

GLOBAL ACADEMIC RESEARCH INSTITUTE

COLOMBO, SRI LANKA



GARI International Journal of Multidisciplinary Research

ISSN 2659-2193

Volume: 12 | Issue: 02

On 30th June 2026

<http://www.research.lk>

Author: M. H Krishantha Sameera

Askleya Group LTD, UK

GARI Publisher | Computer Science | Volume: 12 | Issue: 02

Article ID: IN/GARI/JOU/2026/X07/FR | Pages: 17-30 (14)

ISSN 2659-2193 | Edit: GARI Editorial Team

Received: 08.04.2026 | Published: 30.06.2026

DOI:<https://doi.org/10.63572/gari1569>

RULES AS DATA: A GOVERNED, DUAL-TRANSPORT MODEL CONTEXT PROTOCOL SERVER FOR SAFE LARGE LANGUAGE MODEL ACCESS TO ENTERPRISE ERP DATA

M. H Krishantha Sameera

Askleya Group LTD, UK

ABSTRACT

Large language models (LLMs) can translate business questions into SQL, but connecting them to production enterprise resource planning (ERP) databases raises two unresolved problems: the model does not know organisation-specific metric definitions, and the queries it generates are untrusted code. This paper presents the design, implementation and evaluation of a Model Context Protocol (MCP) server that gives Anthropic's Claude governed, read-only access to Microsoft Dynamics 365 Business Central data held in Azure SQL Database. Following a design science approach, the artefact contributes three mechanisms. First, a rules-as-data knowledge base stores business rules and key performance indicator (KPI) formulas as rows of a database table; the rows are composed into the server's instructions when a session is initialised and can be refreshed at run time without changing code. Second, a dual-transport server exposes identical tools and controls over both Streamable HTTP and the legacy HTTP+SSE transport. Third, a four-layer defence-in-depth design intercepts every generated query with a single-pass lexer, statement rules and a parenthesis-depth-aware FROM-list state machine that enforces a table allow-list, backed independently by least-privilege database grants. The system has been in production since June 2026. The evaluation combines a 38-case adversarial test suite (23 attacks blocked, 15 legitimate queries accepted, no failures), end-to-end protocol tests and an analysis of three vulnerabilities found and fixed during development. The results suggest that separating what the model is

told from what the system enforces is a practical pattern for deploying LLM agents on enterprise data.

Keywords: Model Context Protocol, text-to-SQL security, enterprise resource planning

INTRODUCTION

Large language models have made natural-language access to relational data a realistic prospect. On cross-domain benchmarks such as Spider, LLM-based systems now generate correct SQL for a large share of questions (Yu et al., 2018; Gao et al., 2024). Results on benchmarks built from real databases are more sobering. On BIRD, which pairs 12,751 questions with 95 databases from 37 professional domains, GPT-4 reached 54.89% execution accuracy against 92.96% for human experts (Li et al., 2023). The authors attribute much of the gap to dirty values and to external knowledge: facts about the data that cannot be read from the schema.

Enterprise resource planning systems sharpen both problems. Microsoft Dynamics 365 Business Central, the ERP studied here, stores each company's data in tables whose names combine a company prefix, a table name and an application GUID, for example [Company\$Sales Header\$<GUID>]. It represents blank dates with the placeholder 1753-01-01. More importantly, the metrics managers ask about are defined by the organisation, not by the database. Whether an order counts towards On-Time-In-Full (OTIF)

performance depends on its status, its requested delivery date, whether it is a sample, and a per-order cap that prevents early over-shipment inflating the result. A general model cannot infer these rules, and a plausible guess produces a different number.

Connecting an LLM to production data also creates a security problem. SQL written by a model must be treated as untrusted input. It may be wrong, it may be steered by a malicious user, and it may be steered indirectly by instructions planted in data the model reads (Greshake et al., 2023). Pedro et al. (2025) demonstrated such prompt-to-SQL injections against LLM-integrated web applications, and the OWASP Foundation (2024) lists prompt injection and excessive agency among the main risks of LLM applications. The Model Context Protocol, released by Anthropic in November 2024, standardises how AI assistants connect to tools and data sources (Anthropic, 2024; Model Context Protocol, 2025). MCP removes much of the integration effort, but it deliberately leaves domain knowledge and data governance to each server. Little has been published on how such a server should be built for enterprise data that is both semantically complex and sensitive. This paper reports the design and production deployment of an MCP server for a distribution company running Business Central, whose identity is withheld. The system has served business users since June 2026. It addresses three research questions:

- RQ1. How can an MCP server expose ERP data to an LLM over both current and legacy transports without duplicating security logic?
- RQ2. How can organisation-specific business rules be supplied to the model so that they are governed, consistent and changeable without code changes?
- RQ3. How can untrusted AI-generated SQL be intercepted and constrained so that no single defect exposes data?

