On 'YeSQL: "You extend SQL" with rich and highly performant user-defined functions in relational databases'

Telman Hüseynli
University of Bamberg

ABSTRACT

This paper presents the study conducted on "YeSQL: 'You extend SQL' with Rich and Highly Performant User-Defined Functions in Relational Databases." The motivation behind this study stems from the increasing demand for incorporating user-defined functions (UDFs) into SQL to enhance its functionality and performance.

This paper outlines the proposal of an innovative approach that enables the smooth integration of rich and highly performant UDFs into SQL. This proposed approach is YeSQL: which extends SQL functionalities with efficient user-defined functions (UDFs) in relational databases. The sources and materials utilised in this study involve an analysis of existing approaches in the UDF implementation, an investigation of current alternatives by comparison, and the design and application of the UDF extension strategy. The study also addresses the limitations and difficulties encountered when integrating UDFs with SQL in relational databases.

The outcomes of this research enable users to extend SQL with custom functions and open up new possibilities for data processing and analysis. The evaluation demonstrates YeSQL's improved performance compared to former software and states its capacity to scale and efficiently manage complex tasks with large-scale data. This study also provides the foundation for future studies and innovations in UDF extensions.

1 INTRODUCTION

User-defined functions (UDFs) have gained considerable interest due to their capacity to improve the efficiency of SQL queries. This paper delves into the topic of "YeSQL: 'You extend SQL' with Rich and Highly Performant User-Defined Functions in Relational Databases"[10]. Because all of the following material is based on [10] and [16] citations are not supplied clearly after each text and figure. It sums up the theory, and the author's innovative technique to overcome the related challenges. The integration of UDFs with SQL has considerable importance as information technologies improve. Recently, the industry is focused on analysing big datasets and obtaining useful information. This focus makes custom functions in SQL crucial. Former studies in this area have contributed various discoveries for integrating UDFs, nonetheless, these discoveries were limited to achieving rich functionality and high performance. Python is often the favoured programming language among UDF developers due to its usability and expressiveness for UDF implementation. Most database systems work with Python UDFs; these systems have several challenges to offering good usability, expressiveness, and efficiency. However MonetDB[2] and PostgreSQL[3] are extensible with Python UDFs, they are not able to determine the data type of the query results. They require a static return schema and lack standard boosters like Just-In-Time (JIT) compilers.

YeSQL, as an SQL extension, provides more usable, portable, expressive functionalities and better performance as Python UDFs. Both the server-based and embedded DBMSs can be extended with YeSQL. The features such as scalar, aggregator, or table UDFs are now fully integrated into the YeSQL extension. The users are able to implement complex algorithms with YeSQL functions. The objective of this research is to integrate user-defined functions that have featured functionalities and better efficiency versus relational SQL queries. The authors present an innovation by considering the previous methodologies and the necessary delimitations. They aimed to overcome the existing limitations and provide users with easy implementation of UDFs into SQL, granting them efficient query capabilities. Their study addresses the implications of incorporating UDFs into SQL and techniques to resolve the limitations.

On the other hand, the authors don't focus on the details of DBMS or the specific optimization techniques for UDF execution. Instead, they aim to extend the SQL framework in a functional and efficient way. This proposal provides the technical overview of extending SQL with UDFs, highlights the significance in the relevant field, and presents the research aim and the author's solution. The next paragraphs discuss the related work, the methods, and the evaluation of the proposed technique.

2 FORMER APPROACHES

Many studies have explored the implementation of UDFs by discussing the limitations and problems related to their usability. The existing research approaches have supplied a foundation of study for improvements in SQL UDF support. However, alternative database systems were needed to overcome the incompatibility between the evaluation of UDFs and relational DBMSs. The authors demonstrate that the performance of YeSQL is superior compared to Python UDFs optimised with the current compilers or source-to-source compiling techniques.

The recent data management techniques focus on implementing reusable functionalities under user-defined operators. Many data management platforms support Python UDFs. Additionally, there are studies to improve the performance of Python UDFs by speeding up the compile process. GraalPython[1] enables the Just-In-Time compilation by producing bytecode files for Java Virtual Machine. Its interpreter uses Truffle language. Numba[12] presumes data types and generates array-structured data to produce efficient machine code. It also enables the Just-In-Time compilation. Pyston[13] uses low-level components to achieve a monitorable JIT Python compiler. It is similar to the technique of implementation of Python in C. However, these compilers address particular challenges in improving the performance of UDF integration, they have other drawbacks such as compatibility issues, being restricted to a few libraries, or providing primarily experimental analysis. Moreover, Cython [7] enables the users to develop their program in Python and it compiles the program into C. Its performance is improved due to low-level language advantages. Static typing requirements and

compatibility restrict it to reach satisfactory efficiency. Nuitka[4] also translates Python code to the low-level language C++. Since the compilation process consumes a lot of time, it is disadvantageous to implement UDFs with many calculations.

