Automatización de la Migración de Bases de Datos con Abstracciones de LLM

María José Ortín Ibáñez, Fabián Solá Durán, José Ramón Hoyos Barceló, and Jesús García Molina

Universidad de Murcia, Murcia, España
{mjortin,f.soladuran,jose.hoyos,jmolina}@um.es

Resumen La migración de bases de datos es un proceso complejo que requiere transformar el esquema, los datos y, en muchos casos, el código de acceso. La necesidad de soluciones que automaticen total o parcialmente este proceso es creciente. La irrupción de los modelos de lenguaje de gran escala (LLM) y la IA generativa ha revolucionado la forma en que se construyen herramientas software, tanto mejorando la productividad como potenciando nuevas funcionalidades. En este artículo se presenta un trabajo inicial hacia la construcción de una herramienta impulsada por LLM para automatizar la migración de bases de datos, empleando un framework de abstracciones LLM. Se describe un prototipo independiente del modelo usado, centrado en la migración de esquemas y datos, y se discuten los resultados obtenidos con distintos escenarios.

Keywords: migración de bases de datos · LLM · GenAI · automatización · NoSQL

1. Introducción

La migración de bases de datos es una tarea común pero compleja en el ciclo de vida del software, la cual exige un gran esfuerzo para convertir el esquema, los datos almacenados, y el código de las aplicaciones que la usan en los que se ajustan al sistema destino. Por ello, es de gran interés disponer de soluciones que automaticen estas conversiones.

La adopción del *cloud-computing* ha supuesto la aparición de un buen número de herramientas comerciales que automatizan la migración de sistemas relacionales a la nube, entre las que podemos destacar los servicios DMS de Amazon, Microsoft y Google. Algunas de estas soluciones soportan la migración de algún sistema NoSQL, normalmente MongoDB.

La emergencia de los modelos grandes de lenguaje (LLM) y con ellos la Inteligencia Artificial Generativa (GenAI) ha supuesto una verdadera revolución en la automatización de todo tipo de tareas, en especial la producción de software. Los LLMs también están siendo usados para mejorar las prestaciones de las aplicaciones existentes a través de interacciones en lenguaje natural. Un ejemplo, es el uso del LLM Gemini en DMS-Google[1]

[1] https://cloud.google.com/database-migration?hl=en

Con el fin de facilitar la integración de LLM en aplicaciones han surgido recientemente frameworks basados en *abstracciones LLM* (en adelante, *framework LLM*) que proporcionan servicios para apoyar la ingeniería de prompts, el flujo de control o la creación de asistentes, tales como LangChain [2] y LlamaIndex [3]. Un marco teórico sobre estos frameworks ha sido propuesto por Zhong et al. [4].

En este artículo corto se presenta el trabajo inicial de un proyecto de tesis doctoral destinado a abordar la definición e implementación de una estrategia genérica para la automatización de la migración de bases de datos NoSQL y relacionales. Para ilustrar el objetivo se mostrará un prototipo creado con un framework LLM y que es independiente de un LLM particular. Hasta donde sabemos, no han aparecido herramientas de esta naturaleza hasta el momento.

2. Un prototipo de prueba de concepto

En el mencionado trabajo de Zong et al. [4] se propone una jerarquía de 7 niveles de abstracción (Language Model System Interface Model, LMSI). El nivel más bajo representa la interacción directa con la arquitectura y parámetros del LLM, y el nivel más alto corresponde a la interfaz de usuario. La mayoría de frameworks LLM se centran en tres capas: la capa de *Prompting*, que permite introducir un texto en el LLM utilizando una API; la capa de *Prompt Constraint*, para definir una serie de restricciones o reglas sobre los prompts, y la capa de *Control*, que da soporte para bucles y flujos condicionales.

Para construir un prototipo de prueba de concepto, hemos elegido el framework Spring AI [3] que soporta los dos primeros niveles mencionados arriba, *Prompting* y *Prompt Constraint*, así como algunas funcionalidades básicas de la capa de *Control* y de una capa de *optimización* de algunos aspectos de un LLM.

La figura 1 muestra cómo el prototipo creado realiza la migración. Aunque Spring AI soporta la independencia de un LLM concreto, se utiliza la API unificada OpenRouter [4] para reducir el esfuerzo de implementación y mantenimiento. Mientras que OpenRouter sólo necesita un cliente REST, Spring AI necesita uno por LLM usado. Las pruebas se han ejecutado con GPT-4o de OpenAI.

Como se observa en la figura 1, la entrada son 4 prompts destinados a: (1) migrar el esquema de base de datos, (2) validar el esquema generado, (3) migrar los datos, y (4) validar la migración de datos, los cuales se ejecutan en ese orden. Siguiendo buenas prácticas de ingeniería de prompts [1], se ha creado una plantilla para cada uno de los tipos de prompts que deben ser lanzados.

