

Extending ResourceLink: Patterns for Large Dataset Processing in MCP Applications

Scott Frees

Ramapo College of New Jersey
Department of Computer Science
Mahwah, New Jersey
sfrees@ramapo.edu

Abstract

Large language models translate natural language into database queries, yet context window limitations prevent direct deployment in reporting systems where complete datasets exhaust available tokens. The Model Context Protocol specification defines ResourceLink for referencing external resources, but practical patterns for implementing scalable reporting architectures remain undocumented. This paper presents patterns for building LLM-powered reporting systems that decouple query generation from data retrieval. We introduce a dual-response pattern extending ResourceLink to support both iterative query refinement and out-of-band data access, accompanied by patterns for multi-tenant security and resource lifecycle management. These patterns address fundamental challenges in LLM-driven reporting applications and provide practical guidance for developers building them.

data for query validation. Once the LLM generates a valid query, the host application may execute it independently and render visualizations without LLM involvement, reducing token consumption while maintaining natural language interfaces.

MCP specifications define ResourceLink [2] for referencing rather than embedding data in tool responses, however patterns for reporting tool architectures remain undocumented. Existing literature focuses on conversational analysis workflows where LLMs consume complete datasets, not query generation scenarios. We address this gap by introducing a dual-response pattern combining ResourceLink with sampling validation and metadata for iterative query refinement. We also present multi-tenant server strategies for data isolation and resource discovery mechanisms for dynamic introspection.

1 Introduction

Large language models (LLMs) serve two fundamentally different roles in enterprise data tools - *analysis* and *retrieval*. For analysis tasks, the LLM ingests and synthesizes data within context window to provide discrete answers to a user; e.g. “What is the top trending product in terms of sales nation-wide“. In contrast, retrieval tasks use the LLM to generate queries from natural language - where the results are presented to the user or used downstream for data exports and visualizations; e.g. “Show me the nation-wide sales of our products“.

This distinction significantly impacts context window utilization. Analysis exhausts tokens proportional to result size, creating computational overhead and latency. However, retrieval tasks need only schema-representative samples or aggregate meta-

2 Background

2.1 Context Window Budget

LLMs typically use MCP tools to access data when used in reporting systems, and tool responses place strain context window limitations. The self-attention mechanism’s quadratic complexity [5] requires $O(n^2 \cdot d)$ time and memory, creating direct conflict between reporting scale and inference performance. Extended contexts degrade both latency and accuracy. Liu et. al [9] demonstrated LLMs exhibit the “lost in the middle“ phenomenon, struggling with information in extended sequences. Leng et al. [8] found only a subset of models maintain accuracy above 64,000 tokens, with substantial latency increases.

2.2 Tool Call Context Consumption

Tool specifications and results consume substantial context budget. When LLMs invoke external tools for reporting data, function parameters, metadata, and returned structured data all occupy context capacity. Gim et al. [6] noted synchronous tool-use paradigms exacerbate this - each tool call’s complete result must be incorporated before subsequent reasoning. Reporting pipelines invoking multiple sources may find results occupy 70-80% of context before analysis begins, forcing a choice between comprehensive data and reasoning capacity.

2.3 MCP ResourceLink

MCP specification version 2025-06-18 [2] introduced ResourceLink, enabling servers to reference resources via URI-based handles rather than embedding complete payloads. Resource links provide references without inline transmission. The schema includes URI, description, MIME type, and size, enabling clients to understand characteristics without retrieving full contents. Resource links are ephemeral in nature, they do not represent (necessarily) persistent entities on the server, rather *artifacts* of tools invocations. When reporting tools return ResourceLinks to query results, LLMs receive only handles and metadata - actual results remain unconsumed. Host applications subsequently retrieve full datasets via URI for rendering, decoupling query formation from data transmission.

2.4 Zero-Shot Accuracy Challenge

Leading text-to-SQL systems achieve execution accuracy exceeding 80% on cross-domain benchmarks [10], yet this implies 15-20% error rates invisible to end users lacking technical expertise to validate outputs. Error rates reduce when models iterate, observing output and refining queries [3, 14, 15].

