Future Database Engine Development: You Will Only Need One Programming Language

Tianzheng Wang
Simon Fraser University
tzwang@sfu.ca

Database systems must make good use of the hardware for high performance. This is usually done by implementing their core components (storage engine, optimizer and query execution engine) in a low-level programming language (PL) such as C/C++ that can directly "talk to the hardware." But these PLs traditionally lacked high-level abstractions, lowering DBMS developer productivity. Some systems [10, 16, 18] then mix different PLs to balance productivity and performance. For example, Presto [16] and Spark [18] originally used Java but are now replacing their query engines with new ones [1, 12] built in C++ for higher performance. However, doing so brings such non-trivial challenges as interacting with different PL runtimes [1].

Recent advances in PLs, in particular C++ and Rust, have the potential of removing such need. They introduce many desirable features that improve productivity without sacrificing performance. This means core DBMS components could all be implemented in one PL (although other non-performance critical parts may still stick with higher-level PLs). To see why, let us first review the requirements DBMSs pose for PLs.

What does a DBMS Need from a PL?[1] Different DBMS components pose varying requirements on productivity and performance. The optimizer is logic-centric, so there is a strong need for coding productivity to easily express various optimizations. The storage and query engines, however, need to handle parallelism and concurrency by taking full advantage of the hardware, making performance the top priority. Developers are willing to tolerate extra complexity for better performance, such as implementing lock-free indexes [11]. A query engine also needs to implement complex relational algebra and extensions, posing higher requirements on coding productivity than storage engines. As a result, the de facto standard for programming storage and query engines has been traditional C/C++, whereas developers may choose a higher-level PL for optimizers. This has led to the state of mixing PLs for DBMS components.

[1]This paper targets PLs used to implement DBMSs themselves, instead of "database programming languages" which explored the convergence of database systems and PLs [2].

The Case for Modern C++. At a first glance, C++ may be too "low-level" as a native language. However, *modern* C++ (17, 20 and later) comes with features that improve performance and productivity. For example, C++20 allows dynamic memory allocations at compile-time, improving performance by shifting calculations to compile-time and generating smaller binaries [6]. They can also improve productivity as the compiler can detect certain undefined behaviors and leaks at compile-time [7]. In a DBMS, memory used by optimizers typically has a lifespan of queries, yet for storage engines the lifespan is a transaction, requiring different allocators. Some may be a bump allocator that reclaims all memory chunks as a whole, while others should support "real" deallocations of individual chunks. The polymorphic memory resource (`std::pmr`) [17] provides a promising solution, with a set of utilities to manage runtime polymorphism of memory allocations with unified interfaces.

Another example is C++20 coroutines [5] which are being adopted by recent work [9, 13]. Traditionally, query engines implement the iterator model with little PL support and thus an operator must manually maintain states and intermediate data, such as current scan cursor position. C++20 coroutines can help alleviate this problem with a generator-based model. A scan operator's `get_next` can be turned into a coroutine that directly yields each valid row without having to maintain the current row cursor, simplifying implementation.

Future Directions. I believe it is now feasible to satisfy the needs of core DBMS components using a single PL, and many more modern PL features remain to be explored. Beyond C++, Rust [15] and Carbon [4] have gained much attention. It is promising to quantitatively compare these PLs for DBMS implementation. Earlier efforts such as EXODUS [3] and the E programming language [8, 14] have contributed to PL designs. Revisiting them and devising new PLs for DBMS implementation is also promising. Despite the new PL features, compiler and ecosystem support often fall short for serious DBMS development. It is time again for DBMS developers to participate and influence PL and compiler design to push the desired features early into future PLs.

References

[1] A. Behm, S. Palkar, U. Agarwal, T. Armstrong, D. Cashman, A. Dave, T. Greenstein, S. Hovsepian, R. Johnson, A. Sai Krishnan, P. Leventis, A. Luszczak, P. Menon, M. Mokhtar, G. Pang, S. Paranjpye, G. Rahn, B. Samwel, T. van Bussel, H. van Hovell, M. Xue, R. Xin, and M. Zaharia. Photon: A fast query engine for lakehouse systems. In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, page 2326–2339, New York, NY, USA, 2022. Association for Computing Machinery.

[2] M. J. Carey and D. J. DeWitt. Of objects and databases: A decade of turmoil. In *Proceedings of the 22th International Conference on Very Large Data Bases*, VLDB '96, page 3–14, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

[3] M. J. Carey, D. J. DeWitt, D. Frank, M. Muralikrishna, G. Graefe, J. E. Richardson, and E. J. Shekita. The architecture of the EXODUS extensible DBMS. In *Proceedings on the 1986 International Workshop on Object-Oriented Database Systems*, page 52–65, 1986.

[4] C. Carruth. Carbon language: An experimental successor to C++. *CppNorth*, 2022.

[5] Coroutines. `https://en.cppreference.com/w/cpp/language/coroutines`, 2022.

[6] A. Fertig. C++20 dynamic allocations at compile-time, 2021. `https://andreasfertig.blog/2021/08/cpp20-dynamic-allocations-at-compile-time/`.

[7] B. Filipek. constexpr dynamic memory allocation, C++20, 2021. `https://www.cppstories.com/2021/constexpr-new-cpp20/`.

[8] E. N. Hanson, T. M. Harvey, and M. A. Roth. Experiences in DBMS implementation using an object-oriented persistent programming language and a database toolkit. In *Conference Proceedings on Object-Oriented Programming Systems, Languages, and Applications*, OOPSLA '91, page 314–328, New York, NY, USA, 1991. Association for Computing Machinery.

[9] Y. He, J. Lu, and T. Wang. Corobase: Coroutine-oriented main-memory database engine. *PVLDB*, 14(3):431–444, Nov 2020.

[10] M. Kornacker, A. Behm, V. Bittorf, T. Bobrovytsky, C. Ching, A. Choi, J. Erickson, M. Grund, D. Hecht, M. Jacobs, I. Joshi, L. Kuff, D. Kumar, A. Leblang, N. Li, I. Pandis, H. Robinson, D. Rorke, S. Rus, J. Russell, D. Tsirogiannis, S. Wanderman-Milne, and M. Yoder. Impala: A modern, open-source SQL engine for Hadoop. 2015.

[11] J. J. Levandoski, D. B. Lomet, and S. Sengupta. The Bw-Tree: A B-tree for new hardware platforms. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 302–313, 2013.

[12] P. Pedreira, O. Erling, M. Basmanova, K. Wilfong, L. Sakka, K. Pai, W. He, and B. Chattopadhyay. Velox: Meta's unified execution engine. *PVLDB*, 15(12):3372–3384, Aug 2022.

[13] G. Psaropoulos, T. Legler, N. May, and A. Ailamaki. Interleaving with coroutines: A practical approach for robust index joins. *PVLDB*, 11(2):230–242, Oct 2017.

[14] J. E. Richardson, M. J. Carey, and D. T. Schuh. The design of the E programming language. *ACM Trans. Program. Lang. Syst.*, 15(3):494–534, Jul 1993.

[15] Rust Foundation. Rust programming language, 2022.

[16] R. Sethi, M. Traverso, D. Sundstrom, D. Phillips, W. Xie, Y. Sun, N. Yegitbasi, H. Jin, E. Hwang, N. Shingte, and C. Berner. Presto: SQL on everything. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1802–1813, 2019.

[17] `std::pmr::polymorphic_allocator`. `https://en.cppreference.com/w/cpp/memory/polymorphic_allocator`, 2022.

[18] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, USA, 2012. USENIX Association.