

## Chatting with your ERP: A Recipe

Jorge Ruiz Gómez<sup>a\*</sup> (jorge.ruiz@itcl.es)  
, Lidia Andrés Susinos<sup>a</sup> (lidia.andres@itcl.es)  
, Jorge Álamo Olivé<sup>a</sup> (jorge.alamo@itcl.es)  
Sonia Rey Osorno<sup>a</sup> (sonia.rey@itcl.es)  
Manuel Luis Gonzalez Hernández<sup>a</sup> (manuel.gonzalez@itcl.es)

<sup>a</sup> Instituto Tecnológico de Castilla y León, Burgos, Spain

**Corresponding author at: Instituto Tecnológico de Castilla y León,  
Burgos, Spain**

Jorge Ruiz Gómez

Instituto Tecnológico de Castilla y León, Burgos, Spain

Tel: +34 947 29 84 71

Email: jorge.ruiz@itcl.es

# Chatting with your ERP: A Recipe

Jorge Ruiz Gómez<sup>a,\*</sup>, Lidia Andrés Susinos<sup>a</sup>, Jorge Álamo Olivé<sup>a</sup>, Sonia Rey Osorno<sup>a</sup>, Manuel Luis Gonzalez Hernández<sup>a</sup>

<sup>a</sup>*Instituto Tecnológico de Castilla y León, Burgos, Spain*

---

## Abstract

This paper presents the design, implementation, and evaluation behind a Large Language Model (LLM) agent that chats with an industrial production-grade ERP system. The agent is capable of interpreting natural language queries and translating them into executable SQL statements, leveraging open-weight LLMs. A novel dual-agent architecture combining reasoning and critique stages was proposed to improve query generation reliability.

*Keywords:* LLMs, Text to SQL, AI Agents

---

## 1. Introduction

Enterprise Resource Planning (ERP) systems are complex software platforms that integrate and manage core business processes across departments such as manufacturing, logistics, finance, and human resources. These systems are essential for coordinating operations, ensuring data consistency, and enabling data-driven decision-making in industrial environments. However, the complexity and technical nature of ERP databases often require users to have specialized knowledge of database schemas to retrieve relevant information.

Recent advancements in artificial intelligence (AI), specially in the field of Large Language Models (LLMs), have created new possibilities for enhancing the interaction between users and industrial manufacturing workflows. This interaction can be developed thanks to LLMs being capable of interpreting human-like language and generating human-like responses, and this generation can be accomplished while having access to complex data sources, bridging the gap between natural language and structured data systems.

This work introduces an intelligent software agent designed to interface with the ERP of a defense manufacturing company. This agent leverages open-weight LLMs to translate user queries expressed in natural language into executable

---

\*Corresponding author.

*Email addresses:* `jorge.ruiz@itcl.es` (Jorge Ruiz Gómez), `lidia.andres@itcl.es` (Lidia Andrés Susinos), `jorge.alamo@itcl.es` (Jorge Álamo Olivé), `sonia.rey@itcl.es` (Sonia Rey Osorno), `jorge.alamo@itcl.es` (Manuel Luis Gonzalez Hernández)

SQL statements, enabling intuitive and efficient access to ERP data. To enhance robustness and reliability, the system incorporates a dual-agent architecture combining reasoning and critique components. This structure allows for iterative refinement and validation of generated queries, significantly improving their accuracy and relevance based on a partial result analysis.

## 2. Related Work

The task of converting Natural Language into SQL, commonly known as Text-to-SQL (Text2SQL), has witnessed rapid development in the latest years thanks to the widespread adoption of LLMs.

Early approaches to solve this problem relied on rule-based systems Li & Jagadish (2014) and supervised learning with DNN and LSTM architectures Hochreiter & Schmidhuber (1997).

Recent attention has shifted towards leveraging LLMs as zero- or few-shot agents capable of performing semantic parsing through prompt-based interaction, thus circumventing the need for task-specific models.

This paradigm shift has catalyzed a new line of research focused on LLM agents, which integrate LLMs with external tools and structured reasoning workflows to improve SQL generation performance without modifying model parameters.