There are several studies to advance data processing systems by resolving the drawbacks of Python UDFs extension. Tuplex[15] is a compiler, particularly for Python that enabled Just-in-Time compilation. However, its functionality is not sufficient for current industrial requirements. The study by Technical University Ilmenau provides valuable insights related to the challenges of Python UDF extensions [11]. It analyses some of the aforementioned compilers by comparing and studying the significance of vectorization, compilation, and parallelisation in speeding up the run time of Python UDFs. Another proposal studied the performance of executing SQL translation of whole Python programs [8]. It is observed that SQL queries are more optimised when the parallel execution of data processing is dynamically divided into tuples. However, the readability of SQL codes was difficult for developers. Various research studies focus on the conversion of UDFs to SQL queries and analysed obstacles in performance improvement. These studies show the overhead of UDF execution is reduced when it is optimised by SQL engines. On the other hand, some studies propose UDF fusion techniques to reduce execution overhead. UDF fusion optimises execution by fusing operations of multiple functions into one call. Current works with this technique are compatible with only a few libraries. YeSQL study extends these researches and enables UDF fusion with all Python libraries.

3 METHODOLOGY

3.1 Design and Integration

The server-based DBMS and the compact embedded DBMS are frequently utilised with Python UDFs.Figure 1 illustrates architecture's essential parts on the basis of both DBMS concepts. YeSQL can be added to server-based as an innovative UDF extension. There are two user roles in the extension design: application developers and UDF developers. Application developers implement queries for the front-end functionalities of the program. UDF developers design custom query functions documented in the database management system. The YeSQL functionalities are translated into SQL queries to be executed. This task is operated by the layer called textitthe Connection and Function Manager (CFM). The UDF developers are able to use flexible SQL syntax and let the execution run with SQL optimizers. The architecture of the CFM layer consists of the parser, the code producer, and the function manager. UDFs are registered in the database management system by the function manager. UDFs written in Python are recorded in the designated UDF folder of the extension. They are classified into scalar, aggregate, and table functions. In the server-based DBMS, the function manager assumes the responsibility of submitting user-defined functions (UDFs) to the DBMS. The CFM parser verifies the accuracy of the syntax of UDFs and converts them into SQL. The CFM code generator provides platform-specific operations for UDFs with dynamically typed data. It generates the code according to particular data types and functionalities and enables seamless integration of UDFs.

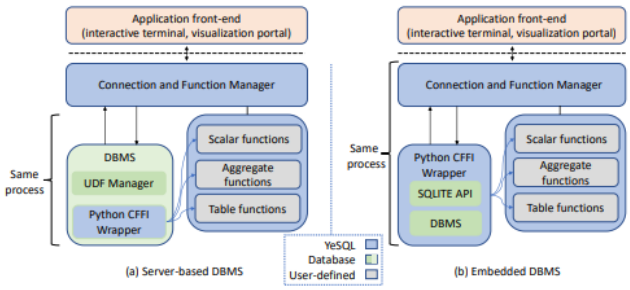

Figure 1: Architecture Alternatives for YeSQL [10]

In the embedded DBMSs, YeSQL extension is added by utilising a programming interface. The CFM operates UDFs as callback routines, promoting easy communication between the executed queries and Python UDFs. In this instance, the functioning of the CFM layer and the integrated DBMS takes place within a common runtime system. Python CFFI[14] wrapper enables uncomplicated data interchange between the SQL queries and Python UDFs.The SQLite API provides native support for extended SQL functionality through C UDFs.The Python CFFI wrapper serves as a bridge between Python and the database engine, facilitating the execution of UDFs. The API provides integration of extension through C UDFs, enhancing the capabilities of YeSQL in an embedded DBMS environment.

YeSQL is a DBMS module that provides high compatibility with widely used operating systems. In the server-based DBMSs, YeSQL improves the performance of the processing framework. In this case, MonetDB uses vectorized execution. A single instruction with multiple elements efficiently transfers big data over the low system API to execute in parallel. In the event of an embedded database, YeSQL makes use of the API's internal streaming architecture to process the data efficiently. The API enables YeSQL to retrieve and operate on data concurrently in co-routines using Python generators. Moreover, YeSQL's design supports UDFs implemented in other programming languages that have function interfaces for not native instructions and a tracing JIT compilation.

