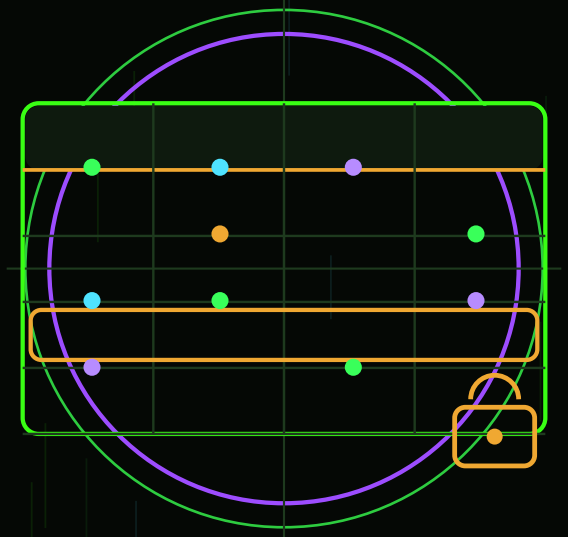


DB CONSOLE

DATABASE CONSOLE

Your databases — browsed, queried, and changed only when you say so, with a backup underneath.

HOW IT WORKS

USER MANUAL · OCTOBER 2026

777 SOFTWARE

777 ADMIN TOOLS

1 — WHAT DB CONSOLE IS

A database is where a program keeps its facts: the users, the orders, the settings, the history. A database admin console is the workbench for looking inside one — seeing its tables, reading its rows, asking it questions in its own language (SQL), and, when the admin decides, changing what's there.

DB Console is that workbench as a program that lives on your own computer. It began as a small tool built for this house's own admin work; the standalone keeps that tool's soul exactly — it finds your databases for you, it backs up before it writes, and it blocks every change until the admin says so — and grows it into a desktop program any admin can point at their own SQLite and PostgreSQL databases.

> DB Console's whole personality is protecting the admin from the fat-finger. Every database tool on Earth will happily run the worst query of your life at 3 a.m. This one treats your own worst moment as a design case: it starts read-only, it makes you review every change before it runs, it copies the database before any write lands, and it makes the truly dangerous statements hard to fire by accident. Sections 5–7 are that system, piece by piece.

What it is not: it is not a server, and it is not a service that runs when you're not looking. It opens when you open it, it connects only where you point it, and when you close it, it's closed. There are no accounts, nothing reports back to anyone, and the program ships knowing how to talk to databases — never where yours live (§10).

2 — CONNECTING TO YOUR DATABASES

DB Console speaks to two kinds of database today. They connect differently, because they live differently:

Kind	How you connect
SQLite — a database that is a file	A SQLite database is one file on a disk. Open one directly, or let the console find them for you: you approve the folders it may look in, it checks every candidate file's identity header to be sure it really is a SQLite database, and lists each one it finds — with its size and table count — once, even if a file turns up in two places. It looks only in folders you approved. There is no house list baked in: the program arrives knowing no folders at all.

Kind	How you connect
PostgreSQL — a database that is a server	A PostgreSQL database lives on a server, so you fill in a short form: the server's address, its port, the database name, and your username — an address like `db.example.local`. The address field ships empty: the first address it ever holds is one you typed. The connection it makes is a real one, to the server you named — the console never pretends to be connected to something it hasn't actually reached.

Saved connections

- Connections you keep are saved by name, in your own words — “Shop Postgres”, “Laptop data store.” A saved connection remembers the address, port, username, and file paths — the directions, not the keys.
- Passwords are never written into that saved list. The saving code itself strips out anything shaped like a password before the list touches the disk. Passwords live in one place only: the encrypted vault (§8).
- Every new connection starts read-only (§5) — including ones you saved long ago and simply haven't flipped. Reading is always allowed; changing anything takes a decision you make on purpose.

3 — LOOKING AROUND: TREE, GRID, STRUCTURE

Once you're connected, the window gives you three ways of looking at the same database. Together they answer “what's in here, and what does it look like?”

View	What it shows you
The schema tree	The database as an outline: its tables and views, each opening to show its columns. This is the map — what's here, before you touch anything.
The data grid	One table's rows, 50 to a page, with page controls. Click a column heading to sort by that column. A search box hunts your words across every column at once. Empty cells show as NULL — the database's word for “nothing stored here” — and binary data shows as a plain label like [blob 12B]: shown as what it is, never guessed at, never a crash.
The structure view	One table, taken apart: every column with its type and default, the primary key, the foreign keys that link it to other tables (in both directions), its indexes, and the table's real CREATE statement — the exact words that built it, readable and copyable.