The paper makes four contributions: (1) a transport-independent server design in which both MCP transports share one capability factory; (2) a rules-as-data pattern that keeps KPI definitions in a governed database table and binds them to each session at initialisation; (3) a query guard combining a single-pass lexer with a depth-aware FROM-list state machine, layered with least-privilege database grants; and (4) an evaluation including an analysis of three real vulnerabilities and the layer that contains each. The remainder of the paper reviews related work, describes the methodology and artefact, presents the evaluation data, and discusses implications and limitations.

LITERATURE REVIEW

Text-to-SQL with large language models

Research on text-to-SQL moved from trained semantic parsers to prompting general-purpose LLMs. Rajkumar et al. (2022) showed that a code-trained LLM could reach competitive accuracy on Spider with zero-shot prompting. Pourreza and Rafiei (2023) decomposed the task into schema linking, classification and self-correction steps (DIN-SQL). Gao et al. (2024) systematically compared prompt representations and example selection, and emphasised the token efficiency of prompts. These studies show that the representation of the schema and the context in the prompt strongly affects accuracy. Benchmarks drawn from real databases expose the limits of schema-only prompting. BIRD explicitly supplies external knowledge evidence with each question because many questions cannot be answered from the schema alone (Li et al., 2023). Enterprise reporting is an extreme case: the knowledge needed is not general world knowledge but the organisation's own definitions, which differ between companies and change over time.

Tool use and the Model Context Protocol

Agent frameworks let a model interleave reasoning with actions such as tool calls (Yao et al., 2023), and models can learn when and how to call external tools (Schick et al., 2023). MCP standardises this interaction as JSON-RPC messages between a client, which hosts the model, and a server, which offers tools, resources and prompts. Its initialize result may carry optional instructions describing how to use the server (Model Context Protocol, 2025). The 2025-06-18 specification defines two standard transports, stdio and Streamable HTTP. It marks the earlier HTTP+SSE transport as deprecated and describes how servers can keep supporting it for older clients. It also requires Streamable HTTP servers to validate the Origin header, and recommends binding to localhost and authenticating all connections.

Supplying domain knowledge

Retrieval-augmented generation supplies external knowledge by retrieving relevant passages at query time (Lewis et al., 2020). In analytics, the older tradition of conformed dimensions and metrics in data warehousing (Kimball & Ross, 2013), and its modern form in semantic layers, centralises metric definitions so that every report computes a KPI the same way. The present work borrows the governance goal of that tradition but applies it to the context of an LLM rather than to a query engine: definitions are kept as governed data and handed to the model, which still writes the SQL itself.

Security of LLM-generated queries

Classical SQL injection defences assume that a trusted program assembles queries and an attacker controls only data values (Halfond et al., 2006). LLM integration breaks this assumption, because the model writes the whole query. Greshake et al. (2023) showed that indirect

prompt injection can redirect a model's actions through content it processes. Pedro et al. (2025) showed that prompt-to-SQL injections can read and modify data through LangChain-based applications, and evaluated defences including database permission hardening and query rewriting. Their findings, together with the principle of least privilege (Saltzer & Schroeder, 1975), motivate defences that do not depend on the model behaving well.

Research gap

Prior work evaluates text-to-SQL accuracy on public benchmarks, or attacks on LLM-integrated applications in general. There is little design-level evidence on how to build an MCP server that combines governed organisational knowledge with layered controls on generated SQL, deployed against a real ERP. This paper addresses that gap with a production artefact and its evaluation.

METHODOLOGY

Research approach

The study follows design science research (Hevner et al., 2004). The artefact was built to solve a real organisational problem, evaluated against explicit requirements, and refined over successive build-and-evaluate cycles. The author designed, implemented and deployed the system as its sole engineer. Evaluation used three methods: automated adversarial testing of the query guard, end-to-end protocol testing against a stand-in database module, and analysis of the vulnerabilities found during development, including the layer that contains each.

Case context and requirements

The case organisation reports on two areas. Delivery performance covers OTIF and Order Acknowledgement (OA) from Business Central sales headers, lines and shipment lines. Sales performance covers actual, budget, forecast and margin figures held in a flat reporting table. Managers

previously depended on analysts to write queries, and the same KPI was sometimes computed in different ways. Table 1 lists

the requirements derived from this context.