UDFs in YeSQL can yield multiple output formats by dynamically specifying data types based on the input operations. The dynamic type declaration supports polymorphic functions. In embedded DBMSs, polymorphism is supported using a virtual table interface. In server-based DBMSs, the output format is not specified beforehand and is temporarily stored in memory by Python Loader.

3.2 Functionality

In this paragraph, the core functionalities of YeSQL are stated briefly. The primary purpose of YeSQL design is to enable developers to use SQL methods easily and efficiently. YeSQL adds new syntax features and functionalities to standard SQL services. The Python UDFs are employable by current common Python frameworks. The data processing in YeSQL leaves filters and joins to be executed by database management systems as convenient. As mentioned earlier, it is able to yield flexible output types by regulating the data according to system requirements.

YeSQL presents particular syntax components to advance the adaptability and efficiency of UDFs. For example, YeSQL offers additional options for common SQL aggregate functions. It also provides polymorphic table methods which produce output that is identical to typical tables during query execution. It has scalar functions that perform tasks per row and can produce multiple element structures per row-specified output schema (*textwindow*). With such functions, complicated data processing tasks can be performed on a per-row basis and the result can be produced in compound structures.

YeSQL offers UDF fusion. It creates a fused function to invoke multiple UDFs with one call. JIT compilation enables optimisation of the fusion by analysing the data dependencies of UDFs during runtime. The corresponding example is shown in the Figure 1 below.

```
def multiply(input):
    return len(input)
def add(input):
    return add+1
#If they are fusable according to the query plan of a specific query,
#the CFFI wrapper creates a wrapper UDF like the following:
@ffi.def_extern()
def fused_add_multiply(input, count, result):
    for i in range(count):
        val = ffi.string(input[i])
        result[i] = add(multiply(val))
        return 1
```

Figure 2: UDF Fusion Example [10]

Another important feature of YeSQL is syntax inversion. YeSQL's design enables users to easily implement chained UDFs with simpler syntax and adaptable parameters. CFM table translates compact instruction syntax to standard subqueries. The corresponding example is shown in the Figure 1 below.

```
select * from
  sample(10000, 'select * from
         rowidvt(''select * from
           xmlparse(''''select xml from table'''')'')');

sample 10000 rowidvt xmlparse select xml from table;
```

Figure 3: Syntax Inversion Example [10]

In summary, YeSQL offers rich support for polymorphic Python UDFs, UDF fusion for parallel data processing, and syntax inversion for the convenient composition of UDFs, enhancing the flexibility and expressiveness of data processing tasks.

3.3 Improved Performance

There are two primary costly operations in data flow between Python and SQL: a significant amount of function calls and transition of data structure. Both operations create considerable overhead. The authors aimed to overcome these in YeSQL's design with several approaches such as vectorized execution, tracing JIT compilation, parallelism, UDF fusion, and stateful UDF execution. The authors show that the order of these approaches is an important factor in performance rate when they are applied together. Particular alignment according to system requirements can advance the efficiency of query executions. The study states that such a combination can reach a 33-fold improvement in comparison with isolated Python interpreters. The particular combination is 10 times faster than execution with embedded vectorization.

Tracing Just-In-Time: JIT compilation means compiling the code into the low-level language during execution. JIT compilation can be achieved with method-based and tracing techniques. Method-based JIT compiles the functions when they are invoked. On the other hand, Tracing JIT focuses on instruction sequences or loops that are often executed. YeSQL architecture supports tracing JIT compilation and optimises repeated calculations. The authors' solution made use of PyPy[9], a dynamic Python compiler. PyPy's tracing compiler determines feasible execution for big instruction sets that are called multiple times during execution.

