DBCraft AI Studio — User Guide

Product version 1.0.0 · Windows desktop edition

Table of contents

Part I — Getting started

1. About this guide

1.1 Who this guide is for

This guide is written for two audiences, and most chapters serve both:

- **Developers** who write SQL and PL/SQL, browse and modify schema objects, debug stored programs, and build automated tests.
- **Database administrators** who monitor sessions, examine storage, run health checks, compare schemas, tune SQL, and script routine maintenance.

No prior experience with DBCraft AI Studio is assumed. Working knowledge of Oracle Database concepts (schemas, tablespaces, sessions, PL/SQL) is assumed.

1.2 Conventions used

Convention	Meaning
Bold	A label you see on screen — a button, menu item, tab, or field name.
`Monospace`	Something you type, a file path, an identifier, or SQL.
File → Preferences…	A path through menus; each arrow is one level down.
`Ctrl+K`	Press these keys together.
□	A warning: the action changes or destroys data, or affects other users.
□	A tip or a piece of context worth knowing.

1.3 How to use this guide

Read Part I once, end to end, before your first session. After that the guide is a reference — each module has its own chapter, and each chapter is self-contained. The keyboard shortcut table (chapter 37) and the troubleshooting chapter (42) are the two pages worth bookmarking.

---

2. What DBCraft AI Studio is

DBCraft AI Studio is a desktop IDE and DBA toolkit for **Oracle Database**. It combines, in one window, the tools that a typical Oracle team otherwise spreads across half a dozen applications:

- a multi-tab **SQL and PL/SQL editor** with a persistent transaction, autocomplete, formatting and explain plans;
- a **schema browser** with DDL generation, script generation, object comparison and data import;

- a real **PL/SQL debugger** driven by Oracle's `DBMS_DEBUG` — breakpoints, stepping, variables, watches, call stack;
- a **unit test manager** for PL/SQL, with suites, expected outputs and performance thresholds;
- **DBA dashboards** — live metrics, health scanning, tablespace and datafile browsing, ASM and RAC overviews;
- **session monitoring** with a blocking-lock tree and session termination;
- **schema compare and synchronise**, DDL export, data import/export;
- **object version history** with diffs and one-click restore;
- an **ER diagram modeler** that reverse-engineers a schema and can forward-engineer DDL;
- an **index advisor**, a **SQL profiler/optimizer** and a **concurrent-session load tester**;
- a **workflow automation** designer with cron scheduling;
- a **report catalogue** with parameters, HTML templates and branding;
- a SQL*Plus-style **terminal**;
- an **AI assistant** you point at your own AI endpoint.

2.1 How it is built (and why that matters to you)

The application ships as a single Windows program, but internally it has two parts:

- **The user interface** — the Electron desktop window you interact with.
- **A local service** — a small background process that runs on your own machine, started and supervised by the Electron application. Every database operation goes through it.

Two consequences are worth understanding, because they explain behaviour you will notice:

1. **Nothing leaves your machine.** The local service connects directly to your Oracle database. There is no cloud component, no telemetry, no account server. The only outbound connection the product ever makes is the one-off licence activation call (chapter 4), and any calls the AI assistant makes to the AI endpoint *you* configure (chapter 34).
2. **Most operations are stateless.** Each action opens a fresh database connection, does its work, and closes it. This keeps the tool from holding idle sessions on your server. There are three deliberate exceptions, all documented where they matter: the SQL editor's transaction (§16.6), the debugger's session (chapter 20), and a test run's shared session (chapter 21).

If the local service is not running, every panel reports the same thing: "*Cannot reach the backend.*" See chapter 41.

2.2 Where the product stores its own data

DBCraft keeps its own feature data — saved reports, test suites, load tests, object version history — in local database files, entirely separate from the Oracle database you connect to. Your Oracle database is never modified to make DBCraft work; there are no repository tables to install. See Appendix A for exact locations.

3. Installation

3.1 System requirements

	Requirement
Operating system	Windows 10 or Windows 11 (64-bit)
Disk	~500 MB for the application, plus space for logs and local feature data
Memory	4 GB minimum; 8 GB recommended when using the ER modeler or load tester
Network	TCP access to your Oracle listener (default port 1521). Internet access is needed once, for licence activation — and an offline path exists if you have none.
Oracle client	**Not required.** The database driver is built in. You do not need Oracle Instant Client, `tnsnames.ora`, or `ORACLE_HOME`.

3.2 Installing

1. Run the supplied `DBCraft AI Studio` MSI installer.
2. Read the **End User Licence Agreement** and accept it. The installer will not continue until you do.
3. Choose the installation folder, or accept the default.
4. Finish the installer and launch the application from the Start menu or desktop shortcut.

There is no activation step inside the installer — licensing happens on first launch, inside the application.

3.3 Upgrading and uninstalling

Run a newer MSI over the top to upgrade; your settings, saved reports, test suites and version history are preserved. Uninstall from **Settings** → **Apps** in Windows. Uninstalling does not remove your licence from the machine — if you are moving the licence to another computer, **deactivate it first** (§38.5).

---

4. Activating your licence

The first time you launch DBCraft AI Studio you see the **Activate DBCraft AI Studio** screen. Until the product is entitled to run, this screen is a hard gate — you cannot reach the login screen or any feature behind it.

4.1 Online activation (the normal path)

You will have received a licence key by e-mail when your registration was approved. It looks like:

```
DBCK-XXXXX-XXXXX-XXXXX-XXXXX
```

1. Paste the key into the **Enter your license key** box.
2. Click **Activate** (or press `Enter`).

That is the whole procedure. Behind the scenes the application generates a challenge unique to this machine, exchanges it with the licensing authority, and verifies the signed response locally. The licence is then stored on this machine in protected form. Your machine needs internet access for those few seconds only — after that, DBCraft never needs to go online again.

 The **machine code** shown at the bottom of the screen is a fingerprint of this computer's hardware. Support may ask for it. It contains no personal information.

4.2 Offline activation

For machines with no internet access, click **No internet on this machine? Activate offline**. The three-step exchange is:

Step	What you do
1	Click **Generate Activation Code**. A `DBCH1-…` code appears; copy it (there is a copy button). If you need a fresh one, click **Generate a new code**.
2	On any device that *does* have a browser, open your DBCraft licensing portal, paste the activation code, and copy the response code it returns.
3	Back in DBCraft, paste the response (`DBL1-…`) into the box and click **Activate**.

The verification is done entirely on your machine — the response carries a cryptographic signature that DBCraft checks offline.

To go back to the simpler path, click **Have a license key? Activate online**.

4.3 The evaluation period

If your build is configured with a trial, the activation screen shows **Evaluation period — *n* days remaining** and a **Continue trial** button in the bottom-right. Clicking it dismisses the screen for the current window. The notice reappears the next time you start the application, until you either activate or the trial ends.

When the trial ends, the gate becomes hard again and the banner reads "*Your evaluation period has ended.*"

4.4 Other states you may see

Banner	Meaning	What to do
This installation is not activated	No licence and no trial.	Enter a licence key.
Your license has expired	A time-limited licence passed its end date.	Renew — the panel switches to **Renew License**; the procedure is identical to activation.
Your license is bound to different hardware	Significant hardware changed, or the licence file was copied from another machine.	Contact your licence administrator to release the old seat, then activate again.

Banner	Meaning	What to do
System clock tampering was detected	The system clock is behind the last time DBCraft ran.	Set the clock correctly and restart the application. Licensing is suspended until then; no activation is possible in this state.
The stored license is invalid	The licence file is corrupt.	Re-activate.

4.5 Where to manage the licence later

Once you are inside the application, **Help → License…** opens the licence dialog at any time. It shows who the product is licensed to, the edition, licence ID, seat number, issue and expiry dates, and the machine code — and lets you enter a new licence or deactivate this machine. See chapter 38.

---

5. Connecting to a database

DBCraft has no user accounts of its own. **Connecting to a database *is* logging in.** When you sign out, you disconnect.

5.1 The login screen

![Connect to Oracle Database]

Field	Notes
Host	Hostname or IP of the database server. Defaults to `localhost`.
Port	Listener port. Defaults to `1521`.
Type	**Service** or **SID**. Choose which identifier your database is registered under.
Service Name / SID	The value for the type you chose. Defaults to `ORCL` for service name.
Username	The database user.
Password	That user's password.
Remember password	Off by default. See §5.3 before you turn it on.

Click **Connect & Login**. DBCraft tests the connection; on success the IDE opens. On failure the reason is shown in red at the top of the form — usually `ORA-01017` (bad username/password), `ORA-12541` (no listener), or `ORA-12514` (the listener does not know that service).

 Connecting as `SYS`. If you enter `sys` as the username, DBCraft automatically connects with `SYSDBA` privileges. You do not need to type `as sysdba`. The connection chip and the left panel will show a red **SYSDBA** badge so you always know when you are working with that authority.

5.2 What the connection determines

The account you connect as decides what the whole application can do:

- **The default schema** in the Object Browser, snippet templates and the Visual Query Builder is your own schema.
- **DBA-only panels** — the **System** section of Settings — unlock only when the account holds DBA privileges.
- **Health checks, session lists, storage views and ASM/RAC panels** need `SELECT` on the corresponding `DBA_*` / `V$` views. Where a privilege is missing, DBCraft reports the specific check as "*Unable to evaluate*" with the exact grant needed, instead of failing the whole panel.
- **Killing sessions** and **opening/closing PDBs** need `ALTER SYSTEM` and a `SYSDBA` session respectively.

5.3 Remembering the password (and the security behind it)

The **Remember password** checkbox is deliberately off by default. Here is exactly what happens when you tick it:

- Everything *except* the password (alias, host, port, service, username) is saved on this machine regardless of the checkbox — so the login form is pre-filled next time.
- The password is saved **only** when the box is ticked, and **only** as ciphertext produced by the Windows credential vault (DPAPI). It is never written in plain text.
- If the OS vault is unavailable, the checkbox is disabled and the text under it explains that the password cannot be saved. There is **no plaintext fallback** — by design.
- The password is never written to log files, never included in a bug report, and never transmitted anywhere except to your Oracle database.

Clear saved credentials — the small red link beneath the checkbox, shown when a saved profile exists — erases the saved profile. This is a deep erase, not a simple delete: the underlying storage is rewritten so that no remnant of the old value survives, while your settings and theme are preserved. The link also shows when the profile was last used.

 Signing out clears the saved profile too. "Remember" means "*don't ask me again next launch*", not "*keep it after I explicitly sign out*". If you sign out via **File → Sign out** or the profile menu, you will need to enter your credentials again.

5.4 Reconnecting and session restore

On launch, if a saved profile with a saved password exists, DBCraft restores the connection automatically and takes you straight to the IDE. If a profile exists but no password was saved, you land on the login screen with the fields pre-filled.

5.5 Multitenant (CDB/PDB) databases

If you connect to a container database, DBCraft detects it and shows a **container chip** next to the connection chip in the header. See chapter 9.

---

6. A tour of the workspace

When you connect, the IDE opens on the SQL Editor. The window never changes shape: the same regions stay in the same places, and every module you switch to appears in the same central area. This chapter walks through that window from the top down, so the rest of the guide can refer to each region by name.

6.1 What you see when the IDE opens

Reading the window from the top down you will find two header rows — the menus and the connection chip, then the icon toolbar — a narrow vertical strip of icons down the left edge, a schema tree beside it, the module area filling the centre, and a single-line status strip along the bottom. The AI assistant and the bottom dock stay out of sight until you ask for them.

Nothing in this tour is destructive. Switching modules, opening panels and dragging edges only change what you are looking at, and every layout choice is remembered between launches: rail position, panel widths, dock height, which dock tabs are open, and whether the AI panel is showing.

6.2 The header — menus and connection

The top row holds eight menus on the left — File, Edit, Session, Database, Tools, Debug, Terminal and Help — and on the right the notification bell, the light/dark toggle, the settings gear, the user menu and the Ctrl+K palette button.

In the middle sits the **connection chip**: the alias you connected with, your username and the host. On a container database a second chip names the container you are working in, and a red **SYSDBA** badge appears whenever you are connected with that authority — so the privilege you hold is always visible without going looking for it.

The second row is the **icon toolbar**: execute, stop, commit, rollback, explain plan, query builder, formatter, file operations, export and import, new window, history and snippets. It acts on the SQL editor no matter which module you are looking at.

Chapters 7, 8 and 9 document every menu, button and chip in these two rows.