Table 1 – Design requirements

ID	Requirement
R1	Answer business questions in natural language from live ERP data
R2	Apply the organisation's own KPI definitions consistently
R3	Let administrators change rules without code changes
R4	Never modify data; never read outside approved tables
R5	Support current and older MCP clients
R6	Keep client identifiers out of source code

System architecture

Figure 1 shows the architecture. Users work in a Claude client, which hosts the model and connects to the server over HTTPS with an API key. An nginx reverse proxy terminates TLS and forwards requests to the Node.js server, which listens only on the loopback interface. The server is about 525 lines of JavaScript built on the official MCP SDK (version 1.30.1), the mssql driver and the Zod validation library. Inside, an HTTP router authenticates every request before handing it to one of two transports. Both transports connect to the same MCP server core, which holds the instructions and four tools. Database queries leave the core only through the query guard, and run under a login that can read nine tables.

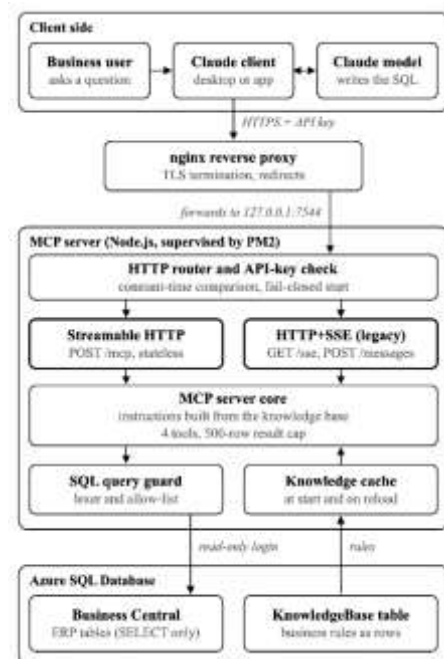


Figure 1 – System architecture

Dual-transport design (RQ1)

Claude Desktop connected through the HTTP+SSE transport when the system was built, while newer clients use Streamable HTTP. The specification allows a server to support both by hosting

the old endpoints next to the new one (Model Context Protocol, 2025). The router checks routes in a fixed order, so authentication always precedes MCP handling (Table 2).

Table 2 – Request routing order

Route	Key	Handler
GET /health	No	Liveness response
well-known/*, /register	No	404, so clients skip OAuth discovery
any other route	Yes	401 if the key is missing or wrong
POST / or /mcp	Yes	Streamable HTTP, stateless
GET /sse	Yes	Open HTTP+SSE stream and session
POST /messages	Yes	Deliver message to an SSE session

The two transports differ in state. Streamable HTTP runs statelessly: each POST receives a new transport and a new server instance, and both are closed when the response ends. The server issues no session identifier, which the specification permits, and answers GET on the endpoint with 405, which signals that it offers no server-initiated stream. Statelessness removes session memory that could leak or be lost on restart, and lets any process serve any request. HTTP+SSE cannot be stateless, because responses travel on a long-lived stream opened by an earlier request. Its transports are therefore kept in a map keyed by session identifier and removed when the stream closes. The key design decision is that transport is decoupled from capability. Both paths call the same factory, `buildMcpServer()`, which constructs the instructions, the four tools and their validation. Security controls therefore exist once, and a third transport could be added without touching them. Figure 2 shows the resulting message sequence for one question over Streamable HTTP; the SSE path carries

the same messages over different endpoints.

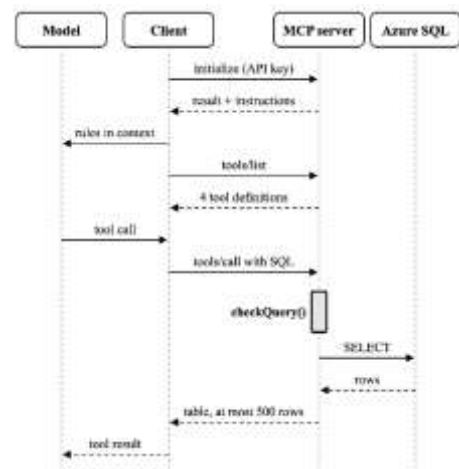


Figure 2 – MCP message sequence for one question

The model sees only two things: the instructions, returned in the initialize result, and four tools with Zod input schemas. `query_database` runs a guarded query and returns at most 500 rows as a Markdown table. `list_tables` returns the allow-list rather than the real catalogue. `describe_table` returns column metadata for allow-listed tables through a bound parameter. `reload_knowledge_base` refreshes the rules. A rejected query returns an explanatory error, which lets the model correct its SQL without human intervention.

