Aplicación de grandes modelos de lenguaje en la evaluación y mejora de la experiencia de desarrollador (DX)

**Rubén Rodríguez de la Fuente**

Máster Analista de datos

Área del trabajo final

|  |  |
| --- | --- |
| |  | | --- | |  | |

**Alba Argüelles Folgueira**

**Susana Acedo Nadal**

**Antoni Meseguer Artola**

06/2025

  
Copyright © 2025-RUBÉN RODRÍGUEZ DE LA FUENTE.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

A copy of the license is included in the section entitled "GNU Free Documentation License".

**FICHA DEL TRABAJO FINAL**

|  |  |
| --- | --- |
| **Título del trabajo:** | *Aplicación de grandes modelos de lenguaje en la evaluación y mejora de la experiencia de desarrollador (DX)* |
| **Nombre del autor:** | *Rubén Rodríguez de la Fuente* |
| **Nombre del consultor/a:** | *Alba Argüelles Folgueira* |
| **Nombre del PRA:** | *Nombre y dos apellidos* |
| **Fecha de entrega (mm/aaaa):** | 06/2025 |
| **Titulación:** | *Máster* |
| **Área del Trabajo Final:** | *Analista de datos* |
| **Idioma del trabajo:** | *Castellano* |
| **Palabras clave** | *DX, Conversational analytics, Text-to-SQL* |
| **Resumen del Trabajo (máximo 250 palabras):** *Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.* | |
| Este proyecto propone el desarrollo de un sistema para la consulta de bases de datos sobre experiencia del desarrollador (DX) en lenguaje natural mediante grandes modelos de lenguaje (LLM). El sistema consta de tres componentes principales:  1. Un módulo de configuración y alimentación de una base de datos vectorial con información estructurada del almacén de datos, como el modelo de datos, ejemplos representativos de consultas y o un glosario de negocio.  2. Un módulo de procesamiento de consultas que recupera el contexto relevante de la base de datos vectorial; genera, ejecuta y optimiza consultas SQL de manera iterativa con el LLM; y, por último, maneja errores y optimiza rendimiento  3. Un módulo de visualización que traduce los resultados de consultas en visualizaciones y permite la personalización mediante lenguaje natural.  El sistema está especialmente enfocado para el análisis de datos de DX, aunque la arquitectura propuesta sería adaptable a otros dominios de datos estructurados. No obstante, un sistema de estas características resulta particularmente necesario en el campo de DX, ya que, al abarcar todo el ciclo de vida de desarrollo del software, los datos son muy heterogéneos y multidimensionales, lo que requiere un conocimiento profundo de múltiples sistemas y herramientas, así como de las relaciones entre estos componentes. Además, las necesidades de información varían mucho según el nivel de la organización (desarrolladores, mandos intermedios, directivos) y un sistema de estas características permite cubrir todas estas necesidades de personalización sin depender de analistas dedicados. | |
| **Abstract (in English, 250 words or less):** | |
| This project aims to develop a system for querying developer experience (DX) databases in natural language using large language models (LLM). The system consists of three main components:  1. A configuration module that feeds a vector database with structured information from the data warehouse, such as the data model, representative query examples, and a business glossary.  2. A query processing module that retrieves relevant context from the vector database; generates, executes, and optimizes SQL queries iteratively with the LLM; and, finally, handles errors and optimizes performance.  3. A visualization module that translates query results into visualizations and enables customization using natural language.  The system is specifically focused on DX data analysis, although the proposed architecture would be adaptable to other structured data domains. However, a system of this nature is particularly necessary in the field of DX, since, by covering the entire software development lifecycle, the data is highly heterogeneous and multidimensional, requiring in-depth knowledge of multiple systems and tools, as well as the relationships between these components. Furthermore, information needs vary greatly depending on the level of the organization (developers, middle managers, executives), and a system of this nature makes it possible to cover all these customization needs without relying on dedicated analysts. | |

**Índice**

[1. Introducción 1](#_Toc197277914)

[1.1 Contexto y justificación del Trabajo 1](#_Toc197277915)

[1.2 Objetivos del Trabajo 1](#_Toc197277916)

