Departamento de
Ingeniería Industrial
Universidad de Concepción

"Optimización de Procesos con respuestas generadas con Inteligencia Artificial en CIO De Forestal Arauco S.A"

POR

Gabriel Eduardo Valderrama Chesta

Memoria de Título presentada a la Facultad de Ingeniería de la Universidad de Concepción para optar al título de profesional de Ingeniero Civil Industrial.

Profesor guía
Hernaldo Del Carmen Reinoso Alarcón

Profesional Supervisor
Mauricio Murgas

JULIO 2025

Concepción (Chile)

Sumario

En la industria forestal, caracterizada por su complejidad logística y grandes volúmenes de datos operacionales, existe una brecha significativa en la capacidad de acceder, consultar y analizar información en tiempo real. Los sistemas actuales presentan limitaciones para integrar múltiples fuentes de datos, responder preguntas dinámicas y generar visualizaciones útiles para la toma de decisiones operativas.

El objetivo general de este proyecto fue desarrollar un sistema conversacional inteligente, capaz de interactuar en lenguaje natural con una base de datos forestal compleja, generando consultas automáticas, análisis personalizados y visualizaciones dinámicas, con especial foco en apoyar la planificación y ejecución operacional del negocio forestal de ARAUCO.

El proyecto se desarrolló bajo un enfoque iterativo, siguiendo un ciclo continuo de pruebas y mejoras. Para ello se combinaron buenas prácticas para redactar instrucciones a la IA, tareas automáticas con la plataforma n8n, el uso de modelos de lenguaje de última generación y la implementación de servicios en la nube (Supabase, Railway y Streamlit). Se creó un flujo de trabajo para organizar los datos, una estructura por módulos que colaboran entre sí y una página web conversacional conectada a herramientas de gráficos y a las bases de datos.

Como resultado, se construyó un agente conversacional SQL llamado Tronix, capaz de interpretar preguntas complejas y generar consultas SQL multitabla con alta precisión. El sistema permite obtener información estratégica como stock, producción, despachos y proyecciones operativas por zona o especie, todo en lenguaje natural. Además, se integraron herramientas gráficas que permiten visualizar dinámicamente los resultados y generar dashboards personalizados. Su viabilidad operativa fue demostrada con más de 1.500 consultas procesadas, un costo por consulta menor a 3 centavos de dólar y una alta aceptación por parte de usuarios reales.

Abstract

In the forestry industry, characterized by its logistical complexity and large volumes of operational data, there is a significant gap in the ability to access, query, and analyze information in real time. Current systems are limited in integrating multiple data sources, answering dynamic questions, and generating visualizations useful for operational decision-making.

The overall objective of this project was to develop an intelligent conversational system capable of interacting in natural language with a complex forestry database, generating automated queries, customized analyses, and dynamic visualizations, with a special focus on supporting the planning and operational execution of ARAUCO's forestry business.

The project was developed using an iterative approach, following a continuous cycle of testing and improvements. This combined best practices for writing instructions for AI, automated tasks using the n8n platform, the use of state-of-the-art language models, and the implementation of cloud services (Supabase, Railway, and Streamlit). A workflow was created to organize the data, along with a structure based on collaborative modules and a conversational website connected to graphing tools and databases.

As a result, a conversational SQL agent called Tronix was built, capable of interpreting complex queries and generating multi-table SQL queries with high accuracy. The system provides strategic information such as inventory, production, shipments, and operational projections by region or species, all in natural language. In addition, graphical tools were integrated to dynamically visualize results and generate customized dashboards. Its operational viability was demonstrated with more than 1,500 queries processed, a cost per query of less than 3 cents, and high acceptance by real users.

Contenido

Agradecimientos

Agradezco y dedico esta memoria y todo el proceso relacionado a este trabajo a mis padres, Ana María Chesta y Francisco Valderrama, quienes me enseñaron el significado de tenacidad, esfuerzo y resiliencia. Eternamente Agradecido.

Agradezco a Leonardo de la Fuente y Mauricio Murgas, quien me prestó apoyo incondicional desde la empresa.

1. Introducción

La Central Integrada de Operaciones (CIO) de Forestal Arauco constituye el "centro de mando" del negocio forestal: allí se decide —prácticamente en tiempo real— qué madera se corta, desde qué predio se envía y a qué planta o aserradero se destina, de modo que toda la cadena productiva fluya sin interrupciones. El éxito de esta misión estratégica depende de una materia prima intangible pero crítica: información fiable, granular y oportuna sobre inventarios, despachos y producción diaria.

No obstante, gran parte de esos datos continúa dispersa en planillas Excel heterogéneas, sin un modelo relacional compartido ni una interfaz que permita consultarlos de manera directa. Esta fragmentación genera cuellos de botella operativos: los analistas tardan horas en combinar archivos, los planificadores dependen de especialistas para correr consultas y la dirección carece de indicadores en línea para evaluar desviaciones o anticipar rupturas de stock. El resultado es una brecha creciente entre la velocidad del negocio y la capacidad analítica de sus sistemas.

Frente a este escenario, se plantea la necesidad de un nuevo paradigma de interacción: un agente conversacional que permita consultar la base forestal en lenguaje natural, traduzca las preguntas a SQL multitabla, ejecute la consulta sobre datos reales y devuelva la respuesta —tabular o visual— en segundos. Este proyecto propone justamente eso: Un agente SQL, un sistema inteligente que combina un modelo de lenguaje de última generación, un pipeline ETL robusto y una arquitectura modular en la nube para poner los datos operacionales "al alcance de la voz".

El desarrollo del Agente SQL aborda tres desafíos clave.

1. **Ingeniería de datos**: se diseñó y automatizó un proceso ETL que normaliza más de diez fuentes distintas, eliminando inconsistencias y convirtiendo archivos Excel en tablas PostgreSQL optimizadas.
2. **Arquitectura escalable**: el sistema se desplegó como microservicios independientes (backend conversacional en n8n, microservicio de gráficos en Streamlit y frontend React conectado a Supabase), garantizando alta disponibilidad y bajo costo operativo.
3. **Inteligencia conversacional**: se diseñó un prompt estructurado que guía al modelo en la detección de intención, generación de SQL y creación de visualizaciones dinámicas, todo con trazabilidad y sin "alucinaciones" de datos.

Con ello, el agente SQL transforma un ecosistema de dashboards estáticos y procesos manuales en una plataforma conversacional donde operarios, planificadores y gerentes pueden:

- Preguntar “¿Cuál es el stock de podado en Viñales hoy?” y recibir la cifra exacta más un gráfico de tendencia.
- Comparar la planificación de despachos con la ejecución real por zona y producto.
- Proyectar la evolución de inventarios aplicando consumos calendar-aware y escenarios simulados.
- Extracción de información en unos pocos segundos.

Además de su valor práctico, el proyecto contribuye académicamente al demostrar cómo los LLM pueden integrarse de forma segura y eficiente con bases de datos empresariales complejas, allanando el camino para nuevas aplicaciones de IA generativa en sectores intensivos en datos operativos.

Los capítulos siguientes profundizan en el marco teórico que sustenta la solución, la metodología híbrida empleada, el diseño del pipeline de datos y la evaluación comparativa de modelos de IA; finalmente se presentan los resultados, conclusiones y lineamientos para futuras líneas de trabajo.

Para facilitar la comprensión de ciertos conceptos técnicos utilizados a lo largo del documento, se incluye un glosario en el Anexo N°9 con definiciones breves sobre términos clave como LLM, SQL, API, JSON, entre otros.

2. Marco Teórico

2.1 Inteligencia Artificial Generativa

Los sistemas de inteligencia artificial conversacional han evolucionado significativamente con el auge de los modelos de lenguaje de gran escala (LLMs), abriendo nuevas posibilidades en la interfaz entre lenguaje natural y bases de datos estructuradas. Uno de los avances más relevantes en este ámbito es el desarrollo de sistemas Text-to-SQL (también conocidos como NL2SQL), cuyo propósito es transformar preguntas formuladas en lenguaje natural en consultas SQL ejecutables sobre bases de datos relacionales.

Este subcampo ha sido impulsado por la necesidad de democratizar el acceso a la información almacenada en sistemas complejos, permitiendo que usuarios sin conocimientos técnicos puedan interactuar directamente con bases de datos. Según Yu et al. (2018), el benchmark Spider representa uno de los marcos de evaluación más exigentes y representativos, ya que incluye tareas de generación de consultas multitabla, con filtros, agregaciones y distintos niveles de complejidad semántica. A diferencia de otros conjuntos de datos sintéticos, Spider exige que los modelos comprendan el esquema relacional (columnas, tipos y relaciones entre tablas) y que adapten su salida en función de dicho contexto (*schema-aware parsing*).

Los desafíos principales del problema Text-to-SQL incluyen:

- Identificar la intención del usuario y la operación solicitada (selección, filtrado, agregación, comparación).
- Mapear términos del lenguaje natural a columnas, tablas o alias específicos.
- Comprender relaciones entre entidades y decidir cuándo aplicar operadores como JOIN, GROUP BY o ORDER BY.
- Generar consultas que sean correctas tanto en su lógica como en su sintaxis.

La investigación reciente ha explorado múltiples enfoques para resolver esta tarea, desde modelos entrenados específicamente sobre bases de datos anotadas (como WikiSQL, Spider o BirdSQL), hasta técnicas basadas en instrucciones estructuradas (*prompt engineering*) para guiar a modelos de propósito general. Además, se han desarrollado herramientas complementarias que permiten mejorar el rendimiento en entornos reales, como asistentes de autocompletado, analizadores de esquemas o marcos de evaluación interactiva. Estas técnicas han permitido aumentar significativamente la

precisión en la generación de SQL, reduciendo errores semánticos y mejorando la usabilidad de estos sistemas en contextos empresariales.

Soporte tecnológico para sistemas NL2SQL

El desarrollo de agentes de inteligencia artificial capaces de generar consultas SQL y operar de forma autónoma sobre bases de datos estructuradas requiere una infraestructura tecnológica robusta y flexible. En este contexto, los modelos de base de datos como servicio (DBaaS) han transformado la gestión de datos empresariales, al ofrecer escalabilidad automática, alta disponibilidad y menores costos operativos. Las soluciones basadas en PostgreSQL resultan especialmente relevantes por su capacidad transaccional y su compatibilidad con estructuras relacionales complejas, mientras que los enfoques NoSQL orientados a documentos ofrecen ventajas en flexibilidad y rendimiento para ciertos escenarios de datos semi-estructurados. Complementariamente, las plataformas modernas de despliegue en la nube incorporan principios de orquestación automática, integración continua y escalabilidad bajo demanda, facilitando la implementación de arquitecturas distribuidas con integración nativa a repositorios de código. Esta infraestructura resulta clave para habilitar sistemas distribuidos que permite ejecutar flujos conversacionales, consultas dinámicas y visualizaciones de datos de manera eficiente y segura. Tal como señala Marr (2018), el diseño de agentes inteligentes en entornos productivos exige arquitecturas capaces de adaptarse a condiciones cambiantes, con mecanismos de validación de datos, tolerancia a fallos y manejo autónomo de excepciones, garantizando así la continuidad operativa del sistema.

2.2 Marco teórico de Metodologías de Desarrollo de Software

Desarrollo Ágil con Scrum

El marco de trabajo Scrum, documentado por Schwaber y Sutherland (2020) en la Guía Oficial de Scrum, proporciona un enfoque iterativo e incremental para el desarrollo de software. Este marco se caracteriza por ciclos breves de desarrollo con retroalimentación continua, lo que permite adaptarse rápidamente a cambios en los requerimientos y validar funcionalidades de manera temprana en el proceso de desarrollo.

Minería de Datos con CRISP-DM

El estándar CRISP-DM (Cross Industry Standard Process for Data Mining), establecido en el año 2000, define un proceso sistemático para proyectos de minería de datos que incluye seis fases fundamentales: comprensión del negocio, comprensión de los datos, preparación de datos, modelado,

evaluación e implementación. Este modelo proporciona una metodología robusta para abordar proyectos que involucran el análisis y transformación de grandes volúmenes de datos.

