SQLite CLI Shell for SQLite

CONCEPTS

Dot-commands are shell commands, not SQL: they sit alone on one line, need no semicolon and cannot be used from a library. SQL statements need a closing ;.

Input file, SQL arg, stdin
↓ `-cmd -init`
Shell dot-commands + SQL
↓ `.mode .headers`
Formatter box, csv, json...
↓ `.once .output`
Output stdout, file or pipe

Without a filename (or with :memory:) you get a temporary database that vanishes on exit. Use .save <file> to keep it.

OPEN AND RUN

`sqlite3 <file.db>`
Open (or create) a database

`sqlite3 <file.db> "<sql>"`
Run one statement and exit

`sqlite3 <file.db> < <script.sql>`
Run a SQL script

`.read <script.sql>`
Run a script from inside the shell

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

`sqlite3 -readonly <file.db>`
Open without write access

`.help`
List all dot-commands

`.quit`
Exit the shell

`.shell <cmd>`
Run a shell command

`sqlite3 -bail -batch <file.db>`
Stop at the first error

OUTPUT MODES

`.mode box`
Unicode box table

`.mode column`
Aligned plain columns

`.mode table`
ASCII table

OUTPUT MODES (cont.)

`.mode markdown`
Markdown table

`.mode csv`
Comma separated values

`.mode json`
JSON array of objects

`.mode line`
One column per line

`.mode list`
Default pipe-separated

`.headers on`
Print column names

`.nullvalue <text>`
Show NULL as text

`.show`
Show current settings

INSPECT SCHEMA

`.tables`
List tables and views

`.tables <pattern>`
Filter with a LIKE pattern

`.schema`
All CREATE statements

`.schema <table>`
CREATE statements for one table

`.schema --indent`
Pretty-printed schema

`.indexes <table>`
List indexes

`.databases`
Attached databases and files

`pragma table_info(<table>);`
Columns and types

IMPORT AND EXPORT

`.import --csv <file> <table>`
Load CSV into a table

`.import --csv --skip 1 <file> <table>`
Load into an existing table

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

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

`.output`
Back to stdout

BACKUP AND RECOVER

`.backup <file>`
Binary copy of main database

`.restore <file>`
Load a backup into this database

`.save <file>`
Write database to a file

`.dump`
Whole database as SQL text

`.dump <table>`
One table as SQL text

`sqlite3 <old.db> .dump | sqlite3 <new.db>`
Rebuild into a fresh file

`pragma integrity_check;`
Check for corruption

DEBUG AND TUNE

`.timer on`
Time each statement

`.eqp on`
Show query plan automatically

`explain query plan <sql>;`
Plan for one statement

RECIPES

CSV into a new database

```
sqlite3 app.db
# inside the shell:
.import --csv people.csv people
.schema people
select count(*) from people;
```

Export a query to CSV

```
sqlite3 -header -csv app.db
"select * from people" >
people.csv
# or inside the shell:
.headers on
.mode csv
.once people.csv
select * from people;
```

Safe live backup

```
sqlite3 app.db ".backup
'snap.db'"
sqlite3 snap.db "pragma
integrity_check;"
```

Quick look at an unknown database

```
.tables
.mode box
.headers on
.schema --indent
select * from <table> limit 5;
```