6.3 The docking rail — switching modules

The vertical strip of icons down the left edge is the **docking rail**, and it is how you move around the product: one icon per module, development tools grouped at the top and DBA tools below. Hovering an icon names it; clicking it replaces the centre of the window and leaves everything else — including your open editor tabs — exactly as it was.

The rail itself can be docked, floated as a small palette, or collapsed off the edge when you want the width back. Chapter 10.

6.4 The left panel — finding your objects

Beside the rail sits the **left panel**: your connection at the top, and beneath it the schema object tree. Expand a schema to reach tables, views, procedures, packages, triggers and the rest, and right-click any object for the actions that apply to it — query its data, generate DDL, compare it with another, import into it.

The panel appears on the SQL Editor, Object Browser, Debugger and Test Manager, and is absent from the DBA modules because those have nothing to browse. Drag its right edge to resize it. Chapter 11.

6.5 The centre — where the work happens

The large area in the middle is the **module area**, and it shows whichever module the rail has selected: query tabs on the SQL Editor, and dashboards, lists, grids or diagrams on everything else. Parts III and IV of this guide are, chapter by chapter, a description of what appears here.

On the SQL Editor and the Debugger one further panel — the **AI assistant** — slides in from the right when you press Ctrl+Shift+A and slides away when you press it again. It is optional and inactive until you configure an AI endpoint of your own (chapter 34).

6.6 The bottom dock — results, terminal and monitors

Query results do not appear inside the editor; they appear in the **bottom dock**, which rises from the foot of the window as soon as a statement returns. Ctrl+` opens and closes it by hand.

The dock is tabbed. **Data Grid** is always there and holds the results of the last statement. **Terminal**, **Sessions**, **Storage** and **Live Metrics** open on demand from the menus or the command palette, and each can be closed individually without disturbing the others. Drag the top edge of the dock to make it taller. Chapter 12.

6.7 The status bar — reading the state at a glance

The single line across the bottom of the window is the **status bar**, and it answers the questions you would otherwise go looking for: which account is connected and to what, whether the last statement succeeded or failed, how many rows came back and how long they took, where the cursor is, which container you are in, the database version, and — when the editor is holding work you have not committed — a pending-transaction marker. Chapter 13.

6.8 Your first five minutes

If you would rather learn the window by using it, this is the shortest useful path through it:

1. **Connect.** Enter host, port, service name and your credentials; the IDE opens as soon as the connection is verified (chapter 5).
2. **Look around the left panel.** Expand your own schema and find a table you recognise (chapter 11).
3. **Ask for its data.** Right-click the table and choose the option that queries it. A new editor tab opens with the SQL already written for you.
4. **Run it.** Press F8, or the ▶ button on the toolbar. The bottom dock opens with the rows in the Data Grid (chapter 16).
5. **Read the status bar.** Row count, elapsed time, and whether anything is now sitting uncommitted (chapter 13).

That is the loop the whole product is built around — find an object, write or generate SQL against it, run it, read the result. Every other module is a specialised version of the same cycle.

6.9 The quickest way to get oriented

Press Ctrl+K to open the command palette, type a few letters of what you want, and press Enter. Every module and every major action is in there. If you would rather browse, choose Help → User guide to open a sliding panel that describes each rail icon in one sentence and jumps you straight to it.

Part II — The workspace

7. The menu bar

The top row of the header holds eight menus. This is the complete list.

File

Item	Action
New connection…	Opens the connection dialog (§7.9).
Preferences…	Opens **Settings** at the Appearance section (chapter 36).
Sign out	Disconnects, clears the saved credential profile, and returns to the login screen.

Edit

Item	Action
Find in editor	Switches to the SQL Editor. Use the editor's own `Ctrl+F`.
Snippets	Switches to the SQL Editor; open the snippet library from the toolbar star icon.

Session

Item	Action
Show / Hide AI panel	Toggles the AI assistant (`Ctrl+Shift+A`).
Show / Hide terminal	Toggles the bottom dock (`Ctrl+\``).
Sessions monitor	Opens the Session Browser module.

Database

Item	Action
Object browser	Opens the Object Browser module.
Schema tools	Opens Schema Tools.
ER diagrams	Opens the ER Diagram Modeler.
Storage / health	Opens DBA Dashboards.
Export Tables…	Opens the Export Tables wizard (chapter 35).
Import Tables…	Opens the Import Tables wizard (chapter 35).

Tools

Item	Action

Item	Action
Test manager	Opens the Unit Test Manager.
Load tester	Opens the Load Tester.
Automation	Opens the Automation designer.
Settings…	Opens Settings.

Debug

Item	Action
Debugger	Opens the PL/SQL Debugger.
Performance profiler	Opens the Performance Profiler.

Terminal

Item	Shortcut
New Terminal	`Ctrl+Shift+``
Split Terminal	`Ctrl+Shift+5`
New Terminal Window	`Ctrl+Shift+Alt+``

Help

Item	Action
User guide	Opens the sliding feature guide.
Report a bug…	Opens the feedback dialog with logs attached (chapter 39).
License…	Opens the licence dialog (chapter 38).

7.8 The right-hand controls

Control	What it does
Notifications	A bell with a dot when unread items exist. The dropdown lists app events — connection results, query failures, health-check summaries, commit/rollback outcomes, container switches. **Mark all read** and **Clear all** are in the header of the dropdown.
/ **Theme**	Switches between light and dark instantly.
Settings	Opens Settings at Appearance.
User menu	Shows the display name, the connected username, and a role badge. Contains **Profile settings**, **Preferences**, and **Sign out**.
⌘ **Ctrl K**	Opens the command palette.

7.9 The connection dialog

File → New connection… opens a dialog for describing a connection profile:

- **Connection Name** and a **colour** chip (six preset colours) used to identify it visually.
- **Connection Type** — Service Name, SID, or a full **TNS String** (paste a `(DESCRIPTION=(ADDRESS=…))` descriptor).
- **Host**, **Port**, and the service/SID value.
- **Username** and **Password**.
- **Environment** — Development, Testing, Staging or Production. This value colours the status bar's leftmost segment: production turns it red, staging amber, testing blue, development green. It is a visual safety rail — glance at the bottom-left before you run something destructive.
- **Test Connection** verifies the settings and reports success or failure without leaving the dialog.

---

8. The icon toolbar

The second header row is the action toolbar. Every button dispatches to the active SQL editor unless noted.

Icon	Tooltip	Action
▶ green	**Execute statement (F8)**	Runs the current statement (or the selection).
□ green	**Execute script (F9)**	Runs the whole editor contents as a script.
□ red	**Stop running query**	Marks the running query as cancelled.
⎇	**Commit**	Commits the SQL editor's open transaction. Lights up green when work is pending.
↩	**Rollback**	Rolls back the SQL editor's open transaction. Lights up amber when work is pending.
☰	**Explain plan (F6)**	Produces the execution plan for the current statement.
⛓	**Visual query builder**	Opens the point-and-click SELECT builder.
¶	**Format SQL**	Reformats the editor contents.
□+	**New query (new tab)**	Adds an empty editor tab.
□	**Open query from file**	Loads a `.sql`, `.pks`, `.pkb` or `.txt` file into a new tab.
□	**Save query to file**	Saves the active tab.
□↓	**Save As…**	Saves the active tab under a new name and location.
□⧉	**Save all tabs**	Writes every open tab to its own file.
⬇ blue	**Export Tables…**	Opens the Export Tables wizard.

Icon	Tooltip	Action
⬆ blue	**Import Tables…**	Opens the Import Tables wizard.
⧉	**New IDE window**	Opens a second, independent IDE window that prompts for its own connection.
🕘	**Query history**	Opens the history dropdown.
☆	**SQL snippets**	Opens the snippet library.
▭ chip	*(display only)*	Shows `SCHEMA@connection` — the schema your statements run against.

At the right end of the row:

Icon	Action
🤖 **Bot**	Show/hide the AI assistant (`Ctrl+Shift+A`). Only appears on the Editor and Debugger modules.
▦ **Table**	Show/hide the bottom dock's Data Grid.

🔒 **Commit and Rollback are greyed out** unless a manual-commit session is active. See §16.6 for what that means.

---

9. The connection chip and container switcher

9.1 The connection chip

Centre-right of the top row. It shows a green dot when connected, the connection alias, and `username@host`. Clicking it opens a dropdown listing the current connection and a **New connection…** entry.

9.2 The container switcher (Oracle Multitenant)

If the connected database is a container database (CDB), a second chip appears showing the current container — `CDB$ROOT` in amber, or a PDB name. Clicking it discovers the pluggable databases visible to your session.

Each PDB row shows:

- A coloured dot — **green** = `READ WRITE`, **amber** = `READ ONLY`, **grey** = `MOUNTED` or in migration.
- The PDB name, a padlock if the PDB is in restricted-session mode, the open mode, and its size in MB.
- A tick against the container you are currently in.

Switching containers. Click any open PDB to move the whole IDE into it — every panel, the object browser, and the editor's session re-target immediately. Because the tool is stateless, "switching" is simply re-pointing the connection at the PDB's default service. `CDB$ROOT` is reachable again as long as DBCraft knows which service originally landed you there (it remembers this automatically when you first connect to the root).

Opening and closing PDBs. If you are connected as `SYS`, each non-current PDB row gains a small control: ▶ to open a mounted PDB, □ to close an open one. Closing asks for confirmation. These controls appear only for `SYS` because Oracle requires a `SYSDBA` session for PDB lifecycle operations.

If no PDBs are listed. Oracle hides `V$PDBS` rows from common users by default. When DBCraft detects this it offers a one-click remedy in the dropdown: **Enable PDB visibility —** `ALTER USER … SET CONTAINER_DATA=ALL`. If your account cannot run that statement, the error is shown in place.

---

10. The docking rail

The narrow strip of icons down the left edge selects the module shown in the main area. It is grouped in two sections separated by a divider.

Development

Icon	Module
`</>`	**SQL Editor**
□	**Object Browser**
□	**PL/SQL Debugger**
⚗	**Unit Test Manager**
□	**Reports**

Administration

Icon	Module
⇄	**Schema Tools**
□	**Session Browser**
⚕	**DBA Dashboards (Health)**
⚯	**ER Diagram Modeler**
⚙	**Automation & Jobs**
□	**Performance Profiler**
□	**Index Advisor**
□	**Load Tester**

Hover any icon for its name; the active module carries a coloured bar on its inner edge.

10.1 Three rail layouts

The rail can be arranged three ways, and your choice is remembered:

Layout	How to get there	Behaviour
Docked (default)	The pin button, from either other layout.	Fixed 52 px strip at the left edge.

Layout	How to get there	Behaviour
Floating	The small pop-out arrow at the top of the rail.	A free-floating panel you can drag anywhere by its grip handle. Click the pin to re-dock.
Collapsed	Drag the small grip on the rail's right edge to the left.	The rail slides off-screen. A thin strip remains at the very edge — hover it and the rail slides back in, taskbar-style. Click the pin inside it to dock permanently.

---

11. The left panel (cluster drawer)

Shown for the **SQL Editor**, **Object Browser**, **Debugger** and **Unit Test Manager** modules. Drag its right border to resize (150–400 px).

It is organised into collapsible clusters:

Connection — the connected username, a **SYSDBA** or **Schema owner** badge, and the target `host:port/service`.

Schema Objects — the full Object Browser tree (chapter 19).

When the Debugger is active, the schema tree is replaced by three debug clusters that mirror the debugger's live state read-only:

- **Breakpoints** — every line with a breakpoint, with the current line marked.
- **Watch Expressions** — each watch and its current value.
- **Call Stack** — the frames, current frame first.

---

12. The bottom dock

The dock slides up from the bottom like the VS Code integrated terminal. Toggle it with `Ctrl+``, the ▦ toolbar button, or **Session** → **Show terminal**. Drag the small centred grip above it to resize (120–600 px).

12.1 Dock tabs

Data Grid is permanent. Four more panels can be added and removed individually:

Tab	Contents
Data Grid	Query results (chapter 17). A badge shows the row count.
Terminal	SQL*Plus-style console (chapter 33).
Sessions	The full Session Browser (chapter 25).
Storage	The full Storage Browser (chapter 27).
Live Metrics	Live CPU / memory / sessions / wait events (§26.2).

Add a panel with the + button in the tab strip; remove one with the × on its own tab. The set of open tabs is remembered between launches. The × at the far right closes the entire dock.