Rules-as-data knowledge base (RQ2)

All business knowledge is held in one table, `dbo.KnowledgeBase`, in the same database as the data it describes (Table 3). Each row is a topic: a short name and its content, written as natural-language definitions combined with exact SQL fragments where precision matters. The production knowledge base has 12 topics in two reporting areas, about 22,000 characters or roughly 5,500 tokens (Table 4). An index topic routes questions to the right area.

Table 3 – KnowledgeBase schema

Column	Role
Id (int, PK)	Orders topics; general rules come first
Topic (nvarchar 100)	Becomes a heading in the instructions
Content (nvarchar max)	Rule text plus exact SQL fragments
UpdatedAt, UpdatedBy	Lightweight provenance for each rule

Table 4 – Knowledge base topics

Topic	Tells the model
Knowledge Base Index	Which topics and tables answer which questions; ask before querying
Overview	Data source and Business Central naming
Allowed Tables	The nine queryable tables
Order Inclusion Rules	Status, date and exclusion filters, as a WHERE clause
OTIF Calculation	Denominator, ETA cutoff, per-order cap, buckets
OA Calculation	Acknowledgement days; handling of nulls and negatives
KPI Reference	Reference values for sanity checks
Important Rules	Cross-cutting rules, e.g. month by requested date
Glossary	Business Central field names and terms
Sales table reference	Every column of the flat sales table
Sales query patterns	Tested templates for actual, budget, forecast
Business terms	Company terms mapped to columns and values

The server reads all rows in Id order and serialises them as Markdown, one heading per topic. The result is placed between a fixed preamble and fixed formatting rules in the MCP instructions field. The fixed text contains only safety and presentation rules; every business definition comes from the table. Equation (1) shows one rule as stored, the OTIF definition, where S is the quantity shipped by order o on or before its cutoff date and D is the order's denominator quantity.

$$\text{OTIF (\%)} = \left[\frac{\sum_o \min(S_o, D_o)}{\sum_o D_o} \right] \times 100 \quad (1)$$

The cap in the numerator prevents early over-shipment from inflating the result. The same topic defines the cutoff as the first production ETA, or else the production ETA, plus one day, with the placeholder date treated as blank. A general model given only the schema would plausibly group by order date, include sample orders, or allow OTIF above 100%. Each of these changes the headline figure. Rules are bound at three moments (Figure 3). At start-up the server reads the table into a cache. Each new session embeds the current cache in its instructions. When an administrator edits a row and a user asks for a reload, the tool re-reads the table and replaces the cache. A conversation's instructions are fixed once it is initialised, so the tool also returns the full updated rules as its result, which brings the requesting conversation up to date. Rules are thus read on demand and bound to sessions at initialisation, not queried for every session.

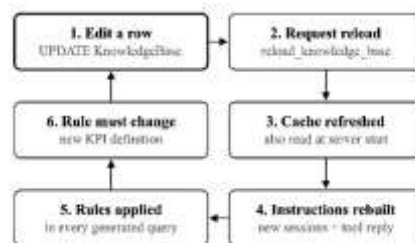


Figure 3 – Knowledge lifecycle

The allow-list appears in three places that must agree: the server code, the Allowed Tables topic and the database grants. The knowledge base tells the model the rule, while the code and the database enforce it. Because the server's login can read but not write KnowledgeBase, the model can quote its rules but cannot change them.

Threat model

The security design assumes that the model may act against the organisation's interests. An adversary may control the user's question directly, or influence the model indirectly through text stored in database rows that the model later reads (Greshake et al., 2023). The model is therefore treated as an untrusted query author, even though it is a trusted product. The protected assets are the confidentiality of every table outside the allow-list, the integrity of all data, and the availability of the database. The server code, the host operating system, the reverse proxy and the database administrators are trusted. Compromise of the host and insider misuse by administrators are out of scope. Under this model, the safety goal is stated as an invariant: whatever SQL the model produces, the database executes only single read-only statements over the nine approved tables. Instructions to the model can make violations less frequent, but they cannot establish the invariant, because the adversary may control the model's input. The invariant must instead be enforced by components the adversary cannot influence.

Query interception and defence in depth (RQ3)

Every query passes a single interception point between the tool call and the database driver. Queries are validated and either accepted or rejected; the guard never rewrites SQL. Figure 4 shows the eight stages of the guard function, check Query.

