



**Escuela Superior
de Ingeniería y Tecnología**
Universidad de La Laguna

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Automatización de consultas en lenguaje natural a SQL con GPT-4o

Automation of natural language queries to SQL with GPT-4o

Febe Fariña Aguirre

La Laguna, 24 de mayo de 2024

D. **Jose Luis Roda García**, profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

D. **Ginés León Rodríguez**, responsable del Departamento de Big Data & Data Science de T.I.T.S.A, como cotutor

C E R T I F I C A (N)

Que la presente memoria titulada:

“Automatización de consultas en lenguaje natural a SQL con GPT-4o”

ha sido realizada bajo su dirección por D. **Febe Rubén Fariña Aguirre**.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 24 de mayo de 2024.

Agradecimientos

Me gustaría agradecer, en primer lugar, a mi familia y amigos por el apoyo incondicional durante estos 4 años de carrera. En especial, agradecer a mi pareja por acompañarme durante los buenos y malos momentos.

Luego, agradecer a todos los profesores que me han impartido clase durante mi paso por la universidad, sobre todo a José Luis Roda por su labor de tutor y a la Cátedra Cajasiete Blockchain, Open Data & Big Data por facilitarme los recursos necesarios para desarrollar este trabajo. De la misma forma agradecer a Ginés León, tutor externo de este trabajo, por la aportación de ideas que han hecho este proyecto posible.

Finalmente quiero agradecer a todos los compañeros de clase y amigos que he hecho en la carrera, con los que he trabajado codo con codo para poder superar cualquier desafío.

Agradecer a la “Secta del mod”: Aday, Mario, Adriano, Eduardo, Alexander, Facundo, Fernando, Daniel, Christopher, Pablo, Sergio, Muha y Brayan. Gracias a ellos he pasado muchos buenos momentos y me he sentido acompañado durante estos años.

Licencia



© Esta obra está bajo una licencia de Creative Commons Reconocimiento-NoComercial-SinObraDerivada 4.0 Internacional.

Resumen

La popularidad que han ganado estos últimos años los procesadores de lenguaje natural es innegable. Con la aparición de grandes modelos de lenguaje como GPT, PaLM o LLaMA, ha habido un crecimiento exponencial en el número de herramientas que deciden implementar la inteligencia artificial en sus soluciones. Estos modelos se encargan de procesar el lenguaje humano, etiquetarlo gramaticalmente en base a su contexto, desambiguar y elegir un significado para cada palabra y reconocer cada entidad que forma parte del discurso. Con esto el modelo es capaz de analizar el texto o voz de cualquier fuente y tomar acciones en base a ello.

Por otra parte, nos encontramos completamente inundados en un mar de datos. Intentar buscar un dato en particular o saber cómo está construida la base de datos en una gran organización puede resultar desde difícil hasta imposible.

Es por esto que se plantea este proyecto, una herramienta que, haciendo uso de la tan poderosa herramienta que son los grandes modelos de lenguaje, facilite la asimilación de una gran base de datos. Siendo más precisos, usaremos el modelo GPT-4 para desarrollar una aplicación web capaz de responder consultas hechas en lenguaje natural sobre una base de datos en concreto.

Palabras clave: GPT-4, OpenAI, SQL, Bases de datos, Inteligencia Artificial, Procesamiento de Lenguaje Natural.

Abstract

The popularity that natural language processors have gained in recent years is undeniable. With the appearance of big language models such as GPT, PaLM or LLaMA, there has been an exponential growth in the number of tools that decide to implement artificial intelligence with their solutions. These models are in charge of processing human language, grammatically tagging it based on its context, disambiguating and choosing a meaning for each word and recognizing each entity that takes part in the speech. With this, the model is capable of analyzing text or voice from any source and taking actions based on it.

On the other hand, we are completely flooded in a data sea. Trying to look for a particular data or know how the database is built in a big organization could result from hard to impossible.

This is why this project is proposed, a tool that, making use of the so powerful tool that big language models are, makes it easy to assimilate a big database. Being more precise, we will use the GPT-4 model to develop a web application capable of answering queries made in natural language over a concrete database.

Keywords: GPT-4, OpenAI, SQL, Databases, Artificial Intelligence, Natural Language Processing.

Índice general

Capítulo 1. Introducción	1
1.1 Antecedentes	1
1.2 Problemática	1
1.3 Alcance del proyecto	2
1.4 Fases y desarrollo del proyecto	2
Capítulo 2. Fundamentos teóricos	4
2.1 Modelos de Lenguajes Grandes (LLMs)	4
2.1.1 OpenAI	5
2.2 Bases de datos relacionales	6
2.3 Interfaz de programación de aplicaciones (APIs)	6
Capítulo 3. Tecnologías y arquitectura del prototipo	9
3.1 Tecnologías	9
3.1.1 Langchain	9
3.1.2 Agentes	10
3.1.3 Express	12
3.1.4 PostgreSQL	12
3.2 Arquitectura	12
Capítulo 4. Desarrollo	14
4.1 Datos empleados	14
4.1.1 Base de datos de ejemplo: Películas	14
4.1.2 Matriz de origen destino TITSA Marzo 2024	15
4.2 Funcionamiento de la aplicación	17
4.3 Ejemplos de ejecución	24
4.3.1 Ejemplos en base de datos de Películas	24
4.3.2 Ejemplos en base de datos de TITSA	29
Capítulo 5. Conclusiones y líneas futuras	33
5.1 Conclusiones	33
5.2 Líneas futuras	34
Chapter 5. Summary and Conclusions	35
5.1 Conclusions	35
5.2 Future sights	36
Capítulo 6. Presupuesto	37
6.1 Costes hardware	37
6.2 Costes Recursos Humanos	37

Índice de figuras

Figura 2.1: Arquitectura del modelo transformador	5
Figura 2.2: Formato de petición HTTP	7
Figura 2.3: Formato de respuesta HTTP	8
Figura 3.1: Arquitectura de LangChain	10
Figura 3.2: Diagrama de pensamientos de un agente	11
Figura 3.3: Arquitectura del prototipo	13
Figura 4.1: Modelo entidad-relación de base de datos de películas	14
Figura 4.2: Diagrama matriz origen-destino	16
Figura 4.3: Método de test de conexión	18
Figura 4.4: Inicialización de la base de datos	19
Figura 4.5: Inicialización del modelo y las herramientas SQL	20
Figura 4.6: Prefijo del prompt inicial	20
Figura 4.7: Creación e invocación del agente	21
Figura 4.8: Extracción de la respuesta y la consulta SQL devueltas	22
Figura 4.9: Diagrama de flujo de la herramienta	23
Figura 4.10: Interfaz gráfica para realizar peticiones a la API	24
Figura 4.11: Respuestas devueltas en base de datos de Películas	25
Figura 4.12: Traducciones devueltas en base de datos de Películas	25
Figura 4.13: Ejemplo de ejecución de consulta devuelta por el agente en base de datos de Películas	26
Figura 4.14: Ejecución de consultas de distintos nivel en bas de datos de Películas	29
Figura 4.15: Ejecución de consultas en base de datos de TITSA	32

Índice de tablas

Tabla 4.1: Resultados de ejecución en base de datos de Películas	27
Tabla 7.1: Costes hardware	37
Tabla 7.2: Costes Recursos Humanos	37
Tabla 7.3: Costes totales del proyecto	37

Capítulo 1. Introducción

Durante la introducción vamos a hablar sobre el problema que surge al explorar bases de datos de gran tamaño, tarea que ocupa una considerable cantidad de tiempo y recursos. Plantearemos también la solución planteada a este problema, el alcance del proyecto y las fases de su desarrollo.

1.1 Antecedentes

Los grandes modelos de lenguaje (LLM) se han vuelto extremadamente populares recientemente. Esto es debido a la exponencial evolución que ha experimentado la inteligencia artificial en los últimos años [1]. Es gracias a esta popularidad que han surgido un sinnúmero de herramientas que implementan la IA en sus soluciones. Desde la creación de videos o música, hasta correctores ortográficos y traductores [2]. Concretamente nos centraremos en herramientas que trabajan sobre bases de datos relacionales. Esta idea ha sido bastante explotada, pues a día de hoy existen bastantes herramientas como AI2sql, Text2SQL o AI Query [3]. Estas herramientas permiten construir consultas SQL sin tener conocimientos previos sobre el mismo, pues únicamente se emplea el lenguaje natural. alguna de las desventajas que tienen estas herramientas es que suelen ser de pago por mensualidades o por consultas.

Este trabajo parte del TFG realizado por Adrián Hernández Suárez, titulado “*Uso de modelos de lenguaje y Machine Learning para la consulta y predicción de pasajeros en TITSA*”. También colaborando con la empresa de Transportes Interurbanos de Tenerife (TITSA), el trabajo trata facilitar la consulta de datos sobre pasajeros y predecir la demanda de los mismos a través de una herramienta que integra modelos de lenguaje y machine learning [4].

1.2 Problemática

Cuando se afronta por primera vez una base de datos, es indispensable entender su estructura y contenidos. Esto no es ningún problema para pequeñas bases de datos, pero esto resulta en un problema cuando nos encontramos con una gran base de datos, pues el usuario tendría que dedicar mucho tiempo y recursos en familiarizarse. Podríamos hablar hasta decenas de miles de tablas si se trata de una gran empresa. Es por esta razón que se plantea este proyecto.

1.3 Alcance del proyecto

En pos de facilitar la tarea de familiarización con una base de datos de gran tamaño, se plantea una herramienta en la que el usuario realiza una pregunta en lenguaje natural. La pregunta es procesada por un gran modelo de lenguaje y devolverá la consulta traducida a SQL (en caso de necesitarla), además de una breve respuesta que explique el resultado obtenido. El usuario no solo sería capaz de realizar preguntas relacionadas con consultas SQL, también podría preguntar sobre el esquema de la base de datos. Esto es, las relaciones entre las distintas tablas, una breve descripción de cada columna en una tabla o incluso el tipo de datos que se almacena en cada una de ellas.

Se desarrollará una API para que un usuario pueda hacer peticiones a un servidor donde se localiza la base de datos. La aplicación será capaz de conectarse a bases de datos relacionales de distintos tipos, obtener de ahí la información que requiera para la consulta y devolver una respuesta al usuario. Por otra parte, se garantizará una interfaz que facilite la interacción con la herramienta. El usuario solo deberá introducir los datos de conexión a la base de datos y la consulta que desee realizar en lenguaje natural.

1.4 Fases y desarrollo del proyecto

Desde el inicio del proyecto, se plantearon el siguiente plan de trabajo:

- **Semana 1-3: Exploración y Requerimientos:** Durante estas primeras semanas, se llevó a cabo una exploración exhaustiva de las bases de datos relevantes, identificando los requisitos específicos para el prototipo. Se recopilaron detalles sobre la estructura de los datos, los tipos de consultas comunes y cualquier otro aspecto relevante para el desarrollo del proyecto.
- **Semana 4-6: Preprocesamiento y Diseño del Prototipo:** En esta etapa, se realizó el preprocesamiento de los datos obtenidos, preparándose para su uso en el desarrollo del prototipo. Además, se llevó a cabo una investigación para seleccionar las librerías y APIs más adecuadas, y se diseñó el prototipo basado en los datos recopilados y los requisitos identificados anteriormente.
- **Semana 7-10: Desarrollo y Evaluación Preliminar:** Durante estas semanas, se llevó a cabo el desarrollo del prototipo, implementando las funcionalidades necesarias según lo especificado en el diseño. Se realizaron pruebas preliminares para evaluar el rendimiento del prototipo, realizando ajustes según sea necesario para mejorar su funcionalidad y eficiencia.

- Semana 11-13: Validación y Ajustes: En esta fase, se realizó una validación exhaustiva del prototipo, asegurando que cumpla con los requisitos establecidos y resuelva eficazmente la problemática identificada. Se realizaron ajustes adicionales según los comentarios y resultados de las pruebas realizadas.
- Semana 14-15: Despliegue Final y Entrega: En las últimas semanas del proyecto, se llevó a cabo el despliegue final del prototipo en un entorno de producción. Se realizaron pruebas exhaustivas para garantizar su funcionalidad y usabilidad, y se completó la memoria del TFG. Además, se elaboró una presentación y se realizó una revisión final antes de la entrega final del proyecto.