12.2 Terminal groups and splits

The Terminal tab has its own sub-toolbar:

Control	Shortcut	Effect
+ New Terminal	`Ctrl+Shift+``	Adds a new terminal group as a sub-tab.
⫴ Split Terminal	`Ctrl+Shift+5`	Splits the active group into side-by-side consoles.
⧉ New Terminal Window	`Ctrl+Shift+Alt+``	Opens a standalone terminal in its own OS window.
× on a sub-tab	—	Kills that terminal group.

Every group stays mounted while the Terminal tab is open, so scrollback and each console's session survive switching between them.

 Running a query automatically opens the dock and focuses the Data Grid.

---

13. The status bar

A single monospaced strip at the very bottom. Segments appear as they become relevant:

Segment	Shows
Connection context	`username@alias · environment`. Colour-coded: red for production, amber for staging, blue for testing, green for development.
Query state	*Executing* (pulsing amber), *Success* (green) or *Error* (red). Clears after five seconds.
Uncommitted	Amber warning that the SQL editor's session holds uncommitted DML — and therefore row locks.
Rows	Row count of the last result.
Time	Execution time in milliseconds.
Ln / Col	Cursor position in the SQL editor.
CDB / PDB	Which container the session is in (multitenant only). Amber when you are in `CDB$ROOT`.
Server version	The Oracle version banner reported by the server.
Environment	The environment label with a CPU icon.
Version	The DBCraft version.

---

14. The command palette

Press `Ctrl+K` (or click the ⌘ **Ctrl K** button). Type to filter; `↑`/`↓` to move; `Enter` to run; `Esc` to close. Commands are grouped, and the group name is shown at the left of each row.

Group	Commands
run	Execute statement, Format SQL, Commit, Rollback
view	Show Explain Plan, Show Session Monitor, Open Sessions in bottom panel, Toggle AI Copilot, Toggle Terminal
theme	Customize palette — primary & secondary
conn	Switch connection (shows the current one, marked SYSDBA where applicable), New connection…
module	Open any of: SQL Editor, Object Browser, PL/SQL Debugger, Unit Test Manager, Reports, Schema Tools, Session Browser, Health Check, ER Diagram Modeler, Automation & Jobs, Performance Profiler, Load Tester; plus Export Tables… and Import Tables…
help	Open user guide

---

15. Extra windows

DBCraft can open three kinds of additional window:

Window	How	Behaviour
New IDE window	⧉ toolbar button	A complete second IDE instance. It prompts for its own connection and gets its **own editor transaction** — it will not share or roll back the main window's uncommitted work.
Pop-out editor	⧉ button at the right of the SQL editor's tab strip	Opens the active tab's SQL in a standalone editor window (1000×800). It also gets its own transaction.
Pop-out terminal	Terminal → **New Terminal Window**	A standalone SQL console window (900×600).

Part III — Development modules

16. SQL Editor

The default module, and the one you will use most.

16.1 Tabs

The tab strip runs across the top of the editor.

Action	How
New tab	**+** at the right of the strip, the □+ toolbar button, or the command palette. New tabs are named `Query 1`, `Query 2`, … reusing the lowest free number.
Rename a tab	**Double-click** the tab name, type, press `Enter` (or `Esc` to cancel).
Close a tab	The **×** on the tab. The last remaining tab cannot be closed.
Pop out	The ⧉ button at the far right of the strip.

An amber dot on a tab means unsaved changes.

Tabs survive restarts. Every tab's title and contents are saved locally as you type, along with which tab was active. Closing and reopening the application brings your work back exactly as it was.

Besides ordinary SQL tabs, the editor also hosts three special tab types, all created from the Object Browser:

- **Object tabs** — a read-only viewer with Columns / Data / DDL sub-tabs (§19.5).
- **Program tabs** ("*(Edit)*") — an object's source loaded for editing and recompiling.
- **Diff tabs** ("*Diff: A vs B*") — a side-by-side read-only comparison of two objects' DDL.

16.2 Writing SQL

The editor is a full code editor with:

- **Syntax highlighting** for SQL and PL/SQL.
- **A minimap** down the right edge.
- **Word wrap**, line numbers, and bracket matching.
- **Error markers.** When a statement fails with a line number, that line is underlined in red and hovering it shows the compilation error.
- **Colourblind-safe themes.** If you select a colour-vision mode in Settings → Accessibility, the editor's syntax palette switches to a matched scheme (red-green safe, tritan safe, or monochrome with bold keywords) in both light and dark.
- **Adjustable font size**, set in Settings → Appearance → Editor font size (10–24 px).

Autocomplete triggers on space and on `.`:

- After a dot (`HR.`) it lists the tables and views in that schema.
- Otherwise it offers SQL keywords plus the tables in your default schema.

Object lists are cached after the first fetch, so completion stays responsive.

16.3 Executing statements

Method	Effect
`F5`	Execute.
`Ctrl+Enter`	Execute.
▶ toolbar button	Execute.
Command palette → **Execute statement**	Execute.

What gets executed:

- If you have **selected text**, only the selection runs. This is the fastest way to run one statement out of a long script.
- Otherwise the **entire tab contents** run. The server splits the text into statements and executes them in order, handling PL/SQL blocks correctly.

Multiple result sets. A script that contains several `SELECT`s produces several grids — the results panel shows one tab per result set (**Result 1**, **Result 2**, …) plus a **Script Output** tab summarising statements that produced no grid.

While a query runs the toolbar collapses to a single **Stop** button, the status bar shows a pulsing *Executing*, and the tab bar stays usable.

16.4 Explain plan

Press the ☰ toolbar button or use the palette's **Show Explain Plan**. The plan is returned as a text plan table and displayed in the results grid. A trailing semicolon is stripped automatically for plain SQL (but preserved for PL/SQL, where it is part of the syntax).

16.5 Formatting

`Shift+Alt+F`, or the ¶ toolbar button, reformats the entire tab using PL/SQL formatting rules — consistent keyword casing, indentation and line breaks.

16.6 Transactions — manual commit mode

This is the single most important behavioural difference between DBCraft's SQL editor and its other panels. **Read this section.**

The SQL editor runs in **manual-commit mode**. It holds one long-lived Oracle session whose transaction spans your statements. Nothing you write with `INSERT`, `UPDATE`, `DELETE` or `MERGE` is durable until you commit.

Concept	Detail

Concept	Detail
Pending work	When the session holds uncommitted DML, the ⎇ Commit icon turns green, the ↩ Rollback icon turns amber, and the status bar shows an amber **Uncommitted** segment.
Commit	⎇ toolbar button, or palette → **Commit**. A notification confirms the result.
Rollback	↩ toolbar button, or palette → **Rollback**.
DDL commits implicitly	Oracle commits automatically on DDL. If you run a `CREATE`, `ALTER` or `DROP`, any pending DML is committed with it. The pending indicator is read from the database itself, so it always reflects reality.
Failed statements do not discard earlier work	If statement 5 fails, statements 1–4 remain pending.
Switching modules is safe	The transaction lives on the local service, not in the panel. Clicking away to Reports and back does not roll anything back.
Closing the window rolls back	Closing the application tells the service to abandon the session, which rolls it back and releases the row locks immediately rather than waiting for a timeout.
Idle sessions are reaped	A session left idle for 30 minutes is closed (and therefore rolled back).

 Other panels do not see your uncommitted rows. Reports, the Object Viewer's Data tab, the Terminal, the Test Manager and every DBA panel use separate auto-committed connections. If a report does not show a row you just inserted, you have not committed it.

 Uncommitted DML holds row locks. Other users' sessions can block behind yours. If the Session Browser shows blocking chains pointing at your session, commit or roll back.

16.7 Query history

The toolbar button opens the history dropdown: up to 50 recent statements, newest first, each with the time it ran and how many rows it returned.

- **Click an entry** to open it in a new tab.
- **Click the star** to mark a favourite — favourites sort to the top.
- **Search** the history with the box at the top of the dropdown.

Keyboard history recall. With the cursor in the editor:

- `Ctrl+↑` or `Alt+↑` — step back through history into the current tab.
- `Ctrl+↓` or `Alt+↓` — step forward; stepping past the newest restores what you had typed.

If the tab is empty, plain `↑` and `↓` do the same thing.

16.8 Snippet library

The ⭑ toolbar button opens a three-group library. Choosing a snippet opens it in a new tab named after the template.

Basic Queries (DML) — SELECT, INSERT (with sequence `NEXTVAL`), UPDATE, DELETE, INNER/LEFT JOIN templates.

Structure (DDL) — CREATE TABLE (with an identity column, unique constraint, default and timestamp), CREATE PROCEDURE (with `IN`/`OUT` parameters), CREATE FUNCTION (with an exception handler), CREATE TRIGGER (row-level `BEFORE INSERT`).

DBA & Metadata — Show DB version (`v$version`), List tables & sizes (`user_segments`), Find invalid objects (`user_objects`), Inspect table columns (`user_tab_columns`).

16.9 Visual Query Builder

The ⛓ toolbar button opens a point-and-click SELECT builder.

1. The left column lists the tables in your schema. Hover a table and click **+** to add it to the canvas.
2. Each added table appears as a card listing its columns. Click columns to tick them.
3. The **Generated SQL Preview** at the bottom updates live.
4. **Insert SQL into Editor** opens the generated statement in a new tab.

 The builder produces the `SELECT` list and `FROM` clause. Add joins and predicates in the editor afterwards.

16.10 File operations

Action	Toolbar	Behaviour
Open		Native file dialog; accepts `.sql`, `.pks`, `.pkb`, `.txt`. The file opens in a new tab named after the file.
Save		Saves the active tab to a `.sql` file named after the tab, and clears its modified dot.
Save As…	↓	Native save dialog; you choose the name and folder. The tab is renamed to match.
Save all tabs		Writes every open tab to its own `.sql` file.

---

17. Results grid and data editing

Results appear in the bottom dock's **Data Grid** tab (or in a split panel beneath the editor in the standalone pop-out window).

17.1 The results toolbar

Element	Purpose
Results: n rows · tms	Row count and elapsed time.
Edit Mode switch	Enables in-grid cell editing (§17.5).

Element	Purpose
Filters switch	Reveals a search box and per-column filter boxes (§17.3).
Pagination	`‹ page / total ›`, shown when there is more than one page.
Graph menu	Charts the result set (chapter 18).
Export menu	CSV, JSON, HTML, Excel, `.fmt` (§17.6).
Generate Update Script	Appears once you have edited cells (§17.5).

Beneath the grid a status line reads, SQL*Plus style: `[14:32:07] 341 rows selected in 0.084 seconds`.

17.2 Result tabs and Script Output

- One tab per result set when a script produced several.
- A **Script Output** tab appears when some statements produced no grid (DDL, DML, PL/SQL), when nothing produced a grid, or on error. It shows the server message, the statement count and the elapsed time. On error the tab is titled **Error** and shown in red.

17.3 Searching and filtering

Turn on the **Filters** switch to get:

- A **Filter all columns…** box that matches text in any column.
- A small filter box under **every column header** for per-column matching.

Filters are case-insensitive substring matches, combine with AND, and reset the page to 1. Turning the switch off clears them.

17.4 Paging

The grid pages at **100 rows** per page. Use the `‹` and `›` buttons. Filtering applies to the whole result set, not just the current page.

17.5 Editing cells and generating an UPDATE script

Turn on **Edit Mode**, then:

- **Double-click** a cell to edit it in place. `Enter` commits the edit to the grid, `Esc` cancels.
- **Right-click** a cell to open the **Large Data Editor** — a full code editor for long text, with automatic JSON and XML detection, a **Format JSON** button, and syntax highlighting. Useful for CLOB and JSON columns.
- Edited cells are highlighted.

 Editing the grid does not change the database. DBCraft never writes silently. Instead, click **Generate Update Script** — it builds a reviewable `UPDATE` statement for every changed row and opens it in a new editor tab.

The generated script uses `[TABLE_NAME]` as a placeholder (the grid does not always know which table a column came from) and builds the `WHERE` clause from every original column value, with `IS NULL` for nulls.

Review it, set the table name, and check the `WHERE` clause identifies exactly the rows you intend before running it.

17.6 Exporting results

Format	Notes
CSV	Proper RFC-style quoting for values containing commas, quotes or newlines.
JSON	An array of objects keyed by column name, pretty-printed.
HTML	A simple bordered table.
Excel	A real `.xlsx` file, generated by re-running the query on the server so the full result set is exported — not just the rows loaded into the grid.
.fmt	A SQL*Loader / BCP format file describing the column layout.

