PGECons
PostgreSQL Enterprise Consortium

2021年度WG3活動成果報告
拡張機能開発ことはじめ

PostgreSQL エンタープライズコンソーシアム
WG3 パブリッククラウド検証チーム

Contents

- PostgreSQLにおける拡張機能開発
- 拡張機能開発手順
- 拡張機能開発イメージ: test_explain
  - 今年度検証で挑戦した内容をご紹介
- 総括

責任範囲

- 本資料は、PGEConsが独自に検証した結果であり、結果はPGEConsの責任の元、公開しています。

PostgreSQLにおける拡張機能開発

PostgreSQLにおける拡張機能開発

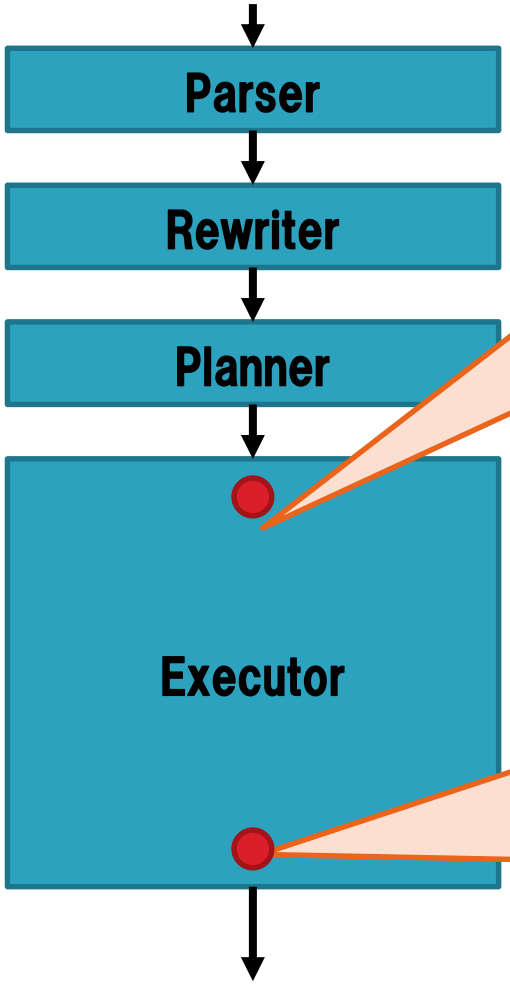
- PostgreSQLは拡張機能を導入することにより、機能を追加することが出来る
  - 拡張機能は自分で開発することも可能

- 今年度は”Hook”を用いた拡張機能開発を通じて、PostgreSQLの内部処理の介入に挑戦!
  - 既存拡張機能 (auto_explain) の機能変更
  - 開発手順＋実装 (検証) 過程、結果のご紹介

- Hook…PostgreSQL内で共有されるポインタ
  - こちらに独自関数を差し込むことで内部処理に割込可
    - イメージ： SQL実行後に処理を追加

PostgreSQLにおける拡張機能開発

- Hookの例:auto_explainにおける利用



ExecutorStart_hook

Executorの実行開始時に処理を定義

auto_explainでは…

実行計画取得準備(実行時間の閾値設定等)