PyPy has many noteworthy advantages for easy and efficient UDF execution. (a) It is compatible with the standard Python library and commonly used frameworks. It is regularly updated to adapt more libraries. (b) It does not require a static schema for outputs and optimises the execution of the program during runtime. (c) It successfully manages thrown exceptions by YeSQL UDFs. (d) It enables effective translation of the code to low-level language by using a foreign function interface. Moreover, it is able to utilise CPython for the functionalities that PyPy is not compatible with. *Easy communication with DBMS:* YeSQL has a smooth interface with Database Management Systems (DBMS). The Foreign Function Interface encapsulates UDFs [14]. The CFFI acquires data during the runtime by means of pointers to cdata objects, thereby avoiding any redundancy of data. *UDF Fusion:* The study states that implementing simple and reusable UDFs can enhance operational efficiency, particularly in the context of text mining. The processing speed is accelerated by UDF fusion through the reduction of instruction call overhead. When it is possible to combine the logic of two distinct functions, a merged UDF is created to optimise pipeline execution by leveraging a C Foreign Function Interface (CFFI) wrapper. *Parallelism:* Parallelized execution leverages the speed and scalability of computer programs. However, concurrency is not possible within Python interpreter, since the Global Interpreter Lock (GIL) restricts the acceleration of runtime. The interpreter lock lets one thread run instructions. The instantiation of Python objects also invokes GIL, thereby resulting in more overhead. PyPy is faster in this matter. It uses GIL more efficiently. The C Foreign Function Interface (CFFI) accelerates execution by performing a gradual and unsynchronized release when unlocking the interpreter. *Stateful UDF:* The YeSQL provides stateful UDFs while the majority of DBMSs are only compatible with stateless Python UDFs. The stateful UDFs are able to hold static values as an internal state that is available between invocations. The stateful UDFs offer potential opportunities for the advancement of query algorithms. In the embedded DBMSs, UDFs are usually stateful by the standard. In server-based DBMSs, states can be accessed across other rows of inputs and UDFs.

4 COMPARATIVE ANALYSIS

The efficiency and functionalities of YeSQL were assessed via distinct tests with distinct data sets and environments. The efficacy of

YeSQL's workflow was evaluated through the Zillow and Flights pipelines, alongside the OpenAIRE[5] text-mining pipeline. Regarding pipeline performance, YeSQL utilising tracing JIT on MonetDB demonstrated better outcomes compared to other approaches, it is faster with concurrent execution enabled.

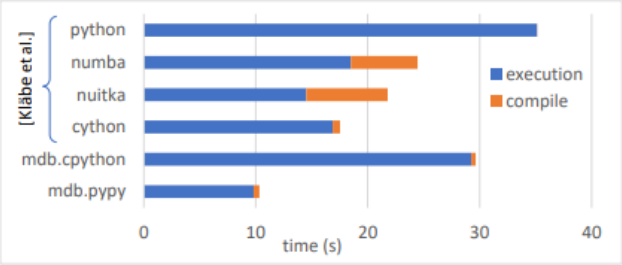

Figure 4: The impact of incorporating JIT into UDF execution [10]

This study investigated comparatively the YeSQL extension and standard SQL functionalities. The authors prove that YeSQL has segments with improved optimization owing to the UDF fusion, which resulted in fewer execution durations. The conducted experiments have additionally underscored the benefits of incorporating seamless integration and the influence of data adjustments on performance in general. The research was primarily concentrated on assessing tracing JIT compilation. The experiments show that YeSQL utilising PyPy (as the tracing JIT) and CFFI (for C conversion) results in a threefold acceleration versus CPython. The study conducted a comparative analysis of YeSQL and alternative DBMSs that support Python UDFs. The results indicated that the YeSQL extension has better performance when compared to alternatives such as PostgreSQL, Spark, dbX, Pandas, Tuplex, and NumPy. The corresponding experiment results are shown in the Figure 4 below.

Furthermore, the conducted experiments investigated the impact of stateful User-Defined Functions (UDFs). The Stateful UDFs use segments that compiled former invocations and recorded them. This technique leads to fewer execution durations. To evaluate the performance of UDF fusion, the authors measured the performance of fused execution of multiple UDFs by comparing them with standard execution. The corresponding experiment results are shown in the Figure 5 below. This research involved profiling the space consumption of YeSQL execution when combined with PyPy, Tuplex, and PySpark[6]. The analysis revealed that YeSQL utilises less memory in comparison to the alternative solutions such as PySpark, and Tuplex.

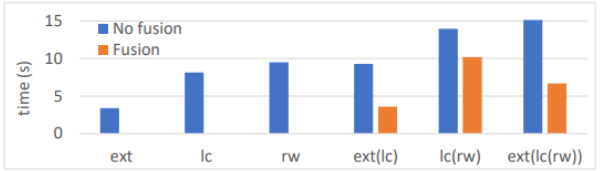

Figure 5: Impact of UDF Fusion [10]

To conclude, the experiments proved the benefits of different optimization and integration techniques, and the advantages of utilising stateful UDFs and tracing JIT to enhance execution times and scalability.

