Peloton UDF Final Presentation

SeungHyun Lee, Yang Zhang, Zheyuan Bu
Proposed Goals

- 75%: Basic support for C UDFs
- 100%: Test code, benchmark, basic support for stored procedure (pl/pgSQL)
- 125%: Make stored procedure support reliable (not done)

Current Progress

- Support for C UDFs
- Support for PL/PGSQL
- Testing Code - SQL scripts and expected outputs
- Benchmark
  - C vs PL/pgSQL UDF functions
  - stored procedure vs prepared statement with batch update
Overview of executing a UDF

- **Registering function**
  - Function registration request → Function Validation → Insert an entry into the catalog

- **Calling C UDF**
  - Create a UDF expression node → Evaluate the expression → Go to the function manager → Directly call the function

- **Calling pl/pgSQL UDF**
  - Directly call function handler → Parse the pl/pgSQL code → Execute the code via SPI
Testing

● Issue: UDF components are not easily separable
  ○ Components are closely interleaved.
  ○ C UDF functions are called directly using the function pointer.
  ○ SPI layer is dependent on the query execution code.
  ○ PL/PgSQL handler is a C UDF function that uses SPI layer.

● Our approach
  ○ Write SQL scripts that uses UDFs and compare with the expected output.
  ○ Make UDFs be called at different places.
  ○ Use different types of input/output data.
DEMO
UDF Experiment - C vs pl/pgSQL

- **String operations**: concat_text, replace_vowels
- **Float calculations**: calc_tax
- **Integer calculations**: integer_manipulate
- **SQL Queries**: item_sales_sum
- **Loops & control flow**: countdown, fib
UDF Experiment - Stored procedure vs Prepared Stmt

![Graph showing comparison between stored procedure and prepared statements in batch and individually for insert and update tests.](image-url)
Future Direction

- Remove the dependency of postgres catalog system
- Add the native support for pl/pgSQL (non-sandboxed).
  - Instead of registering the handler as a C UDF, directly process UDFs to make it faster.