実装次第では、以下のようにカスタマイズも可能

・独自のフラグを設定

ExecutorEnd_hook

Executorの実行完了時に処理を定義

auto_explainでは…

スロークエリの実行計画をログ出力

実装次第では、以下のようにカスタマイズも可能

・任意のファイルに出力

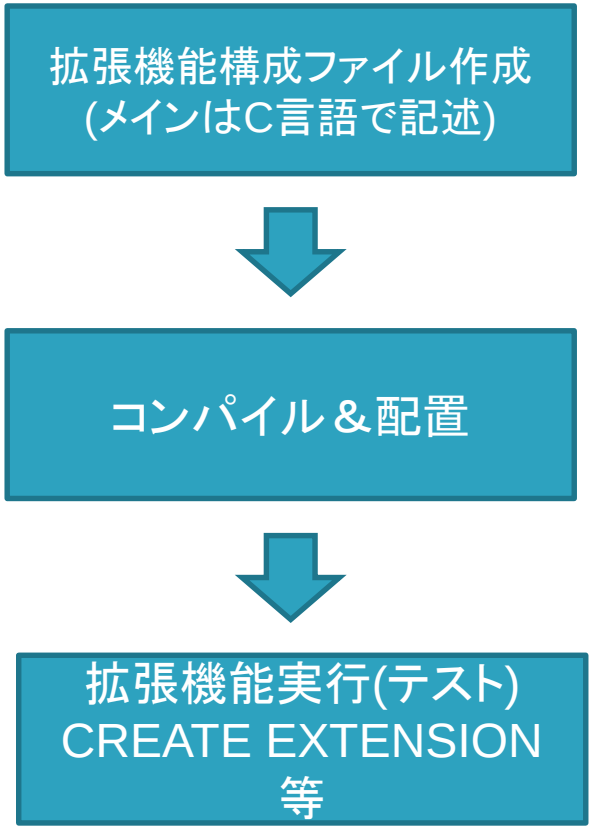
・Selectできるようにテーブル管理に変更

注意点

- パブリッククラウドでDBサービスを利用する場合（PaaS）においても拡張機能は導入可ではあるが、ベンダー側で事前に用意されたものしか利用できないケースが多い
  - 自前で用意したものの活用や、既存の変更はNG

拡張機能開発手順（Hook）

拡張機能開発手順（Hook）：概要



- ”Hello World”拡張機能の実装内容＆手順をご紹介
  - SQL実行時に“Hello World”とコンソール出力

```
testdb=# select 1;
NOTICE:  HELLO WORLD!
 ?column?
----------
        1
(1 row)
```

- 拡張機能は基本、4ファイル構成
  - hello_world.control
  - hello_world--<ver>.sql
  - hello_world.c
  - Makefile

hello_world.control

- CREATE EXTENSION時の動作設定ファイル
  - 一旦作成し、基本的には変更しなくてOK

- 主要な構成（設定）要素
  - directory：SQLファイルのパス
  - default_version：バージョン指定
  - module_pathname：モジュール（soファイル）のパス
  - relocatable：拡張機能の別スキーマでの利用可否
    - 全スキーマで利用したい場合はTRUE

hello_world.control

- “Hello World”における例

こちらのパスにコンパイルしたファイル(.soファイル)を配置すればOK

```
default_version = '1.0'
module_pathname = '/opt/test_postgres/lib'
relocatable = true
```

hello_world--<ver>.sql

- CREATE EXTENSION時に実行されるSQLファイル
  - 拡張機能でテーブルや関数を利用したい場合に記述
  - 定義したテーブル等はDROP EXTENSION時に自動削除

- Controlファイルのdefault_versionで
  指定したバージョンのSQLが実行される

- 特にテーブル等作成する必要がない場合は空でOK
  - “Hello World”においては空ファイルを作成

hello_world.c

- ここに作成したい拡張機能を実装!

- 今回利用するHook：EXECUTOR_START
  - エグゼキュータの開始時に呼ばれるHook
    - プランナによる実行計画作成後にエグゼキュータ実行
  - EXECUTOR_STARTに独自関数を定義して差し込み、処理後に元の処理に引き渡すように変更
    - 独自関数内で”Hello World”を実装

hello_world.c

- 今回における例

```
#include "postgres.h"
#include "executor/executor.h"

PG_MODULE_MAGIC;
extern void _PG_init(void);
extern ExecutorStart_hook_type ExecutorStart_hook ;

static void
HL_ExecutorStart_hook(QueryDesc *queryDesc, int eflags)
{
    elog(NOTICE, "HELLO WORLD!");
    standard_ExecutorStart(queryDesc, eflags);
}

void
_PG_init(void)
{
        ExecutorStart_hook = HL_ExecutorStart_hook;
}
```

hello_world.c

- 今回における例

```
#include "postgres.h"
#include "executor/executor.h"

PG_MODULE_MAGIC;
extern void _PG_init(void);
extern ExecutorStart_hook_type ExecutorStart_hook ;

static void
HL_ExecutorStart_hook(QueryDesc *queryDesc, int eflags)
{
    elog(NOTICE, "HELLO WORLD!");
    standard_ExecutorStart(queryDesc, eflags);
}

void
_PG_init(void)
{
        ExecutorStart_hook = HL_ExecutorStart_hook;
}
```