Looking changes nothing. You can browse a production database all day in this window and the database will never know you were there — that is what read-only means, and it is the state you start in (§5).

4 — THE SQL EDITOR: ASK IT YOURSELF

The editor is where you write SQL — the database's own language — when the tree and grid aren't the question you need answered. Type a statement, run the whole thing or just the part you've selected, and the answer lands in a results grid under the editor, in the same clean table as the browser.

- The 500-row cap, stated out loud. A question can match a million rows; the grid shows the first 500 and says so. A capped answer is labeled capped — never quietly trimmed and presented as the whole truth.
- Take the answer with you. Results export to a CSV or JSON file on your own disk — yours, in open formats, to open in anything.
- Reading statements run. A question (a SELECT) just runs — reading is what the console is for.
- Changing statements don't run — not yet. If your statement would change something (INSERT, UPDATE, DELETE, DROP, and their kin), it does not execute when you press run. It goes to the pending-changes tray (§6), and the safety system takes it from there. That detour is not a refusal; it is the product working as designed.

> One sentence to remember: in DB Console, running a change and landing a change are two different acts. You can write anything. Nothing lands until it has passed the tray, the warnings, and your Commit.

5 — READ-ONLY BY DEFAULT: THE GOLD FRAME

Every connection — a file you just opened, a server you just saved, one you saved a year ago — starts read-only, and stays that way until you personally flip that connection to writable. The flip is per connection and remembered: your scratch database can live writable while the production one never leaves gold.

And you can always see which world you're in. The window frame itself carries the mode, in color, at all times:

Frame	What it means
GOLD — READ-ONLY	You can look at everything and change nothing. Changing statements are refused a landing place — they can't even enter the tray. This is the starting state of every connection.
RED — WRITABLE	You flipped this connection on purpose. Changes are possible — through the tray, the backup, and the guards (§6–§7), never around them. The red edge is there so you can see your danger level from across the room.

Why default to gold? Because most admin work is reading — checking a value, hunting a row, answering a question — and the one time in a hundred that you're changing something is exactly the time you should have to say so out loud first. Read-only by default means the accident has to be deliberate before it can be real.

6 — THE PENDING-CHANGES TRAY: REVIEW BEFORE IT RUNS

On a writable connection, a changing statement begins a journey with four stops — and it can be thrown away at any of them:

1. It is written. You type the statement in the editor and run it. Nothing has happened to the database.
2. It waits in the tray. The statement collects in the pending-changes tray, which shows you — in plain SQL, exactly as it will run — every statement about to execute. A counter in the window frame keeps the count in front of you: “3 changes pending.”
3. It is warned about, if it deserves a warning. Some statements trip the guards (§7) and stop here for a plain-words conversation before they can go further.
4. You decide. Commit runs the whole tray — after the backup (§7) is safely made. Discard throws it away, and the database never hears about it.

The tray is the difference between a console that does what you typed and one that does what you meant. A typo in a statement is a curiosity while it sits in the tray, readable at your leisure. The same typo, executed on click, is an incident. DB Console keeps every change a curiosity until you personally promote it.

> Close the window with changes pending and the console asks you, by name, what to do with them. Nothing pending ever evaporates silently — and nothing pending ever sneaks through, either.

7 — BACKUP BEFORE WRITE, AND THE GUARDS

The backup

When you press Commit, the console's first act is not the change — it is the copy. For a SQLite database, the whole file is copied to a timestamped backup (the same name, with .bak and the time added), and only then does the tray run. The newest ten backups are kept. Two details matter:

- The copy is made before anything executes — so if a write then fails halfway, the backup still exists. The net is woven before the jump, not after the fall.
- A bad write is therefore always undoable: put the backup file back in the database's place and you are standing where you stood before the change — every row, every table.

For a PostgreSQL server there is no single file to copy, and this manual won't pretend otherwise. There, your net is the rest of the system, and it is real: read-only by default (§5), every change reviewed in the tray (§6), and the guards below — plus whatever backups the server itself keeps, which stay the server admin's own good practice.