Experiencia de Usuario Conversacional

McTear (2016) estableció fundamentos importantes para el diseño de experiencias de usuario conversacionales, enfocándose en la creación de interfaces que emulen la conversación natural mientras mantienen la eficiencia operativa. Estos principios son especialmente relevantes para el desarrollo de sistemas que deben ser utilizados por usuarios con diferentes niveles de experiencia técnica.

Automatización de Procesos de Datos

La automatización de procesos mediante motores de flujos de trabajo (workflow engines) se ha consolidado como un paradigma clave para orquestar tareas repetitivas, reducir errores humanos y aumentar la eficiencia operativa en contextos industriales. Según Tanenbaum y van Steen (2017), estos sistemas permiten ejecutar procesos complejos de manera coordinada y con mínima intervención manual, garantizando consistencia y reproducibilidad en entornos de producción.

En paralelo, las plataformas de integración visual modernas permiten construir flujos dinámicos, conectarse con APIs externas, manejar errores condicionales y extender funcionalidades mediante scripting personalizado. Estas herramientas, como n8n, combinan lo mejor de ambos mundos: interfaces visuales accesibles y capacidad de integración programática avanzada, facilitando la conexión entre sistemas heterogéneos (bases de datos, microservicios, motores de IA) dentro de arquitecturas modulares.

Procesos ETL (Extract, Transform, Load)

Los procesos ETL constituyen una metodología fundamental para la integración de datos desde múltiples fuentes heterogéneas hacia sistemas de almacenamiento estructurados. Kimball y Caserta (2004) establecieron los principios fundamentales de ETL, definiendo tres fases críticas: extracción de datos desde fuentes originales, transformación mediante reglas de negocio y validación de calidad, y carga hacia el sistema de destino. Esta metodología es especialmente relevante cuando se trabaja con datos dispersos en múltiples formatos y estructuras inconsistentes.

Los procesos de transformación de datos, según Rahm y Do (2000), requieren la implementación de estrategias sistemáticas para el manejo de inconsistencias, valores nulos, estandarización de formatos y validación de integridad referencial. Estas técnicas son esenciales para garantizar la calidad y confiabilidad de los datos en sistemas de análisis y consulta.

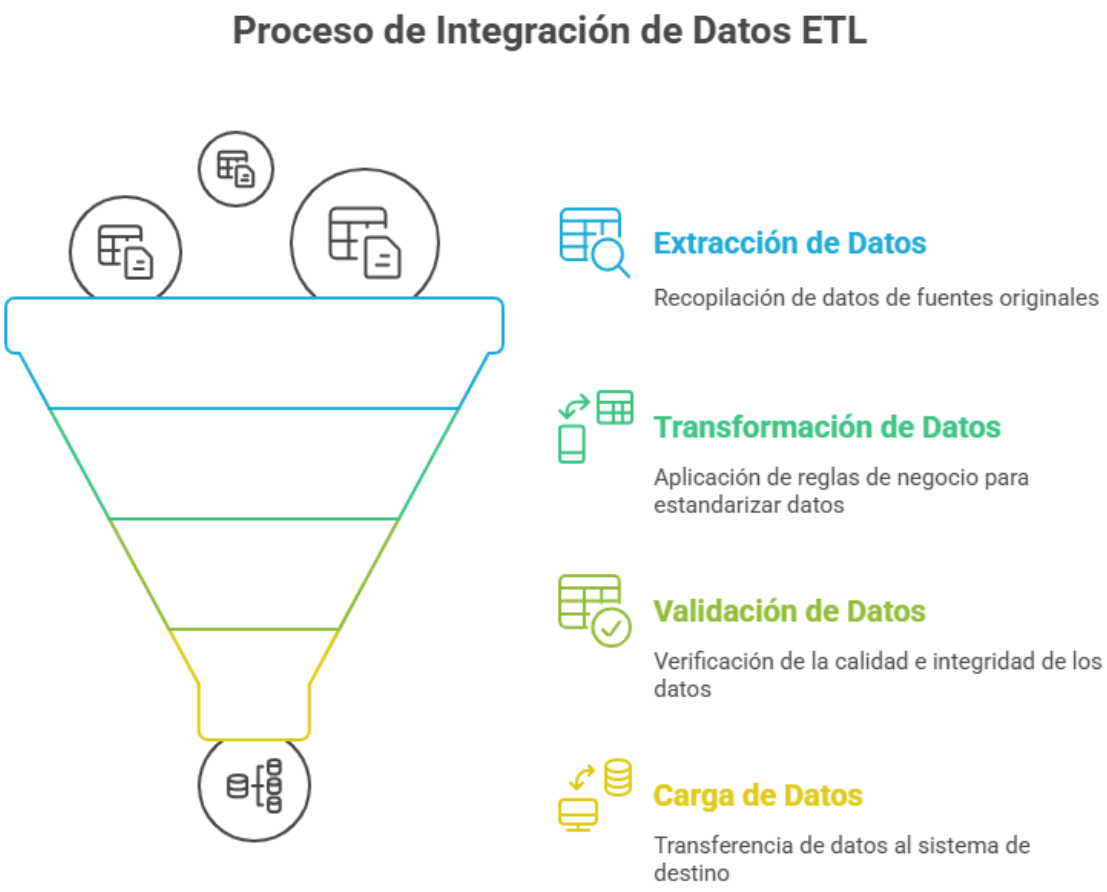


Figura 2.1: Fases del proceso ETL aplicado en el proyecto.

Fuente: Elaboración Propia

3. Metodología

El presente proyecto se enmarca dentro de una investigación aplicada con enfoque tecnológico, orientada a resolver un problema operativo real en el ámbito de la gestión forestal digital. Para abordar esta problemática, se adoptó una metodología híbrida que combina principios del desarrollo ágil, minería de datos, diseño centrado en el usuario e inteligencia artificial conversacional.

3.1 Enfoque Metodológico General

El desarrollo del sistema se basó en una combinación flexible de marcos metodológicos como Scrum, CRISP-DM y enfoques centrados en el usuario. Se trabajó de forma iterativa, integrando funcionalidades en ciclos breves con retroalimentación constante. El tratamiento de datos siguió una lógica sistemática inspirada en CRISP-DM, mientras que el diseño se orientó a la experiencia real de los usuarios. La aplicación de estos enfoques no fue estrictamente secuencial, sino adaptada orgánicamente a los requerimientos específicos del proyecto. En términos prácticos:

- **Scrum** fue adoptado de forma intuitiva como marco de trabajo ágil, estructurando el desarrollo en ciclos breves, con revisiones frecuentes y mejoras progresivas. Aunque no se utilizó una herramienta formal como Jira ni se implementaron todos los artefactos oficiales, se respetó la lógica iterativa e incremental propia del enfoque ágil. El proceso no fue documentado formalmente en cada etapa, ya que se fue construyendo de manera orgánica, guiado por un objetivo final claro, pero permitiendo la adaptación continua. Esta flexibilidad facilitó ajustes rápidos frente a desafíos emergentes, priorizando la funcionalidad del sistema por sobre la rigidez metodológica.
- **CRISP-DM** sirvió como guía estructural para el tratamiento de datos: desde la comprensión del problema, la limpieza de archivos, hasta la integración en una base de datos relacional. Fue especialmente útil en el diseño del pipeline ETL, donde cada fase (extracción, transformación y carga) fue abordada con lógica sistemática, aunque no se formalizaron entregables como en un proyecto de minería de datos tradicional.

- **Enfoque pensado en el usuario** priorizando la experiencia de uso como criterio fundamental. Cada herramienta, visualización o flujo conversacional fue testeado empíricamente con usuarios reales (como operadores y analistas), lo que permitió refinar las funcionalidades en función de su retroalimentación. Aunque no se aplicaron metodologías formales de diseño, el proceso se desarrolló de manera iterativa, con ciclos constantes de prueba y mejora, guiado por la necesidad de asegurar utilidad y usabilidad en contextos reales de operación.

En resumen, más que una aplicación ortodoxa de cada marco, el proyecto combinó elementos funcionales y prácticos de los tres enfoques, seleccionando lo más pertinente según la etapa y los desafíos específicos. Esta metodología híbrida, aplicada con flexibilidad, fue clave para avanzar de forma eficiente, adaptable y centrada en la solución real del problema detectado. En la Tabla 3.1 se puede observar la relación entre fase del proyecto y metodología aplicada.

Tabla 3.1: Metodologías Aplicadas por fase del proyecto

Fase del Proyecto	**Scrum (Desarrollo Ágil)**	**CRISP-DM (Minería de Datos)**	**Enfoque Centrado en el Usuario**
Diagnóstico y comprensión del problema	Entrevistas breves iterativas para entender necesidades reales	Comprensión del negocio y de los datos disponibles	Empatía con usuarios reales (operadores, planificadores)
Preparación y tratamiento de datos	Tareas planificadas en ciclos cortos (automatización, pruebas de carga)	Limpieza, transformación, integración de datos (pipeline ETL)	Validación de legibilidad y claridad de datos desde el usuario
Desarrollo del sistema (backend, IA)	Iteraciones constantes del agente SQL, integración de funciones una por una	Modelado estructurado para consultas SQL eficientes	Testing funcional con preguntas reales de usuarios
Visualización y herramientas gráficas	Despliegue incremental del microservicio de gráficos (Streamlit)	Generación de datos visualizables desde múltiples fuentes	Validación visual de comprensión: ¿el usuario entiende el gráfico?
Diseño del frontend conversacional	Releases progresivos de funcionalidades (login, historial, renderizado)	No aplica directamente en esta fase	Priorización de claridad, accesibilidad y experiencia fluida en la interfaz
Validación y mejora continua	Feedback frecuente → ajustes → nueva iteración	Evaluación funcional de calidad de datos y resultados	Iteración con feedback directo de los usuarios en terreno

Fuente: Elaboración Propia

3.2 Fases de Desarrollo

Diagnóstico del Problema

En la primera etapa del proyecto se realizó un diagnóstico de los flujos operativos actuales en unidades forestales, detectando como principal problema la fragmentación de la información entre

múltiples archivos Excel con formatos dispares, estructuras inconsistentes y ausencia de conexiones relacionales entre datos clave. Esta situación generaba lentitud en las búsquedas, dificultades en el cruce de información y limitaciones para realizar análisis históricos o comparativos.

Selección de Tecnologías y Herramientas

Para la implementación concreta del sistema, se realizó una evaluación técnica de distintas herramientas de automatización, bases de datos y plataformas de despliegue, priorizando criterios de escalabilidad, flexibilidad y compatibilidad con la lógica del agente conversacional. Los detalles específicos de esta evaluación comparativa se desarrollan en profundidad en el capítulo de Desarrollo.

Desarrollo del Pipeline de Transformación de Datos

Con base en el enfoque de CRISP-DM y siguiendo los principios de la metodología ETL, se desarrolló un pipeline de transformación de datos en Python, utilizando librerías como pandas, numpy y pyarrow.

Fase de Extracción (Extract)

La fase de extracción implicó la lectura sistemática de múltiples archivos Excel dispersos con formatos heterogéneos, implementando rutinas automatizadas para la identificación y carga de datos desde diferentes fuentes. Se desarrollaron funciones específicas para el manejo de distintos formatos de archivo y la detección automática de estructuras de datos.

Fase de Transformación (Transform)

Esta fase constituyó el núcleo del pipeline, donde se aplicaron las reglas de negocio específicas del dominio forestal. Las transformaciones incluyeron: homogeneización de archivos recibidos, renombrado coherente de columnas, corrección de formatos inconsistentes en fechas y unidades, estandarización de tipos de datos, aplicación de reglas específicas para el tratamiento de valores nulos, y validación de integridad referencial entre tablas relacionadas.

Fase de Carga (Load)

La fase de carga se enfocó en la inserción eficiente de los datos transformados hacia la base de datos PostgreSQL en Supabase, implementando estrategias de carga incremental y manejo de transacciones para garantizar la consistencia de los datos.

A lo largo de todo el proceso ETL, se incorporó un sistema de validaciones automáticas y registro de transformaciones (logging), garantizando trazabilidad y transparencia en todas las etapas del proceso

de depuración. Este enfoque metodológico aseguró la calidad, confiabilidad y auditabilidad de los datos resultantes.