ExecutorStartのHookに、
今回実装する関数を定義

hello_world.c

- 今回における例

```
#include "postgres.h"
#include "executor/executor.h"

PG_MODULE_MAGIC;
extern void _PG_init(void);
extern ExecutorStart_hook_type ExecutorStart_hook ;

static void
HL_ExecutorStart_hook(QueryDesc *queryDesc, int eflags)
{
    elog(NOTICE, "HELLO WORLD!");
    standard_ExecutorStart(queryDesc, eflags);
}

void
_PG_init(void)
{
        ExecutorStart_hook = HL_ExecutorStart_hook;
}
```

本来の処理(standard_ExecutorStart)の前にelog(コンソール出力)を挟み、割込み処理を実現

Makefile

- Cファイルコンパイル時の制御ファイル
  - contribの拡張機能のMakefileを一部流用（赤字部分）
  - MODULES：soファイル名（拡張子抜き）
  - EXTENSION：拡張機能名
  - DATA：SQLファイル名

```
MODULES = hello_world
EXTENSION = hello_world
DATA = hello_world--1.0.sql

subdir = contrib/hello_world
top_builddir = ../..
include $(top_builddir)/src/Makefile.global
include $(top_srcdir)/contrib/contrib-global.mk
```

各ファイルの配置

- hello_world.c , Makefile
  - 任意の位置

- hello_world.control, hello_world--1.0.sql
  - PostgreSQLインストール時の${SHARELIB}配下に配置
    - shellで”pg_config --sharedir”コマンドで確認可能
  - 例：/usr/pgsql-13/share/extension

- hello_world.so（コンパイル後ファイル）
  - controlファイルのmodule_pathnameに記載のパス

コンパイル、実行手順

1. cファイルをmakeしてsoファイル作成
2. soファイルをcontrolファイル記載の位置に配置
3. 拡張機能読み込み
   - postgres.confのshared_preload_librariesに追記
   - 接続後にCREATE EXTENSION (LOAD)
4. 実行し動作確認

拡張機能開発・実行例

- 最小構成のAmazon EC2で実施
  - PostgreSQLのバージョンは13で実施
  - コンパイル (Makefile) はcontribのソースを流用
    - contribのビルドと同様の手順でmake可能

- 実施手順例
  1. PostgreSQL ver13のソースを配置
  2. コンパイル準備のためのconfigure (必要に応じてmake)
     - “<配置ディレクトリ>/configure”コマンド実行
  3. <配置ディレクトリ>/contrib配下にmkdirしファイル配置
     - hello_worldディレクトリを作成しMakefile,hello_world.cを配置
  4. makeコマンド実行でsoファイル作成
     - hello_world.o , hello_world.soファイルが作成される
  5. 実行させるPostgreSQLの${SHARELIB}にsoファイル配置＆実行

実装イメージ：test_explain

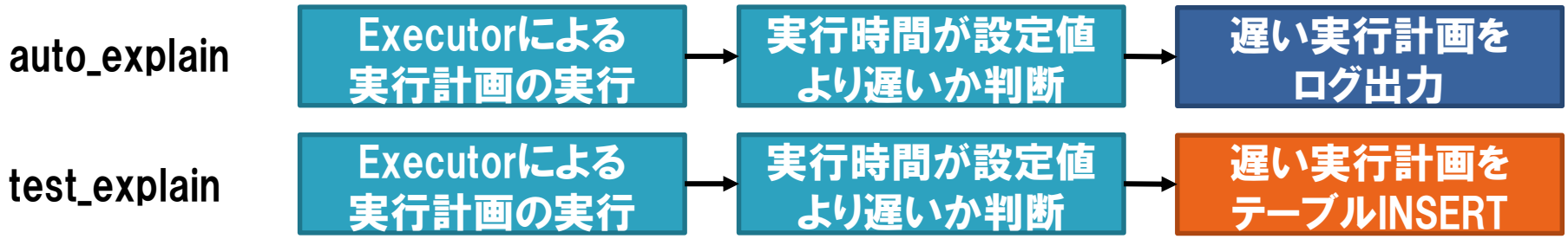
Hook開発イメージとしてのtest_explain

- contribの拡張機能:auto_explainの機能変更版として、test_explainの実装に挑戦
  - 当初の実装方法の想定でトラブル（実装方法変更）があったため、経緯も含めてご紹介