Initial iterations of LLM-based Text-to-SQL demonstrated the surprising effectiveness of zero- and few-shot LLMs such as GPT-3 and GPT-4 in generating syntactically valid and semantically accurate SQL from natural language Rajkumar et al. (2022). Liu et al. (2023) Gao et al. (2023).

These single-pass prompting strategies exhibited limitations in handling complex queries, ambiguous schemas, or large-scale relational contexts, especially in cross-domain or enterprise-level settings.

To address these challenges, recent studies Xie et al. (2024) Cen et al. (2025) have introduced LLM-based agentic architectures that include multi-step reasoning, execution feedback, schema understanding and query refinement, imitating human-like problem solving workflows without the need of fine-tuning the language models.

Additionally, the use of chain-of-thought reasoning and least-to-most prompting also integrates well into the agent paradigm, allowing for hierarchical decomposition of complex user intent into compositional SQL fragments.

This agentic designs that are based on pretrained LLMs seem to offer several advantages. First, the agentic design mitigate the dependency on large annotated datasets, secondly, enhance the results through modular reasoning, and last, increase the adaptability across unfamiliar schemas and unknown domains without the need of retraining the model. However, agentic architectures often face significant challenges, such as high inference latency due to iterative LLM calls and brittle orchestration pipelines that rely heavily on very specific prompts, designed to work well with a handful of models and loss of information, due to the agent not sharing the reasoning behind their decisions and actions.

### 3. Architecture

Our chatbot driven Text2SQL architecture implements a REACT-Based (Reason + Act) conversational agent that can query a database with natural language via a second agent, the SQL Agent. The REACT framework enables the agent to combine iterative reasoning with action execution. The architecture follows a modular design, where reasoning, execution, and evaluation are handled by distinct components. A Human-in-the-Loop (HITL) mechanism is included to ensure clarity and correctness before executing any database queries.

#### 3.1. REACT Agent

The system’s entry point is the REACT agent. Upon receiving user input, the agent initiates a multi-step process: it first interprets the user’s intent, then determines whether the intent can be addressed through a database query. If so, the REACT agent delegates the task to the SQL agent.

The SQL agent processes the query and returns the query results to the REACT agent, which then performs reasoning over the output and answers the user with a natural language response.

#### 3.2. SQL Agent

The SQL Agent transforms natural language queries into optimized SQL queries. The core design of this agent is a two-agent collaborative loop that is designed to generate an optimal and refined query. The first agent in this loop, the **the reasoner**, generates, executes and improves SQL Queries based on the user request and feedback. The second agent, the **critic**, validates the query, the results and provides feedback to refine it. This cooperative process ensures syntactic correctness, semantic adequacy, and usability of the generated queries, even in complex database scenarios.

Both agents share a message state and behave as if they are participating in a collaborative conversation, exchanging structured messages to iteratively improve the SQL query.

##### 3.2.1. SQL Reasoner

The Reasoner agent is responsible for the initial comprehension of user queries and the generation of candidate SQL statements. It implements a self-debugging strategy inspired by Chen et al. (2023)).

To begin, the agent constructs a preliminary SQL query based on the user’s intent and an internal Markdown-based representation of the database schema. It then attempts to execute this initial query.

If execution fails—due to syntax errors, undefined identifiers, or improper joins—the agent analyzes the resulting error message and iteratively refines the query to resolve the issue.

This stage aims to address common low-level errors, such as malformed syntax, invalid field references, and typographical mismatches in table or column names. The Reasoner can generate up to  $N$  attempts per query to produce an executable SQL statement.

At this stage, the agent lacks the ability to assess the semantic adequacy or pragmatic usefulness of the result set. Consequently, it is complemented by a dedicated **Critic agent**.

