ISSN 2278-3091
Volume 15, No.2, March – April 2026
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse011522026.pdf
https://doi.org/10.30534/ijatcse/2026/011522026

Automation of SQL Query Processing and Generation Using a RAG Model in a Cloud Environment

Mukayev Timur
Master's Degree, University of Bristol, Bristol, United Kingdom
timurmukayev@rambler.ru

Received Date: February 5, 2026 Accepted Date: March 12, 2026 Published Date: April 06, 2026

ABSTRACT

This article examines the architectural and applied aspects of automating SQL query generation using Retrieval-Augmented Generation in a cloud environment. The evolution of language models from Seq2Seq architectures to large language models such as T5, GPT, and PaLM is analyzed, with emphasis on their application to natural language–to–SQL transformation tasks. The importance of contextual retrieval mechanisms for improving generation accuracy and adapting to the structures of specific databases is highlighted. Using Google Cloud Platform as a case study, the paper demonstrates the feasibility of building scalable and load-resilient infrastructures that support integration with analytical platforms. Practical application scenarios in BI environments are also considered, along with discussions of generation quality, system performance, and potential resource optimization strategies for deploying such systems.

Key words : SQL, query generation, Retrieval-Augmented Generation, large language models, Google Cloud Platform, BI environments, cloud technologies.

1. INTRODUCTION

In the context of the rapid growth of data volumes and the increasing complexity of business analytics processes, the automation of interactions with relational databases has become particularly relevant. The traditional approach based on manual SQL query writing requires substantial time and expert resources, which limits decision-making speed and increases the workload of analytical teams. At the same time, the widespread adoption of large language models (LLMs), capable of generating syntactically and semantically correct text, has opened new opportunities for intelligent natural-language query processing. One promising direction is the use of the Retrieval-Augmented Generation (RAG) architecture, in which generative models are complemented by external information retrieval mechanisms to improve the accuracy and contextual relevance of generated SQL queries.

The aim of this study is to analyze the RAG architecture in the context of automatic SQL query generation and to examine architectural and implementation aspects of its deployment using Google Cloud Platform (GCP) services. The paper examines the architectural characteristics of the approach, demonstrates integration capabilities with business intelligence environments and analytical platforms, and presents practical application examples that illustrate the potential of this technology for automating analytical operations and improving data accessibility for users without SQL expertise.

2. THEORETICAL FOUNDATIONS OF SQL GENERATION USING LLMS

To understand the processes of automatic database query generation using language models, it is first necessary to define SQL (Structured Query Language) as a formal language designed for data management in relational databases [1]. It allows for data retrieval, filtering, aggregation, and modification using declarative statements without the need to know the underlying execution mechanisms of database operations. SQL is not considered a general-purpose programming language, yet the latest empirical data show that its usage is at a steady level, with some periods showing an increase in its use among developers. This trend reflects the stable and continuing role of SQL in modern data-oriented information systems (figure. 1).

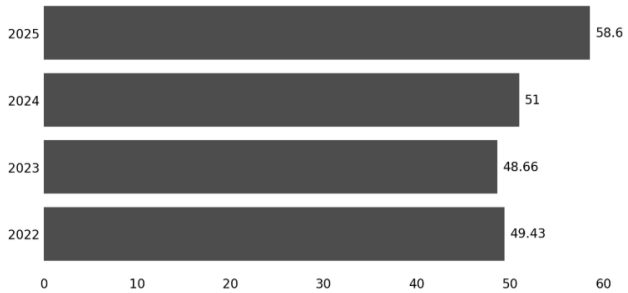

Figure 1: Dynamics of SQL Usage among Developers in 2022–2025, % of Respondents [2]

The obtained data confirm the sustained demand for SQL in professional environments, which necessitates a more detailed examination not only of the language itself but also of its core mechanism of practical application – SQL queries that enable the formalization and execution of data operations in relational databases. To systematize their key properties and to identify characteristics relevant to the automatic generation of queries using language models, the technical characteristics of SQL queries are considered (table 1).

