AgentSQL

Zhidun

User Guide

v4.6.0

agentsql.cn

v4.6.0 Documentation

Contents

v3.9.0 Oracle and SQL Server controlled reads

v3.8.0 ClickHouse budgets and Milvus tenant filters

v3.7.0 MongoDB and Sentinel controlled reads

v3.2.0 SQLBot Integration /sqlbot

Capability Catalog and Delivery Diagnosis

v2.9.0 Policy Package Governance and External Audit Witness

v2.8.0 Milvus controlled vector retrieval

v2.7.0 ClickHouse Controlled Analysis Query

v2.6.0 Redis/Valkey controlled key reading

1. Overview /

Access Wizard /onboarding (v1.9.0)

v2.4.0: openGauss 7.0.0 LTS specified instance controlled read-only

v2.3.0: SQL Server 2025 specified instance controlled read-only authentication

v2.2.0: Oracle Free specified instance controlled read-only verification

v2.1.0: YashanDB specified instance controlled read-only verification

v2.0.0: Fixed dual gateway with PostgreSQL column policy registration

2. Audit /audit

    Filter parameters

    Export

3. Demonstration stand /playground

    Static evaluation: all normal deployments have

    Live Demo: Demo mode only

4. Agent /agents

5. Data source /datasources

    YashanDB client configuration

6. Permission /policies

    objects and actions

    Column allowlist

row_filter
Usage boundary of temporary read-only authorization /task-read-grants (v1.0.0)
Application, approval and receipt of task authorization (v1.0.0)
MCP user identity delegation (v1.0.0 optional)
7. Rule /rules
Built-in rules directory
Current editability
Thresholds and dynamic dependencies
8. Approval /approvals
9. masking /mask-rules
Type and algorithm capability matrix
Three scopes and selection suggestions
Configure range value bucketing
Configure range date truncation
API fields and full update semantics
Exact matching and fail-closed security
Sensitive discovery workflow
Verification and conflict error reporting
range Behavior and Boundaries
masking key rotation and equivalence boundary
Known limitations and recommendations
10. Configuration reference
server
store
defaults
theme
demo
redaction
environment variables
11. Operation and maintenance
agentsqlctl public subcommand
health

Upgrades, rollbacks and backups

Performance caliber

12. Current version capability boundaries

Online installation package upgrade preview (different from schema check)

v4.2.0 offline delivery and diagnostics

v4.2.0 native data Q&A

v4.3.0 charts, dashboards and embedded applications

v4.4.0 account administration and datasource relationships

AgentSQL User Manual

Language: English | Simplified Chinese

> Current guidance: Applies to v4.4.0 and compatible 4.4.x maintenance releases. Historical introduction versions and acceptance records retain their original facts; see the document index for the current entry point.

> This guide covers AgentSQL v4.4.0. Before installing or pulling v4.4.0, check the official assets and signatures in GitHub Releases and confirm the matching GHCR image can be pulled anonymously. The source repository is `github.com/cuipengdba/agentsql`.

This manual follows the v4.4.0 console menus. Start with Getting Started; see the MCP integration guide for clients, the v4.4.0 release notes and design and acceptance record for verified scope, and the notification guide for Webhook and Syslog boundaries.

The console login page and the language button in the upper right corner after logging in can switch between Chinese and English, and the selection will be saved in the current browser; `?lang=en` or `?lang=zh` can specify the first opening language. The left menu can be scrolled independently when the effective viewport height is insufficient, and the bottom options can also be accessed with the keyboard after focusing the menu. Language switching only changes the interface copy and does not modify the SQL, query results and audit records. Complete English documentation is available through the language link at the top. Capability claims are bounded by the corresponding acceptance record.

v3.9.0 Oracle and SQL Server controlled reads

- For specified Oracle 26ai Free, choose `verify-full`, supply an absolute CA file path, and use a connection host matching the certificate name. The gateway verifies the certificate on every new connection. Oracle wallets, mTLS and RAC remain outside this acceptance scope.
- Enable `{"oracle":{"explain":true}}` in `read_control_json` only with a dedicated non-system readonly account and a verified `PLAN_TABLE`. Authorized narrow SELECTs then receive bounded plan checks; unavailable or invalid plans fail closed. Existing Oracle sources keep plan checks disabled by default.
- Use MCP `query` for both Oracle and SQL Server. On SQL Server, keep strict certificate validation and scoped SELECT/SHOWPLAN grants. Validate allowed and denied calls, masking and durable audit IDs after certificate rotation and recovery. The public Demo remains synthetic PostgreSQL/MySQL. See the release notes, upgrade guide and acceptance record.

v3.8.0 ClickHouse budgets and Milvus tenant filters

- On the datasource page, select ClickHouse or Milvus and fill in optional `read_control_json`. ClickHouse budgets tighten rows read, bytes read, and memory; the DBA must first configure constrained per-query settings on the readonly role. For Milvus, identify the tenant field and approved equality-filter fields. Collection tenant values must match AgentSQL tenant IDs.
- Grant the Agent exact tables or collections and all output, filter, and Milvus tenant fields. Call MCP `query_analytics` or `search_vectors` to verify allow, deny, masking, audit, and cross-tenant isolation. The public Demo does not include these native databases. See the design, upgrade guide, and acceptance record for configuration and limits.

v3.7.0 MongoDB and Sentinel controlled reads

- In the datasource page, set `read_control_json` with topology and approved-index metadata only. A MongoDB replica set must list three trusted members and collection indexes; Redis/Valkey Sentinel must list three Sentinels, allowed masters and a separate ACL username. Provision the Sentinel password to each gateway using the `sentinel_password_env` shown in datasource details, never in JSON.
- Grant the Agent the exact MongoDB collection and required fields, or the Redis/Valkey logical database, Hash key and each field. Call MCP `find_documents` or `read_hash_fields` and check allowed and denied outcomes, masking and audit IDs. Wrong certificates, unknown members, missing indexes, revoked ACLs and audit outages must deny delivery. The public Demo does not include these native topologies. See the NoSQL support scope, upgrade guide and acceptance record for configuration and tested boundaries.

v3.2.0 SQLBot Integration `/sqlbot`

Under Settings and Integrations → SQLBot Integration, map an ordinary login user, workspace, and datasource name exactly to an existing AgentSQL person, read-only Agent, and PostgreSQL or MySQL datasource. For MySQL, the discovery schema is the datasource database name. Enter workspace and user IDs as strings. Creating, editing, or rotating connector credentials requires strategy, user, and datasource management permissions together. The person must retain query execution permission, and the Agent must remain active and read-only.

Connector credentials are displayed only once, when created or rotated. Store them and the independent Agent Key in separate environment variables on the SQLBot server; exported JSON contains only variable names. Editing, disabling, deleting, or rotating a binding invalidates previously issued SQLBot person tokens on subsequent requests. Revoke the Agent Key to stop all access by that Agent. Configuration diagnostics do not certify a successful live business query.

After installing the adapter for pinned SQLBot v1.10.2, ordinary questions filter model-visible tables and fields before execution through the gateway. Denials do not trigger automatic rewriting and retry. Users can explicitly request read-only approval, inspect its outcome, and execute approved sealed SQL once at SQLBot's `/api/v1/agentsql/console.html`. See the SQLBot Integration Guide for complete configuration, the supported SQL subset, and trust boundaries.

Capability Catalog and Delivery Diagnosis

The "Data Source" page of the console displays Agent capability levels, entry tools, limited scope, and acceptance documents by type. The `support` field of `/api/v1/datasource-types` comes from the same server directory; MCP `list_datasources` only comes with the same description for the data sources visible to the current Agent. The levels are `controlled_sql`, `bounded_sql`, `controlled_read_tool` and `connection_preview`. The last layer only represents the connection or management preview and cannot be used as Agent read permissions. Real requests are not automatically authorized at any level and still need to pass through identity, object and column policies, enforcement, masking and persistent auditing.

Run `agentsqlctl doctor --config /etc/agentsql/config.yaml --format json` in the target configuration in an isolated environment before upgrading. An exit code of 0 indicates that the migration history for the configured storage is ready, 2 indicates that a migration is pending, and 1 indicates that the configuration or storage check failed. This command does not migrate the schema, does not create missing SQLite libraries, and does not output DSN, passwords or keys. Ready migration history does not mean that the business data is complete or available for production. It still needs to be backed up and restored according to Deployment Guide . `--format text` can be used to view short human read results.

v2.9.0 Policy Package Governance and External Audit Witness

The "Policy Package Management" page exports the current tenant policy package and summary. Generate a difference plan after editing the JSON, and can fill in up to 32 precise Agent/data source/table cases to view offline table policy candidate changes. After the initiator submits the proposal, he hands the proposal ID to another administrator with "Policy Management" and "Audit View" permissions for approval; only the approver can apply it. If the source policy has changed during application, it needs to be re-exported, generated and approved. Changing the policy directly after approval will be displayed as `drifted`. For the changeable scope of this version, the 128 strategies, and the upper limit of 32 changes, see and design .