Modelado de Base de Datos

El modelado de la base de datos constituyó una fase crítica, en la cual se diseñó una estructura relacional clara con definición explícita de claves primarias y foráneas, tablas maestras normalizadas (como especies, zonas y equipos) y vistas especializadas para resolver análisis complejos. Para asegurar la eficiencia del sistema ante consultas intensivas, se aplicaron técnicas de indexación y optimización de consultas SQL.

Evaluación Comparativa de Proveedores de IA

Se realizó una evaluación comparativa entre las principales empresas líderes en inteligencia artificial generativa: OpenAI, Anthropic, Google DeepMind (Gemini), xAI (Grok) y DeepSeek. Para cada proveedor, se probaron distintos modelos de lenguaje en escenarios controlados, midiendo su desempeño en tareas como interpretación de preguntas en lenguaje natural, generación precisa de consultas SQL multitabla, manejo de filtros y agregaciones, y adaptación a esquemas de base de datos relacionales. El criterio principal de evaluación fue la calidad y funcionalidad de las respuestas, considerando exactitud sintáctica, relevancia semántica y autonomía del modelo.

Desarrollo del Agente Conversacional

El agente conversacional fue configurado mediante un prompt estructurado que define con precisión cómo debe analizar las preguntas del usuario, consultar el esquema de la base (get_schema), ejecutar queries (run_query) y generar visualizaciones (gen_chart). Se optó por un agente único, robusto y autónomo, capaz de interpretar consultas complejas mediante combinaciones de JOINs, subconsultas, filtros y agregaciones sin necesidad de intervención humana.

Desarrollo de Herramientas de Visualización

Se desarrolló una herramienta de gráficos embebibles mediante Streamlit, funcionando como microservicio desacoplado. Al detectar que una consulta requiere un gráfico, el agente construye un conjunto de parámetros estructurados (etiquetas, valores, tipo de gráfico y título), los envía al generador visual, y recibe como respuesta una URL pública con el gráfico generado. Esta URL es integrada automáticamente en la conversación.

Arquitectura y Despliegue

La arquitectura del sistema fue desplegada en Railway, una plataforma que permite gestionar contenedores de forma escalable. Se levantaron servicios independientes para el backend conversacional, la herramienta de gráficos y el frontend web. El control de versiones y la automatización del flujo de despliegue se gestionaron mediante GitHub, configurando integración continua a través de GitHub Actions.

Desarrollo del Frontend

Se desarrolló un frontend web conversacional utilizando React y una estructura modular en JavaScript. Esta interfaz, conectada directamente con Supabase, permite a los usuarios autenticarse mediante login, enviar consultas en lenguaje natural, revisar su historial de preguntas y guardar respuestas relevantes. La autenticación se realiza de forma segura mediante el sistema de tokens persistentes y gestión de sesiones provisto por Supabase.

Validación y Evaluación

El sistema fue validado en terreno con casos de uso reales, incluyendo operadores de terreno, analistas y personal de planificación forestal. Se evaluó el desempeño del agente ante distintos tipos de preguntas, su capacidad para manejar ambigüedades del lenguaje, los tiempos de respuesta bajo distintas cargas, y la precisión de los resultados generados. Los resultados indicaron una tasa de éxito superior al 90% en consultas correctas, tiempos promedio de respuesta inferiores a los tres segundos y una experiencia de uso valorada positivamente por los usuarios.

Documentación y Replicabilidad

El proceso completo fue documentado en su totalidad, incluyendo flujos de datos, diseño del prompt, configuración de herramientas y mecanismos de mantenimiento, lo cual asegura la replicabilidad y escalabilidad del sistema a futuro. La documentación técnica incluye las guías oficiales de Supabase y Streamlit (2024) como referencias de implementación.

4. Desarrollo

4.1 Diagnóstico

La etapa inicial del desarrollo se centró en comprender los procesos operativos asociados a la gestión de información en unidades forestales. Esta fase diagnóstica ayudó a para identificar los principales cuellos de botella, definir los requerimientos del sistema y orientar la posterior selección de tecnologías.

A partir de un análisis de campo, entrevistas con personal técnico y revisión documental, se constató una fragmentación de la información operacional, la cual se encuentra dispersa en múltiples archivos Excel. Estos archivos presentaban una gran heterogeneidad en su estructura: diferencias en nombres de columnas, formatos de fechas, unidades de medida y criterios de codificación, lo que imposibilitaba su integración directa. Además, no existía una relación referencial explícita entre tablas con información clave como especies, zonas, maquinarias o turnos, lo que dificultaba enormemente el cruce de datos y la generación de reportes consistentes.

Actualmente, la empresa utiliza la plataforma Qlik Sense como principal herramienta de análisis de datos, mediante la cual se generan dashboards visuales para la toma de decisiones operativas y estratégicas. No obstante, este enfoque presenta limitaciones importantes: la creación de visualizaciones, la subida de datos y los permisos para que los distintos usuarios tengan acceso a la información necesaria requiere intervención técnica especializada. Además, Qlik Sense no permite realizar consultas en lenguaje natural ni adaptar dinámicamente los reportes según las preguntas específicas del usuario, lo que restringe la flexibilidad y la autonomía en el análisis de datos.

Esta situación provocaba consecuencias significativas en la operación diaria. Por un lado, los tiempos necesarios para realizar consultas o reportes eran elevados, ya que cada búsqueda implicaba revisar manualmente múltiples fuentes de datos. Por otro lado, las inconsistencias entre archivos aumentaban el riesgo de errores en el análisis, afectando la confiabilidad de los informes técnicos y de planificación. Finalmente, la ausencia de una base de datos relacional centralizada limitaba severamente la posibilidad de realizar análisis históricos, comparativos entre múltiples tablas que apoyaran la toma de decisiones estratégicas en poco tiempo o sin la necesidad de tener que pedir permisos a otras áreas de la empresa.

En este contexto, se definió como objetivo general del desarrollo el diseño e implementación de una solución conversacional asistida por inteligencia artificial, capaz de consultar datos históricos en lenguaje natural, automatizar flujos de integración y facilitar la visualización interactiva de la

información. Esta solución debía responder no solo a los requerimientos técnicos de integración de datos, sino también a criterios de usabilidad, adaptándose a usuarios con distintos niveles de experiencia digital.

4.2 Selección de Tecnologías y Herramientas

Con el diagnóstico ya establecido, se procedió a seleccionar las tecnologías más adecuadas para abordar las problemáticas detectadas. Esta elección se realizó considerando tanto los requerimientos funcionales del sistema como criterios de escalabilidad, flexibilidad y facilidad de integración con otros componentes del ecosistema tecnológico existente.

4.2.1 Evaluación de motores de automatización

Uno de los desafíos clave fue diseñar un sistema capaz de integrar datos desde múltiples fuentes, aplicar transformaciones automáticas y disponibilizar la información de manera estructurada para su consulta. Para ello, se evaluaron distintos motores de automatización de flujos de trabajo, entre ellos n8n y Make (antes Integromat).

Ambas plataformas permitían construir flujos visuales, conectarse con APIs externas y ejecutar lógica condicional. Sin embargo, n8n fue seleccionada como la herramienta principal debido a su naturaleza de código abierto, su capacidad de despliegue en servidores propios y su flexibilidad para ser extendida mediante funciones personalizadas en JavaScript. Además, al no depender de planes comerciales limitantes, n8n ofrecía mayor control y autonomía sobre la infraestructura de automatización del sistema.

4.2.2 Selección de base de datos

Otro aspecto crítico fue la elección de una base de datos relacional escalable, segura y fácilmente integrable con otros servicios. Se compararon dos opciones principales: Supabase, basado en PostgreSQL, y Firebase, una base de datos NoSQL orientada a documentos.

Finalmente, se optó por Supabase debido a su enfoque moderno, su compatibilidad con SQL tradicional y su sólida arquitectura relacional. Entre sus principales ventajas destacan su API RESTful completa, la posibilidad de definir políticas de acceso (Row Level Security) que permiten controlar con precisión qué aplicaciones pueden interactuar con la base de datos, y una gran compatibilidad con herramientas modernas utilizadas en el proyecto. Estas características permitieron una integración fluida con el agente conversacional, los microservicios desarrollados y el frontend web.

4.2.3 Plataforma de despliegue

Para el despliegue del sistema se evaluaron distintas plataformas que ofrecieran orquestación automática, escalabilidad, monitoreo y facilidad de integración con repositorios de código. Se compararon Railway, Render y Contabo.

La elección final recayó en Railway, una plataforma que ofrecía la mejor relación potencia-precio en su plan inicial (8 vCPUs y 8 GB de RAM), junto con una actualización en tiempo real desde GitHub, y una alta compatibilidad con el resto del stack tecnológico utilizado. Su interfaz amigable, facilidad de despliegue y capacidad para levantar múltiples servicios de forma independiente la convirtieron en la opción más eficiente para este proyecto.

4.2.4 Control de versiones e integración continua

Para gestionar el código fuente y mantener un control riguroso sobre las actualizaciones del sistema, se utilizó GitHub como repositorio central. GitHub permitió almacenar y organizar todos los archivos necesarios para el despliegue de herramientas como el frontend conversacional, el backend del agente y los microservicios de visualización.

Además de su robustez como sistema de control de versiones, GitHub ofrece una amplia compatibilidad con múltiples lenguajes como JavaScript y Python, los principales usados en este proyecto. Su integración directa con Supabase y Railway permitió configurar flujos de trabajo automatizados mediante GitHub Actions, mejorando significativamente la eficiencia del desarrollo y la confiabilidad del proceso de despliegue continuo.

En la figura 4.1, se presenta una vista general de la arquitectura tecnológica utilizada:

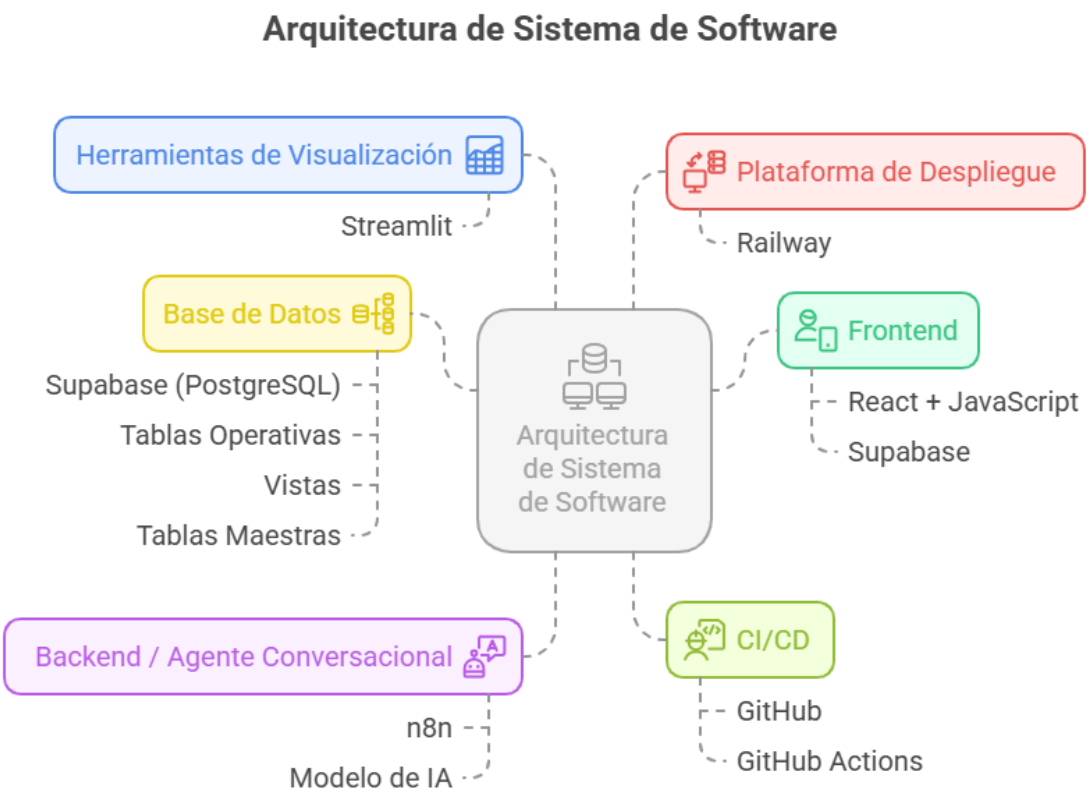


Figura 4.1: Diagrama Explicativo de la Arquitectura del Proyecto

Fuente: Elaboración Propia

4.3 Pipeline ETL y Modelado para Agente Conversacional Inteligente

4.3.1 Visión general del proceso ETL

Una de las piezas centrales del desarrollo del sistema fue la construcción de un pipeline de procesamiento y modelado de datos, diseñado para transformar fuentes dispersas, heterogéneas y ruidosas en una base estructurada, coherente y optimizada para consultas inteligentes.