ResourceLink’s handle-based architecture suggests LLMs generate correct queries without observing execution samples. Without validating expected schema, row counts, or distributions, LLMs cannot detect errors in joins, aggregations, or filters until rendering. This produces syntactically valid but semantically incorrect queries. This necessitates patterns enabling iterative refinement through result inspection rather than zero-shot generation.

2.5 User Experience Needs

Effective LLM-based reporting must support exploratory questioning, iterative refinement [13],

and **persistent** artifacts reflecting updated data [1]. Business intelligence research shows systems must support goal-oriented workflows with multi-dimensional manipulation [12]. Substantive work such as data retrieval, transformation, and rendering occurs in specialized backend services [4]. This necessitates hybrid context management: resource handles or opaque identifiers reference server-side results, allowing LLMs to reason about structures without materializing contents, while small samples or summary statistics enable direct question-answering. This dual-mode operation (lightweight samples for exploration, handle-based references for artifacts) preserves context for multi-turn dialogues while enabling reports backed by arbitrarily large datasets.

3 Dual Response Pattern

The Model Context Protocol specification defines a ResourceLink primitive for referencing external resources within tool responses:

```
1 interface ResourceLink {  
2     uri: string;  
3     name: string;  
4     description?: string;  
5     mimeType?: string;  
6     size?: number; // bytes  
7 }
```

Listing 1: ResourceLink Schema as defined in MCP Specifications

3.1 Extending for Dual Response

We extend ResourceLink with a dual response pattern that addresses the fundamental tension between LLM reasoning requirements and scalable data access. This pattern augments tool responses with two distinct components: (1) preview data suitable for LLM analysis, and (2) a ResourceLink for out-of-band retrieval of complete datasets.

The extended response schema incorporates both immediate analytical samples and persistent resource references. This design enables LLMs to perform immediate analysis on representative samples while preserving access to complete datasets for comprehensive reporting. The preview data flows through the LLM context window, enabling pattern recognition, validation, and direct question answering for queries where samples suffice. The ResourceLink provides a stable identifier for subsequent pagination and retrieval operations that bypass the context window entirely.

```

1 interface DualResponseToolResult {
2   // Limited results for LLM reasoning
3   results: Array<Record<string, any>>;
4   // Reference to complete dataset
5   resource: ResourceLink;
6   // Query context and constraints
7   metadata: QueryMetadata;
8 }
9
10 interface QueryMetadata {
11   // Total records matching query
12   total_count: number;
13   // ISO 8601 timestamp
14   executed_at: string;
15   // Schema information (optional)
16   columns: ColumnDefinition[];
17   // Resource expiration timestamp
18   expires_at?: string;
19 }

```

Listing 2: Expanded schema for dual response tool results

3.2 Abstraction and Execution

The pattern deliberately abstracts query semantics to support heterogeneous backend systems; e.g. SQL databases, document stores with aggregation pipelines, graph query languages, or custom analytical engines. The server receives structured query specifications from LLM-generated tool calls and returns both preview samples and resource identifiers regardless of underlying implementation. Tools prompt the LLM to generate queries in the format the server can interpret.

A critical consideration involves sampling strategies. While random sampling provides statistical representativeness, servers should faithfully execute LLM-specified query constraints including ordering, limiting, and filtering operations. An LLM requesting `ORDER BY timestamp DESC LIMIT 10` may be specifically seeking recent records for temporal analysis; substituting random samples would violate the query semantics. We recommend implementing preview generation by applying the complete query specification with an additional limit constraint, ensuring the preview represents what the LLM actually requested rather than an arbitrary sample.

3.3 Resource Lifecycle

Servers may implement resource persistence through multiple strategies: (1) storing complete query results, (2) storing query definitions for re-execution, or (3) hybrid approaches with time-bounded

caching. For workloads with frequently-changing data, storing queries enables fresher results on subsequent accesses, albeit with re-execution costs and potential consistency variations.

