アドバイザリー機能は本当に効果があるのか

JPUG PGCon 2025

山田 達朗 (Tatsuro Yamada)
NTT Open Source Software Center

2025/11/21

自己紹介

山田 達朗

- NTT Open Source Software Center
- PostgreSQLのサポート・研究開発・コミュニティ活動を担当
- Senior researcher

コミュニティ活動

Development

- PostgreSQL Contributor
- Oracle_fdw Committer
- pg_plan_advsr Author など

Talks

- PGCon, PGConf.Eu, PGConf.Asia,
  JPUG PGCon, COSCUP, PGConf.devなど

趣味

- EXPLAINコマンド
- 音楽
- よなよなビール
- 皇居ラン (New)

Agenda

1. 背景・目的
2. アドバイザリー機能とは
3. 代表的なユースケース
4. アドバイザリー機能の今後
5. Q&A

質問はこちらから投入

Google Forms
https://forms.gle/LiDCvAi88ZXvBK6EA

1. 背景・目的
2. アドバイザリー機能とは
3. 代表的なユースケース
4. アドバイザリー機能の今後

1. 背景・目的

背景

- アドバイザリー機能は設定・インデックスなどの推奨案を提案する機能
- 多くのDBMSで利用可能
- PostgreSQLでは外部の拡張機能やサービスとして提供

目的

- PostgreSQLで利用可能なアドバイザリー機能を知っていただく
  - メリット・留意点・効果測定結果を共有
- PostgreSQLのユースケースの拡大、コミュニティの発展に貢献

1. 背景・目的
2. アドバイザリー機能とは
3. 代表的なユースケース
4. アドバイザリー機能の今後

2. アドバイザリー機能とは

- ユーザに推奨案を提示する機能
- 種類は多岐にわたり、**幅広いユーザにメリット**

何が便利なのか

新人DBA/開発者に対して

■ アドバイスをたたき台として活用できる

経験者に対して

■ 自分の考えとアドバイスを比較し漏れや別案を発見できる

アドバイザリー機能の種類

主な種類と対応状況

No.	種類	商用DBMS	PostgreSQL
1	**コンフィグ**	✓	✓
2	**テーブル／パーティショニング**	✓	
3	**インデックス**	✓	✓
4	**マテビュー**	✓	
5	**SQLの書き方**	✓	
6	**実行計画**	✓	✓
7	**拡張統計**		✓

2024年時点の調査結果より

アドバイザリー機能を持つ商用DBMS

Oracle、SQL Serverなど主要製品で広く提供

No.	DBMS	機能名	対象	Note
1	**EDB Postgres Advanced Server**	**EDB Query Advisor**	**Index**	https://www.enterprisedb.com/docs/epas/latest/managing_performance/
2	**Google AlloyDB**	**index advisor**	**Index**	https://cloud.google.com/alloydb/docs/use-index-advisor
3	**Oracle Database Enterprise Edition**	**SQL Tuning advisor, SQL Access advisor**	**Object stats, Rewrite query, Index, Materialize View, Partitioning**	https://docs.oracle.com/cd/F19136_01/tgsql/sql-tuning-advisor.html#GUID-73AB38C1-A7F6-401E-9010-B4476E173673
4	**Microsoft SQL Server**	**Database tuning advisor**	**Index, View, Partitioning**	https://learn.microsoft.com/ja-jp/sql/relational-databases/performance/database-engine-tuning-advisor?view=sql-server-ver16
5	**Postgres Pro Enterprise**	**aqo**	**Execution Plan**	https://postgrespro.com/docs/postgrespro/16/aqo ※Applying a patch is needed

2024年時点の調査結果より

PGで利用可能なアドバイザリー機能は

■公式では提供していない

■すべて拡張機能/サービスとして提供 (有償/無償)

■商用DBMSと比較し種類は少ない状況

次の章で代表的なユースケース別に紹介します

1. 背景・目的
2. アドバイザリー機能とは
3. 代表的なユースケース
4. アドバイザリー機能の今後

3. 代表的なユースケース

1. コンフィグの値はどのように設定すべき？
2. 作成すべきインデックスは？
3. クエリ実行時間を短縮したい

3. 代表的なユースケース

1. **コンフィグの値はどのように設定すべき？**
2. 作成すべきインデックスは？
3. クエリ実行時間を短縮したい

コンフィグに関する課題とツールのメリット

コンフィグに関する課題

■DB構築時にパラメータ設定で悩む

■特に新規システムではワークロード予測が困難

ツールのメリット

■たたき台となるコンフィグを提案してくれる

コンフィグのアドバイザリー機能の一覧

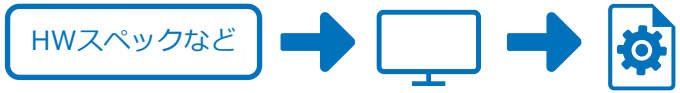
NTT

No.	Name	Author	Target	Note
1	PGTune	Greg Smith, Oleksii Vasyliev	Parameters	https://pgtune.leopard.in.ua/
2	PGConfig	Sebastian Webber	Parameters	https://www.pgconfig.org/
3	CYBERTEC pgconfigurator	Cybertec	Parameters	https://pgconfigurator.cybertec-postgresql.com/
4	**pgconfigurator**	Tantor Labs	Parameters	https://tantorlabs.ru/pgconfigurator
5	pgAnalyze	pgAnalyze	Parameters, Index, Vacuum	https://pganalyze.com/ ※有償
6	DBtune	DBtune	Parameters	https://www.dbtune.com/ ※有償