Capítulo 2. Fundamentos teóricos

Para entender mejor el contexto en el que se sitúa el proyecto, se hablará a continuación de los fundamentos teóricos relacionados con el desarrollo de esta herramienta.

2.1 Modelos de Lenguajes Grandes (LLMs)

Un modelo de lenguaje grande, también conocido como LLM, consiste en un modelo computacional capaz de realizar una variedad de tareas relacionadas con la generación de lenguaje y el procesamiento del lenguaje natural. Este logro se debe al vasto conocimiento que ha adquirido a partir de un extenso conjunto de datos. [5]

La historia del procesamiento del lenguaje natural (NLP) se remonta a los años 50, cuando los investigadores de IBM en la Universidad de Georgetown en Washington D.C. buscaron desarrollar un sistema capaz de traducir automáticamente frases del ruso al inglés. Durante las décadas siguientes, los investigadores intentaron resolver esta cuestión utilizando sistemas basados en reglas y ontologías, pero no lograron obtener resultados robustos. [6] Fue en los años 60 cuando surgió Eliza, el primer chatbot del mundo, desarrollado por Joseph Weizenbaum del MIT. Este programa utilizaba reconocimiento de patrones para simular conversaciones, lo que impulsó la investigación en NLP y sentó las bases para futuros avances.

En el año 2018, OpenAI introdujo GPT-1, el primer gran modelo de lenguaje. Este avance fue posible gracias al desarrollo del transformador por parte de Google el año anterior. Estos modelos de aprendizaje profundo emplean el mecanismo de autoatención, que asigna diferentes pesos a cada parte de la entrada. La arquitectura es similar a la de los modelos seq2seq, con un codificador que procesa la entrada iterativamente por capas y un decodificador compuesto por capas que procesan la salida del codificador. [7]

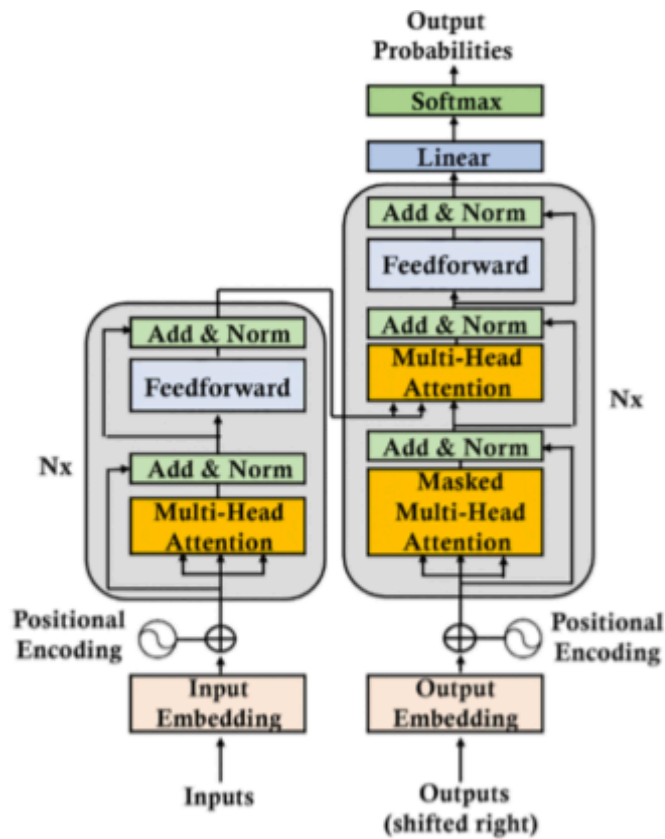


Figura 2.1: Arquitectura del modelo transformador

Algunos de los modelos más destacados en la actualidad son los de la serie GPT de OpenAI, de los cuales hablaremos más adelante. Además, merecen especial mención modelos como PaLM y Gemini de Google, así como LLaMA de Meta, debido a su notable importancia en el campo del procesamiento del lenguaje natural.

2.1.1 OpenAI

OpenAI es una empresa de investigación y despliegue de inteligencia artificial fundada en 2015. De entre los proyectos que han desarrollado sobresalen DALL-E3, un modelo de generación de imágenes; ChatGPT, un chatbot de increíble capacidad de respuesta; y GPT, una serie de modelos de lenguaje.

El último de estos modelos de lenguaje es GPT-4, basado en el modelo transformer y ajustado mediante aprendizaje por refuerzo a partir de retroalimentación humana e inteligencia artificial [8]. En comparación con su predecesor GPT-3, este último modelo consiguió aumentar su número de parámetros de 175.000 millones a 100 billones.

2.2 Bases de datos relacionales

Las bases de datos relacionales son un tipo de base de datos que cumple con el modelo relacional [9]. Esto es, basado en la lógica de predicados y en la teoría de conjuntos. Estas bases de datos se componen en tablas, cada una de las cuales con sus columnas y entradas. Las relaciones entre tablas se llevan a cabo gracias a las claves primarias y las claves ajenas. Una clave primaria consiste en un elemento principal por el que se puede identificar una entrada dentro de una tabla. Es por esta razón que una clave primaria no puede repetirse dentro de la misma tabla. Por otra parte, una clave ajena hace referencia a la clave primaria de otra tabla, formando así una relación entre tablas.

Para poder administrar y obtener información de una base de datos, se emplea el lenguaje de consulta estructurada (SQL). Este lenguaje hace uso del álgebra y cálculo relacional para realizar cambios sobre la base de datos y conseguir información de la misma. Este lenguaje es aplicado a un sistema gestor de base de datos, un software encargado de administrar una base de datos. De entre los más conocidos se encuentran MySQL, MariaDB, PostgreSQL, etc.

A los distintos tipos de consultas que podemos realizar sobre una base de datos los podemos clasificar en los siguientes grupos[10]:

- **DDL, Data Definition Language:** Define la creación, modificación y borrado de tablas, filas y columnas.
- **DML, Data Manipulation Language:** Consiste en la manipulación de tablas y columnas. Esto es insertar filas, actualizarlas y borrarlas.
- **DCL, Data Control Language:** Es usado para el control de acceso a la base de datos.
- **TCL, Transaction Control Language:** Hace que los datos perduren en la base de datos, es decir, hacer permanentes los cambios hechos en la misma.
- **DQL, Data Query Language:** Engloba a todas las consultas de recuperación de información, empezando todas estas por *SELECT*.

2.3 Interfaz de programación de aplicaciones (APIs)

Una API es un fragmento de código que permite la conexión y comunicación entre distintas aplicaciones. De esta forma, estas aplicaciones pueden transmitir información entre sí.

Una de las formas más comunes de implementar dicha comunicación es mediante

el uso de peticiones HTTP[11]. Este protocolo permite la transferencia de datos en las capas de transporte y red (TCP e IP). Una comunicación típica comienza con un mensaje de petición sobre una URL en el puerto 80. Este mensaje se estructura de la siguiente manera:

- **Cabecera del mensaje de la petición:**
 - **Línea de petición:** Indica el método de la petición junto a la ruta sobre la que se realiza y la versión del protocolo.
 - **Cabecera de petición:** Indica el host al que se realiza la petición además de otras cabeceras opcionales, como el tipo de contenido que se acepta, el tamaño de la petición o información sobre el agente de usuario (navegador, sistema operativo...)
- **Cuerpo del mensaje petición:** Mensaje que se envía junto a la petición. Suele contener parámetros introducidos por el usuario.

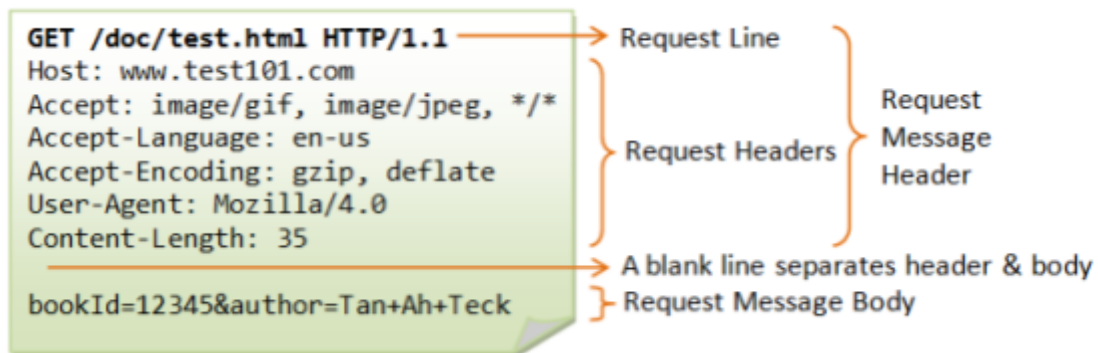


Figura 2.2: Formato de petición HTTP

Los métodos de petición definen el propósito que tiene la misma. Algunos de los métodos más usados son GET, POST, PUT, DELETE, o HEAD. Tras recibir una petición, se devuelve una respuesta en el siguiente formato:

- **Cabecera del mensaje de respuesta**
 - **Línea de estatus:** Indica la versión del protocolo junto a un mensaje y código de estatus.
 - **Cabecera de respuesta:** No cuenta con campos obligatorios, aunque puede contener información como el tipo de dato devuelto o el tamaño del contenido.
- **Cuerpo del mensaje de respuesta:** Mensaje que se envía junto a la respuesta. Es

el contenido que la petición esperaba recibir.

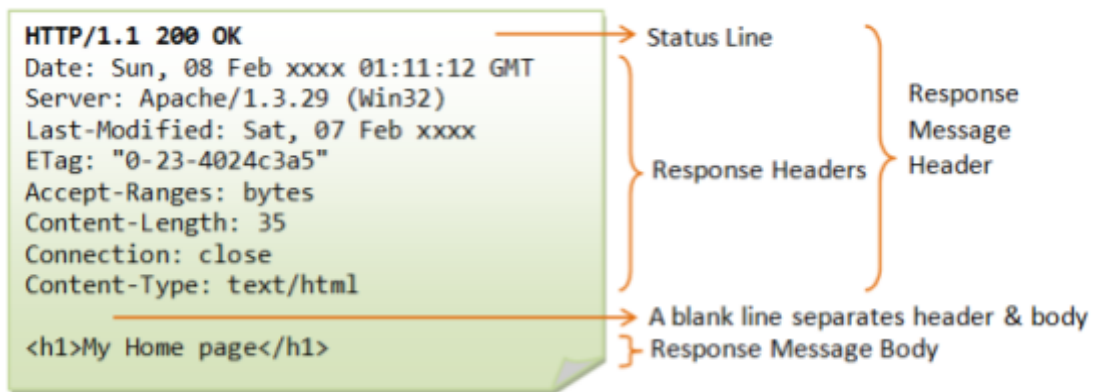


Figura 2.3: Formato de respuesta HTTP

Los códigos de estatus definen el resultado de la petición realizada. Estos códigos se pueden separar en 5 clases, definidas por el primer dígito del código. Los otros dos dígitos se usan para definir la respuesta de forma más específica. Estas clases son las siguientes:

- **1xx respuesta informativa:** la petición ha sido recibida y el proceso continúa.
- **2xx respuesta satisfactoria:** la petición ha sido recibida con éxito.
- **3xx redirecciones:** para completar la petición, se lleva a otro lado.
- **4xx error del cliente:** la petición contiene mala sintaxis o no puede ser resuelta.
- **5xx error del servidor:** el servidor falló en resolver una petición aparentemente válida.

Capítulo 3. Tecnologías y arquitectura del prototipo

3.1 Tecnologías

3.1.1 Langchain

Langchain es un framework para el desarrollo de aplicaciones que necesiten de un modelo de lenguaje para su solución. Esta herramienta es capaz de conectar un modelo de lenguaje a distintas fuentes, como pueden ser documentos de diversas extensiones (CSV, PDF o JSON), archivos dentro de un directorio o incluso bases de datos relacionales y no relacionales [12].

