PL/PGSQL

AN INTRODUCTION ON USING IMPERATIVE PROGRAMMING IN POSTGRESQL

Reactive.IO

Robert Sosinski

Founder & Engineering Fellow

AGENDA

PL/pgSQL: what it is and why it matters

Volatility: living in an uncertain data world

Upserts: a little update with a little insert

Triggers: keeping data consistent

Code Blocks: control when you need it

Summary: bringing it all together

Questions: fire away

WHAT IS PL/PGSQL

Procedural

Language

post**g**res

Structured

Query

Language

HOW PL MEETS SQL

WHY SQL IS NOT ENOUGH

With SQL, every statement must be executed individually by the database server

1. **Complex actions can require multiple round-trips**
2. **Intermediate results have to be marshaled and transferred**
3. **Multiple clients must implement redundant logic**
4. **Secure information might require exposure**
5. **Indexes and common filters might need complex logic**
6. **Consistency can require client-side interaction**

TWO WAYS TO PL/PGSQL

Stored Procedure

- Stored directly on the database server
- Can be used in queries and indexes
- Able to be activated upon trigger conditions
- May take input parameters and return values

Anonymous Code Block

- Sent to and parsed by the database when needed
- Able to write to the database but unable to return a value

MORE PL THAN JUST PGSQL

PL/Perl

PL/Python

PL/TCL

PL/Ruby

PL/Java

PL/V8

PL/PHP

DIVING HEAD FIRST

Secure access database

for a fictitious company

with high performance needs.

LET'S WRITE SOME FUNCTIONS

Open Postgres Terminal

DEALING WITH VOLATILITY

Volatility Categories

provide assumptions

the the query planner on
how to optimize a function call

IMMUTABLE

Characteristics

- Cannot modify the database
- Should not read from the database (side effect free)
- Similar inputs should always return the same output
- Can be used when creating indexes
- Database may replace a function call with a constant

Examples

- Our check_access_for_user function for user types
- The lower function for string types

STABLE

Characteristics

- Cannot modify the database
- Able to read from database
- Similar inputs should always return the same output on a per statement basis
- Can be used in index conditions, but not creation
- Can give caching benefits when called multiple times

Examples

- The current_timestamp and now function

VOLATILE

Characteristics

- Default volatility classification if none is specified
- Can read from and write to the database
- Can return different data on each call
- Can be used as trigger functions
- Planner will not make any assumptions for optimization

Examples

- The random and timeofday functions
- Sequence functions such as nextval and currval

LETS TRY TO WRITE SOME STABLE FUNCTIONS

Open Postgres Terminal

PULLING THE TRIGGER

Trigger Function

- The function to be run when a triggered event happens
- Written using PL/pgSQL or any other PL language
- Has special arguments available for control flow
- If returns NULL, the event is canceled, else it continues

Trigger Definition

- The event that causes the trigger to be run
- Always attached directly to the table or view
- Can be run BEFORE, AFTER or INSTEAD OF
- Can be run run for EACH ROW or EACH STATEMENT

LET'S PULL SOME TRIGGERS

Open Postgres Terminal

ANONYMOUS CODE BLOCKS

The Good

- You do not have to store the function on the server
- Can run arbitrary PL/pgSQL when it is needed
- Good for admin functions stored in your application

The Bad

- Cannot take arguments, so must be injected by your app
- Unable to return a value, but you can get feedback
- Must be parsed each time they are used

LET'S "DO" SOMETHING

Open Postgres Terminal

SUMMARY

Powerful: have direct imperative access to your data

Easy: syntax is familiar and simple to use

Fast: because you are never leaving the database

Secure: you do not need to send data back and forth

Flexible: store the logic on the database or in your app

CONDENSED SUMMARY

Direct access

to your data means

better control

of your business

QUESTIONS

Reactive.IO

Robert Sosinski

robert.sosinski@reactive.io