Developers implementing this pattern must recognize that LLMs generate queries speculatively during iterative refinement processes. As an LLM explores a problem space by testing different aggregations, refining filters, or constructing complex joins, it may invoke query tools dozens of times before producing a final response. Each invocation creates a new resource with an associated `ResourceLink`, yet the user remains unaware of these intermediate queries. This speculative execution model necessitates aggressive lifecycle management to prevent unbounded resource accumulation.

Resource expiration represents the primary lifecycle mechanism. The `expires_at` timestamp in `QueryMetadata` enables automatic garbage collection of queries. Users may wish to preserve specific queries for reuse in dashboards, reports, or recurring analyses - which is supported by lifecycle endpoints outlined in the next section.

3.4 Data Limiting

To prevent context window exhaustion MCP servers should enforce limits on preview responses through query rewriting or post-processing, regardless of LLM-specified parameters. Even when an LLM generates a query with `LIMIT 10000`, the server should cap data at a reasonable threshold (typically 10-100 records) while accurately reporting `total_count` based on the original query.

This dual responsibility requires careful implementation. For SQL backends, servers may wrap queries with additional `LIMIT` clauses to restrict sample size while executing separate `COUNT(*)` queries with identical `WHERE` and `JOIN` logic to compute `total_count`. For aggregation pipelines, servers inject limits as terminal stages while computing counts through parallel execution. For API-backed sources, servers retrieve limited result sets while extracting pagination metadata for totals.

Critically, both preview limiting and count computation must respect multi-tenant isolation filters. The `total_count` reflects records accessible to the authenticated user’s tenant scope, ensuring preview samples and counts remain consistent with eventual full dataset retrieval through `ResourceLink` access.

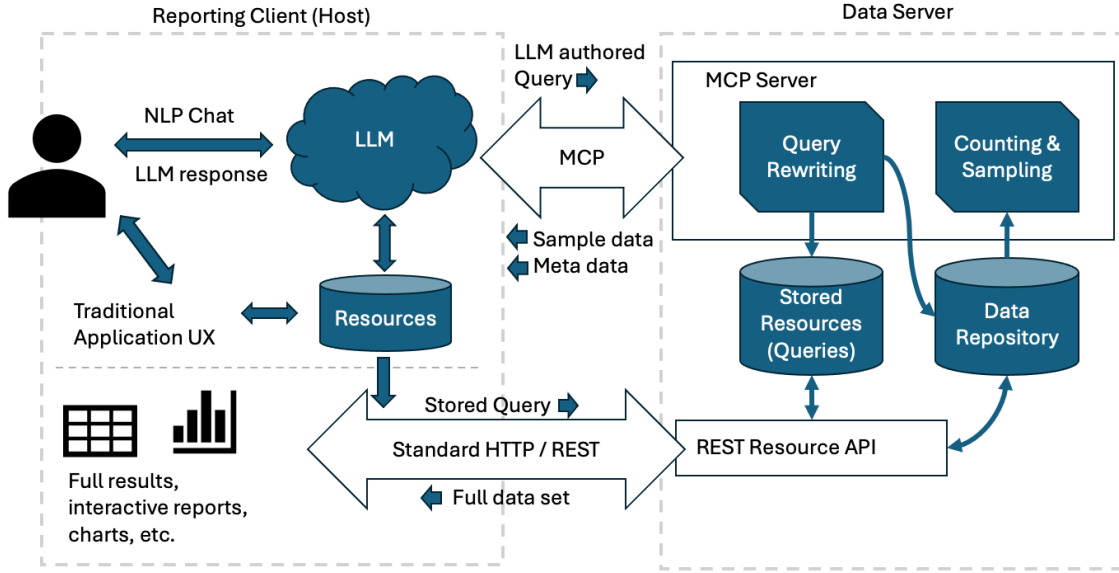


Figure 1: General architecture for dual response pattern integrating LLM tool calls with out-of-band data retrieval. The MCP server returns both preview samples for LLM inference and ResourceLinks for complete dataset access, after altering or augmenting query to provide multi-tenant protection and data sampling. Clients retrieve full data through RESTful endpoints, enabling reporting without consuming context.