Table 1: Technical Properties of SQL Queries Relevant to Automated Generation [3, 4]

Property	Technical description	Implications for LLM-based generation
Declarative semantics	Query specifies the desired result set without defining execution procedures.	Enables direct mapping from natural language intents to query structure.
Schema coupling	Query correctness depends on table structures, column types, and relational constraints.	Requires explicit schema grounding during generation.
Deterministic execution	Given a fixed database state, identical queries produce consistent results.	Supports reproducibility and automated validation of generated queries.
Compositional syntax	Queries consist of hierarchically nested clauses and expressions.	Allows stepwise or modular query construction.
Join dependency	Query semantics rely on explicit or implicit join conditions between relations.	Increases risk of semantic errors without relational context.
Context sensitivity	Query validity depends on business rules and integrity constraints.	Necessitates retrieval of external contextual information.
Strict syntactic constraints	SQL follows a rigid grammar with limited tolerance for deviations.	Increases the cost of syntactic errors in generative models.

The technical properties of SQL queries discussed above determine the complexity of their automatic generation and allow this problem to be classified as a **semantic parsing task**, which involves transforming unstructured natural language input into formal executable code that conforms to the syntax of relational query languages.

Historically, this task was addressed using **sequence-to-sequence (Seq2Seq) models** based on the encoder–decoder architecture, in which the input text was transformed into a latent representation and subsequently decoded into an SQL structure. However, such models, particularly in their early implementations, exhibited limited generalization capability, high sensitivity to the structure of the input text, and low robustness to changes in the target database schema.

The transition to **LLMs,** such as T5, GPT, and PaLM, has significantly expanded the capabilities of automatic SQL generation. Trained on large-scale corpora encompassing diverse text formats, these models have acquired the ability to generalize syntactic structures, semantic relationships, and lexical variations [5]. LLMs also offer advantages in contextual learning, including few-shot and zero-shot settings, which enable adaptation to new tasks without the need for rigid fine-tuning on task-specific datasets (table 2).

Table 2: Theoretically Relevant Properties of Llms for SQL Generation

Property	Technical description	Implications for SQL generation
Transformer-based architecture	Self-attention mechanisms model long-range dependencies in token sequences.	Enables coherent construction of multi-clause SQL queries.
Large-scale pretraining	Training on heterogeneous text and code corpora.	Provides implicit knowledge of SQL syntax and common query patterns.
Contextual learning capability	Support for zero-shot and few-shot inference.	Allows adaptation to new database schemas without task-specific fine-tuning.
Semantic generalization	Ability to map diverse natural language expressions to shared representations.	Reduces sensitivity to linguistic variability in user inputs.
Autoregressive sequence generation	Token-by-token probabilistic decoding.	Supports stepwise query generation but introduces cumulative error risk.
Limited schema awareness	No inherent access to concrete database schema or constraints.	Requires external schema grounding mechanisms.
Probabilistic output modeling	Output tokens sampled from probability distributions.	May lead to syntactic or semantic inconsistencies in generated queries.

However, even with high-quality generation, the issue of properly aligning the generated SQL query with the requirements of a particular database remains relevant. The most common errors fall into two categories: **syntactic errors** (incorrect query structure or incompatibility with a specific SQL dialect of the target DBMS) and **semantic errors** (incorrect selection of tables, fields, or aggregate functions, as well as flawed filtering or join logic).

To address these shortcomings, **the architectural technique of RAG is of utmost importance**. In this technique, the generative model is provided with not only the original query from the user but also other context-related information, such as descriptions of the database schema, field descriptions, business glossaries, and examples of queries related to the user's current intent. This has a significantly positive effect on semantic accuracy and ensures that the generated output is aligned with the details of a particular application domain. The retrieval process acts as a dynamic knowledge source, addressing the shortcomings of static training and increasing the context available to the model during generation. Consequently, the incorporation of external information into LLMs changes them from general-purpose language tools to domain-specific assistants for data interaction, with the ability to generate SQL queries that are very much aligned with the user's needs and the details of the target database.

3. RAG: PRINCIPLES AND VARIATIONS

The RAG architecture represents a hybrid approach to text generation in which a language model is augmented with an external information retrieval mechanism [6]. Unlike traditional generative models that rely exclusively on parameters learned during training, RAG incorporates

retrieved context – documents or text fragments extracted from an external store based on the user query and supplied to the language model at generation time. This method also assists in overcoming limitations due to the amount of training data, reducing hallucinations, and improving accuracy, especially when performing tasks that demand specific and/or updated knowledge, like generating SQL queries specific to a certain database (figure 2).