Este framework consiste en distintas partes:

- **Librerías de Langchain:** Pudiendo implementarse en JavaScript y Python, contiene las interfaces e integraciones necesarias para que el usuario pueda implementar de forma rápida y sencilla su herramienta.
- **Plantillas de Langchain:** Exclusivamente para Python, contiene una colección de arquitecturas de rápido despliegue que cubren una gran variedad de tareas.
- **LangServe:** Exclusivamente para Python, permite desplegar LangChain como una REST API
- **LangSmith:** Plataforma de desarrollo que permite depurar, comprobar, evaluar y monitorizar una cadena implementada en cualquier framework de LLM.

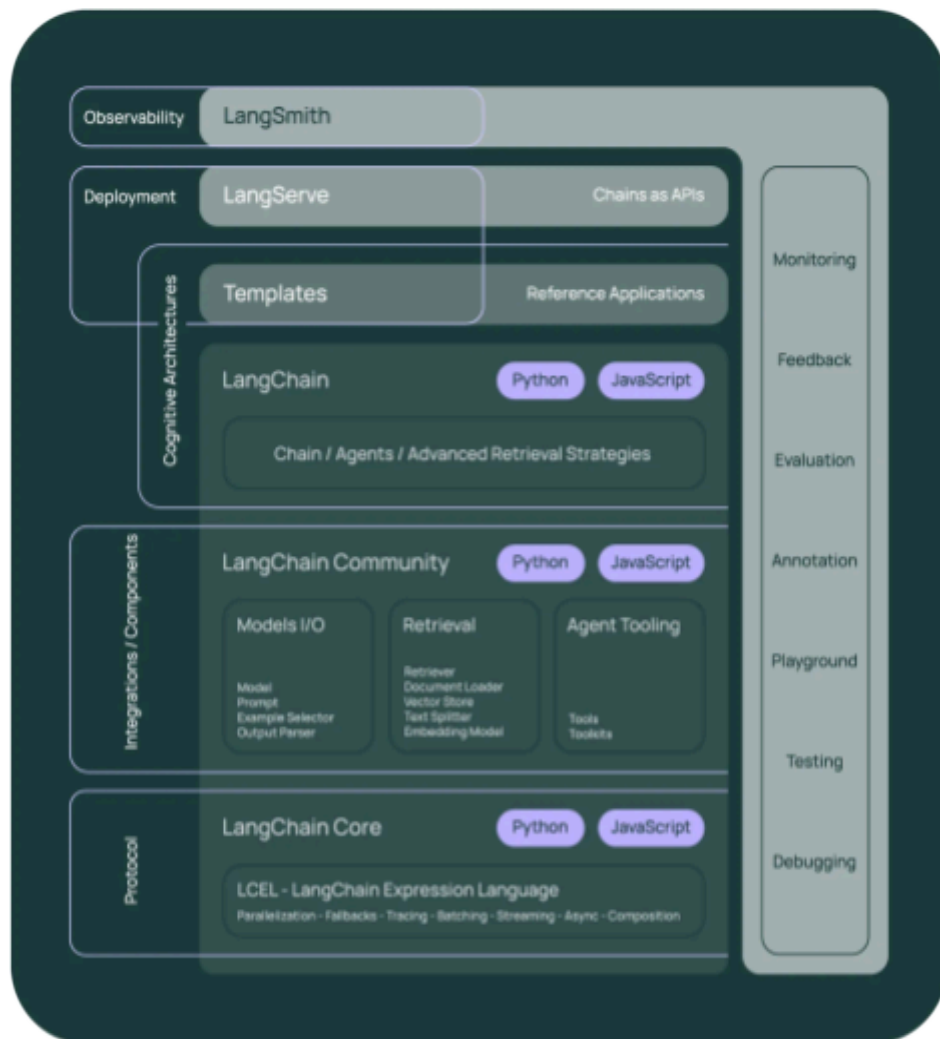


Figura 3.1: Arquitectura de LangChain.
 Fuente(https://js.langchain.com/v0.1/docs/get_started/introduction/)

3.1.2 Agentes

Una de las funcionalidades que ofrece la librería de Langchain son los agentes. Un agente emplea un modelo de lenguaje como motor de pensamiento. De esta forma, cuando el agente recibe un input, luego determina qué acciones tomar para llegar a una respuesta. Este proceso de pensamiento es iterativo, pues sus acciones son observadas por sí mismo y, a partir de estas, generan un nuevo pensamiento.

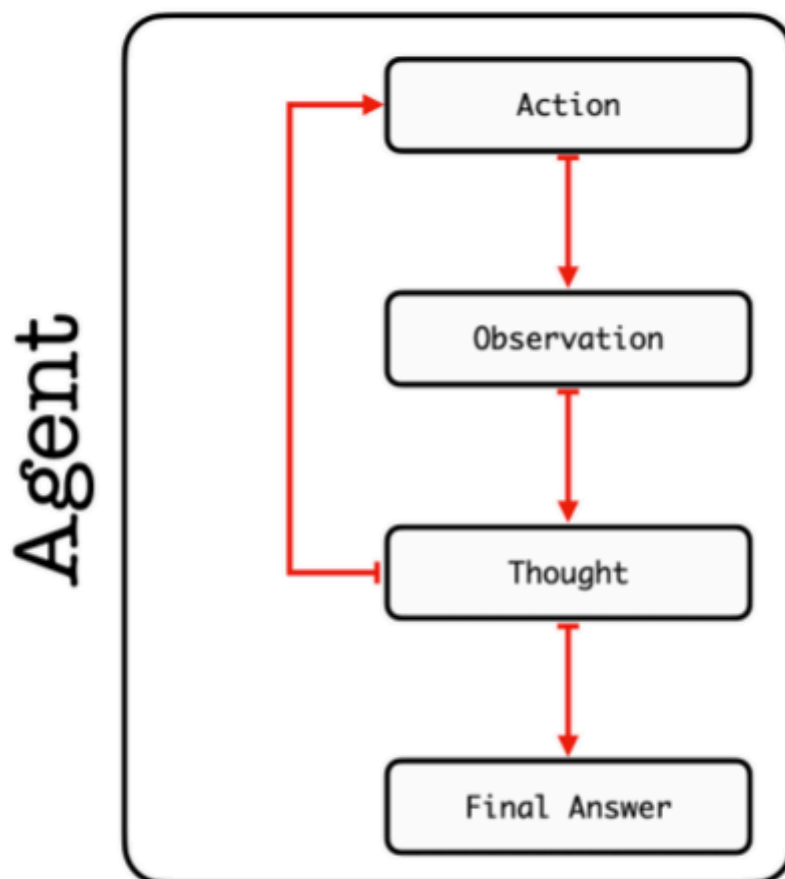


Figura 3.2: Diagrama de pensamientos de un agente.

Fuente:

<https://medium.com/@saumitra1joshi/langchain-agents-tools-chains-memory-for-utilizing-the-full-potential-of-lms-211e5dfee3fa>

En esta figura podemos observar el anteriormente mencionado proceso de pensamiento. En primer lugar, al agente se le proporciona un input junto con un pensamiento inicial. Este pensamiento inicial lleva a una acción por parte del agente y, después, a una observación. Durante la observación, el agente analiza cómo ha afectado la acción que ha realizado. Este análisis lleva a un pensamiento sobre qué hacer después y, en caso de creer que ha finalizado y que conseguirá responder al input, terminará este ciclo con una respuesta final.

Para que el agente sea capaz de interactuar con otros elementos, se le proporciona una serie de herramientas (tools). Estas herramientas consisten en interfaces que agrupan el nombre de la herramienta, una breve descripción de la misma, un

esquema de los inputs, la función a la que llama y si el resultado debe ser devuelta al usuario. Estas herramientas pueden ser agrupadas en los definidos kits de herramientas (toolkits).

3.1.3 Express

Express es un framework sobre Node.js para el desarrollo de aplicaciones web y APIs. Desde este marco de trabajo podemos definir una serie de rutas que atienden a las peticiones HTTP de un usuario. Además, podemos definir para cada ruta un método de los disponibles en las peticiones HTTP (GET, POST, PUT...) para definir una acción en función al método empleado[13].

Dentro de cada ruta, podemos hacer uso de la información pasada por el usuario. Por ejemplo, podemos acceder al cuerpo de la petición y obtener la información que ha sido previamente formateada como JSON por el cliente. Otra forma por la que se puede enviar información a través de una petición HTTP son como parámetros en la URL.

Finalmente podemos enviar una respuesta de vuelta al cliente acompañado de un código de respuesta. De esta forma se puede implementar dentro de la ruta un manejo de excepciones capaz de enviar un mensaje y código de éxito en caso de que el programa haya funcionado correctamente o, por otra parte, enviar un mensaje y código de error cuando se capture un error durante el proceso de ejecución.

3.1.4 PostgreSQL

PostgreSQL es un sistema gestor de base de datos relacional de código abierto[14]. Esto quiere decir que el desarrollo del mismo es llevado por una comunidad abierta sin esperar una compensación económica. Un sistema gestor de base de datos se encarga de proporcionar a un usuario las herramientas para administrar una base de datos.

3.2 Arquitectura

El prototipo consta de dos partes: el lado del cliente y el lado del servidor. En el lado del cliente, el usuario introducirá los datos de conexión a la base de datos y la consulta a realizar a través de una interfaz gráfica. Luego, esta información es pasada al lado del servidor a través de una petición HTTP. Esta petición será capturada por la API y usando los datos introducidos por el usuario se establece la conexión con la base de datos. Una vez iniciada, se llama a un agente de Langchain para que resuelva la consulta realizada. Cuando llegue a una respuesta, esta será devuelta por la API para llegar al usuario.



Figura 3.3: Arquitectura del prototipo

Esta arquitectura permite tener una única instancia de la aplicación ejecutándose en el mismo servidor en el que se encuentra la base de datos. Por otra parte, el servidor podría recibir varias peticiones de distintos usuarios que quieran realizar consultas al mismo tiempo.

Capítulo 4. Desarrollo

4.1 Datos empleados

4.1.1 Base de datos de ejemplo: Películas

Como primera base de datos sobre la que ejecutar pruebas, se ha elegido una base de datos de películas de la página web [databasestar.com](https://www.databasestar.com)[15]. Se ha elegido esta base de datos debido a que cuenta con las suficientes relaciones entre tablas para que no se trate de una base de datos muy sencilla pero, por otra parte, no se trata de un esquema extremadamente complicado. La base de datos ha sido cargada en un servidor local de PostgreSQL.

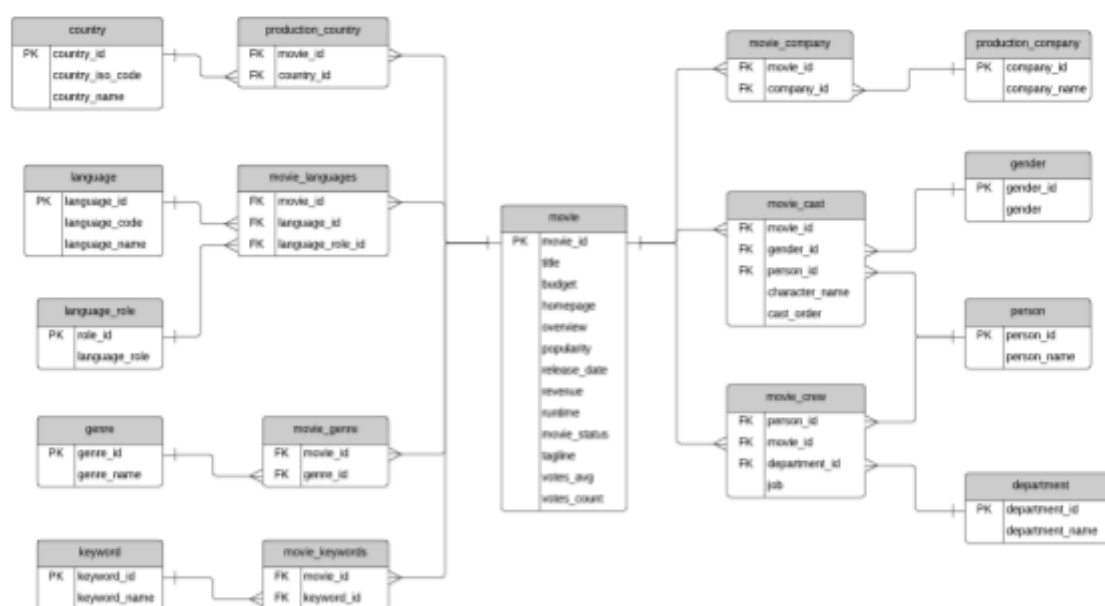


Figura 4.1: Modelo entidad-relación de base de datos de películas.