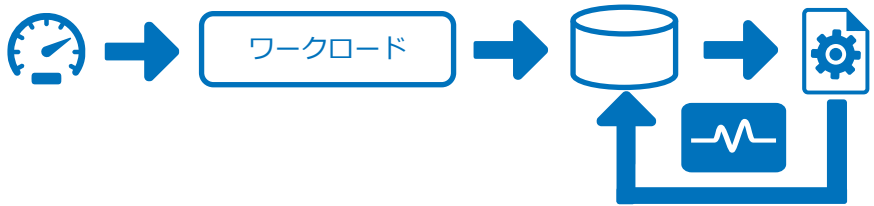
仕組みは？

大きく２種類

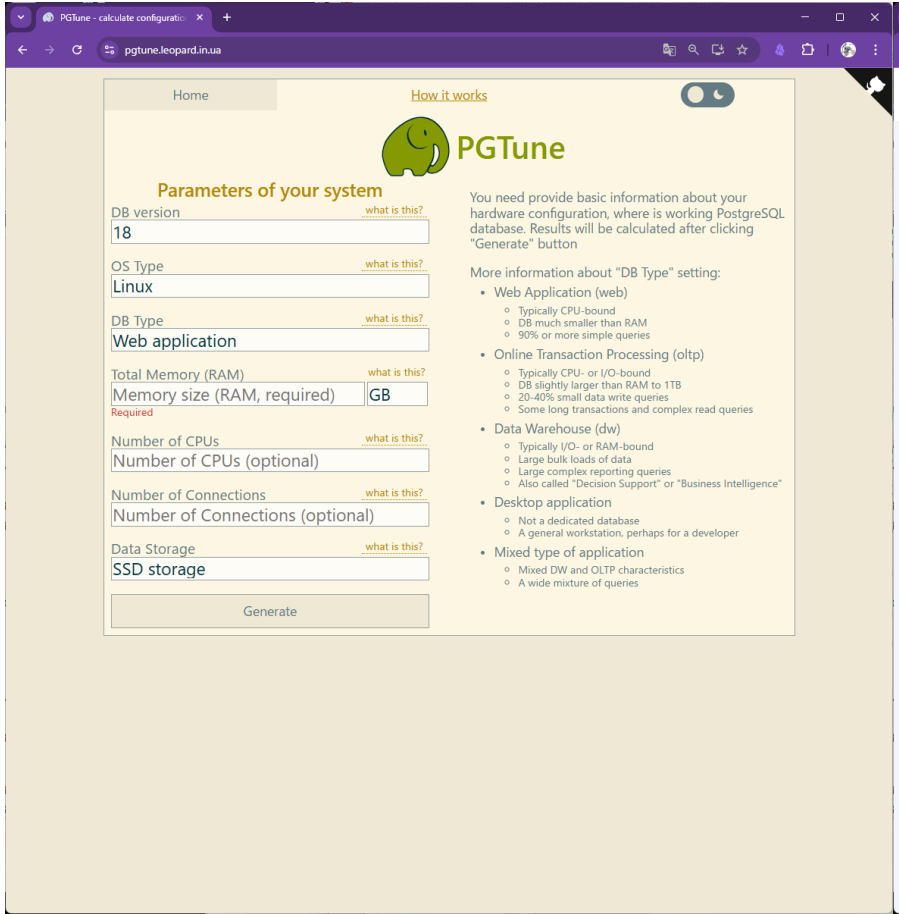
- H/Wスペックや想定ワークロードの情報をベースとするもの
  - **一般的な経験則に基づく提案**
  - PGTune、PGConfig、pgconfiguratorなど

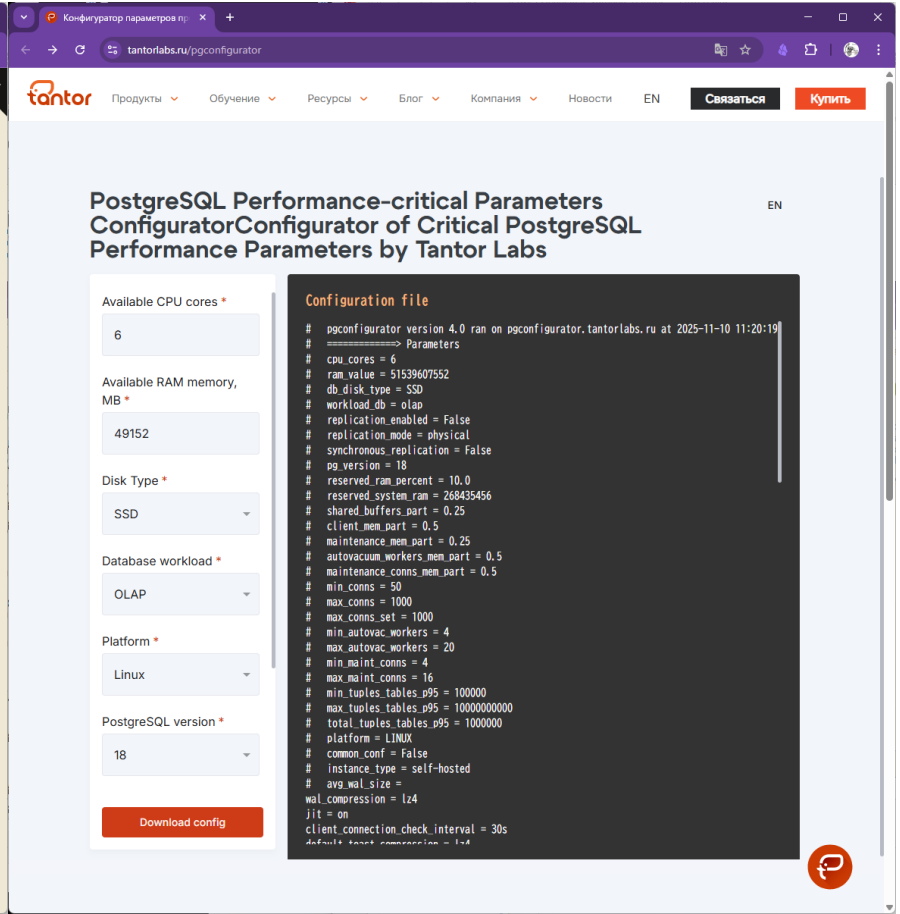


- 実際のワークロードを利用した機械学習的なアプローチのもの
  - **実際のワークロードをかけつつパラメータを変化させて最適化を図る**
  - DBTuneなど



画面のサンプル


PGTune - calculate configuratio
pgtune.leopard.in.ua
Home
How it works
PGTune
Parameters of your system
DB version
what is this?
18
OS Type
what is this?
Linux
DB Type
what is this?
Web application
Total Memory (RAM)
what is this?
Memory size (RAM, required)
GB
Required
Number of CPUs
what is this?
Number of CPUs (optional)
Number of Connections
what is this?
Number of Connections (optional)
Data Storage
what is this?
SSD storage
Generate
You need provide basic information about your hardware configuration, where is working PostgreSQL database. Results will be calculated after clicking "Generate" button
More information about "DB Type" setting:
Web Application (web)
Typically CPU-bound
DB much smaller than RAM
90% or more simple queries
Online Transaction Processing (oltp)
Typically CPU- or I/O-bound
DB slightly larger than RAM to 1TB
20-40% small data write queries
Some long transactions and complex read queries
Data Warehouse (dw)
Typically I/O- or RAM-bound
Large bulk loads of data
Large complex reporting queries
Also called "Decision Support" or "Business Intelligence"
Desktop application
Not a dedicated database
A general workstation, perhaps for a developer
Mixed type of application
Mixed DW and OLTP characteristics
A wide mixture of queries