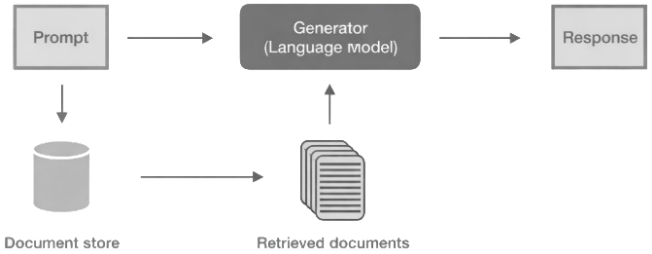

Figure 2: RAG Architecture Using an External Document Store

The basic RAG workflow consists of a sequence of interconnected stages illustrated in fig. 2. The user query (prompt) serves as the starting point of the process and is used both to query the external document store and to construct the input to the language model. During the retrieval stage, relevant textual fragments are searched for in the document store, after which the retrieved documents are provided to the language model as additional contextual input. Based on the combined input – consisting of the original prompt and the retrieved context – the generator produces the final response. This configuration can be applied in both open-domain scenarios and domain-specific environments, including corporate knowledge bases, database schema descriptions, technical documentation, and collections of example SQL queries.

To guarantee the optimal functioning of the retrieval stage in the RAG pipeline, various strategies for information retrieval can be used, each of which is based on different mechanisms for relevance estimation, as well as different trade-offs in terms of precision, stability, and computational cost. The selection of the strategy for information retrieval has a direct impact on the quality of the contextual information fed to the language model, as well as on the correctness of the generated SQL queries. The most commonly used retrieval strategies and their technical characteristics are summarized in table 3.

Table 3: Retrieval strategies in RAG architectures [7, 8]

Retrieval strategy	Technical description	Advantages	Limitations
Top-k semantic retrieval	Selection of the top k documents based on cosine similarity between query and document embeddings.	High retrieval speed; simple implementation; effective for semantically close queries.	Sensitive to embedding noise; limited handling of ambiguity and domain-specific terminology.
Lexical retrieval (BM25, TF-IDF)	Keyword-based matching using term frequency and inverse document frequency statistics.	Interpretable results; robust to exact term matching.	Poor performance on paraphrased or semantically implicit queries.
Reranking with cross-encoders	Secondary relevance scoring using joint query–document encoding with cross-attention.	High precision; improved semantic alignment.	High computational cost; limited scalability.
Hybrid retrieval	Combination of semantic embedding-based search and lexical matching.	Balanced precision and recall; improved robustness to linguistic variation.	Increased architectural complexity; higher tuning overhead.

In addition to retrieval strategy selection, the effectiveness of the RAG architecture strongly depends on **prompt engineering**, particularly on how inputs to the generative model are structured. For SQL generation, prompts typically combine the user query with retrieved schema information and example queries. The use of structured templates with explicit instructions improves model stability and control, while chain-of-thought prompting can further enhance performance in complex analytical tasks involving multiple conditions or aggregations.

Thus, the RAG architecture enables an effective combination of the generative capabilities of LLMs with access to external, continuously updated information sources. With proper configuration of the retriever component and well-designed prompt engineering, RAG models demonstrate high accuracy, domain adaptability, and robustness in solving automatic SQL generation tasks under conditions of complex and evolving data structures.

4. USE OF CLOUD TECHNOLOGIES IN RAG IMPLEMENTATION

Cloud infrastructure provides the practical foundation for deploying RAG as a production-ready pipeline, where scalability, manageability, and predictable response latency are critical, particularly for SQL generation in interactive analytics. On GCP, RAG components can be implemented as managed services – including Vertex AI for generation, vector-based retrieval, document storage, and serverless orchestration – reducing infrastructural coupling and operational complexity, as reflected in the Google Cloud reference architecture for RAG with Vector Search (fig. 3).

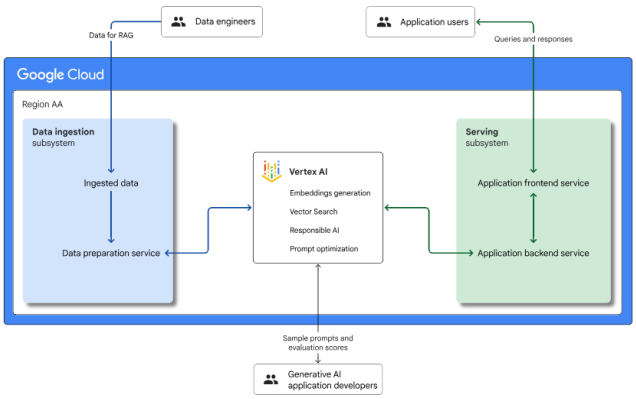

Figure 3: Reference Cloud Architecture for Implementing RAG on GCP [9]