[1.3 Enfoque y método seguido 2](#_Toc197277917)

[1.4 Planificación del Trabajo 3](#_Toc197277918)

[1.5 Breve sumario de productos obtenidos 3](#_Toc197277919)

[1.6 Breve descripción de los otros capítulos de la memoria 3](#_Toc197277920)

[2. Estado del arte 4](#_Toc197277921)

[2.1 Marcos analíticos para la medición de DX 4](#_Toc197277922)

[2.2 Herramientas comerciales 4](#_Toc197277923)

[2.3 Integración con modelos de lenguaje 5](#_Toc197277924)

[3. Solución propuesta 6](#_Toc197277925)

[3.1 Métricas seleccionadas 6](#_Toc197277926)

[3.2 Diseño de la solución 6](#_Toc197277927)

[3.3 Implementación de la solución 7](#_Toc197277928)

[3.4 Presentación de la solución 7](#_Toc197277929)

[4. Evaluación de la solución 8](#_Toc197277930)

[5. Líneas de investigación futuras 8](#_Toc197277931)

[6. Conclusiones 8](#_Toc197277932)

[7. Glosario 9](#_Toc197277933)

[8. Bibliografía 10](#_Toc197277934)

[9. Anexos 11](#_Toc197277935)

**Lista de figuras**

**No se encuentran elementos de tabla de ilustraciones.**

# 1. Introducción

## 1.1 Contexto y justificación del Trabajo

El desarrollo de software es una disciplina capital para muchas empresas, ya sea porque es su actividad principal (p. ej., Microsoft, Google) o porque resulta clave para su actividad principal. Amazon es una empresa de comercio electrónico, pero para el desarrollo de esta actividad depende de software personalizado que desarrolla internamente. No es de extrañar, por tanto, que haya mucho interés en medir la experiencia de desarrollador (DX) con el fin de encontrar oportunidades de mejora que reviertan en el negocio.

No obstante, dada la complejidad del proceso de desarrollo de software, nos encontramos con una serie de retos únicos:

* Necesidad de integrar datos de muy diversas fuentes (registros de sistemas ERP, sistemas de control de versiones, sistemas de integración continua, encuestas de satisfacción internas).
* Dificultad para encontrar un grupo reducido de métricas que representen de forma completa el proceso.
* Variedad de perfiles de usuarios de este tipo de informes, desde mandos intermedios hasta directivos, con necesidades de información muy diferentes.
* Dada la abundancia de métricas y la complejidad de las interacciones entre ellas, la interpretación de los datos requiere un importante esfuerzo analítico adicional que no siempre puede plasmarse en un simple cuadro de mandos.

En este contexto, no es de extrañar una afirmación que repite habitualmente una directiva de mi empresa: “Nos sobran datos, pero nos falta información”. Se entiende que los grandes modelos de lenguaje pueden contribuir a paliar esta situación. Pueden combinarse con bases de datos relacionales para generar visualizaciones personalizadas en función de las necesidades del usuario final y asistir en la interpretación de los datos, y todo ello mediante una interfaz de lenguaje natural. Dada la posible confidencialidad de los datos, se buscará utilizar modelos de lenguaje alojados localmente.

## 1.2 Objetivos del Trabajo

Desarrollar un sistema que permita a usuarios no técnicos realizar análisis complejos sobre datos de DX mediante consultas en lenguaje natural, automatizando la generación de consultas SQL y visualizaciones. De manera más específica:

* Diseñar e implementar una arquitectura para alimentar bases de datos vectoriales con metadatos de almacenes de datos relacionales
* Implementar un sistema de traducción de consultas en lenguaje natural a SQL optimizado
* Crear un sistema de generación automática de visualizaciones basadas en los resultados de consultas
* Evaluar la precisión, eficiencia y usabilidad del sistema
* Documentar las mejores prácticas y patrones para la integración de LLMs con sistemas de bases de datos

## 1.3 Enfoque y método seguido

Solicité a mi empresa autorización para utilizar datos internos reales, pero no ha podido tramitarse a tiempo. Opté entonces por generar datos sintéticos inspirados en el esquema y perfil de la tabla de hechos donde se integran los registros de fuentes como el sistema de control de versiones, la aplicación de gestión de proyectos o la plataforma de despliegue de software.