3.5 Prompt integration

This pattern integrates naturally with structured output specifications where LLMs return conformant JSON schemas. Tool definitions specify the dual response format as the required return type, constraining LLM behavior through schema validation rather than prompt engineering alone. System prompts should explicitly instruct LLMs on decision logic: analyze preview data for questions answerable from samples; return ResourceLink references for comprehensive reports, exports, or visualizations requiring complete datasets.

4 Out-of-Band Retrieval

While MCP tools facilitate query construction and preview generation, complete dataset retrieval occurs through RESTful HTTP endpoints that bypass the LLM context entirely. This architectural separation enables client applications to implement sophisticated reporting interfaces; e.g. paginated tables, interactive visualizations, bulk exports - without consuming context window resources. The MCP server exposes a consistent resource URL prefix (e.g., `https://server.com/resources/`) that clients

discover through server capability negotiation, discussed below.

4.1 Metadata Fetch

Client applications retrieve current resource metadata through HTTP GET requests to the resource endpoint:

```
1 GET /resources/{resourceId}
2 Authorization: Bearer {token}
```

Listing 3: Fetching resource metadata

The metadata response provides essential information for client-side rendering and validation, and is the same schema provided by the MCP tool response:

```
1 interface QueryMetadata {
2   total_count: number;
3   executed_at: string;
4   columns: ColumnDefinition[];
5   expires_at?: string;
6 }
```

Listing 4: Resource metadata response

The `total_count` field may differ from the original preview response if the underlying data source

has changed, enabling clients to detect stale references. The status field allows graceful handling of expired or processing resources before attempting data retrieval. Servers may update `expires_at` timestamps on access to extend lifetimes for frequently-used queries.

4.2 Data Retrieval

Full dataset access occurs through HTTP POST requests with pagination and sorting parameters:

```

1 POST /resources/{resourceId}
2 Authorization: Bearer {token}
3 Content-Type: application/json
4
5 {
6   "offset": 0,
7   "limit": 1000,
8   "sort": {
9     "field": "timestamp",
10    "order": "desc"
11  }
12 }
```

Listing 5: Retrieving full resource data

Servers return structured responses with pagination metadata enabling sequential page retrieval:

```

1 interface DataResponse {
2   total_count: number;
3   returned_count: number;
4   offset: number;
5   data: Array<Record<string, any>>;
6
7   pagination: {
8     has_next: boolean;
9     has_previous: boolean;
10    next_offset: number;
11  };
12 }
```

Listing 6: Full resource data response

Note that this pattern diverges from the MCP ResourceLink specification which suggests opaque cursor-based pagination where servers return encoded position tokens (e.g., `next_cursor: "eyJpZCI6MTIzNH0="`). While cursors prevent inconsistencies when underlying data changes between requests, offset-based pagination offers superior developer ergonomics through (1) direct page access without sequential traversal, (2) stateless server implementation, (3) natural mapping to conventional UI patterns ("showing 1-100 of 5,000") and (4) immediately readable URLs for debugging.

4.3 Resource Lifecycle Management

Clients manage resource persistence through HTTP PUT and DELETE operations. The "pinning" operation removes automatic expiration, converting ephemeral queries to persistent artifacts. The effect is the removal of the `expires_at` timestamp, allowing indefinite retention until explicit deletion:

```

1 PUT /resources/{resourceId}
2 DELETE /resources/{resourceId}
```

Listing 7: Saving and deleting a resource

This operation typically occurs when users explicitly save queries, incorporate them into dashboards, or schedule recurring reports. Conversely, explicit deletion through HTTP DELETE enables immediate cleanup of unwanted resources without waiting for expiration timers. The reader can also explore utilizing PATCH for edits where applicable.

5 Multi-Tenant Access

Implementing secure multi-tenant query access requires careful data isolation. We identify five distinct patterns with specific trade-offs for MCP server implementations.

