THE DUCKDB HANDBOOK

A COMPREHENSIVE GUIDE TO IN-MEMORY ANALYTICAL PROCESSING

- BLAZING FAST ANALYTICS
- SQL SIMPLICITY POWERFUL ENGINE
- EMBEDDABLE EVERYWHERE
- FROM LAPTOP TO PRODUCTION

ABBY TAYLOR

The DuckDB Handbook

A Comprehensive Guide to In-Memory Analytical Processing

Steve Publications

This book is available at https://leanpub.com/theduckdbhandbook

This version was published on 2026-07-29

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Contents

A Comprehensive Guide to In-Memory Analytical Processing **1**

Introduction: Why DuckDB? **3**
The Analytical Data Problem 3
Enter DuckDB: A Different Kind of Database 3
When to Use (and Not Use) DuckDB 4
How This Book Is Organized 5

Chapter 1: Architecture and Internals **7**
Vectorized Query Execution Explained 7
The Columnar Storage Engine 8
Query Optimization Pipeline 9
Memory Management Model 11
Concurrency and Threading 12

Chapter 2: Getting Started with DuckDB **14**
Installation Across Platforms 14
The DuckDB Command-Line Interface 15
Your First Analytical Queries 18
Working with Databases and Connections 19
Basic Data Exploration Workflow 21

Chapter 3: Data Types, Schemas, and Tables **23**
Sample Database Setup for Chapters 3 through 7 23
Primitive and Composite Data Types 23
Structs, Lists, Maps, and Unions 23
Creating and Managing Tables 23
Constraints and Integrity 23
Temporary and Transient Tables 23

Chapter 4: Core SQL Operations **25**

SELECT Statements and Projection 25
Filtering with WHERE and Boolean Logic 25
Sorting, Limiting, and Offset 25
DISTINCT and Deduplication Patterns 25
Common Pitfalls and Best Practices 25

Chapter 5: Joins and Set Operations 26
Inner Joins and Outer Joins 26
Cross Joins and Semi/Anti Joins 26
Join Algorithms and When DuckDB Chooses Each 26
UNION, INTERSECT, EXCEPT 26
Large-Scale Join Strategies 26

Chapter 6: Aggregations and Window Functions 27
GROUP BY and Aggregate Functions 27
HAVING and Filtering Aggregates 27
Window Functions Syntax and Semantics 27
Analytical Patterns with Windows 27
Performance Considerations for Heavy Aggregation 27

Chapter 7: Advanced SQL Features 28
Common Table Expressions (CTEs) 28
Subqueries and Correlated Queries 28
Views and Materialized Patterns 28
Macros and Procedural Extensions 28
Transactions and Isolation Levels 28
LATERAL Joins for Row-by-Row Computation 28
PIVOT and UNPIVOT for Data Reshaping 29
QUALIFY Clause for Window Function Filtering 29
SAMPLE Clause for Efficient Data Sampling 29
POSITIONAL Joins for Row-Order Matching 29
Advanced STRUCT Operations 29

Chapter 8: Data Ingestion and Export 30
CSV Import and Export 30
JSON Processing 30
Parquet Integration 30
Avro, ORC, and Other Formats 30
COPY Statements and Bulk Operations 30

Chapter 9: The Extensions Ecosystem **31**
Extension Architecture and Loading 31
Spatial and Geospatial Extensions 31
HTTP and Web Data Access 31
Apache Iceberg and Delta Lake 31
MotherDuck and Cloud Integration 31

Chapter 10: Programming APIs and Language Bindings **32**
Python API and duckdb Package 32
R Integration via dbplyr and duckdb 32
Java and JDBC Connectivity 32
C/C++ Embedding API 32
Go, Rust, Node.js, and Other Bindings 32

Chapter 11: DataFrame and Ecosystem Integrations **33**
Pandas Integration and Lazy Evaluation 33
Polars and Arrow Interoperability 33
Apache Arrow Flight and Memory Sharing 33
Ibis Integration 33
dbt and Data Transformation Workflows 33
Jupyter, Notebooks, and Interactive Analysis 33

Chapter 12: Performance Engineering **35**
Understanding Vectorized Execution Performance 35
Parallel Query Execution Tuning 35
Storage Layout and Compression Strategies 35
Predicate Pushdown and Projection Pushdown 35
Late Materialization 35
Join Optimization and Hash Tables 35
Memory Limits, Caching, and Spilling 36
Reproducible Benchmark Suite 36

Chapter 13: Embedding DuckDB in Applications **37**
The Embedded Database Paradigm 37
Building Analytics Into Your Application 37
Connection Management and Lifecycle 37
Packaging and Distribution Considerations 37
Real-World Embedding Patterns 37