The retrieval component is a key performance bottleneck in the RAG pipeline, as it directly determines both the speed and the quality of the context supplied to the model. This is particularly critical for NL2SQL tasks: even syntactically correct SQL generation fails to produce meaningful results without precise alignment with the database schema and business semantics. Google Cloud benchmarks for Vertex AI Vector Search demonstrate the ability to maintain low latency while achieving high retrieval accuracy at very large scales; reported results include a P95 latency of 9.6 ms at a recall of 0.99 under a load of approximately 5,000 queries per second on an index containing 1 billion vectors [10]. In practical corporate knowledge and metadata scenarios, this performance enables the integration of the retrieval stage into interactive BI workflows without a substantial increase in end-to-end response time, while also supporting hybrid retrieval approaches that combine semantic and lexical signals. According to Vertex AI documentation, queries using dense and sparse embeddings can be completed within milliseconds even when operating over millions or billions of indexed elements.

The quantitative aspects of SQL generation quality are well illustrated by contemporary Text-to-SQL benchmarks. For example, studies published in 2023–2024 report that LLM-based approaches, when combined with appropriately designed prompting and verification procedures, achieve high levels of execution accuracy on the Spider benchmark, a widely used cross-domain dataset. In particular, the DAIL-SQL method reports an execution accuracy of 86.6 % on Spider. At the same time, the more production-oriented BIRD benchmark – comprising 12,751 question–SQL pairs across 95 databases with a total size of 33.4 GB and covering 37 professional domains – reveals a substantial performance drop as task realism increases. Specifically, by incorporating real database values, external knowledge, and execution efficiency constraints, the dataset description indicates that ChatGPT achieved only 40.08% execution accuracy compared to a human performance level of 92.96 % [11].

Cloud-based RAG for NL2SQL is therefore considered an effective engineering approach for improving the accuracy and controllability of intelligent access to corporate data by supplying schema descriptions, domain definitions, and query examples to the generative model as explicit contextual constraints. In this study, the capabilities of LLMs and GCP services are analyzed in terms of technical performance (generation reproducibility, latency, and robustness to schema changes) and practical applicability in analytical platforms and BI environments. Within the GCP ecosystem, this scenario is supported by data products such as BigQuery and Gemini, which provide capabilities for SQL generation, explanation, and correction, enabling practical adoption by users without specialized SQL expertise.

The practical relevance of such architectures is supported by corporate deployments in related knowledge access and policy retrieval scenarios, where outcomes depend on retrieval quality and accurate contextual interpretation by generative models. In one Google Cloud–based implementation, a reduction in request handling time and improved efficiency of searching internal policies and procedures were reported [12]. The measurable effect observed in the "retrieval + LLM" configuration is methodologically comparable to NL2SQL-RAG, as both rely on precise context retrieval and its correct transformation into a target action, such as an SQL query or a user-facing response.

The economic efficiency of cloud-based RAG implementations for SQL generation is achieved through a combination of architectural techniques (fig. 4).

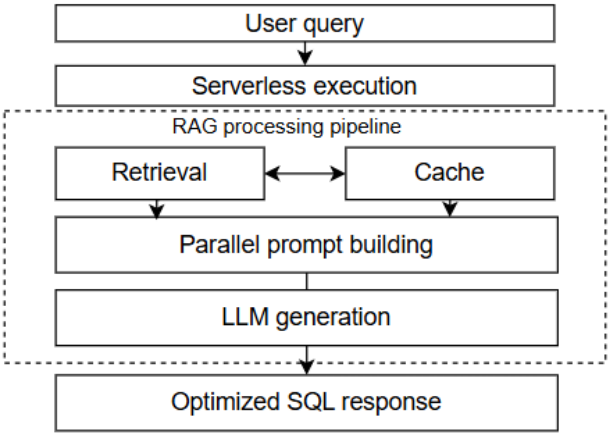

Figure 4: Cost and Performance Optimization Loop of the RAG Pipeline for SQL Generation in a Cloud Environment

It should be emphasized that, in corporate analytics deployments, system robustness is achieved not only through architectural design but also through quality assurance practices, including automated validation of generated SQL queries (syntax checks, test datasets, regression testing, and error monitoring), which aligns with broader findings on the role of test automation in enhancing the effectiveness of business digital transformation [13]. Google Cloud materials on Vertex AI Vector Search also report increased cost-effectiveness – up to a fourfold improvement compared to alternative solutions – while maintaining low latency and high relevance on large-scale indexes, which is critical for high-volume BI workloads.