The guards — plain words before plain damage

Statement	What the console says and does
DELETE or UPDATE with no WHERE	A WHERE clause is the part that says which rows. Without one, the statement means every row — so the console says exactly that, in plain words, before the change can be committed: this will change every row in the table. Probably not what you meant — and if it is what you meant, you can say so and proceed, eyes open.
DROP or TRUNCATE	These don't change a table; they end one. They stay disarmed until you type the table's exact name to arm the Commit — no checkbox to tick by accident, no button a stray click can fire. And this guard lives in the console's core, not just its buttons: a DROP that hasn't been armed with the typed name cannot commit, by any road.

Everything else that changes data gets the tray's review as its examination. Nothing destructive-looking ever gets a silent run.

8 — THE VAULT: WHERE PASSWORDS LIVE

A server connection needs a password. DB Console's rule is absolute: passwords are never plain text, anywhere — not in the saved-connections list (§2), not in a log, not on a screen after you've typed them. They live in an encrypted vault: one file, on your own computer, sealed with a master password that you choose.

- The master password is never stored. Not in the vault, not anywhere. If you forget it, the vault stays sealed — that is not a flaw in the design; that is the design. Choose one you'll remember.
- How the sealing works, in plain words: your master password is run through a one-way mill — a derivation called PBKDF2 — 600,000 times, with a random ingredient (a salt) mixed in, to make the key. The secrets themselves are sealed with AES-GCM, the standard authenticated encryption: change one bit of the sealed file and it will not open, rather than open wrong.
- The vault file is locked to your account alone (permissions 0600) — other users of the same machine can't read it at all, and without the master password neither can anyone else.
- A wrong password fails loudly. Type the wrong master password and the vault refuses, in plain words. It never half-opens, never guesses, never lets a wrong key rattle around inside.
- A fallback is built in, and proven. On a system missing the standard encryption library, the console seals the vault with a built-in method built on the same principles — same

600,000-round derivation, same account-only lock — instead of ever falling back to plain text. There is no road in this program that ends with a password sitting readable on a disk.

> The whole credential story in one line: directions in the open, passwords in the vault, and the vault's key exists only in your head.

9 — THE HONEST LIMITS

A tool you can trust tells you what it cannot do — and what it simply doesn't do yet. This is the complete list for this build:

- Two engines today. This build speaks SQLite and PostgreSQL. MySQL and MariaDB are on the plan for a later phase — they are not in this build, and nothing in this manual should be read as saying they are.
- Changes are written, not clicked. In this phase you change data by writing SQL in the editor; the changes then pass through the tray, the backup, and the guards like any other write. A point-and-click row-editing form is later-phase work — the safety system was built first, on purpose.
- The backup copy is a file's story. The timestamped .bak (§7) protects SQLite databases, which are files. A PostgreSQL server's safety here is the read-only default, the review tray, and the guards — stated plainly, because pretending a file copy exists for a server would be a lie with a progress bar.
- Two small cosmetics, owned. On a narrow window (800 pixels wide) the grid's page buttons can clip at the right edge, and on a short window (600 pixels high) the schema tree gets squeezed. Both are known, both are cosmetic, and neither touches a single byte of your data.
- Proven on test databases; the founder's test is the gate. Every protection in this manual was proven against purpose-built test databases — 37 core checks and 15 window checks, all passing. The program has never been pointed at a real working database in testing, on purpose. It is released only after the founder has used it with his own hands.

10 — YOUR MACHINE, YOUR BUSINESS

The privacy story is short enough to print whole:

- No accounts. No telemetry. No update checks. The console never reports back to anyone, because there is nobody for it to report to.
- Its only network traffic is connections you define — to your own database servers, when you connect to them. Before this build was allowed out, its insides were scanned string by string: it contains no built-in addresses at all. Not ours, not anyone's. Every address it will ever hold is one you typed.

- Everything it keeps lives with you: the saved connections, the vault, your exported CSV and JSON files — plain local files on your own machine, exportable and deletable, yours.
- It can only hurt what you point it at — and the whole safety system (§5–§7) exists so that even there, hurting something takes a reviewed, backed-up, deliberate act. An admin tool should make the careful thing the easy thing. This one was built to.

> That's the whole of it. Open your databases, read them in the gold frame, and when a change is truly needed — flip the connection, review the tray, let the backup land, and commit with your eyes open. The console's promise is not that nothing can go wrong. It's that nothing goes wrong by accident.

DB Console — Enclave DB Console · 777 Software · 777 Admin Tools. This manual ships with the program. Your databases are yours, run your way; the shape of this tool — read-only first, review before running, a backup under every write — does not change.