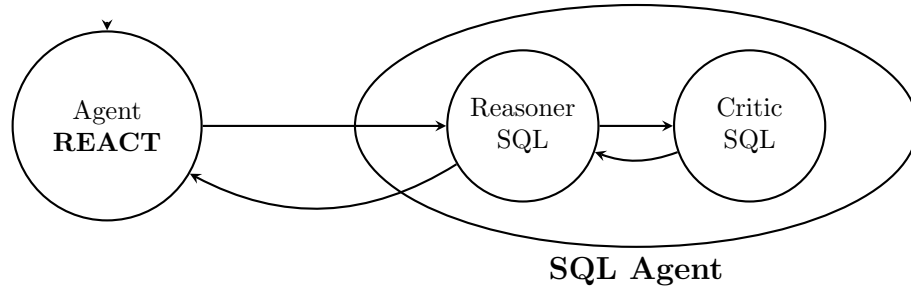
### 3.2.2. SQL Critic

The Critic agent functions as an internal evaluator of the SQL queries produced by the Reasoner. Its role is to assess the quality, correctness, and efficiency of the query output according to several formal and practical criteria.

First, it ensures syntactic validity by verifying that the SQL query is a well-formed SELECT statement. Second, it evaluates semantic appropriateness, determining whether the query responds to the user’s intent by considering table relationships, column semantics, and logical constraints. In this step, the agent also reviews the readability of the query, making sure that the columns are meaningful and related to the user intent. Third, the critic reviews query efficiency, identifying redundant operations, unnecessary joins, or overly complex subqueries that could hinder performance.

Lastly, the agent reviews a subset of the results, checking that they conform the user intent.

When deficiencies are detected, it returns structured feedback to the Reasoner, initiating a refinement cycle. This iterative dialogue between Critic and Reasoner is limited to a maximum of  $M$  rounds.



### 3.3. Database Schema

To ensure context-aware SQL query generation, a structured schema of the database is injected into the SQL Agent. This schema acts as a bridge between the agent and the underlying data structures, improving the agent contextual understanding of the problem.

The schema, encoded as markdown, is organized into two well differentiated sections. The first one, is a handcrafted semantic description of the tables and its relations by an expert developer, while the second one is an auto-generated list of tables and columns.

### 3.3.1. Semantic Description

The first section of the schema is a carefully curated natural language description of the tables, their semantic representation, their most important columns and their relations.

This section helps the agent to connect the concepts expressed by the users in natural language with the internal structure of the database, as no every concept has a direct representation in the schema, and automatic schema descriptions are often ambiguous or technical.

This section is organized into the following subsections:

- **Introduction:** A general description of the role and purpose of the database
- **Concepts:** Explanations of core domain entities and how they are related.
- **Table Summaries:** A concise description of each relevant table, highlighting its purpose, key columns, and relationships with other tables.
- **High-Level Relationships:** A set of manually defined foreign-key-like mappings that are not formally declared within the database.

### 3.3.2. Autogenerated Schema

Complementing the expert-authored layer is an automatically extracted schema that includes a comprehensive enumeration of all tables, columns, data types, key constraints, and known relationships.

For each row, a markdown list item is created with the following fields:

- Column name and data types
- Column description, if available
- Sample values to illustrate typical field contents
- Primary and foreign key indicators
- Explicit table-to-table relationships, if available

## 3.4. Human in the Loop

To ensure safety, interpretability and alignment with the user’s intent, the REACT agent implements a human in the loop interaction mechanism that tries to clarify the user intent.

An LLM analyzes the user intent and tries to prevent incorrect or overly vague natural language prompts that might waste precious compute time building incorrect queries.

If the intent is clear, specific, can be interpreted without confusion and answerable with the information present in the schema, then the intent does not require additional information from the user and a SQL Query is iterative generated, otherwise the user is asked to clarify its intent.

### 3.5. Reasoned Structured Outputs

To ensure compatibility with LLMs that do not support structured outputs or tool calling, while maintaining those functionalities, the system implements a hybrid reasoning-extraction pipeline that decouples the LLM reasoning from the less-creative structured output generation.

In the first step, an LLM is prompted to reason and solve a task. This model is not required to support structured outputs, however, it must present the partial or final results, generated through the reasoning process, within markdown code blocks. This format encourages human-like interaction, a feature which most instruct models excel at.

In a second step, a smaller LLM with structured output support is asked to extract the most relevant reasoned markdown block, matching an specific structured schema.

## 4. Experimental Set-Up

This section outlines the methodology used to execute the SQL Agent designed to interact with a production-ready relational database.

### 4.1. Database

The experiments were performed within a production-grade SQL Server 2017 database connected with an ERP. This SQL server contains 7 tables and 321 columns, as shown in 1.