Конфигуратор параметров пр
tantorlabs.ru/pgconfigurator
tantor
Продукты
Обучение
Ресурсы
Блог
Компания
Новости
EN
Связаться
Купить
PostgreSQL Performance-critical Parameters ConfiguratorConfigurator of Critical PostgreSQL Performance Parameters by Tantor Labs
EN
Available CPU cores *
6
Available RAM memory, MB *
49152
Disk Type *
SSD
Database workload *
OLAP
Platform *
Linux
PostgreSQL version *
18
Download config
Configuration file
pgconfigurator version 4.0 ran on pgconfigurator.tantorlabs.ru at 2025-11-10 11:20:19
# ==========> Parameters
# cpu_cores = 6
# ram_value = 51539607552
# db_disk_type = SSD
# workload_db = olap
# replication_enabled = False
# replication_mode = physical
# synchronous_replication = False
# pg_version = 18
# reserved_ram_percent = 10.0
# reserved_system_ram = 268435456
# shared_buffers_part = 0.25
# client_mem_part = 0.5
# maintenance_mem_part = 0.25
# autovacuum_workers_mem_part = 0.5
# maintenance_conns_mem_part = 0.5
# min_conns = 50
# max_conns = 1000
# max_conns_set = 1000
# min_autovac_workers = 4
# max_autovac_workers = 20
# min_maint_conns = 4
# max_maint_conns = 16
# min_tuples_tables_p95 = 100000
# max_tuples_tables_p95 = 10000000000
# total_tuples_tables_p95 = 1000000
# platform = LINUX
# common_conf = False
# instance_type = self-hosted
# avg_wal_size =
wal_compression = lz4
jit = on
client_connection_check_interval = 30s

効果を確認してみる: pgconfigurator

NTT

測定条件

- HammerDB TPC-H (22クエリ)、Scale Factor: 1 (1GB)、pg_prewarm利用
- pgconfigurator: OLAP, CPU6, MEM48GB, SSDを指定しコンフィグ生成
- PG18利用

測定結果

No.	利用したコンフィグ	クエリ実行時間の合計
1	デフォルト	34.637秒
2	Pgconfigurator	35.727秒

→ 今回は改善効果なし　※3回測定した中央値

ワークロードや環境が異なれば違う結果に

留意点

- ツールにより提案対象パラメータ数に差
  - 例. PGTune: 17個, pgconfigurator: 80個
- 性能に特化した提案が多い (運用系パラメータは対象外)
- 実行計画が変化する場合があり確認必須
- 実際のワークロードを利用するものは事前に業務アプリが必要

3. 代表的なユースケース

1. コンフィグの値はどのように設定すべき？
2. **作成すべきインデックスは？**
3. クエリ実行時間を短縮したい

インデックスに関する課題とツールのメリット

インデックス作成の課題

- 開発現場ではインデックス作成漏れが発生
- 複雑なクエリでは最適なインデックスを見つけるのが困難

ツールのメリット

- 適切なインデックスを自動提案
- 人間によるクエリ・実行計画の解析作業が不要

インデックスアドバイザの一覧

No.	Name	Author	Target	Note
1	**pg_qualstats**	powa-team (Julien Rouhaud)	Index	https://github.com/powa-team/pg_qualstats
2	hypoPG	Julien Rouhaud	Index	https://github.com/HypoPG/hypopg

仕組みは？(pg_qualstatsの場合)

クエリ実行時の情報を活用

- クエリ実行時のWHERE条件・結合条件などを収集
- それを元にインデックス作成用のDDLを生成

効果を確認してみる: pg_qualstats

NTT

方法

1. HammerDB TPC-H Q17のインデックス削除
2. pg_qualstatsで情報収集後にインデックス提案
3. 当初のインデックスが復元できたか確認

```
SELECT distinct v::text
    FROM json_array_elements(
        pg_qualstats_index_advisor()->'indexes') v
    ORDER BY 1;
                                                    v
-----------------------------------------------------------------------------------------------------------------------
 {"ddl" : "CREATE INDEX ON public.lineitem USING btree (l_partkey)", "queryids" : [9038505691112966471]}
 {"ddl" : "CREATE INDEX ON public.part USING btree (p_container, p_brand)", "queryids" : [9038505691112966471]}
(2 rows)
```

※Q17のクエリテキストはAppendix参照

提案されたインデックスは十分か？

ツールにより提案されたインデックスとデフォルトのインデックスを比較

- Lineitem表向けのインデックスはほぼ再現※
- 当初は存在しなかったPart表に対する新規インデックスの提案あり

No.	種類	インデックスのカラム	
		Lineitem表	Part表
1	デフォルト	l_partkey, l_suppkey	-
2	**ツールによるインデックス**	**l_partkey**	**p_container, p_brand**