- 本章の構成
  - test_explainの仕様紹介
  - 実装方法紹介
  - テスト時のトラブル
  - 代替案の検討

test_explainの仕様

- auto_explainを機能変更し、実行計画をログではなく実テーブルに挿入する拡張機能の開発
- test_explainで実装する機能
  - 実行計画の取得
    - Executorに反応するHookを用いて実行計画を取得
  - 初期テーブル作成
    - 実行計画の格納先テーブルを作成
  - SQL実行（実テーブルINSERT）
    - Executorに反応するHookを用いて実行計画をテーブルにINSERT
  - 例外処理
    - 実行計画の挿入エラーなどのエラー処理を追加



test_explainの実装

- 実行計画の取得
  - Executor Hookを用いて実行計画を取得
    - auto_explainのExecutorの後処理(ExecutorEnd)を参考に、実行時間の遅いクエリのプランツリーをテキスト変換
    - auto_explainの内容を流用

- 初期テーブル作成
  - 日時、実行時間、実行計画をカラムとするテーブルを作成
    - GitHubで公開されているpg_stat_statementsを参考にして、test_explain--<ver>.sqlにテーブル作成SQLを定義
    - CREATE EXTENSION実行によりテーブルを作成し、DROP EXTENSIONによってテーブルを削除
    - test_explain.controlで指定したバージョンが使用される

test_explainの実装

- SQL実行（実テーブルINSERT）
  - 拡張機能に似たような実装がないか調査
    - Githubで調査するもSQL実行させる既存拡張機能は見当たらず
    - 参考URL:
      - https://github.com/postgres/postgres/tree/master/contrib
      - https://github.com/topics/postgresql-extension
      - https://pgxn.org/
  - テーブルINSERT時に呼ばれる関数を調査
    - gdb（デバッガツール）を用いてINSERT時に呼ばれる関数を特定
    - 本体側の関数であるクエリ実行関数（exec_simple_query）を用いてExecutorEnd内でクエリの実行を検討
  - PostgreSQL本体のソースにあるexec_simple_query関数を参考にしながら実装
    - 参考URL: https://doxygen.postgresql.org/index.html

test_explainの実装

- 例外処理
  - 実行計画の取得 ※auto_explainから変更部分なし
    - 実行計画の取得はExecutorで実行されているため、例外処理もExecutorで実施
  - SQL実行 (実テーブルINSERT)
    - PostgreSQL本体のソースから流用

test_explainのテスト

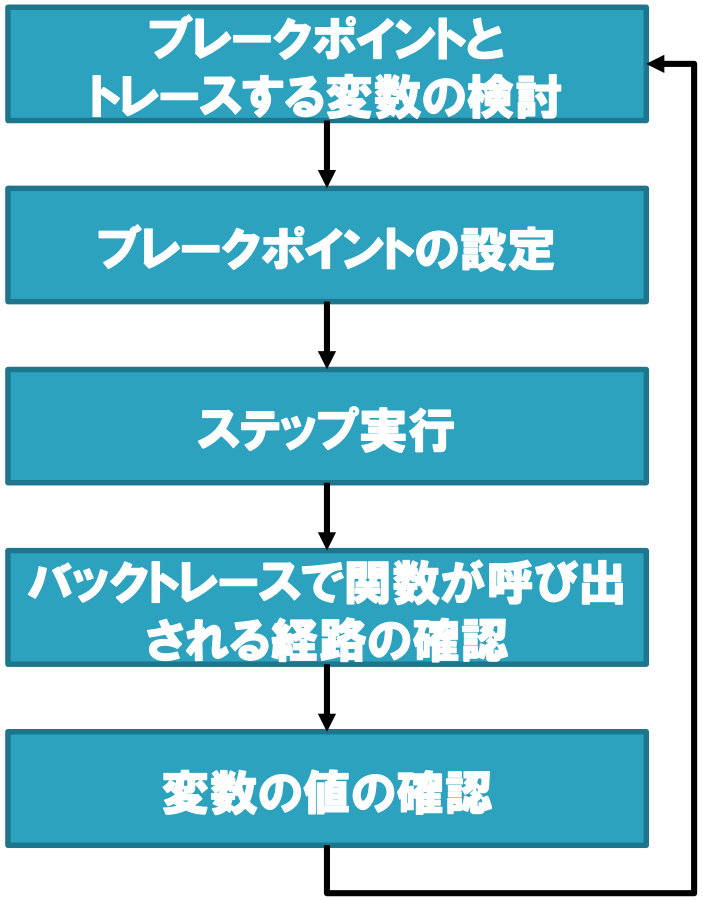
- Executor Hookでのテーブルへの INSERTに失敗
  - 本体側の関数であるクエリ実行関数（exec_simple_query）でINSERTを試みるもエラー発生
  - トランザクション2重起動によるreference leak と思われるエラー