Fuente : <https://www.databasestar.com/sample-database-movies/>

A continuación se describen las tablas que forman parte de la base de datos:

- **movie**: es la tabla principal que contiene la información relacionada con películas. Cuenta con un título y una descripción sobre la misma, además de diversos campos como el presupuesto, fecha de salida, duración, etc.
- **country**: lista de países. Se relaciona con la tabla películas por medio de la tabla *production_country*, definiendo que una película ha sido producida e cierto país.
- **language**: lista de idiomas. Se relaciona con la tabla películas por medio de la tabla *movie_languages*, definiendo que una película cuenta con cierto idioma.

- **language_role**: contiene dos entradas: "Original" o "Spoken". Formando parte también de la tabla *movie_languages*, indica si un idioma es el original de la película o si se trata de otro idioma que es hablado dentro de la misma.
- **genre**: lista de géneros. Se relaciona con la tabla películas por medio de la tabla *movie_genre*, definiendo que una película pertenece a cierto género.
- **keyword**: lista de palabras clave. Se relaciona con la tabla películas por medio de la tabla *movie_keywords*, definiendo que a una película se le puede atribuir cierta palabra clave.
- **production_company**: lista de productoras. Se relaciona con la tabla películas por medio de la tabla *movie_company*, definiendo que una película ha sido producida por cierta empresa.
- **gender**: género de una persona, pudiendo ser "Male", "Female" o "Unspecified".
- **person**: lista de personas. Se relaciona con la tabla de películas de dos formas. La primera a través de la tabla *movie_cast*, definiendo que en una película, una persona ha formado parte del elenco de personajes. Además, a través de esta misma tabla se define el género de la persona por medio de la tabla anteriormente mencionada. Por otra parte, a través de la tabla *movie_crew* se define que una en una película, una persona ha formado parte del equipo de rodaje.
- **department**: lista de departamentos dentro del rodaje de una película. También forma parte de la tabla *movie_crew* para indicar el departamento en el que trabaja cierta persona en cierta película.

4.1.2 Matriz de origen destino TITSA Marzo 2024

En segundo lugar, contamos con distintos archivos en formato CSV con datos relacionados a los viajes realizados en el mes de marzo de la empresa TITSA [16]. Al estar los datos en otro formato, se han tenido que limpiar y procesar antes de ser cargados en la base de datos local. A diferencia de la base de datos de películas, esta base de datos no cuenta con muchas tablas, teniendo únicamente cinco. Lo que hace esta base de datos interesante es que nos encontramos con tablas de aproximadamente un millón de entradas.

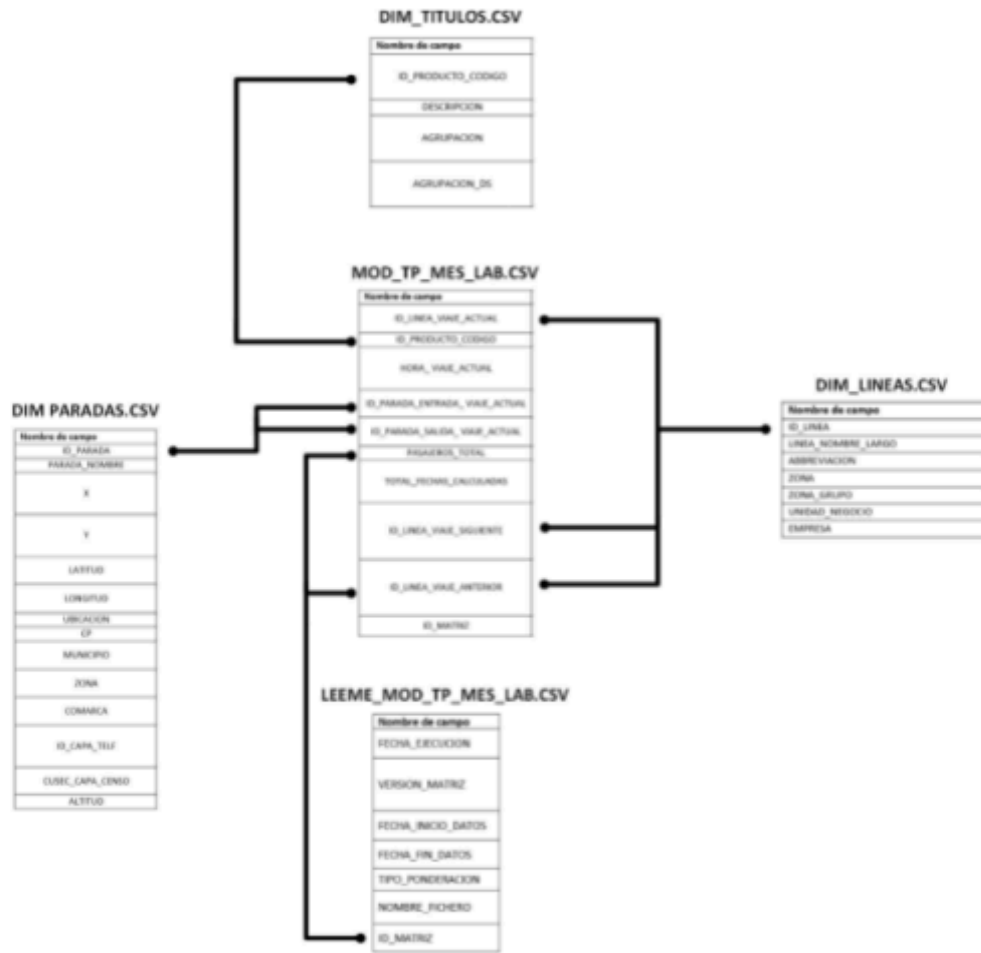


Figura 4.2: Diagrama matriz origen-destino. Fuente: <https://portalde transparencia.titsa.com/index.php/estadisticas/informes-estadisticos-2021>

Este diagrama que muestra la relación entre los datasets podría interpretarse como el modelo entidad-relación en el que va a resultar una vez cargado en la base de datos. La descripción de las tablas es la siguiente:

- **Paradas:** Información sobre las distintas paradas, su localización, nombre e identificadores.
- **Títulos:** Lista de títulos posibles que puede tener un pasajero en TITSA.
- **Líneas:** Información sobre las distintas líneas, su nombre, zona e identificador.
- **Leeme_mes:** Metadatos referentes a la matriz de origen-destino.
- **Viajes_mes: laborables:** Matriz de origen-destino sobre un cierto mes en días laborables, indicando el identificador de la línea del viaje del que se trata, el identificador del título usado durante el viaje, los identificadores de parada de entrada y de salida del pasajero, identificadores de línea del viaje anterior y siguiente en caso de haberlos, número de pasajeros en total e identificador de la matriz.
- **Viajes_mes: no laborables:** Similar a la anterior tabla, pero recogiendo los

viajes realizados durante días no laborales. Cuenta con las mismas tablas a excepción del tipo de día, pudiendo ser Sábado, Domingo o festivo.

4.2 Funcionamiento de la aplicación

Tratándose de una API, se han definido distintas rutas sobre las que se pueden realizar peticiones. La instancia de la API se ha creado para escuchar en el puerto 3001 en la máquina local. Entonces, si quisiéramos realizar una petición a la API, tendríamos que hacerla sobre la URL "<http://localhost:3001>". Si fuésemos a ofrecer este servicio a nuestra red, se debe cambiar el "localhost" por la IP de la máquina que ejecuta el programa. Por otra parte, sería necesario listar las IPs a las que permitamos la conexión a través de CORS.

La primera y más simple de las rutas se encarga de probar la conexión con la base de datos. Esta ruta se encuentra en "/ping" y estará escuchando peticiones con el método POST. Luego, toma los parámetros de conexión pasados por el usuario y crea una instancia de "DataSource". Esta clase está implementada por la librería "typeorm" de JavaScript. Esta instancia se encarga de realizar la conexión con la base de datos. El código capturará cualquier tipo de error ocurra durante la ejecución y, si la conexión es exitosa, la API devolverá código 200. En caso contrario, enviará el error al usuario. La petición a la API deberá de contar con los siguientes parámetros en formato JSON:

- **databaseInfo:**
 - **databaseType:** tipo de base de datos (postgres, mysql, mariadb...)
 - **host:** nombre del host en el que se aloja la base de datos
 - **port:** puerto que da servicio a la base de datos
 - **username:** nombre de usuario con el que se desea conectar
 - **password:** contraseña del usuario
 - **database:** nombre de la base de datos a la que se desea conectar
 - **schema:** esquema de la base de datos a la que se desea conectar

```
const connect = async (params) => {
  const datasource = new DataSource({
    type: params.databaseType,
    host: params.host,
    port: params.port,
    username: params.username,
    password: params.password,
    database: params.database,
    schema: params.schema,
    synchronize: true,
  });
  console.log("Connecting to database");
  try {
    await datasource.initialize();
    console.log("Connected to database");
  } catch (e) {
    console.log("Error connecting to database");
    console.log(e);
  }
}
```

Figura 4.2: Método de test de conexión

Luego contamos con la ruta más importante, pues es la que se encarga de recibir los datos de conexión de la base de datos, instanciar al agente de LangChain y resolver la consulta del usuario. Para ello, en primer lugar, crea de forma similar a la descrita anteriormente una instancia de "DataSource". Luego, a partir de esta instancia, se crea un objeto de la clase `SQLDatabase` proporcionada por LangChain para la representación de bases de datos.

```

app.post("/", async (req, res) => {
  try {

    console.log("Creating database connection...");

    const datasource = new DataSource({
      type: req.body.databaseInfo.databaseType,
      host: req.body.databaseInfo.host,
      port: req.body.databaseInfo.port,
      username: req.body.databaseInfo.username,
      password: req.body.databaseInfo.password,
      database: req.body.databaseInfo.database,
      schema: req.body.databaseInfo.schema
    });

    console.log("Connecting to database");

    const db = await SQLiteDatabase.fromDataSourceParams({
      appDataSource: datasource,
    });
  }
}

```

Figura 4.4: Inicialización de la base de datos

Luego, se inicializa el modelo de ChatGPT a través del envoltorio proporcionado por LangChain para acceder al endpoint del mismo. A este se le especifica el tipo de modelo que se va a usar además de una temperatura de 0 para menor uso de valores aleatorios y más precisos. En cuanto a la clave de la API, se ha decidido almacenar como variable de entorno por razones de seguridad. Además, se han inicializado una serie de herramientas que proporciona LangChain para la interacción de agentes con bases de datos SQL. Las herramientas que incluye este kit son las siguientes:

- **InfoSQLDatabaseTool:** a esta herramienta se le introduce una lista de tablas y se devuelve el esquema y filas de ejemplo de dichas tablas.
- **ListSQLDatabaseTool:** devuelve una lista de las tablas en la base de datos.
- **QuerySQLCheckerTool:** se usa para comprobar doblemente si la petición es correcta antes de ejecutarla.
- **QuerySQLDatabaseTool:** a esta herramienta se le introduce una petición correcta y detallada en SQL y se devuelve un resultado de la base de datos.

```

console.log("Connected to database");
const llm = new ChatOpenAI(
  {
    temperature: 0,
    model: "gpt-4o ",
  }
);
const sqlToolkit = new SqlToolkit(db, llm);

console.log("Creating agent...");