Chapter 14: Production Deployment and Operations **38**

Security Model and Access Control . . . 38
Configuration and Pragmas Reference . . . 38
Profiling Queries and Debugging Performance . . . 38
Migration Strategies from SQLite and PostgreSQL . . . 38
Enterprise Patterns and Best Practices . . . 38

Conclusion: The Future of Analytical Databases . . . 39
Where DuckDB Fits in the Modern Stack . . . 39
Emerging Trends and Capabilities . . . 39
Choosing Your Tool: A Decision Framework . . . 39

References . . . 40

Index . . . 41
A . . . 41
B . . . 41
C . . . 41
D . . . 41
E . . . 41
F . . . 41
G . . . 42
H . . . 42
I . . . 42
J . . . 42
L . . . 42
M . . . 42
N . . . 42
O . . . 43
P . . . 43
Q . . . 43
R . . . 43
S . . . 43
T . . . 43
U . . . 43
V . . . 44
W . . . 44
Z . . . 44

A Comprehensive Guide to In-Memory Analytical Processing

This book is dedicated to the DuckDB community, whose contributions, feedback, and innovation have made this project one of the most important tools in modern data engineering.

All code examples in this book are tested against DuckDB version 1.5.0 and later releases. While every effort has been made to ensure accuracy, software evolves rapidly. Consult the official DuckDB documentation at duckdb.org for the latest information on features, syntax, and best practices.

Technical Reviewers and Acknowledgments

Special thanks to the DuckDB core team for their openness to technical scrutiny and documentation improvements. Thanks also to the broader data engineering community whose benchmarks, analyses, and discussions informed this work.

DuckDB has emerged as one of the most important tools in modern data engineering, offering analytical database performance inside a single process with the simplicity of SQLite. This book is your definitive guide to mastering DuckDB from first principles through expert-level usage. You will learn how DuckDB works under the hood, how to write optimal queries, how to integrate it into applications across every major programming language, and how to deploy it in production systems. Whether you are a data analyst running ad-hoc queries on Parquet files, a data engineer building local-first transformation pipelines, or a software developer embedding analytics directly into your product, this handbook gives you the depth and practical knowledge to use DuckDB effectively and confidently.

Introduction: Why DuckDB?

The Analytical Data Problem

Every data professional knows the frustration. You have a CSV file that is too large for Excel, a set of Parquet files scattered across directories, and a question you need to answer right now. You could load everything into a data warehouse and wait for the pipeline to run. You could write a Python script with nested loops and pray it finishes before lunch. Or you could spin up Spark, configure clusters, and spend more time on infrastructure than on the actual analysis.

This is the analytical data problem: we need powerful, flexible, fast query capabilities for data that lives locally or near at hand, but traditional tools make this unnecessarily hard. Data warehouses are expensive and distant. In-memory data frames max out when data grows beyond RAM. Distributed systems require operational overhead that most teams cannot justify for single-machine workloads.

The problem is not new, but its urgency has grown. Organizations generate more data than ever, and the people who need to analyze it are no longer just dedicated data engineers sitting in front of warehouses. Product managers, researchers, software developers, and operations teams all run analytical queries as part of their daily work. They need tools that are fast enough for serious analysis but simple enough to adopt without a database degree.

Enter DuckDB: A Different Kind of Database

DuckDB is an in-process, column-oriented analytical database management system designed specifically for OLAP (online analytical processing) workloads. It runs inside your application process, just like SQLite, but it brings the performance characteristics of modern data warehouses to your laptop, your server, or your embedded device.

The core idea behind DuckDB is elegantly simple: what if we took the best ideas from data warehouse design and packaged them into an embeddable

library that any developer or analyst could use without managing a separate server process? The result is a database that:

- Runs entirely within your application, with no network overhead
- Uses columnar storage for efficient analytical scans
- Processes data in vectorized batches using modern CPU features
- Automatically parallelizes queries across all available cores
- Reads directly from common file formats like Parquet, CSV, and JSON without importing data into a proprietary format
- Supports a rich SQL dialect with advanced analytical functions

DuckDB was created by Mark Raasveldt and Hannes Mühleisen, who published their foundational paper "Data Management for Data Science: Towards Embedded Analytics" at CIDR 2020. Their vision was to bring database-grade analytics to the data science workflow, where analysts and researchers typically work with data in notebooks and local environments.

The project has grown rapidly since its inception. As of this writing, DuckDB is widely adopted across the data engineering ecosystem, integrated into tools ranging from dbt and Ibis to Jupyter and RStudio, and used by organizations including PostHog, Supabase, and countless data teams worldwide. It has established itself as the de facto standard for local analytical processing.