A simple example of an independent audit witness (keys and witness files should be placed outside the database and gateway data volumes):

```
agentsqlctl audit checkpoint-keygen --private-seed-out /secure/audit-checkpoint-seed.hex --public-key-out /secure/audit-checkpoint-public.hex
agentsqlctl chain verify --domain traffic --config /etc/agentsql/config.yaml
agentsqlctl chain status --domain traffic --json --config /etc/agentsql/config.yaml
agentsqlctl audit report --format archive --out /secure/audit-export.zip --config /etc/agentsql/config.yaml
agentsqlctl audit checkpoint-create --in /secure/audit-export.zip --out /secure/audit-checkpoint.json --private-seed-file /secure/audit-checkpoint-seed.hex --chain-id traffic --head-seq "$HEAD_SEQ" --head-hash "$HEAD_HASH"
agentsqlctl audit checkpoint-verify --in /secure/audit-export.zip --checkpoint /secure/audit-checkpoint.json --public-key-file /secure/audit-checkpoint-public.hex --max-age 24h
```

Start with `chain status --json` Check and set `HEAD_SEQ` and `HEAD_HASH` ;Do not copy unverified values directly. Subsequent testimonials require the previous copy `checkpoint-create` Output `sha256` as `--previous-sha256` ;Provide the same value from an independent trusted record to `--expected-previous-sha256` . Archives can contain filtered results, so the link head value is `chain verify` individually verified**statement**, the tool will not recalculate the complete link header from the filtered archive. The file is not a WORM or trusted timestamp.

v2.8.0 Milvus controlled vector retrieval

Select Milvus in the data source, fill in the `default` database, the gRPC TLS port of the 2.6.25 instance, the absolute path of the CA that the gateway can read, the host name of the matching certificate, and the non- `root` account password. The database account should only have Search permissions on the target collection. Create a precise `default.collection` table authorization and column whitelist for the same object for Agent. The whitelist must include vector fields and each returned field. MCP `search_vectors` submits a finite float32 vector, explicit scalar field, and `top_k` of 1–20; the gateway checks the dimensions against the COSINE index and delivers after successful masking, persistence preflight, and final state auditing. Neither the original management preview nor the general SQL `query` is the entrance to this capability. Please refer to v2.8.0 for design and for candidate acceptance .

v2.7.0 ClickHouse Controlled Analysis Query

Select ClickHouse in the data source, fill in the specified database, native TLS port, verifiable CA, certificate host name and read-only account. The database role needs to be fixed to `readonly=1` and limit query time, read rows/bytes, memory, output rows and threads. Establish accurate `database.table` table authorization and column whitelist for the same object for Agent. The whitelist must include return columns and filter columns. MCP `query_analytics` only accepts equivalent conditions of table names, explicit columns, String columns, and an upper limit of 1–100 rows; the gateway only queries MergeTree single tables, rechecks permissions one by one, and delivers after successful masking and persistent preflight/final state audit. The general `query` SQL tool is not the entrance to this capability. The scope of support is subject to v2.7.0 design and acceptance after completion.

v2.6.0 Redis/Valkey controlled key reading

Select Redis or Valkey in the data source, configure fixed DB, strict TLS, verifiable CA and minimum ACL account; establish `key` policy for ordinary Agent, and write the range as `dbN:exact-key` or bounded `dbN:prefix*`. MCP `read_key` Fill in only `datasource_id` and exactly `key`. Policy denial takes precedence, and SQL table policies cannot replace key authorization. The server has fixed `TYPE` and `GETRANGE` of up to 64 KiB, and only accepts UTF-8 strings; it will be delivered after successful masking and persistent pre-checking and final state audit. Rejections or errors can be located using the response audit ID and the original key or value is not saved in the audit record. Only Redis 8.2.9 / Valkey 8.1.10 TLS/ACL single-instance synthetic environment has been verified. For details, see and for acceptance of this version.

1. Overview `/`

The overview brings together recent decisions, trends and real-time events on one page:

- `allow`: Rules and policies allow execution requests to enter controlled execution.
- `deny`: Intercepted by identity, permissions or rules, and does not reach business SQL execution.
- `warn`: Allowed execution, but with warnings that require attention.
- `approve`: Switch to manual approval, SQL is not executed.
- `error`: An error occurred in the parsing, evaluation, storage, or execution link.

```
Request → Authentication → Parsing → Static rules → Necessary dynamic stages → Decision
                                                    ├─ allow → execute → desensitize → audit
                                                    ├─ warn → execute → desensitize → audit
                                                    ├─ deny → do not execute → audit
                                                    ├─ approve → do not execute → pending approval + audit
                                                    └─ error → failure → try to audit
```

Trend graphs are used to observe the number of decisions over time; real-time event streaming uses SSE to push new audit events via `GET /api/v1/stream`. When reverse proxying this path you must turn off caching, response compression, and buffering, and set the idle timeout to greater than 25 seconds. `server.event_stream_max_connections` controls the upper limit of concurrent SSE connections.

Note: Tool layer forward rejections may not have entered the pipeline yet, so not every failure will appear in the event stream or audit page.

Access Wizard `/onboarding` (v1.9.0)

Select the Agent and data source in the current tenant and run "Connection and Permission Precheck". The page checks whether the Agent is enabled or expired, tests the connection using a saved database account, and checks whether a policy exists between the Agent and the data source; Agent SQL will not be executed. When the policy exists, only "Requires Verification" is displayed because the number of policies cannot prove that a certain SQL is authorized.

v1.9.0 Added access instructions, target SQL offline interpretation and audit ID positioning for Dify 1.17.1 / MCP Inspector 0.16.8 on the same page. First fill in the AgentSQL root address reachable by the client, copy the configuration instructions containing the `<Agent API Key>` placeholder, and then fill in the real Key safely on the client; the console does not accept the Key. Remote plaintext HTTP will transmit the Key, so HTTPS should be used or restricted to an isolated intranet. Dify requires explicit `MCP-Protocol-Version: 2025-06-18`, which is automatically added after Inspector 0.16.8 is initialized and cannot be set manually again. `127.0.0.1` in the Dify container points to Dify itself, and the container reachable address needs to be used.

Offline interpretation only returns "clearly denied" or "cannot confirm permission yet" and can list objects, matching policy IDs and static rules; it does not connect to the business database and does not execute SQL. Finally, the real client initiates allow and deny requests respectively, fills in the positive integer audit ID of the response into the wizard to directly access the audit details, and checks the Agent, data source, decision-making, rules and masking. If there is no audit ID, check the authentication, protocol or dependency fault first according to the error code. Wizards are not a substitute for per-object authorization, execution links, or production support acceptance.

v2.4.0: openGauss 7.0.0 LTS specified instance controlled read-only

When creating the `opengauss` data source, you must fill in a host name that can be resolved to the target server and is consistent with the certificate SAN, and configure `verify-full` and a trusted CA; self-signed certificate exemption, plain text, and other TLS modes are rejected. The database account is only granted SELECT to the target ordinary table and keeps the default transaction read-only. The gateway verifies the specified LTS build and read-only status each time a new physical connection is created.

MCP SQL only accepts SELECTs with explicit schema, `FROM ONLY` single ordinary physical table, direct column projection, non-constant WHERE, and 1–10000 integer LIMIT. Views, surfaces, JOINs, functions, aggregates, system catalogs, writes, B2 column-level bindings, and B5 sessions are not open. First, pre-check in the access wizard, and then use the real MCP Inspector or Dify to initiate permission and unauthorized table rejection calls to check masking and persistent records with the audit ID. See [for candidate acceptance] for boundary and failure testing of specified instances.

v2.3.0: SQL Server 2025 specified instance controlled read-only authentication

This version uses strict TLS to verify the local test CA on the designated synthetic instance of SQL Server 2025 Developer x86_64, obtains only the temporary account of the target table `SELECT` and the database level `SHOWPLAN`, and verifies the real HTTP MCP, table policy R010, mobile phone number masking and persistent auditing. Inspector and Dify complete allow/deny calls; revoking SELECT, revoking SHOWPLAN, business database and independent audit library shutdown are all accepted without delivering the original results. Please use a verifiable server certificate and a dedicated minimum privilege account when creating data sources; static R010s without permission tables terminate at the gateway and are not replaced with database permission errors. This result does not cover Windows, Always On, other editions, production certificate rotation, complex SQL, write, B2/B5, or full production support. See [candidate acceptance] for details.

v2.2.0: Oracle Free specified instance controlled read-only verification

In this version, in the isolated instance of Oracle AI Database 26ai Free, `FREEPDB1`, and server `23.26.3.0.0`, the temporary account that only obtains `CREATE SESSION` and the target table `SELECT` is used to verify the real HTTP MCP, table policy R010, mobile phone number masking, and persistent auditing. The allow/deny calls of Inspector and Dify, database SELECT revocation, business database and independent audit library shutdown are all re-examined using synthetic data. The Oracle running account has not verified `PLAN_TABLE` write permission, and the gateway does not pre-execute `EXPLAIN PLAN`; commercial edition, TCPS, active and backup, complex SQL, writing and complete production support are not included in this version statement. For details, see candidate acceptance .

v2.1.0: YashanDB specified instance controlled read-only verification

In this version, in the isolated instance of YashanDB Enterprise Edition 23.4.1.109, `COMPAT_VECTOR=yashan`, a temporary minimum privilege account is used to verify the real HTTP MCP, table policy R010 rejection, mobile phone number masking and persistent auditing. The allow/deny calls of MCP Inspector and Dify, database SELECT revocation, business database and independent audit library shutdown have also been tested with synthetic data. This conclusion is limited to the specified version, mode, and narrow SELECT subset; writes, transactions, EXPLAIN, server-side TLS, active-standby, client cancel/timeout remain unvalidated or remain closed. See candidate acceptance and YashanDB dialect boundary for details.

When the authenticated Agent calls `query` or `execute_write` on an invisible data source, the gateway first writes a preflight without original SQL to reject the audit, and then returns the audit ID; when the audit library fails, an error is returned and the data is not delivered. Non-SQL tools such as `list_schema` and authentication failures do not belong to this traffic audit path.

v2.0.0: Fixed dual gateway with PostgreSQL column policy registration

A dual-gateway Kubernetes deployment only hosts stateless MCP read-only requests. PostgreSQL metadata and independent audit libraries are shared between two pinned replicas, with each replica using its own B2 instance ID; B5 cross-request transactions, HTTP status sessions, and event resumption are turned off in this configuration. Operation and maintenance personnel must perform migration inspection, maintenance window switching, fault drills and rollback according to Dual Gateway Operation Guide . Public Live Demo for a single gateway synthetic data environment.

Administrators with policy management permissions can first create the `object_type: column` and `action: allow` policies for a single ordinary PostgreSQL table. Using this policy, the current `ETag` calls `POST /api/v1/policies/{id}/bind-postgres`, and the request body is `SELECT` of a projected precise column list. The registration process only analyzes the SQL and database directories, and does not execute the SELECT; after success, the permission, denial, masking and persistent audit must be verified on the real MCP client. Object mismatches, extra referenced columns, multiple statements, views, cross-tables, wildcard projections, or native extended modes will be rejected; old `ETag` returns 412. This API only covers PostgreSQL single-table scope in this version of closed binding mode.

2. Audit `/audit`

The audit page records the SQL, objects, rule hits, decisions, estimated number of rows, number of returned rows, time consumption and error information that pass through the application pipeline. It is intended for engineering troubleshooting and behavioral tracing, not regulatory grade WORM.