Pre-filtering and Stage Whitelisting: Frameworks like MongoDB enable runtime query modification by injecting tenant-specific `t$match` stages before user operations. Servers maintain mappings of collections to tenant-identifying fields and automatically prepend isolation filters. This requires whitelisting permitted aggregation stages—excluding `$out`, `$merge`, and write operations—while handling `$lookup` stages to prevent cross-tenant joins.

Row-Level Security and Read-Only Connections: PostgreSQL and similar databases support native row-level security (RLS) policies that automatically filter query results based on session context [10]. MCP servers establish connections with tenant-specific credentials, where database policies restrict visibility to authorized rows. Combined with read-only connections, this delegates security enforcement to the database engine. This proves particularly robust though it requires per-tenant connection pooling [10].

SQL Abstract Syntax Tree Rewriting: For systems lacking native RLS, SQL parsing and rewriting provides similar protections. Servers parse LLM-generated SQL into abstract syntax trees, inject tenant predicates into WHERE clauses, and validate table access against whitelists [11]. This enables fine-grained control but requires sophisticated

parsing logic to handle dialect variations and complex subqueries.

View-Based Simplification: Restricting MCP tools to pre-defined database views rather than direct table access naturally encapsulates security logic, denormalizes schemas for LLM comprehension, and mitigate injection attacks. This can simplify both security implementation and LLM prompt engineering.

API-Mediated Access: Exposing RESTful endpoints instead of direct query interfaces maximizes security control but constrains LLM analytical capabilities, limiting users to pre-anticipated access patterns.

Authentication and Authorization: OAuth 2.0 authenticates MCP tool invocations and authorizes REST requests to resource endpoints. The Bearer token should encode or allow derivation of authorization metadata: tenant identifiers, resource scope, permission levels. This rich context enables the multi-tenant strategies described.

```

1 {
2   "methods": {
3     "metadata": {
4       "method": "GET",
5       "path": "/{id}"
6     },
7     "data": {
8       "method": "POST",
9       "path": "/{id}",
10      "accepts": ["offset", "limit", "sort"]
11    },
12    "save": {
13      "method": "PUT",
14      "path": "/{id}"
15    },
16    "delete": {
17      "method": "DELETE",
18      "path": "/{id}"
19    }
20  }
21 }

```

Listing 9: REST endpoing specification

6 Discovery

MCP servers supporting out-of-band resource access advertise capabilities through two complementary mechanisms. During MCP initialization, servers may declare resource endpoint support. This minimal advertisement indicates: (1) tools return dual responses with preview data and ResourceLinks, and (2) where REST endpoints for resource access are located. Clients can construct resource URLs by appending ResourceLink URIs to the baseUrl.

```

1 {
2   "protocolVersion": "2025-06-18",
3
4   "capabilities": {
5     "tools": {},
6     "resources": {
7       "resourceLinks": {
8         "dualResponse": true,
9         "baseUrl": "https://server.example.com/resources"
10      }
11    }
12  }
13 }

```

Listing 8: MCP initialization

Following OAuth 2.0’s .well-known pattern [7], servers may expose complete operational metadata at /.well-known/resource-link-service found at the baseUrl defined in the MCP initialization exchange.

7 Agent and Tool Prompting

LLM Output Schema Enforcement: Clients use structured output schemas to constrain LLM responses, ensuring resource links from tool calls are returned alongside text responses. Prompts instruct the LLM to: extract `uri`, `name`, and `contentType` from `resource_link` objects; analyze preview data when `total_count ≤ results.length`; and include resources links when `total_count` exceeds preview size or users request artifacts.

Derivative Artifacts: Additional MCP tools can create visualizations or dashboards by accepting resource identifiers as inputs and generating new resources. The LLM invokes these sequentially - querying data first, then passing resourceIds to visualization tools.

MCP Server Tool Design Patterns: Effective MCP servers implement progressive schema discovery through hierarchical tools: collection/Entity discovery tools with fuzzy search for natural language queries; Property/field schema tools providing detailed metadata with batch retrieval; query tools returning `DualResponseToolResult` structures; and help tools embedding query patterns.