Una vez desarrollada la aplicación, su rendimiento se evaluará de forma cuantitativa y cualitativa, según se describe a continuación.

Técnicas cuantitativas:

* **Métricas de evaluación**: tiempo de ejecución, tasa de éxito en la generación de SQL válido, precisión de las respuestas (usando datos de prueba como verdad fundamental).
* **Análisis de logs**: recopilación y análisis de logs de uso para identificar patrones de consulta y áreas de mejora.

Técnicas cualitativas:

* **Análisis de contenido**: revisión de las consultas generadas y las visualizaciones producidas para evaluar su calidad y relevancia.

En cuanto a herramientas, se utilizarán las siguientes para desarrollar la aplicación:

* **Modelos LLM**: Llama3 como LLM, desplegado localmente mediante Ollama.
* **Bases de datos vectoriales**: Chroma
* **RAG y visualizacion de datos**: Vanna AI
* **Bases de datos relacionales**: DuckDB para almacenamiento de datos de DX

## 1.4 Planificación del Trabajo

El proyecto constará de las siguientes fases:

1. Revisión de literatura: análisis de literatura existente sobre LLMs aplicados a consultas de bases de datos.
2. Diseño del sistema: definición de la arquitectura y componentes.
3. Implementación: desarrollo de la solución
4. Evaluación: pruebas de rendimiento, usabilidad y precisión.
5. Refinamiento: mejoras basadas en los resultados de la evaluación.
6. Documentación: registro del proceso y resultados.

## 1.5 Breve sumario de productos obtenidos

* Solución funcional para consultar base de datos de DX en lenguaje natural
* Informe de análisis cuantitativo
* Informe de análisis cualitativo
* Prácticas recomendadas

## 1.6 Breve descripción de los otros capítulos de la memoria

En el capítulo 2, Estado del arte realizamos una revisión de la literatura académica relevante en cuanto a medición de DX, herramientas comerciales que ofrezcan métricas de DX y, por último, de las técnicas de conversión de texto a SQL.

En el capítulo 3, Implementación de la solución, presentamos los detalles técnicos para implementar la solución.

En el capítulo 4, Evaluación de la solución, analizamos el rendimiento de la solución desde un punto de vista cuantitativo y cualitativo, teniendo en cuenta parámetros como la precisión de las respuestas, el tiempo de respuesta o la calidad de las visualizaciones.

En el capítulo 5, Líneas de investigación futuras, presentamos aspectos que no hemos podido cubrir en el presente trabajo, pero que nos permitirían desarrollar una solución más robusta.

En el capítulo 6, Conclusiones, hacemos una reflexión crítica sobre el grado de consecución de los objetivos y el aprendizaje obtenido durante el proyecto.

# 2. Estado del arte

A la hora de valorar el estado del arte para este proyecto, tenemos que centrarnos en tres vertientes:

* Marcos analíticos para la medición de DX
* Herramientas comerciales
* Interacción entre bases de datos y grandes modelos de lenguaje

## 2.1 Marcos analíticos para la medición de DX

La evaluación de la experiencia de desarrollador ha evolucionado desde enfoques centrados en la productividad individual hacia modelos más holísticos que consideran tanto el rendimiento como el bienestar del equipo.

Uno de los primeros enfoques extendidos fue el uso de métricas de productividad directa como story points completados o líneas de código escritas. Sin embargo, estudios como los de Forsgren et al. [1] cuestionan la validez de estas métricas como representaciones del impacto real, ya que fomentan la cantidad sobre la calidad y desatienden el trabajo colaborativo.

El marco DORA (DevOps Research and Assessment) introduce cuatro métricas clave que correlacionan directamente con los resultados del negocio: frecuencia de despliegue, tiempo de entrega de cambios, tiempo medio de recuperación y tasa de fallos en los cambios [2]. Estas métricas han sido validadas empíricamente y se han convertido en un estándar de facto en la industria para medir la eficiencia operativa de los equipos de desarrollo.