Filter parameters

parameters	meaning
`time_start` / `time_end`	RFC3339 time range
`agent_id`	Agent ID
`datasource_id`	Data source ID
`session_id`	Call session ID
`mcp_tool`	MCP tool name
`decisions`	Decision list, comma separated
`stmt_types`	List of statement types, comma separated
`risk_min` / `risk_max`	Risk scope
`keyword`	SQL or error message keyword
`object`	object name

The default paging is `page=1&page_size=20`, and the valid range of `page_size` is 1–100. The details drawer displays the request identity, SQL, objects, rule hits, time spent at each stage and final decision, forming an evidence chain for a single judgment.

Export

The export follows the filter conditions of the current page and supports NDJSON (JSONL), CSV, PDF and compliance archiving (ZIP). The upper limit is 10,000 items at a time. If the upper limit is exceeded, HTTP 422 is returned, and the page prompts to reduce the time range or add filter conditions. Export responses with `Cache-Control: no-store` and `X-Content-Type-Options: nosniff`.

NDJSON (JSONL, machine-oriented)

- `Content-Type: application/x-ndjson`, file name `agentsql-audit.jsonl`, no BOM, one JSON per line (snake_case).
- Suitable for importing into SIEM, log platform, or further analysis with script / `jq`; when the fields are too long (very long SQL, detailed JSON), it is not limited by the cell length of the table software.

CSV (for manual compliance reports)

- `Content-Type: text/csv; charset=utf-8`, file name `agentsql-audit.csv`, file header with UTF-8 BOM, Microsoft Excel / WPS can be opened directly by double-clicking without garbled characters.
- The first line is a fixed 25-column Chinese table header: time, audit ID, decision, risk level, Agent ID, data source, database type, session ID, conversation ID, MCP tool, statement type, hit object, hit rule, original SQL, normalized SQL, estimated number of rows, number of returned rows, milliseconds taken, client IP, model, action, executor type, executor ID, error message, details JSON.
- The time column is unified to RFC3339 of UTC (with nanoseconds and ends with `Z`); the numeric column is pure numeric, and empty values are left blank; fields containing commas, quotation marks, and line breaks (such as original SQL, detail JSON) are wrapped in English double quotes according to the CSV specification, and the quotation marks in the field are escaped into two double quotation marks, and the multi-line content is still located in the same cell.
- Formula injection protection: To prevent text starting with `=`, `+`, `-`, `@` (including full-width `＝` `＋` `－` `＠`) or starting with Tab / Carriage Enter / Line Feed from being mistakenly executed as a formula in Excel / WPS, such text cells will be preceded by an English single quote. This is a measure to reduce the risk of formula injection in spreadsheet software, and it does not constitute an absolute guarantee for all office software. If you need to perform secondary processing on the original text, please use JSONL export as the standard. Do not delete the leading single quotes in the spreadsheet software and then save and overwrite it.
- Consistency and boundaries: The export is read in paging order and is not a transaction snapshot; if the audit data continues to be written during the export period, it may be duplicated or omitted in rare cases. If strict and consistent evidence collection is required, please query the audit database directly. CSV is highly sensitive data. Please save and transfer it according to internal compliance requirements and do not transfer it to others at will.

- Cell length: The upper limit of a single cell in some spreadsheet software is about 32767 characters. Overly long SQL/details may be displayed abnormally in the table. In such scenarios, please use JSONL instead.

PDF and compliance filing

- The PDF is generated by pure Go on the backend and does not rely on the browser or external commands; it contains filtering metadata, decision/action distribution, and event summary. Long fields are bounded and truncated in the PDF. The exact original text is still based on the JSONL in the archive.
- The compliance archive is named `agentsql-audit-archive.zip` and always contains `audit.csv`, `audit.jsonl`, `report.pdf`, `summary.json`, and `manifest.json`. The manifest records the media type, byte count, and SHA-256 for the four payloads.
- Available `agentsqlctl audit verify-archive --in <file.zip>` offline verification member set, size and summary. Verification does not equal digital signature, source authentication, trusted timestamp, WORM retention, or "this is the latest copy"; if an attacker can rewrite the payload and manifest at the same time, he can still construct a new self-consistent package.

export behavior traces (audit.export): Each successful audit export will add a management operation record to the audit library, with the action `audit.export` and the executor type `admin`, which is used to trace "who exported at what time, from which source IP, in what format (JSONL/CSV/PDF/archive), under what filter conditions, and how many items were exported." This record only contains the meta-information of the export behavior (format, number of items, filtering dimensions, source IP), and does not include the exported audit content; when using keyword search, it only records whether the keyword (`has_keyword`) is used, and the original text of the keyword is not saved. The source IP is the server's directly connected peer address, and forged request headers such as `X-Forwarded-For` are not trusted. The trace is a best-effort operation: even if the trace writing fails, it will not affect the successful export. It only records alarms on the server and does not trigger real-time notifications. The `audit.export` record itself will appear in subsequent audit lists and exports, and count toward the single export limit of 10,000 records (one row added for each successful export).

3. Demonstration stand `/playground`

The demo bench has two different sets of capabilities and cannot be confused.

Static evaluation: all normal deployments have

`POST /api/v1/playground/assess` only parses SQL and runs static rules. It does not connect to the business database, execute SQL, or write audits. The six built-in examples are:

scene	expected
Normal check	`allow`, hit without rules
None `WHERE` Full table update	`deny`, hit R002
stack injection	`deny`, at least hit R001, R006
`pg_sleep` Dangerous function	`deny`, hit R007
MySQL batch write with `WHERE` and without `LIMIT`	`approve`, hit R202
Incomplete SQL	`error`, parsing phase fail-closed

`stageSamples` in the interface is only used to show the allow/deny/approve/warn/error process sample, not the third set of executable scenarios.

Live Demo: Demo mode only

`POST /api/v1/playground/run` is only registered in dedicated Demo mode. It uses fixed synthetic data sources and demonstration identities. The first five cards will actually enter the controlled pipeline, and the sixth card is an audit review:

1. PostgreSQL allows read-only queries, returns no more than 5 rows and writes audits.
2. MySQL None `WHERE` update is rejected by R002 and Demo read-only barriers and does not touch the database.
3. MySQL `phone`, `email` results are masked.
4. PostgreSQL large results hit R005 under Demo threshold, truncated to at most 20 rows.
5. Unauthorized `internal_notes` rejected by R010.
6. View audit details with recent `audit_id` and jump to the overview live stream.

Normal production deployments do not have `/playground/run` and Live controls. For the complete startup method, see DEMO.md.

4. Agent `/agents`

Agent is an independent identity that calls AgentSQL.

Field	Description
`id`	Unique ID
`name`	Name
`owner`	person in charge
`level`	`readonly`, `dml` or `ddl`
`status`	`active` or `disabled`

Field	Description
`expires_at`	Optional expiration time

Ability level:

- `readonly`: Only queries are allowed.
- `dml`: Allows querying with `INSERT`, `UPDATE`, `DELETE`.
- `ddl`: Contains DDL capabilities; specific SQL is still subject to policies and rules.

The API Key generated when creating an Agent is only displayed once. After the key is rotated, the old key becomes invalid immediately and there is no concurrent validity period. When the Agent is disabled, expires, deleted, or rotated, the original Key can no longer pass authentication.

Do not understand `qps_per_agent` as an editable field for each Agent: it is a global default. The connection upper limit `conn_limit` is configured on the data source and is not the Agent field.

The console administrator login token is valid for 12 hours; it has a different purpose than the Agent API Key.

5. Data source `/datasources`

The data source saves the business database connection and execution limit. An independent minimum-privileged database account must be created for AgentSQL, and superuser, owner, or root are prohibited.

Field	Constraints and default values
`id`	Required, cannot be modified after creation
`name`	display name
`db_type`	Select from the console registered types; `highgo` is the 9.0.10 Enterprise Edition PG mode qualified read-only type
`host`	Database host
`port`	1–65535; PostgreSQL default 5432, MySQL default 3306, HighGo default 5866
`database`	Database name
`username`	Run account
`password`	Required for new creation; leave blank for editing to retain the original value; API will never echo
`conn_limit`	Default 5, minimum 1
`stmt_timeout_ms`	Default 5000
`row_limit`	Default 1000

Field	Constraints and default values
`tls_mode`	PostgreSQL protocol path and specified Vastbase G100 Build 9 PG mode instance can choose `verify-full` (recommended) or `disable` (isolated testing only); SQL Server uses its independent TLS mode; other databases are judged according to the actual capabilities of the adapter
`tls_ca_file`	PostgreSQL `verify-full` is required and is the absolute path of the CA file that can be read by the gateway server.
`tls_server_name`	PostgreSQL `verify-full` Optional; verify the connection host name when not filled in, verify the name in the certificate when filled in

After saving, use "Test Connection" to verify. The MySQL driver turns off multi-statements; AgentSQL itself only accepts a single SQL.

When selecting `highgo`, use an independent account with minimum SELECT permissions. Currently, only the 9.0.10 Enterprise Edition PG schema and single-instance synthetic data environment are verified; the connection successively verifies the vendor version, schema, and read-only transaction status, and only accepts explicit schema, single-table direct columns, non-constant WHERE and LIMIT SELECTs. Metadata discovery is available, but sample collection and PostgreSQL B2/B5 capabilities are not open; see v1.7.0 acceptance record for details.

> **TLS Boundary:**v1.7.0 Verified `verify-full`, bad CA/hostname rejection and certificate rotation for PostgreSQL protocol path on KingbaseES V9R1C10 PG mode isolated single instance; v1.8.0 Completed standalone acceptance on specified Vastbase G100 Build 9 PG mode instance. The Vastbase server needs to enable `ssl=on`, and set `hostnossl ... reject` for read-only accounts that takes precedence over normal rules; the `require_ssl=on` of this build is not compatible with the current pgx protocol 3.0 startup package. Existing data source null mode retains the old behavior and is not considered TLS authenticated. Explicit TLS options for HighGo remain rejected. Since v3.3.0 only the pinned DM8 Pack3 build can explicitly select `verify-full` with an absolute gateway-local CA path and certificate name; bad certificates or encryption downgrade are denied; other databases cannot inherit these validation conclusions. For certificate deployment, see deployment guide and v1.8.0 design note.

The data source password is encrypted using `AGENTSQL_SECRET`. Losing or replacing SECRET will cause the existing password to be unable to be decrypted; database/control plane backup and SECRET must be saved in pairs, see Deployment Guide for details.

YashanDB client configuration

The vendor has verbally authorized redistribution of the YashanDB C client; no written authorization is stored in the repository. Current Linux packages and GHCR images include client 23.4.7.100. Formal release checks cover amd64/arm64 architecture, library inclusion and hashes. Source builds require the same client, `CGO_ENABLED=1`, the `yashan` build tag, Go driver v1.4.4 and the runtime library directory. See the deployment guide.

