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](#_Toc197026700)

[1.1 Contexto y justificación del Trabajo 1](#_Toc197026701)

[1.2 Objetivos del Trabajo 1](#_Toc197026702)

[1.3 Enfoque y método seguido 2](#_Toc197026703)

[1.4 Planificación del Trabajo 3](#_Toc197026704)

[1.5 Breve sumario de productos obtenidos 3](#_Toc197026705)

[1.6 Breve descripción de los otros capítulos de la memoria 3](#_Toc197026706)

[2. Estado del arte 4](#_Toc197026707)

[2.1 Marcos analíticos 4](#_Toc197026708)

[2.2 Herramientas comerciales 5](#_Toc197026709)

[2.3 Integración con modelos de lenguaje 5](#_Toc197026710)

[3. Solución propuesta 7](#_Toc197026711)

[3.1 Métricas seleccionadas 7](#_Toc197026712)

[3.2 Diseño de la solución 7](#_Toc197026713)

[3.3 Implementación de la solución 8](#_Toc197026714)

[3. Conclusiones 8](#_Toc197026715)

[4. Glosario 9](#_Toc197026716)

[5. Bibliografía 10](#_Toc197026717)

[6. Anexos 11](#_Toc197026718)

**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

Estado del arte,

Descripción de la solución

Evaluación de la solución

Explicación de los contenidos de cada capítulo y su relación con el trabajo en global.

# 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

Como señalan en *Accelerate*, en un primer momento se utilizaron métricas basadas en la productividad, como la velocidad de desarrollo (p. ej. story points por iteración) o el número de líneas de código. Este enfoque está más centrado en la actividad que en su impacto: por ejemplo, un programa de 10 líneas de código puede resolver un problema de negocio de una forma más eficaz que otro de 1000 líneas. Además, se centran más en el individuo que en el equipo o la organización

A continuación, encontramos [DORA](https://dora.dev/guides/dora-metrics-four-keys/) (DevOps Research and Assessment), que identifica cuatro métricas clave para medir el rendimiento de los equipos de desarrollo: frecuencia de despliegue, tiempo de entrega de cambios, tiempo medio de recuperación y tasa de fallos en los cambios. Desde 2014, DORA publica el *State of DevOps Report,* donde se analizan datos de miles de profesionales de todo el mundo con el objeto de clasificar a las organizaciones en niveles de rendimiento y correlacionar las prácticas DevOps con los resultados de negocio (por ejemplo, la estabilidad o la rentabilidad). DORA se ha establecido como un estándar de facto para evaluar DevOps y sirve como referencia a las empresas para mejorar en este apartado. Cabe apuntar, no obstante, que a pesar de su gran utilidad las métricas de DORA no capturan toda la complejidad del entorno de desarrollo y obvian aspectos muy importantes, como por ejemplo la calidad del código.

En 2021, Nicole Forsgren (coautora de DORA) y otros investigadores de Microsoft Research y GitHub proponen [SPACE](https://queue.acm.org/detail.cfm?id=3454124), que busca dar una visión más holística del campo de desarrollo de software. SPACE es un acrónimo de cinco dimensiones: Satisfaction and well-being, Performance, Activity, Communication and Collaboration, Efficiency and flow. SPACE no define un conjunto fijo de métricas para cada dimensión, sino que propone tipos de métricas y da ejemplos. Corresponde a la organización debe elegir en función de su contexto. SPACE es más completo que DORA en cuanto que incorpora factores humanos y culturales. No obstante, está menos adoptado por el momento en la industria y es más difícil de operacionalizar.

DORA y SPACE pueden utilizarse de forma complementaria, tomando DORA para medir la eficiencia operativa y SPACE para evaluar el bienestar y la colaboración.

## 2.2 Herramientas comerciales

Es frecuente el dilema *build versus buy* en empresas que quieren empezar a utilizar estos marcos analíticos. Las herramientas comerciales permiten empezar a explotar estos datos con mayor rapidez, aunque a la larga pueden generar un coste mayor y no siempre tienen la flexibilidad de adaptarse a todas las peculiaridades de cada organización. Detallamos a continuación algunas de las más destacadas:

* Harness Software Engineering Insights (SEI) es una plataforma empresarial para desarrollo de software basado en datos. Se integra con distintas fuentes de integración y despliegue continuos y genera informes automáticos de DORA, tanto en vista ejecutiva como vista de equipo.
* LinearB proporciona cuadros de mando en tiempo real de las métricas DORA y alertas automatizas. Está fundamentalmente orientada a jefes de ingeniería.
* IBM DevOps Velocity cuenta con una función de flujo de valor que permite monitorizar todo el ciclo de desarrollo, dando acceso a elementos individuales como builds, pull requests, bugs y tests. Su plataforma Insights muestra visualizaciones con métricas DORA, enfocadas en equipos de desarrollo.
* Waydev proporciona reportes sobre métricas DORA, pull requests, tiempos de ciclo y colaboración. Diseñada para CTOs y ejecutivos.
* Mención aparte merecen Google Cloud Platform y Azure DevOps. Si bien ofrecen métricas DORA, solo tienen sentido si la empresa despliega en GCP o utiliza el stack de Microsoft.