Este componente no solo resuelve un problema operacional evidente —la fragmentación de la información forestal—, sino que constituye el núcleo informacional sobre el cual opera el agente conversacional SQL. Si los datos no están limpios, estandarizados y correctamente modelados, el agente no puede interpretar preguntas, generar consultas precisas ni entregar respuestas confiables.

Por ello, se diseñó un proceso ETL robusto, automatizado y trazable, capaz de convertir archivos Excel y CSV crudos en tablas relacionales eficientes, aplicando transformaciones semánticas, reglas de negocio del dominio forestal, controles de calidad y validaciones referenciales.

Todo el proceso de extracción, transformación y carga fue implementado mediante scripts en Python, utilizando librerías especializadas como pandas, numpy y pyarrow. Cada una de las tablas fue procesada a través de funciones específicas que automatizan tareas como la normalización de nombres, la eliminación de columnas innecesarias, la estandarización de formatos numéricos y de fechas, y el relleno controlado de valores faltantes. Este enfoque programático no solo permitió escalar el tratamiento de grandes volúmenes de datos forestales, sino que también garantizó trazabilidad y reproducibilidad técnica en cada fase del pipeline.

Dado que el agente opera en un contexto sensible a tokens y precisión estructural, el diseño del pipeline priorizó la reducción de dimensionalidad, la consistencia de formatos y la semántica explícita en los nombres de columna. El resultado es un esquema de datos optimizado, que habilita consultas más rápidas, JOINs precisos y respuestas más económicas en términos computacionales.

En la figura 4.2, se describen las fases clave del proceso ETL y su aplicación sobre cada una de las tablas del sistema, detallando su propósito funcional, transformaciones realizadas y su impacto sobre la capacidad de razonamiento del agente conversacional.

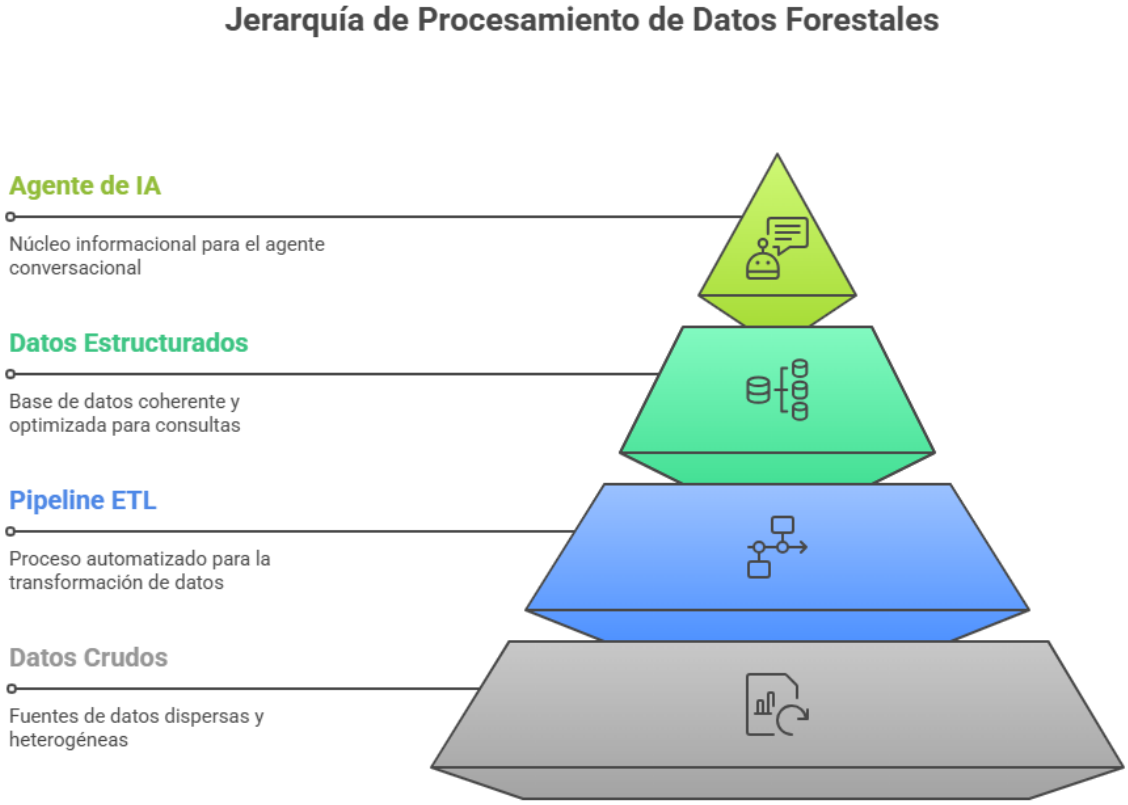


Figura 4.2: Jerarquía de procesamiento de datos forestales

Fuente: Elaboración Propia

4.3.2 Tablas transformadas: Estructura, lógica e impacto

En esta sección se presentan las transformaciones específicas aplicadas a cada una de las tablas operativas principales, es decir, aquellas que contienen datos históricos extraídos directamente de archivos Excel, CSV o informes externos. Estas tablas fueron tratadas como prioridad debido a su rol como insumo base para el funcionamiento del agente conversacional.

Para cada caso, se documenta su propósito funcional, las reglas de limpieza y estandarización implementadas mediante funciones Python, las columnas clave resultantes y el impacto que cada estructura tiene sobre el rendimiento y precisión del agente IA.

Tabla: "produccion" (antes EncabezadoNoc)

Propósito funcional:

Registro de rollizos producidos por zona, por equipo y por largo, indicando predio de origen y otras informaciones.

Transformaciones principales:

Tabla 4.1: Transformaciones aplicadas a la tabla Producción.

Paso	Objetivo	Ejemplo / detalle
Eliminación de 44 campos irrelevantes	Suprimir metadatos administrativos que no aportan a las consultas de negocio.	rut_empresario, id_unidad, destino, etc. son descartados.
Normalización de nombres	Unificar estilo snake_case y acortar identificadores para que los JOINs del agente sean más legibles.	fecha_recepcion → fecha, volumen_m3_ssc → volumen.
Conversión de formatos	– Fechas a ISO-8601. – Números europeos "1.234,56" a notación estándar 1234.56.	Función limpiar_valor() detecta y transforma automáticamente.
Re-etiquetado semántico	Hacer que los nombres expliquen el negocio.	codigo_ext_origen → codigo_predio_origen, indicando claramente que es el predio.
Estandarización de zonas	Reducir variantes textuales a cuatro valores canónicos para filtrado rápido.	"FASA Zona Arauco" → arauco.
Logging y trazabilidad	Registrar cada transformación para auditoría posterior.	Se escribe un log por fila procesada antes de la carga.

Fuente: Elaboración Propia.

Columnas finales clave:

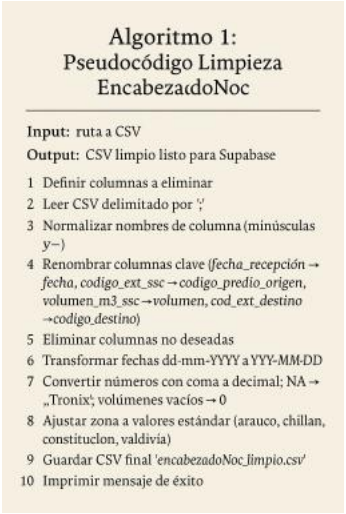
Tabla 4.2: Principales Columnas Finales de la tabla Producción

Columna	Tipo	Descripción
fecha	DATE	Fecha de recepción
codigo_predio_origen	VARCHAR	Predio donde se origina la madera
volumen	NUMERIC(10,2)	Volumen neto recepcionado
zona	VARCHAR	Zona operativa estándar
codigo_producto	VARCHAR	Producto forestal normalizado

Fuente: Elaboración Propia.

Pseudocódigo simplificado:

En la figura 4.3 se presenta el pseudocódigo para el tratamiento de la tabla "producción".



Algoritmo 1:
Pseudocódigo Limpieza EncabezadoNoc

Input: ruta a CSV
Output: CSV limpio listo para Supabase

1 Definir columnas a eliminar
2 Leer CSV delimitado por ';'
3 Normalizar nombres de columna (minúsculas y _)
4 Renombrar columnas clave (*fecha_recepción → fecha, codigo_ext_ssc → codigo_predio_origen, volumen_m3_ssc → volumen, cod_ext_destino → codigo_destino*)
5 Eliminar columnas no deseadas
6 Transformar fechas dd-mm-YYYY a YYY-MM-DD
7 Convertir números con coma a decimal; NA → „Tronix'; volúmenes vacíos → 0
8 Ajustar zona a valores estándar (arauco, chillan, constitucion, valdivia)
9 Guardar CSV final *'encabezadoNoc_limpio.csv'*
10 Imprimir mensaje de éxito

Figura 4.3: PseudoCódigo tratamiento tabla producción.

Fuente: Elaboración Propia

Tabla: "despachos" (antes Valorizado)

Propósito funcional:

La tabla despachos consolida los movimientos de madera transportada desde los predios hacia los distintos destinos posibles (aserraderos, plantas o canchas).

Transformaciones principales:

Tabla 4.3: Transformaciones aplicadas a la tabla Despachos.

Paso	Objetivo	Ejemplo / detalle
Normalización de columnas	Estilo snake_case, sin tildes ni espacios.	fecha natural → fecha_natural.
Eliminación masiva de columnas irrelevantes	Se eliminaron más de 30 campos sin utilidad para consultas conversacionales.	Ej: zona_destino, fuente, fecha_sale_origen, key_planta.
Renombrado semántico	Para facilitar la interpretación automática.	division_txt → zona, producto_forestruck_original → codigo_producto.
Conversión de fecha	De formato yyyymmdd a YYYY-MM-DD.	20250528 → 2025-05-28.
Relleno y sanitización de predio	Reemplazo de valores vacíos por 0 y tipo texto para compatibilidad.	codigo_predio_origen.
Conversión de volumen	De formato latino (1.234,56) a flotante estándar.	Usando función convertir_volumen().
Relleno general	Todos los valores nulos se completaron con "Tronix" para trazabilidad visual.	Evita errores de carga o filtros nulos.

Fuente: Elaboración Propia.

Columnas finales clave:

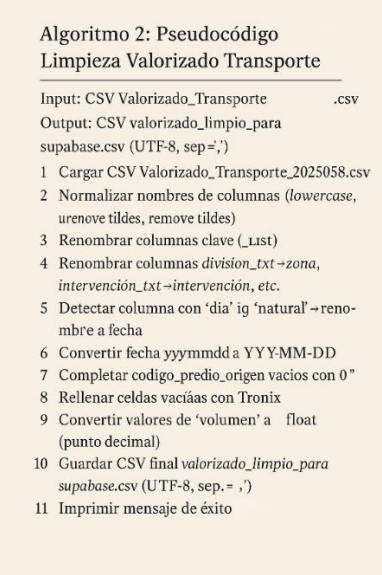
Tabla 4.4: Principales Columnas Finales de la tabla Despachos.

Columna	Tipo	Descripción
fecha	DATE	Fecha del despacho
codigo_predio_origen	VARCHAR	Código SIP del predio de origen
volumen	NUMERIC(10,2)	Volumen despachado
codigo_producto	VARCHAR	Producto forestal normalizado
zona	VARCHAR	Zona de origen

Fuente: Elaboración Propia.

Pseudocódigo simplificado:

En la figura 4.4 se presenta el pseudocódigo para el tratamiento de la tabla "despachos".



