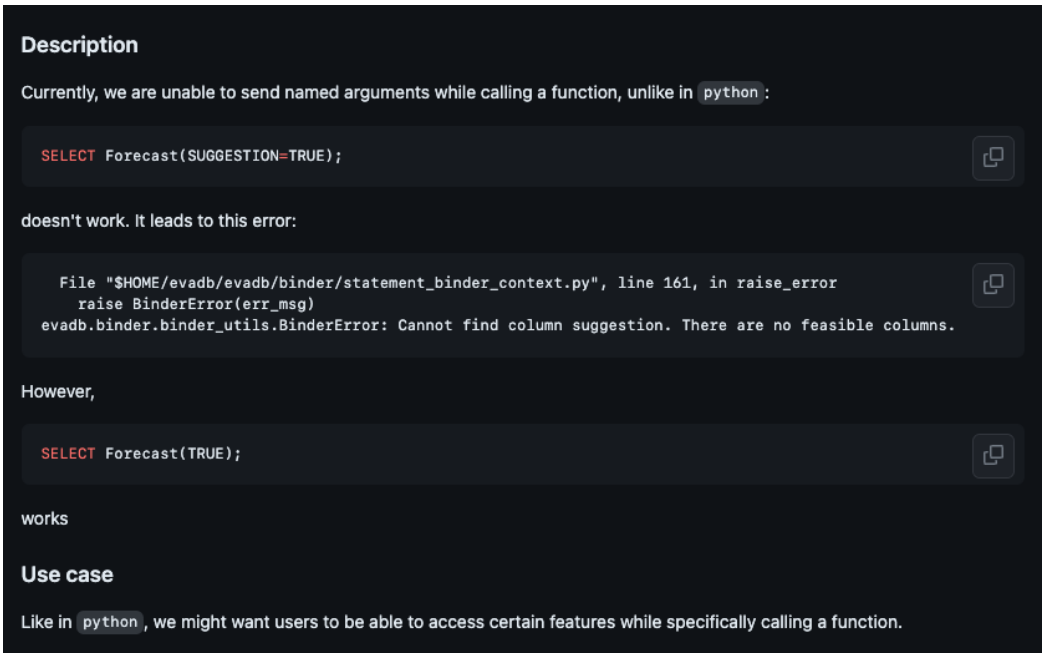
CS6422 Project2 Final Report

Tiancheng Xu

https://github.com/ricardoxu6/EvaDB_Pass_Argument_In_UDF

In this project, I am trying to solve an issue on Github regarding a user experience. This issue is about passing arguments in a user defined function in EvaDB. I made modifications to files of backend codes to solve the issue, but I have not thoroughly tested it so I did not make a PR. Nevertheless, I believe my ideas and the currently made changes are a good progress. It might successfully solve the issue because when I tested it, the resulting workflow meets what I wish to see. I will explain it in the later part of the report, including what effects I want to achieve and what are the testing results. My GitHub repository only contains the files I have made changes to. To test the result, copy the code to the same files in EvaDB source code.



Description

Currently, we are unable to send named arguments while calling a function, unlike in `python`:

```
SELECT Forecast(SUGGESTION=TRUE);
```

doesn't work. It leads to this error:

```
  File "$HOME/evadb/evadb/binder/statement_binder_context.py", line 161, in raise_error
    raise BinderError(err_msg)
evadb.binder.binder_utils.BinderError: Cannot find column suggestion. There are no feasible columns.
```

However,

```
SELECT Forecast(TRUE);
```

works

Use case

Like in `python`, we might want users to be able to access certain features while specifically calling a function.

To fix this issue, I first try to understand the workflow while calling a user-defined function. What I can conclude is that, if we want to find a general solution to the issue, it must modify within the backend code of EvaDB but not only one specific function's implementation file. The intuitive is to look at where the error is raised, in other words, the statement_binder file. Inside it, there are many functions defined to bind columns, tuple expressions, function expressions, and so on. The issue is not on the logic how EvaDB bind them. Instead, it is because EvaDB cannot recognize what inside the parentheses as an argument parameter. It recognizes SUGGESTION as a column name, so it further call the bind_column() function which raises the error. In other words, now EvaDB treats it like a WHERE clause. It treats SUGGESTION, the parameter name,

as a column name, and try to execute the logic to find WHERE SUGGESTION = TRUE(find records whose value of the column matches the specified one).
With the help from Rohith Mulumudy, I started to look at evadb.lark. It defines the grammar specification for the parser. In this file, I found that there is no specific grammar for named argument. Currently, EvaDB will treat SUGGESTION= TRUE as a function_arg with a type of expression. Further, expression can be a predicate, and a predicate can be an expression_atom(full_column_name, constant, and so on) or a binary comparison between two predicates. Then we can understand how EvaDB currently handles SUGGESTION= TRUE. First, it's a function_arg and an expression. The expression is a predicate, which compares two expression_atom(a binary comparison predicate). One is SUGGESTION, which is treated as a full_column_name, and another is TRUE, which is treated as a constant of Boolean. It is totally the same as how EvaDB handles where clause. If the query is WHERE SUGGESTION = TRUE, EvaDB recognizes it as an expression and match the expression's grammar as a "Full Column Name. = Constant" type. So we need to distinguish the named argument expression from this.