La elección de una de estas herramientas dependerá del stack tecnológico de la empresa, la madurez técnica y la escala (equipo vs. organización completa).  
  
No se observa, en ningún caso, integración con modelos de lenguaje que puedan asistir al usuario no técnico en la interpretación de los datos.

## 2.3 Integración con modelos de lenguaje

Para habilitar la consulta de bases de datos en lenguaje natural es necesario convertir de texto a SQL (Text-to-SQL). Este proceso implica tres pasos:

* Entender la consulta en lenguaje natural
* Encontrar las tablas, columnas y filas relevantes
* Escribir la consulta en SQL

Entre las dificultades de esta tarea encontramos la posible ambigüedad de la consulta formulada en lenguaje natural, la necesidad de conocimiento especializado sobre el dominio, la complejidad de los esquemas de las bases de datos, como por ejemplo las relaciones entre tablas y columnas, o los posibles problemas de calidad de las bases de datos.

En el [NL2SQL Handbook,](https://arxiv.org/abs/2408.05109) hacen un repaso de la evolución de estas técnicas. Los primeros intentos estuvieron basados en reglas. Resultaban poco adaptables y escalables y se centraban en consultas de una sola tabla. Las redes neuronales mejoraron la gestión de sinónimos y la detección de la intención y permitieron empezar a trabajar con escenarios de varias tablas. El poder generalizador de las redes neuronales dependía del tamaño del modelo y la cantidad de datos de entrenamiento disponibles. En la actualidad se trabaja con grandes modelos de lenguaje, que mejoran sensiblemente la parte de comprensión del lenguaje natural y permiten centrarse en las dificultades específicas de la base de datos.

En [Large Language Model Enhanced Text-to-SQL Generation: A Survey](https://arxiv.org/abs/2410.06011) encontramos las siguientes técnicas para la tarea Text-to-SQL:

* El prompt engineering trata de diseñar prompts que guíen a los LLM para generar consultas SQL más precisas, por ejemplo proporcionando ejemplos o utilizando técnicas como cadenas de pensamiento, donde se intenta generar capacidades de razonamiento complejas mediante pasos de razonamiento intermedios.
* El ajuste fino adapta modelos preentrenados, ya sea de todos los parámetros o *parameter-efficient,* donde solo se modifican algunos parámetros o capas del modelo.
* Entrenamiento específico para la tarea, donde se entrena un modelo desde cero específicamente para esta tarea, con estrategias similares a los LLM, como el uso de Transformers o la combinación de expertos.
* Agentes LLM, donde se combinan varios agentes y herramientas externas. Por ejemplo, un modelo de descomposición parte una consulta compleja en sub-problemas más sencillos antes de generar la consulta final, mientras que un modelo de reparación ejecuta la consulta con herramientas externas y corrige los errores que puedan surgir.
* Se indican también métodos de preprocesamiento (como la identificación de las tablas y columnas más relevantes, o la integración de conocimiento específico de dominio) y posprocesamiento (como la corrección de errores de sintaxis o la repetición de la consulta y selección del resultado más repetido).

Para evaluar los sistemas de Text-to-SQL existen diferentes conjuntos de datos que incluyen preguntas en lenguaje natural y una consulta SQL de referencia. Destacan [Spider](https://paperswithcode.com/sota/text-to-sql-on-spider), que incluye bases de datos relacionales complejas con varias tablas, [BIRD](https://bird-bench.github.io/), que incluye funciones y operaciones SQL que no están presentes en SPIDER, o [BEAVER](https://arxiv.org/html/2409.02038v1), extraído de almacenes de datos corporativos y significativamente mucho más complejo. Entre las métricas de evaluación contamos con:

* Coincidencia de componentes, donde se mide la coincidencia media de los elementos SELECT, WHERE, GROUP BY, ORDER BY, SQL KEYWORDS.
* Coincidencia exacta, que comprueba si la consulta generada es idéntica a la incluida en los datos de referencia.
* Coincidencia del resultado, donde se mide la coincidencia del resultado de la consulta SQL. Evita falsos negativos de coincidencia exacta (es decir, casos en que la consulta SQL consigue un resultado correcto, pero con sintaxis diferente), pero también puede introducir falsos positivos (por ejemplo, consultas que sean semánticamente diferentes, pero devuelvan NULL en ambos casos). El mejor resultado obtenido hasta el momento en BIRD es del 77%, frente al 92% conseguido por especialistas humanos. Tomaremos estos valores como referencia a la hora de valorar el desempeño de nuestra solución.

Aunque estos conjuntos de datos incluyen datos de diferentes dominios, el campo de DevOps no se encuentra representando.

# 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. 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.

# 4. Glosario

Definición de los términos y acrónimos más relevantes utilizados dentro de la Memoria.

# 5. Bibliografía

Forsgren, N. et al. (2018) *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations.* IT Revolution Press.

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>

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>

Forsgren, N. *The SPACE of Developer Productivity.* Disponible en <https://queue.acm.org/detail.cfm?id=3454124>

<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/

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.

# 6. 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.