➡ クエリ単独で見た場合は
同等以上のインデックス生成に成功

※l_suppkeyが提案されなかった理由は
クエリ内で利用されていなかったため

実行時間はどうなるか

Q17を利用し実行時間を比較

■ 実行時間は**ほぼ同等で問題なし**

No.	種類	Q17実行時間 (ms)
1	デフォルト	2791.7
2	**ツールによるインデックス**	**2725.1**

他のクエリには影響があるか

■ デフォルトのインデックスは外部キー制約をカバー。しかし、
提案されたものはカバーできていない。そのため他のクエリには影響あり

留意点

ツールの仕様把握が重要

- WHERE/JOIN句のみが収集対象 ※
  - ORDER BY句や外部キー制約は対象外

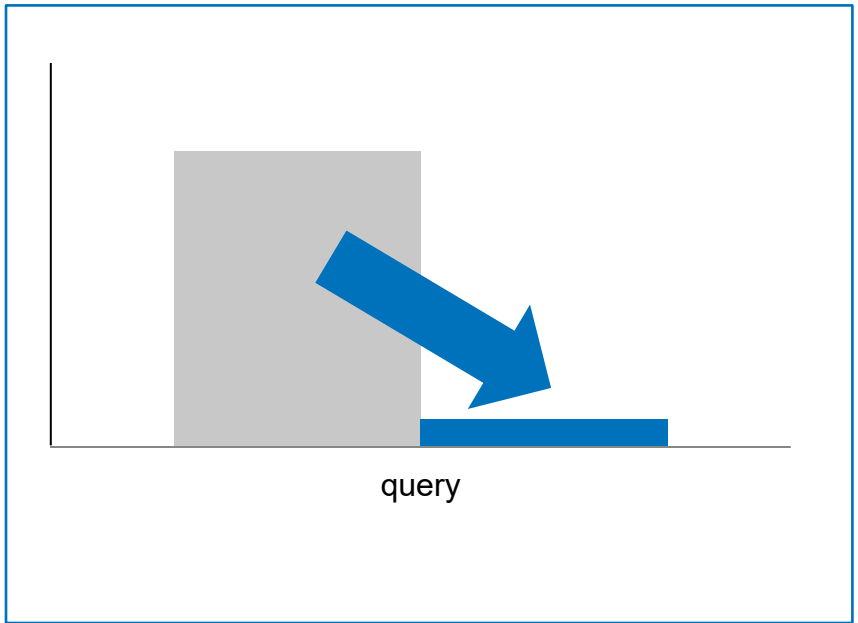
クエリ単位で提案させるとインデックスショットガンとなるので注意

- 複数クエリをまとめて提案させるのが理想的

※pg_qualstatsの場合

3. 代表的なユースケース

1. コンフィグの値はどのように設定すべき？
2. 作成すべきインデックスは？
3. **クエリ実行時間を短縮したい**



実行時間に関する課題とツールのメリット

NTT

実行時間に関する課題

- 商用サービス運用中にクエリが長時間化
  - インデックスチューニングだけでは解決できず性能要件を満たせない
- 複雑なクエリほど実行計画チューニングが困難だが有識者が不足

ツールのメリット

- 効率的な実行計画を提案 (自動チューニング)
- 作成すべき拡張統計を提案 ※

※利用により特定のケースでコスト計算の精度向上

実行計画関連のアドバイザリー機能の一覧

NTT

No.	Name	Author	Target	Note
1	**aqo**	Postgres Professional	Execution Plan	https://postgrespro.com/docs/postgrespro/16/aqo https://github.com/postgrespro/aqo ※Applying a patch is needed
2	**pg_plan_advsr**	Tatsuro Yamada (NTT Open Source Software Center)	Execution Plan, Optimizer Hint, Extended Statistics	https://github.com/ossc-db/pg_plan_advsr.git ※it uses pg_hint_plan, pg_store_plan, pg_qualstats internaly
3	**pg_plan_inspector**	Hironobu Suzuki	Execution Plan	https://github.com/s-hironobu/pg_plan_inspector
4	**pg_index_stats**	Andrei Lepikhov (Postgres Professional)	Extended Statistics	https://eventyay.com/e/55d2a466/session/9083
5	**pg_stat_advisor**	Ilia Evdokimov (Tantor Labs LLC)	Extended Statistics	https://eventyay.com/e/55d2a466/session/9099

仕組みは？

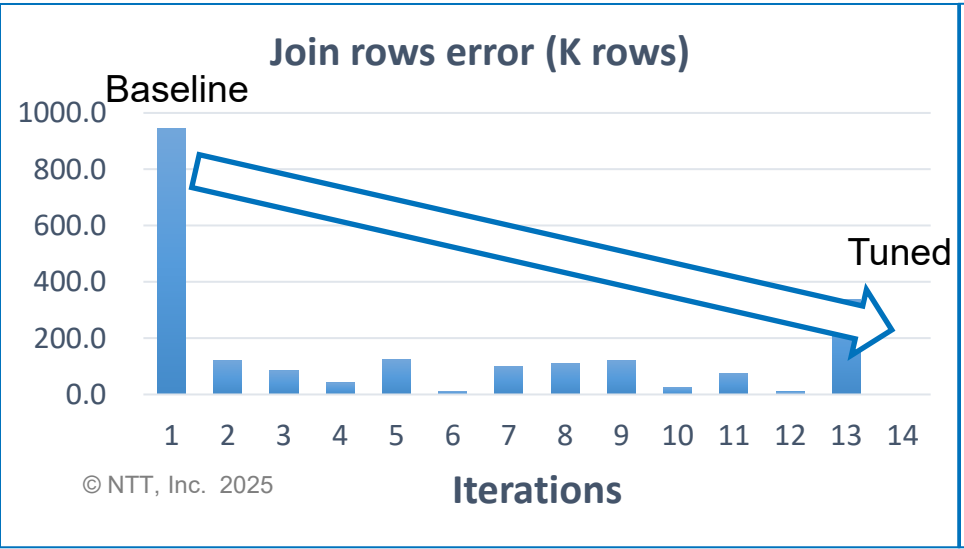
自動チューニング・拡張統計共にクエリ実行時の情報を活用

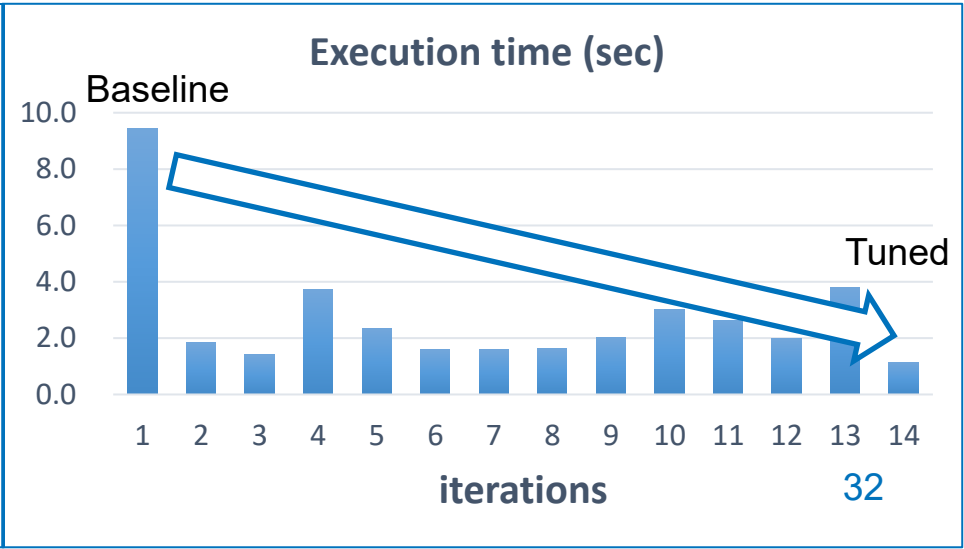
- 自動チューニング: 見積行数誤差を検出し、次回の見積り時に補正
- 拡張統計: 作成した方が良いカラムを取得しDDLを生成

自動チューニングの効果を確認

- Join Order Benchmark Q31、PG16、pg_plan_advsrを利用
- クエリ実行14回後に見積行数誤りはゼロへ
- **実行時間は8.2倍高速化** (9.44秒 -> 1.15秒)