For a manually installed client, set the library directory before starting AgentSQL:

```
export LD_LIBRARY_PATH="/path/to/yashandb-client/lib${LD_LIBRARY_PATH:+:$LD_LIBRARY_PATH}"
./agentsql --version
./agentsql serve -c ./deploy/systemd/config.yaml
```

Linux archives include the client under `lib/yashandb`; the installer places it under `/usr/local/lib/agentsql/yashandb`. GHCR images use `/opt/yashandb-client/lib`. Verify the official inventory, signatures, independently trusted public key and target architecture before use.

YashanDB connections fail closed when the client is missing or cannot be loaded; other data sources do not depend on it. v2.1.0 validated a single x86_64 Enterprise Edition 23.4.1.109 instance with `COMPAT_VECTOR=yashan` and synthetic data: least-privilege HTTP MCP, narrow SELECT, R010 denial, phone-number masking, Inspector/Dify, persistent audit, SELECT revocation, and fail-closed recovery from business-database and separate-audit-database outages. See instance acceptance and formal release evidence.

TLS, client cancellation, SQL timeouts, active/standby operation, sustained production loads and aarch64 YashanDB servers remain unverified. Writes, transactions, EXPLAIN and unknown output remain denied; these results do not establish complete production support. For `libyascli.so: cannot open shared object file`, check the actual process environment for `LD_LIBRARY_PATH`. For undefined symbols, architecture errors or `YAS-02143`, check the Go driver, independent client, server version and CPU architecture.

6. Permission `/policies`

A policy binds an Agent to a data source. The default is denial without authorization; creating data sources and agents does not automatically grant any tables.

objects and actions

- Object type: `database`, `schema`, `table`, `column`.
- Action: `allow`, `deny`.
- Legal object names: `*`, `schema.*`, `table`, `schema.table`.

- Illegal object names: null value, leading and trailing spaces, more than one dot, empty fragment, `*.x`, `*.*`.

> **high-risk configuration warning: `*` and `schema.*` are both legal full table and full column authorizations, which will invalidate the column-level whitelist of the matching table. Don't think of them as generic examples or convenience defaults. Only `*.*` and `*.x` are illegal writing methods.**

The three object types `database`, `schema`, and `table` currently use the same set of table schema matching, and do not have independent database level or schema level parsing semantics.

Column allowlist

Column-level policies must satisfy:

- The object name is accurate to `table` or `schema.table`.
- No wildcards are allowed.
- Fill in at least one column.
- The action can only be `allow`.

Since v1.0.0, SELECT requests with column policies cannot fall back to table-only authorization. PostgreSQL requires an enabled, available B2 binder to validate column references in projections, predicates, JOINs and self-joins; when B2 is unavailable, requests fail closed before execution. HighGo and Vastbase cannot rely on column policies without equivalent column-binding evidence. MySQL requires its separate, restricted B2 subset to be enabled. The permissions page explains this boundary. Offline preflight can identify explicit denial or an unknown outcome; it does not establish authorization for actual execution.

v3.4.0 verified B2 column authorization only for restricted single-table `SELECT` through the PostgreSQL-protocol path on the pinned KingbaseES V9R1C10 PG-mode single instance. Check the kernel version, B2 readiness, and live allowed/denied columns before connection; views, joins, functions and statements without provable column lineage remain denied. B5 affected-row semantics did not pass on this kernel, so gateway write and session entry points are closed. The Vastbase G100 Build 9 B2 probe did not pass, and column policies remain fail closed. Neither result implies complete PostgreSQL support; see live acceptance.

`row_filter`

`row_filter` does not yet enforce rows. The current candidate rejects new policies containing this field. If a legacy row-filter policy matches, policy resolution and B2 column authorization fail closed instead of executing without the filter. This is not supported row-level security and must not be configured, advertised, or used in place of database RLS or security views.

Usage boundary of temporary read-only authorization `/task-read-grants` (v1.0.0)

An applicant with `query.execute` submits one Agent, one data source and one exact simple `SELECT`, valid for at most 24 hours. Another user in the same tenant with both `strategy.manage` and `query.execute` reviews and approves the SQL. The original applicant then claims the Key once. The list shows only the SQL hash and status; the original SQL is visible only to authorized reviewers in the approval details. Task Keys expose only `query` on both `/mcp` and `/mcp/2026-07-28`, with no write, approval or B5 tools. They only narrow existing Agent permissions; object policies, rules, masking and the database read-only account continue to apply.

The original SQL bytes when called must be the same as when issued; spaces, case, parameters or comment changes may cause the request to be rejected. Currently only a single entry, a conservative subset starting with `SELECT` and without parentheses, is accepted. Functions, subqueries, `WITH`, `EXPLAIN`, row locks and writes are not applicable. After the task is completed, it is canceled on the same page; new requests after the cancellation are rejected, and there is still a concurrency window for results already in progress. Task requests entering the SQL pipeline will record the authorization ID and issuer, and the task key or SQL original text will not be written to the traffic audit; the tool layer's preemptive rejection may not have a persistent record. For complete boundaries, see task-level temporary read-only authorization .

Application, approval and receipt of task authorization (v1.0.0)

Applicants need `query.execute` permission to submit a precise and simple `SELECT`, Agent, data source, expiration time of up to 24 hours and a result budget of up to 1–10 times; when the budget is omitted, the default is 1 time. The list displays SQL SHA-256, status, applicant, approver, and authorization ID, but does not display the original SQL text or task key. Another user with the same tenant who owns both `strategy.manage` and `query.execute` checks the SQL in the application details and approves or rejects it. After approval, the original applicant can only receive the task key once; copy and safely save the response, and refresh the page and cannot obtain it again. The legacy direct issuance interface returns HTTP 410.

Authorization list shows `已用 / 上限`; budget is atomically deducted after persistent audit but before results are delivered. The old version of empty budget authorization retains the original validity period and revocation constraints. The authorization ID can be brought to the "Audit" page to filter related management and SQL pipeline records. Revoked by the applicant or administrator when immediate deactivation is required; only the authorization ID, application ID, SQL hash, and operator are retained in the audit record. The original `AGENTSQL_SECRET` must be retained when upgrading or restoring the metadata database, otherwise the encrypted SQL in the unfinished application cannot be verified by the approver. For more conditions and restrictions, see task-level temporary read-only authorization .

MCP user identity delegation (v1.0.0 optional)

After enabling `mcp.delegation.require_human_token: true` and keeping the console open, people with `query.execute` can issue short-term delegation tokens by Agent ID on this page. The page is only displayed once; the token is valid for up to 30 minutes, is limited to HTTP MCP calls of the specified Agent, cannot access the console API, and cannot replace the Agent Key. The client must send the Agent Key and `X-AgentSQL-User-Authorization` request headers respectively. The token becomes invalid immediately after the user logs out, deactivates, or loses query permission; the stateful MCP session needs to be reinitialized after a user change. Issuance of write management audit, token is not delivered when audit is unavailable. Dual identity mode is currently not available for B5, `request_approval` and `execute_approved`. For configuration examples and request headers, see MCP Access Guide .

7. Rule `/rules`

Built-in rules directory

The severity of the rule page ranges from 1 (low) to 5 (severe); `dynamic` indicates the need for dynamic information such as EXPLAIN, index/table metadata or transaction running status.

ID	Title	database	Severity	dynamic
R001	Multiple statement stacking protection	Universal	5	No
R002	Unconditional batch write protection	Universal	5	No
R003	Read-only Agent write protection	Universal	5	No
R004	Large-scale scanning and approval	Universal	4	Yes
R005	Large result set limit reminder	Universal	3	Yes
R006	Comment and unknown statement injection protection	Universal	5	No
R007	Dangerous function blacklist	Universal	5	No
R008	Request rate and concurrency limits	Universal	5	No
R009	SQL complexity limit	Universal	4	No
R010	Unauthorized table access protection	Universal	5	No

ID	Title	database	Severity	dynamic
R101	High-risk DROP protection	PostgreSQL	5	No
R102	Sustained lock operation approval	PostgreSQL	4	No
R103	Management and file function protection	PostgreSQL	5	No
R104	COPY External program protection	PostgreSQL	5	No
R105	No index batch write approval	PostgreSQL	4	Yes
R106	Large table structure change approval	PostgreSQL	4	Yes
R107	Long transaction and idle transaction alarm	PostgreSQL	3	Yes
R201	MySQL file read and write protection	MySQL	5	No
R202	MySQL dangerous batch write approval	MySQL	4	No
R203	MySQL high-risk management commands	MySQL	5	No
R204	MySQL large transaction approval	MySQL	4	Yes

Title, severity, and `dynamic` attributes are subject to `web/src/constants/ruleMeta.ts`.

Current editability

- The built-in rule only reads `enabled` when running. The console can only be enabled or disabled, and the modification will take effect on the next real request.
- Built-in rules cannot be deleted.
- Editing of runtime thresholds, actions, or risk scores is not currently provided; "Restore Default" simply re-enables the rule.
- Custom rules support CRUD, and the fields are `id`, `db_type`, `title`, `risk_level`, `pattern_type=ast`, `definition`, and `enabled`.
- **custom rule ID will not be assembled into the execution engine.** This is just to configure the recording capability and currently has no actual interception effect.

Thresholds and dynamic dependencies

Internal threshold keys include:

- Universal: `max_scan_rows`, `row_limit`, `qps_per_agent`, `max_conns_per_datasource`, `max_sql_length`, `max_nesting_depth`, `max_union_count`.
- PostgreSQL: `large_table_rows`、`long_transaction_ms`、`idle_transaction_ms`。
- MySQL: `mysql_transaction_age_ms`、`mysql_transaction_affected_rows`。

These thresholds belong to the configuration/runtime layer and are not editable on the rules page. `defaults`, which currently exposes strict YAML, only exposes `row_limit`, `qps_per_agent`, `max_conns_per_datasource`, and execution timeout `statement_timeout_ms`; although the other names are engine internal threshold keys, `internal/config/config.go` has not yet exposed the corresponding YAML fields. Do not directly add unexposed keys to the configuration, otherwise strict parsing will refuse to start; the public configuration entry needs to be confirmed by the master before adding it.