A named argument should be in the format like "[parameter name] = [value]", but such a format is also a valid binary comparison predicate and thus a valid expression. If we exclude this from the valid formats of "expression", then WHERE clauses cannot work anymore. If we want to add a grammar *named_parameter: uid "=" constant,* EvaDB has no way to distinguish it from an binary comparison predicate. Here for convenience, we defined it to be "*named_parameter: uid "EQUALS" constant* ". In other words, if the statement is "SUGGESTION = TRUE", it is a binary comparison predicate and EvaDB will treat SUGGESTION as a column name. If we want it to be a named parameter, we should write it as "SUGGESTION EQUALS TRUE". Though it does not totally satisfy the grammar of named parameter as that in Python, but it gives a relatively simple way to distinguish named parameters in EvaDB. Users only need to be aware to use EQUALS when he wants it to be a named parameter and use "=" when he wants it to be a column name. Currently, function_args is composed of some function_arg, so if we want to support named_parameter as a function_arg, we simply add named_parameter as a possible option of function_arg. Now we have *function_arg: constant | named_parameter | expression*. This defines the grammar so that EvaDB can recognize our expression as a named_parameter.

```
1  SELECT Forecast(SUGGESTION EQUALS TRUE);
```

Then we need to modify the parser to handle the named parameter. We assume that currently EvaDB can execute Function(1,2,3), then the intuitive is to modify the code so that if we have an expression like Function(a EQUALS 1, b EQUALS 2, c EQUALS 3), EvaDB can recognize that the function_args is composed of 3 named_parameter expressions, and 1,2,3 are the three actual values needed for Function(). EvaDB needs to parse it to Function(1,2,3). Then EvaDB will just execute the logic to handle Function (1, 2, 3). If we use the example of our Forecast () function, Forecast (True) can work because in the implementation file of Forecast(), the line 81 of forecast.py is:

if len(data) == 0 or list(list(data.iloc[0]))[0] is True

Here, the variable *data* is the expression inside the parentheses of Forecast(). And if we translate this line of code, it means that if there is nothing inside the parentheses, or if there is only one expression which is True, it will make the variable SUGGESTION used in forecast.py to be true. It defines a valid way to handle the parameter we pass in. This helps explain our assumption that "Function (1, 2, 3) should be a valid call". If our statement is Forecast(True, 6422), forecast.py cannot find a way to handle the two values passed in function call, this call is invalid. Then even if we can parse Forecast(SUGGESTION EQUALS TRUE, X EQUALS 6422) to Forecast(True, 6422), since Forecast(True, 6422) itself is invalid, this conversion is meaningless.

Then obviously we should add some logic when EvaDB recognizes the grammar of named_parameter while parsing the query. We only meet named_parameters when we are processing function_args. And the logic is defined in _functions.py. If the child contains the information of function_args, EvaDB will visit it to get all the arguments.

```
            function_args = [TupleValueExpression(name="*")]
    if isinstance(child, Tree):
        if child.data == "simple_id":
            function_name = self.visit(child)
        elif child.data == "dotted_id":
            function_output = self.visit(child)
        elif child.data == "function_args":
            function_args = self.visit(child)
```

My modification is done in the function_args() function in _functions.py. This function will get all the arguments in the function call. When it loops through tree.children, if for one child, child.children[0]'s data is "named_parameter", it means that this child node, which contains the information of one function_arg, satisfies the grammar of a named_parameter. And child.children[0]'s children will contain 2 elements, which are the named parameter's name(SUGGESTION) and its value(TRUE). Since we want to parse Forecast(SUGGESTION = TRUE) like Forecast(TRUE), the only thing we need to do is to pop the first element in child.children[0]'s children.

To test if this change successfully converts Forecast(SUGGESTION = TRUE) to Forecast(TRUE), in the test file for select statement, I tried to call both the two functions and see what EvaDB parses them to after handling the function_args. At this step, where we already get the function expression after processing the function_args, in both calls, the variables end up in the same state. For instance, the variable function_args only has one ConstantValueExpression object, which is TRUE. So, the changes I made realized the goal to make EvaDB recognize a named_parameter and convert it by giving up the "name" part and only leaving the "value" part.