As the database contains sensitive data, by client request, all the data in this document will be anonymized.

All of these tables are related, but only one foreign key is defined as shown in 2. Only 119 columns are NLP documented, and none of the comments contain references to other fields.

As stated in 3.3.1, an expert developer has crafted a semantic description of the database schema to enhance the agent’s understanding of the data structure.

Table 1: Anonymized table structure with placeholder column counts

Table Name	Number of Columns
T_A	28
T_B	77
T_C	15
T_D	37
T_E	43
T_F	67
T_G	54

Table 2: Anonymized relationship mapping between database tables

Source Table	Related Table	Relationship Type	Linking Key / Comment
<b>T_A</b>	T_B	1 to N	idA
<b>T_A</b>	T_C	1 to 1	idA
<b>T_A</b>	T_D	1 to N (implicit)	via <code>CurrentCode</code> $\rightarrow$ <code>UnitNumber</code>
<b>T_A</b>	T_G	1 to N (implicit)	via <code>CurrentCode</code> $\rightarrow$ <code>UnitNumber</code>
<b>T_B</b>	T_E	1 to 1	idB
<b>T_C</b>	T_E	1 to N	idC
<b>T_D</b>	T_F	1 to N	ID $\rightarrow$ PathID (Foreign Key)
<b>T_G</b>	T_F	1 to 1	<code>UnitNumber</code> $\rightarrow$ <code>UnitNumber</code>

#### 4.2. Expert Validation

The agent was evaluated using a set of twelve distinct queries, each curated by an experienced ERP system user to reflect realistic and domain-relevant information needs. Given the variability in query outcomes, due to the use of aliases or dynamic column selection, each response was validated by an expert.

#### 4.3. LLM Models and Prompts

Due to the data privacy policy, the experiments were performed using eleven different open-weight LLMs. Each model was employed for the tasks of sql-reasoning and sql-critic. Additionally, Qwen2.5-32B was employed to handle multi-turn dialogue, while LLaMA 3.1-8B was used to extract structured outputs from the reasoning process.

All models were deployed using Ollama, with each instance configured to support a context window of 65,536 tokens and a maximum token generation time of three minutes. The evaluation was performed on a high-performance workstation equipped with an AMD Ryzen Threadripper 7960X processor featuring 24 cores, 126 GB of system memory, and three NVIDIA RTX 4090 graphics cards, each with 24 GB of dedicated VRAM.

The prompts have been built around Qwen 2.5 32B, so it is expected that not every model will perform equally well. Differences in model architecture, training data, and instruction-following capabilities can lead to significant variations in how effectively each model interprets and responds to the same prompt, so a disadvantage is expected.

## 5. Results

This section presents a comparative of the performance of the agent across a set of eleven models and eleven questions. 3 summarizes the correctness of each model’s responses, where a checkmark (✓) denotes a correct answer and a cross (✗) indicates an incorrect one. The accuracy column reflects the total number of correct responses out of eleven.

The evaluation encompasses models of varying scales and fields of expertise. Notably, Devstral 24B Q4 model achieved the highest accuracy (10/11), followed closely by Qwen 2.5 32B Q4 (9/11), indicating strong performance in



this benchmark setting. In contrast, several models—including Codestral 22B Q4 and Magistral 24B Q4—did not produce any correct answers, highlighting significant variability in performance across different architectures.

Table 3: Model performance on questions 1–11. ✓ = correct, ✗ = incorrect

Model	1	2	3	4	5	6	7	8	9	10	11	Accuracy
Llama 3.1 8B FP16	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	1/11
Qwen 2.5 7B FP16	✓	✗	✗	✗	✓	✗	✓	✗	✗	✗	✗	3/11
Qwen 2.5 32B Q4	✓	✓	✗	✓	✓	✓	✓	✓	✓	✗	✓	9/11
<b>Devstral 24B Q4</b>	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	<b>10/11</b>
Codestral 22B Q4	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	0/11
Magistral 24B Q4	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	0/11
Deepseek Coder 33B Q4	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	0/11
Gemma 3 27B Q4	✓	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	2/11
Qwen 3 32B Q4	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	2/11

It should be noted that Qwen 3 32B Q4 and Llama 3.3 70B Q4 were also included in the evaluation but were unable to complete inference due to generation time constraints and hardware limitations. Specifically, their memory requirements exceeded 100 GB of VRAM, which led to execution failure due to the three minute request time limit.