Algoritmo 2: Pseudocódigo
Limpieza Valorizado Transporte

Input: CSV Valorizado_Transporte .csv
Output: CSV valorizado_limpio_para supabase.csv (UTF-8, sep=',')

1 Cargar CSV Valorizado_Transporte_2025058.csv
2 Normalizar nombres de columnas (*lowercase*, urenove tildes, remove tildes)
3 Renombrar columnas clave (_ʟɪsᴛ)
4 Renombrar columnas *division_txt→zona*, *intervención_txt→intervención*, *etc.*
5 Detectar columna con 'dia' ig 'natural'→reno-mbre a fecha
6 Convertir fecha yyymmdd a YYY-MM-DD
7 Completar codigo_predio_origen vacios con 0"
8 Rellenar celdas vacíaas con Tronix
9 Convertir valores de 'volumen' a float (punto decimal)
10 Guardar CSV final *valorizado_limpio_para supabase.csv* (UTF-8, sep. = ,')
11 Imprimir mensaje de éxito

Figura 4.4: PseudoCódigo tratamiento tabla Despachos.

Fuente: Elaboración Propia

Tabla: "stock_canchas" (antes SO_CANCHA)

Propósito funcional:

Refleja el inventario físico de madera en patios/canchas antes del procesamiento en planta.

Transformaciones principales:

Tabla 4.5: Transformaciones aplicadas a la tabla Stock en Canchas.

Paso	Objetivo	Detalle
Normalizar nombres	snake_case, sin tildes.	Ej.: Fecha_Calculo → fecha_calculo.
Renombrar claves	Claridad semántica.	cod_sap → codigo_origen, prd_ft_or → codigo_producto.
Eliminación selectiva (≈ 30 campos)	Suprimir meta-datos de cálculo y planificación que no afectan consultas conversacionales.	Ej.: paisaje, prioridad, tipo_faena.
Conversión de fecha	DD-MM-YYYY → YYYY-MM-DD.	Función transformar_fecha().
Estandarización numérica	Cambiar comas decimales por puntos sólo en campos volumen.	1 234,56 → 1234.56.
Relleno controlado	Vacíos → 0 (volumen) o "Tronix" (texto).	Evita nulos en Supabase.

Fuente: Elaboración Propia.

Columnas finales clave:

Tabla 4.6: Principales Columnas Finales de la tabla Stock en Canchas.

Columna	Tipo	Descripción
fecha	DATE	Fecha de corte del inventario.
codigo_origen	VARCHAR	Código SAP de la cancha.
codigo_producto	VARCHAR	Producto forestal normalizado.
volumen_m3	NUMERIC(10,2)	Stock disponible.
zona	VARCHAR	Zona operativa.

Fuente: Elaboración Propia.

Pseudocódigo simplificado incluido en la sección Anexos

Tabla: “stock_predios” (antes SO_BOSQUE)

Propósito funcional:

Consolida el inventario de madera en pie o apilada en predios antes del despacho.

Transformaciones principales:

Tabla 4.7: Transformaciones aplicadas a la tabla Stock en Predios

Paso	Objetivo	Detalle
Normalizar y renombrar	Homogeneizar nombres de columnas para mantener consistencia semántica.	cod_sip → codigo_predio para referirse al origen geográfico; prd_ft_or → codigo_producto para facilitar los JOINs.
Eliminación extensiva (≈ 35 campos)	Reducir dimensionalidad y eliminar campos sin valor analítico.	Se eliminaron columnas como planificada, clase_material, calidad_inicial, especie, material y otros campos administrativos.
Estándar de fechas y volúmenes	Uniformar formatos para compatibilidad con el motor SQL.	Se aplicaron las funciones transformar_fecha() y transformar_numero() para convertir a ISO 8601 y notación decimal estándar.
Formateo de volúmenes	Asegurar precisión y coherencia numérica en cálculos posteriores.	Todos los campos de volumen fueron redondeados a dos decimales (float) con formato fijo tipo 1234.50.
Relleno de vacíos	Evitar errores de carga y mantener trazabilidad de datos faltantes.	Volúmenes vacíos se completaron con 0.00; textos vacíos con "Tronix" como marcador de limpieza controlada.

Fuente: Elaboración Propia

Columnas finales clave:

Tabla 4.8: Principales Columnas Finales de la tabla Stock en Predios

Columna	Tipo	Descripción
fecha	DATE	Fecha del inventario.
codigo_predio	VARCHAR	Identificador del predio.
codigo_producto	VARCHAR	Producto forestal.
volumen_m3	NUMERIC(10,2)	Volumen disponible.
zona	VARCHAR	Zona de origen.

Fuente: Elaboración Propia

Pseudocódigo simplificado incluido en la sección Anexos

Tabla: “terceros” (antes BD_Terceros)

Propósito funcional:

Documenta los movimientos de madera de proveedores externos con trazabilidad y modalidad de adquisición.

Tabla 4.9: Transformaciones aplicadas a la tabla Terceros

Paso	**Objetivo**	**Ejemplo / detalle**
Normalización de nombres	Columnas en minúsculas y sin espacios	fecha_terceros → fecha, zona_terceros → zona
Renombrado semántico	Clarificar propósito de cada campo	m3ssc recepción → volumen, predio_origen_terceros → fundo_/_origen_madera
Eliminación masiva de columnas	Suprimir ~20 campos administrativos irrelevantes	periodo_terceros, dia_terceros, vlaaaaaa, factor_costo
Conversión de fechas	Formato DD-MM-YYYY a ISO 8601	15-06-2025 → 2025-06-15
Filtrado temporal	Eliminar registros del año 2024	Exclusión automática de fechas 2024
Conversión numérica	Formato europeo a estándar en volúmenes	1.234,56 → 1234.56
Mapeo de modalidades	Códigos SAP a descripciones claras	ZPLA → PPP, ZCCH → Cancha, ZORC → OC
Limpieza de calidad	Fechas erróneas convertidas a estándar	Fechas detectadas → PLLF
Filtrado de registros	Eliminar filas sin aplicabilidad	Exclusión donde aplica_terceros esté vacío o sea "Tronix"
Relleno controlado	Valores faltantes con defaults	Volúmenes → 0, largo → astilla, resto → Tronix

Fuente: Elaboración Propia

Columnas finales clave:

Tabla 4.10: Principales Columnas Finales de la tabla Terceros

Columna	**Tipo**	**Descripción**
fecha	DATE	Fecha de recepción.
zona	VARCHAR	Zona operativa.
volumen	NUMERIC(10,2)	Volumen recepcionado.
modalidad_terceros	VARCHAR	Modalidad de compra.
fundo_/_origen_madera	VARCHAR	Origen de la madera.

Fuente: Elaboración Propia

Pseudocódigo simplificado incluido en la sección Anexos

Tabla: “giros_programados” (antes Informe_Detalle_Giros)

Propósito funcional:

Detalle de fletes programados con origen, destino, volumen y operador, clave para trazabilidad logística.

Transformaciones principales:

Tabla 4.11: Transformaciones aplicadas a la tabla Giros Programados

Paso	Objetivo	Ejemplo / detalle
Normalización de headers	Minúsculas, sin espacios, sin caracteres especiales	Código Origen → codigo_origen, eliminación de \ufeff
Renombrado semántico específico	Clarificar campos clave del negocio	producto_forestruck_original → codigo_producto, fecha_programa → fecha
Eliminación de sufijos redundantes	Quitar terminaciones _original sistemáticamente	camion_original → camion, categoria_original → categoria
Eliminación masiva de columnas	Suprimir ~40 campos de control y metadatos	orden, giro_asicam, tiempo_carguio, km_pavimento, coordenadas_guia
Filtrado de campos temporales	Eliminar columnas con sufijo _final y relacionadas a tiempo	codigo_camion_final, llegada_carguio, inicia_carguio
Conversión de fechas	Formato DD-MM-YYYY a ISO 8601	28-05-2025 → 2025-05-28
Conversión numérica	Formato europeo a estándar decimal	1.234,56 → 1234.56, 306,185 → 306.185
Estandarización de volumen	Valor fijo para homogeneizar registros	Todos los registros → 28 m³
Mapeo de códigos origen	Renombrado para consistencia con otras tablas	codigo_origen_original → codigo_sip_origen
Relleno controlado	Valores faltantes con identificador estándar	Campos vacíos → Tronix

Fuente: Elaboración Propia

Columnas finales clave:

Tabla 4.12: Principales Columnas Finales de la tabla Giros Programados

Columna	Tipo	Descripción
fecha	DATE	Fecha de programación.
codigo_sip_origen	VARCHAR	Código de origen.
codigo_producto	VARCHAR	Producto transportado.
camion	VARCHAR	Camión asignado.
volumen	NUMERIC(10,2)	Volumen estandarizado.

Fuente: Elaboración Propia

Pseudocódigo simplificado incluido en la sección Anexos

Tabla: gruas (antes Gruas)

Propósito funcional:

La tabla grúas consolida la información operativa de las grúas forestales, incluyendo productividad, ubicación, turnos y volúmenes procesados.

Transformaciones principales:

Tabla 4.13: Transformaciones aplicadas a la tabla Grúas

Paso	Objetivo	Ejemplo / detalle
Detección automática de delimitador	Identificar separador correcto del CSV	Auto-detección entre ; y , usando csv.Sniffer
Normalización de headers	Minúsculas, sin espacios, sin caracteres especiales	Día Natural → fecha, eliminación de \ufeff
Renombrado semántico específico	Clarificar campos clave del negocio	día_natural → fecha, volumen_m3 → volumen
Eliminación selectiva de columnas	Suprimir 6 campos de control y metadatos	tipo, dia_mes, key_turno, no_considerar, aplica_descarguio, periodo
Conversión de fechas	Formato YYYYMMDD a ISO 8601	20250201 → 2025-02-01
Conversión numérica	Formato europeo a estándar decimal	1.234,56 → 1234.56, 123,45 → 123.45
Manejo de encoding	Soporte para caracteres especiales	Codificación utf-8-sig para compatibilidad
Relleno controlado	Valores faltantes con identificador estándar	Campos vacíos → Tronix
Preservación de índices	Mantener orden de columnas relevantes	Mapeo de índices para campos a conservar
Validación de formato	Verificación de patrones numéricos y fecha	Regex para detectar formatos específicos

Fuente: Elaboración Propia

Columnas finales clave:

Tabla 4.14: Principales Columnas Finales de la tabla Grúas

Columna	Tipo	Descripción
fecha	DATE	Fecha de operación.
codigo_grua	VARCHAR	Identificador de la grúa.
volumen	NUMERIC(10,2)	Volumen procesado.
zona	VARCHAR	Zona operativa.

Fuente: Elaboración Propia

Pseudocódigo simplificado incluido en la sección Anexos

4.3.3 Impacto global en el agente conversacional IA

La transformación de los datos mediante el pipeline ETL no solo tuvo efectos sobre la organización y limpieza de las tablas, sino que constituyó la base operativa sobre la cual se sustenta el razonamiento del agente conversacional SQL. Al estructurar de forma rigurosa los datos forestales —eliminando redundancias, normalizando columnas y estandarizando claves semánticas— se logró optimizar tanto el rendimiento del sistema como la calidad de las respuestas generadas por el modelo de lenguaje.

Los beneficios de este trabajo de depuración se traducen directamente en cinco impactos clave sobre la operación del agente:

- **Reducción de tokens procesados:** Al eliminar más del 70 % de columnas irrelevantes, las consultas generadas por el modelo son más compactas, eficientes y menos costosas en términos computacionales.
- **JOINs más simples y precisos**: La estandarización de columnas clave como fecha, codigo_predio_origen, volumen y zona permite relaciones entre tablas claras, sin ambigüedades ni necesidad de lógica adicional.
- **Mayor precisión del modelo:** Al operar sobre datos limpios, tipados correctamente y sin valores nulos inesperados, se reduce drásticamente el riesgo de errores, inconsistencias o alucinaciones al momento de generar SQL.
- **Semántica clara para el modelo:** El uso de nombres explícitos como volumen_proyectado, estado_madera o modalidad_terceros permite al modelo comprender el propósito de cada columna sin requerir ingeniería de prompt adicional o definiciones auxiliares.
- **Consultas más rápidas y relevantes:** La preestructuración de datos y la aplicación de filtros anticipados reducen la carga de procesamiento en tiempo real, permitiendo respuestas ágiles y enfocadas en lo esencial.