```

Figura 4.5: Inicialización del modelo y las herramientas SQL

A continuación, se va a definir un prefijo del prompt inicial en el que se le da contexto al modelo sobre lo que tiene que hacer. El prefijo es el siguiente:

```

const SQL_PREFIX = `You are an agent designed to interact with a SQL database.
Given an input question, create a syntactically correct {dialect} query to run, then look at the results of the query and return the answer and
Unless the user specifies a specific number of examples they wish to obtain, always limit your query to at most {top_k} results using the LIMIT
You can order the results by a relevant column to return the most interesting examples in the database.
Never query for all the columns from a specific table, only ask for a the few relevant columns given the question.
You have access to tools for interacting with the database.
Only use the below tools.
Only use the information returned by the below tools to construct your final answer.
You MUST double check your query before executing it. If you get an error while executing a query, rewrite the query and try again.`

+ req.body.customPrompt +

`DO NOT make any DML statements (INSERT, UPDATE, DELETE, DROP etc.) to the database.

If the question does not seem related to the database, just return "I don't know" as the answer.
The format of the output should ALWAYS be "SQL Query: <query>" and "Answer: <answer>". If a SQL query is not needed, return "SQL Query: N/A" and
`;

```

Figura 4.6: Prefijo del prompt inicial

De entre las líneas que podemos destacar dentro de este prefijo, se encuentra la que especifica que debe hacer siempre una doble comprobación de la consulta antes de ejecutarla y que, si hubiese algún error, la vuelva a escribir y ejecutar. De esta forma podemos asegurarnos que la consulta SQL que nos sea devuelta sea sintácticamente correcta. Por otra parte, cabe destacar la línea sobre no hacer ninguna instrucción DML. Estas son cualquier consulta que altere de alguna forma la base de datos. Esto nos asegura que el agente no haga ninguna modificación de forma accidental. Una segunda capa de seguridad que se le puede dar al agente es en los permisos del usuario que especificamos en los datos de conexión. Podemos crear un usuario que únicamente pueda realizar consultas DQL, es decir, consultas de recuperación de información. Por otra parte, se le indica al agente el formato en el que tiene que devolver la respuesta.

Además, al final del prefijo se añadirá un prompt introducido por el usuario (en caso de haberlo). La finalidad de este prompt es hacer la experiencia de usuario más personalizada, de forma que el usuario final de la herramienta pueda modificar cómo es devuelta la respuesta, o que el agente tenga ciertas consideraciones a la hora de mostrar datos. Por ejemplo, se le podría pedir al agente que muestra las fechas en formato DD-MMM-YYYY.

En algunas partes del prompt aparecen parámetros entre llaves. Estos parámetros pueden ser especificados más adelante para permitir una mayor personalización en los resultados. Estos parámetros son el dialecto de SQL usado para realizar las consultas y un tope de resultados devueltos, por defecto 10.

Finalmente se declara al agente enlazando el modelo inicializado, las herramientas definidas y el prompt. Usamos el método de Langchain `createSqlAgent` para poder invocar al agente con la petición pasada por el usuario, tal como se muestra en la figura 4.7. El resultado que nos devuelva estará formado por dos partes, tal y como se indica en el prompt inicial. Para poder devolver ambas respuestas de forma correcta, se pasa el resultado por varios métodos de manipulación de cadenas de texto para partir la respuesta en “`sqlQuery`” y “`answer`” y devolverlo a través de la API. Estos métodos de manipulación de cadenas se pueden ver en la figura 4.8.

```
const sqlAgent = createSqlAgent(  
  llm,  
  sqlToolkit,  
  {  
    prefix: SQL_PREFIX,  
  }  
);  
  
console.log("Invoking agent...");  
  
const result = await sqlAgent.invoke({ input: req.body.query });  
console.log("Result obtained");  
console.log(result.intermediateSteps)  
console.log(result.output)
```

Figura 4.7: Creación e invocación del agente

```

const outputParts = result.output.split(/(SQL Query:|Answer:)/);

const sqlQueryIndex = outputParts.indexOf("SQL Query:") + 1;
const answerIndex = outputParts.indexOf("Answer:") + 1;

const sqlQuery =
  sqlQueryIndex < outputParts.length
    ? outputParts[sqlQueryIndex].trim()
    : null;
const answer =
  answerIndex < outputParts.length ? outputParts[answerIndex].trim() : null;

```

Figura 4.8: Extracción de la respuesta y la consulta SQL devueltas

Todo este proceso descrito se encuentra envuelto en un bloque “try”, de forma que si surgiese algún error durante la creación y ejecución del agente, se intercepta y se manda un mensaje de error a través de la API junto a un código de error 500.

A continuación se muestra un diagrama de flujo que muestra una ejecución típica del programa. El usuario envía a través de la API la información de conexión a la base de datos, un prompt personalizado y la pregunta. Con los datos de conexión se inicializa la base de datos y, en caso de haber algún error, se devuelve al usuario. Luego se declara el modelo a utilizar por el agente y el set de herramientas SQL. Luego, se define el prefijo del prompt inicial, al que se le incrusta el prompt personalizado introducido por el usuario. Después se inicializa el agente pasándole el modelo, herramientas y prompt. Una vez creado, se invoca al agente con la pregunta formulada por el usuario. El agente parte con un prompt inicial, el cuál indica que el agente debería de buscar información en las tablas disponibles. A través de este pensamiento, el agente hace uso de alguna de las herramientas del kit SQL descrito anteriormente para ejecutar una acción. Luego, observa cómo ha afectado la acción y genera un nuevo pensamiento. Si el agente llega a la conclusión de que ese pensamiento es la respuesta final, termina y devuelve la respuesta. De no ser así, ejecuta otra acción y repite el proceso. Finalmente, se extrae del resultado la respuesta a la pregunta y la traducción SQL y se envía al usuario.

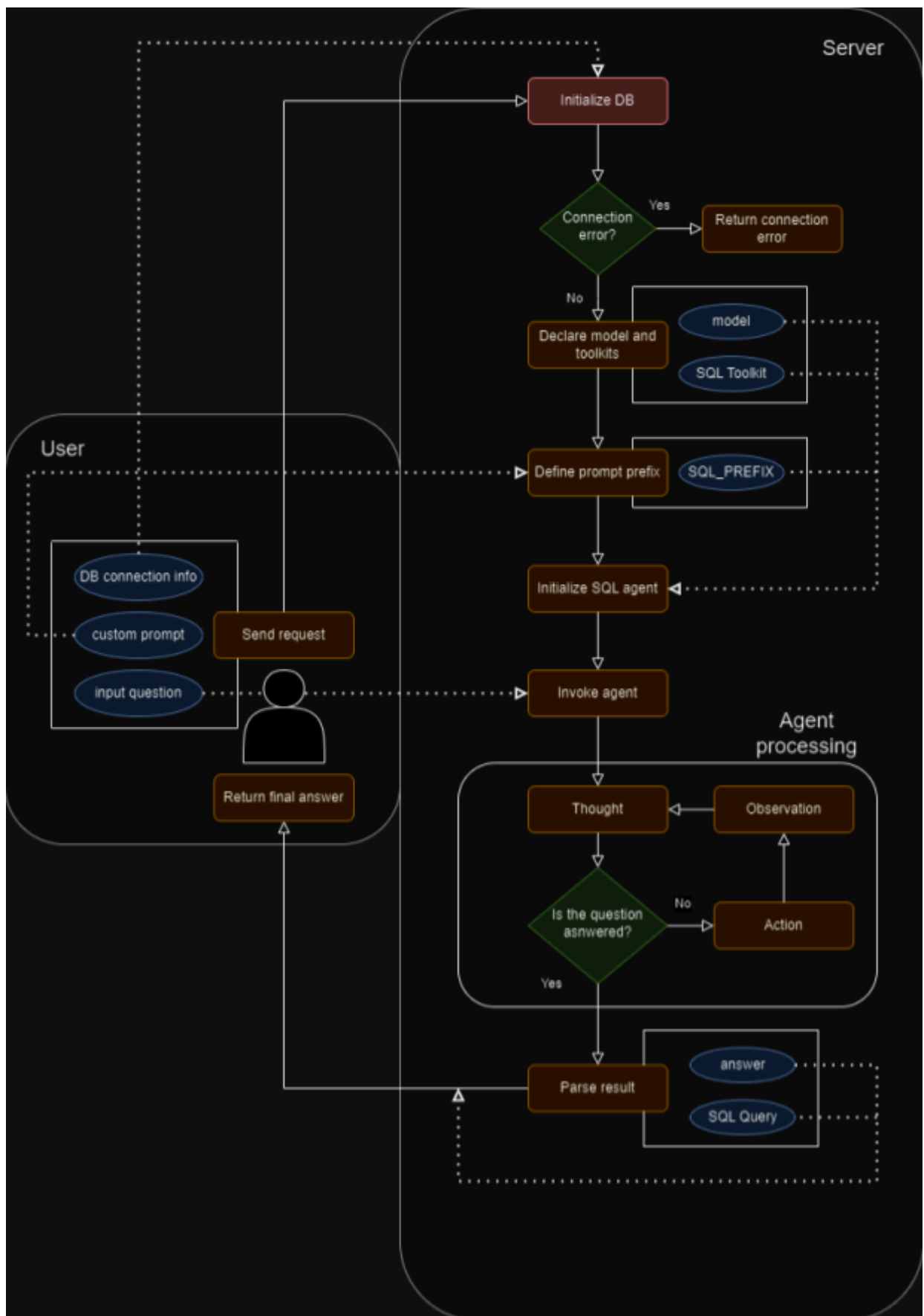


Figura 4.9: Diagrama de flujo de la herramienta

4.3 Ejemplos de ejecución

Para visualizar de forma más sencilla los datos devueltos por la API, se ha creado una interfaz web en React. La interfaz contiene a la izquierda un menú en el que el usuario introduce los datos de conexión a la base de datos. Luego, el usuario será capaz de introducir su pregunta en lenguaje natural, además de instrucciones adicionales en caso necesario. Luego, cuando el usuario usa el botón de “Generar SQL”, se comprueban los datos de conexión por medio del endpoint “/ping” y, si la conexión es correcta, se realiza la petición. Luego, la herramienta devolverá la pregunta realizada, la respuesta a la misma y una traducción en SQL. Además, la interfaz cuenta con un botón para mostrar un ejemplo de pregunta.

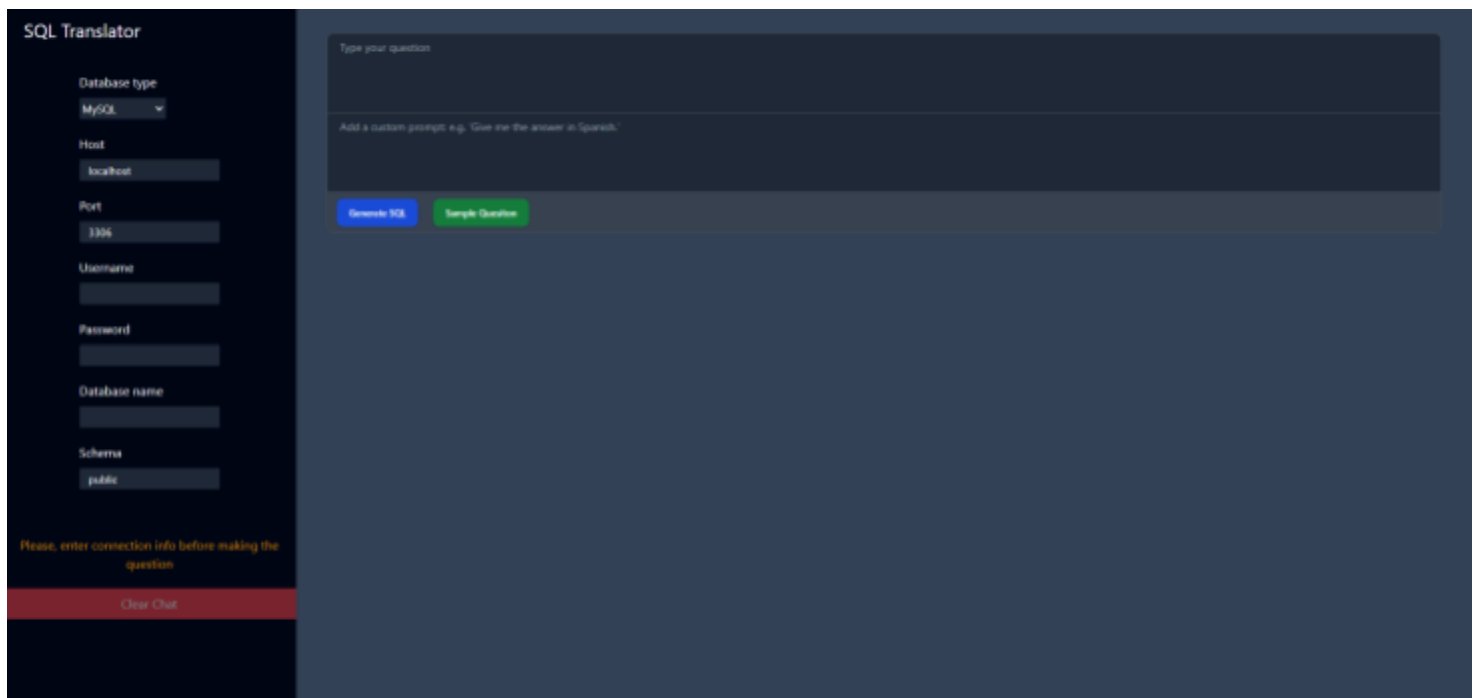


Figura 4.10: Interfaz gráfica para realizar peticiones a la API

Los ejemplos de ejecución que se muestran a continuación se van a visualizar en la interfaz gráfica y, en otras ocasiones, como resultado de la respuesta recogida directamente de la API. Esto se hace para poder visualizar mejor una gran cantidad de respuestas que, normalmente, sería difícil de mostrar en la aplicación.