---

18. Graphs

Any result set with at least one numeric column can be charted. Open the **Graph** menu on the results toolbar:

- **Wizard…** — opens the graph dialog with the full wizard.
- **Bar / Line / Area / Pie** — opens straight into that chart type.
- **From template** — applies a saved configuration.

18.1 The graph window

A large chart on the left; a wizard sidebar on the right.

Sidebar section	Controls
Graph Wizard	Four chart-type buttons: Bar, Line, Area, Pie.
X-Axis (category)	Which column supplies the categories.
Y-Axis series (numeric)	Tick one or more numeric columns to plot. For pie charts this becomes a single **Value (numeric)** selector. Numeric columns are detected automatically by sampling the data.
Options	**Data labels** (values printed on the chart), **Legend**, **Auto Refresh**, and a manual **Refresh** button.
Templates	Name the current configuration and save it; saved templates are listed and can be applied or deleted.
Export	PNG, JPEG (both include the legend) or SVG (plot only).

18.2 Limits and behaviour

- Cartesian charts plot at most **500 rows**; the footer tells you when it is showing the first 500 of a larger set.
- Pie charts show the **top 8 categories** and fold the remainder into an **Other** slice.
- With **Auto Refresh** off, the chart holds a snapshot and the footer says so; **Refresh** re-runs the query.
- The palette is colour-vision-validated, with separate light and dark variants, and follows the app theme.

---

19. Object Browser and Object Viewer

The Object Browser is available as its own module and as the left panel of the Editor, Debugger and Test Manager.

19.1 Navigating

- **Schema** dropdown — every schema visible to your account. Defaults to your own.
- **Refresh** — re-reads the object list.
- **Search objects…** — filters the tree by name as you type; empty type groups disappear.

The tree is `Schema → Object type → Object`. Each type node shows a count badge. Objects that are `INVALID` carry a red **INVALID** badge — invaluable after a bulk recompile.

Supported types, with distinct icons: **Table**, **View**, **Procedure**, **Function**, **Package**, **Trigger**, **Sequence**, **Index**, **Synonym**, **Type**, **Job**.

Selecting an object shows a **SELECTED** footer with its fully-qualified name and type.

19.2 The object context menu

Right-click any object:

Item	Available on	Effect
Edit & Compile	Procedure, Function, Package, Trigger	Loads the source into an editable tab. Change it and press Execute to recompile. Compilation errors are marked on the failing line.
View Details	Everything else	Opens the Object Viewer (§19.5).
Query Data	Everything else	Opens a new tab containing `SELECT * FROM schema.object;`.
Generate DDL	All	Opens the Object Viewer on its DDL tab.
Compare…	All	Opens the compare dialog (§19.3).

Item	Available on	Effect
Generate Script ▸	Table, View	Generates a full **SELECT**, **INSERT**, **UPDATE** or **DELETE** statement listing every column, with `/* value */` placeholders where you must fill in.
Import Data…	Table	Opens the import wizard (§19.4).
Drop Object	All	Opens a drop tab for the object.

19.3 Comparing two objects

Compare… opens a dialog showing the source object and asking for a target:

1. Pick the **Target Schema**.
2. Pick the **Target Object** — the list is restricted to objects of the *same type*, so you cannot compare a table with a package.
3. **Compare DDL**.

A **Diff** tab opens in the editor showing the two definitions side by side with additions and deletions highlighted. The diff view is read-only.

19.4 Importing data into a table

Import Data… on a table opens a small wizard:

1. **Select File** — CSV, Excel (`.xlsx`/`.xls`) or `.fmt`.
2. Click **Start Import**.

 The file's column headers must exactly match the target table's column names. The wizard reports the number of rows inserted, or the database error if the import fails.

19.5 The Object Viewer

Opened by **View Details** or **Generate DDL**. Three tabs, each loaded on demand and cached until you press Refresh:

Tab	Contents
Columns	Ordinal, column name, data type, length, and nullable flag.
Data	The first **100 rows** of the object. Row numbers are frozen at the left; long values are truncated with the full value in the tooltip.
DDL	The complete `CREATE` statement from the data dictionary, in a read-only syntax-highlighted editor with a minimap.

20. PL/SQL Debugger

A true source-level debugger. It attaches to your procedure through Oracle's `DBMS_DEBUG` package and gives you breakpoints, stepping, live variable values, watch expressions and a call stack.

20.1 Prerequisites

Requirement	Why
An **Oracle** connection	The debugger uses `DBMS_DEBUG`; it is Oracle-only.
`DEBUG CONNECT SESSION` privilege	Required to attach a debug session.
The program compiled with debug information	`ALTER PROCEDURE my_proc COMPILE DEBUG;` (or `PLSQL_DEBUG=TRUE`). Without it Oracle reports no line information and stepping will not stop where you expect.
`DEBUG ANY PROCEDURE`, or ownership	To debug code in another schema.

20.2 Starting a session

The toolbar in the idle state has three controls:

1. **Procedure Name** — type the program's name. Accepts `NAME` or `SCHEMA.NAME`. As you type, DBCraft loads the source automatically (after a short pause) so you can set breakpoints *before* you start. It tries PROCEDURE, then FUNCTION, then PACKAGE. The ⟳ button next to the field reloads on demand.
2. **Execution SQL Block** — the anonymous block that calls your program, e.g.

```
BEGIN my_proc(103, 10); END;
```

1. **Start Debug** — attaches the session and runs to the first breakpoint (or the first executable line).

20.3 Breakpoints

Click in the **gutter** — the narrow column to the left of the line numbers — to toggle a breakpoint. A red dot marks it.

Breakpoints can be added and removed **while execution is paused**; DBCraft registers the change with the running session immediately and logs the fact to the console.

20.4 Stepping

While paused, the toolbar offers:

Button	Meaning
Resume	Continue to the next breakpoint or to the end.
Step Over	Execute the current line; do not descend into calls.
Step Into	Descend into the called program.
Step Out	Run to the end of the current program and return to the caller.
Stop	End the debug session.

The current line is highlighted amber with a ► marker in the gutter. If **Step Into** takes you into a program whose source is not the one on screen, the highlight is suppressed and the status badge tells you which program you are actually in — `Paused at L42 in INNER_PROC`.

The status badge on the right shows **Paused at L*n*** (amber, pulsing), **Running…** (blue) or **Completed** (green).

20.5 The inspection panel

Three tabs down the right side:

Variables — every in-scope variable and its current value, refreshed at each stop. Additionally, **any variable name appearing in the source is highlighted inline**; hover it to see its current value in a tooltip. This is often faster than reading the list.

Watch — type a variable name or expression and press `Enter` (or the **+**). Watches are evaluated at every step. Values that are `NULL` or unavailable are shown greyed. Hover a watch to reveal its delete button.

Call Stack — the frames, current frame first with a ► marker, each showing the program name, type badge, line and depth.

20.6 The Debug Console

The strip along the bottom logs every event with a timestamp: session attach, step results, dynamic breakpoint changes, `DBMS_OUTPUT` from your program, and errors. Colour-coded — green for success, red for errors, amber for program output, grey for informational.

20.7 Exporting a debug log

Export on the toolbar writes a plain-text file `debug_<PROC>_<timestamp>.txt` containing the console log, the final variable values, all watch expressions and values, the call stack, and the execution block. Useful for attaching to a defect report.

20.8 The offline simulator

If no live database connection is available, **Start Debug** falls back to a built-in simulator that walks through a sample procedure. Every console line is prefixed `[SIMULATOR]`. This exists for demonstration and for learning the interface; it does not touch a database.

20.9 Launching the debugger from elsewhere

A failed test in the Unit Test Manager offers a **Launch in Debugger** button that switches to this module with the procedure name and execution block already filled in.

---

21. Unit Test Manager

Author, organise and run repeatable PL/SQL tests. Suites and their results are stored locally by DBCraft — nothing is installed in your database.

21.1 Layout

- **Left column** — your test suites for this connection. Each shows a pass count (green), a fail count (red), and the last-run time. A dot appears next to a suite with unsaved edits.
- **Right pane** — the suite editor, with a toolbar and three tabs: **Definition**, **Notes**, **Run**.

21.2 Creating a suite

+ in the left header, or **New Test Suite** on the toolbar. Give the suite a **Suite Name** and an optional **Description**, then **Create Suite**.

21.3 The suite toolbar

Button	Action
+	New Test Suite.
	Import Test Suite — reads a `.dbctest.json` file.
	Save — enabled only when there are unsaved changes.
↓	**Export Test Suite** — writes the suite to a `.dbctest.json` file for version control or for sharing with a colleague.
▶	**Run** — runs the suite.

While running, a progress bar and percentage appear next to the Run button.

21.4 Defining test scripts (Definition tab)

The upper list shows the scripts in execution order, numbered, each with a status icon and badges for *new session*, *disabled*, and any performance limit. The buttons down the right side:

Button	Action
+	Add a new test script.
×	Remove the selected script.
↑ / ↓	Move the selected script up or down — this is the execution order.
	Load script from file — replaces the selected script's SQL from a `.sql`, `.pls`, `.pks`, `.pkb` or `.txt` file.

The **Definition** fieldset below edits the selected script:

Field	Meaning
Name	The test's name, shown in results.
Description	Free text.
Object Name	The program under test, e.g. `HIRE_EMP`. Used by **Launch in Debugger** when the test fails.
Test script	The PL/SQL to execute. Defaults to a template using the `:out_result` bind.

Field	Meaning
Expected output	What `:out_result` must equal for the test to pass. Use `*` to accept any output.
New session	Run this script on its own connection instead of the suite's shared session. Use it to test something that depends on a fresh session state.
Enabled	Unticked scripts are **skipped** by a run (reported as *skipped*, not *failed*).
Performance	A time limit in seconds. If the script takes longer, it fails. Blank means no limit.

How pass/fail is decided:

1. If the script assigns the `:out_result` bind variable, its value is compared with **Expected output**. `*` accepts anything.
2. If the script binds nothing, it **passes if it runs without raising an error**.
3. Any Oracle error fails the test, and the error message is shown in the results.
4. Exceeding the **Performance** limit fails the test.

Example:

```
BEGIN
  :out_result := hire_emp(p_name => 'TEST', p_dept => 20);
END;
```

with **Expected output** `SUCCESS`.

21.5 Notes tab

Edit the **Suite Name**, **Description** and a free-form **Notes** field — the right place for setup requirements, known issues and ownership.

21.6 Running a suite

Press ▶. What happens:

1. Any unsaved edits are saved first — the run always executes what is on disk.
2. One shared database session is opened for the suite.
3. Each script runs in order. Scripts marked **New session** get their own connection.
4. Results stream into the **Run** tab as each script finishes.
5. The shared session is closed at the end, and the suite counters are refreshed.

21.7 Reading results (Run tab)

A summary bar shows **Passed**, **Failed**, **Skipped** and a **Pass Rate** percentage (green at 100 %, amber at 80 % or above, red below).

Each script gets a card with its status icon, name, description, duration in milliseconds, and a status badge. A failure expands to show:

- The **error message**, in a red monospaced block.
- The **Actual Output**, when the script produced one.
- A **Launch in Debugger** button (when an **Object Name** is set) that opens the debugger pre-loaded with that procedure and execution block.

---

22. Reports

A catalogue of ready-made and custom queries, with parameters, exports and printable document templates.

22.1 The report list

The left column groups reports by category; click a category header to collapse or expand it (collapsed state is remembered). Built-in reports carry a **System** badge and cannot be edited or deleted — but they can be **duplicated** into editable copies.

The built-in catalogue:

Category	Reports
DBA	Initialization Parameters · NLS Database Parameters · Role Privileges · Roles · Rollback Segments · Server Components · System Privileges · Tablespaces · Total Free Space · Users
Objects	All Tables · All Views · All Indexes · All Sequences · All Synonyms · All Triggers · Constraints · Invalid Objects · Index Status
PL SQL	Procedures · Functions · Packages · Package Bodies · Types · Invalid PL/SQL Objects · Compilation Errors
User *(no DBA privileges needed)*	My Tables · My Objects · My Indexes · My Sequences · My Role Privileges · My System Privileges · My Tablespace Quotas · My Sessions
Storage	Tablespace Usage
Performance	Top SQL by CPU
Security	Lock Waits
Scheduling	Scheduler Jobs

 The **User** category is the one to start with if your account has no DBA role — those reports read `USER_*` views and always work.

22.2 Running a report

Select a report and press **Run Report**. Its SQL is shown read-only above the results; the results table below shows the row count and elapsed time. Errors appear in a red monospaced block instead.