最終的な実行計画は
ヒント句でダンプ





留意点

クエリの種類で使い分けが必要

- 分析系クエリ (結合や集約が多い): 効果大
- OLTP系クエリ (PKで1行取得など): 効果見込めず

検証環境での利用推奨

- 自動チューニングによるリソース消費で業務影響を与えないように

利用難易度は少々高い

- すべてソース提供のためビルド必須
- ドキュメントは英語のみ ※

※ドキュメント翻訳を推奨

1. 背景・目的
2. アドバイザリー機能とは
3. 代表的なユースケース
4. アドバイザリー機能の今後

今後ということでLLMを活用してみる

NTT

問い

- LLMによるアドバイザリー機能の実現性は？

確認方法

- デフォルトのインデックス削除後にインデックスを提案させ評価
  (pg_qualstatsの効果測定と同様)

測定条件

- 使用モデル: gpt-oss:20b
- TPC-H Q17を利用
- PG18

LLMに与えるプロンプト

プロンプトの要旨

あなたは優秀なDBエンジニアです。以下の実行計画を改善し実行時間を短縮するためのインデックスを提案してください。…

```
Aggregate  (cost=2254269.79..2254269.80 rows=1 width=32) (actual time=13971258.715..13971258.766 rows=1.00 loops=1)
  Buffers: shared hit=88148584 read=632511383
  ->  Hash Join  (cost=6464.29..2254264.61 rows=2069 width=8) (actual time=36905.924..13971258.332 rows=467.00 loops=1)
        Hash Cond: (lineitem.l_partkey = part.p_partkey)
        Join Filter: (lineitem.l_quantity < (SubPlan 1))
        Rows Removed by Join Filter: 5105
        Buffers: shared hit=88148584 read=632511383
        ->  Seq Scan on lineitem  (cost=0.00..189284.19 rows=5997219 width=19) (actual time=0.118..665.125 rows=5997334.00 loops=1)
              Buffers: shared hit=15802 read=113510
        ->  Hash  (cost=6461.70..6461.70 rows=207 width=6) (actual time=21.476..21.525 rows=186.00 loops=1)
              Buckets: 1024  Batches: 1  Memory Usage: 16kB
              Buffers: shared read=4191
              ->  Gather  (cost=1000.00..6461.70 rows=207 width=6) (actual time=3.159..21.452 rows=186.00 loops=1)
                    Workers Planned: 2
                    Workers Launched: 2
                    Buffers: shared read=4191
                    ->  Parallel Seq Scan on part  (cost=0.00..5441.00 rows=86 width=6) (actual time=1.162..18.822 rows=62.00 loops=3)
                          Filter: ((p_brand = 'Brand#51'::bpchar) AND (p_container = 'LG BAG'::bpchar))
                          Rows Removed by Filter: 66605
                          Buffers: shared read=4191
                          ・・・
```

LLMの回答は賢そう

2つのインデックスが提案された！

```
CREATE INDEX idx_lineitem_partkey_qty ON lineitem(l_partkey, l_quantity);

   理由: l_partkey を使った結合と、l_quantity でのフィルタが高速化され、スキャンコストが劇的に低減される。
         カバリングインデックスにより、追加のテーブルアクセスが不要になる。


CREATE INDEX idx_part_brand_container ON part(p_brand, p_container, p_partkey);

   理由: p_brand と p_container で先にフィルタリングでき、p_partkey が結合キーとして使われるため、
         インデックススキャンで必要な行だけを取得できる。カバリングインデックスで JOIN も同時に解決できる。
```

※テキストは整形済み

インデックスのカラムを比較

- LLMにより2つの表のためのカバリングインデックスが提案された
- 実行時間を短縮が期待できる？

No.	種類	インデックスのカラム	
		Lineitem表	Part表
1	デフォルト※	l_partkey, l_suppkey	-
2	ツールによるインデックス※	l_partkey	p_container, p_brand
3	**LLMによるインデックス**	**l_partkey, l_quantity**	**p_brand, p_container, p_partkey**

※再掲

実行時間を比較すると

Q17を利用し実行時間を確認

- クエリ単独では実行時間は約270倍高速化
- カバリングインデックスの効果を発揮 (インデックスオンリースキャン利用)

No.	種類	Q17実行時間 (ms)
1	デフォルト※	2791.7
2	ツールによるインデックス※	2725.1
3	**LLMによるインデックス**	**10.1**

※再掲

- 実行計画の詳細はAppendixを参照

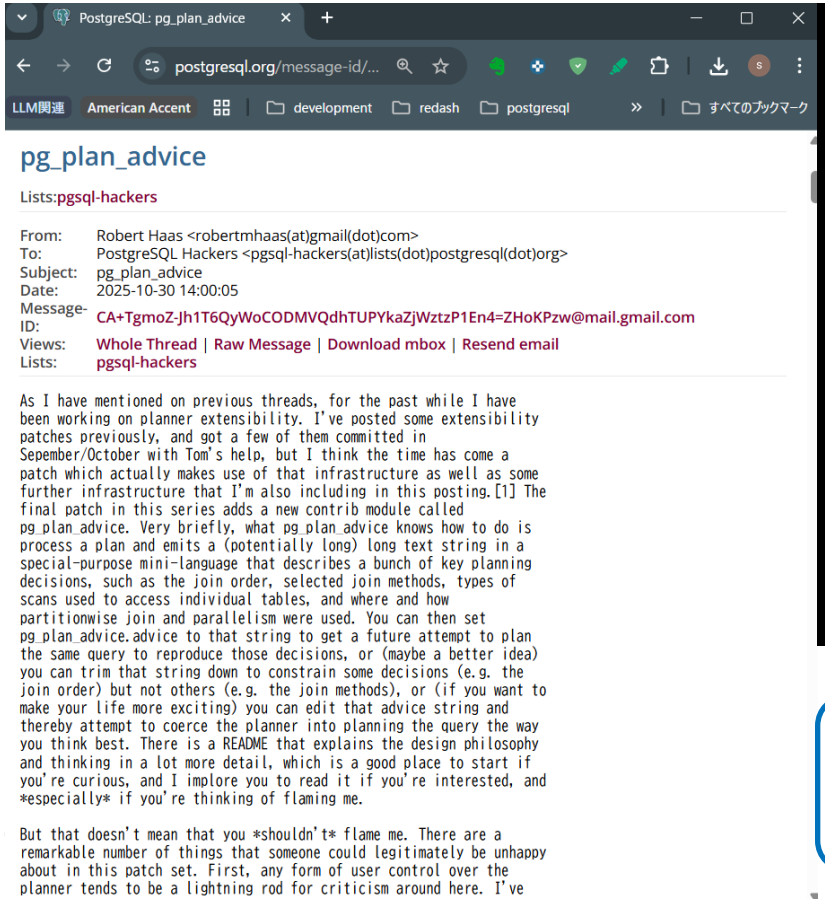
アドバイザリー機能の今後

今後の展望

- LLMの活用は自然な流れ
  - 現時点では電力消費の観点で従来ツールにも価値あり
- DB特化モデル登場でLLMを活用したアドバイザリー機能はデファクトに？