4.3.4 Tablas de planificación y proyección – Estructura, lógica e impacto

Una vez transformadas las tablas operativas de base, el siguiente paso del proceso fue abordar las tablas de planificación y proyección, las cuales contienen estimaciones futuras, metas operativas y directrices estratégicas del negocio forestal. A diferencia de las tablas anteriores, estas fuentes no registran eventos pasados ni movimientos concretos de madera, sino que modelan el comportamiento

esperado del sistema en función de reglas de planificación, criterios técnicos o decisiones administrativas.

Estas tablas son fundamentales para habilitar análisis prospectivos, generar comparativas real vs plan y construir proyecciones automatizadas desde el agente. Sin embargo, presentaban una alta variabilidad en sus formatos de origen, con estructuras tipo “tabla dinámica”, múltiples codificaciones internas y ausencia de estándares comunes en fechas, zonas o productos.

El proceso de transformación aplicado aquí tuvo como objetivo alinear semánticamente estas estructuras con el resto del ecosistema de datos, facilitando su integración con las tablas operativas ya normalizadas. Se aplicaron técnicas como el desanidamiento de formatos pivot, la detección de terceros, la estandarización de zonas y productos, y la validación de consistencia temporal.

A continuación, se describen en detalle las transformaciones realizadas sobre cada una de estas tablas, su estructura final y el impacto concreto que tienen sobre la capacidad del agente para razonar, proyectar y responder preguntas estratégicas en lenguaje natural.

Tabla: plan_despachos (antes plan de despachos)

Propósito funcional:

Consolida la planificación de abastecimiento y despacho hacia destinos, permitiendo tener una guía mensual de despachos que hacer hacía las distintas instalaciones, tanto de celulosa como aserraderos, de los distintos productos que necesitan.

Transformaciones principales:

Tabla 4.15: Transformaciones aplicadas a la tabla Plan Despachos

Paso	Objetivo	Ejemplo / detalle
Reestructuración de formato	Convertir de formato ancho (pivot) a formato largo (normalizado) para análisis eficiente.	Columnas de productos se transforman en filas con melt().
Normalización de nombres	Unificar estilo snake_case y renombrar columnas para mayor claridad semántica.	cod_destino → codigo_destino para consistencia.
Filtrado de datos irrelevantes	Eliminar registros que no aportan valor al análisis de negocio.	Productos con "QUEMADO" y destino "C035" son descartados.
Conversión de formatos	Fechas a ISO 8601 y normalización de strings.	pd.to_datetime() con formato '%Y-%m-%d'.
Detección de terceros	Identificar productos de terceros vs producción propia.	Productos con "PPP" → es_tercero = True.

Paso	Objetivo	Ejemplo / detalle
Separación de códigos	Extraer código de producto y zona de operación.	'PRODUCTO-BC01' → codigo_producto='PRODUCTO', zona_operacion='BC01'.
Mapeo de zonas	Estandarizar zonas a valores canónicos para filtrado rápido.	'BC01' → 'ARAUCO', 'FT01' → 'CONSTITUCION'.
Limpieza de códigos	Eliminar descriptores innecesarios de los códigos de producto.	'PRODUCTOMANCHADO' → 'PRODUCTO'.
Eliminación de columnas	Descartar campos intermedios y descriptivos innecesarios.	descripcion_destino, destino, codigo_producto_original.

Fuente: Elaboración Propia

Columnas finales clave:

Tabla 4.16: Principales Columnas Finales de la tabla Plan despachos

Columna	Tipo	Descripción
fecha	DATE	Fecha de planificación del despacho.
codigo_destino	VARCHAR	Centro de destino.
codigo_producto	VARCHAR	Producto planificado.
zona_origen	VARCHAR	Zona operativa estandarizada.
volumen_planificado	NUMERIC(10,2)	Volumen planificado.
es_tercero	BOOLEAN	Indica si es producto de terceros.

Fuente: Elaboración Propia

Pseudocódigo simplificado incluido en la sección Anexos

Tabla: proyecciones_produccion (antes proyecciones de producción)

Propósito funcional:

Registra estimaciones de volumen futuro de producción forestal por predio, especie y fecha proyectada. Es clave para análisis estratégicos y comparación real vs proyectado.

Transformaciones principales:

Tabla 4.17: Transformaciones aplicadas a la tabla de Proyecciones

Paso	Objetivo	Ejemplo / detalle
Normalización de nombres	Unificar estilo snake_case y renombrar columnas para mayor claridad semántica.	volssc → volumen_proyectado, tipo_prod → calidad.
Eliminación masiva de columnas	Descartar ~20 campos administrativos que no aportan valor al análisis de negocio.	m3ssc_proy_100, secuencia, podado, pulpa, sap, etc.
Renombrado semántico	Hacer que los nombres expliquen claramente el propósito de negocio.	cod → codigo_predio_origen, est_madera → estado_madera.
Conversión inteligente de fechas	Manejar múltiples formatos de fecha incluyendo números seriales de Excel.	Números de 5 dígitos → conversión desde 1899-12-30 base.
Transformación de números	Convertir formatos europeos y limpiar separadores de miles.	"1,234.56" → "1234.56", "1234,56" → "1234.56".
Manejo de valores nulos	Establecer valores por defecto según el tipo de columna.	Columnas de volumen → '0', otras → 'Tronix'.
Detección de patrones	Identificar automáticamente formatos numéricos complejos.	Regex para detectar separadores de miles y decimales.

Fuente: Elaboración Propia

Columnas finales clave:

Tabla 4.18: Principales Columnas Finales de la tabla Proyecciones

Columna	Tipo	Descripción
fecha	DATE	Fecha proyectada de cosecha/intervención.
codigo_predio_origen	VARCHAR	Predio forestal.
destino	VARCHAR	Destino proyectado.
volumen_proyectado	NUMERIC(10,2)	Volumen estimado.
calidad	VARCHAR	Clasificación de calidad del producto.
intervencion	VARCHAR	Tipo de intervención forestal.
estado_madera	VARCHAR	Estado de la madera.

Fuente: Elaboración Propia

Pseudocódigo simplificado incluido en la sección Anexos

Tabla: plan_produccion_opticort (antes OptiCort)

Propósito funcional:

Contiene la planificación mensual operativa generada por el sistema OptiCort, incluyendo calidad, fecha y volumen planificado por predio.

Transformaciones principales:

Tabla 4.19: Transformaciones aplicadas a la tabla Plan producción OptiCort

Paso	Objetivo	Ejemplo / detalle
Normalización de nombres	Unificar estilo snake_case y limpiar espacios en blanco.	Conversión automática: "Total SSC" → "total_ssc".
Eliminación de columnas operativas	Descartar campos específicos del sistema OptiCort que no aportan al análisis de negocio.	sap_ems, emsefor, predio, pendi, pertenencia_máquina, tipo_maquina, temporada, opticort.
Renombrado semántico	Hacer que los nombres expliquen claramente el propósito de negocio.	total_ssc → volumen_proyectado, producto → calidad, zona2 → zona.
Filtrado temporal específico	Mantener solo registros del período operativo actual (junio 2025).	Filtro: mes == 6 AND año == 2025.
Conversión de fechas	Normalizar todas las columnas de fecha a formato ISO estándar.	pd.to_datetime() → '%Y-%m-%d'.
Manejo de errores	Implementar control de errores robusto para columnas de fecha.	Try/catch con errors='coerce' para datos malformados.

Fuente: Elaboración Propia

Columnas finales clave:

Tabla 4.20: Principales Columnas Finales de la tabla Plan producción OptiCort

Columna	Tipo	Descripción
fecha	DATE	Fecha planificada.
codigo_predio	VARCHAR	Identificador del predio.
zona	VARCHAR	Zona operativa asignada.
volumen_proyectado	NUMERIC(10,2)	Volumen planificado.
calidad	VARCHAR	Calidad del producto.

Fuente: Elaboración Propia

Pseudocódigo simplificado incluido en la sección Anexos

4.3.5 Impacto global en el agente conversacional IA (planificación y proyecciones)

La transformación de las tablas de planificación y proyección permitió que el agente SQL pudiera razonar de forma prospectiva con el mismo nivel de precisión y estructura que en las consultas históricas. Entre los principales beneficios operativos de esta parte del desarrollo, destacan:

- **Estructura normalizada y formato largo:** El reprocesamiento de tablas con formato pivot a estructuras verticales permitió construir consultas simples y legibles, facilitando los JOINs y mejorando la compatibilidad con el modelo.
- **Separación semántica explícita:** Al distinguir claramente entre planificación propia y terceros, el agente puede ejecutar consultas más específicas sin lógica adicional, lo que se traduce en mayor precisión en las respuestas.
- **Consistencia temporal estandarizada:** La conversión de fechas a formato ISO 8601 habilita agrupamientos, filtros y análisis por períodos de forma robusta, reduciendo errores de interpretación y mejorando la trazabilidad.
- **Uniformidad en zonas y productos:** La estandarización de estos atributos evita ambigüedades en predicciones multitabla, mejora la coherencia entre fuentes y reduce errores sintácticos al construir SQL.
- **Reducción del procesamiento en tiempo real:** Con los datos ya estructurados y limpios, el agente puede generar respuestas estratégicas y proyecciones automatizadas con una sola consulta optimizada, sin necesidad de aplicar transformaciones en el momento.

En conjunto, estas mejoras consolidan la capacidad del agente para responder preguntas complejas relacionadas con metas operativas, estimaciones futuras o cumplimiento de planificación, manteniendo la trazabilidad y eficiencia de todo el sistema conversacional.

4.3.6 Tablas maestras – Estandarización y modularidad

Una vez consolidadas y normalizadas tanto las tablas operativas como las de planificación, se procedió a diseñar un conjunto de tablas maestras que actúan como ejes semánticos del sistema. Estas estructuras no contienen registros transaccionales, sino que representan dimensiones clave como productos, zonas, lugares y orígenes, funcionando como referencias centralizadas para garantizar la coherencia del modelo de datos completo.

El propósito de estas tablas es doble: por un lado, permiten reducir la ambigüedad semántica al traducir códigos técnicos en descripciones legibles; por otro, habilitan una arquitectura modular, escalable y mantenible, donde cualquier cambio en nombres, clasificaciones o atributos se gestiona desde una fuente única sin necesidad de modificar múltiples tablas operativas.

Gracias a esta estandarización, el agente conversacional SQL puede interpretar con mayor precisión preguntas formuladas en lenguaje natural, establecer relaciones complejas entre entidades y entregar

respuestas más claras y relevantes. Además, estas tablas facilitan la construcción de dashboards, vistas dinámicas y filtros transversales, lo que potencia el uso estratégico de la plataforma.

A continuación, se describen las principales tablas maestras utilizadas, junto con sus columnas clave y el impacto funcional que tienen sobre el rendimiento, escalabilidad y precisión del sistema.

Tabla: origenes

Propósito:

Contiene el listado único de todos los puntos de origen de la madera, tanto predios forestales como canchas. Es clave para traducir códigos crudos en nombres legibles y para segmentar análisis por zona.

Columnas clave:

Tabla 4.21: Columnas tabla maestra origenes

Columna	Tipo	Descripción
codigo_origen_sip	VARCHAR	Código único del sistema SIP.
nombre_origen	VARCHAR	Nombre legible del predio o cancha.
zona	VARCHAR	Zona operativa (arauco, constitucion, etc.).

Fuente: Elaboración Propia

Tabla: lugares

Propósito:

Catálogo unificado de ubicaciones operativas: predios, canchas, plantas y patios. Permite que el agente IA utilice una lógica común para distintos tipos de lugar.

Columnas clave:

Tabla 4.22: Columnas tabla maestra lugares

Columna	Tipo	Descripción
codigo	VARCHAR	Identificador único del lugar.
tipo_origen	VARCHAR	Clasificación del lugar (predio, cancha, planta, patio).
nombre	VARCHAR	Nombre legible del lugar.
zona	VARCHAR	Zona operativa donde se encuentra.

Fuente: Elaboración Propia

Tabla: productos

Propósito:

Contiene el detalle técnico de cada producto forestal, su especie, calidad, largo y otros atributos. Es fundamental para análisis logístico, planificación y trazabilidad.