Dynamic rule dependencies: R004 and R005 depend on EXPLAIN; R105 depends on index metadata; R106 depends on the number of table rows; R107 and R204 depend on transaction running status.

> **R005 Special note: The** ordinary production pipeline uses EXPLAIN in the dynamic stage to estimate the result size; the unbounded query that exceeds the data source `row_limit` is estimated to hit R005. The common execution path and the PostgreSQL column-level authorization path also supplement R005 hits when the actual `row_limit` is truncated and written to the same persistent audit. EXPLAIN or remain fail-closed when the result size fact cannot be reliably obtained.

1–5 of the page directory is the rule severity. There are also decision-making priorities of `1=deny`, `2=approve`, `3=warn`, and `4=info` inside the pipeline; the two are not the same unified score and cannot be mixed.

8. Approval `/approvals`

Common SQL approval statuses include `pending`, `approved`, `claimed`, `completed`, `failed`, `unknown`, `rejected`, `expired`. New applications are valid for 24 hours; after expiration, they cannot be claimed even if the status still shows `approved`. The old application is missing the seal field and is not executable.

Administrators can only decide on the `pending` order. The decision request body is:

```
{
  "decision": "approve",
  "comment": "可选"
}
```

`decision` only allows `approve` or `reject`, and the status becomes `approved` or `rejected` after saving.

> **Core semantics:** Administrator approval will not automatically execute SQL. The original Agent can call `execute_approved(approval_id)` once within the validity period; the identity, data source, current policy and rules will be re-verified during execution. Manual approval cannot bypass deny rules. Claimed records may not be automatically retried.

There are two paths to the tool layer:

- When `execute_write` returns `decision:"approve"`, the system has automatically created the `pending` order and returned `approval_id`. Do not call `request_approval` again.
- `request_approval` is used only when the caller actively wants to submit it for review first; it also only creates orders but does not execute them.

The `reason` passed in by the caller is saved as the application description; the rule evaluation reasons and approval notes are saved separately. Normal SQL approval currently only accepts literal SQL and does not provide parameter array input.

9. masking `/mask-rules`

masking is performed at the query result layer by column and does not modify the original value of the database. There are currently four runnable algorithms for preserving readable fragments, generating associable fingerprints, integer blocking, and preserving coarse-grained distributions:

algorithm	Support type	Keep original text fragments	Can be equivalently related	Key required	Typical scenario
Coding `mask`	Mobile phone number `phone`, email address `email`, ID card `idcard`, bank card `bankcard`, IP address `ip`, date of birth `birthdate` Category six	Yes, partially masked by format	No	No	Columns that need to retain a small amount of readable information such as customer service verification and operation and maintenance troubleshooting
Hash fingerprint `hash`	Category nine	No; v1 outputs 34 characters `h.<32hex>`, v2+ outputs `h.<版本>.<32hex>`	Only under the same version and the same key, the same string bytes will get the same fingerprint	Yes; single key or strictly multi-version manifest, at least 32 bytes per material	Downstream equivalent correlation, deduplication and grouping within the same version window

algorithm	Support type	Keep original text fragments	Can be equivalently related	Key required	Typical scenario
Block `block`	Category nine	No; each non-null value is fixedly output `***`, and the original text length is not retained.	No	No	The most sensitive columns such as complete ID, password/key, salary, health or illegal details
Bucketing/Truncation `range`	Value `number`, date `date`	No; output numerical range or year/quarter/month	Yes; values in the same bucket can be associated	No; use type-specific parameters	Age group, amount range, and date of birth statistics by year/quarter/month

When selecting an algorithm, `mask` is used first for readability checks, `hash` is used when stable association is required and the original text cannot be returned, `block` is used when no non-empty original value fragments should be returned, and `range` is used when coarse-grained distribution of values or dates needs to be observed. `range` is not anonymized: values in the same bucket can still be associated, and sensitive scenarios should use `block` instead or be included in the approval.

Type and algorithm capability matrix

Sensitive type	`mask`	`hash`	`block`	`range`
Mobile phone number `phone`	✅	✅	✅	❌
Email `email`	✅	✅	✅	❌
ID card `idcard`	✅	✅	✅	❌
Bank card `bankcard`	✅	✅	✅	❌
IP address `ip`	✅	✅	✅	❌
Date of birth `birthdate`	✅	✅	✅	❌
Universal sensitivity value `generic`	❌	✅	✅	❌
Value `number`	❌	✅	✅	✅
Date `date`	❌	✅	✅	✅

`number` / `date` Represents column semantics, `range` Only open to these two categories. `birthdate` Still using birthday only `mask`, cannot be selected even if the original value can be parsed as a date `range`; Six categories: mobile phone, email, ID card, bank card, IP, birthday and `generic` Can't choose either `range`。 `hash` / `block` Can handle nine categories, the administrator can `range`、`hash`、`block` There is no need to switch between `number` / `date` Change the label to `generic`。

`generic` does not rely on format recognition such as mobile phone number, email address, etc., and can be used as `hash` or `block` for the complete original string of the hit column; it cannot be used with `mask`, and `mask+generic` is an illegal combination. With the exception of null sentinels, there is no trim before hashing, and no case, Unicode, date, time zone, or decimal normalization, so representing different values will result in different fingerprints. `block` also does not recognize the format and has no parameters. Any non-null value (including malformed values, values with leading and trailing spaces, and values whose original value is exactly `***`) will output a fixed `***`.

> `***` is the normal success output of `block`; `[REDACTED]` is the fail-closed cover when `mask` or `range` cannot safely handle non-null values. The literals and algorithmic semantics of the two are different. The rule management interface and rule configuration can identify the algorithm through `algo=block`, but query audit, SSE, and notification do not record the specific masking algorithm, and the result projection itself cannot reliably deduce the cause of `***`; this version does not extend the audit protocol.

Empty strings, trimmed strings, and `NULL` / `<nil>` sentinels that ignore case are returned as is and are not counted in `MaskedCells`; therefore `block` will expose the result unit to be empty/NULL. SQL NULL and database empty strings have been stringified into `""` before entering masking, and the current result model cannot distinguish them.

Sensitive column discovery always only identifies the first six categories and generates `algo=mask` disabled draft; it will not output `hash`、`block` or `range` rules and will not be found `generic`、`number` or `date`。`number` / `date` of `range` Rules need to be created manually on the "Anonymization Rules" page; see the complete description of the discovery process.Sensitive Column Discovery Guide。

There are four items in the algorithm drop-down list of the console: "Mask", "Hash", "Block" and "Bucketing/Truncation (Range)". When configuring `block`:

1. Enter "masking Rules" and create or edit a rule, and select "Block" for the algorithm.
2. Select any support type among the nine categories, fill in the target result column and scope; the page description and examples are fixed to display `***`, and the hash key warning is not displayed.
3. Save and enable the rule; `block` does not require a key or other parameters, does not participate in hash key startup fail-fast, and will not return `HASH_REDACTION_UNAVAILABLE` / HTTP `503`.

When the hash key is not configured, `mask`, `block`, and `range` can be created, enabled, and run normally; only `hash` is subject to key access control. At this time, `hash` can be saved as a deactivation rule, but the new rules or enable operations enabled by default will be rejected. The server response corresponding to the page is HTTP `503`, `HASH_REDACTION_UNAVAILABLE`, and the rules will not be written. Configure at least a 32-byte key via `AGENTSQL_REDACTION_HASH_KEY` or `redaction.hash_key` and reboot before enabling `hash`. When the type/algorithm combination is illegal, the server returns HTTP `422`, `INVALID_MASK_RULE`.

Three scopes and selection suggestions

Scope	`schema_name`	`table_name`	When to use
Global column rules	`""`	`""`	When columns with the same name need to be protected uniformly across all tables; for example, `phone` in any query results should be coded
Table.Column Rules	`""`	real table name	Only protect the columns of a certain table; **recommends** by default. This is the table-only draft generated by sensitive discovery.
Schema.Table.Column Rules	real schema name	real table name	In scenarios such as PostgreSQL, when tables with the same name exist across schemas and must be distinguished

These three files are the relational scope of column rules and are orthogonal to the data source scope. When the data source is not selected, the rule takes effect on all data sources; when the data source is selected, the rule only takes effect on that data source. `schema_name` cannot be filled in separately from `table_name`.

Configure global column rules

1. Enter "masking Rules" and select "New Rule".
2. Select the data source as needed; leave both "Schema" and "Table Name" blank.
3. Fill in the column name, select the sensitivity type, algorithm and enabled status and save.

configuration table. Column rules

1. Enter "Anonymization Rules", select "New Rule" and select the data source.
2. Leave "Schema" blank, and fill in "Table Name" with the real table name, such as `customers`.
3. Fill in the column name, sensitive type and algorithm; manually create a new one and enable it directly. If you find the generated draft, you should review it before enabling it.

Configuration mode.Table.Column rules

1. Enter "Anonymization Rules", select "New Rule" and select the data source.
2. Fill in the "Schema" and "Table Name" at the same time, such as `tenant_a` and `customers`.
3. Fill in the column name, sensitivity type, algorithm and enabled status and save. Only explicitly written and exact matching schema can hit this file in the query.

PostgreSQL's table and column rules will match any pattern when the schema is left blank, and `public` will not be automatically filled in. MySQL table names are compared with case-sensitive exact strings, and column names are not case-sensitive; leaving the schema blank means the table-only semantics of the current library. PostgreSQL's `customers` and `"Customers"` are also two different table keys.

Column names will remove leading and trailing whitespace and single-layer quotation marks, backticks or square brackets, and then unify lowercase; table names and schema will retain the original case and compare accurately. There can only be one rule for the same physical scope and normalized column name, and the disabled draft also occupies this unique key.

Configure `range` value bucketing

Numerical rules select sensitivity type `number` and algorithm `range`. The console form will display two integer inputs: "Bucket Width" and "Offset":

- `range_bucket_width`: bucket width, required, must be an integer of `1..1,000,000,000`.
- `range_bucket_offset`: offset, can be left blank, defaults to `0`, must be an integer of `-1,000,000,000..1,000,000,000`.

Intuitively, the offset determines the starting point of the bucket boundary, and the bucket width determines how many values are covered by each segment. The formulas are `lower = floor((v - offset) / width) * width + offset`, `upper = lower + width`, where `floor` rounds to negative infinity. As a result, when using `[lower,upper)` with left closed and right open: `width=10` and `offset=0`, `42` outputs `[40,50)` and `-3` outputs `[-10,0)`.