When to Use (and Not Use) DuckDB

DuckDB is exceptionally good at certain things, but it is not a universal database replacement. Understanding where DuckDB excels and where it falls short will help you make better architectural decisions.

Use DuckDB when:

- You need fast analytical queries on datasets that fit on a single machine (from a few megabytes to hundreds of gigabytes)
- You want to query files directly without importing them into a database
- You are building a data pipeline that transforms data locally before loading it into a warehouse
- You need embedded analytics inside an application without managing a separate database server

- You are doing exploratory data analysis in a notebook or interactive environment
- You want to combine data from multiple sources (CSV, Parquet, JSON, other databases) using SQL
- You are teaching or learning SQL and want a lightweight, feature-rich environment

Do not use DuckDB when:

- You need high-concurrency OLTP transactions with many simultaneous writers
- Your dataset requires distributed processing across multiple machines in the terabyte to petabyte range
- You need a multi-user database with fine-grained access control and user authentication
- Your workload consists primarily of single-row point lookups and updates
- You require a client-server architecture with network-based connections from many clients

DuckDB is not a replacement for PostgreSQL or MySQL in transactional applications. It is not a replacement for Apache Spark or ClickHouse when you need to process petabytes across clusters. But within its domain, it is remarkably capable.

How This Book Is Organized

This book is structured to take you from foundational concepts through expert-level mastery. Each chapter builds on the previous ones, but you can also use it as a reference, jumping directly to the topics you need.

The Introduction sets up the context and motivation for DuckDB, which we are reading now. Chapter 1 dives into DuckDB's architecture and internals, explaining how vectorized execution, columnar storage, and query optimization make it fast. Understanding these concepts will help you write better queries and troubleshoot performance issues.

Chapter 2 gets you hands-on with installation, the command-line interface, and your first queries. Chapter 3 covers DuckDB's complete type system and

schema management. Chapters 4 through 7 form the SQL core: fundamental operations, joins, aggregations, window functions, CTEs, macros, and transactions.

Chapter 8 focuses on data ingestion and export across all supported file formats. Chapter 9 explores the extensions ecosystem, including spatial data, cloud storage, and lakehouse integrations. Chapters 10 and 11 cover programming APIs and ecosystem integrations: Python, R, Java, Go, Rust, Node.js, as well as Pandas, Polars, Arrow, Ibis, dbt, and Jupyter.

Chapter 12 is a deep dive into performance engineering, explaining how to tune DuckDB for your workloads and how it compares to other systems. Chapter 13 covers embedding DuckDB into production applications. Chapter 14 addresses operational concerns: security, profiling, migration strategies, and enterprise best practices.

The Conclusion synthesizes the book's themes and discusses where DuckDB is headed in the evolving data landscape.

Throughout, every code example is complete and executable against current DuckDB releases. We explain not just how to use each feature but why it exists, how it works internally, what trade-offs it involves, and how to use it effectively in real-world scenarios.

Let us begin.

Chapter 1: Architecture and Internals

Vectorized Query Execution Explained

At the heart of DuckDB's performance is its vectorized query execution model. To understand why this matters, we need to start with how traditional databases execute queries.

Classic relational databases like PostgreSQL and MySQL historically use what is called the "Volcano" iterator model. In this model, each operator in a query plan exposes a simple Next() method that returns one row at a time. A filter operator calls Next() on its child to get a row, checks the predicate, and either returns the row or calls Next() again. This continues until all rows are processed.

The Volcano model is elegant and easy to implement, but it has a fundamental performance problem: function call overhead. For a table with one million rows, every operator in the query plan must be called one million times. Each call involves stack setup, parameter passing, and return handling. When you multiply this across dozens of operators and millions of rows, the overhead becomes enormous.

DuckDB takes a different approach. Instead of processing one row at a time, it processes data in fixed-size batches called vectors. The standard vector size in DuckDB is 2048 tuples. When an operator executes, it receives a vector containing 2048 values and performs its operation on all of them in a single function call.

Consider a simple filter: SELECT * FROM sales WHERE amount > 1000. In the Volcano model, the filter predicate is evaluated one million times for one million rows. In DuckDB's vectorized model, the same predicate is evaluated roughly 489 times (one million divided by 2048), each time operating on a batch of 2048 values.