## 6. Recipe for a SQL Agent in your ERP

To implement this SQL in an ERP, a few steps must be followed.

The first step involves exposing the internal SQL database of the ERP in read-only mode. This allows the SQL agent to safely query the system without risking data corruption. The agent must have access to the schema and the underlying data.

Once access is established, the most relevant fields—those frequently used or carrying business-critical information—should be clearly documented. This documentation should bridge the semantic gap between the internal field names and the domain-specific concepts they represent.

After documenting the fields, the ERP expert creates a natural language description of the ERP. This description includes the domain-specific concepts that the ERP represents, table relationships, roles, and such.

The last step is LLM selection and configuration. In our design, every single agent mentioned can be configured separately. A two model approach seems to offer low latency with good performance. This approach includes a lightweight model that has the responsibility of extracting structured data from reasoned responses, and a larger one for the rest of the tasks.

## 7. Conclusion and Future Work

This work has explored the feasibility and practical challenges of integrating LLMs into industrial environments for natural language interaction with

structured data systems such as ERPs.

Through the development and iterative refinement of an LLM-based software agent, we have shown that open-weight models can be employed effectively in scenarios involving complex SQL query generation.

However, two significant challenges constrain the agent capabilities.

First, the agent needs manual alignment with the database through a manual natural language description written by an expert. This setup step limits the scalability and usability of the approach, preventing a plug-and-play agent.

Second, the agent has difficulties understanding which columns has to select based on vague prompts. Identifying the columns based solely on the ID format or without detailing what an id refers to relies on the model choosing the correct columns.

Also, the REACT agent sometimes fails to summarize the entire user intent to the SQL Agent, so some details or corrections might be skipped.

Future work will focus on solving both issues. First, a new agent to describes the database is proposed, this agent should reduce or even stop the need of a database expert to write down a natural language description of the database. Second, we will investigate improved methods for intent interpretation and column selection under uncertainty.

## References

- Cen, J., Liu, J., Li, Z., & Wang, J. (2025). Sqlfixagent: Towards semantic-accurate text-to-sql parsing via consistency-enhanced multi-agent collaboration. URL: <https://arxiv.org/abs/2406.13408>. arXiv:2406.13408.
- Chen, X., Lin, M., Schärli, N., & Zhou, D. (2023). Teaching large language models to self-debug. URL: <https://arxiv.org/abs/2304.05128>. arXiv:2304.05128.
- Gao, D., Wang, H., Li, Y., Sun, X., Qian, Y., Ding, B., & Zhang, J. (2023). Text-to-sql empowered by large language models: A benchmark evaluation. URL: <https://arxiv.org/abs/2308.15363>. arXiv:2308.15363.
- Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9, 1735–1780. URL: <https://doi.org/10.1162/neco.1997.9.8.1735>. doi:10.1162/neco.1997.9.8.1735. arXiv:<https://direct.mit.edu/neco/article-pdf/9/8/1735/813796/neco.1997.9.8.1735.pdf>.
- Li, F., & Jagadish, H. V. (2014). Constructing an interactive natural language interface for relational databases. *Proc. VLDB Endow.*, 8, 73–84. URL: <https://doi.org/10.14778/2735461.2735468>. doi:10.14778/2735461.2735468.
- Liu, A., Hu, X., Wen, L., & Yu, P. S. (2023). A comprehensive evaluation of chatgpt’s zero-shot text-to-sql capability. URL: <https://arxiv.org/abs/2303.13547>. arXiv:2303.13547.

- Rajkumar, N., Li, R., & Bahdanau, D. (2022). Evaluating the text-to-sql capabilities of large language models. URL: <https://arxiv.org/abs/2204.00498>. arXiv:2204.00498.
- Xie, W., Wu, G., & Zhou, B. (2024). Mag-sql: Multi-agent generative approach with soft schema linking and iterative sub-sql refinement for text-to-sql. URL: <https://arxiv.org/abs/2408.07930>. arXiv:2408.07930.