Dissecting DuckDB: The internals of the “SQLite for Analytics”

Pedro Holanda, Mark Raasveldt

[1] Centrum Wiskunde & Informatica (CWI)
Amsterdam, The Netherlands.

{holanda,raasveld}@cwi.nl

Abstract. *The immense popularity of SQLite shows that there is a need for unobtrusive in-process data management solutions. However, there is no such system yet geared towards analytical workloads. We present DuckDB, a novel data management system designed to execute analytical SQL queries while embedded in another process. In our talk, we give an in-depth overview of the internals of DuckDB and the design choices that were made to cater to the use case of embedded analytics. DuckDB is available as Open Source software under a permissive license.*

Resumo. *A imensa popularidade do SQLite demonstra a necessidade de sistemas de gerenciamento de banco de dados (SGBD) embarcados. No entanto, ainda não existe um SGBD embarcado voltado para cargas de trabalho analíticas. Nessa palestra apresentamos o DuckDB, um novo SGBD projetado para executar consultas analíticas enquanto incorporado em outro processo. Apresentamos uma visão geral dos aspectos internos do DuckDB e das decisões de design feitas para atender as cargas analíticas em SGBDs embarcados. O DuckDB já está disponível para download e uso.*

Introduction

In this talk, we present the internal structure of our new system, *DuckDB*. DuckDB is a new purpose-built embeddable relational database management system. DuckDB is available as Open-Source software under the permissive MIT license[1]. DuckDB is no research prototype but built to be widely used, with millions of test queries run on each commit to ensure correct operation and completeness of the SQL interface. DuckDB was built specifically to support the use case of embedded analytics, and focused on fulfilling the following requirements of this use case:

- Efficient transfer of tables to and from the database is essential. Since both database and application run in the same process and thus address space, there is a unique opportunity for efficient data sharing which needs to be exploited.
- High efficiency for OLAP workloads, but without completely sacrificing OLTP performance. For example, concurrent data modification is a common use case in dashboard-scenarios where multiple threads update the data using OLTP queries, and other threads run the OLAP queries that drive visualizations simultaneously.

[1]https://github.com/cwida/duckdb

- High degree of stability, if the embedded database crashes, for example, due to an out-of-memory situation, it takes the host down with it. This can never happen. Queries need to be able to be aborted cleanly if they run out of resources, and the system needs to gracefully adapt to resource contention.
- Practical "embeddability" and portability, the database needs to run in whatever environment the host does. Dependencies on external libraries (e.g., `openssh`) for either compile- or runtime have been found to be problematic. Signal handling, calls to `exit()` and modification of singular process state (locale, working directory, etc.) are forbidden.

Outline

The talk is catered primarily towards people that have a basic understanding of core database systems, and the goal of the talk is to make users more familiar with the internals of modern columnar database systems and specifically the internals of DuckDB. The talk is divided into six sections of 30 minutes each:

1. DuckDB: Introduction and Motivation
2. Parser, Binder and Logical Planner
3. Physical Execution
4. Optimizers
5. Transactions & Storage Layer
6. Interactive Demo

DuckDB: Introduction and Motivation. In the talk, we start by giving a brief overview of DuckDB and the motivation behind embedded analytical systems.

Parser, Binder and Logical Planner. In this section we discuss the parser, binder and logical planner of the DuckDB system. We do this by taking you on a journey of the life of a query inside the database system. We show the internal structures that are created and how the query string is converted into a logical plan.

Physical Execution. After showing how the logical plan is constructed, we show how the system executes the physical plan using its vectorized execution engine [Boncz et al. 2005]. We discuss the techniques that are used for the execution of scans, indexes [Leis et al. 2013], joins, aggregates, sorting and window functions [Leis et al. 2015].

Optimizers. After the base logical plan is created, we show how the optimizers work to create a fast plan. We discuss join order optimization [Moerkotte and Neumann 2008], subquery unnesting [Neumann and Kemper 2015], filter pushdown as well as various simple scalar optimizers such as constant folding and common subexpression elimination.

Transactions & Storage Layer. We discuss the MVCC model [Neumann et al. 2015] that is used by DuckDB and how it works to maintain transactional integrity and atomicity. We also discuss the storage layer of DuckDB. We describe how the data is laid out on disk and talk about how the buffer manager loads and caches data into memory [Leis et al. 2018].

Interactive Demo. We guide people in setting up and using DuckDB in combination with Python. We have a prepared data set and demo use case in which users use DuckDB to wrangle a set of census data concerning US voters, after which they use *sci-kit le-arn* to build a full machine learning pipeline that attempts to classify individual voters [Raasveldt et al. 2018].

Referências

Boncz, P. A., Zukowski, M., and Nes, N. (2005). Monetdb/x100: Hyper-pipelining query execution. In *CIDR*.

Leis, V., Haubenschild, M., Kemper, A., and Neumann, T. (2018). Leanstore: In-memory data management beyond main memory. In *ICDE*.

Leis, V., Kemper, A., and Neumann, T. (2013). The adaptive radix tree: Artful indexing for main-memory databases. In *ICDE*.

Leis, V., Kundhikanjana, K., Kemper, A., and Neumann, T. (2015). Efficient processing of window functions in analytical sql queries. *VLDB*.

Moerkotte, G. and Neumann, T. (2008). Dynamic programming strikes back. In *SIG-MOD*. ACM.

Neumann, T. and Kemper, A. (2015). Unnesting arbitrary queries. *Datenbanksysteme für Business, Technologie und Web (BTW 2015)*.

Neumann, T., Mühlbauer, T., and Kemper, A. (2015). Fast serializable multi-version concurrency control for main-memory database systems. In *SIGMOD*.

Raasveldt, M., Holanda, P., Mühleisen, H., Manegold, S., et al. (2018). Deep integration of machine learning into column stores. *EDBT*.