Como evolución de DORA, el marco SPACE (Satisfaction, Performance, Activity, Communication, Efficiency) [3] aporta una visión multidimensional que incluye factores humanos, sociales y organizacionales. A diferencia de DORA, SPACE no prescribe un conjunto fijo de métricas, sino que sugiere ejemplos y deja margen para la adaptación contextual. Sin embargo, su adopción industrial todavía es incipiente y su implementación puede resultar más compleja por su carácter cualitativo.

Ambos marcos no son excluyentes. Mientras que DORA proporciona indicadores de eficiencia fácilmente operacionales, SPACE permite capturar dimensiones más intangibles como la satisfacción o la colaboración. Una estrategia combinada ofrece una visión más equilibrada de la DX.

## 2.2 Herramientas comerciales

Numerosas plataformas comerciales han surgido para implementar estas métricas y facilitar su adopción. Las empresas deben decidir entre desarrollar soluciones propias o adquirir herramientas existentes, decisión que dependerá del grado de personalización requerido y la madurez técnica de la organización.

Algunas soluciones destacadas incluyen:

* Harness SEI: automatiza la recopilación de métricas DORA desde sistemas de integración y despliegue continuos. Su doble vista (ejecutiva y de equipo) facilita el uso transversal.
* LinearB: Ofrece cuadros de mando en tiempo real y alertas inteligentes. Su orientación está claramente dirigida a líderes técnicos.
* IBM DevOps Velocity: Incluye visualización del flujo de valor, integrando datos granulares como pull requests o bugs.
* Waydev: Añade análisis de colaboración y tiempos de ciclo, especialmente útiles para CTOs y responsables de productividad.
* Google Cloud y Azure DevOps: Permiten visualizar métricas DORA directamente, aunque su valor depende del uso del stack correspondiente.

No obstante, ninguna de estas herramientas incorpora integración con modelos de lenguaje que permitan consultas o análisis en lenguaje natural. Este vacío justifica la propuesta de este trabajo: democratizar el acceso al análisis de DX sin requerir conocimientos técnicos avanzados en SQL o visualización de datos.

## 2.3 Integración con modelos de lenguaje

La interfaz en lenguaje natural con bases de datos estructuradas es un reto clásico conocido como Text-to-SQL. Este proceso implica comprender la intención del usuario, mapearla al esquema de la base de datos y generar consultas SQL correctas y eficientes.

Inicialmente, los enfoques eran simbólicos, basados en reglas, pero eran poco escalables y frágiles ante la ambigüedad del lenguaje natural. Con la llegada del aprendizaje profundo, especialmente de arquitecturas basadas en Transformers, se ha producido un salto cualitativo.

El uso de grandes modelos de lenguaje (LLMs) ha abierto nuevas posibilidades. Según el NL2SQL Handbook [4], los principales enfoques actuales incluyen:

* Prompt engineering: Mejora la precisión mediante plantillas, ejemplos o razonamiento paso a paso (chain-of-thought).
* Fine-tuning: Adapta modelos preentrenados al dominio específico mediante aprendizaje adicional.
* Entrenamiento desde cero: Diseñado para tareas específicas, aunque requiere gran volumen de datos y cómputo.
* Agentes LLM: Descomponen tareas complejas en subtareas, integrando herramientas externas para depuración o verificación de resultados.
* Además, se aplican estrategias de preprocesamiento (por ejemplo, extracción de esquemas o glosarios) y de postprocesamiento (validación sintáctica o ejecución múltiple con votación) para mejorar la robustez.

Para la evaluación de estos sistemas, se emplean conjuntos como Spider, BIRD y BEAVER, que difieren en complejidad, realismo y cobertura. Las métricas incluyen:

* Coincidencia de componentes: Evalúa selectividad, condiciones, agregaciones.
* Coincidencia exacta: Comparación sintáctica entre la consulta generada y la esperada.
* Coincidencia del resultado: Evalúa si ambas consultas devuelven el mismo resultado, aunque con posibles falsos positivos.

