Supporting UDFs in Peloton

By
Haoran Wang
Nasrin Jaleel
Prashasthi Prabhapar
Proposed Goals

● 75% Goal: Registering a UDF
  ○ CREATE OR REPLACE FUNCTION increment (i int) RETURNS integer AS $$
  ○ BEGIN
  ○   RETURN i + 1;
  ○ END;
  ○ $$ LANGUAGE plpgsql;

● 100% Goal: Simple Add-One example
  ○ Ex: select increment(1);
  ○ Ex: select increment(a) from table;

● 125% Goal: Support complex constructs in UDF
  ○ Support if-else statement
  ○ Support Recursive
CREATE OR REPLACE FUNCTION increment (i int)
RETURNS integer AS $$
BEGIN
    RETURN i + 1;
END;
$$ LANGUAGE plpgsql;
CREATE FUNCTION statement

**Parser**
- Parses the CREATE FUNCTION statement into Postgres parse tree object
- Converts the parsed Postgres object to Peloton parse tree object

**Optimizer**
- Translates the Peloton parse tree object into the Peloton Plan object
- Bundles the necessary fields into a form that can easily be used with the executor
- Necessary fields include: function name, udf language, input parameter names, input parameter types, return type, function body

**Executor**
- Inserts the UDF into the catalog - pg_proc
- Function body inserted as a string

**Pg_proc Catalog**
- Stores all the registered UDFs
- Created during Bootstrapping
- Supports querying by oid and function name
UDF Invocation Example

select increment(2);
  -> Returns 3
Select increment (2);

**Parser**
- Parses the select UDF statement into Postgres parse tree
- Convert the parsed Postgres object to Peloton parse tree object

**Optimizer**
- Translates the Peloton parse tree object into the Peloton Plan object
- Identifies whether the function invoked is a built-in function or a UDF

**Executor**
- Retrieves the UDF from the catalog
- Send the UDF to the UDF_handler
- Returns the result returned by the UDF handler

Output: 3

**Pg_proc Catalog**

**UDF Handler**
How to implement UDF?

● Link the library of Postgres
  ○ Clean Interface?
  ○ Transformation!

● Include all files under Postgres UDF directories
  ○ Global variables & Dependencies
  ○ Transformation Again

● Build it from scratch
  ○ Start by small steps
How to implement UDF?

- **Link the library of Postgres**
  - Clear Interface?
  - Transformation!

- **Include all files under its plpgsql directories**
  - Global variables & Dependencies
  - Transformation Again

- **Build it from scratch**
  - Start by small steps
UDF Implementation & Interface

● Build the Yacc and Lex for UDF
  ○ If-else & return_stmt are supported

● UDF Handler is an interpreter
  ○ climb the parse-tree, execute and return the final value
  ○ UDF_Handle::Execute(vector<Value> values)

● UDF_Stmt is the base class with virtual method Evaluate()
  ○ UDF_Return_Stmt & UDF_IFELSE_Stmt are the children class
If-else Example

CREATE OR REPLACE FUNCTION OddEven (i int)
RETURNS integer AS $$
BEGIN
  IF i % 2 = 0 THEN
    RETURN i;
  ELSE
    RETURN - i;
  END IF
END;
$$ LANGUAGE plpgsql;
UDF Handler Implementation & Interface

- Build the Yacc and Lex for UDF
  - If-else & return_stmt are supported
- UDF Handler is an interpreter
  - climb the parse-tree, execute and return the final value
  - UDF_Handle::Execute(vector<Value> values)
- UDF_Stmt is the base class with virtual method Evaluate()
  - UDF_Return_Stmt & UDF_IFELSE_Stmt are the children class
Execution of SQL Expression

- Take “Select i + 1;” as the example
- The Token “i” is replaced by its value, say 5
  - “Select i + 1;” -----> “Select 5 + 1;”
- The replaced string is executed as normal SQL statement by
  - traffic_cop.ExecuteStatement(sql_expr)
    - Which does parse(), plan() and execute()
- This means, an identical plan is generated every time
- [TODO] Value injection happens after the plan is generated
  - So that “Select i + 1;” is parsed and node “i” is replaced by its value in every invocation
Future Work

1. Supporting more functionalities within the UDF - requires more work on yacc and lex side
2. Supporting transactions in expressions
3. Function syntax and argument validations
4. Storing function pointer (UDFhandle) in the catalog
5. Performance overhead