonten doch verwendet wird

Planknoten-Overrides
Implementierung

```
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
```

Planknoten-Overrides

Weitere Parameter

credativ

- Kosten-Parameter können angepasst werden
  - `random_page_cost`
  - `seq_page_cost`
  - `cpu_index_tuple_cost`
  - `cpu_operator_cost`
  - `cpu_tuple_cost`
- Join-Umsortierung kann verboten werden
  - `join_collapse_limit = 1`

Planknoten-Overrides

Komplexere Abfrage

credativ

```
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

Externe Erweiterung

pg_hint_plan
Einführung

credativ

- `https://github.com/ossc-db/pg_hint_plan`
- BSD-Lizenz, Maintainer ist Michael Paquier
- Steht bei allen großen Managed-Postgres Providern zur Verfügung
  - Amazon RDS
  - Microsoft Azure Flexible Server
  - Google Cloud SQL
- Verfügbar in PGDG Community-Repositories (`{apt,yum,zypp}.postgresql.org`)
- Ausgereifte Erweiterung, dupliziert allerdings größere Teile des Planer-Codes
- Dokumentation: `https://pg-hint-plan.readthedocs.io/en/latest/`

pg_hint_plan
Einführung

credativ

- Ermöglicht Optimizer Hints auf zwei Arten:
  - Inline SQL-Kommentar mit `/*+ <hint> */` Markup
  - `hints`-Tabelle mit hinterlegten Hints

pg_hint_plan
Inline SQL-Kommentar Beispiele

credativ

```
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-Kommentar Beispiele

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-Tabelle

credativ

- `hint_plan.hints` Tabelle
  - Wenn `pg_hint_plan` Erweiterung in Datenbank erstellt wurde
- `queryid` seit Version 17, davor `norm_query_string`
- Parameter `pg_hint_plan.enable_hint_table` muss aktiv sein

```
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-Tabelle

credativ

- Query wird via Query-ID identifiziert, z.B. aus
  - `pg_stat_statements`
  - `EXPLAIN (VERBOSE)`
  - `compute_query_id` ggf. auf `on` (statt `auto`) setzen

```
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-Tabelle

credativ

- Query-ID und Hint werden in `hints`-Tabelle eingetragen
- Leerer String bei `application_name` bedeutet beliebiger `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-Tabelle

credativ

- Hints in `hints`-Tabelle haben Priorität über Hint-Kommentaren

```
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
Sequential/Index Scan Forcierung

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
Sequential/Index Scan Unterdrückung

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
Join-Typ Forcierung

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
Join-Typ Forcierung

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
Join-Reihenfolge Forcierung

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
Join-Ergebnis Korrektur

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)
```

Interner Plan Advisor

Interner Plan Advisor
Einführung

credativ

- PostgreSQL 19 wird (voraussichtlich) zwei neue Erweiterungen enthalten
  - `pg_plan_advice`
  - `pg_stash_advice`
- Hauptsächlich für Plan-Stabilisierung gedacht
  - Kann auch für Optimizer Hints verwendet werden
- Implementiert durch Entfernung von möglichen Planknoten
  - Basiert auf neuer `enable_[...] = off` Variante in PG18
  - Kann individuelle Planknoten deaktivieren, nicht nur global
- Gewünschter (Teil-)Plan wird erhalten
  - Durch Deaktivierung alle anderen möglichen Planknoten-Optionen

pg_plan_advice
Beispiel

credativ

- Zwei Komponenten
  - `EXPLAIN`-Option, die einen Plan-Advice ausgibt
  - Advice-Tags analog zu `pg_hint_plan`-Syntax
- `EXPLAIN (PLAN_ADVICE)`-Option gibt generierten Plan-Advice aus

```
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
Beispiel

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
Beispiel

credativ

- Plan-Advice kann via `pg_plan_advice.advice` Parameter gesetzt werden

```
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 zeigt den verwendeten Advice an
  - `/* matched */` - Advice wurde angewendet
  - `/* partially matched */` - Advice wurde teilweise angewendet
  - `/* not matched */` - Advice ist nicht für die Query anwendbar
- Sonstige Fehlermeldungen
  - `/* inapplicable */` - Advice ist nicht für die Query anwendbar
  - `/* failed */` - Advice ist nicht für die Query anwendbar
  - `/* conflicting */` - Widersprüchlicher Advice, dieser wurde nicht angewendet

Advice Tags
Scan-Methoden Forcierung

credativ

- 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
Join-Typ Forcierung

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
Join-Typ Forcierung

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
Join-Reihenfolge Forcierung

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
Überblick

credativ

- Zentrale Speicherung von Plan-Advice pro Query
- Mehrere parallele Stashes möglich, Stashes werden mit ihrem Namen identifiziert
- Persistierung der Plan-Advices über
  - Client-Reconnects
    - `LOAD 'pg_stash_advice'` nötig
  - Server-Restarts
    - Eintrag in `shared_preload_libraries` nötig
    - Background-Worker persistiert Stashes alle 30s

pg_stash_advice
Setup

- Stash wird mit `pg_create_advice_stash(<stash>)` Funktion initialisiert
- Parameter `pg_stash_advice.stash_name` wählt Stash aus

```
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
Plan-Stash Setzen

credativ

- Query wird via Query-ID identifiziert
- `pg_set_stashed_advice(<stash>, <queryid>, <plan_advice>)` setzt 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 Verwenden

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 Verwenden

credativ

- Plan-Stash via Nutzer/Datenbank-Setting auch für Anwendungsnutzer setzbar

```
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 */
```

Zusammenfassung

Zusammenfassung

credativ

- Postgres 19 wird zum ersten Mal nativ Plan-Pinning/Optimizer Hints erlauben
  - `pg_plan_advice`/`pg_stash_plan` Erweiterungen
- In vorherigen Versionen externe `pg_hint_plan` Erweiterung verfügbar
  - Infrastruktur-Änderungen in PG19 für `pg_hint_plan` hilfreich
  - ossc-db/pg_hint_plan#234 (`+1.577, −3.982` Code-Zeilen)
- Optimizer-Pläne können sehr kompliziert werden
- `EXPLAIN (PLAN_ADVICE)` erlaubt Anzeige von Plan-Tags und Plan-Pinning

Fragen?