El mejor desempeño en BEAVER alcanza un 77% frente al 92% de expertos humanos, lo que refleja tanto el potencial como las limitaciones actuales de estos sistemas. Cabe destacar que ninguno de estos conjuntos representa datos específicos del dominio de DevOps o DX, lo cual representa una oportunidad de innovación para este trabajo.

# 3. Solución propuesta

## 3.1 Métricas seleccionadas

En la solución propuesta, están representadas las métricas del marco DORA (frecuencia de despliegue, tiempo de espera para cambios, tiempo medio de recuperación y tasa de errores de cambio) y se amplían con dos más del marco SPACE: la satisfacción de los desarrolladores y la eficiencia de flujo. Con la satisfacción de los desarrolladores cubrimos la primera dimensión, S - satisfacción y bienestar. Con las métricas DORA, cubrimos la segunda, P – rendimiento. Por último, con la eficiencia de flujo cubrimos la quinta dimensión, E – eficiencia y flujo.

Cumplimos de esta manera con la recomendación de SPACE de centrarnos en tres dimensiones para mejorarlas. Conseguimos así un equilibrio razonable entre el esfuerzo de recopilación de datos y una visión más o menos completa de la actividad de desarrollo del software.

## 3.2 Diseño de la solución

La columna vertebral de la solución es Vanna AI, una herramienta de código abierto desarrollada en Python que permite generar consultas en SQL a partir de preguntas formuladas en lenguaje natural. Soporta la generación aumentada por recuperación (RAG) y nos permite entrenar un modelo con metadatos de nuestra base de datos, como esquemas, documentación y ejemplos de consultas. El entrenamiento puede repetirse periódicamente, a medida que acumulamos nuevo material, de modo que es factible una estrategia de aprendizaje continuo. Vanna AI es una solución madura y con creciente adopción en el ámbito empresarial, como demuestra el hecho de que haya recibido recientemente la certificación Google Cloud Ready – Big Query, con la que Google valida soluciones de *partners* para asegurar una experiencia de usuario óptima.

Para la gestión de LLMs, utilizamos Ollama, una solución que permite ejecutar los modelos localmente, preservando la confidencialidad y seguridad de los datos. Como modelo de lenguaje, optamos por Llama 3, que tiene un buen desempeño en tareas de text-to-SQL pero también es versátil para asumir otras tareas que puedan surgir en el contexto de este proyecto, como la generación de gráficos a partir de los resultados de las consultas.

## 3.3 Implementación de la solución

El código, datos y otros materiales empleados para implementar la solución se han recogido en un repositorio de [GitHub](https://github.com/menpente/devops_llm).

Para asegurarnos de que no haya conflictos entre versiones, generamos en primer lugar un entorno virtual donde instalamos vanna, duckdb y ollama, entre otras librerías. Creamos un fichero requirements.txt mediante pip freeze para registrar todas las librerías y versiones requeridas.

Escribimos y ejecutamos en primer lugar el *script* src/prep/load\_data.py para crear una base de datos de DuckDB y cargar los datos. A continuación, escribimos y ejecutamos el *script* src/prep/explore\_db.py para explorar la base de datos y validar que funcione correctamente.

Con el script src/setup\_vanna.py, gestionamos la configuración y el entrenamiento del modelo. Como material de entrenamiento, utilizamos una DDL, un glosario de negocio y muestras de consultas SQL, que incluimos en el anexo.

Por último, escribimos y ejecutamos src/run\_vanna.py para lanzar la interfaz web del modelo. Se añade código para registrar las interacciones con el modelo y poder recopilar nuevo material de entrenamiento.

A diferencia de las muestras de código incluidas en la documentación de Vanna, optamos por separar el código de la configuración y entrenamiento del modelo del código de ejecución, de modo que no tengamos que repetir todo el proceso cada vez que queramos utilizar el modelo.

## 3.4 Presentación de la solución

Al ejecutar src/run\_vanna.py, lanzamos la interfaz web del modelo. En ella tenemos una sección sobre los datos de entrenamiento, donde podemos revisarlos y modificarlos, si fuera necesario, eliminando ejemplos o añadiendo otros nuevos.

También contamos con una sección de depuración, donde podemos ver seguir los pasos que realiza el modelo en caso de necesitar la información para resolver problemas.

