Profilování PlpgSQL nástroje

Pavel Stěhule

Pavel Stěhule

- Extenze
  - `Orafce`, `plpgsql_check`
- Pager `pspg`
- Patche
  - Variadické parametry, defaultní parametry, ...
  - funkce `format`, `xmltable`, ...
- Patche pro PLpgSQL
  - `RETURN QUERY`
  - `GET STACKED DIAGNOSTICS`

Historie

- 1998 PostgreSQL 6.4
  PLpgSQL - Jan Wieck
- 2007 PostgreSQL 8.3
  PLpgSQL Debug API - Korry Douglas
- 2009 PostgreSQL 8.4
  track_functions, auto_explain

Historie

- 2011 plpgsql_lint
- 2012 plProfiler separated from plDebugger
- 2013 plpgsql_check
- 2015 plProfiler revitalized by OpenSGC
- 2016 plProfiler + GraphUI
- 2018 plpgsql_check + profiler

DBG API

- Set of callback functions used by PLpgSQL executor
  - func_init
  - func_begin
  - func_end
  - stmt_begin
  - stmt_end
  - error_callback
  - assign_expression

Motivace

- Zjištění chování aplikce
- Vyšší výkon
- Lepší uživatelská zkušenost
- Stabilita
- Znalost aplikace umožňuje správná strategická rozhodnutí

Zdroje problémů

- Komplexita kódu
- Nedostatečné znalosti použitých technologií
- Sw hw chyby na všech úrovních

Co chceme

- Najít pomalé nebo často používané funkce
- Najít pomalý kód uvnitř funkcí
- Najít pomalé dotazy

Example

```
CREATE OR REPLACE FUNCTION slow_one(n int, v int)
RETURNS int[] AS $$
DECLARE r int[] = ´{}´;
BEGIN
  FOR i IN 1..n
  LOOP
    r := r || v;
  END LOOP;
  RETURN r;
END;
$$ LANGUAGE plpgsql;
```

Proč plpgsql může být pomalý

- Změna immutable hodnoty
- Implicitní přetypování
- Špatná optimalizace
  - Neodpovídající si typy
  - Prepared statements
- **Problémy nejsou v samotném PLpgSQL**

PLpgSQL

- Jednoduchý AST interpret
  - Rychlé, ale některé příkazy není možné jednoduše implementovat (GOTO)
- Každý výraz je SELECT (aktuálně bez podpory JIT)
- Navržen jako procedurální obálka SQL příkazů (nikoliv jako náhrada SQL)
- Není navržen pro výpočetně náročné operace

Co hledáme?

- Často volané funkce
- Pomalé funkce
- Pomalé dotazy

Example

```
CREATE TABLE users(id bigint PRIMARY KEY, …);

CREATE OR REPLACE FUNCTION slow_01(user_id numeric)
RETURNS void AS $$
BEGIN
  IF EXISTS(SELECT * FROM users WHERE id = user_id) THEN
    ...
```

Nástroje

- track_functions
- auto_explain (nested)
- plProfiler
- plpgsql_check

track_functions

```
SET track_function to ´all´;
…
postgres=# select * from pg_stat_user_functions ;
```

funcid	schemaname	funcname	calls	total_time	self_time
16699	public	fast_func	100	0.407	0.407
16701	public	test_func	1	1034.994	4.203
16698	public	slow_func	100	1030.383	1030.383

```
(3 rows)
```

auto_explain

- log_nested_statements to on
-

plpgsql_check (profiler)

```
load 'plpgsql_check';
set plpgsql_check.profiler to on;

select * from plpgsql_profiler_function_tb('slow_func');
```

plpgsql_check

- C extenze
- Kontrola SQL identifikátorů vložených dotazů
- Detekce několika výkonnostních problémů
  - (implicitní přetypování, špatné omarkování funkcí)
- Detekce bezpečnostních problémů (SQL injection)
- Integrovaný profiler
- Jednoduché používání

plProfiler

```
plprofiler run --command "select test_func(100)" \
   --output test_func.html -h localhost
```

plProfiler

- C extenze, Python pro reporty
- HTML reporty
- Pouze profiler
- Vytváří a vizualizuje graf volání

Example

```
CREATE OR REPLACE FUNCTION district_name(dc varchar)
RETURNS varchar AS $$
BEGIN
  RETURN (SELECT name FROM districts WHERE code = dc);
END;

SELECT town FROM towns WHERE district_name(district) = ´Krasnoarmejsk´;
```

Example

```
CREATE OR REPLACE FUNCTION district_name(dc varchar)
RETURNS varchar AS $$
BEGIN
  RETURN (SELECT name FROM districts WHERE code = dc);
END;

SELECT town FROM towns WHERE district_name(district) = ´Krasnoarmejsk´;
```

BAD

WHY?

Example

```
CREATE OR REPLACE FUNCTION district_code(dn varchar)
RETURNS varchar AS $$
BEGIN
  RETURN (SELECT code FROM districts WHERE name = dn);
END;

SELECT town FROM towns WHERE code = district_code(´Krasnoarmejsk´);
```

Example

```
CREATE OR REPLACE FUNCTION district_code(dn varchar)
RETURNS varchar AS $$
BEGIN
  RETURN (SELECT code FROM districts WHERE name = dn);
END;

SELECT town FROM towns WHERE code = district_code(´Krasnoarmejsk´);
```

BAD

WHY?

Example

```
CREATE OR REPLACE FUNCTION district_code(dn varchar)
RETURNS varchar AS $$
BEGIN
  RETURN (SELECT code FROM districts WHERE name = dn);
END STABLE;

SELECT town FROM towns WHERE code = district_code(´Krasnoarmejsk´);
```

NOT TOO BAD

Proč jsou některé dotazy pomalé

- Planner Postgresu je jiný než planner Oracle, nesnaží se o redukci (opravu) neefektivně napsaných dotazů
- Postgres nemá žádné statistiky (odhady) pro funkce – Oracle, MSSQL dokáže odhadnout statistiku volání funkcí COALESCE, ..
- Konfigurace optimalize
  from_collapse_limit, join_collapse_limit, GEQO
- Postgres nemá plan cache
- Při startu dotazu dochází k načítání a kontrole některých dat ze systémového katalogu
- Intenzivní UPDATE, UPDATE nad tabulkami s hodně indexy je pomalejší (někdy výrazně) než na Oracle