PostgreSQL公式で開発が始まった！

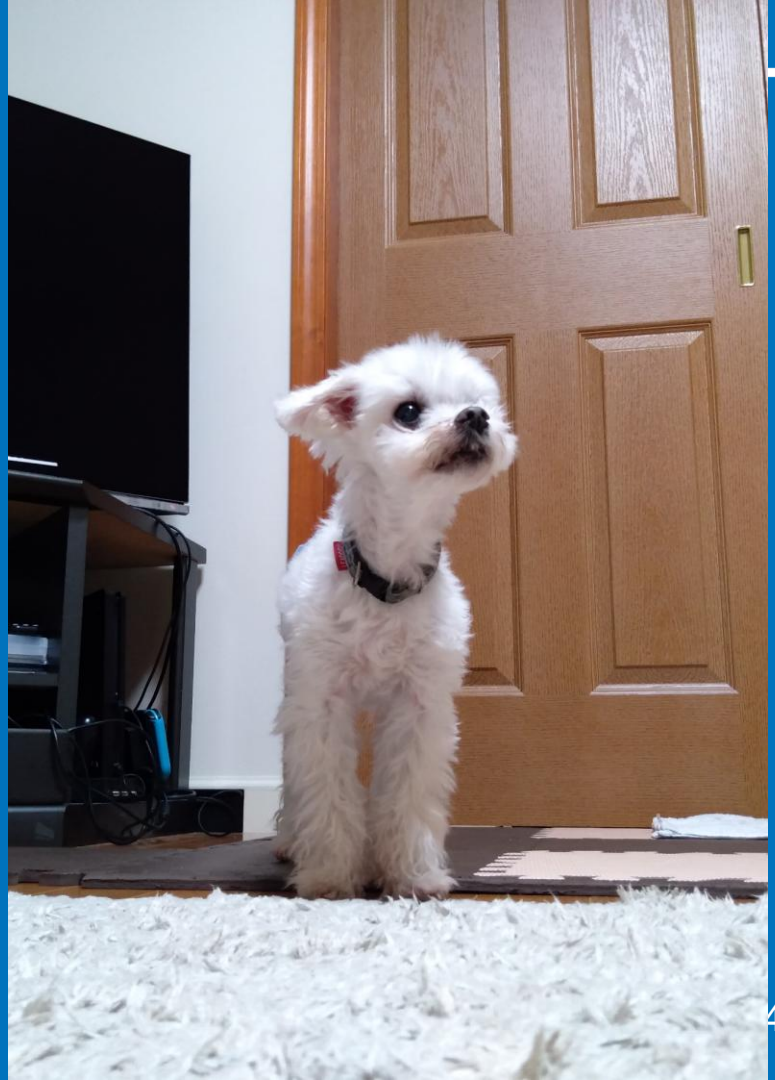
NTT




```
790 +SET LOCAL pg_plan_advice.advice = 'gather((f d))';↓
791 +EXPLAIN (COSTS OFF, PLAN_ADVICE)↓
792 +   SELECT * FROM gt_fact f JOIN gt_dim d ON f.dim_id = d.id;↓
793 +                    QUERY PLAN                          ↓
794 +------------------------------------------------------↓
795 + Gather↓
796 +   Workers Planned: 1↓
797 +   ->  Parallel Hash Join↓
798 +         Hash Cond: (f.dim_id = d.id)↓
799 +         ->  Parallel Seq Scan on gt_fact f↓
800 +         ->  Parallel Hash↓
801 +               ->  Parallel Seq Scan on gt_dim d↓
802 + Supplied Plan Advice:↓
803 +   GATHER((f d)) /* matched */↓
804 + Generated Plan Advice:↓
805 +   JOIN_ORDER(f d)↓
806 +   HASH_JOIN(d)↓
807 +   SEQ_SCAN(f d)↓
808 +   GATHER((f d))↓
809 +(14 rows)↓
```

pg_hint_plan + pg_plan_advsrの
機能のインスパイア系？！

5. まとめ



まとめ

- PostgreSQLには様々な種類のアドバイザリー機能が存在
- コンフィグ/インデックス/実行計画で効果測定結果と留意点を共有
- 使いどころを理解すれば明確に価値あり (公式でも開発開始)
- LLMとの組み合わせによりさらに発展

ご清聴ありがとうございました！

Innovating a Sustainable Future for People and Planet


NTT

6. Q&A

いただいた質問と回答

No.	質問	回答
1	OS（Linuxカーネルやファイルシステム）のパラメータをチューニングするツールはあるのでしょうか？	PostgreSQL界隈では存在しないと考えます。
2	OtterTuneが撤退したように、アドバイザリー機能を主軸としたビジネスは厳しそうですが、今後上手くいく可能性があるでしょうか。AIとも競合しそうです。	アドバイザリー機能のインフラとなりそうな機能開発がPostgreSQL公式で着手されましたが、外部で開発されている既存のアドバイザリー機能と競合するまでに成長するかは疑問が残ります。AIとの競合についてはGPUをDBサーバで標準的に利用可能になるまでは、既存のアドバイザリー機能が利用されるものと考えます。将来的にはアドバイザリー機能がAIを取り込む形で残り続けるのではと考えます。補足: アカデミックな分野では新しい研究が継続して行われているそうです。
3	インデックスや実行計画のアドバイザ機能はGeneric PlanとCustom Planの違いも考慮してくれるのでしょうか？（データの偏りを考慮したアドバイスをしてくれるか？）	インデックスアドバイザではクエリ実行時に利用された実行計画を解析しインデックスが提案されます。従って、Generic/Custom Planの違いを考慮してくれません。もし両者に大きな違いがある場合はそれぞれに対応するインデックスが提案されると考えます（試していませんが）