    【エラー内容】
    WARNING: Snapshot reference leak: Snapshot 0x2c461f8 still referenced
    ERROR: snapshot reference 0x2c461f8 is not owned by resource owner Portal

  - gdbを用いて、ブレークポイントを打ちながら ステップ実行し、原因の考察を実施

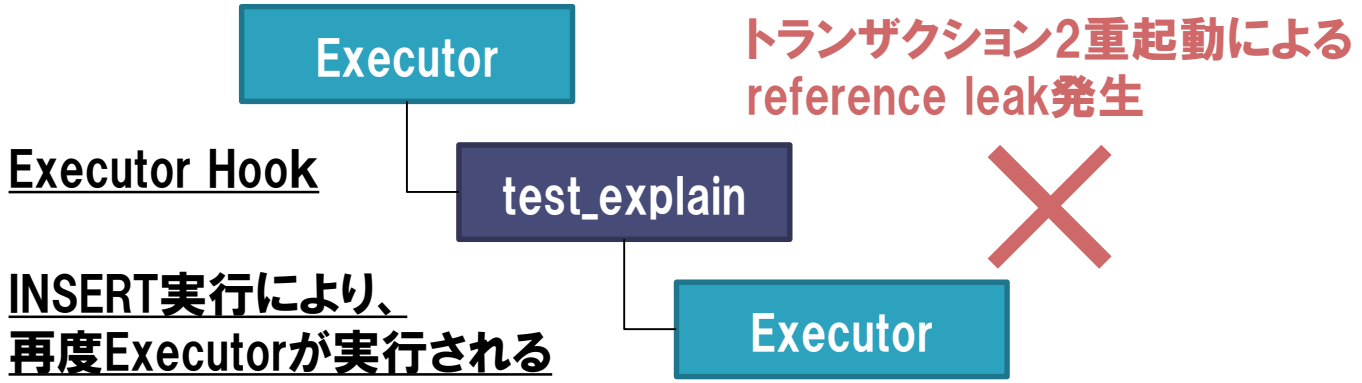
test_explainのテスト

- gdbで利用した機能
  - ブレークポイントの設定
  - ステップ実行
  - 変数のトレース
  - バックトレース



test_explainのテスト

- エラーの原因考察
  - トランザクション2重起動によりreference leakが発生したと推測
    - gdbを用いて、Executor Hookが2回呼び出されていることを確認



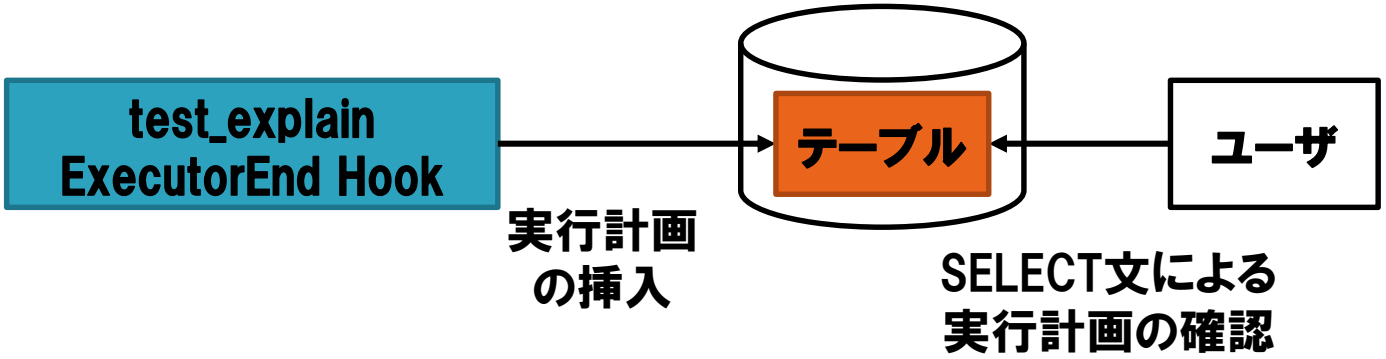
  - Executor Hook内でクエリを実行させるのは厳しい？？
    - クエリ実行関数(exec_simple_query)は使えないと思われる