This reduction in function calls is significant but not the whole story. The real power comes from how vectors enable CPU-level optimizations:

- **SIMD instructions**: Modern CPUs support Single Instruction Multiple Data operations that can process multiple values simultaneously. DuckDB's vectorized operators are written to take advantage of SIMD, performing arithmetic and comparison operations on multiple values in parallel using a single instruction.
- **Cache efficiency**: Processing data in contiguous batches keeps working sets small and predictable. The CPU prefetcher can anticipate memory access patterns and load data into cache before it is needed.
- **Compiler optimization**: Tight loops over vectors are easier for compilers to optimize than recursive iterator chains. Register allocation, loop unrolling, and instruction scheduling all work better.

DuckDB's execution is push-based rather than pull-based. Data chunks flow through the operator tree from bottom to top. A scan operator reads data from storage into a vector, pushes it to the next operator (perhaps a filter), which transforms it and pushes the result further up. This design works naturally with parallelism and pipelining.

The implications for users are straightforward: DuckDB is dramatically faster than row-oriented databases for analytical queries that scan and transform large amounts of data. The vectorized model shines when you are doing the same operation on many values, which is exactly what analytical workloads do.

The Columnar Storage Engine

Vectorized execution works best with columnar storage, and DuckDB uses a native columnar format for its internal tables. In a row-oriented database like SQLite or PostgreSQL, all columns of a single row are stored together on disk. If you have a table with customer_id, order_date, product_name, quantity, and price, each row stores all five fields contiguously.

In DuckDB's columnar format, each column is stored separately. All customer_id values are stored in one place, all order_date values in another, and so on. This design choice has profound consequences for analytical performance.

Consider a query that calculates average price: SELECT AVG(price) FROM orders. In a row-oriented database, the system must read every row from disk,

including customer_id, order_date, product_name, and quantity, even though only the price column is needed. For wide tables with many columns, most of this I/O is wasted.

In DuckDB, the query reads only the price column. If the table has twenty columns but you need one, DuckDB ignores nineteen of them entirely. This projection efficiency means less I/O, less data to move through memory, and more data fitting into CPU cache.

Columnar storage also enables superior compression. Values within a single column tend to be similar or follow patterns. A price column might contain many repeated values. An order_date column is often sorted or has a limited range. Column-oriented compression algorithms exploit these properties:

- **Dictionary encoding**: Repeated string values are replaced with small integer indices into a dictionary. The word "Electronics" appearing ten thousand times becomes the index 42 stored ten thousand times.
- **Run-length encoding**: Consecutive repeated values are compressed into (value, count) pairs.
- **Delta encoding**: Sequentially increasing values like timestamps or IDs are stored as differences from the previous value, which compress much better.
- **Bit-packing**: Integer columns that use only a subset of their range can be stored using fewer bits.

DuckDB's internal format combines these lightweight encodings with general-purpose compression codecs. The result is typically 5x to 10x compression compared to uncompressed row storage, though exact ratios depend on the data.

Beyond compression and projection, columnar storage enables zone maps: min/max statistics stored for each segment of a column. When filtering on a range, DuckDB can skip entire segments whose min/max values fall outside the filter condition. This dramatically reduces I/O for selective queries.

There is a trade-off: columnar storage is less efficient for operations that need all columns of a single row, such as point lookups or single-row inserts. This is why DuckDB excels at analytical scans but is not designed for high-throughput OLTP workloads.

Query Optimization Pipeline

Before DuckDB executes any query, it goes through a sophisticated optimization pipeline. Understanding this pipeline helps explain why DuckDB is fast by default and how to write queries that the optimizer can handle efficiently.

The pipeline has several distinct phases:

Parsing: The SQL text is converted into tokens and then into an abstract syntax tree (AST). At this stage, DuckDB understands the grammatical structure of the query but knows nothing about tables, columns, or types. It does not validate whether referenced tables exist.

Binding: The AST is resolved against the database catalog. Table names are matched to actual tables, column references are validated, and data types are determined. Type coercion rules are applied where necessary. If any reference cannot be resolved, the query fails at this stage with a clear error message.

Logical Planning: The bound query is converted into a logical query tree using LogicalOperator nodes. This tree represents what the query should do semantically but not how it should be executed physically. A JOIN node means "join these two inputs" without specifying hash join, merge join, or nested loop.

Optimization: This is where the magic happens. DuckDB applies a series of optimization passes to the logical plan:

- **Expression rewriter**: Simplifies expressions algebraically. Constant folding evaluates constant expressions at planning time (2 + 3 becomes 5). Redundant operations are eliminated.
- **Filter pushdown**: Predicate conditions are pushed as close to the data source as possible. Instead of joining two tables and then filtering, DuckDB filters each table first and then joins the smaller results.
- **Projection pushdown**: Unneeded columns are dropped early. If a query only needs three columns from a twenty-column table, the plan is adjusted to read only those three.
- **Common subexpression elimination**: If the same expression appears multiple times in a query, it is computed once and reused.
- **Join order optimization**: For queries with multiple joins, DuckDB uses the DPhyp algorithm (Dynamic Programming Strikes Back) to find an efficient join order. Different orders can have performance differences of several orders of magnitude.

