DuckDB Analytical SQL shell

START HERE

`duckdb <file.db>`
Open or create a database

`COPY <table> TO '<file.parquet>';`
Export to a file

`FROM '<file.csv>';`
Query a file directly

`.exit`
Leave the shell

`.tables`
List tables

`DESCRIBE <table>;`
Show columns and types

`SUMMARIZE <table>;`
Per-column statistics

`.mode markdown`
Change output format

Files CSV, Parquet, JSON → `read_csv / FROM` → **SQL engine** in-process → `CREATE TABLE AS` → **Database file** .db, optional → `COPY ... TO` → **Output files** CSV, Parquet, JSON

CONCEPTS

With no file argument DuckDB runs in memory and everything is lost on exit; pass a file to persist.

Files are first-class tables: no import step is needed to query them, only to keep them in a database.

Dot commands start the line, take no semicolon and are not SQL; SQL statements end with a semicolon.

START AND RUN

`duckdb`
Start an in-memory session

`duckdb <file.db>`
Open or create a persistent database

`duckdb -readonly <file.db>`
Open without write access

`duckdb -c "<sql>"`
Run one statement and exit

`duckdb -csv -c "<sql>"`
Print the result as CSV

`duckdb -json -c "<sql>"`
Print the result as JSON

`duckdb <file.db> -f <script.sql>`
Run a script file and exit

`duckdb < <script.sql>`
Feed SQL from stdin

START AND RUN (cont.)

`duckdb -version`
Print the version

DOT COMMANDS

`.help`
List all dot commands

`.tables`
List tables with columns and row counts

`.schema <pattern>`
Show CREATE statements

`.databases`
List attached databases

`.open <file.db>`
Switch to another database

`.read <file.sql>`
Run SQL from a file

`.timer on`
Show query timings

`.maxrows <n>`
Rows shown in the default table mode

`.quit`
Leave the shell
Ctrl+D also exits.

OUTPUT

`.mode <mode>`
Set the output format
Modes include duckbox (default), box, table, csv, json, jsonlines, markdown, html, line, list, tabs, insert <table>.

OUTPUT (cont.)

`.mode line`
One value per line, good for wide rows

`.headers on`
Show header row (not in duckbox)

`.output <file>`
Send all output to a file

`.once <file>`
Send only the next result to a file

`.output`
Send output back to stdout

EXPLORE DATA

`FROM <table>;`
FROM-first query, SELECT * implied

`DESCRIBE <table>;`
Columns and types

`DESCRIBE FROM '<file>';`
Infer the schema of a file

`SUMMARIZE <table>;`
Min, max, nulls, quantiles per column

`SHOW TABLES;`
List tables

`FROM <table> USING SAMPLE 10 ROWS;`
Random sample of 10 rows

`SELECT count(*) FROM '<file>';`
Row count of a file

READ FILES

`FROM '<file.csv>';`
Query a CSV directly

`FROM '<file.parquet>';`
Query a Parquet file

`FROM '<file.json>';`
Query a JSON file

`FROM '<dir>/*.parquet';`
Glob many files as one table

`FROM read_csv('<f>', header=true, delim=';');`
CSV with explicit options

`FROM read_csv('<f>', all_varchar=true);`
Read every column as text

`FROM read_csv('<f>', columns={'a':'INT', 'b':'DATE'});`
Set the schema by hand
Disables type auto-detection.

`FROM read_csv('<f>', ignore_errors=true);`
Skip malformed rows

`FROM read_json('<f>');`
JSON with explicit reader

`FROM read_csv('/dev/stdin');`
Read piped input

TABLES AND IMPORT

`CREATE TABLE <t> AS FROM '<file>';`
Load a file into a table

TABLES AND IMPORT (cont.)

`CREATE OR REPLACE TABLE <t> AS <query>;`
Rebuild a table

`INSERT INTO <t> FROM '<file>';`
Append file rows

`CREATE VIEW <v> AS <query>;`
Save a query

EXPORT

`COPY <table> TO '<file.csv>' (HEADER);`
Export a table as CSV

`COPY (<query>) TO '<file.csv>';`
Export a query result

`COPY <table> TO '<file.parquet>' (FORMAT parquet);`
Export as Parquet

`COPY ... (COMPRESSION zstd);`
Parquet compression
Options: snappy (default), gzip, zstd, brotli, lz4.

`COPY (<query>) TO '<file.json>' (FORMAT json, ARRAY true);`
Export a JSON array

`COPY <table> FROM '<file.csv>' (HEADER);`
Load a CSV into an existing table

EXPORT (cont.)

`EXPORT DATABASE '<dir>';`
Dump schema and data to a directory

`IMPORT DATABASE '<dir>';`
Restore an export into an empty database

SQL EXTRAS

`SELECT * EXCLUDE (<col>) FROM <t>;`
All columns except some

`SELECT * REPLACE (<expr> AS <col>) FROM <t>;`
Change one column in place

`SELECT <col>, count(*) FROM <t> GROUP BY ALL;`
Group by every non-aggregate column

`SELECT COLUMNS('<regex>') FROM <t>;`
Select columns by pattern

`SELECT * FROM <t> QUALIFY row_number() OVER (PARTITION BY <k> ORDER BY <c>) = 1;`
Filter on a window function

`PIVOT <t> ON <col> USING sum(<val>);`
Rows to columns

SETTINGS AND EXTENSIONS

`ATTACH '<file.db>' AS <name>;`
Attach another database