Columnas clave:

Tabla 4.23: Columnas tabla maestra productos

Columna	Tipo	Descripción
codigo_producto	VARCHAR	Código principal del producto.
largo	NUMERIC(5,2)	Largo nominal en metros.
calidad	VARCHAR	Clasificación de calidad.
tiene_largo_variable	BOOLEAN	Indica si el producto puede tener distintos largos.
factor_corteza	NUMERIC(4,2)	Factor de ajuste por corteza.
especie	VARCHAR	Especie forestal.
codigo_producto_abreviado	VARCHAR	Código corto para visualizaciones y prompts.

Fuente: Elaboración Propia

4.3.7 Impacto consolidado de las tablas maestras

Las tablas maestras constituyen el núcleo semántico del sistema, ya que definen explícitamente el significado operativo de conceptos clave como predios, productos y zonas. Al centralizar esta información en estructuras normalizadas, se logra:

- **Interpretación precisa en lenguaje natural**: El agente puede mapear términos legibles a estructuras SQL sin necesidad de lógica adicional.
- **Consultas más simples y eficientes**: Se optimizan los JOINs y se reducen errores al eliminar ambigüedad semántica.
- **Escalabilidad y mantenibilidad**: Cualquier cambio en clasificaciones, nombres o atributos se gestiona desde una única fuente, sin alterar múltiples tablas operativas.
- **Ahorro de tokens y precisión del modelo**: Los códigos abreviados permiten prompts más compactos, manteniendo el significado contextual.
- **Marco ontológico explícito**: Las tablas maestras le otorgan al agente un marco de razonamiento coherente, esencial para interpretar relaciones complejas.

Este diseño estructurado está alineado con los principios modernos de *semantic parsing* y *knowledge-grounded reasoning*, donde la calidad del esquema y las relaciones definidas impactan directamente en la eficiencia, trazabilidad y precisión del modelo conversacional (McTear, 2016; Yu et al., 2018).

4.4 Modelado de base de datos

Tras completar el proceso de transformación y estandarización de las distintas fuentes de datos —tanto operativas como proyectadas—, se procedió a consolidar una estructura lógica robusta que permitiera consultas complejas, integración eficiente entre entidades y un funcionamiento óptimo del agente SQL. Esta etapa de modelado fue clave para traducir el ecosistema de datos ya depurado en una arquitectura relacional clara y optimizada.

Se diseñó una base de datos estructurada en torno a principios de normalización, con definición explícita de claves primarias y foráneas que aseguran la integridad referencial entre las distintas entidades. Este diseño permite que las consultas ejecutadas por el agente mantengan consistencia semántica, precisión sintáctica y escalabilidad técnica ante grandes volúmenes de información.

Como parte esencial del modelado, se desarrollaron vistas SQL especializadas orientadas a encapsular lógicas complejas directamente dentro del motor de base de datos. Estas vistas no solo agilizan el análisis por parte de los usuarios, sino que permiten que el agente realice comparaciones, filtros y cálculos sin tener que construir consultas extensas desde cero. Entre las más relevantes destacan:

- **comparativa_despachos:** Permite contrastar en tiempo real los volúmenes planificados versus los efectivamente despachados, distinguiendo entre terceros y propios, y aplicando filtros por zona, fecha y tipo de producto.
- **comparativa_produccion_teams:** Evalúa el cumplimiento de metas por equipo (team), cruzando proyecciones con registros reales de forma granular y ordenada.
- **plan_despachos_vs_productos:** Verifica el grado de alineación entre planificación y ejecución logística por producto, detectando desviaciones en tiempo y volumen.

Estas vistas se actualizan dinámicamente cada vez que se modifican las tablas base, lo que garantiza que el agente siempre trabaje sobre información actualizada y preprocesada. Esta estrategia permite delegar el "trabajo sucio" —combinaciones, filtrados, agrupaciones y cálculos— directamente al motor SQL, liberando al modelo de lenguaje de tener que procesar esa lógica en tiempo real.

El resultado no solo es un sistema más rápido y robusto**,** sino también más económico**:** al reducir la complejidad de las respuestas generadas, también se reduce la cantidad de tokens necesarios en cada

interacción. Dado que los modelos de lenguaje cobran por volumen de tokens procesados, cada optimización a nivel estructural contribuye a una disminución directa del costo operativo del sistema. Esta eficiencia cobra especial relevancia cuando el sistema se encuentra en producción y es consultado por múltiples usuarios en paralelo, donde cada milisegundo y cada token cuentan.

4.5 Evaluación comparativa de proveedores de IA

Una vez definido el modelo de datos y optimizada la arquitectura del sistema, el siguiente paso fue seleccionar el motor de inteligencia artificial que serviría como núcleo del agente conversacional SQL. Para ello, se realizó una evaluación comparativa rigurosa entre los principales proveedores de modelos de lenguaje a gran escala (LLMs), con el objetivo de identificar aquel con mayor capacidad para interpretar lenguaje natural, razonar sobre estructuras complejas, generar consultas SQL eficientes y entregar respuestas precisas en tiempo real.

La evaluación incluyó a los cinco actores más relevantes del mercado: OpenAI, Anthropic, Google DeepMind (Gemini), xAI (Grok) y DeepSeek, seleccionados por su liderazgo en benchmarks recientes de generación, razonamiento y codificación, así como por la calidad de sus APIs (Helicone, 2025; Shakudo, 2025; Upmarket, 2025).

Se aplicaron tres criterios técnicos clave para la selección:

1. Madurez tecnológica, evaluada según su desempeño en pruebas como Spider y SQLEval (Yu et al., 2018; Wang et al., 2023) y su posición en el estado del arte;
2. Disponibilidad de APIs públicas, indispensable para integrar los modelos en entornos productivos de manera segura;
3. Calidad de documentación y soporte técnico, crucial para asegurar la mantenibilidad y escalabilidad del sistema (Amershi et al., 2019).

Se seleccionaron los modelos más recientes y avanzados de cada proveedor, priorizando aquellos con capacidades de razonamiento multitabla, generación estructurada de SQL y soporte para herramientas externas (function calling). La elección se fundamentó también en rankings como Helicone (2025), que comparan precisión, latencia y costo. Los modelos fueron evaluados en escenarios controlados que replicaban consultas reales del sistema Tronix, con preguntas escalonadas desde tareas simples de conexión hasta desafíos complejos de integración multitabla y generación contextualizada de SQL.

Las preguntas evaluadas incluyeron:

1. “Hola” – Test básico de conexión y coherencia.

2. “¿Cuáles son los productos con mayor volumen actual?” – Requiere interpretación de esquema, filtros y ordenamiento.
3. “¿Qué productos son los que más se están produciendo?” – Implica análisis temporal y tablas de producción.
4. “¿Qué productos están con balance negativo (stock - producción + despachos)?” – Exige integración multitabla y cálculos.
5. “¿Puedes explicar el balance negativo del podado?” – Demanda razonamiento complejo y explicación en lenguaje natural.

El principal criterio de evaluación fue la funcionalidad, es decir, la capacidad del modelo para entregar respuestas correctas, estructuradas y razonadas. También se consideraron la latencia de respuesta y el costo por consulta, claves para la escalabilidad del sistema.

Cabe destacar que esta prueba inicial se realizó sin utilizar prompts estructurados. Esto permitió comparar sus capacidades intrínsecas de razonamiento, comprensión semántica y generación de SQL estructurado de forma objetiva y sin sesgos inducidos por el diseño del sistema.

OpenAI y Anthropic lideraron los resultados. GPT-4 Turbo destacó por su precisión y velocidad, aunque con un costo elevado. Claude 3.5 Haiku se posicionó como la opción más eficiente por su buen desempeño a bajo costo, mientras que Claude 4 Sonnet ofreció respuestas más completas, pero con mayor latencia y precio. La tabla 4.24 resume la comparativa entre los modelos más prometedores.

.Tabla 4.24: Tabla comparativa rendimiento sin prompt de modelos de IA.

Modelo	**Costo Total (USD)**	**Calidad de Respuesta**	**Tiempo de Respuesta**	**Resultado Global**
Claude 3.5 Haiku	$0.08 aprox.	Rápido, económico. Superó todas las preguntas sin errores.	Muy rápido	Modelo balanceado
Claude 4 Sonnet	$0.57	Alta calidad. Respuestas completas y detalladas. Superó todo el test.	Medio	Muy buen desempeño
GPT-4.5 Turbo	$5.00	Respuestas excelentes y muy rápidas. Costos desproporcionados.	Muy rápido	Costo alto
GPT-4.1 Turbo	$3.80 aprox.	Rápido y eficiente. Falló en pregunta 4 al no razonar correctamente.	Rápido	Falló razonamiento
OpenAI o1	$1.52	Respuestas claras, buen lenguaje. Falló en pregunta 4.	Rápido	Razonamiento parcial

Fuente: Elaboración Propia

A partir de estos resultados se diseñó una fase final de evaluación del sistema conversacional, donde se realizó una evaluación práctica con los tres modelos finalistas: Claude 3.5 Haiku, Claude 4 Sonnet

y GPT-4 Turbo, bajo condiciones reales, con acceso a la base de datos y el prompt modular del sistema, desarrollado específicamente para este proyecto. Esta prueba buscó reflejar las condiciones reales de uso del sistema, evaluando la precisión de las consultas SQL generadas, la velocidad de respuesta y la eficiencia en el uso de tokens bajo cargas de trabajo representativas y solicitudes de todo tipo.

El modelo Claude 4 Sonnet demostró ser el más preciso y completo, con un 94% de exactitud en la generación de consultas complejas y multitabla, y tiempos de respuesta cercanos a los 38 segundos promedio en tareas pesadas. Su capacidad para interpretar correctamente la intención del usuario y traducirla a SQL fue superior, especialmente en solicitudes complejas como proyecciones, transferencias o balances.

Por su parte, Claude 3.5 Haiku ofreció una excelente relación costo-rendimiento. Aunque mostró menor precisión que Sonnet (89% de exactitud), resultó más económico y suficientemente rápido para operaciones habituales. En pruebas con cargas livianas o medianas, Haiku respondió en menos de 40 segundos, lo que lo convierte en una opción muy eficiente para consultas frecuentes en entornos de alta demanda o solicitudes simples. Sin embargo en solicitudes pesadas, como proyecciones, mostró tiempos de espera largos llegando hasta 4 minutos en algunas ocasiones.

GPT-4 Turbo, en cambio, mostró buen rendimiento general, pero presentó fallas en consultas con múltiples condiciones (WHERE, JOIN), y una mayor propensión a errores semánticos en esquemas no vistos.

Estas observaciones fueron coherentes con benchmarks públicos como LM Arena (2025) y BirdSQL (2025), donde Claude Sonnet se posiciona como uno de los modelos más robustos para tareas Text-to-SQL, y los modelos Claude destacan por su baja tasa de errores y buena generalización.

La exactitud fue determinada revisando manualmente cada consulta SQL generada, ejecutándola directamente sobre la base de datos y comparando sus resultados con los datos oficiales contenidos en los archivos Excel utilizados por la empresa. Este proceso permitió verificar no solo la validez estructural y semántica de cada consulta, sino también la fidelidad de las respuestas respecto a la información operativa real.

Finalmente, y considerando los costos operativos del sistema, se optó por Claude 3.5 Haiku como modelo principal de producción, al ofrecer un equilibrio entre velocidad, precisión y eficiencia de recursos. Sin embargo, Claude 4 Sonnet no fue descartado: su uso queda reservado para escenarios más exigentes o futuros upgrades del sistema, donde se requiera mayor capacidad de razonamiento contextual o menor latencia en cargas pesadas.

Tabla 4.25: Tabla comparativa rendimiento modelos finalistas de IA.

Modelo	**Exactitud SQL (%)**	**Tiempo promedio de respuesta**	**Observaciones relevantes**
Claude 4 Sonnet	94%	38.4 segundos	Alta precisión, excelente en queries multitabla; balance ideal para producción
Claude 3.5 Haiku	89%	2 minutos 17 segundos	Menor capacidad general en comparación a Sonnet; rápido en consultas simples, lento en tablas grandes.
GPT-4 Turbo (OpenAI)	82%	47.2 segundos	Buen rendimiento general, pero más propenso a errores en WHERE o JOIN

Fuente: Elaboración Propia

4.6 Desarrollo del Agente Conversacional

Luego de seleccionar el modelo de lenguaje más adecuado según criterios de precisión, costo y velocidad (Claude 3.5 Haiku), se avanzó en el desarrollo del agente conversacional, encargado de interpretar preguntas en lenguaje natural y traducirlas en consultas estructuradas sobre la base de datos forestal.