- **Build/probe side selection**: For hash joins, DuckDB decides which table to build the hash table on and which to probe. Building on the smaller table is almost always better.

Physical Planning: The optimized logical plan is converted into a physical operator tree. At this stage, specific algorithms are chosen: hash join versus merge join, sort-based aggregation versus hash aggregation, and so on. The physical plan is what actually gets executed.

You can inspect DuckDB's optimization pipeline using EXPLAIN and EXPLAIN ANALYZE. We will cover these tools in detail later, but they are essential for understanding how DuckDB handles your queries and for diagnosing performance issues.

Memory Management Model

DuckDB is designed as an in-memory analytical engine, and its memory management is optimized accordingly. Understanding how DuckDB uses memory is critical for avoiding out-of-memory errors and tuning performance.

The primary memory control is the memory_limit configuration setting. By default, DuckDB sets this to 80% of the system's physical RAM. On a machine with 16 GB of RAM, DuckDB will attempt to use up to 12.8 GB. This is aggressive by design: analytical workloads benefit enormously from having as much data in memory as possible.

The memory_limit applies specifically to the buffer manager, which handles page caching and intermediate result storage. However, some operations allocate memory outside the buffer manager. Aggregate functions with complex state (such as list aggregation or quantile computation) and hash tables for joins can consume additional memory. For this reason, it is often wise to set memory_limit to 50-60% of available RAM in production environments to leave headroom.

DuckDB requires approximately 125 MB of memory per thread. If you configure DuckDB to use 8 threads, you need at least about 1 GB of memory just for the threading infrastructure. In memory-constrained environments, reducing the thread count is often more effective than fighting with memory limits.

When operations exceed available memory, DuckDB spills to disk. Large sorts, hash joins, and aggregations can write intermediate results to temporary files in the temp_directory (by default, the system's temporary directory). This prevents out-of-memory crashes but significantly impacts performance. The max_temp_directory_size setting controls how much disk space DuckDB can use for spilling.

DuckDB uses a Write-Ahead Log (WAL) for durability. Changes are written to the WAL before being applied to the main database file. This ensures that committed transactions survive crashes. The WAL is stored alongside the main database file and is automatically managed.

For in-memory databases (created with :memory: or an empty path), all data lives in RAM and is lost when the connection closes. These are ideal for temporary analysis and testing but not for persistent storage.

Concurrency and Threading

DuckDB's concurrency model reflects its design as an embedded, analytical database. It is optimized for a specific pattern: one process doing heavy analytical work with bulk operations, rather than many processes doing concurrent point transactions.

At the process level, DuckDB supports two modes:

- **Read-write mode**: One process can both read and write to the database. This is the default and most common mode.
- **Read-only mode**: Multiple processes can read from the same database simultaneously, but no process can write. You enable this by setting access_mode = 'READ_ONLY' when connecting.

Within a single read-write process, DuckDB supports multiple writer threads using Multi-Version Concurrency Control (MVCC) combined with optimistic concurrency control. MVCC works by maintaining multiple versions of modified rows. When a transaction reads data, it sees a consistent snapshot as of its start time. When it writes, it creates new versions rather than overwriting existing ones. This allows readers and writers to proceed without blocking each other.

Optimistic concurrency control means that DuckDB does not acquire locks upfront. Instead, it lets transactions proceed and checks for conflicts at commit time. If two transactions try to modify the same row, the second one to commit is aborted. This works well for analytical workloads where write conflicts are rare.

The isolation level is snapshot isolation, which guarantees repeatable reads within a transaction. You will not see partial updates from concurrent transactions.

DuckDB does not support multiple processes writing to the same database simultaneously. If you need multi-process writes, you must implement coordination at the application level (using file locks or an external coordinator) or use a different database for the write path and query it from DuckDB using connectors.

For query execution, DuckDB uses morsel-driven parallelism. A query is broken into pipelines, and each pipeline processes data in small chunks called morsels. Worker threads pull morsels from a shared queue and process them. This design allows DuckDB to automatically scale across all available CPU cores without manual configuration. The number of threads is controlled by the threads setting.

This concurrency model is well-suited for analytical workloads but not for high-concurrency OLTP. If you need hundreds of simultaneous writers with strict isolation guarantees, DuckDB is not the right tool. But for its intended use case, it provides a clean, efficient model that works well in practice.