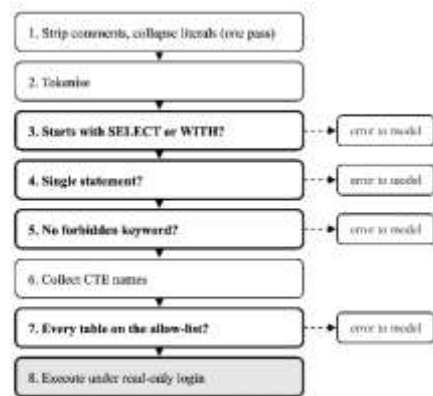


Figure 4 – Query guard pipeline

Lexical stage. Comments and string literals must be removed before parsing, so that a table name inside a comment cannot be hidden and words inside strings cannot cause false alarms. The guard does this in one left-to-right pass with a single regular expression whose alternatives are, in order, bracketed identifiers, double-quoted identifiers, string literals (including Unicode N'...' literals), line comments and block comments. Identifiers are kept, literals collapse to an empty literal, and comments become whitespace. Because the scan is left to right, each construct is recognised in context: a -- inside a string or a bracketed name is not taken for a comment. SQL Server nests block comments, while the pass ends a comment at the first closing marker. That difference only ever leaves the guard seeing more of the query than the server executes, never less.

Statement stage. The first token must be SELECT or WITH. A semicolon may appear only as trailing tokens, which blocks stacked statements. No token may match a list of 29 keywords covering data modification, definition, permissions, execution (EXEC, EXECUTE), variables (DECLARE, SET), SELECT ... INTO, external data access (OPENROWSET, OPENQUERY, OPENDATASOURCE, OPENXML, BULK), delays (WAITFOR) and server control. Keywords are matched

only as bare words, so a bracketed column named [Delete] is unaffected. Table stage. The guard keeps a stack of flags, one per parenthesis depth, recording whether the scanner is inside a FROM list at that depth (Figure 5). FROM, JOIN and APPLY set the flag and introduce a table reference. A comma while the flag is set introduces another. Eleven clause keywords (WHERE, GROUP, ORDER, HAVING, UNION, EXCEPT, INTERSECT, OPTION, FOR, SELECT, WINDOW) clear it. Each opening parenthesis pushes a fresh level, so subqueries, derived tables and set operations at any depth are covered by the same rules.

```
for (let i = 0; i < tokens.length; i++)
{
  const t = tokens[i], up = t.toUpperCase();
  if (t === "(") { depth++; inFrom[depth] = false; continue; }
  if (t === ")") { depth = Math.max(0, depth - 1); continue; }
  if (up === "FROM" || up === "JOIN" || up === "APPLY") {
    inFrom[depth] = true; checkRef(i + 1);
  } else if (t === "," && inFrom[depth]) {
    checkRef(i + 1); // FROM a, b
  } else if (isBareWord(t) && FROM_LIST_END.has(up)) {
    inFrom[depth] = false;
  }
}
```

Figure 5 – FROM-list state machine (excerpt of sql-guard.js)

Each reference is read as a dotted name with parts $p_1 \dots p_n$. With A the allow-list, C the set of common table expression (CTE) names defined at depth zero, and $\sigma(x)$ the schema of allowed table x, a reference not followed by an opening parenthesis is accepted if and only if:

$$(n = 1 \wedge p_1 \in C) \vee (n = 1 \wedge p_1 \in A) \vee (n = 2 \wedge p_2 \in A \wedge p_1 = \sigma(p_2)) \quad (2)$$

A reference followed by a parenthesis is a table-valued function and is always rejected, as is any three-part name, which would address another database. Matching

CTE names only against one-part references closes a shadowing attack, in which a query defines a CTE named like a protected table and then reads the real table through a qualified name.

Database stage. The server connects with an encrypted, certificate-validated connection under a dedicated login. The provisioning script removes the login from the db_datareader role, revokes database-wide VIEW DEFINITION, and grants SELECT on exactly the nine allow-listed tables. No write, definition or execute permission is granted. Company-specific table names are supplied as script variables and environment variables, so the repository holds no client identifiers. Other input controls. Request bodies are limited to 1 MB and must be valid JSON. Tool arguments are validated by Zod schemas. The API key is hashed with SHA-256 and compared with a constant-time function, and the server refuses to start when the key or the company identifiers are missing. Figure 6 summarises the resulting layers, and Table 5 maps attack classes to the layers that stop them.