22.3 Parameters (bind variables)

A report whose SQL contains `:name` bind variables gets a **Parameters** bar above the SQL, with an input box per variable. Fill them in and run.

DBCraft detects binds intelligently: it ignores `:=` assignments, string literals and comments, so PL/SQL in a report body will not produce spurious parameters. Values that look numeric are substituted as numbers; everything else is quoted and escaped.

22.4 Exporting

The **Export** menu (enabled once a report has produced rows) offers **CSV**, **Excel**, **JSON** and **HTML**. Excel is generated server-side so the complete result set is exported.

22.5 Generating a formatted document

Generate renders your results into an HTML document you can save or print (and therefore save as PDF).

The dialog first offers **Branding**:

Control	Effect
Theme colour	A colour picker plus a hex box. Invalid hex values are flagged.
Company logo	Upload an image (under 1 MB). A preview appears; the × removes it.
Logo position	Top left or top right.

Branding applies to the two built-in templates, **Classic Corporate** (accent colour, company logo, striped rows, print-ready) and **Minimal Document** (serif document with accent, logo, and the report SQL printed).

Each template row then offers:

- **HTML** — downloads the rendered document.
- **Print** — opens it in a new window and triggers the print dialog. Choose "Save as PDF" there for a PDF.

22.6 Managing templates

The button at the top of the report list opens **Report Templates**. Built-in templates can be **previewed** and **copied**; your own can be edited and deleted.

A template is plain HTML with placeholders:

Placeholder	Substituted with
`{{title}}`	The report name
`{{category}}`	The report category
`{{date}}`	Current date and time
`{{user}}`	The connected username
`{{connection}}`	`host:port/service`

Placeholder	Substituted with
`{{sql}}`	The report's SQL
`{{rowCount}}`	Number of rows
`{{table}}`	The full results table as HTML
`{{themeColor}}`	The branding colour
`{{logo}}`	The branding logo block

Preview renders the template with your current results, or with sample data if you have not run the report yet. All substituted values except `{{table}}` and `{{logo}}` are HTML-escaped, so data cannot break your layout.

22.7 Creating a custom report

+ at the top of the report list. Provide a **Report Name**, a **Category** (type a new one to create it) and the **SQL Query**. Use `:name` binds to prompt for values at run time. **Save Report**.

Existing custom reports can be **Edited**, **Duplicated** or deleted (the bin icon appears on hover). Duplicating a built-in report is the standard way to tweak one: the copy lands in the **Custom** category and is fully editable.

Part IV — DBA modules

23. Schema Tools

Four tabs: **Schema Compare**, **DDL Export**, **Data Import/Export** and **Version History**.

23.1 Schema Compare

1. Choose a **Source Schema** and a **Target Schema** (the target list excludes the source).
2. **Compare Schemas**.

Results are summarised in three chips and listed one row per difference:

Difference type	Meaning
Modified (amber)	The object exists in both schemas but its definition differs.
Missing in Target (red)	The object exists in the source only.
Missing in Source (blue)	The object exists in the target only.

Every difference starts selected. Click a row to include or exclude it — deselected rows dim.

Generate Sync Script produces the DDL that would bring the target in line with the source for the selected differences only. The script is shown in a scrollable block with a **Download** button.

 The sync script is not executed. Review it, and run it in the SQL editor when you are satisfied. Treat `DROP` statements in the script with particular care.

23.2 DDL Export

Choose a **Schema**, then toggle which **Object Types** to include — TABLE, VIEW, PROCEDURE, FUNCTION, PACKAGE, TRIGGER, SEQUENCE (all on by default; click a badge to toggle). **Export Selected DDL** downloads `<SCHEMA>_DDL_Export.sql`.

23.3 Data Import / Export

Two side-by-side panels.

Export Data — pick a **Schema**, a **Table**, and a **Format**:

Format	Output
CSV Format	Comma-separated values.
JSON Format	An array of row objects.
INSERT Statements	Ready-to-run `INSERT` statements.

Import Data — pick a **Target Schema** and **Target Table**, choose a file (`.csv`, `.xlsx`, `.xls`, `.json`), and press **Import**. The format is detected from the file extension. Column headers must match the target table's column names.

23.4 Version History tab

Hosts the object version history module — see chapter 24.

---

24. Version History

Object-level revision history, like a document's version history. Available as the fourth tab of Schema Tools.

24.1 How versions are captured

- **Automatically.** Every `CREATE OR REPLACE` you execute in the SQL editor snapshots that object's DDL as a new version. You do not have to do anything.
- **Manually.** Select an object and press **Snapshot** to save its current state. If nothing has changed since the newest version, DBCraft says so and does not create a duplicate.
- **Automatically before a restore.** Restoring a version first snapshots the current state, so every restore is itself reversible.

24.2 The three panes

Left — Objects. Every object with history, searchable, showing owner, type badge and version count.

Centre — DDL viewer. The selected version's DDL. When a previous version exists, a **Changes vs *vn*** toggle switches between the plain DDL and a line diff: additions in green with `+`, deletions in red with `-`, unchanged lines dimmed. Very large sources skip the diff and show the plain DDL.

Above the DDL: **Copy** (copies the DDL to the clipboard) and **Restore this version**.

Right — Versions timeline. Newest first, each entry showing the version number, a source badge, and the timestamp:

Badge	Origin
Compiled	Captured automatically from a `CREATE OR REPLACE`.
Snapshot	You pressed Snapshot.
Restored	Produced by restoring an earlier version.
Before restore	The automatic safety snapshot taken immediately before a restore.

Hover a version for two icons: **Name this version** (attach a label such as "*Before Q3 release*") and **Delete version**.

24.3 Restoring

Restore this version asks for confirmation, then recompiles the object from that version's DDL against the live database. The confirmation dialog reminds you that the current state is snapshotted first.

 Restoring runs DDL against your database. On Oracle, DDL commits implicitly — see §16.6.

25. Session Browser

Monitor and manage the sessions connected to your database. Available as a module and as a bottom-dock panel.

25.1 Summary tiles

Tile	Counts
Total Sessions	All sessions in the list.
Active	Status `ACTIVE` or `RUNNING`.
Waiting	Sessions in a real wait — idle `SQL*Net` waits are excluded, so this number means something.
Blocked	Sessions waiting on a lock held by another session.
Blockers	Sessions holding a lock that someone is waiting on.

25.2 Auto-refresh

The **Refresh** dropdown offers **Off** (default), **Every 2s**, **5s**, **10s**, **30s** and **Every 1m**. Your choice is remembered. The manual **Refresh** button works at any time, and the header shows when the list was last updated.

25.3 All Sessions view

A full table: **SID · Username · Status · OS User · Program · Machine · Logon Time · Wait Event · Blocked By · CPU (s) · I/O Reads · PGA (MB) · SQL Text · Actions**.

- Blocked sessions are tinted red; real waits show an amber warning triangle.
- **DBCraft's own monitoring session** is tinted blue and tagged **this tool** so you never mistake it for a user session.
- **Search sessions…** matches username, program, machine, SQL text or SID.
- **Hide Inactive Sessions** and **Hide this tool's session** are two independent checkboxes.

Per-row actions: □ **view details** and ⊗ **terminate**.

25.4 Blocking Tree view

Switch to **Blocking Tree** (the button carries a badge with the blocked count) to see lock contention as a hierarchy:

- **Roots** are the ultimate blockers, marked with a red shield icon.
- **Children** are the sessions they block, marked with an amber ban icon, nested to any depth — `A blocks B blocks C`.
- Each blocked node shows how long it has been waiting and on which event; each blocker shows how many sessions it is blocking.
- Sessions whose blocker is not in the list are still shown, so nothing is hidden.

When nothing is blocked, the panel says **No blocking chains — all clear.**

25.5 Session details

The button opens a dialog listing every attribute the database reports for that session, plus the full **Current SQL** in a scrollable block.

25.6 Terminating a session

 This ends someone's work. The ⊗ button opens a confirmation dialog naming the session (`SID,serial`) and the user, and stating plainly that the session's current transaction will be rolled back and that a DBA account (`ALTER SYSTEM`) is required.

Choose a **Method**:

Method	Statement	Effect
Kill Session	`ALTER SYSTEM KILL SESSION`	Marks the session `KILLED` and rolls it back. The session may linger in `KILLED` state until the client notices.
Disconnect	`ALTER SYSTEM DISCONNECT SESSION`	Also kills the underlying server process.

The **Immediate** checkbox (on by default) tells Oracle not to wait for the session's current call to finish.

If the session you are about to kill is blocking others, the dialog warns you and notes that terminating it should release the waiters.

After a successful termination the row disappears and the list refreshes shortly after to show the true state.

---

26. DBA Dashboards (Health)

Five tabs: **Live Metrics**, **Health Scan**, **Storage**, **ASM** and **Cluster (RAC)**. The last two appear for Oracle connections.

26.1 Live Metrics

Polls the database every **5 seconds** while the tab is open. **Pause** / **Resume** stops and restarts polling; a pulsing **Live Polling Active** badge shows the state.

Panel	Shows
CPU Utilization (%)	A rolling line chart of the last 12 samples.
Memory Usage (MB)	A rolling line chart of the last 12 samples.
Active User Sessions	A single large number.
Top Wait Events	A donut chart of the current wait profile.

If a chart stays empty, the panel tells you why: the metric requires privileges your account does not hold. The other panels keep working.

 Live Metrics is also available as a bottom-dock panel, so you can watch it while working in another module.

26.2 Health Scan

Run Health Check performs a static analysis and produces a scored report.

Summary row:

- A **Health Score** out of 100 (the proportion of checks that passed), badged **Healthy** (≥80), **Warning** (≥60) or **Critical**.
- A pass/warn/fail donut.
- **Tablespace Usage** bars, with anything above 80 % highlighted amber.

The database version is printed under the heading.

Checks performed:

Category	Check	What it looks at
Performance	**Buffer Cache Hit Ratio**	Cache efficiency from `V$SYSSTAT`.
Performance	**Active Sessions**	Current session load from `V$SESSION`.
Objects	**Invalid Objects**	Count of `INVALID` objects in `DBA_OBJECTS`.
Storage	**Tablespace Usage**	Per-tablespace fullness from `DBA_DATA_FILES` / `DBA_FREE_SPACE`. One check per tablespace.
Security	**DBA Role Grants**	Who holds the DBA role, from `DBA_ROLE_PRIVS`.
Configuration	**Archive Log Mode**	Whether the database is in ARCHIVELOG mode, from `V$DATABASE`.

Results are grouped by category with pass/warn/fail counts per group. Each result shows the check name, a message, the measured value, and — where relevant — an italic **recommendation** (for example, "*Add datafiles to USERS tablespace or enable autoextend.*").

 Missing privileges do not break the scan. A check your account cannot evaluate is reported as a *warning* naming the exact grant needed — e.g. "*Grant SELECT on V$SYSSTAT (or connect as a privileged user) to evaluate this metric.*"

Export CSV writes the whole report (category, check name, status, value, message, recommendation) to `oracle_health_report.csv`.

A notification summarises the outcome when the scan finishes.

26.3 Storage tab

Hosts the Storage Browser — see chapter 27.

26.4 ASM tab

Reads Oracle's ASM views. If the database stores its files on a regular file system, the tab says **No ASM storage detected** and explains why (including the exact error, if `V$ASM_DISKGROUP` could not be read).

When ASM is in use:

- **Disk group cards** — name, state badge (Mounted / Connected / Dismounted / Broken), redundancy, a usage bar with percentage, total and free space, allocation-unit size, and a red count of any offline disks.
- **Rebalance Operations In Progress** — shown only when a rebalance is running: group, operation, state, power, progress and estimated minutes.
- **ASM Disks** — group, disk name, path, failgroup, state/mode, size and free space.
- **Connected Database Clients** — instance, database, status and software version.

26.5 Cluster (RAC) tab

A badge at the top reads **RAC — *n* instances** or **Single Instance**.

- **Instance cards** — one per instance: instance name and number, status badge (Open / Mounted / Started), host, version, startup time and uptime in days, database status and active state, whether logins are allowed, and the session counts (user, active, total) for that instance.
- **Database Services** — instance, service name and network name.

On a single-instance database the panel explains that `CLUSTER_DATABASE = FALSE` and shows the instance details anyway. Views the account cannot read are reported as an amber note rather than an empty panel.

---

27. Storage Browser

Tablespace, segment and datafile analysis. Available in the Health module's **Storage** tab and as a bottom-dock panel.