Chapter 2: Getting Started with DuckDB

Installation Across Platforms

DuckDB is designed to be trivially easy to install. There are no external dependencies, no server process to manage, and no complex configuration. Depending on your use case, you can install it as a command-line tool, a language library, or both.

Command-Line Interface

The DuckDB CLI is the fastest way to start experimenting. Download it from the official website at duckdb.org. Pre-built binaries are available for macOS, Linux, and Windows.

On macOS with Homebrew:

```
brew install duckdb
```

On Ubuntu/Debian, download the binary from GitHub releases:

```
wget https://github.com/duckdb/duckdb/releases/download/v1.5.0/duckdb_cli-linux-amd64.zip
unzip duckdb_cli-linux-amd64.zip
chmod +x duckdb
./duckdb --version
```

On Windows, download the executable from the releases page or use Scoop:

```
scoop install duckdb
```

After installation, verify by running:

```
duckdb --version
```

Python

The Python client is the most popular way to use DuckDB. Install it with pip:

```
pip install duckdb
```

This installs the latest stable version along with native extensions for your platform. The package requires Python 3.9 or later. To use specific features like Parquet reading, you may need additional packages:

```
pip install duckdb pyarrow
```

R

Install the DuckDB R package from CRAN:

```
install.packages("duckdb")
library(duckdb)
```

The duckplyr package provides seamless integration with dplyr:

```
install.packages("duckplyr")
```

Other Languages

DuckDB offers official clients for many languages. Install via your language's package manager:

- Go: go get github.com/marcboeker/go-duckdb
- Java: Include the duckdb_jdbc JAR from Maven Central or the releases page
- Rust: cargo add duckdb
- Node.js: npm install @duckdb/node-api
- C/C++: Download the libduckdb package from GitHub releases

For a complete list of supported languages and installation instructions, refer to the official documentation.

The DuckDB Command-Line Interface

The DuckDB CLI is a full-featured interactive SQL shell. Launch it by running duckdb without arguments. This creates an in-memory database that is discarded when you exit.

```
duckdb
D v1.5.0 (a1f3b2c)
Type .help for help
D
```

To create or connect to a persistent database file, pass a path:

```
duckdb my_analytics.duckdb
```

The first time you run this command, DuckDB creates the file. Subsequent runs open the existing database.

Basic Commands

Create a table and insert data:

```
CREATE TABLE sales (
    id INTEGER,
    product VARCHAR,
    quantity INTEGER,
    price DECIMAL(10,2),
    sale_date DATE
);

INSERT INTO sales VALUES
    (1, 'Laptop', 2, 999.99, '2025-01-15'),
    (2, 'Mouse', 5, 29.99, '2025-01-16'),
    (3, 'Keyboard', 3, 79.99, '2025-01-16'),
    (4, 'Monitor', 1, 349.99, '2025-01-17');
```

Run a query:

```
SELECT product, SUM(quantity) AS total_quantity,
       SUM(quantity * price) AS revenue
FROM sales
GROUP BY product
ORDER BY revenue DESC;
```

Output:

```
┌──────────┬────────────────┬─────────┐
│ product  │ total_quantity │ revenue │
│ varchar  │ bigint         │ double  │
├──────────┼────────────────┼─────────┤
│ Laptop   │              2 │ 1999.98 │
│ Monitor  │              1 │ 349.99  │
│ Keyboard │              3 │ 239.97  │
│ Mouse    │              5 │ 149.95  │
└──────────┴────────────────┴─────────┘
```

Dot Commands

The CLI supports dot commands for meta-operations:

- .help: Show available commands
- .tables: List all tables in the database
- .schema table_name: Show the schema of a table
- .mode csv/table/json: Set output format
- .output filename: Redirect output to a file
- .read filename.sql: Execute SQL from a file
- .exit or .quit: Exit the CLI

Example usage:

```
.tables
.schema sales
.mode csv
.output results.csv
SELECT * FROM sales;
.exit
```

Querying Files Directly

One of DuckDB's most powerful features is querying files directly without importing them. You can query CSV, Parquet, and JSON files as if they were tables:

```
-- Query a CSV file directly
SELECT * FROM 'sales_data.csv' LIMIT 10;

-- Query a Parquet file
SELECT product, SUM(quantity)
FROM 'warehouse/sales.parquet'
GROUP BY product;

-- Use glob patterns to query multiple files
SELECT * FROM 'data/sales_*.csv';
```

DuckDB automatically detects the file format from the extension and infers the schema. For CSV files, you can override auto-detection with parameters:

```
SELECT * FROM read_csv('sales.csv',
    delim = ',',
    header = true,
    columns = {price: 'DECIMAL(10,2)'})
LIMIT 5;
```

Your First Analytical Queries

Let us walk through a realistic analytical workflow using DuckDB. Suppose you have sales data in a CSV file and want to understand performance by region and product category.