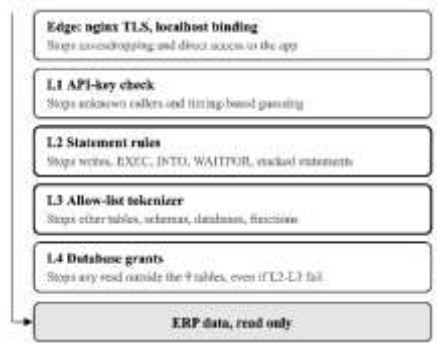


Figure 6 – Defence-in-depth layers

Table 5 – Attack classes and blocking layers

Attack class	Application	Database
Write or DDL	Prefix rule, keywords	No write grants
Stacked statements	Single statement	–
Unlisted tables	Allow-list (L3)	No SELECT grant
Other schema or database	Schema rule, 3-part ban	No grant
External data access	Keyword list	No grant
Time-based denial of service	WAITFOR banned	Driver timeout
Very large results	500-row output cap	–

DATA ANALYSIS

Implementation profile

Table 6 summarises the artefact. Its small size is deliberate: the query guard has no runtime dependencies and can be tested without a database, which made the adversarial testing below fast to repeat.

Table 6 – Implementation profile

Measure	Value
Server code (index.js)	362 lines
Query guard (sql-guard.js)	163 lines, no dependencies
Runtime dependencies	4 (MCP SDK 1.30.1, mssql 11.0.2, Zod 3.25.76, dotenv 16.6.1)

Measure	Value
MCP transports / tools	2 / 4
Allow-listed tables	9
Forbidden keywords	29
Knowledge base	12 topics, about 5,500 tokens
In production since	June 2026

Adversarial test suite

The guard was evaluated with a suite of 38 hand-constructed queries, run with the Node.js test runner. Attack queries were written to cover each threat class: statement-level attacks such as stacked statements and execution; table-level attacks such as comma joins, schema hopping and CTE shadowing; and lexical evasion, which hides text from the guard inside comments, strings or identifiers. Legitimate queries include the tested templates from the knowledge base and constructs that a naive guard might wrongly reject, such as commas inside window functions or keywords inside string literals. Table 7 reports the results: every attack was blocked and every legitimate query was accepted.

Table 7 – Adversarial test results

Category	Cases	Correct
Statement-level attacks	8	8 blocked
Table-level attacks	9	9 blocked
Lexical-evasion attacks	6	6 blocked
Legitimate: core syntax	11	11 accepted

Category	Cases	Correct
Legitimate: lexical edge cases	4	4 accepted
Total	38	38 (100%)

fails 5 of the 38 cases, all of them lexical-evasion attacks. The earlier version removed comments before string literals, so a comment marker inside a literal concealed the rest of the query; `SELECT '-' AS a, * FROM SecretTable` was accepted. The single-pass lexer fixes all five, and the 33 cases that passed before still pass. The suite therefore works both as evidence of the guard's current behaviour and as a regression test.

Knowledge base content

The 12 production topics were analysed by the kind of knowledge they supply. Five kinds recur. Definitional knowledge gives formulas, such as OTIF and OA. Filtering knowledge states which records count, such as order statuses, date windows and the exclusion of samples. Data-quality knowledge describes quirks of the source system, such as the 1753-01-01 placeholder date and negative acknowledgement times caused by pre-confirmed production. Mapping knowledge ties business vocabulary to columns and values. Behavioural knowledge tells the model how to work, for example to ask clarifying questions before querying and to aggregate rather than list raw rows. These kinds overlap with the external knowledge that BIRD supplies with its questions, which distinguishes numeric reasoning, domain, synonym and value-illustration knowledge (Li et al., 2023). Two kinds found here have no direct counterpart there: data-quality knowledge specific to one source system, and behavioural knowledge about how the assistant should interact. Both are organisational rather than general, which supports keeping them

in governed data owned by the organisation. The largest topics are the two that describe the flat sales table, at 5,245 and 5,110 characters, while the delivery-performance rules are compact, at 464 to 1,521 characters each.

End-to-end protocol tests

The complete server was exercised over HTTP with real MCP messages, with the database driver replaced by a stand-in module so that production data was not touched. Table 8 lists the checks and outcomes.

Table 8 – End-to-end protocol tests

Check	Result
Missing or wrong API key	401
Malformed JSON body	400
GET on stateless endpoint	405
initialize, then tools/list	4 tools; knowledge base in instructions
Comma-join attack via tools/call	Rejected with allow-list error
700-row result	Capped at 500 with a note
reload_knowledge_base	Confirmation and new rules returned
SSE connect and initialize	202; response on the stream
Listening address	127.0.0.1 only

Vulnerability analysis

Three security defects were found and fixed during development (Table 9). Each is analysed by asking which remaining layer would have contained it. This analysis provides direct evidence for, and against, the defence-in-depth claim.