4.3.1 Ejemplos en base de datos de Películas

Sobre la base de datos de películas, hemos ejecutado la misma petición 25 veces para comprobar que los resultados que devuelve son correctos, el tiempo de ejecución de cada uno, si la traducción de la consulta a SQL es sintácticamente correcta y el tiempo medio de ejecución. La consulta es *“List the top 10 most popular movies, including their titles, release dates, and the number of genres they belong to.”*

```

"answers": [
  "The top 10 most popular movies, including their titles, release dates, and the number of genres they belong to are:\n1. Minions (2015-06-16), Genres: 4\n2. I
  "The top 10 most popular movies, including their titles, release dates, and the number of genres they belong to, are:\n1. Minions (2015-06-16), Genres: 4\n2.
  "The top 10 most popular movies, including their titles, release dates, and the number of genres they belong to are:\n1. Minions (Released on June 16, 2015) -
  "The top 10 most popular movies, including their titles, release dates, and the number of genres they belong to are:\n1. Minions, 2015-06-16, 4 genres\n2. Int
  "The top 10 most popular movies, including their titles, release dates, and the number of genres they belong to, are:\n1. Minions (2015-06-16), Genres: 4\n2.
  "The top 10 most popular movies, including their titles, release dates, and the number of genres they belong to, are:\n1. Minions (Released on June 16, 2015) -
  "The top 10 most popular movies, including their titles, release dates, and the number of genres they belong to, are:\n1. Minions (2015-06-16), Genres: 4\n2.
  "The top 10 most popular movies, including their titles, release dates, and the number of genres they belong to, are:\n1. Minions (2015-06-16), Genres: 4\n2.
  "The top 10 most popular movies, including their titles, release dates, and the number of genres they belong to, are:\n1. Minions (2015-06-16), Genres: 4\n2.
  "The top 10 most popular movies, including their titles, release dates, and the number of genres they belong to, are:\n1. Minions (Released on June 16, 2015) -
  "The top 10 most popular movies, including their titles, release dates, and the number of genres they belong to, are:\n1. Minions (2015-06-16), Genres: 4\n2.
  "The top 10 most popular movies, including their titles, release dates, and the number of genres they belong to, are:\n1. Minions (2015-06-16), Number of Genr
  "The top 10 most popular movies, including their titles, release dates, and the number of genres they belong to, are:\n1. Minions (2015-06-16), Genres: 4\n2.
  "The top 10 most popular movies, including their titles, release dates, and the number of genres they belong to are:\n1. Minions - June 16, 2015 - 4 genres\n2.
  "The top 10 most popular movies, including their titles, release dates, and the number of genres they belong to are:\n1. Minions (2015-06-16), Genres: 4\n2. I
  "The top 10 most popular movies, including their titles, release dates, and the number of genres they belong to, are:\n1. Minions (2015-06-16), Number of Genr
  "The top 10 most popular movies, including their titles, release dates, and the number of genres they belong to are:\n1. Minions (Released on June 16, 2015) -
  "The top 10 most popular movies, including their titles, release dates, and the number of genres they belong to, are:\n1. Minions (2015-06-16), Genres: 4\n2.

```

Figura 4.11: Respuestas devueltas en base de datos de Películas

Siendo la película “Minions” la más popular, podemos ver cómo ha respondido correctamente en 22 ocasiones, fallando la consulta en las otras 3.

```

"sqlQueries": [
  "SELECT m.title, m.release date, COUNT(mg.genre_id) AS genre_count\nFROM movie m\nJOIN movie_genres mg ON m.movie_id = mg.movie_id\nGROUP BY m.title, m.releas
  "SELECT m.title, m.release date, COUNT(mg.genre_id) AS genre_count\nFROM movie m\nJOIN movie_genres mg ON m.movie_id = mg.movie_id\nGROUP BY m.title, m.releas
  "The top 10 most popular movies, including their titles, release dates, and the number of genres they belong to are:\n1. Minions (Released on June 16, 2015) -
  "The top 10 most popular movies, including their titles, release dates, and the number of genres they belong to are:\n1. Minions, 2015-06-16, 4 genres\n2. Int
  "SELECT m.title, m.release date, COUNT(mg.genre_id) AS genre_count\nFROM movie m\nJOIN movie_genres mg ON m.movie_id = mg.movie_id\nGROUP BY m.title, m.releas
  "The top 10 most popular movies, including their titles, release dates, and the number of genres they belong to are:\n1. Minions (Released on June 16, 2015) -
  "SELECT m.title, m.release date, COUNT(mg.genre_id) AS genre_count\nFROM movie m\nJOIN movie_genres mg ON m.movie_id = mg.movie_id\nGROUP BY m.title, m.releas
  "SELECT m.title, m.release date, COUNT(mg.genre_id) AS genre_count\nFROM movie m\nJOIN movie_genres mg ON m.movie_id = mg.movie_id\nGROUP BY m.title, m.releas
  "SELECT movie.title, movie.release date, COUNT(movie_genres.genre_id) AS genre_count\nFROM movie\nJOIN movie_genres ON movie.movie_id = movie_genres.movie_id\
  "SELECT m.title, m.release date, COUNT(mg.genre_id) AS genre_count\nFROM movie m\nJOIN movie_genres mg ON m.movie_id = mg.movie_id\nGROUP BY m.title, m.releas
  "SELECT movie.title, movie.release date, COUNT(movie_genres.genre_id) AS genre_count\nFROM movie\nJOIN movie_genres ON movie.movie_id = movie_genres.movie_id\
  "SELECT m.title, m.release date, COUNT(mg.genre_id) AS genre_count\nFROM movie m\nJOIN movie_genres mg ON m.movie_id = mg.movie_id\nGROUP BY m.title, m.releas
  "SELECT m.title, m.release date, COUNT(mg.genre_id) AS genre_count\nFROM movie m\nJOIN movie_genres mg ON m.movie_id = mg.movie_id\nGROUP BY m.title, m.releas
  "SELECT m.title, m.release date, COUNT(mg.genre_id) AS genre_count\nFROM movie m\nJOIN movie_genres mg ON m.movie_id = mg.movie_id\nGROUP BY m.title, m.releas
  "The top 10 most popular movies, including their titles, release dates, and the number of genres they belong to are:\n1. Minions (Released on June 16, 2015) -
  "SELECT m.title, m.release date, COUNT(mg.genre_id) AS num_genres\nFROM movie m\nJOIN movie_genres mg ON m.movie_id = mg.movie_id\nGROUP BY m.title, m.releas
  "SELECT m.title, m.release date, COUNT(mg.genre_id) AS genre_count\nFROM movie m\nJOIN movie_genres mg ON m.movie_id = mg.movie_id\nGROUP BY m.title, m.releas
  "The top 10 most popular movies, including their titles, release dates, and the number of genres they belong to are:\n1. Minions - June 16, 2015 - 4 genres\n2
  "SELECT m.title, m.release date, COUNT(mg.genre_id) AS genre_count\nFROM movie m\nJOIN movie_genres mg ON m.movie_id = mg.movie_id\nGROUP BY m.title, m.releas
  "SELECT m.title, m.release date, COUNT(mg.genre_id) AS num_genres\nFROM movie m\nJOIN movie_genres mg ON m.movie_id = mg.movie_id\nGROUP BY m.title, m.releas
  "The top 10 most popular movies, including their titles, release dates, and the number of genres they belong to are:\n1. Minions (Released on June 16, 2015) -
  "SELECT m.title, m.release date, COUNT(mg.genre_id) AS genre_count\nFROM movie m\nJOIN movie_genres mg ON m.movie_id = mg.movie_id\nGROUP BY m.title, m.releas

```

Figura 4.12: Traducciones devueltas en base de datos de Películas

Por otra parte, podemos ver que, en algunas ocasiones, el agente falla en devolver la consulta SQL utilizada. Siendo más concretos, no ha sido capaz de devolverlo en 7 ocasiones, 1 de ellas en una consulta errónea y las otras 2 en consultas correctas. Si ejecutamos alguna de las respuestas exitosas sobre la base de datos, podemos comprobar que la sintaxis es correcta y el resultado coincide con el mostrado.

Query Query History

```

1 SELECT m.title, m.release_date, COUNT(mg.genre_id) AS genre_count
2 FROM movie m
3 JOIN movie_genres mg ON m.movie_id = mg.movie_id
4 GROUP BY m.title, m.release_date, m.popularity
5 ORDER BY m.popularity DESC
6 LIMIT 10;

```

Data Output Messages Notifications

	title character varying (1000)	release_date date	genre_count bigint
1	Minions	2015-06-17	4
2	Interstellar	2014-11-05	3
3	Deadpool	2016-02-09	3
4	Guardians of the Galaxy	2014-07-30	3
5	Mad Max: Fury Road	2015-05-13	4
6	Jurassic World	2015-06-09	4
7	Pirates of the Caribbean: The Curse of the Black Pe...	2003-07-09	3
8	Dawn of the Planet of the Apes	2014-06-26	4
9	The Hunger Games: Mockingjay - Part 1	2014-11-18	3
10	Big Hero 6	2014-10-24	5

Figura 4.13: Ejemplo de ejecución de consulta devuelta por el agente en base de datos de Películas

Calculando el tiempo de ejecución de cada una de las peticiones, se ha hallado que, de media, una consulta tarda en ejecutarse 10727.48 milisegundos. Contando únicamente las respuestas correctas, llegamos a la cifra de 10941.27 milisegundos de tiempo de ejecución promedio.