Configure `range` date truncation

The date rule selects sensitivity type `date` and algorithm `range`, without using bucket width or offset. `range_granularity` can be left blank and defaults to `year`; the explicit value can only be year `year`, quarter `quarter` or month `month`. For example, `1990-08-21` outputs `1990` on a yearly basis, `1990Q3` on a quarterly basis, and `1990-08` on a monthly basis.

Common date writings such as full date, datetime, RFC3339, and `YYYY` / `YYYY-MM` can be processed; when the input precision is not sufficient to generate the selected granularity, the date is illegal or unrecognizable, `[REDACTED]` will be safely displayed.

API fields and full update semantics

Use `POST /api/v1/mask_rules` to create three new rules. Here is a global column rule:

```
{
  "id": "global-phone-mask",
  "datasource_id": "ds-pg",
  "schema_name": "",
  "table_name": "",
  "column_name": "phone",
  "sensitive_type": "phone",
  "algo": "mask",
  "enabled": true
}
```

Leave `schema_name` of the table.column rule blank, for example, only protect `customers.phone`:

```
{
  "id": "customers-phone-mask",
  "datasource_id": "ds-mysql",
  "schema_name": "",
  "table_name": "customers",
  "column_name": "phone",
  "sensitive_type": "phone",
  "algo": "mask",
  "enabled": true
}
```

Schema.Table.Column rules fill in both scope fields at the same time:

```
{
  "id": "tenant-a-customers-phone",
  "datasource_id": "ds-pg",
  "schema_name": "tenant_a",
  "table_name": "customers",
  "column_name": "phone",
  "sensitive_type": "phone",
  "algo": "mask",
  "enabled": true
}
```

Updated to use `PUT /api/v1/mask_rules/{id}`. PUT is a complete update, and `datasource_id`, `schema_name`, `table_name`, `column_name`, `sensitive_type`, `algo` and `enabled` that you want to keep should be sent each time; `{id}` in the path is the target rule ID, and `id` does not need to be repeated in the request body. For example:

```
PUT /api/v1/mask_rules/customers-phone-mask
Content-Type: application/json
```

```
{
  "datasource_id": "ds-mysql",
  "schema_name": "",
  "table_name": "customers",
  "column_name": "phone",
  "sensitive_type": "phone",
  "algo": "block",
  "enabled": false
}
```

For the two scope fields `schema_name` and `table_name`, the default, JSON `null` and empty strings are treated as null values; to avoid accidentally changing the scope during PUT, it is recommended to always pass the empty string explicitly. `datasource_id` default, `null` or empty string indicates all data sources.

In the masking rule creation/update JSON, `range` additionally uses `range_bucket_width`, `range_bucket_offset`, and `range_granularity`. Numeric rules cannot carry `range_granularity`, and date rules cannot carry `range_bucket_width` or `range_bucket_offset`. A minimum value rule request body is as follows:

```
{
  "id": "amount-range",
  "schema_name": "",
  "table_name": "orders",
  "column_name": "amount",
  "sensitive_type": "number",
  "algo": "range",
  "range_bucket_width": 10,
  "range_bucket_offset": 0,
  "enabled": true
}
```

A minimum date rule request body is as follows:

```
{
  "id": "birth-date-range",
  "schema_name": "",
  "table_name": "customers",
  "column_name": "birth_date",
  "sensitive_type": "date",
  "algo": "range",
  "range_granularity": "month",
  "enabled": true
}
```

Rule creation and updates adopt full PUT semantics: missing `range` parameters in the request will be written back to `NULL`; when switching from `range` to other algorithms, or switching between `number` / `date`, inapplicable parameters will be cleared. The numeric offset `0` is a legal value and will be retained by the API; when the offset is omitted, it is normalized to `0`.

Exact matching and fail-closed security

When the top-level directly projected column can be uniquely attributed to a physical relationship, the table-level rules are accurately matched according to the source table and source column; when there are schema precise rules, use the precise rules first, and then try the table.column rules of `schema_name=""`. A naked column can only be uniquely owned if the top-level `FROM` scope has exactly one visible source, and that source is a physical relationship. After the table is aliased, it must be qualified by the alias; the original table name has been obscured.

JOIN bare columns, `SELECT *`, and CTE outer columns may not determine the physical origin. If an unparsed result column hits a table-level rule, and the protected table appears in the possible relationship set of the entire statement, AgentSQL will not return the original text due to "unable to prove the source", nor will it reject the entire query; it will perform a non-closable fail-closed safety policy on the column and block all non-empty cells as `***`. This cover does not use the `mask` / `hash` / `range` business algorithm configured in a certain table because the source has not been determined.

The preferred way to eliminate fratricide is to refine the SQL: qualify columns with table names or table aliases, and avoid using `SELECT *` to project protected columns. If a column with the same name should be protected uniformly across all tables, the column can be configured as a global column rule.

PostgreSQL JOIN example (assuming `orders` does not have `phone` column, the SQL is legal in the database, but the parser does not query the schema to guess the column ownership):

```
-- Before modification: Naked columns from multiple sources cannot be uniquely attributed and may
trigger *** cover-up
SELECT phone
FROM public.customers AS c
JOIN public.orders AS o ON o.customer_id = c.id;

-- After modification: qualified by table alias, accurately attributed to public.customers.phone
SELECT c.phone
FROM public.customers AS c
JOIN public.orders AS o ON o.customer_id = c.id;
```

MySQL `SELECT *` example:

```
-- Before modification: star does not produce a top-level direct projection that can be accurately
attributed
SELECT *
FROM customers;

-- After modification: Explicitly project and qualify protected columns
SELECT customers.id, customers.full_name, customers.phone
FROM customers;
```

PostgreSQL CTE example:

```
-- Before modification: The outer layer only sees the non-physical source x, and the phone may trigger
***.
WITH x AS (
  SELECT phone FROM public.customers
)
SELECT phone FROM x;

-- After modification: directly limit the physical relationship
SELECT customers.phone
FROM public.customers;
```

In the MCP `query` response, the actual changed column of a non-empty cell will be recorded as `redact.unresolved_scoped_columns`. Currently, the Playground REST response only projects `touched_columns` and `masked_cells`, and neither the Playground nor the audit UI displays the `unresolved_scoped_columns` count. When you see the result `***` on these pages, first check whether the query uses JOIN bare columns, `SELECT *` or CTE outer columns, and then refine the SQL as described above.

Sensitive discovery workflow

1. Enter sensitive discovery on the "Data Source" page, select 1-20 visible basic tables at a time, and enable controlled sample scans of 1-20 rows per table (default 10 rows) as needed.

2. Reviewed the candidate columns, type, confidence level and evidence count; found that only six categories were recognized: mobile phone, email, ID card, bank card, IP and date of birth.
3. Apply candidate after selection. The system identifies candidates by physical origin `(datasource, schema, table, column)` and generates table-only drafts of `schema_name=""`, `table_name=<真实表名>`, `algo=mask`, `enabled=false`.
4. Go to the "masking Rules" page, check the table name, column name, sensitivity type and algorithm one by one, and confirm before activating. Drafts are disabled by default and do not affect queries immediately after they are created.

Columns with the same name in different tables will generate multiple independent table-only drafts. If a candidate column is already covered by an enabled global column rule, the application result will be marked `CoveredByGlobal` and the new creation will be skipped; disabled global drafts will not block new table-only drafts.

Verification and conflict error reporting

HTTP / error code	meaning	Processing method
`422 INVALID_MASK_RULE`	`schema_name` is not empty but `table_name` is empty, the scope identifier is illegal, or the type/algorithm/parameter combination is illegal	Fill in the schema and table at the same time, remove leading and trailing blanks, wildcards, control characters or `.` in a single field, and check the type and algorithm
`409 MASK_RULE_CONFLICT`	The same physical key `(datasource_scope,schema_name,table_name,归一化 column_name)` already exists; the disabled row is also considered a conflict	Edit existing rules or delete one; deactivating alone does not release the physical unique key
`409 MASK_RULE_SCOPE_CONFLICT`	Within the valid scope of the same datasource, the algorithm of enabling global column rules with the same column name is inconsistent with the table-level rule algorithm.	Align the algorithms or delete one; you can first deactivate the rule to be adjusted, then modify the other one and re-enable it in sequence

`MASK_RULE_SCOPE_CONFLICT` only compares enabled rules, disabled draft exemptions. For the `range` rule, "algorithm consistency" also requires that the sensitive types are the same; the numerical rule compares the bucket width and the normalized offset, and the date rule compares the normalized truncation granularity.

`range` Behavior and Boundaries

- `range` No key configuration is required; services can be created, enabled, and run even without hash keys.
- Null value sentinels such as empty strings, empty strings after trim, and case-ignoring `NULL` / `<nil>` are returned as they are and are not counted in `MaskedCells`; other non-null values are counted regardless of successful generalization or output of `[REDACTED]`.

- Generalization is done in memory after the query results are retrieved by the gateway, and does not overwrite database data, reduce database queries, or prevent the database from executing `WHERE`, `JOIN`, or `GROUP BY` with the original values.
- The generalization result is a string, which is not guaranteed to be compatible with the original value or date schema. The consumer should process it as text.
- `range` does not promise k-anonymity; values in the same bucket can still be associated, and the value range, date period and null value status are still visible. Small buckets, monthly granularity, small samples, multiple offsets or auxiliary queries may increase the risk of re-identification, and `block` or approval should be used in sensitive scenarios.

Deterministic hash fingerprints expose equality relationships and frequencies, and may also form correlation traces between libraries or tables that share the same key. Irreversible does not equal anonymity: key leaks, hash oracles, low-entropy dictionary guessing or auxiliary data may still re-identify the original value. Environments or tenants that should not be related to each other should use different keys. For complete key management and rotation restrictions seeDeployment Guide "masked Hash Key Management"。

masking key rotation and equivalence boundary

v0.4 supports multi-version key manifest, registration reconciliation, historical fingerprint verification and **restart plan switching**. It does not support hot loading, blue and green mixed version services, automatic recalculation or automatic double writing. The fingerprint of v1 looks like `h.<32 位小写十六进制>`; the fingerprint of v2 and later looks like `h.<版本>.<32 位小写十六进制>`.