Table 9 – Vulnerabilities found and the containing layer

Defect	Effect	Contained by
Fail-open authentication	Unset key let keyless requests in	Guard only; login then had db_datareader
Regex table matching	FROM a, secret was accepted	Not contained: login then had db_datareader
Lexical ordering	Comment marker in a literal hid tables	L4 grants

The first defect, found in code review, compared the presented key with an environment variable directly. When the variable was unset, a request without a key compared two undefined values and was admitted. The fix refuses to start without a key and uses constant-time comparison. Had it been exploited, only the query guard would have stood between a caller and the data. The second defect matched only the first identifier after FROM or JOIN, so a comma-joined table escaped the allow-list. At that time the database login held the broad db_datareader role, so no layer contained the defect; this finding motivated the least-privilege grants. The third defect, the lexical-ordering bypass described above, would have been contained by those grants, because the database refuses to read any table outside the nine.

The analysis shows that the application guard alone was bypassable twice, and that the database grants are what turn two independent layers into a real safety margin. It also shows how a system can carry a defence-in-depth design on paper while one layer is missing in practice: before the grants were tightened, the second defect had no backstop.

DISCUSSION

Telling versus enforcing

The central design principle to emerge is a strict separation between what the model is told and what the system enforces. The knowledge base tells the model which tables exist, how KPIs are defined and how to format answers. Instructions of this kind are advisory: a model may misread them, and a prompt injection may override them. Every property that matters for safety is therefore enforced outside the model, by the guard and by the database. This answers RQ3 and agrees with the threat model of Greshake et al. (2023) and Pedro et al. (2025), in which the model itself may be adversarially steered.

Rules as data

For RQ2, keeping business rules in a database table delivers governance properties that prompts written into code lack. Rules sit next to the data they describe, carry provenance, can be listed and queried, and can be changed by the people who own the definitions, without a release. Compared with retrieval-augmented generation (Lewis et al., 2020), the design injects the whole rule set rather than retrieving fragments. At about 5,500 tokens this is affordable, and it guarantees that every interacting rule, such as inclusion filters and the OTIF cap, is present together. Compared with semantic layers in the data warehousing tradition (Kimball & Ross, 2013), the pattern is lighter: it requires no metric engine, and the model still writes the SQL, guided by definitions rather than constrained to predefined metrics. That flexibility is also its weakness, because correctness depends on the model applying the rules, which is why answer accuracy must be measured rather than assumed.

Comparison with prior defences

Pedro et al. (2025) evaluated defences against prompt-to-SQL injection, including hardening database permissions and rewriting queries. The present design adopts permission hardening as its final layer but rejects rewriting: a guard that edits SQL must understand it fully to edit it safely, while a guard that only accepts or rejects can fail safe by rejecting what it does not understand. The design also avoids a second model as a judge of queries, keeping every check deterministic and testable without network access. The cost of this conservatism is occasional false rejection of unusual but legitimate SQL, such as table-valued functions; in this case the cost was judged acceptable, because the model receives the reason and can rephrase the query.

Transport independence

For RQ1, routing both transports through a single capability factory meant the deprecation of HTTP+SSE in the specification required no change to tools or controls. Running Streamable HTTP statelessly simplified the server and removed a class of session-management defects. In the first version, session-tracking code existed but never ran, because the transport was in fact running in stateless mode. This illustrates the value of making protocol state explicit rather than relying on library defaults.

Practical implications

For practitioners, the findings suggest a checklist for MCP servers over enterprise data. Treat generated SQL as untrusted. Validate it, never repair it. Perform lexical analysis in a single context-aware pass. Track table references at every nesting depth. Grant the database login exactly the tables the allow-list names. Keep business definitions in governed data. Keep client identifiers in configuration. Each item corresponds to a defect or design decision observed in this case.

Limitations and threats to validity

Several limitations qualify these results. First, the evaluation measures security behaviour and protocol correctness, not answer accuracy. No benchmark of business questions with verified answers has yet been run, so the benefit of the knowledge base on accuracy is argued, not measured. Second, the adversarial suite is hand-constructed by the developer, which risks blind spots; generated or fuzzed queries would provide stronger evidence. Third, the guard is a focused lexer and state machine, not a complete T-SQL grammar, so its soundness rests on the database layer rather than on parser completeness. Fourth, the system uses one shared API key, so queries cannot be attributed to individuals, and it does not yet validate the Origin header as the specification requires for Streamable HTTP. Fifth, injecting the full knowledge base will not scale to much larger rule sets without retrieval. Finally, the study covers one organisation and one ERP, and the author is also the system's developer, which limits generalisability and objectivity.