As a result, cloud technologies, and GCP in particular, enable RAG-based SQL generation to be implemented as a manageable and scalable system tailored to practical intelligent data access scenarios in analytical platforms and BI environments. The retrieval layer provides millisecond-level delivery of domain- and schema-aware context, while the LLM component generates SQL under controlled input conditions, reducing semantic error rates and improving the applicability of NL2SQL in complex and evolving corporate data structures.

5. CONCLUSION

The development and deployment of SQL generation architectures based on RAG in a cloud environment demonstrate high effectiveness for intelligent access to relational data. By combining LLMs with contextual retrieval mechanisms, this approach enables the generation of syntactically correct and semantically relevant SQL queries from natural language input. The use of cloud infrastructure, such as GCP, provides scalability, reliability, and secure data interaction, making the solution suitable for real-world production scenarios, including corporate BI environments.

Integrating such solutions into analytical platforms facilitates the democratization of analytics by enabling data access without explicit SQL expertise, accelerating query generation, and reducing dependence on technical specialists. RAG serves as a flexible architecture adaptable to diverse domains and data structures, enhancing user experience and overall analytical efficiency. Future work may focus on optimizing the retrieval component, incorporating learning from user interactions, and strengthening quality control mechanisms as analytical complexity continues to increase.

REFERENCES

1. M. E. Timadi, E. C. M. Obasi. **Integrating Zero-Trust Architecture with Deep Learning Algorithm to Prevent Structured Query Language Injection Attack in Cloud Database**, *University of Ibadan Journal of Science and Logics in ICT Research*, vol. 13, no. 1, pp. 52-62, 2025.
2. 2025 Developer Survey / Stack Overflow // URL: https://survey.stackoverflow.co/2025/technology (date of application: 24.01.2026).
3. V. Čolić. **Text-to-SQL Systems: Development, Characteristics, Practical Applications, and Future Challenges**, *International Conference "New Technologies, Development and Applications"*, pp. 109-120, 2025.
4. A. Tripathi, V. Patle, A. Jain, A. Pundir, S. Menon, A. K. Singh, D. Herremans. **End-to-End Text-to-SQL with Dataset Selection: Leveraging LLMs for Adaptive Query Generation**, *2025 International Joint Conference on Neural Networks (IJCNN)*, pp. 1-9, 2025.
5. A. Y. Ishankhonov, D. V. Pshychenko, E. A. Mozharovskii, A. S. Aluev. **The Role of LLM in Next-Generation Integrated Development Environments**, *Software Systems and Computational Methods*, no. 4, pp. 140-150, 2024.
6. P. Sharma, S. Bhattarai. **A Review on Retrieval-Augmented Generation: Architectures, Research Challenges, and Emerging Frontiers**, *Journal of Future Artificial Intelligence and Technologies*, vol. 2, no. 4, pp. 616-628, 2026.
7. S. Kiruba, C. S. Anish, R. S. B. Krishna, S. J. Rayen. **A Comprehensive Study - Retrieval-Augmented Generation (RAG)**, *2025 10th International Conference on Smart Structures and Systems (ICSSS)*, pp. 1-8, 2025.
8. A. James, M. Trovati, S. Bolton. **Retrieval-Augmented Generation to Generate Knowledge Assets and Creation of Action Drivers**, *Applied Sciences*, vol. 15, no. 11, p. 6247, 2025.
9. RAG infrastructure for generative AI using Vertex AI and Vector Search / Google Cloud // URL: https://docs.cloud.google.com/architecture/gen-ai-rag-vertex-ai-vector-search (date of application: 25.01.2026).
10. D. Gao, H. Wang, Y. Li, X. Sun, Y. Qian, B. Ding, J. Zhou. **Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation**, *Proceedings of the VLDB Endowment*, vol. 17, no. 5, pp. 1132-1145, 2024.
11. J. Li, B. Hui, G. Qu, J. Yang, B. Li, B. Li, Y. Li. **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*, vol. 36, pp. 42330-42357, 2023.
12. Discover Financial Services. Discover Financial Services Deploys Google Cloud's Generative AI to Transform Customer Service / Discover Financial Services // URL: https://investorrelations.discover.com/newsroom/press-releases/press-release-details/2024/Discover-Financial-Services-Deploys-Google-Clouds-Generative-AI-to-Transform-Customer-Service/ (date of application: 25.01.2026).
13. Y. Drogunova. **The Impact of Test Automation on the Effectiveness of Business Digital Transformation**, *Cold Science*, no. 16, pp. 53-60, 2025.