PL/Java:
Extend the power of your database

Charly Batista

PERCONA

I am Charly Batista

I’m the Postgres Tech Lead at Percona. I’m passionate about database and algorithmic efficiency. I’ve been working with Database and Development for more than 20 years always trying to get the best performance of the databases, data structures and algorithms... Other than that, I’m a Brazilian who lives in China for more than 6 years but currently located in South America.

You can find me at https://www.linkedin.com/in/charlybatista
or more contacts info by the end of this presentation

Agenda

- What is this talk about?
- Java on Postgres? Why?
- No way I’m using it!
- Ok, I’m using it. Now, what? (Recap)
- Questions

What is this talk about?

What is this about?

- Did you think it was a developer talk? Nope! It’s a database conference :D
- We will try to understand the reasonings for having and having not PL/Java in our stack;
- We’ll walk a bit on the implementation details. Not much, just the necessary to understand why and when we should use or not;
- Discuss about
  - security…
  - limitations…
  - costs…
  - etc.

What is this not about?

- Here I have a disclaimer!

  I don’t want to convince you to use or not to use PL/Java, but if you use I want you to know the possible issues and complications you might have, so you can use it within safe boundaries.

  Well, if you don’t want to use but don’t know why, it’s probably not a good idea to use it, and this talk may help you understand why! Or may convince you otherwise. You are warned :D

No live demo today but we have an implementation!

- 1st because I don't have good memories about live demos :(
- 2nd, this talk is part of a series of posts I started some time ago and can be found at
  https://www.percona.com/blog/postgresql-pl-java-how-to-part-1
  Also on my github https://github.com/elchinoo/blogs
- The post series is focused on the implementation side with few examples;
- The 3rd part is coming soon with an example on how to use PL/Java to integrate Postgres and HashiCorp Vault;
- Probably a final one will come after with a summary and the content of the discussion we are having here today.
- Yeah, it’s more of a discussion than a presentation, so feel free to ask any question or make comments while we walk through!

Java on Postgres? Why?

Java on Postgres? Why?

- We need to firstly ask “Why should I use anything other than SQL on my DB”?
  - Extend the capabilities of the database (IO, Networking, encryption, etc);
  - Enforce referential integrity beyond what can be done using SQL semantics;
  - Advanced pattern recognition;
  - Advanced support for technologies like XML, JSON, YAML, O/R mappers, etc;
  - Summary is, solve problems that are too expensive, complex, impractical, etc, for a declarative language like SQL;

Java on Postgres? Why?

- Ok, I got why we may need something else other than SQL. Why PL/Java?
  - 1st one is already having Java stack in-house;
    - Java was the 2nd most popular language in 2021 and probably the most used by large enterprises;
  - Optimized for performance and security.
    - Minimize round trips, reduce network overhead with “local” computation, etc;
    - It might also be a problem, we’ll discuss it later when talking about resources;

Ok, but how does it work?

- Some behavior is implemented in Java using familiar Java objects;
- Some is implemented in “C” with an object-oriented approach using C structs;
  - For example, the class ***Type***, which extends a ***TypeClass***, which inherits from ***PgObject***;
- PL/Java uses the standard java.util.logging.Logger:
  - Logger is hardcoded to a handler that maps ***log_min_messages*** to a valid Logger level;
  - Logger outputs all messages using the backend function ***ereport()***;
  - Newer versions of PL/Java can make usage of of ***log_min_messages*** and ***client_min_messages***;
  - Logger uses the Postgres settings at the time PL/Java was first used in the current session!

Ok, but how does it work?

- PL/Java uses JNI, but what matters for us here is that:
  - It has one JVM/connection, not one thread per connection inside the same JVM;
    - The overhead of multiple processes is already present anyway;
    - Improves and simplifies the security management;
      - Separate JVMs gives you a much higher degree of isolation;
    - New JVM versions have significantly reduced footprint and startup time;

Ok, but how does it work?

- PL/Java uses JNI, but what matters for us here is that:
  - PL/Java function can be annotated with a level of “parallel safety”
    - *UNSAFE* by default, which means the query cannot be parallelized;
    - PostgreSQL parallel query processing uses multiple OS processes, so…
      - *If a PL/Java function is PARALLEL SAFE and is pushed by the planner to run in the parallel worker, each new process will start its own JVM!*
    - ***parallel_setup_cost*** should be increased to add this new cost!

Ok, but how does it work?

- PL/Java has its own JDBC internal implementation:
  - This driver is written directly on top of the PostgreSQL internal SPI routines;
  - This driver is essential to reuse the same transactional boundaries between triggers and function calls associated with the caller;
  - Complex types and sets are passed using the standard JDBC ResultSet class
  - It simplifies the development as the developer can interact with the database as it is just another JDBC call;

No way I’m using it!

What are the cons?

- It’s not easy to install into a Postgres server;
- One JVM per connection…
  - Actually we can end up with many VMS if using parallel queries!
  - More resources consumption, specially memory, but be careful with CPU!
- The JVM is created at connection time, which increases the latency
  - Can be improved with connection poolers but it has its own drawbacks;
- The internal JDBC comes with its own issues like complexity to use cursors, etc;

What are the cons?

- Increase in logic complexity on database side;
- Can cause unexpected behaviour
  - For example, if something is logged wit ERROR log level the connection will be terminated;
- While it can enforce security, if not well designed can be a bigh threat:
  - PL/Java creates two named languages, java and javaU. The former is declared as TRUSTED with all the limitations implied to trusted languages in Postgres but the latter is UNTRUSTED and can be granted access to filesystem and even for network connections!
- While it has very good data processing performance it’s not easily scalable.

Ok, I’m using it. Now, what?

What now?

- Resources consumption. Discussed many times above but you know, many JVM's!
- Make sure you understand the permission models in both Postgres and JVM…
  - You can use ***SECURITY DEFINER*** instructions and also the good old ***GRANT*** on Postgre;
  - Make sure the JVM has given all the necessary permissions to your method/class/user;
- Default to the ***java*** language instead of ***javaU***. Use the later one carefully!
- Make sure you have increased ***parallel_setup_cost*** accordingly…
  - You don't want to end up with 3+ JVM's per connection!
- Avoid creating too many callbacks, savepoints, complex tree/graph calls;

What now?

- Avoid the temptation to open network connections, process large files, do anything that would exponentially increase the latency.
  - You are inside of a DB transaction and we don’t want it to implode our database!
- It can still be of good usage:
  - Migration from proprietary database that has some embedded Java code;
  - Cryptographic features, including digital signatures, data anonymization, obfuscation, etc;
  - Authentication and authorization features including integration with external tools;
  - Complex data processing, complex data type processing;
  - Many more!

What should we take from here?

- PL/Java like any other language isn’t evil, it’s just another language that can be put for good or bad usage. Be careful, spend time on the design of the solution, understand the problem you are trying to solve, understand the pros and cons of the language, the stack and human resources available and if after that it still seem a good fit, then it’s time to redo the process, compare with other languages and drop us a message, you may have missed something on the process :D

  No, I mean, it can help solving the problem you have and a good POC might help to give you the answers and confidence you need to use or reject it.


Are you passionate about Open Source?!
We're hiring!
Join us!
#RemoteWork
percona.com/careers

Questions?

I will be at the Percona's booth if you have more questions
You can find me on LinkedIn at https://www.linkedin.com/in/charlybatista
Or drop me an email at charly.batista@percona.com

Thank You!