Por último, tenemos la interfaz de chat en la que podemos introducir nuestras consultas. Sobre ella, se muestran unas preguntas de ejemplo. Una vez realizada nuestra consulta, se muestra el código SQL por si queremos corregirlo manualmente. En caso contrario, se ejecuta y nos devuelve la visualización que hayamos solicitado.

Al script para lanzar la interfaz añadimos código de telemetría para poder analizar el rendimiento de la solución.

# 4. Evaluación de la solución

# 5. Líneas de investigación futuras

# 6. Conclusiones

Este capítulo tiene que incluir:

* Una descripción de las conclusiones del trabajo: Qué lecciones se han aprendido del trabajo?.
* Una reflexión crítica sobre el logro de los objetivos planteados inicialmente: Hemos logrado todos los objetivos? Si la respuesta es negativa, por qué motivo?
* Un análisis crítico del seguimiento de la planificación y metodología a lo largo del producto: Se ha seguido la planificación? La metodología prevista ha sido la adecuada? Ha habido que introducir cambios para garantizar el éxito del trabajo? Por qué?
* Las líneas de trabajo futuro que no se han podido explorar en este trabajo y han quedado pendientes.

# 7. Glosario

| **Término** | **Definición** |
| --- | --- |
| **Agente LLM** | Arquitectura compuesta que combina grandes modelos de lenguaje con herramientas externas (como ejecutores de SQL, reparadores de errores o analizadores semánticos) para resolver consultas complejas de forma iterativa. |
| **BEAVER** | Conjunto de datos para la evaluación de sistemas Text-to-SQL, extraído de almacenes de datos empresariales reales, que presenta alta complejidad estructural y semántica. |
| **DORA (DevOps Research and Assessment)** | Marco analítico que define cuatro métricas clave para evaluar el rendimiento de equipos de desarrollo: frecuencia de despliegue, tiempo de entrega de cambios, tiempo medio de recuperación y tasa de fallos. |
| **Fine-tuning** | Técnica de ajuste de modelos preentrenados para adaptarlos a tareas o dominios específicos, mediante entrenamiento adicional sobre datos representativos. |
| **LLM (Large Language Model)** | Modelo de lenguaje basado en aprendizaje profundo y entrenado sobre grandes volúmenes de texto para entender y generar lenguaje natural. Ejemplos: GPT, LLaMA. |
| **NL2SQL / Text-to-SQL** | Tecnología que permite traducir consultas formuladas en lenguaje natural a sentencias SQL que pueden ser ejecutadas sobre una base de datos. |
| **Prompt Engineering** | Técnica de diseño de entradas (prompts) específicas para guiar a un LLM hacia respuestas más precisas, a menudo incluyendo ejemplos o razonamientos intermedios. |
| **RAG (Retrieval-Augmented Generation)** | Técnica de IA que combina modelos generativos con recuperación de información relevante (por ejemplo, desde una base vectorial) para mejorar la precisión de la salida. |
| **SPACE Framework** | Marco propuesto por Microsoft Research para evaluar la experiencia del desarrollador de forma holística. Sus dimensiones son: Satisfacción, Rendimiento, Actividad, Comunicación y Eficiencia. |
| **Spider** | Benchmark ampliamente usado en Text-to-SQL, con múltiples bases de datos relacionales y consultas de complejidad creciente. |
| **Vanna AI** | Herramienta de código abierto que permite generar consultas SQL y visualizaciones a partir de lenguaje natural, utilizando RAG y entrenamiento personalizado con metadatos. |

# 8. Bibliografía

1. Forsgren, N. et al. (2018) *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations.* IT Revolution Press.
2. Graves Portman D., *Are you an Elite DevOps performer? Find out with the Four Keys Project*. Disponible en <https://cloud.google.com/blog/products/devops-sre/using-the-four-keys-to-measure-your-devops-performance>
3. Forsgren, N. *The SPACE of Developer Productivity.* Disponible en <https://queue.acm.org/detail.cfm?id=3454124>
4. <https://cloud.google.com/blog/products/devops-sre/announcing-the-2024-dora-report>

https://bird-bench.github.io/

<https://developer.microsoft.com/es-es/developer-experience>

<https://microsoft.github.io/code-with-engineering-playbook/developer-experience/>

Trummer, Immanuel (2025) Data Analysis with LLMs. Manning.