Appendix

pgconfiguratorによるコンフィグパラメータの出力例 (PG18, OLAPを指定) 1/2

wal_compression = lz4
jit = on
client_connection_check_interval = 30s
default_toast_compression = lz4
enable_async_append = on
autovacuum_vacuum_insert_threshold = 2234
autovacuum_vacuum_insert_scale_factor = 0.01
logical_decoding_work_mem = 64MB
maintenance_io_concurrency = 128
wal_keep_size = 2022MB
hash_mem_multiplier = 8.0
max_parallel_maintenance_workers = 5
max_parallel_workers = 6
max_logical_replication_workers = 7
max_sync_workers_per_subscription = 5
autovacuum = on
autovacuum_max_workers = 5
autovacuum_work_mem = 1019MB
autovacuum_naptime = 15s
autovacuum_vacuum_threshold = 2234
autovacuum_analyze_threshold = 1117
autovacuum_vacuum_scale_factor = 0.001
autovacuum_analyze_scale_factor = 0.0007
autovacuum_vacuum_cost_limit = 2389
vacuum_cost_limit = 8000
autovacuum_vacuum_cost_delay = 10ms
vacuum_cost_delay = 10ms
autovacuum_freeze_max_age = 500000000
autovacuum_multixact_freeze_max_age = 800000000
shared_buffers = 10GB
max_connections = 132
max_files_per_process = 1782
superuser_reserved_connections = 4
work_mem = 149MB
temp_buffers = 16MB
maintenance_work_mem = 1090MB
huge_pages = try
fsync = on
wal_level = replica
synchronous_commit = off
full_page_writes = off
wal_buffers = 31MB
wal_writer_delay = 511ms
wal_writer_flush_after = 5215kB
min_wal_size = 1543MB
max_wal_size = 5215MB
max_replication_slots = 0
max_wal_senders = 0
wal_sender_timeout = 0
wal_log_hints = off
hot_standby = off
wal_receiver_timeout = 0
max_standby_streaming_delay = -1
hot_standby_feedback = off
wal_receiver_status_interval = 0
checkpoint_timeout = 60min
checkpoint_warning = 30s
checkpoint_completion_target = 0.9
commit_delay = 418
commit_siblings = 15
bgwriter_delay = 59ms
bgwriter_lru_maxpages = 532
bgwriter_lru_multiplier = 7.0
effective_cache_size = 31GB
cpu_operator_cost = 0.0025
default_statistics_target = 500
random_page_cost = 1.1
seq_page_cost = 1
join_collapse_limit = 10
from_collapse_limit = 10
geqo = on
geqo_threshold = 12

pgconfiguratorによるコンフィグパラメータの出力例 (PG18, OLAPを指定) 2/2

```
geqo_threshold = 12
effective_io_concurrency = 128
max_worker_processes = 12
max_parallel_workers_per_gather = 3
max_locks_per_transaction = 325
max_pred_locks_per_transaction = 325
statement_timeout = 86400000
idle_in_transaction_session_timeout = 86400000
old_snapshot_threshold = 4320
```

実はPG18用は不具合がありました

✓ old_snapshot_threshold は利用不可(PG17で廃止)

TPC-H Q17のクエリテキスト

```
select sum(l_extendedprice) / 7.0 as avg_yearly
from lineitem, part
where p_partkey = l_partkey
and p_brand = 'Brand#51'
and p_container = 'LG BAG'
and l_quantity < ( select 0.2 * avg(l_quantity) from lineitem where l_partkey = p_partkey);
```

TPC-H Q17の実行計画 デフォルト(1/2)

```
Aggregate  (cost=212773.62..212773.63 rows=1 width=32) (actual time=2791.537..2791.585 rows=1.00 loops=1)
  Buffers: shared hit=190652 read=132923
  ->  Hash Join  (cost=6464.29..212768.44 rows=2069 width=8) (actual time=36.379..2791.418 rows=467.00 loops=1)
        Hash Cond: (lineitem.l_partkey = part.p_partkey)
        Join Filter: (lineitem.l_quantity < (SubPlan 1))
        Rows Removed by Join Filter: 5105
        Buffers: shared hit=190652 read=132923
        ->  Seq Scan on lineitem  (cost=0.00..189284.19 rows=5997219 width=19) (actual time=0.112..675.417 rows=5997334.00 loops=1)
              Buffers: shared hit=6212 read=123100
        ->  Hash  (cost=6461.70..6461.70 rows=207 width=6) (actual time=20.296..20.343 rows=186.00 loops=1)
              Buckets: 1024  Batches: 1  Memory Usage: 16kB
              Buffers: shared read=4191
              ->  Gather  (cost=1000.00..6461.70 rows=207 width=6) (actual time=2.748..20.279 rows=186.00 loops=1)
                    Workers Planned: 2
                    Workers Launched: 2
                    Buffers: shared read=4191
                    ->  Parallel Seq Scan on part  (cost=0.00..5441.00 rows=86 width=6) (actual time=1.343..17.896 rows=62.00 loops=3)
                          Filter: ((p_brand = 'Brand#51'::bpchar) AND (p_container = 'LG BAG'::bpchar))
                          Rows Removed by Filter: 66605
                          Buffers: shared read=4191
```

TPC-H Q17の実行計画 デフォルト(2/2)

```
        SubPlan 1
          ->  Aggregate  (cost=127.70..127.71 rows=1 width=32) (actual time=0.066..0.066 rows=1.00 loops=5572)
                Buffers: shared hit=184440 read=5632
                ->  Bitmap Heap Scan on lineitem lineitem_1  (cost=4.67..127.62 rows=31 width=5) (actual time=0.019..0.053 rows=30.98 loops=5572)
                      Recheck Cond: (l_partkey = part.p_partkey)
                      Heap Blocks: exact=172602
                      Buffers: shared hit=184440 read=5632
                      ->  Bitmap Index Scan on lineitem_part_supp_fkidx  (cost=0.00..4.67 rows=31 width=0) (actual time=0.013..0.013 rows=30.98 loops=5572)
                            Index Cond: (l_partkey = part.p_partkey)
                            Index Searches: 5572
                            Buffers: shared hit=17156 read=314
      Planning:
        Buffers: shared hit=67 read=14
      Planning Time: 0.663 ms
      Execution Time: 2791.675 ms
     (35 rows)
```

