Why DuckDB

IN THIS ARTICLE
Si...
Po...
Fe...
Fast
Ext...
Free
Th...
Pe...
Ot...
Sta...

There are many database management systems (DBMS) out there. But there is no one-size-fits all database system. All take different trade-offs to better adjust to specific use cases. DuckDB is no different. Here, we try to explain what goals DuckDB has and why and how we try to achieve those goals through technical means. To start with, DuckDB is a relational (table-oriented) DBMS that supports the Structured Query Language (SQL).

Simple

SQLite is the world's most widely deployed DBMS. Simplicity in installation, and embedded in-process operation are central to its success. DuckDB adopts these ideas of simplicity and embedded operation.

DuckDB has **no external dependencies**, neither for compilation nor during run-time. For releases, the entire source tree of DuckDB is compiled into two files, a header and an implementation file, a so-called "amalgamation". This greatly simplifies deployment and integration in other build processes. For building, all that is required to build DuckDB is a working C++11 compiler.

Search cmd+k

Installation

Documentation

- Getting Started
- Connect
- Data Import
- Client APIs
- SQL
- Configuration
- Extensions
- Guides
- Operations Manual
- Development
- Internals

Sitemap

Why DuckDB

Media

FAQ

Code of Conduct

Live Demo

For DuckDB, there is no DBMS server software to install, update and maintain. DuckDB does not run as a separate process, but completely **embedded within a host process**. For the analytical use cases that DuckDB targets, this has the additional advantage of **high-speed data transfer** to and from the database. In some cases, DuckDB can process foreign data without copying. For example, the DuckDB Python package can run queries directly on Pandas data without ever importing or copying any data.

Portable

Thanks to having no dependencies, DuckDB is extremely portable. It can be compiled for all major operating systems (Linux, macOS, Windows) and CPU architectures (x86, ARM). It can be deployed from small, resource-constrained edge devices to large multi-terabyte memory servers with 100+ CPU cores. Using DuckDB-Wasm, DuckDB can also run in web browsers and even on mobile phones.

DuckDB provides APIs for Java, C, C++, Go, Node.js and other languages.

Feature-Rich

DuckDB provides serious data management features. There is extensive support for **complex queries** in SQL with a large function library, window functions, etc. DuckDB provides **transactional guarantees** (ACID properties) through our

Search cmd+k

Installation

Documentation

- Getting Started
- Connect
- Data Import
- Client APIs
- SQL
- Configuration
- Extensions
- Guides
- Operations Manual
- Development
- Internals

Sitemap

Why DuckDB

Media

FAQ

Code of Conduct

Live Demo

custom, bulk-optimized Multi-Version Concurrency Control (MVCC). Data can be stored in persistent, **single-file databases**. DuckDB supports secondary indexes to speed up queries trying to find a single table entry.

DuckDB is deeply integrated into Python and R for efficient interactive data analysis.

Fast

DuckDB is designed to support **analytical query workloads**, also known as online analytical processing (OLAP). These workloads are characterized by complex, relatively long-running queries that process significant portions of the stored dataset, for example aggregations over entire tables or joins between several large tables. Changes to the data are expected to be rather large-scale as well, with several rows being appended, or large portions of tables being changed or added at the same time.

To efficiently support this workload, it is critical to reduce the amount of CPU cycles that are expended per individual value. The state of the art in data management to achieve this are either vectorized or just-in-time query execution engines. DuckDB uses a **columnar-vectorized query execution engine**, where queries are still interpreted, but a large batch of values (a "vector") are processed in one operation. This greatly reduces overhead present in traditional systems such as PostgreSQL, MySQL or SQLite which process each row

sequentially. Vectorized query execution leads to far better performance in OLAP queries.

Search cmd+k

Installation
Documentation
- Getting Started
- Connect
- Data Import
- Client APIs
- SQL
- Configuration
- Extensions
- Guides
- Operations Manual
- Development
- Internals

Sitemap
Why DuckDB
Media
FAQ
Code of Conduct
Live Demo

Extensible

DuckDB offers a flexible extension mechanism that allows defining new data types, functions, file formats and new SQL syntax. In fact, many of DuckDB's key features, such as support for the Parquet file format, JSON, time zones, and supports for the HTTP(S) and S3 protocols are implemented as extensions. Extensions also work in DuckDB Wasm.

Free

DuckDB's development started while the main developers were public servants in the Netherlands. We see it as our responsibility and duty to society to make the results of our work freely available to anyone in the Netherlands or elsewhere. This is why DuckDB is released under the very permissive MIT License. DuckDB is open-source, the entire source code is freely available on GitHub. We invite contributions from anyone provided they adhere to our Code of Conduct.