Chen, S. (2024)

*TableRAG: Million-Token Table Understanding with Language Models.* Disponible en <https://arxiv.org/abs/2410.04739>

Lista numerada de las referencias bibliográficas utilizadas dentro de la memoria. En cada lugar donde se utilice una referencia dentro del texto, hay que indicarla citando el número de la referencia, por ejemplo: [7].

Es muy importante incluir **todas** las referencias utilizadas y citarlas apropiadamente, es decir, incluyendo toda la información necesaria para identificar la referencia. La información mínima que hay que incluir según el tipo de referencia es:

* **Libro:** Autores, Título, Edición (si se tercia) Editorial, Ciudad, Año.
* **Artículo de revista:** Autores, Título, Nombre de la Revista, Número de Página inicial y final, Número de la revista / Volumen, Año.
* **Web:** URL y fecha en que se ha visitado.

# 9. Anexos

DDL

CREATE TABLE synthetic\_data (

change\_failure\_rate FLOAT,

cycle\_time\_in\_days FLOAT,

date DATE,

developer\_satisfaction\_agree\_count SMALLINT,

developer\_satisfaction\_disagree\_count SMALLINT,

developer\_satisfaction\_neutral\_count SMALLINT,

flow\_efficiency FLOAT,

lead\_time\_in\_days FLOAT,

prod\_deployments\_failed\_count SMALLINT,

prod\_deployments\_success\_count SMALLINT,

through\_put FLOAT,

time\_to\_restore\_in\_hours FLOAT,

total\_developer\_satisfaction SMALLINT,

total\_incidents\_resolved SMALLINT,

team\_lead VARCHAR(255)

);

Glosario de negocio

Column Name | Data Type | Description

change\_failure\_rate | FLOAT | Percentage rate of deployment changes that caused failures (0 to 1 scale).

cycle\_time\_in\_days | FLOAT | Time (in days) taken for work items to complete their lifecycle.

date | DATE | Date associated with the record (within 2025 range).

developer\_satisfaction\_agree\_count | SMALLINT | Number of developers who agreed (feedback/survey).

developer\_satisfaction\_disagree\_count | SMALLINT | Number of developers who disagreed (feedback/survey).

developer\_satisfaction\_neutral\_count | SMALLINT | Number of developers who were neutral (feedback/survey).

flow\_efficiency | FLOAT | Flow efficiency metric calculated for work items (can have wide variance).

lead\_time\_in\_days | FLOAT | Number of days from idea to delivery for a work item.

prod\_deployments\_failed\_count | SMALLINT | Number of production deployments that failed.

prod\_deployments\_success\_count | SMALLINT | Number of production deployments that succeeded.

through\_put | FLOAT | Number of work items completed over a period.

time\_to\_restore\_in\_hours | FLOAT | Time (in hours) taken to restore service after incidents.

total\_developer\_satisfaction | SMALLINT | Aggregated score representing overall developer satisfaction.

total\_incidents\_resolved | SMALLINT | Total number of incidents resolved in a given period.

team\_lead | VARCHAR(255) | Username of the team lead responsible for the work or project.

Script de entrenamiento

import pandas as pd

from vanna.ollama import Ollama

from vanna.chromadb import ChromaDB\_VectorStore

class MyVanna(ChromaDB\_VectorStore, Ollama):

def \_\_init\_\_(self, config=None):

if config is None:

config = {}

config["persist\_directory"] = "./vanna\_chroma\_db"

ChromaDB\_VectorStore.\_\_init\_\_(self, config=config)

Ollama.\_\_init\_\_(self, config=config)

def read\_file(filepath):

with open(filepath, 'r') as f:

return f.read()

def main():

vn = MyVanna(config={'model': 'llama3'})

# Connect to DuckDB

vn.connect\_to\_duckdb(url='/Users/rubendelafuente/tfm/data/dora\_llm.duckdb')