TPC-H Q17の実行計画 ツールによるインデックス利用 (1/2)

```
Aggregate  (cost=206979.89..206979.90 rows=1 width=32) (actual time=2724.903..2724.906 rows=1.00 loops=1)
  Buffers: shared hit=199536 read=119928
  ->  Hash Join  (cost=662.68..206974.94 rows=1979 width=8) (actual time=10.842..2724.748 rows=467.00 loops=1)
        Hash Cond: (lineitem.l_partkey = part.p_partkey)
        Join Filter: (lineitem.l_quantity < (SubPlan 1))
        Rows Removed by Join Filter: 5105
        Buffers: shared hit=199536 read=119928
        ->  Seq Scan on lineitem  (cost=0.00..189290.61 rows=5997861 width=19) (actual time=0.103..661.344 rows=5997334.00 loops=1)
              Buffers: shared hit=14776 read=114536
        ->  Hash  (cost=660.20..660.20 rows=198 width=6) (actual time=1.138..1.139 rows=186.00 loops=1)
              Buckets: 1024  Batches: 1  Memory Usage: 16kB
              Buffers: shared hit=11 read=178
              ->  Bitmap Heap Scan on part  (cost=6.45..660.20 rows=198 width=6) (actual time=0.149..1.034 rows=186.00 loops=1)
                    Recheck Cond: ((p_container = 'LG BAG'::bpchar) AND (p_brand = 'Brand#51'::bpchar))
                    Heap Blocks: exact=186
                    Buffers: shared hit=11 read=178
                    ->  Bitmap Index Scan on part_p_container_p_brand_idx  (cost=0.00..6.40 rows=198 width=0) (actual time=0.053..0.054 rows=186.00 loops=1)
                          Index Cond: ((p_container = 'LG BAG'::bpchar) AND (p_brand = 'Brand#51'::bpchar))
                          Index Searches: 1
                          Buffers: shared read=3
```

TPC-H Q17の実行計画 ツールによるインデックス利用 (2/2)

```
SubPlan 1
        ->  Aggregate  (cost=127.70..127.71 rows=1 width=32) (actual time=0.065..0.065 rows=1.00 loops=5572)
            Buffers: shared hit=184749 read=5214
            ->  Bitmap Heap Scan on lineitem lineitem_1  (cost=4.67..127.62 rows=31 width=5) (actual time=0.018..0.051 rows=30.98 loops=5572)
                Recheck Cond: (l_partkey = part.p_partkey)
                Heap Blocks: exact=172602
                Buffers: shared hit=184749 read=5214
                ->  Bitmap Index Scan on lineitem_l_partkey_idx  (cost=0.00..4.67 rows=31 width=0) (actual time=0.012..0.012 rows=30.98 loops=5572)
                    Index Cond: (l_partkey = part.p_partkey)
                    Index Searches: 5572
                    Buffers: shared hit=17083 read=278
 Planning:
   Buffers: shared hit=63 read=14
 Planning Time: 0.635 ms
 Execution Time: 2725.063 ms
(35 rows)
```

TPC-H Q17の実行計画 LLMによるインデックス利用 (1/2)

```
Aggregate  (cost=10503.10..10503.12 rows=1 width=32) (actual time=10.009..10.011 rows=1.00 loops=1)
  Buffers: shared hit=5462 read=309
  ->  Nested Loop  (cost=5.90..10497.83 rows=2109 width=8) (actual time=0.118..9.886 rows=467.00 loops=1)
        Buffers: shared hit=5462 read=309
        ->  Index Only Scan using idx_part_brand_container on part  (cost=0.42..12.66 rows=212 width=6) (actual time=0.039..0.137 rows=186.00 loops=1)
              Index Cond: ((p_brand = 'Brand#51'::bpchar) AND (p_container = 'LG BAG'::bpchar))
              Heap Fetches: 0
              Index Searches: 1
              Buffers: shared hit=1 read=5
        ->  Index Scan using idx_lineitem_partkey_qty on lineitem  (cost=5.48..49.36 rows=10 width=19) (actual time=0.007..0.009 rows=2.51 loops=186)
              Index Cond: ((l_partkey = part.p_partkey) AND (l_quantity < (SubPlan 1)))
              Index Searches: 186
              Buffers: shared hit=1025
              SubPlan 1
                ->  Aggregate  (cost=5.04..5.05 rows=1 width=32) (actual time=0.041..0.041 rows=1.00 loops=186)
                      Buffers: shared hit=4436 read=304
                      ->  Index Only Scan using idx_lineitem_partkey_qty on lineitem lineitem_1  (cost=0.43..4.96 rows=30 width=5) (actual time=0.021..0.033 rows=29.96 loops=186)
                            Index Cond: (l_partkey = part.p_partkey)
                            Heap Fetches: 0
                            Index Searches: 186
                            Buffers: shared hit=4436 read=304
```

TPC-H Q17の実行計画 LLMによるインデックス利用 (2/2)

```
Planning:
   Buffers: shared hit=134 read=17
 Planning Time: 0.891 ms
 Execution Time: 10.066 ms
(25 rows)
```

JOB Q31のクエリテキスト

```
EXPLAIN ANALYZE
SELECT MIN(mi.info) AS movie_budget,
       MIN(mi_idx.info) AS movie_votes,
       MIN(n.name) AS writer,
       MIN(t.title) AS violent_liongate_movie
FROM cast_info AS ci,
     company_name AS cn,
     info_type AS it1,
     info_type AS it2,
     keyword AS k,
     movie_companies AS mc,
     movie_info AS mi,
     movie_info_idx AS mi_idx,
     movie_keyword AS mk,
     name AS n,
     title AS t
WHERE ci.note IN ('(writer)',
                  '(head writer)',
                  '(written by)',
                  '(story)',
                  '(story editor)')
  AND cn.name LIKE 'Lionsgate%'
  AND it1.info = 'genres'
  AND it2.info = 'votes'
  AND k.keyword IN ('murder',
                    'violence',
                    'blood',
                    'gore',
                    'death',
                    'female-nudity',
                    'hospital')
  AND mi.info IN ('Horror',
                  'Action',
                  'Sci-Fi',
                  'Thriller',
                  'Crime',
                  'War')
  AND t.id = mi.movie_id
  AND t.id = mi_idx.movie_id
  AND t.id = ci.movie_id
  AND t.id = mk.movie_id
  AND t.id = mc.movie_id
  AND ci.movie_id = mi.movie_id
  AND ci.movie_id = mi_idx.movie_id
  AND ci.movie_id = mk.movie_id
  AND ci.movie_id = mc.movie_id
  AND mi.movie_id = mi_idx.movie_id
  AND mi.movie_id = mk.movie_id
  AND mi.movie_id = mc.movie_id
  AND mi_idx.movie_id = mk.movie_id
  AND mi_idx.movie_id = mc.movie_id
  AND mk.movie_id = mc.movie_id
  AND n.id = ci.person_id
  AND it1.id = mi.info_type_id
  AND it2.id = mi_idx.info_type_id
  AND k.id = mk.keyword_id
  AND cn.id = mc.company_id;
```