La plantilla de "Migración Esquema" debe ser rellenada con los sistemas origen y destino, parámetros de conexión, esquema de la base de datos origen, y los requisitos de migración del esquema y del acceso a datos. El formato del *Esquema Origen* dependerá del sistema origen (ej., SQL para Postgres o Mongoose para MongoDB), aunque podría utilizarse nuestro lenguaje unificado Athena [2], o incluso ser inferido de los datos por el LLM con supervisión del usuario. Los

[2] `https://www.langchain.com/`
[3] `https://www.llamaindex.ai/`
[4] `https://openrouter.ai/`

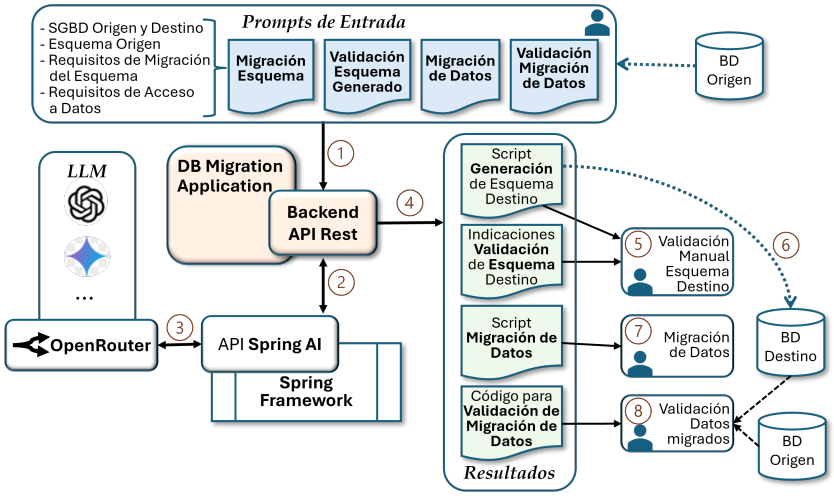

Figura 1. Migración de una base de datos basada en un LLM de propósito general

Requisitos de Migración del Esquema son indicaciones para la creación del esquema destino (ej., el tipo de datos para un campo cuando varias opciones son posibles). Por último, los *Requisitos de Acceso a Datos* proporcionan información como las consultas más frecuentes o el volumen de los datos, con el fin de crear un esquema orientado a optimizar el acceso.

La figura 1 también muestra los trabajos de validación manual que deben realizarse una vez se genera automáticamente el esquema destino y se migra la base de datos con el script generado. Antes de lanzar cada prompt, se ejecuta un *mensaje de sistema* que proporciona al LLM el contexto necesario para realizar la tarea: el rol que va a asumir y una breve descripción de la tarea, indicando de dónde obtener la información necesaria. La memoria de la conversación se recupera y añade como colección de mensajes al prompt. También se utiliza un *mensaje de usuario* que describe la tarea de forma más precisa, haciendo referencia a objetos de la memoria de la conversación.

3. Experimentos y resultados

Hemos ejecutado varios experimentos con distintos casos de prueba: 1) una migración de MySQL a MongoDB con esquema sencillo, 2) una migración de MySQL a PostgreSQL con esquema complejo, 3) una migración de MongoDB a MySQL con esquema complejo, y 4) una migración de MongoDB a PostgreSQL con el esquema del caso 3, añadiendo requisitos para garantizar que los identificadores se migran correctamente para mantener las referencias en el esquema generado. Dado que los LLM son no deterministas, hemos ejecutado cada caso de prueba cuatro veces para comparar resultados. Los ejemplos empleados en

los experimentos y el código de la aplicación están disponibles en un repositorio Github público [5].

Hemos comprobado que organizar la migración en 4 prompts mejora notablemente los resultados frente a un único prompt: el LLM genera información más rica, precisa y específica. La validación manual confirmó que los scripts de creación del esquema de la base de datos destino y de migración de datos eran correctos, completos y consistentes en sus referencias. Cuando la BD origen es relacional, el esquema destino se genera correctamente en todos los casos. En cambio, con BDs de documentos, la migración de agregados puede producir un esquema relacional incompleto: se crean tablas para los agregados, pero sin referencias a la entidad contenedora si no se usa un campo estándar "`_id`" como identificador. Al no existir restricciones como `PRIMARY_KEY` o `FOREIGN_KEY`, el LLM infiere los identificadores por nombre y contexto, lo que no siempre es correcto. Por ello, en el experimento 4 indicamos explícitamente los identificadores, mejorando la generación de claves y referencias.

Las indicaciones generadas por el LLM para validar el esquema son precisas y cubren los aspectos clave. Para la validación de datos, propone consultas equivalentes en las BDs origen y destino, así como otras alineadas con los requisitos de acceso a los datos, verificando así que el esquema destino permite satisfacerlos.

Como líneas futuras, se prevé extender el prototipo hacia la migración completa de aplicaciones, incluyendo la transformación automática del código de acceso a datos, así como evaluar su rendimiento en casos reales y su integración con entornos de desarrollo.

Agradecimientos Este trabajo es parte del proyecto PID2020-117391GB-I00 financiado por el Ministerio Español de Ciencia, Innovación y Universidades y la Agencia Estatal de Investigación (MICIU/AEI /10.13039/501100011033) y ERDF, EU.

Referencias

1. Brousseau, C., Sharp, M.: LLMs in Production: From Language Models to Successful Products. Manning (February 2025)
2. Hernández Chillón, A., Sevilla Ruiz, D., Garcia-Molina, J.: Athena: A Database-Independent Schema Definition Language. In: Advances in Conceptual Modeling - ER 2021 Workshops CoMoNoS, St.John's, NL, Canada. vol. 13012, pp. 33–42 (2021). `https://doi.org/10.1007/978-3-030-88358-4`
3. Parasuraman, B.: Mastering Spring AI: The Java Developer's Guide for Large Language Models and AI Integrations. Apress (2024)
4. Zhong, P.Y., He, H., Khattab, O., Potts, C., Zaharia, M., Miller, H.: A guide to large language model abstractions. Two Sigma (2023), `https://www.twosigma.com/articles/a-guide-to-large-language-model-abstractions/`

[5] https://github.com/modelum/JISBD2025-migration