# Train on info schema

df\_information\_schema = vn.run\_sql("SELECT \* FROM INFORMATION\_SCHEMA.COLUMNS")

plan = vn.get\_training\_plan\_generic(df\_information\_schema)

vn.train(plan=plan)

# Train on DDL

ddl\_content = read\_file('doc/DDL')

vn.train(ddl=ddl\_content)

# Train on documentation

documentation\_content = read\_file('doc/dictionary')

vn.train(documentation=documentation\_content)

# Train on SQL examples

sql\_examples = pd.read\_csv('test/sample\_SQL\_queries\_and\_results.csv')

for sql\_query in sql\_examples['query']:

vn.train(sql=sql\_query)

print("✅ Setup and training complete!")

if \_\_name\_\_ == "\_\_main\_\_":

main()

Script de consulta

import time

import logging

import os

from setup\_vanna import MyVanna

from vanna.flask import VannaFlaskApp

# OpenTelemetry imports

from opentelemetry import trace

from opentelemetry.sdk.trace import TracerProvider

from opentelemetry.sdk.trace.export import SimpleSpanProcessor, ConsoleSpanExporter

# Setup tracing

trace.set\_tracer\_provider(TracerProvider())

tracer = trace.get\_tracer(\_\_name\_\_)

trace.get\_tracer\_provider().add\_span\_processor(

SimpleSpanProcessor(ConsoleSpanExporter()) # Logs spans to console

)

# Setup file logger for traces

logging.basicConfig(

filename='trace.log',

level=logging.INFO,

format='%(asctime)s - %(levelname)s - %(message)s'

)

# --- SQL Error Logger ---

def log\_bad\_sql(bad\_sql, error\_message):

log\_file = "bad\_sql\_log.csv"

file\_exists = os.path.isfile(log\_file)

with open(log\_file, "a") as f:

if not file\_exists:

f.write("bad\_sql,error\_message,corrected\_sql,notes\n")

f.write(f'"{bad\_sql}","{error\_message}","",""\n')

# --- Safe SQL Execution ---

def safe\_run\_sql(vn, sql\_query):

try:

result = vn.run\_sql(sql\_query)

print("✅ SQL ran successfully!")

return result

except Exception as e:

print(f"❌ SQL Error: {e}")

log\_bad\_sql(sql\_query, str(e))

return None

# --- Traced Vanna Class ---

class TracedVanna(MyVanna):

def ask(self, question: str):

with tracer.start\_as\_current\_span("ask\_question") as span:

start\_time = time.time()

print(f"\n🧠 [TRACE] Asking: {question}")

logging.info(f"ASK | Question: {question}")

span.set\_attribute("user.question", question)

response = super().ask(question)

duration = time.time() - start\_time

print(f"✅ [TRACE] Response in {duration:.2f}s: {response}\n")

logging.info(f"ASK | Response: {response}")

logging.info(f"ASK | Duration: {duration:.2f} seconds\n")

span.set\_attribute("llm.response", response)

span.set\_attribute("duration", duration)

return response

def ask\_sql(self, question: str):

with tracer.start\_as\_current\_span("ask\_sql") as span:

start\_time = time.time()

print(f"\n🧠 [TRACE] Asking for SQL: {question}")

logging.info(f"ASK\_SQL | Question: {question}")

span.set\_attribute("user.question", question)

sql\_query = super().ask\_sql(question)

duration = time.time() - start\_time

print(f"✅ [TRACE] SQL generated in {duration:.2f}s:\n{sql\_query}\n")

logging.info(f"ASK\_SQL | SQL Generated: {sql\_query}")

logging.info(f"ASK\_SQL | Duration: {duration:.2f} seconds\n")

span.set\_attribute("llm.sql", sql\_query)

span.set\_attribute("duration", duration)

# Run SQL safely

result = safe\_run\_sql(self, sql\_query)

return result

# --- Main App Start ---

def main():

vn = TracedVanna(config={'model': 'llama3'})

vn.connect\_to\_duckdb(url='devops\_data.duckdb')

app = VannaFlaskApp(vn)

app.run()

if \_\_name\_\_ == "\_\_main\_\_":

main()