Thorough Testing

While DuckDB was originally created by a research group, it was never intended to be a research prototype. Instead, it was intended to become a stable and mature database system. To facilitate this

Search cmd+k

Installation
Documentation
Getting Started
Connect
Data Import
Client APIs
SQL
Configuration
Extensions
Guides
Operations Manual
Development
Internals
Sitemap
Why DuckDB
Media
FAQ
Code of Conduct
Live Demo

stability, DuckDB is intensively tested using Continuous Integration. DuckDB's test suite currently contains millions of queries, and includes queries adapted from the test suites of SQLite, PostgreSQL, and MonetDB. Tests are repeated on a wide variety of platforms and compilers. Every pull request is checked against the full test setup and only merged if it passes.

In addition to this test suite, we run various tests that stress DuckDB under heavy loads. We run the TPC-H and TPC-DS benchmarks, and run various tests where DuckDB is used by many clients in parallel.

Peer-Reviewed Papers and Thesis Works

- Runtime-Extensible Parsers (CIDR 2025)
- Robust External Hash Aggregation in the Solid State Age (ICDE 2024)
- These Rows Are Made for Sorting and That's Just What We'll Do (ICDE 2023)
- Join Order Optimization with (Almost) No Statistics (Master thesis, 2022)
- DuckDB-Wasm: Fast Analytical Processing for the Web (VLDB 2022 Demo)
- Data Management for Data Science - Towards Embedded Analytics (CIDR 2020)
- DuckDB: an Embeddable Analytical Database (SIGMOD 2019 Demo)

Search cmd+k

Installation
Documentation
Getting Started
Connect
Data Import
Client APIs
SQL
Configuration
Extensions
Guides
Operations Manual
Development
Internals
Sitemap
Why DuckDB
Media
FAQ
Code of Conduct
Live Demo

Other Projects

To learn about projects using DuckDB, visit the Awesome DuckDB repository.

Standing on the Shoulders of Giants

DuckDB uses some components from various open-source projects and draws inspiration from scientific publications. We are very grateful for this. Here is an overview:

- **Execution engine:** The vectorized execution engine is inspired by the paper MonetDB/X100: Hyper-Pipelining Query Execution by Peter Boncz, Marcin Zukowski and Niels Nes. MonetDB/X100 later became the Vectorwise (Actian Vector) database system.
- **Optimizer:** DuckDB's optimizer draws inspiration from the papers Dynamic programming strikes back by Guido Moerkotte and Thomas Neumann as well as Unnesting Arbitrary Queries by Thomas Neumann and Alfons Kemper.
- **Concurrency control:** Our MVCC implementation is inspired by the paper Fast Serializable Multi-Version Concurrency Control for Main-Memory Database Systems by Thomas Neumann, Tobias Mühlbauer and Alfons Kemper.
- **Secondary indexes:** DuckDB has support for secondary indexes based on the paper The Adaptive Radix Tree: ARTful Indexing for Main-Memory

Databases by Viktor Leis, Alfons Kemper and Thomas Neumann.
- **SQL window functions:** DuckDB's window functions implementation uses Segment Tree Aggregation as described in the paper Efficient Processing of Window Functions in Analytical SQL Queries by Viktor Leis, Kan Kundhikanjana, Alfons Kemper and Thomas Neumann.
- **SQL inequality joins:** DuckDB's inequality join implementation uses the IEJoin algorithm as described in the paper Lightning Fast and Space Efficient Inequality Joins Zuhair Khayyat, William Lucia, Meghna Singh, Mourad Ouzzani, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Nan Tang and Panos Kalnis.
- **Compression of floating-point values:** DuckDB supports the multiple algorithms for compressing floating-point values:
  - Chimp by Panagiotis Liakos, Katia Papakonstantinopoulou and Yannis Kotidi
  - Patas, an in-house development, and
  - ALP (adaptive lossless floating-point compression) by Azim Afroozeh, Leonard Kuffo and Peter Boncz, who also contributed their implementation.
- **SQL Parser:** We use the PostgreSQL parser that was repackaged as a stand-alone library. The translation to our own parse tree is inspired by Peloton.
- **Shell:** We use the SQLite shell to work with DuckDB.

Search cmd+k

Installation

Documentation

- Getting Started
- Connect
- Data Import
- Client APIs
- SQL
- Configuration
- Extensions
- Guides
- Operations Manual
- Development
- Internals

Sitemap

Why DuckDB

Media

FAQ

Code of Conduct

Live Demo

- **Regular expressions:** DuckDB uses Google's RE2 regular expression engine.
- **String formatting:** DuckDB uses the fmt string formatting library.
- **UTF wrangling:** DuckDB uses the utf8proc library to check and normalize UTF8.
- **Collation and time:** DuckDB uses the ICU library for collation, time zone, and calendar support.
- **Test framework:** DuckDB uses the Catch2 unit test framework.
- **Test cases:** We use the SQL Logic Tests from SQLite to test DuckDB.
- **Result validation:** Manuel Rigger used his excellent SQLancer tool to verify DuckDB result correctness.
- **Query fuzzing:** We use SQLsmith via the `sqlsmith` extension to generate random queries for additional testing.
- **JSON parser:** We use yyjson, a high performance JSON library written in ANSI C, to parse JSON in DuckDB's JSON Extension.