CONCLUSION

This paper presented a production MCP server that lets an LLM answer business questions from a live Business Central ERP while keeping the organisation's definitions governed and its data protected. Three mechanisms carry the design: a transport-independent capability factory serving both MCP transports; a rules-as-data knowledge base bound to each session at initialisation and refreshable at run time; and four layers of control over generated SQL, ending in least-privilege database grants. A 38-case adversarial suite passed in full, end-to-end protocol tests confirmed the intended behaviour, and the analysis of three real defects showed both the value and the necessity of an independent database layer.

Future work will close the gaps identified above. Priorities are a benchmark of real business questions, to measure answer accuracy with and without the knowledge base; query audit logging, to collect production telemetry on usage, rejections and retries; fuzz testing of the guard; per-user authentication and Origin validation; and retrieval for larger rule sets. More broadly, the separation of telling from enforcing offers a simple test for any LLM agent on enterprise data: every property that must hold should hold even if the model ignores what it was told.

Acknowledgments

The author thanks the case organisation for access to its systems and for permission to report this work in anonymised form. Claude Code, an AI coding assistant, was used for help with code syntax during development. An AI assistant (Claude) was used to help draft and edit this manuscript. The author reviewed all content and takes full responsibility for it.

REFERENCES

- Anthropic. (2024, November 25). *Introducing the Model Context Protocol*. <https://www.anthropic.com/news/model-context-protocol>
- Gao, D., Wang, H., Li, Y., Sun, X., Qian, Y., Ding, B., & Zhou, J. (2024). *Text-to-SQL empowered by large language models: A benchmark evaluation*. *Proceedings of the VLDB Endowment*, 17(5), 1132–1145. <https://doi.org/10.14778/3641204.3641221>
- Greshake, K., Abdelnabi, S., Mishra, S., Endres, C., Holz, T., & Fritz, M. (2023). *Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection*. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security* (pp. 79–90). ACM. <https://doi.org/10.1145/3605764.3623985>
- Halfond, W. G. J., Viegas, J., & Orso, A. (2006). *A classification of SQL-injection attacks and countermeasures*. In *Proceedings of the IEEE International Symposium on Secure Software Engineering*. IEEE.
- Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). *Design science in information systems research*. *MIS Quarterly*, 28(1), 75–105.
- Kimball, R., & Ross, M. (2013). *The data warehouse toolkit: The definitive guide to dimensional modeling* (3rd ed.). Wiley.
- Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., & Kiela, D. (2020). *Retrieval-augmented generation for knowledge-intensive NLP tasks*. *Advances in Neural Information Processing Systems*, 33, 9459–9474.
- Li, J., Hui, B., Qu, G., Yang, J., Li, B., Li, B., Wang, B., Qin, B., Geng, R., Huo, N., Zhou, X., Ma, C., Li, G., Chang, K., Huang, F., Cheng, R., & Li, Y. (2023). *Can LLM already serve as a database interface? A big bench for large-scale database grounded text-to-SQLs*. *Advances in Neural Information Processing Systems*, 36 (Datasets and Benchmarks Track).
- Model Context Protocol. (2025). *Specification, version 2025-06-18*. <https://modelcontextprotocol.io/specification/2025-06-18>
- OWASP Foundation. (2024). *2025 Top 10 risk & mitigations for LLMs and Gen AI apps*. <https://genai.owasp.org/llm-top-10/>
- Pedro, R., Castro, D., Carreira, P., & Santos, N. (2025). *From prompt injections to SQL injection attacks: How protected is your LLM-integrated web application?* In *Proceedings of the IEEE/ACM 47th International Conference on Software Engineering (ICSE 2025)*. IEEE. <https://arxiv.org/abs/2308.01990>
- Pourreza, M., & Rafiei, D. (2023). *DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction*. *Advances in Neural Information Processing Systems*, 36.

- Rajkumar, N., Li, R., & Bahdanau, D. (2022). Evaluating the text-to-SQL capabilities of large language models (arXiv:2204.00498). *arXiv*. <https://arxiv.org/abs/2204.00498>
- Saltzer, J. H., & Schroeder, M. D. (1975). The protection of information in computer systems. *Proceedings of the IEEE*, 63(9), 1278–1308.
- Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., & Scialom, T. (2023). Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36.
- Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). ReAct: Synergizing reasoning and acting in language models. In *Proceedings of the International Conference on Learning Representations (ICLR 2023)*.
- Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., Ma, J., Li, I., Yao, Q., Roman, S., Zhang, Z., & Radev, D. (2018). Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing* (pp. 3911–3921). ACL.