test_explainの代替案

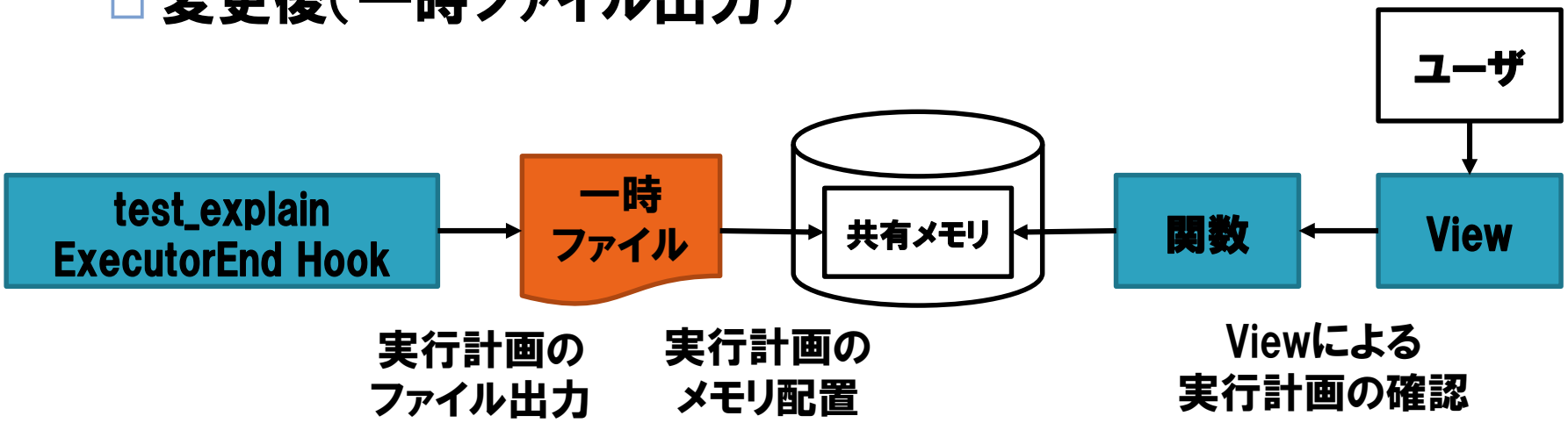
- 代替案の検討
  - 類似処理が存在するpg_stat_statementsを調査
    - 実テーブルへのINSERTではなく、一時ファイルに統計情報を出力し、統計情報を参照できる関数とViewの作成部分を利用できないか検討
  - 以下機能の実装方針を変更
    - 実行計画の実テーブル保存
      → 一時ファイル読み書きによる実行計画の保存に変更
    - テーブル作成
      → 実行計画を参照できる関数とViewの定義に変更
    - 例外処理
      → 上記変更に伴い例外処理も変更

test_explainの代替案

- 代替案の検討
  - 計画当初(実テーブルINSERT)