Número de consulta	Tiempo de ejecución (ms)	Estado de la respuesta
1	9872	Éxito
2	10874	Éxito
3	9716	Éxito
4	9004	Éxito
5	10686	Éxito
6	10198	Éxito
7	9706	Éxito
8	6914	Fallo
9	10898	Éxito
10	11532	Éxito
11	10948	Fallo
12	12200	Éxito
13	9617	Fallo
14	12455	Éxito
15	11499	Éxito
16	10230	Éxito
17	9808	Éxito
18	10180	Éxito
19	12611	Éxito
20	12422	Éxito
21	11328	Éxito
22	12016	Éxito
23	11122	Éxito
24	10777	Éxito
25	11574	Éxito

Tabla 4.1: Resultados de ejecución en base de datos de Películas

A continuación, se van a formular 5 preguntas distintas sobre la base de datos. Estas preguntas van a ser previamente resueltas de forma manual y, después, el agente tratará de responderlas. Haremos que cada pregunta sea más compleja que la anterior y, de esta forma, comprobar hasta qué nivel de complejidad es capaz de responder. Las preguntas y las respuestas tanto esperadas como recibidas son las siguientes:

PREGUNTA 1: *"Retrieve the names of 10 genres in the database."*

RESPUESTA ESPERADA 1: *"1. Adventure, 2. Fantasy, 3. Animation, 4. Drama, 5. Horror, 6. Action, 7. Comedy, 8. History, 9. Western, 10. Thriller."*

RESPUESTA RECIBIDA 1: *Adventure, Fantasy, Animation, Drama, Horror, Action, Comedy, History, Western, Thriller*

PREGUNTA 2: *"Find the top 5 movies with the highest revenue."*

RESPUESTA ESPERADA 2: *"The top 5 movies with the highest revenue are Avatar, Titanic, The Avengers, Jurassic World and Furious 7."*

RESPUESTA RECIBIDA 2: *"The top 5 movies with the highest revenue are Avatar, Titanic, The Avengers, Jurassic World, and Furious 7."*

PREGUNTA 3: *"List the names of 5 languages used in the movies along with the count of movies each language is used in, ordered by the number of movies in descending order."*

RESPUESTA ESPERADA 3: *"1. English, 8990; 2. Français, 507, 3. Español, 383, 4. Deutsch, 289, 5. Italiano, 202."*

RESPUESTA RECIBIDA 3: *"The names of 5 languages used in the movies along with the count of movies each language is used in are English (8990 movies), Français (507 movies), Español (383 movies), Deutsch (289 movies), and Italiano (202 movies)."*

PREGUNTA 4: *"Retrieve the names of 5 the persons who have worked in more than one department (e.g., directing, writing, etc.) on the same movie."*

RESPUESTA ESPERADA 4: *1. "A. Kitman Ho", 2. "A.K. Waters", 3. "Aaron Burns", 4. "Aaron Hann", 5. "Aaron Norris."*

RESPUESTA RECIBIDA 4: *"A. Kitman Ho, Aaron Burns, Aaron Dem, Aaron Downing, Aaron Drane."*

PREGUNTA 5: *"Find the person who has worked in the most diverse range of genres, considering both their roles in movie crew and cast, and count the genres they've been involved in along"*

RESPUESTA ESPERADA 5: *"The person who has worked in the most diverse range of genres is Bill Corso with 18 different genres."*

RESPUESTA RECIBIDA 5: *"Quentin Tarantino has worked in the most diverse range of genres with 3 genres."*

<p>Query</p> <p>Find the person who has worked in the most diverse range of genres, considering both their roles in movie crew and cast, and count the genres they've been involved in along</p>	<p>Answer</p> <p>Quentin Tarantino has worked in the most diverse range of genres with 3 genres.</p>	<p>SQL Query</p> <pre>SELECT p.person_name, COUNT(DISTINCT mg.genre_id) AS num_genres FROM person p JOIN movie_crew mc ON p.person_id = mc.person_id JOIN movie m ON mc.movie_id = m.movie_id JOIN movie_genres mg ON m.movie_id = mg.movie_id GROUP BY p.person_name ORDER BY num_genres DESC LIMIT 1.</pre>
<p>Query</p> <p>Retrieve the names of 5 the persons who have worked in more than one department (e.g., directing, writing, etc.) on the same movie.</p>	<p>Answer</p> <p>A. Kitman Ho, Aaron Burns, Aaron Dem, Aaron Downing, Aaron Drane.</p>	<p>SQL Query</p> <pre>SELECT p.person_name FROM movie_crew mc JOIN person p ON mc.person_id = p.person_id GROUP BY p.person_name HAVING COUNT(DISTINCT mc.department_id) > 1 LIMIT 5;</pre>
<p>Query</p> <p>List the names of 5 languages used in the movies along with the count of movies each language is used in, ordered by the number of movies in descending order.</p>	<p>Answer</p> <p>The names of 5 languages used in the movies along with the count of movies each language is used in are English (8990 movies), Franu00e7ais (507 movies), Espau00f1ol (383 movies), Deutsch (289 movies), and Italiano (202 movies).</p>	<p>SQL Query</p> <pre>SELECT language.language_name, COUNT(movie_languages.movie_id) AS movie_count FROM language JOIN movie_languages ON language.language_id = movie_languages.language_id GROUP BY language.language_name ORDER BY movie_count DESC LIMIT 5;</pre>
<p>Query</p> <p>Find the top 5 movies with the highest revenue.</p>	<p>Answer</p> <p>Avatar, Titanic, The Avengers, Jurassic World, Furious 7.</p>	<p>SQL Query</p> <pre>SELECT title, revenue FROM movie ORDER BY revenue DESC LIMIT 5;</pre>
<p>Query</p> <p>Retrieve the names of 10 genres in the database.</p>	<p>Answer</p> <p>Adventure, Fantasy, Animation, Drama, Horror, Action, Comedy, History, Western, Thriller.</p>	<p>SQL Query</p> <pre>SELECT genre_name FROM genre LIMIT 10;</pre>

Figura 4.14: Ejecución de consultas de distinto nivel en base de datos de Películas

Podemos ver como el agente ha sido capaz de responder correctamente a 4 de 5 preguntas, siendo la equivocada la última. Tras reintentar la consulta una vez más, el agente fue capaz de responder correctamente a dicha pregunta. Aun así, la quinta pregunta se contabilizará como errónea, ya que el agente debería de ser capaz de responder bien en un primer intento.

Concluimos que, sobre la base de datos de películas, el agente ha sido capaz de responder a preguntas de dificultad intermedia a una velocidad bastante rápida. Por otra parte, el agente puede resolver consultas complejas si lo ejecutamos varias veces y comparamos los resultados con los devueltos por la base de datos.

4.3.2 Ejemplos en base de datos de TITSA

La anterior base de datos ha sido trabajada numerosas veces por distintos usuarios en internet, tratándose de una común base de datos de ejemplo para realizar diferentes pruebas. Es por esta razón que se realizaron diferentes pruebas sobre la base de datos de la matriz origen-destino de viajes de marzo de TITSA. Esta base de datos se puede considerar única, pues la única instancia que hay de esta es como archivo CSV en la web de Open Data de TITSA y, por lo tanto, no existen modelos que hayan podido recoger

esta información de referencia para entrenar.

De forma similar a la base de datos de películas, se han formulado 8 preguntas con sus esperadas respuestas. De esta forma, podremos evaluar al modelo en función a la información devuelta. Las preguntas han sido obtenidas por un experto en la matriz de origen-destino, de forma que las preguntas representan consultas típicas que un usuario podría realizar sobre la base de datos. Las preguntas y las respuestas esperadas y recibidas son las siguientes:

PREGUNTA 1: *"¿Cuál es el número promedio de paradas entre dos municipios específicos (ARAFO y CANDELARIA) durante la hora pico?"*

RESPUESTA ESPERADA 1: *"El número promedio de paradas entre ARAFO y CANDELARIA durante la hora pico es de 17.34 "*

RESPUESTA RECIBIDA 1: *"The average number of stops between ARAFO and CANDELARIA during peak hours is approximately 20.83."*

PREGUNTA 2: *"¿Cuál es la línea de transporte público más utilizada durante los fines de semana?"*

RESPUESTA ESPERADA 2: *"La línea de transporte público más utilizada durante los fines de semana es la línea 1"*

RESPUESTA RECIBIDA 2: *"The most used public transportation line during weekends is line with id 1, with a total of 54098 passengers."*

PREGUNTA 3: *"¿Cuántos viajes se registran en un día laborable promedio entre dos municipios diferentes?"*

RESPUESTA ESPERADA 3: *"En un día laborable promedio entre dos municipios diferentes se registran 444944 viajes"*

RESPUESTA RECIBIDA 3: *1044585.*

PREGUNTA 4: *"¿Cuál es el tiempo de viaje promedio entre dos municipios en las horas punta de la mañana?"*

RESPUESTA ESPERADA 4: *1. "El tiempo de viaje promedio entre dos municipios en las horas punta de la mañana (7 y 8) es de 10 minutos"*

RESPUESTA RECIBIDA 4: *"The average travel time between two different municipalities between 7 and 8 in the morning is 7 minutes."*

PREGUNTA 5: *"¿Qué municipio tiene la mayor cantidad de viajes de entrada durante los días laborables?"*

RESPUESTA ESPERADA 5: *"El municipio con la mayor cantidad de viajes de entrada durante los días laborales es SANTA CRUZ DE TENERIFE, con 379857 viajes"*

RESPUESTA RECIBIDA 5: *"SANTA CRUZ DE TENERIFE with 379857 inbound trips during weekdays."*

PREGUNTA 6: *"¿Cómo varía el número de viajes entre dos municipios en función de si es un día laborable o un día festivo?"*

RESPUESTA ESPERADA 6: *"El número de viajes entre dos municipios suele ser mayor en días festivos que en días laborales."*

RESPUESTA RECIBIDA 6: *"Hay más viajes en días festivos que en días laborales"*

PREGUNTA 7: *"¿Cuál es el tipo de día (laborable, fin de semana, festivo) con mayor demanda de transporte público en general?"*

RESPUESTA ESPERADA 7: *"Los días laborables tienen mayor demanda de transporte público en general, con un total de 3428192 pasajeros"*

RESPUESTA RECIBIDA 7: *"Labor day with 1,382,197 passengers."*

PREGUNTA 8: *"¿Cuál es la línea de transporte público menos utilizada durante las horas no laborables?"*

RESPUESTA ESPERADA 8: *"Durante las horas no laborables, la línea menos usada es la 940"*

RESPUESTA RECIBIDA 8: *"The least used public transportation line during non-working hours has the id 940 with a total of 17 passengers."*

Podemos ver cómo el agente ha sido capaz de responder correctamente las preguntas más sencillas. Sin embargo, cuando se trata de consultas complejas o rebuscadas, el agente no es capaz de llegar al resultado esperado. La herramienta ha conseguido responder correctamente 4/8 veces, un 50% de acierto. En algunos casos, se ha tenido que reformular la pregunta, ya que el agente se centraba en alguna palabra del enunciado y comenzaba la búsqueda por columnas similares a esta palabra aunque no tuviese que ver con la respuesta final.

En cuanto a tiempos de ejecución, no varían mucho de los ya calculados en los anteriores ejemplos de ejecución. Estos tiempos rondan los 10 segundos desde que se envía la pregunta y se recibe la respuesta. De esta forma podemos comprobar que la velocidad de respuesta del agente es independiente al tipo de base de datos a la que se pueda enfrentar.

Type your question

When referring to a municipio, it refers to the column parada.municipio

Generate SQL Sample Question

<p>Query</p> <p>¿Qué municipio tiene la mayor cantidad de viajes de entrada durante los días laborables?</p>	<p>Answer</p> <p>SANTA CRUZ DE TENERIFE with 379857 inbound trips during weekdays.</p>	<p>SQL Query</p> <pre>SELECT paradas.municipio, COUNT(*) AS num_trips FROM viajes_marzo_laborables JOIN paradas ON viajes_marzo_laborables.id_parada_entrada_viaje_actual = paradas.id_parada GROUP BY paradas.municipio ORDER BY num_trips DESC LIMIT 1;</pre>
<p>Query</p> <p>¿Qué municipio tiene la mayor cantidad de viajes de entrada durante los días laborables?</p>	<p>Answer</p> <p>The municipality with the highest number of inbound trips during weekdays is the one with the id_parada_entrada_viaje_actual 9181, with a total of 69940 trips.</p>	<p>SQL Query</p> <pre>SELECT id_parada_entrada_viaje_actual, COUNT(*) AS total_trips FROM viajes_marzo_laborables GROUP BY id_parada_entrada_viaje_actual ORDER BY total_trips DESC LIMIT 1;</pre>
<p>Query</p> <p>¿Cuál es la línea de transporte público más utilizada durante los fines de semana?</p>	<p>Answer</p> <p>The most used public transportation line during weekends is line with id 1, with a total of 54098 passengers.</p>	<p>SQL Query</p> <pre>SELECT id_linea_viaje_actual, COUNT(*) AS total_pasajeros FROM viajes_marzo_no_laborales WHERE tipo_dia = 'Sabado' GROUP BY id_linea_viaje_actual ORDER BY total_pasajeros DESC LIMIT 1;</pre>

Figura 4.15: Ejecución de consultas en base de datos de TITSA

En la imagen se visualizan algunas de las preguntas realizadas anteriormente. En la segunda respuesta se puede ver cómo la respuesta que devuelve el agente es incorrecta. Esto es porque confunde el término de municipio con el de parada. Para ello se hace uso del prompt personalizable, en el cual le especificamos al agente que, al referirse a un municipio, se trata de la columna parada.municipio. Al realizar la pregunta de nuevo, podemos ver como el resultado que da es el correcto.