First, examine the data structure:

```
SELECT * FROM 'sales.csv' LIMIT 5;
```

Get a quick summary of all columns:

```
SUMMARIZE 'sales.csv';
```

The SUMMARIZE command is a DuckDB-specific feature that computes min, max, count, approximate unique values, null percentages, and quantiles for every column. It is invaluable for initial data exploration.

Now let us run some analytical queries. Sales by region:

```
SELECT
    region,
    COUNT(*) AS transaction_count,
    SUM(amount) AS total_revenue,
    AVG(amount) AS avg_transaction_value,
    STDDEV(amount) AS revenue_stddev
FROM 'sales.csv'
GROUP BY region
ORDER BY total_revenue DESC;
```

Monthly trends:

```
SELECT
    DATE_TRUNC('month', sale_date) AS month,
    COUNT(*) AS transactions,
    SUM(amount) AS revenue
FROM 'sales.csv'
GROUP BY month
ORDER BY month;
```

Top products with running totals:

```
WITH product_sales AS (
    SELECT
        product,
        SUM(amount) AS total_revenue
    FROM 'sales.csv'
    GROUP BY product
)
SELECT
    product,
    total_revenue,
    SUM(total_revenue) OVER (ORDER BY total_revenue DESC) AS running_total,
    ROUND(100.0 * total_revenue / SUM(total_revenue) OVER (), 2) AS pct_of_total
FROM product_sales
ORDER BY total_revenue DESC;
```

This query demonstrates several advanced features: a Common Table Expression (CTE), window functions (SUM with OVER clause), and aggregation. We will cover all of these in depth in later chapters.

Working with Databases and Connections

DuckDB supports multiple database connection modes, each suited to different scenarios.

In-Memory Database

An in-memory database is created when you connect without specifying a file path. All data lives in RAM and is lost when the connection closes. This is ideal for temporary analysis, testing, or processing pipelines where persistence is not needed.

In Python:

```
import duckdb

conn = duckdb.connect()  # In-memory
conn.execute("CREATE TABLE temp AS SELECT 1 AS x")
result = conn.execute("SELECT * FROM temp").fetchall()
print(result)  # [(1,)]
# Table is gone when conn is closed
```

In the CLI:

```
duckdb :memory:
```

Persistent Database

Specify a file path to create or open a persistent database. Data survives across sessions.

```
import duckdb

conn = duckdb.connect("my_analytics.duckdb")
conn.execute("CREATE TABLE sales AS SELECT 1 AS id, 'Widget' AS product")
# Close and reopen
conn.close()

conn2 = duckdb.connect("my_analytics.duckdb")
result = conn2.execute("SELECT * FROM sales").fetchall()
print(result)  # [(1, 'Widget')]
```

Read-Only Mode

Open a database in read-only mode to prevent accidental modifications:

```python
import duckdb

conn = duckdb.connect("analytics.duckdb", read_only=True)
# conn.execute("INSERT INTO sales VALUES (2, 'Gadget')")  # Would fail
result = conn.execute("SELECT COUNT(*) FROM sales").fetchone()
print(result)
```

Attaching Multiple Databases

DuckDB supports attaching multiple databases in a single session. This is useful for querying data across different sources or schemas:

```sql
ATTACH 'warehouse.duckdb' AS warehouse;
ATTACH 'staging.duckdb' AS staging;

-- Query across databases
SELECT w.product, w.revenue, s.units_shipped
FROM warehouse.sales_summary w
JOIN staging.shipments s ON w.product = s.product;
```

Each attached database has its own schemas and tables. You reference objects using the database name as a prefix.

Basic Data Exploration Workflow

Here is a practical workflow for exploring new data with DuckDB, combining the techniques we have covered:

1. **Load or locate your data**: Identify whether your data is in files, an existing DuckDB database, or another system.
2. **Quick inspection**: Use LIMIT to see sample rows and SUMMARIZE to understand column statistics.

```sql
SELECT * FROM 'data.csv' LIMIT 10;
SUMMARIZE 'data.csv';
```

3. **Check data quality**: Look for nulls, duplicates, and out-of-range values.

```
SELECT
    COUNT(*) AS total_rows,
    COUNT(DISTINCT id) AS unique_ids,
    SUM(CASE WHEN amount IS NULL THEN 1 ELSE 0 END) AS null_amounts,
    MIN(amount) AS min_amount,
    MAX(amount) AS max_amount
FROM 'data.csv';
```