  - 変更後(一時ファイル出力)



test_explainの代替案の実装方針

- 一時ファイル読み書きによる実行計画の保存
  - ファイル書き出し
    - テキスト変換したプランツリーを外部ファイルに出力
  - ファイル読みこみ
    - 一時ファイル記載の実行計画を共有メモリ内に格納
- 実行計画を参照できるSQL関数とViewの定義
  - test_explain--<ver>.sqlに以下定義
    - 共有メモリ上に配置した実行計画を読み込むC言語関数を定義
      (CREATE FUNCTIONでC言語関数を呼びだし)
    - 上記FUNCTIONを呼び出すViewを定義
  - pg_stat_statementsを流用し、カラムのみカスタマイズ

test_explainの代替案の実装方針

- 例外処理
  - “ファイル書き出し”部分の例外処理
    - 排他制御に関してpg_stat_statementsの関数を流用
      - 同一ファイルへの書き込みのため排他制御が必要
  - “ファイル読み込み”部分の例外処理
    - DBユーザ権限の確認、共有メモリへのデータ挿入、排他制御等の例外処理をファイル書き出しと同様に流用

test_explainの代替案：各実装状況

- 一時ファイル読み書きによる実行計画の保存
  - ファイル書き出しはテスト完了
  - ファイル読み込みは現在テスト中

- 実行計画を参照できる関数とViewの定義
  … テスト中

- 例外処理
  … テスト中

test_explain検証のまとめ

- C言語の知識は必要だが、公開されている拡張機能を参考にすれば、実装/テストのハードルは想像より高くない
  - gdbも複雑な機能は使っていないので、そこまで敷居は高くない
- Executorの挙動や排他制御、メモリ構造などPostgreSQL内部構造の知識があれば、よりスムーズに実装出来た
  - クエリ実行(テーブルINSERT)の拡張機能の調査に時間を要したが、そもそもExecutorのHook内ではSQLを実行させることが難しかったと思われるため、類似機能が存在しなかった?
- 今回の検証を通じてPostgreSQLの理解につながった
  - 既存拡張機能のソースを読んだり、gdbなどのデバッガツールを使って内部処理を追ったりすることで理解が進む

総括

総括

- PostgreSQLの拡張機能開発を行うことで、既存機能の一部変更等のカスタマイズが可能
  - C言語の知識＋様々なソースを参考にすればOK
  - 本体側のソースや、内部構造に触れるきっかけに

- 拡張機能開発に是非、挑戦してみてください!
  - 開発した拡張機能を本番環境で利用する場合は、十分なテストが必要な点には注意

ライセンス

本作品はCC-BYライセンスによって許諾されています。ライセンスの内容を知りたい方はこちらでご確認ください。 文書の内容、表記に関する誤り、ご要望、感想等につきましては、PGEConsのサイトを通じてお寄せいただきますようお願いいたします。

- Amazon Web Services、"Powered by Amazon Web Services"ロゴ、Amazon EC2、Amazon S3、Amazon Relational Database Service (Amazon RDS) およびAmazon Auroraは、米国その他の諸国における、Amazon.com, Inc.またはその関連会社の商標です。
- IBMおよびDb2は、世界の多くの国で登録されたInternational Business Machines Corporationの商標です。
- Linux は、Linus Torvalds 氏の日本およびその他の国における登録商標または商標です。
- Red HatおよびShadowman logoは、米国およびその他の国におけるRed Hat,Inc.の商標または登録商標です。
- Microsoft、Microsoft Azure、Windows Server、SQL Server、米国 Microsoft Corporationの米国及びその他の国における登録商標または商標です。
- MySQLは、Oracle Corporation 及びその子会社、関連会社の米国及びその他の国における登録商標です。 文中の社名、商品名等は各社の商標または登録商標である場合があります。
- Oracleは、Oracle Corporation 及びその子会社、関連会社の米国及びその他の国における登録商標です。 文中の社名、商品名等は各社の商標または登録商標である場合があります。
- PostgreSQLは、PostgreSQL Community Association of Canadaのカナダにおける登録商標およびその他の国における商標です。
- TPC, TPC Benchmark, TPC-B, TPC-C, TPC-E, tpmC, TPC-H, TPC-DS, QphHは米国Transaction Processing Performance Councilの商標です。
- その他、本資料に記載されている社名及び商品名はそれぞれ各社が 商標または登録商標として使用している場合があります 。

著者

（企業・団体名順）

版	所属企業・団体名	部署名	氏名
第1.0版 （2021年度 WG3）	日鉄ソリューションズ株式会社	流通・サービスソリューション事業本部 アドバンストテクノロジー部	伊藤　春
			永井　光
			秋山 暉佳