8 Conclusion

This paper describes a dual-response pattern for MCP ResourceLink implementations, addressing fundamental tensions between LLM context constraints and enterprise reporting requirements. By returning samples for LLM reasoning and resource links for out-of-band data retrieval, we enable query construction while preserving iterative refinement necessary for semantic correctness. Our complementary patterns - multi-tenant isolation strategies, resource lifecycle management, and progressive discovery - establish an architectural framework for production reporting systems where natural language interfaces can drive query generation while result rendering occurs outside the model's context.

Future work centers on community-driven standardization of these patterns. We envision formal specification through MCP enhancement proposals or dedicated RFCs that enable interoperability. Standardized discovery mechanisms, consistent REST endpoint contracts, and shared authentication patterns would accelerate ecosystem development and reduce implementation fragmentation.

References

- [1] Yaniv Albo, Joel Lanir, Peter Bak, and Sheizaf Rafaeli. Visualization requirements for business intelligence analytics: A goal-based, iterative framework. In *IEEE 20th International Enterprise Distributed Object Computing Workshop (EDOCW)*, pages 1–10, 2016.
- [2] Anthropic. Model context protocol specification. <https://spec.modelcontextprotocol.io/>, 2024. Accessed: 2025-10-06.
- [3] Arian Askari, Christian Poelitz, and Xinye Tang. Magic: Generating self-correction guideline for in-context text-to-SQL. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(22):23433–23441, 2025.
- [4] Hsinchun Chen, Roger H. L. Chiang, and Veda C. Storey. Business intelligence and analytics: From big data to big impact. *MIS Quarterly*, 36(4):1165–1188, 2012.
- [5] Feyza Duman Keles, Pruthuvi Mahesakya Wijewardena, and Chinmay Hegde. On the computational complexity of self-attention. In Shipra Agrawal and Francesco Orabona, editors, *Proceedings of the 34th International Conference on Algorithmic Learning Theory*, volume 201 of *Proceedings of Machine Learning Research*, pages 1–23. PMLR, 2023.
- [6] In Gim, Seung-seob Lee, and Lin Zhong. Asynchronous LLM function calling. *arXiv preprint arXiv:2412.07017*, 2024.
- [7] Dick Hardt. The OAuth 2.0 authorization framework. RFC 6749, Internet Engineering Task Force, 2012.
- [8] Quinn Leng, Jacob Portes, Samuel Havens, Matei A. Zaharia, and Michael Carbin. Long context RAG performance of large language models. In *NeurIPS 2024 Workshop on Adaptive Foundation Models: Evolving AI for Personalized and Efficient Learning*, 2024.
- [9] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024.
- [10] X. Liu et al. A survey of text-to-SQL in the era of LLMs: Where are we, and where are we going? *IEEE Transactions on Knowledge and Data Engineering*, 37(10):5735–5754, 2025.
- [11] Shariq Rizvi, Alberto Mendelzon, S. Sudarshan, and Prasan Roy. Extending query rewriting techniques for fine-grained access control. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, pages 551–562, 2004.
- [12] Jinwook Seo and Ben Shneiderman. A rank-by-feature framework for interactive exploration of multidimensional data. *Information Visualization*, 4(2):96–113, 2005.
- [13] Ralph H. Sprague. A framework for the development of decision support systems. *MIS Quarterly*, 4(4):1–26, 1980.
- [14] Hanchen Xia, Feng Jiang, Naihao Deng, Cunxiang Wang, Guojian Zhao, Rada Mihalcea, and Yue Zhang. r^3 : This is my SQL, are you with me? a consensus-based multi-agent system for text-to-SQL tasks. In *Proceedings of the 4th Table Representation Learning Workshop*, pages 34–46, Vienna, Austria, July 2025. Association for Computational Linguistics.
- [15] Ruiyuan Zhang, Chrysanthi Kosyfaki, and Xiaofang Zhou. Data-aware Socratic query refinement in database systems. *arXiv preprint arXiv:2508.05061*, 2025.