4. **Understand distributions**: Use histograms and grouping to see how values are distributed.

```
SELECT
    HISTOGRAM(amount, bins := 20) AS amount_distribution
FROM 'data.csv';
```

5. **Answer business questions**: Write queries that address specific analytical needs.

```
-- What are the top 10 customers by revenue?
SELECT customer_id, SUM(amount) AS total_revenue
FROM 'data.csv'
GROUP BY customer_id
ORDER BY total_revenue DESC
LIMIT 10;
```

6. **Save results**: Export your analysis for reporting or further processing.

```
COPY (
    SELECT * FROM top_customers_analysis
) TO 'top_customers.csv' (HEADER, DELIMITER ',');
```

This workflow applies whether you are using the CLI, Python, R, or any other DuckDB client. The SQL is the same everywhere.

Chapter 3: Data Types, Schemas, and Tables

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Sample Database Setup for Chapters 3 through 7

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Primitive and Composite Data Types

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Structs, Lists, Maps, and Unions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Creating and Managing Tables

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Constraints and Integrity

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Temporary and Transient Tables

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Chapter 4: Core SQL Operations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

SELECT Statements and Projection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Filtering with WHERE and Boolean Logic

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Sorting, Limiting, and Offset

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

DISTINCT and Deduplication Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Common Pitfalls and Best Practices

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Chapter 5: Joins and Set Operations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Inner Joins and Outer Joins

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Cross Joins and Semi/Anti Joins

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Join Algorithms and When DuckDB Chooses Each

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

UNION, INTERSECT, EXCEPT

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Large-Scale Join Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Chapter 6: Aggregations and Window Functions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

GROUP BY and Aggregate Functions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

HAVING and Filtering Aggregates

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Window Functions Syntax and Semantics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Analytical Patterns with Windows

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Performance Considerations for Heavy Aggregation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Chapter 7: Advanced SQL Features

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Common Table Expressions (CTEs)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Subqueries and Correlated Queries

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Views and Materialized Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Macros and Procedural Extensions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Transactions and Isolation Levels

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

LATERAL Joins for Row-by-Row Computation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

PIVOT and UNPIVOT for Data Reshaping

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

QUALIFY Clause for Window Function Filtering

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

SAMPLE Clause for Efficient Data Sampling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

POSITIONAL Joins for Row-Order Matching

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Advanced STRUCT Operations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Chapter 8: Data Ingestion and Export

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

CSV Import and Export

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

JSON Processing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Parquet Integration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Avro, ORC, and Other Formats

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

COPY Statements and Bulk Operations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Chapter 9: The Extensions Ecosystem

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Extension Architecture and Loading

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Spatial and Geospatial Extensions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

HTTP and Web Data Access

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Apache Iceberg and Delta Lake

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

MotherDuck and Cloud Integration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Chapter 10: Programming APIs and Language Bindings

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Python API and duckdb Package

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

R Integration via dbplyr and duckdb

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Java and JDBC Connectivity

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

C/C++ Embedding API

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Go, Rust, Node.js, and Other Bindings

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Chapter 11: DataFrame and Ecosystem Integrations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Pandas Integration and Lazy Evaluation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Polars and Arrow Interoperability

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Apache Arrow Flight and Memory Sharing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Ibis Integration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

dbt and Data Transformation Workflows

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Jupyter, Notebooks, and Interactive Analysis

Chapter 12: Performance Engineering

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Understanding Vectorized Execution Performance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Parallel Query Execution Tuning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Storage Layout and Compression Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Predicate Pushdown and Projection Pushdown

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Late Materialization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Join Optimization and Hash Tables

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Memory Limits, Caching, and Spilling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Reproducible Benchmark Suite

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Chapter 13: Embedding DuckDB in Applications

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

The Embedded Database Paradigm

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Building Analytics Into Your Application

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Connection Management and Lifecycle

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Packaging and Distribution Considerations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Real-World Embedding Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Chapter 14: Production Deployment and Operations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Security Model and Access Control

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Configuration and Pragmas Reference

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Profiling Queries and Debugging Performance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Migration Strategies from SQLite and PostgreSQL

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Enterprise Patterns and Best Practices

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Conclusion: The Future of Analytical Databases

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Where DuckDB Fits in the Modern Stack

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Emerging Trends and Capabilities

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Choosing Your Tool: A Decision Framework

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Index

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

A

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

B

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

C

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

D

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

E

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

F

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

G

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

H

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

I

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

J

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

L

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

M

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

N

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

O

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

P

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Q

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

R

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

S

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

T

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

U

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

V

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

W

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.

Z

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/theduckdbhandbook.