The standard switching window is: Announce maintenance → Stop or reject new writes → Wait for the end of in-flight requests from the old instance → Switch the target version to active in the registration table → Modify the manifest `active_version` used by all instances → Restart all instances → Perform strong reconciliation and confirm `/readyz` returns HTTP 200 → Restore traffic. The old instance must be completely exited before the new instance can serve; the `active_version`, corresponding key commitment, and revision of all instances in the same service group must be consistent. v0.4 does not provide a cross-instance coordinator and cannot allow different active versions to serve external parties at the same time.

Rotation will cause **equivalent splitting**: the same plaintext has different fingerprints under v1 and v2. Downstreams that rely on fingerprint columns for JOIN, deduplication, cache keys, CDC comparison, or report association must rehash in the maintenance window, or the downstream must implement dual version transition by itself; v0.4 does not provide automation. When long-term stable association across rotations is required, stable tokens or dual-version tokens should be designed. It cannot be assumed that the current fingerprint is equal across versions.

The rollback is performed in the reverse order: stop writing and draining again → bring the old version of CAS back to active → restore the old `active_version` and restart all instances → perform strong reconciliation and verification with `/readyz` → restore traffic. As long as the old key material has not changed, the same plaintext will be restored to the same old fingerprint byte by byte; the new version of historical fingerprints generated during the switch can still be verified with the corresponding legacy key. Confirm all historical dependencies and do not retire the key before the end of the rollback window; retirement is the final state, and the management verification portal will no longer accept this version.

Empty strings, strings that are empty after trim, and `NULL` / `<nil>` that ignore case are the null sentinels of the existing result model: they are returned as they are without generating fingerprints; SQL NULL and database empty strings are both stringified into `""` before entering masking, making it indistinguishable. Therefore, the literal `"NULL"` will not be hashed, and it cannot be treated as an ordinary business string to participate in equivalence association.

`block` does not output the original value characters, length or equivalence relationship, but still retains the row and column shape, number of rows, column names and whether there are results in the result set, and leaks the above-mentioned null value status; this is consistent with the result layer leakage level of `mask` / `hash` / `range`, and is not anonymized or "zero information". The four algorithms are all processed after the database is executed and the results are returned to AgentSQL, which does not reduce database reads, nor prevent the database side from executing `WHERE`, `JOIN` or `GROUP BY` according to the original values. The columns marked by `block` are opaque strings; fixed `***` does not guarantee compatibility with downstream schemas such as values, dates, JSON, and UUIDs, and consumers should handle them as opaque text.

Known limitations and recommendations

- PostgreSQL/MySQL authenticated function wrap, alias, aggregation, `CAST`, CTE, and set operations track protected columns; when the source of security cannot be uniquely proven and a protected column may be involved, the result is fixedly blocked to `***`. `CONCAT(phone,'') AS x`, `lower(phone) AS y`, etc. have been included in negative use cases. New database functions, view internal semantics, or multi-step statistical queries may still be beyond the scope of the proof and cannot be used to claim complete DLP.
- `RETURNING` passes the write barrier and only returns `RowCount` without entering query result masking. It should not be used as a channel to return masked data.
- View reference names can be matched according to configured rules; the origin of truth within unknown views or functions is not automatically proven by AgentSQL. Database least privileges, security views, and column-level authorization still need to be independently configured and verified.

10. Configuration reference

Configuration files use strict YAML: unknown fields will cause startup to fail. `agentsqlctl init-config` can be used to generate the basic configuration. For production details, see deployment guide .

`server`

Field	Description
`http_listen`	HTTP listening address, such as `127.0.0.1:7780`
`console_enabled`	Whether to enable the management console; defaults to `true` when not explicitly configured
`event_stream`	Whether to enable SSE; default `true` when not explicitly configured
`event_stream_max_connections`	Maximum number of SSE connections, default 100, valid range 1–1000

`store`

`store.sqlite_path` is the abbreviation of SQLite and cannot be used together with `store.metadata`.

Field	Description
`sqlite_path`	Default SQLite control plane path
`auto_migrate`	The default is `true`; in `false`, only the schema version is verified and DDL is not executed.
`metadata.driver`	`sqlite` or `postgres`, default `sqlite`
`metadata.sqlite_path`	metadata path when using SQLite
`metadata.dsn`	metadata DSN when using PostgreSQL
`metadata.max_open_conns`	Default 10
`metadata.max_idle_conns`	Default 5, cannot exceed open
`metadata.conn_max_lifetime`	Default `30m`, use Go duration string
`audit.separate`	Whether to use independent audit storage, default `false`
`audit.driver`	Standalone audit currently only supports `postgres`
`audit.dsn`	Standalone audit PostgreSQL DSN
`audit.max_open_conns`	Default 10
`audit.max_idle_conns`	Default 5, cannot exceed open

`defaults`

Field	Description
`statement_timeout_ms`	Global default statement timeout, cannot be negative; template value 5000
`row_limit`	Global default maximum number of result rows; template value 1000
`max_conns_per_datasource`	Global default data source connection/concurrency limit; template value 5
`qps_per_agent`	Global default QPS per Agent and burst limit; template value 20

The rules also recognize threshold keys such as `max_scan_rows`, `max_sql_length`, `max_nesting_depth`, `max_union_count`, `large_table_rows`, `long_transaction_ms`, `idle_transaction_ms`, `mysql_transaction_age_ms`, `mysql_transaction_affected_rows`, etc., but the currently public `Config` structure does not correspond YAML field, the status is **to be confirmed/unpublished configuration entry**. They cannot be modified in the rules page, and do not include strict YAML directly.

`theme`

Field	Description
`default`	`light` or `dark`

`demo`

Field	Description
`enabled`	Whether to enable dedicated Live Demo
`banner`	Demo mandatory banner; empty value uses fixed default copy
`allowed_datasource_ids`	When enabled, it must be exactly `ds-demo-pg` and `ds-demo-mysql`
`qps_per_agent`	Demo QPS, default 2, valid range 1–5
`anchor_date`	Optional, `YYYY-MM-DD` real date

Do not enable `demo` in the production environment.

`redaction`

Field	Description
`hash_key`	Optional HMAC hash fingerprint key, at least 32 bytes when not empty; `mask`, `block` and `range` can still run when empty or not configured, enabled `hash` will cause fail-fast to start

Field	Description
`hash_keys.active_version`	Active version of multi-version manifest, range 1–9999; must match exactly one `keys[].id`, the assembly needs to be restarted after the change
`hash_keys.revision`	Optional non-secret configuration revision; multiple instances should be consistent, and when omitted, it is derived from the active version and commitments
`hash_keys.keys[]`	Versioned key list; each item can only use one of `key_file` or canonical `key_b64`, the material is at least 32 bytes and meets the diversity check

`hash_key` is a compatible single-key configuration; it cannot be mixed with `hash_keys` or `AGENTSQL_REDACTION_HASH_KEYS_FILE`. It is recommended to point `AGENTSQL_REDACTION_HASH_KEYS_FILE` to a permission-protected absolute path manifest. The key may not be reused or derived from `AGENTSQL_SECRET`, nor may it enter Git, mirrors, logs, audits, or configuration echoes.

environment variables

variable	Purpose
`AGENTSQL_SECRET`	Required, exactly 32 bytes; encrypts the data source password and derives the administrator token signing key
`AGENTSQL_REDACTION_HASH_KEY`	Optional, at least 32 bytes; only used for `hash` to generate irreversible HMAC fingerprints, YAML (including empty string) will be overwritten if present; `mask`, `block` and `range` can still be used when not configured
`AGENTSQL_REDACTION_HASH_KEYS_FILE`	Optional, the absolute path of the multi-version key manifest; mutually exclusive with inline `hash_keys`, `hash_key`, `AGENTSQL_REDACTION_HASH_KEY`
`AGENTSQL_ADMIN_USER`	Administrator username, default `admin`
`AGENTSQL_ADMIN_PASSWORD`	Required when the console is enabled, at least 12 characters and cannot use weak passwords/public examples
`AGENTSQL_API_KEY`	`--api-key` replacement for stdio; use environment variables first to avoid entering process parameters
`AGENTSQL_STORE_METADATA_DSN`	Overrides YAML metadata DSN if present, including explicit nulls
`AGENTSQL_STORE_AUDIT_DSN`	Overrides YAML audit DSN if present, including explicit nulls
`AGENTSQL_INSECURE`	Only local testing allows exposing sample credentials; does not close authentication, security rules, or fail-closed
`AGENTSQL_DEMO`	Demo mode is enabled when the value is `1`
`AGENTSQL_DEMO_RO_KEY`	Demo read-only Agent Key
`AGENTSQL_DEMO_DML_KEY`	Demo DML intercept Agent Key
`LOG_LEVEL`	Log level
`LOG_FORMAT`	`json` (default) or `console`

`AGENTSQL_DOWNLOAD_BASE` is only for installer download source coverage; `AGENTSQL_BENCH_DSN` is for benchmark only. Neither are production run configuration items.

11. Operation and maintenance

`agentsqlctl` public subcommand

subcommand	Purpose
`version`	output version
`init-config`	Generate default configuration
`check-config`	Strictly verify configuration and do not create SQLite directory
`migrate`	Initialize or upgrade metadata/audit schema; read-only available starting from v0.7.0 `--check`
`migrate-sqlite-to-postgres`	Migrating a composite SQLite control plane to PostgreSQL
`health`	Check HTTP health or check control plane storage directly
`demo-seed`	Only demo environment maintenance, using fixed manifest to write Demo data
`redaction-key`	Manage masking key registration, verification and audit outbox relay; subcommands are `reconcile`, `registry-mark-active`, `registry-mark-retired`, `verify`, `relay`
`chain`	Manage the audit integrity chain; subcommands are `status`, `verify`, `provision`

There is no `run-check-config-and-health` subcommand; the string is just the step copy in the migration output.