5 CONCLUSION

In conclusion, the study presented an innovative proposal for an extension of SQL that boasts extensive UDF support with both server-based and embedded DBMSs.The syntax is to optimize the productivity of UDF developers by providing a featured relational framework for constructing multilayered algorithms. YeSQL surpassed comparable implementations in terms of efficiency by utilizing innovative techniques such as tracing JIT compilation, parallelism, UDF fusion, stateful UDFs, and seamless database integration. The implemented solution has been effectively deployed in the real-world environment and by data engineers from several fields and companies such as OpenAIRE. Future objectives include providing assistance for federated and heterogeneous systems, as well as enhancing UDF fusion and query rewriting optimization.

REFERENCES

[1] 2022. GraalVM. https://www.graalvm.org/ Last accessed 13 Juni 2023.
[2] 2022. MonetDB. https://www.monetdb.org/ Last accessed 13 Juni 2023.
[3] 2022. PostgreSQL. https://www.postgresql.org/ Last accessed 13 Juni 2023.
[4] 2023. Nuitka the Python Compiler. https://nuitka.net/ Last accessed 13 Juni 2023.
[5] 2023. OpenAIRE. https://www.openaire.eu/ Last accessed 22 Juni 2023.
[6] 2023. PySpark. https://spark.apache.org/docs/latest/api/python/ Last accessed 22 Juni 2023.
[7] Stefan Behnel, Robert Bradshaw, Craig Citro, Lisandro Dalcín, Dag Sverre Seljebotn, and Kurt Smith. 2011. Cython: The Best of Both Worlds. *Comput. Sci. Eng.* 13, 2 (2011), 31–39. https://doi.org/10.1109/MCSE.2010.118
[8] Mark Blacher, Joachim Giesen, Sören Laue, Julien Klaus, and Viktor Leis. 2022. Machine Learning, Linear Algebra, and More: Is SQL All You Need?. In *12th Conference on Innovative Data Systems Research, CIDR 2022, Chaminade, CA, USA, January 9-12, 2022*. www.cidrdb.org. https://www.cidrdb.org/cidr2022/papers/p17-blacher.pdf
[9] Carl Friedrich Bolz, Antonio Cuni, Maciej Fijalkowski, and Armin Rigo. 2009. Tracing the meta-level: PyPy's tracing JIT compiler. In *Proceedings of the 4th workshop on the Implementation, Compilation, Optimization of Object-Oriented Languages and Programming Systems, ICOOOLPS 2009, Genova, Italy, July 6, 2009*, Ian Rogers (Ed.). ACM, 18–25. https://doi.org/10.1145/1565824.1565827
[10] Yannis E. Foufoulas, Alkis Simitsis, Eleftherios Stamatogiannakis, and Yannis E. Ioannidis. 2022. YeSQL: "You extend SQL" with Rich and Highly Performant User-Defined Functions in Relational Databases. *Proc. VLDB Endow.* 15, 10 (2022), 2270–2283. https://www.vldb.org/pvldb/vol15/p2270-foufoulas.pdf
[11] Steffen Kläbe, Robert DeSantis, Stefan Hagedorn, and Kai-Uwe Sattler. 2022. Accelerating Python UDFs in Vectorized Query Execution. In *12th Conference on Innovative Data Systems Research, CIDR 2022, Chaminade, CA, USA, January 9-12, 2022*. www.cidrdb.org. https://www.cidrdb.org/cidr2022/papers/p33-klaebe.pdf
[12] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. 2015. Numba: a LLVM-based Python JIT compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC, LLVM 2015, Austin, Texas, USA, November 15, 2015*, Hal Finkel (Ed.). ACM, 7:1–7:6. https://doi.org/10.1145/2833157.2833162
[13] Kevin Modzelewski. 2014. Introducing Pyston: an upcoming, JIT-based Python implementation. https://nuitka.net/ Last accessed 13 Juni 2023.
[14] Armin Rigo and Maciej Fijalkowski. 2021. CFFI. https://cffi.readthedocs.io/en/latest/# Last accessed 13 Juni 2023.
[15] Leonhard F. Spiegelberg, Rahul Yesantharao, Malte Schwarzkopf, and Tim Kraska. 2021. Tuplex: Data Science in Python at Native Code Speed. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava (Eds.). ACM, 1718–1731. https://doi.org/10.1145/3448016.3457244
[16] Lefteris Stamatogiannakis Yannis Foufoulas, Alkis Simitsis and Yannis Ioannidis. 2022. YeSQL specifications. https://athenarc.github.io/YeSQL/index.html Last accessed 13 Juni 2023.