Todo el código desarrollado en este proyecto se puede encontrar en <https://github.com/FebeFarina/sql-translator-API>

Capítulo 5. Conclusiones y líneas futuras

5.1 Conclusiones

Cómo resumen, se ha conseguido desarrollar un prototipo de herramienta que es capaz de leer una pregunta en lenguaje natural sobre una base de datos y responder, además de proporcionar el equivalente a la pregunta en SQL.

En primer lugar, la herramienta ha sido capaz de desenvolverse ante una base de datos sobre películas. Ha demostrado tener un considerable porcentaje de acierto y, de haber algún error, ejecutar de nuevo la consulta o reformular la misma resolvería el problema. Por otra parte, destacar la velocidad de respuesta de la herramienta, pudiendo responder cualquier pregunta en aproximadamente 10 segundos.

Luego, las pruebas sobre la base de datos creada a partir de la matriz de origen destino sobre viajes en la empresa TITSA, el agente no ha dado los mismos resultados que en el anterior ejemplo. Esto puede deberse a distintos factores, el primero de todos al uso que se le haya dado a la base de datos de películas para entrenar a modelos para el tratamiento de bases de datos relacionales. Tratándose de una famosa base de datos de ejemplo, no sería raro relacionar el alto índice de acierto con la familiaridad que tendrá el modelo sobre esta base de datos. En cambio, la base de datos creada sobre la matriz de origen destino de TITSA ha sido creada a partir de distintos archivos CSV, por lo que la base de datos empleada no se encuentra publicada en ningún lugar.

Otro de los motivos por los que no ha dado los resultados esperados es por la importancia que tiene la formulación de las preguntas. El mínimo cambio que podamos hacer sobre la pregunta realizada puede alterar completamente el resultado obtenido. Una pregunta debe centrarse en los nombres de las columnas de la base de datos e intentar no usar ningún sinónimo. De la misma forma, es conveniente hacer preguntas cortas y directas para que el agente sepa dónde buscar la información y lo que espera recibir el usuario.

Finalmente, hay que tener especial cuidado, mientras esté a nuestro alcance, de estructurar correctamente la base de datos. Esto es, crear nombres identificables para las tablas y columnas, crear las relaciones correctas entre tablas y asignarles los tipos de valores correctos a los datos recogidos.

En conclusión, la herramienta ha demostrado solucionar el problema planteado ante bases de datos sencillas y preguntas no muy complejas. Asimismo, el agente funciona especialmente bien si la base de datos está bien estructurada y los nombres de las tablas y columnas son reconocibles. Del mismo modo hay que cuidar cómo se

formulan las preguntas y tratar de ser lo más concreto posible.

5.2 Líneas futuras

Tratándose de un prototipo, esta herramienta aún necesita de tiempo y recursos para llegar a una versión totalmente funcional y completa. Una funcionalidad que podría refinar el modelo para que responda de manera más acertada sería un entrenamiento previo con varios ejemplos de preguntas y respuestas sobre una base de datos específica. De esta forma podríamos especializar el modelo para que funcione bien con un conjunto de datos concretos.

Esta herramienta hace uso del modelo GPT-4 para resolver las consultas. Tratándose de un modelo de propósito general, podríamos mejorar la eficacia de la API empleando un modelo que, previamente, haya sido entrenado sobre bases de datos relacionales y consultas SQL. De esta forma, el agente tendría un menor índice de fallo e, incluso, podría entrenarse sobre una base de datos en concreto y especializarse en consultas sobre la misma.

Por otra parte, cabe destacar que el uso de la API de OpenAI no garantiza que los datos usados por la herramienta no puedan ser usados por la empresa para otros fines. Esto puede resultar problemático ya que no cumpliría con el Reglamento General de Protección de Datos. Es por esto que se plantea la siguiente solución: emplear un modelo que se ejecute en una máquina local. Esta solución puede estar limitada por las especificaciones del ordenador en el que se está ejecutando, pero de esta forma se puede garantizar que los datos no sean usados por terceros.

Finalmente, la interfaz sobre la que se utiliza la aplicación podría orientarse a un bot de telegram capaz de hacer llamadas a la API. Sin embargo, esta solución ya ha sido contemplada por el TFG de Adrián Hernández Suárez mencionado al inicio de esta memoria.

Chapter 5. Summary and Conclusions

5.1 Conclusions

In summary, a prototype tool has been developed that is capable of reading a natural language query from a database and responding, as well as providing the equivalent of the query in SQL. Even with a small failure rate and a considerable response time, the tool has shown to solve queries of medium complexity.

First, the tool has been able to cope with a movie database. It has shown to have a considerable percentage of success and, if there were any error, running the query again or rephrasing it would solve the problem. On the other hand, the speed of response of the tool is remarkable, being able to answer any question in approximately 10 seconds.

Then, on behalf of the tests on the database created from the origin-destination matrix on trips in the TITSA company, the agent has not given the same results as in the previous example. This may be due to several factors, the first of all to the use given to the movie database to train models for the treatment of relational databases. Being a famous example database, it would not be unusual to relate the high success rate to the familiarity that the model will have with this database. On the other hand, the database created on TITSA's origin-destination matrix has been created from different CSV files, so the database used is not published anywhere.

Another reason why it has not given the expected results is because of the importance of the formulation of the questions. The slightest change we can make to the question can completely alter the result obtained. A question should focus on the names of the columns in the database and try not to use any synonyms. In the same way, it is convenient to ask short and direct questions so that the agent knows where to look for the information and what the user expects to receive.

Finally, we must take special care, while it is within our reach, to structure the database correctly. That is, create identifiable names for tables and columns, create the correct relationships between tables and assign the correct value types to the data collected.

In conclusion, the tool has proven to solve the problem with simple databases and not very complex queries. Furthermore, the agent works particularly well if the database is well structured and the tables and column names are recognizable. In the same way, it is important to be careful how the questions are formulated and to try to be as specific as possible.

5.2 Future sights

This tool still needs time and resources to come to a complete and functional version, as it is a prototype. One functionality that could refine the model in order to answer in a more exact way would be a previous training with different examples of questions and answers about a specific database. In this way, we could specialize the model so it works well with a particular dataset.

This tool makes use of the GPT-4 model to resolve queries. Being a general purpose model, we could improve the API efficiency by using a model that has been previously trained on relational databases and SQL queries. In this way, the agent would have a lower failure rate and could even be trained on a specific database and specialize in queries on it.

On the other hand, it is worth mentioning that the use of the OpenAI API doesn't guarantee that the data used by the tool couldn't be used by the company for other purposes. This could be problematic as it wouldn't follow the General Data Protection Regulation. For this reason, the following solution is proposed: use a model executing from a local machine. This solution would be limited by the computer specs, but this method could guarantee that the data are not used by third parties.

Finalmente, la interfaz sobre la que se utiliza la aplicación podría orientarse a un bot de telegram capaz de hacer llamadas a la API. Sin embargo, esta solución ya ha sido contemplada por el TFG de Adrián Hernández Suárez mencionado al inicio de esta memoria.

Capítulo 6. Presupuesto

6.1 Costes hardware

Tipos	Descripción	Coste
Ordenador	Ordenador desde el que se ha desarrollado la aplicación	250€

Tabla 7.1: Costes hardware

6.2 Costes Recursos Humanos

Horas de trabajo	Coste/Hora	Coste total
250h	25€/h	6.250€

Tabla 7.2: Costes Recursos Humanos

6.3 Costes totales del proyecto

Coste hardware	Coste RRHH	Coste proyecto
250€	6.250€	6.500€

Tabla 7.3: Costes totales del proyecto

Bibliografía

- [1] Y. Chang *et al.*, «A Survey on Evaluation of Large Language Models», *ACM Trans. Intell. Syst. Technol.*, vol. 15, n.º 3, p. 39:1-39:45, mar. 2024, doi: 10.1145/3641289.
- [2] B. Marr, «Beyond ChatGPT: 14 Mind-Blowing AI Tools Everyone Should Be Trying Out Now», *Forbes*. Accedido: 21 de abril de 2024. [En línea]. Disponible en:
<https://www.forbes.com/sites/bernardmarr/2023/02/28/beyond-chatgpt-14-mind-blowing-ai-tools-everyone-should-be-trying-out-now/>
- [3] «Top 5 SQL AI Tools to Ease Writing SQL 2023», *Bytebase*. Accedido: 21 de abril de 2024. [En línea]. Disponible en:
<https://bytebase.com/blog/top-sql-ai-tools/>
- [4] «Uso de modelos de lenguaje y Machine Learning para la consulta y predicción de pasajeros en TITSA» Disponible en:
<http://riull.ull.es/xmlui/handle/915/33396>.
- [5] «Large language model», *Wikipedia*. 28 de abril de 2024. Accedido: 28 de abril de 2024. [En línea]. Disponible en:
https://en.wikipedia.org/w/index.php?title=Large_language_model&oldid=1221194720#cite_note-Bowman-2
- [6] «Speaking in Tongues», *Science History Institute*. Accedido: 28 de abril de 2024. [En línea]. Disponible en:
<https://www.sciencehistory.org/stories/magazine/speaking-in-tongues/>
- [7] A. Vaswani *et al.*, «Attention Is All You Need». *arXiv*, 1 de agosto de 2023. doi: 10.48550/arXiv.1706.03762.
- [8] OpenAI *et al.*, «GPT-4 Technical Report». *arXiv*, 4 de marzo de 2024. doi: 10.48550/arXiv.2303.08774.
- [9] «Base de datos relacional», *Wikipedia, la enciclopedia libre*. 16 de enero de 2024. Accedido: 5 de mayo de 2024. [En línea]. Disponible en:
https://es.wikipedia.org/w/index.php?title=Base_de_datos_relacional&oldid=157283674
- [10] S. Shinde, «Basics Types of Queries in SQL», *Medium*. Accedido: 5 de mayo de 2024. [En línea]. Disponible en:
<https://medium.com/@sohamshinde156/basics-types-of-queries-in-sql-5f163ee44adf>
- [11] R. T. Fielding, M. Nottingham, y J. Reschke, «HTTP Semantics», *Internet Engineering Task Force, Request for Comments RFC 9110*, jun. 2022. doi: 10.17487/RFC9110.
- [12] «langchain-ai/langchainjs». *LangChain*, 30 de abril de 2024. Accedido:

- 30 de abril de 2024. [En línea]. Disponible en:
<https://github.com/langchain-ai/langchainjs>
- [13] «Express 4.x - Referencia de API». Accedido: 3 de mayo de 2024. [En línea]. Disponible en: <https://expressjs.com/es/4x/api.html>
- [14] «PostgreSQL 16.2 Documentation», PostgreSQL Documentation. Accedido: 3 de mayo de 2024. [En línea]. Disponible en: <https://www.postgresql.org/docs/16/index.html>
- [15] «Sample Database: Movies (ERD and SQL)», Database Star. Accedido: 3 de mayo de 2024. [En línea]. Disponible en: <https://www.databasestar.com/sample-database-movies/>
- [16] «Instrucción_Tecnica_Matriz_OD.pdf». Accedido: 5 de mayo de 2024. [En línea]. Disponible en: https://portaldetransparencia.titsa.com/images/OPEN_DATA/MATRIZ_OD/Instrucci%C3%B3n_Tecnica_Matriz_OD.pdf