`health`

The two modes are mutually exclusive:

- `--url`: Checks `http://127.0.0.1:7780/healthz` by default, requires HTTP 200 and JSON `status:"ok"`.
- `--config` / `-c`: Check metadata/audit storage directly, output `metadata_driver`, `audit_driver`, `audit_separate`.
- `--timeout`: Default 3 seconds.

```
# Applicable premise: the service is already running on this machine
agentsqlctl health --url http://127.0.0.1:7780/healthz --timeout 3s
```

```
# Applicable prerequisite: running user-readable configuration, SECRET and corresponding control plane storage
agentsqlctl health --config /etc/agentsql/config.yaml --timeout 3s
```

Upgrades, rollbacks and backups

The upgrader will perform pre-check, backup and failure rollback; see DEPLOY.md for specific commands, service status check and offline package process. Don't skip external backup confirmation.

v1.0.0 Use `agentsqlctl migrate --check --config /etc/agentsql/config.yaml` to read the configured metadata/audit schema version before the migration window. The exit code `0` indicates all current versions, `2` indicates pending migration, and `1` indicates configuration, connection or migration history errors. `--check` does not execute DDL, does not create the missing SQLite library, and does not verify the entire database structure or cross-database consistency; the new version of the service cannot be started directly in the pending migration state. After the complete set of backups is completed and writing is stopped, the migration owner runs `agentsqlctl migrate`, and then uses the running account to check that `--check` is `0`. The PostgreSQL branch database must back up metadata, audit and corresponding `AGENTSQL_SECRET` in pairs; see the deployment guide and current version operation and maintenance manual for the detailed sequence. The v0.6.0 installation package does not contain `migrate --check`.

The iron rule of backup: The **control plane database and the `AGENTSQL_SECRET` that generates its data source password ciphertext must be backed up and restored in pairs.** Neither database only nor SECRET only can recover an existing data source password.

SQLite → PostgreSQL control plane migration key points: stop old service writing; back up SQLite and SECRET; prepare PostgreSQL metadata/audit target; run `migrate-sqlite-to-postgres` and check row count/hash; switch running account DSN; retain original SECRET; run `check-config` and `health`; verify `/readyz`, read, reject, approve and new audit ID. See DEPLOY.md for complete steps.

Performance caliber

The only external benchmark caliber currently archived is: memory fake executor, 20 workers, 200,000 samples, excluding database network and execution time, gateway CPU path P99≈1.81ms. It's not end-to-end query latency; the real time consuming includes connections, EXPLAIN, database execution, and audit storage.

12. Current version capability boundaries

- Denied without authorization by default; Agent × data source policy must be created explicitly.

- Pre-rejections such as missing `reason`, using `query` to write statements, SQL parsing failure at the tool layer, etc. occur before entering the pipeline, and there is no guarantee that every failure will be audited.
- Custom rules only record configuration and do not enter the execution engine; `row_filter` is not executed, new policies are rejected by the current candidate, and matching legacy policies fail closed.
- The data source does not have fields for SSL mode, CA, client certificate, connection timeout, or DSN additional parameters.
- R005 estimates the alarm by EXPLAIN during the normal production dynamic phase, and supplements the alarm when the common execution and PostgreSQL column-level authorization paths actually trigger `row_limit` truncation.
- Approval will not automatically execute SQL; the original Agent can execute a single seal request based on the unexpired approval document number, and re-verify the current permissions and rules during execution.
- masking is a result layer matching by column name to a uniquely resolvable physical source, not a full DLP. Global columns, table.column, schema.table.column rules are all supported; unresolved projections such as JOIN naked columns, `SELECT *` and CTE outer layers will fail-closed and fixedly blocked to `***` when they may involve protected tables. Function wrapping and renaming, aggregation, `CAST`, `UNION` and view renaming may still not be traceable; JOIN / self-join authorization is still only to the table level; `*` / `schema.*` is the entire table and full column authorization.
- `RETURNING` passes the write barrier and only returns `RowCount` without result masking.
- `hash` only processes the result value returned by the business database and does not participate in JOIN/WHERE in the database; the deterministic fingerprint leaks the equality relationship and frequency. v0.4 supports versioned history verification and restart plan switching, but does not provide hot loading, double writing, online rotation or automatic recalculation; cross-versions are not the same as plain text fingerprints.
- `block` also only processes the result value returned from the database; it uses the opaque string `***` to prevent the result unit from sending the original text, but retains the result shape, row number, column name, result existence and null value status, and does not guarantee downstream schema compatibility. It is not anonymizing, nor does it restrict the database side to filter, correlate, or group by original value.
- `range` is only open to `number` / `date` and outputs strings in the result layer; values in the same bucket can still be associated, and does not promise k-anonymity, nor does it reduce database reads or limit database side filtering, association, and grouping by original values.
- Auditing is an application layer record, not a regulatory-level WORM, and it cannot prevent DBA or other high-privilege accounts from directly connecting to the database.

- PostgreSQL's B2 column-level authorization, B5 cross-request logical sessions and planned transactions are all enabled by factory default; MySQL does not enter the B2 PostgreSQL path and does not support B5 cross-request transactions. Only one top-level SQL is still allowed per operation, subject to preflight plans and session security boundaries.
- The full native path still centers on PostgreSQL 14–18 and MySQL 8. The independent minimal dialects for DM, Oracle and YashanDB, and protocol compatibility paths such as OpenTenBase and PolarDB, remain strictly bounded. In v1.5.0, Dameng DM8 Pack3, KingbaseES V9R1C10 PG mode and Vastbase G100 Build 9 PG mode passed repeat tests of real clients, permission revocation, database disconnection and recovery on specified instances. In the same scope, v1.6.0 passed independent audit database failure, real lock-wait timeout, client cancellation, server-side termination and masking after recovery. The independent HighGo type in v1.7.0 is limited to bounded SELECT in 9.0.10 Enterprise PG mode; KingbaseES TLS is limited to the specified V9R1C10 PG instance. See HighGo acceptance and KingbaseES TLS acceptance. These results do not establish full dialect support or vendor certification.
- v1.0.0of `/mcp/2026-07-28` It is an independent stateless HTTP portal: the Agent Key is re-verified for each POST, the tenant is determined by the Key, and the Agent QPS is shared with the old portal; Version Header, `Mcp-Method`, when applicable `Mcp-Name` Must match the request body. old `/mcp` Continue to serve the old protocol with stdio; the new portal does not have B5 cross-request logic sessions or scheduled transaction facilities. For client configuration and request examples, seeMCP Access Guide。
- v0.4.0Provide linux/arm64 native glibc package (`aarch64` / `arm64`), and continue to provide linux/amd64 packages (`x86_64` / `amd64`) ； v2.4.0The GHCR tag must be used after it is publicly released and accepted through the dual-architecture manifest and anonymous pull.v0.5.0Optional TOTP MFA, OIDC, and LDAP/AD personnel authentication are available; see Configuration and LimitationsIdentity Authentication Guide。 v2.4.0The Kubernetes manifest is limited to fixed dual gateways and specified stateless read-only paths; PostgreSQL native HA, regulatory-level WORM, SIEM, musl / Alpine or CentOS 7 native one-click installation are still not supported, and RBAC / multi-tenancy capabilities are still MVP.

Online installation package upgrade preview (different from schema check)

Prepare a list of HTTPS provided by trusted channels with JSON fields `version`, origin HTTPS `url` and download content `sha256`. Currently only version checking and read-only previews are supported:

```
agentsqlctl upgrade check --manifest-url https://example.org/releases/manifest.json
agentsqlctl upgrade apply --manifest-url https://example.org/releases/manifest.json --backup-dir
/new/backup/path --config config.yaml --dry-run
```

The second command requires the backup target to be a directory that does not yet exist, and reads the download to verify SHA-256, but does not create the directory, backup files, stop services, or install programs. `--yes` cannot start the installation. The list and summary must be verified by independent trusted channels; for details, see Online Upgrade Security Boundary .

v4.2.0 offline delivery and diagnostics

The console Version and Upgrade page provides offline verification for the running version and diagnostic export commands. Verify all 15 official assets using an independently trusted key before installing. `support-bundle` exports only fixed schema checks and product evidence and refuses to overwrite files. See the delivery guide for commands, exit codes and cross-version recovery.

v4.2.0 native data Q&A

Deploy the Go Q&A service with `install.sh install --with-chatbi`, configure a workspace model and personal gateway mappings, and open its separate listener. The Chinese/English interface displays SQL, table results and durable audit IDs. The released v4.2.0 supports up to three successful turns of Q&A context, terms and recommended questions, controlled CSV/XLSX file datasources, and CSV/XLSX export under current permissions. The original SQLBot page has been compared only for the recorded synthetic flows. See the Q&A guide for identity import, offline installation, recovery and remaining compatibility work.

v4.3.0 charts, dashboards and embedded applications

The Q&A page can display the current authorized result as a table, bar chart, line chart or pie chart. Opening a historical chart runs a fresh query under current permissions and writes an audit event. AgentSQL's own dashboard supports text and record-chart cards, layout, tabs, search, preview, and configuration import/export without result rows. The embedded application management page supports creation, edits, enable/disable, signing-key rotation and deletion. Application tokens bind an existing non-admin user, workspace and application state, and conversations remain isolated by application. Only selected synthetic flows of the original SQLBot v1.10.2 page and SDK have been compared. Complex SQL, all pages, all integrations and zero-change replacement of a real business application remain unverified. See the current release notes and acceptance record.

v4.4.0 account administration and datasource relationships

After signing in to the standalone Q&A page, use the bottom of the sidebar to switch the current workspace, open account settings, or manage your API Keys. Administrators can open user management and model configuration. Old tokens should stop working after a user is disabled, membership is revoked, or a password is rotated. Model configuration is for a real local model; model answers in the public Demo are explicitly simulated.

The datasource page manages table selection, table and field remarks, and relationships only for authorized, currently visible server-bound PostgreSQL/MySQL sources. Select visible tables, load fields, then save relationships. Re-read after deselecting a table or revoking a grant; an old relationship cannot authorize a query. Relationships are JOIN hints to the local model. Each generated SQL statement still passes current gateway authorization, execution controls, masking, and durable audit. The page supports editing at most 64 visible tables. File sources and other databases do not gain the same capability from this release.

v4.5.0 validates only the ordinary SQLBot Community v1.10.3 Q&A page, original Python MCP SDK, one-source basic guest assistant, and Type 4 page embed in specified synthetic scenarios. Other Community pages, APIs, embed modes, and the user's actual business-system migration remain unverified. This does not establish complete compatibility or a zero-change cutover. See the v4.5.0 GitHub release notes for the exact scope.

v4.6.0 adds a bounded path for the fixed original Type 1 advanced assistant. Operators register a private third-party catalog, map its IDs to existing authorized gateway aliases, and enable the separate assistant listener only after configuring the host certificate, origin, and non-admin user. The catalog cannot create database connections or replace current authorization. See the Type 1 design and setup and v4.6.0 Release for exact scope. Active role, multi-user, and non-empty allow-list rules and other Community pages/APIs are outside this release's support claim.