27.1 Summary tiles

Tablespaces · Allocated · Used · Datafiles · Critical (≥90 %). The critical tile turns red when any tablespace is at or above 90 % full.

27.2 Tablespace list

Each tablespace shows its name, a contents badge (**PERMANENT** blue, **TEMPORARY** cyan, **UNDO** amber), a usage bar and percentage, the used-of-max figures, and the datafile count. Anything at 90 % or above carries a red warning triangle.

Colour thresholds throughout the panel: **green** below 75 %, **amber** 75–90 %, **red** at 90 % and above.

27.3 Tablespace detail

Selecting a tablespace opens a detail pane showing extent management, segment space management, status, and four figures — **Allocated**, **Used**, **Max (autoextend)** and **Autoextend files** (how many of the datafiles can grow) — plus a large usage bar.

Two tabs beneath:

Segments — owner, segment name with a type icon, type, size, a relative-size bar comparing it with the largest segment in the tablespace, and extent count. The **100 largest segments** are shown; a footer says so when the cap is reached.

Datafiles — file name, type badge, current size, max size, whether it is autoextensible (tick or dash) and its status with a colour dot.

---

28. Performance Profiler and SQL Optimizer

28.1 Profiling a statement

Paste or type SQL into **SQL to Profile** (a sample multi-table join is provided) and press **Run Profiler**.

The result header shows the **Total Cost**, and — when the statement was actually executed — the elapsed time in milliseconds and the row count.

The plan table lists every step: **Operation · Object · Rows · Cost · % Cost**. Each row carries a proportional bar, and the rows are tinted by weight — **red above 50 %** of total cost, **amber above 20 %**. This makes the expensive step obvious at a glance.

Alongside, an **Execution Plan — Cost by Operation** bar chart plots the eight most expensive steps, coloured by the same thresholds.

28.2 AI SQL optimisation

Get AI Suggestion sends the statement *and its execution plan* to the AI provider you configured (chapter 34) and returns a rewritten query in a code block followed by a bullet list of what changed and why.

If no provider is configured, the panel says so and points you at **Settings → AI Configuration**.

 The rewritten SQL is a suggestion. Verify it returns the same rows before using it.

---

29. Index Advisor

Tests candidate indexes and tells you whether the optimizer would actually use them — without permanently building anything.

29.1 Loading the SQL to analyse

Two ways:

By object name. Type a name into the box (accepts `NAME` or `SCHEMA.NAME`). DBCraft resolves the real owner and type first — so it reports the truth rather than guessing — and then:

Object type	What is loaded
View	The view's underlying `SELECT`, extracted from its DDL.

Object type	What is loaded
Table or **Materialized View**	A *starter query* is generated for you: a projection of the first columns and a `WHERE` clause on a selective-looking column, with `<value>` as a placeholder. A note explains that a table has no query of its own — you must set the predicate to describe the access pattern you want indexed.
Anything else	The full source, with a note asking you to trim it to the single `SELECT` you want analysed.

The ⟳ button reloads; the × clears everything.

By pasting. Paste any query into the text area.

29.2 Estimate mode vs Measure mode

Mode	How it works	Cost
Estimate (default)	Creates a hypothetical `NOSEGMENT` index that the optimizer costs but never stores, then drops it.	Fast and light. No storage, no writes.
Measure	Creates a real `INVISIBLE` index, times the query with the index visible and invisible *in this session only*, then drops it.	Slower and needs `CREATE INDEX` privilege, but gives real timings.

The panel explains the active mode in a line beneath the editor.

29.3 Reading recommendations

Press **Analyze**. Each candidate shows:

- The **table** it is on.
- A pill: ***n* % faster** (green) when the optimizer would use it and it helps, or **no gain** when it would not.
- The metric — `cost X → Y` in estimate mode, `Xms → Yms` in measure mode.
- The **columns** the index covers.
- The `CREATE INDEX` **statement**, with a copy button.

When there are no recommendations, the panel says **"The optimizer is already using good access paths."** — a useful answer in itself.

29.4 Explain with AI

Explain with AI sends the query and every candidate (with its metric and whether the optimizer would use it) to your AI provider, and returns plain-language advice on which indexes are worth creating and which are not, including trade-offs — write overhead, storage, redundancy with existing indexes.

---

30. ER Diagram Modeler

Reverse-engineer a schema into an entity-relationship diagram, edit it, and forward-engineer DDL.

30.1 Generating a diagram

Pick a **Schema** and press **Generate**. DBCraft reads the tables, their columns, and the foreign-key relationships, and lays out the diagram automatically.

 Generation is limited to the first 15 tables for performance. The empty-state message says so.

30.2 The toolbar

Control	Purpose
Schema	Which schema to read.
Generate	Build the diagram.
Layout	Choose the automatic layout algorithm; changing it re-lays out an existing diagram.
Zoom in / Zoom out / Fit to view	Navigation.
Export DDL	Downloads `schema.sql` (see §30.5).
Toggle overview	Shows or hides the minimap.
Find table…	Type a name and press `Enter` — the canvas pans and zooms to that table.

30.3 Editing the model

The diagram is not a static picture:

- **Rename a table** by editing its title.
- **Edit a column** — name, data type, length, nullability, primary-key flag.
- **Add** and **delete** columns.
- **Delete a table**.
- **Add Table** (the panel at the bottom of the canvas) creates `NEW_TABLE_n` with an `ID` primary key.
- **Draw a relationship** by dragging from one column's handle to another. The source column is marked as a foreign key automatically.
- **Click a relationship label** to cycle its cardinality; **delete** it from the same control.

30.4 The display panel

Down the right side:

Display Options

- **All Tables** / **Specified Tables** — with Specified, a checklist lets you show or hide each table individually.
- **Referenced Only** — hides tables that participate in no relationship.

Graph Controls

- **View** — **Graph** (the canvas) or **List** (a compact textual outline: each table with its columns, key icons, types and nullability, plus the tables it references).
- **Animate Layout** — smooth transitions when the layout changes.
- **Links to Columns** — attach relationship lines to the specific columns rather than the table box.
- **Show Link Name** — label each relationship.
- **Highlight Links** — hovering a column highlights every relationship it participates in, and the columns at the other end.
- **Node Content** — how much of each table to show.
- **Show Data Type** — include column types (disabled when node content is set to name only).

Overview — **Docked Minimap** toggle.

30.5 Exporting DDL

Export DDL generates a complete `schema.sql`:

- A `CREATE TABLE` for every table, with column definitions, `NOT NULL` where set, and a `CONSTRAINT PK_<table> PRIMARY KEY (…)` for the primary-key columns.
- An `ALTER TABLE … ADD CONSTRAINT FK_… FOREIGN KEY … REFERENCES …` for every relationship.

This includes your edits, so the modeler doubles as a schema design tool.

30.6 Status bar

Along the bottom: current zoom percentage, the number of visible **Tables**, the number of **References**, and the engine and schema.

---

31. Automation and Jobs

Build multi-step maintenance workflows, schedule them with cron expressions, and review their run history.

31.1 Creating a workflow

+ in the left header, give it a **Workflow Name**, **Create**. New workflows start disabled with a daily-at-midnight schedule (`0 0 * * *`) and no steps.

The left list shows each workflow with a status dot (green = enabled), its schedule, and the outcome badge of its last run.

31.2 The workflow header

- **Schedule** with a pencil to edit it.
- **Last run** and total **runs**.

- **Enabled / Disabled** toggle button.
- **Run Now** — executes immediately using the active connection.
- □ — deletes the workflow (with confirmation).

31.3 Steps

Add Step picks a type and an optional custom name. Six step types:

Type	Configuration
Execute Script	A SQL / PL/SQL script to run.
Schema Compare	A **Source Schema** and a **Target Schema**.
Export Data	None — uses the active connection's defaults.
Health Check	None.
Generate Report	None.
Email Notify	**Recipient Emails** (comma-separated) and a **Message Template**, which may include `{WorkflowName}` and `{Status}`.

Steps are numbered and executed in order. **Drag the grip handle** to reorder them; the ⚙ button opens the step's configuration; the □ button removes it.

31.4 Scheduling

The **Configure Schedule** dialog takes a standard five-field cron expression, with a quick reference:

Expression	Meaning
`* * * * *`	Every minute
`0 * * * *`	Hourly
`0 0 * * *`	Daily at midnight
`0 0 * * 0`	Weekly on Sunday

□ A workflow only runs on its schedule while it is **Enabled** and the application is running. Automation is an application feature, not a database job.

31.5 Execution history

The right column lists the last ten runs, newest first: a status icon (success / running / failed), the start time, the duration, and the run's log output in a scrollable block.

---

32. Load Tester

Model a business workflow as a tree of steps, then run it across many concurrent, real database sessions. It surfaces the lock contention and slow SQL that single-session testing never shows.

32.1 Layout

- **Left** — saved load tests for this connection, each with its last-updated time.
- **Right** — the current test: a header with actions, then three tabs — **Definition**, **Options**, **Results**.

32.2 Header actions

Button	Action
Load File	Opens a `.loadtest.json` file. Warns if you have unsaved changes or a run in progress.
Save ▾	**Save to Load Tests** (into DBCraft's local store) or **Save to File…** (a portable `.loadtest.json`, for version control or sharing between machines).
Run / Stop	Starts or stops the workload.

An **unsaved** badge appears next to the test name when there are pending edits.

32.3 Building the workflow (Definition tab)

The workflow is a tree. Three node types:

Node	Icon	Purpose
Step	▶	Executes SQL. Can be repeated *n* times and can pause after each execution.
Loop	⟳	Repeats its child nodes *n* times.
Pause	⏸	Think time — waits a fixed number of milliseconds.

The toolbar down the right side of the tree:

Button	Action
+	Add a step at the top level.
↳	Add a step *as a child* of the selected node.
⟳	Add a loop.
⏸	Add a pause.
↑ / ↓	Move the selected node among its siblings.
🗑	Delete the selected node and its children.

Selecting a node opens its editor below, with up to three tabs:

Properties — **Name**; plus **Repeat count** (steps), **Loop count** (loops), and **Delay after each execution (ms)** / **Pause (ms)**.

SQL (steps only) — the statement or anonymous PL/SQL block. Bind variables use `:name`. Two powerful behaviours:

- `RETURNING … INTO :name` **captures values** for use by later steps.
- A `SELECT`'s **first-row columns become variables** for later steps, under their column names.

Variables (steps only) — define **random variables**, each with a name and a min/max range. A new value is drawn on every execution, so each session hits different rows instead of all of them contending on one. Captured values from earlier steps are available automatically and need no declaration here.

New tests start with a two-node sample workflow (a probe `SELECT` and a 250 ms think time) so there is always something runnable.

32.4 Options tab

Concurrency

Option	Meaning	Default
Concurrent sessions	Threads; each opens its own real database session. Maximum 200.	5
Iterations per session	Complete workflow passes per session.	10
Ramp-up between sessions (ms)	Staggers session starts instead of a thundering herd.	0

Transactions

Mode	Behaviour
Autocommit	Each statement commits.
Commit each iteration	One commit per workflow pass.
Rollback each iteration	Rolls back each pass — keeps the schema clean while still exercising locks, undo and redo.

Scripts

- **Initialization script** — runs once before the test starts.
- **Finalization script** — runs once after all sessions finish.

32.5 Running and reading results

Press **Run**; the view switches to **Results** and polls live every second.

Stat tiles: Elapsed · Sessions (active/total) · Executions · Errors · Status. Errors turn red when non-zero. A fatal error appears in a red banner.

Step metrics table: per step — **Executions · Errors · Avg ms · Min ms · Max ms · Last error**. Rows with errors are tinted red.

Charts:

- **Average execution time by step** — a horizontal bar chart; bars for steps that produced errors are red.
- **Execution time over the run** — a line chart of individual execution durations against elapsed seconds, which is where you see contention building.

Run log — a live streaming log, capped at the most recent 2000 lines.

Export CSV (available once a run has finished) downloads the full result data.

 A load test does real work against a real database. Point it at a test system. If you must run it elsewhere, use **Rollback each iteration** and a low session count, and tell your DBA first.

Part V — Tools and utilities

33. SQL Terminal

A SQL*Plus-style console. Open it from the bottom dock (`Ctrl+`), the **Terminal** menu, or as a standalone window.

33.1 The login sequence

The terminal boots exactly like the real thing — a banner, then:

```
Enter user-name:
Enter password:
```

The password is never echoed. On success it prints **Connected to:** followed by the database's own version banner, then the `SQL>` prompt.

 Press `Enter` at the empty user-name prompt to reuse the active IDE connection's account. This is the fastest way in.

If the account cannot read `V$VERSION`, the terminal probes with `SELECT user FROM dual` before deciding the login failed — a login with limited privileges still works.

33.2 Entering statements

- A statement ends with `;`. Until then, lines keep buffering and the prompt becomes a line number, exactly like SQL*Plus.
- `/` on its own line runs the buffered block — use this for PL/SQL, where the semicolons are part of the code.
- A **blank line** discards the buffer.
- `Ctrl+C` with no selection also clears the buffer.

33.3 Built-in commands

Command	Action
`<statement>;`	Run a SQL statement (multi-line until `;`).
`/`	Run the buffered PL/SQL block.
`desc <object>`	Describe a table or view — name, type (with length/precision) and nullability. Accepts `SCHEMA.OBJECT`.
`connect` (`conn`)	Log in as a different user.
`disconnect` (`disc`)	Log out of the current session.
`show user`	Print the current user.
`history`	List the statements run in this console.
`clear` / `cls`	Clear the screen.

Command	Action
`help`	Print the command list.
`exit` / `quit`	Close the terminal.

33.4 Output and history

Query results are printed as aligned text tables. Long values are truncated at 40 characters; at most **200 rows** are printed, with a note about how many were omitted, followed by the total row count. Every statement reports `Elapsed: n.nns`.

`↑` and `↓` walk the statement history when the buffer is empty.

The header bar shows the connected user and host, a **Clear terminal** button and a **Close panel** button. The console keeps its classic dark palette regardless of the application theme.

 The terminal uses its **own** auto-committed connection. It does not see the SQL editor's uncommitted rows, and statements you run here commit immediately.

---

34. AI Assistant

An optional assistant that uses **your own** AI service and API key. It is off until you configure it.

34.1 Opening it

`Ctrl+Shift+A`, the toolbar button, or **Session** → **Show AI panel**. It is available on the **SQL Editor** and **Debugger** modules and slides in from the right; drag its edge to resize (220–480 px). Its visibility and width are remembered.

34.2 Configuration (bring your own key)

Configuration lives in **one place only: Settings** → **AI Configuration**. Until it is filled in, the panel shows *"AI is not configured"* with a button that takes you straight there; the ⚙ in the panel header opens the same section.

There is **no provider list** to choose from. DBCraft speaks the OpenAI-compatible API, so any endpoint that speaks it will work — a commercial service, a gateway, or a model server running on your own machine.

Field	Notes
API Key	Your key. A show/hide toggle reveals it. Optional for a local server on `localhost`.
Base URL	The endpoint, for example `https://api.openai.com/v1`. A suggestion list is attached, but you may type any URL. A pasted `…/chat/completions` path or a trailing slash is trimmed for you.
Model	Populated automatically by **Test Connection**, and disabled until then.

Test Connection validates the Base URL, checks the key, calls `GET {Base URL}/models`, fills the **Model** dropdown with what came back, and records the detected service name. The **Status** line then reads one of:

- ✓ **Connected** — *Provider: Google Gemini · Models: 12 discovered*.
- ✕ **Connection failed** — with the reason, for example *401 Unauthorized — the API key was rejected*, or *Invalid Base URL*.

If the endpoint does not offer model listing, the status still reports **Connected** and the panel says "*Unable to retrieve models automatically. You can enter a model ID manually.*" — the **Model** field switches to free text. **Enter a model ID manually** / **Choose from discovered models** toggles between the two at any time.

The detected service name — OpenAI, Google Gemini, Groq, OpenRouter, LM Studio, Ollama, or *Unknown OpenAI-Compatible API* — is **informational only**. It is derived from the Base URL, the response headers and the returned model list, and it never changes how a request is made.

Save stores the API key, Base URL, selected model and detected name on this machine. **Clear AI configuration and disable AI** removes them.

Base URLs that are known to work:

```
https://api.openai.com/v1
https://generativelanguage.googleapis.com/v1beta/openai
https://openrouter.ai/api/v1
https://api.groq.com/openai/v1
https://api.together.xyz/v1
http://localhost:11434/v1        (Ollama)
http://localhost:1234/v1         (LM Studio)
```

 Where your key goes. The key is stored locally on this device so you do not have to retype it, and is sent **only** to the Base URL you entered. It never reaches DBCraft's servers — there are none.

34.3 What the assistant can and cannot do

The AI is **strictly assistive**. It returns text. That is the whole of it.

It has **no** ability to:

- execute SQL, or send anything at all to your database;
- run scripts;
- create, alter or drop objects;
- commit, roll back, or open a transaction;
- disconnect a database or modify a saved connection;
- change any application setting;
- read local files.

This is not a rule the model is asked to respect — it is the shape of the integration. The assistant is given no tools and no callback into the application, so a reply reaches nothing but the chat bubble it is drawn into. The only action available on a reply is **Insert into Editor**, which *appends* the snippet to your active editor tab and stops there. Running it is a separate, deliberate step you take yourself, with the usual `F8`.

When a suggestion contains DDL or DML, the reply is flagged: "*This snippet changes data or structure. Review it before you run it.*"

These guarantees are enforced by DBCraft, not by the endpoint you configured, so they hold identically for every service and every model.

What is sent

Each request carries the minimum needed:

- your prompt;
- the **editor context** — the SQL you last executed, or the object you selected in the browser. A chip above the input box shows when context is attached, with a **Remove** link that detaches it. Long context is truncated.

DBCraft **never** sends database passwords, connection strings, credentials, API keys, private keys, secret values or local configuration files. Your API key travels in the request's authorisation header, never in the prompt. As a safety net, any secret that appears in the SQL itself — an `IDENTIFIED BY` clause, a pasted connection string, a key — is replaced with `[redacted]` before the request leaves your machine.

Schema accuracy

The assistant sees only the context above, not your whole catalogue. It is instructed not to invent tables, columns or packages: if you ask about an object it has not been shown, it should tell you it could not verify that the object exists and mark any placeholder names, rather than guessing at a schema.

 What leaves your machine. Your prompt and the attached SQL do travel to the endpoint you configured. Do not use the assistant against systems whose SQL or schema names you may not disclose to a third party. The rest of the product works fully with AI switched off.

34.4 Using the assistant

Pick an **action** — these shape the request:

Action	Purpose
Generate	Write SQL/PL/SQL from a description.
Explain	Explain what code does.
Optimize	Suggest performance improvements.
Fix	Fix errors and issues.
Review	Code review and best practices.

Type in the box and press `Enter` to send (`Shift+Enter` for a new line). Three **quick prompts** are offered as one-click starting points.

Assistant replies render as formatted text with code blocks. Each reply carries:

- **Copy** — copies the whole reply.
- **Insert into Editor** — appears when the reply contains a code block; appends that code to the active SQL editor tab. It is **not** executed (§34.3).

34.5 AI elsewhere in the product

The same configuration — and the same limits as §34.3 — powers two other features:

- **Performance Profiler → Get AI Suggestion** (§28.2)
- **Index Advisor → Explain with AI** (§29.4)

Both send only the statement and the analysis already on screen, and both report clearly when AI is not configured.

---

35. Export Tables and Import Tables

Two wizards for moving table structure and data as a portable `.sql` file. Both are Oracle-only and open from **Database → Export/Import Tables...**, the toolbar, or the command palette.

35.1 Export Tables

Left — what to export

- **Schema** — defaults to your own.
- **Filter tables...** and the table checklist, with **All** / **None** links and a `selected / total` counter.

Right — how to export

Option	Choices
What to export	**Structure + Data** (default) · **Structure only** · **Data only**
Include objects	Constraints, Indexes, Triggers, Comments (all on by default), Grants (off). Disabled when exporting data only.
Where clause	An optional row filter applied to every exported table, e.g. `department_id = 10`. Disabled when exporting structure only.

Export builds the script server-side (table DDL from the data dictionary plus `INSERT` statements) and downloads `export_<SCHEMA>_<date>.sql`. When it finishes you get a summary — how many tables, how many rows, how long — and any per-table errors are listed individually.

35.2 Import Tables

1. **Import file (.sql export)** — choose a file produced by Export Tables.
2. **If a table already exists** — five behaviours:

Option	Behaviour
Skip existing tables	Existing tables are kept as-is; their `CREATE` statements are skipped.
Append data	Existing tables are kept; imported rows are added.
Truncate before import	□ Existing rows are removed before the imported rows are inserted.

Option	Behaviour
Replace (drop & recreate)	□ Existing tables are dropped and recreated from the file.
Create only (fail if exists)	An error is reported when a table already exists.

1. **Commit** — a single transaction (commit at end), or every 100 / 1000 / 10 000 rows.
2. **Continue on error** — on by default; the import keeps going past a failing statement.

Import runs the script and reports statements executed, rows inserted, skipped, failed and elapsed time. A per-table results table shows whether each table was **Created** or already **Existed**, the row count (with failed rows in brackets), and any error. **Show execution log** expands the full log.

Part VI — Configuration and reference

36. Settings reference

File → **Preferences…**, the ⚙ icon, **Tools** → **Settings…**, or the user menu. Five sections down the left.

Changes preview **live** as you make them. The footer shows an amber **unsaved changes** marker; **Save** keeps them, **Cancel** (or `Esc`, or clicking outside) restores everything to how it was when you opened the dialog. **Reset all**, at the bottom of the section list, restores every default — except your profile.

36.1 Appearance

Setting	Detail
Theme	Light or Dark.
Primary / Secondary colour	A full colour picker — saturation/brightness area, hue slider, hex box with copy button, and a **Default** button. The dropdown in the row header chooses which colour you are editing. Primary drives the accent throughout the app; secondary drives surface tints.
Background	**Default**, **Neutral (White / Black)** or **Cool (Slate / Navy)**.
UI font size	11–16 px. Scales the *entire* interface, IDE-zoom style.
Editor font size	10–24 px. Affects the SQL editor and all code views.
Preview	A live SQL sample rendered at your current settings.

36.2 Accessibility

Setting	Detail
Colour vision mode	None · Deuteranopia · Protanopia · Tritanopia · Achromatopsia. Changes the status palette *and* the SQL editor's syntax colours to a matched safe scheme. A live badge preview shows success/warning/error/info in the selected mode.
High contrast	Increases contrast throughout.
Reduce motion	Disables animations and transitions.

 When a colour vision mode is active it takes ownership of the accent colour; a custom primary hex is suspended (not erased) until you turn the mode off.

36.3 AI Configuration

The **only** place the AI settings can be edited (§34.2): **API Key** (with show/hide), **Base URL**, **Model** (a dropdown filled by discovery, or free text), a **Test Connection** button and a **Status** line. A **Clear AI configuration and disable AI** action appears once a complete configuration exists. The configuration is stored only when both a Base URL and a model are present.

36.4 System *(DBA accounts only)*

This section is locked with a padlock unless the connected account holds DBA privileges; the locked view names the account you are connected as.

Setting	Default	Meaning
Query timeout	300 s	Maximum query execution time. Minimum 10.
Export row limit	100 000 rows	Maximum rows for a data export. Minimum 100.
Session timeout	30 min	Idle session timeout. Minimum 1.
Auto-commit	Off	Auto-commit DML after execution.

36.5 Profile

Profile picture — upload an image; it is cropped to a centred square and downscaled to 128 px. **Remove photo** clears it. **Display name** overrides the name shown in the header. Below, the connected e-mail and role are shown read-only.

---

37. Keyboard shortcuts

Global

Shortcut	Action
`Ctrl+K`	Open / close the command palette
`Ctrl+Shift+A`	Show / hide the AI assistant
`Ctrl+``	Show / hide the bottom dock
`Ctrl+Shift+``	New terminal
`Ctrl+Shift+Alt+``	New terminal window
`Ctrl+Shift+5`	Split terminal
`Esc`	Close the active dialog or sliding panel

SQL Editor

Shortcut	Action
`F5`	Execute
`Ctrl+Enter`	Execute
`Shift+Alt+F`	Format SQL
`Ctrl+↑` / `Alt+↑`	Previous statement from history
`Ctrl+↓` / `Alt+↓`	Next statement from history
`↑` / `↓`	History recall, when the tab is empty
`Ctrl+F`	Find (editor's own)

Shortcut	Action
Double-click a tab	Rename it

The toolbar's Execute, Execute script and Explain buttons carry the familiar `F8`, `F9` and `F6` labels from classic Oracle IDEs; use the buttons, the palette, or the keys listed above.

Command palette

Key	Action
`↑` / `↓`	Move selection
`Enter`	Run
`Esc`	Close

Terminal

Key	Action
`Enter`	Submit the line
`↑` / `↓`	Statement history
`Ctrl+C`	Clear the statement buffer

Results grid

Action	How
Edit a cell	Double-click (Edit Mode on)
Large Data Editor	Right-click a cell
Commit an edit	`Enter`
Cancel an edit	`Esc`

---

38. Licence management

Help → License…

38.1 Licence details

For an activated installation the dialog lists:

Licensed to · Edition · License ID · Seat (n / total) · Issued · Valid until (with days remaining, or *Perpetual*) **· Machine code**.

38.2 During a trial

A banner shows the evaluation period, days remaining and the machine code, above the activation panel.

38.3 Entering a new licence

Enter a new license… switches the dialog to the activation panel described in chapter 4. **Back to license details** returns.

38.4 Renewing an expired licence

When the licence has expired the panel's button reads **Renew License**. The procedure is identical to activation — paste the new key and press the button, or use the offline exchange.

38.5 Deactivating a machine

Use this **before** decommissioning a computer or moving your seat.

1. **Deactivate this machine…** → a confirmation appears.
2. **Deactivate**.

DBCraft removes the licence from this machine and tries to release the seat with the licensing authority automatically. If it cannot reach it, a **deactivation code** is shown.

 Save the deactivation code before closing the dialog if one is shown — entering it on the licensing portal is what releases your seat for reuse. If you close the dialog too soon, the code is also shown on the activation screen (in an amber box) until the next activation, so it is not lost.

---

39. Logging, diagnostics and bug reports

39.1 What is logged

DBCraft writes five log files:

File	Contents
`application.log`	General application events.
`session.log`	Connection sessions — start, end, user, engine, server version.
`error.log`	Everything at Error level and above, mirrored here.
`performance.log`	Timing records.
`debug.log`	Verbose diagnostics.

Each record is one line: `Timestamp LEVEL SessionID Module Message`.

 Credentials are stripped from every record before it is written. Request logging records only the method, path, status and duration — **never** request bodies, because every request to the local service carries live database credentials.

39.2 Retention

Retention is **per application session**, not per day. Each time the local service starts, the previous run's files are compressed to `<name>-YYYYMMDD_HHmmss.log.gz` and fresh files begin. The most recent previous session is kept.

This is deliberate: applications are usually restarted *because* something went wrong, and the report is filed from the next run. Discarding on startup would throw away exactly the logs the report exists to carry. There is no mid-session rotation, so a session's records always stay in one file.

39.3 Crash dumps

Unhandled errors write a readable `crash-*.dmp` — a text report containing the exception chain, stack trace, thread, loaded modules, memory state, application version and OS details. It is a text file, not a native minidump, so it survives being e-mailed. Crash dumps are capped by count rather than deleted with the logs.

39.4 Reporting a bug

Help → Report a bug…

1. Choose the kind of feedback (bug, idea, other).
2. Describe **what you did, what you expected, and what happened instead**.
3. Optionally add **your email** so support can reply.
4. **Attach diagnostic logs** (on by default for bugs). The dialog lists exactly which files will be included and their sizes, warns you if crash reports are present, and tells you where the logs live and for how long they are kept.
5. **Include diagnostics** adds a short environment block to the message body. Click **what is included?** to see it in full before sending.
6. Press **Send with logs**.

DBCraft collects the logs, the crash dump, system information and version details into a ZIP named `Logs_YYYYMMDD_HHMMSS.zip`, and opens a **pre-filled Outlook draft with the archive already attached**.

 Nothing is sent automatically. You review the draft and press Send yourself.

If Outlook is unavailable — or the draft opens but the file cannot be attached — DBCraft says so plainly rather than letting you send an empty report. It saves the ZIP, reveals it in Explorer, and opens your default mail client so you can attach the file by hand. The **Show archive** button reopens the folder.

For ideas and general feedback, no logs are collected; the report goes through a plain mail link, with **Copy** and **Save** as fallbacks.

 A bug report never contains your host, port, service name, username, password, or any SQL you have written. Credentials are stripped from the logs before they are written, and no saved credential data enters the archive.

Reports are addressed to the DBCraft support team.

40. Security and data handling

A consolidated summary of the guarantees stated elsewhere in this guide.

Area	Guarantee
Database credentials at rest	The password reaches disk only if you tick **Remember password**, and only as ciphertext from the Windows credential vault. There is no plaintext fallback: if the vault is unavailable, the password simply is not saved.
Credential erasure	**Clear saved credentials** and **Sign out** perform a deep erase that rewrites the underlying storage, so no remnant of a previously saved password survives. Settings and theme are preserved.
Credentials in logs	Scrubbed from every record before it is written. Request bodies are never logged.
Credentials in bug reports	Never included.
Network	All database traffic is between your machine and your database server. The only outbound call the product makes is the one-off licence activation.
AI	Off until you configure it. Your key is stored locally and sent **only** to the Base URL you entered. Your prompt and the SQL in context go to that endpoint — nowhere else, and secrets found in the SQL are redacted first. The assistant is text-only: it cannot execute SQL, reach your database, commit or roll back, or change any setting (§34.3).
Destructive actions	Killing a session, closing a PDB, dropping an object, restoring a version, deleting a suite/report/workflow/test, and truncate/replace imports all require explicit confirmation.
Generated scripts	Schema sync scripts, grid update scripts and ER-model DDL are **generated, never executed**. You review and run them yourself.

---

41. Troubleshooting

"Cannot reach the backend at :5020"

Every panel reports this when the local service is not responding.

1. Close DBCraft completely (check Task Manager for a lingering process) and restart it.
2. Check that nothing else on the machine has taken port 5020.
3. Check whether local security software is blocking loopback connections.

Connection failures at login

Error	Cause	Fix

Error	Cause	Fix
`ORA-01017: invalid username/password`	Wrong credentials.	Re-enter. Note that Oracle passwords may be case-sensitive.
`ORA-12541: TNS:no listener`	Nothing is listening at that host and port.	Verify host, port and that the listener is running.
`ORA-12514: listener does not currently know of service`	Wrong service name, or the database is not registered.	Check the service name; try SID instead.
`ORA-28000: the account is locked`	Account locked.	Ask a DBA to unlock it.

"The password could not be saved securely"

The Windows credential vault was unavailable. Everything except the password is still remembered. This is not an error you can work around by design — the product will not write a password in the clear.

A panel is empty, or a health check says "Unable to evaluate"

Your account lacks `SELECT` on the required dictionary or `V$` view. DBCraft names the exact grant needed in the message. Ask a DBA for that grant, or connect with a privileged account.

My inserted rows are not visible in a report / the terminal / the Data tab

You have not committed. The SQL editor runs in manual-commit mode; every other panel uses its own auto-committed connection. Press the ⎇ Commit button. See §16.6.

Other users are blocked behind my session

Your editor session is holding uncommitted DML and therefore row locks. Commit or roll back. The status bar's amber **Uncommitted** segment tells you when this is the case.

The debugger does not stop at my breakpoints

1. Recompile the program with debug information: `ALTER PROCEDURE my_proc COMPILE DEBUG;`
2. Confirm your account has `DEBUG CONNECT SESSION`.
3. Confirm the source shown is the source that is actually executing — after **Step Into** the highlight is suppressed if execution has moved into a different program (the status badge names it).

The container switcher shows no PDBs

Oracle hides `V$PDBS` rows from common users by default. Use the **Enable PDB visibility** button in the dropdown, or ask a DBA to run `ALTER USER <you> SET CONTAINER_DATA=ALL CONTAINER=CURRENT;`.

PDB open/close controls are missing

They appear only when connected as `SYS`, because Oracle requires a `SYSDBA` session for PDB lifecycle operations.

The AI assistant returns an error

Open **Settings** → **AI Configuration** and press **Test Connection** — it reports exactly what failed (an invalid Base URL, a rejected key, an unreachable host) and refreshes the model list. The endpoint's own error message is shown in the assistant's reply.

The ER diagram is missing tables

Generation is capped at the first 15 tables for performance. Use **Specified Tables** and **Referenced Only** in the display panel to focus the diagram, or model the remainder by hand.

Excel export produced fewer rows than expected

Check **Settings** → **System** → **Export row limit** (DBA accounts only).

Nothing happens when I press Save in the SQL editor

Save writes a `.sql` file to disk; it does not write to the database. If your browser-style download location is set to prompt, look for the dialog. Use **Save As…** to choose the location explicitly.

Appendix A — File and folder locations

What	Where
Application	The folder you chose during installation
Log files and crash dumps	The logs directory shown in the bug-report dialog (**Help** → **Report a bug…**)
Diagnostic archives	`Logs_YYYYMMDD_HHMMSS.zip`, revealed in Explorer when created
Saved reports, report templates	DBCraft's local report store
Test suites	DBCraft's local test store
Load tests	DBCraft's local load-test store
Object version history	DBCraft's local version store
Settings, editor tabs, layout, AI configuration, graph templates, saved credential profile	Local application storage on this machine

Exported files (queries, DDL, CSV/JSON/HTML/Excel, `.dbctest.json`, `.loadtest.json`, debug logs, diagrams) go wherever your save dialog points — usually your Downloads folder unless you choose otherwise.

Appendix B — Backend API reference

The local service exposes the following endpoints on `http://localhost:5020`. This is provided for support and diagnostics; the endpoints are not a documented integration surface and may change between versions.

Connection and session — `POST /api/oracle/test-connection`, `containers`, `pdb-operation`, `execute-query`, `commit`, `rollback`, `close-session`, `sessions`, `kill-session`

Metadata — `schemas`, `list-objects`, `get-ddl`, `resolve-object`, `get-object-columns`, `compile-plsql`, `drop-object`, `schema/relationships`

Analysis — `explain-plan`, `profile-sql`, `index-advisor`, `health-check`, `admin/metrics`

Storage and infrastructure — `storage/tablespaces`, `storage/segments`, `storage/datafiles`, `asm/overview`, `rac/overview`

Schema tools — `schema/compare`, `schema/sync-script`, `schema/export-ddl`

Data movement — `export/csv`, `export/excel`, `export/data`, `import/csv`, `import/excel`, `import/json`, `export-tables`, `import-tables`

Debugger — `POST /api/oracle/debug/start`, `step`, `stop`, `watch`, `breakpoint/toggle`

Testing — `/api/testing/sets`, `cases`, `cases/reorder`, `session/open`, `run`

Reports — `/api/reports` and `/api/reports/templates` (full CRUD)

Version history — `/api/versions/capture`, `objects/{id}`, `history`, `item/{id}/label`, `rollback`

Load testing — `/api/loadtest/tests`, `run`, `run/{id}/status`, `run/{id}/stop`, `run/{id}/export`

Automation — `/api/automation/workflows` and `workflows/{id}/run`

Licensing — `/api/license/status`, `challenge`, `activate`, `activate-online`, `deactivate`

Diagnostics — `/api/diagnostics/status`, `tail`, `log`, `crash`, `session/start`, `session/end`, `bug-report`

Appendix C — Glossary

Term	Meaning
Bind variable	A `:name` placeholder in SQL whose value is supplied at run time. Used by report parameters, test scripts and load-test steps.
CDB / PDB	Container Database / Pluggable Database — Oracle Multitenant. `CDB$ROOT` is the root container.
Cluster (in the left panel)	A collapsible section of the left drawer, not an Oracle cluster.
Dock	The panel that slides up from the bottom of the window.
Estimate mode / Measure mode	Index Advisor strategies — a hypothetical index costed by the optimizer, versus a real invisible index that is actually timed.
Manual-commit mode	The SQL editor's mode: DML is not durable until you press Commit.
Machine code	A hardware fingerprint identifying this installation for licensing. Contains no personal information.
Module	One of the feature areas selected from the docking rail.

Term	Meaning
Rail	The vertical strip of module icons at the left edge.
Seat	One activation of your licence on one machine.
Segment	An Oracle storage structure — a table, index, partition or LOB — occupying space in a tablespace.
Session (database)	A connection to Oracle, listed in the Session Browser.
Session (application)	One run of DBCraft, used as the unit of log retention.
Snippet	A ready-made SQL template from the ⋆ library.
Step (load test)	A node in a load-test workflow that executes SQL.
Suite (test)	A named, ordered collection of PL/SQL test scripts.
Workflow (automation)	A named, ordered sequence of automation steps with a cron schedule.