El diseño del agente se centró en asegurar un funcionamiento coherente, trazable y eficiente, mediante el uso de herramientas predefinidas y un conjunto de reglas que regulan su comportamiento en cada interacción. La arquitectura se apoya en un prompt estructurado, capaz de guiar al modelo en tareas como la detección de intención, la generación de SQL y la creación de visualizaciones cuando el caso lo requiere.

A continuación, se presenta la estructura lógica del sistema, su conexión con Supabase y los mecanismos implementados para mantener la precisión, consistencia y transparencia en el razonamiento del modelo.

4.6.1 Arquitectura General

El funcionamiento del agente sigue un flujo lógico definido por los siguientes pasos:

- Recepción de la consulta en lenguaje natural.
- Análisis de intención y detección de entidades relevantes (fechas, zonas, especies, métricas).
- Consulta opcional del esquema de base de datos mediante get_schema.

- Ejecución de una consulta SQL real mediante run_query.
- Generación opcional de visualizaciones con gen_chart.
- Construcción de la respuesta final en formato Markdown, sin exponer la lógica interna.

En la figura 4.5, se muestra un diagrama que resume visualmente esta arquitectura y las conexiones entre cada componente del sistema:

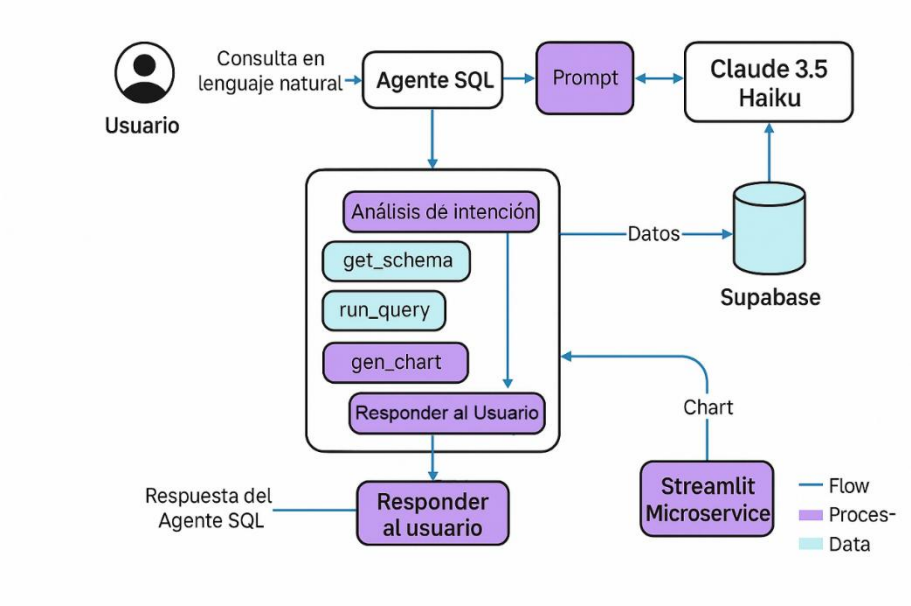


Figura 4.5: Diagrama de Flujo de Funcionamiento del Agente SQL.

Fuente: Elaboración Propia

4.6.2 Conexión a Supabase

Todos los datos estructurados del proyecto —producción, despachos, stock, terceros, proyecciones y tablas maestras— se cargan en una base de datos PostgreSQL alojada en Supabase. Esta base actúa como repositorio central del sistema, ofreciendo acceso en tiempo real a información limpia y normalizada.

El agente interactúa directamente con esta base a través de herramientas internas. Se explican en la tabla 4.26:

Tabla 4.26: Herramientas del Agente SQL y sus funciones.

Herramienta	**Función principal**
get_schema(tabla)	Obtiene la estructura de cualquier tabla o vista.
run_query(sql)	Ejecuta consultas SQL sobre la base de datos real.
gen_chart(payload)	Genera un gráfico embebible a partir de datos reales.

Fuente: Elaboración Propia

4.6.3 Módulo de Memoria Conversacional

El sistema incorpora un módulo adicional de memoria conversacional, conectado a una tabla en Supabase (n8n_chat_histories). Esta tabla almacena:

- Turno de conversación.
- Pregunta y respuesta.
- Rol del participante (humano o IA).
- Fecha y contexto.

Sin embargo, el modelo no tiene permitido usar esta memoria para razonar ni responder. Su propósito es meramente informativo y para mostrar historial al usuario si se solicita.

4.6.4 Integración con el modelo de IA

El agente está implementado sobre un LLM, específicamente Claude 3.5 Haiku de Anthropic, integrado al sistema mediante el nodo "AI Agent" de n8n.
Este modelo recibe instrucciones personalizadas mediante un prompt estructurado, que define su comportamiento en cada etapa del procesamiento:

- Análisis de la intención del usuario.
- Interpretación del esquema de datos.
- Decisión de cuándo y cómo usar herramientas.
- Formato de la respuesta final (Markdown o visualización).

El uso del modelo está regulado por una lógica estricta que prohíbe el uso de memoria previa y obliga al agente a trabajar siempre con datos actuales (ver sección "Regla ANTIMEMORIA").

4.6.5 Prompt como Núcleo Lógico

El comportamiento del agente está gobernado por un prompt experto, estructurado en secciones jerárquicas que imponen reglas estrictas. Estas aseguran trazabilidad, transparencia y rigor lógico en cada consulta procesada.

Regla ANTIMEMORIA

Uno de los principios clave del diseño fue prohibir el uso de información de turnos anteriores. Cada número, listado o gráfico debe generarse en tiempo real utilizando herramientas (run_query, gen_chart) y siempre sobre datos actualizados.

Esto elimina el riesgo de respuestas falsas o supuestas, y asegura trazabilidad completa.

Herramientas Integradas

El prompt incluye una sección dedicada a describir las herramientas disponibles para el agente, junto con las instrucciones precisas sobre cómo y cuándo debe utilizarlas.

Proyecciones Calendar-Aware

El agente es capaz de construir proyecciones de stock o producción futuras aplicando lógica de calendario laboral:

- Domingo y feriados → Despacho = 0
- Sábados → Despacho = 0.5 × promedio
- Días hábiles → Despacho completo
- Producción constante (salvo que el usuario indique lo contrario)

Estas proyecciones se calculan con datos históricos reales y se visualizan automáticamente mediante gen_chart.

Lógica SQL Avanzada

El modelo puede manejar consultas complejas que incluyen:

- Filtros combinados por zona, especie, calidad, o largo.
- Relación entre códigos abreviados y completos de productos.
- Identificación de transferencias entre zonas (JOINs entre origen y lugar).
- Uso obligatorio de vistas especializadas en ciertas comparativas.

- Cálculos agregados por fecha, zona, equipo o especie.

El prompt también incluye un diccionario semántico, permitiendo que términos como “pino podado” o “285 podado”, así como términos como “euca” sean correctamente interpretados por el modelo.

Visualizaciones Embebibles mediante Microservicio

Se desarrolló un microservicio desacoplado utilizando Streamlit, encargado de generar visualizaciones gráficas automáticas a partir de los datos consultados por el modelo. El flujo se compone de tres etapas:

1. **Generación del payload:**
   El agente crea un JSON estructurado con:
   - **labels:** categorías o fechas para el eje X
   - **values:** valores asociados
   - **chart_type:** tipo de gráfico (bar, line, multi-line)
   - **title:** título de la visualización
2. **Envío al microservicio:**
   El payload es enviado automáticamente al backend gráfico en Streamlit.
3. **Entrega del gráfico embebible:**
   El servicio responde con una URL pública, que es integrada en la conversación mediante un iframe o enlace enriquecido.

Esta arquitectura modular permite escalar el sistema visual de forma independiente, manteniendo una separación clara entre razonamiento, datos y presentación.

El proceso completo de generación de visualizaciones automáticas se resume en la figura 4.6:

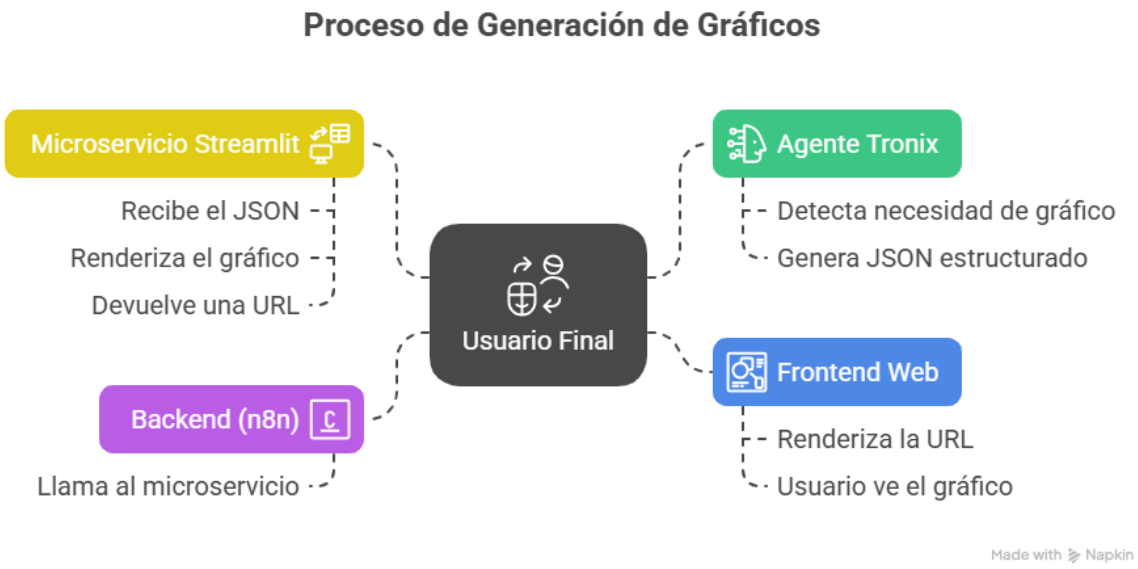


Figura 4.6: Diagrama de Flujo de Funcionamiento de a creación de Gráficos.

Fuente: Elaboración Propia

4.7 Funcionamiento y evaluación del módulo Text-to-SQL

El núcleo funcional del sistema Tronix consiste en un módulo de conversión de lenguaje natural a consultas SQL estructuradas (Text-to-SQL), el cual habilita la interacción conversacional con bases de datos relacionales sin necesidad de conocimientos técnicos. Este módulo fue diseñado para operar bajo un flujo modular que asegura interpretabilidad, precisión y robustez en entornos industriales reales.

El sistema sigue una secuencia lógica para transformar una consulta en lenguaje natural en una instrucción SQL válida:

Análisis de intención: el modelo interpreta la consulta del usuario, extrayendo la intención principal, los elementos clave (fechas, zonas, productos, métricas) y el tipo de operación requerida (búsqueda, comparación, proyección).

Extracción de entidades clave: El modelo identifica los elementos relevantes de la oración — como “productos”, “volumen”, “zona” o “últimos 30 días”— y los clasifica semánticamente como posibles columnas, filtros o métricas. Este paso se basa en el embedding contextual de cada palabra y su relación gramatical.

Resolución de intención y operación principal: A partir del análisis de la estructura de la frase, el modelo determina si la consulta requiere un SELECT, un JOIN, un GROUP BY, o funciones agregadas como SUM() o AVG(). Por ejemplo, "¿cuáles son los productos más producidos?" activa una operación de agregación sobre la tabla produccion.

Mapeo semántico con la base de datos: Una vez identificadas las entidades, el modelo debe mapearlas correctamente a las tablas y columnas reales. Aquí es donde el get_schema(tabla) juega un rol clave, ya que le permite al modelo "ver" las columnas disponibles y validar la existencia de campos como zona, fecha, volumen o codigo_producto.

Construcción de la consulta SQL: Con la información anterior, el modelo compone progresivamente la instrucción SQL. Primero arma el FROM, luego selecciona columnas (SELECT), añade filtros (WHERE), condiciones temporales (INTERVAL, DATE_TRUNC) y finalmente ordenamientos o agrupaciones si son necesarios.

Validación y ejecución segura: Antes de retornar el resultado, el agente encapsula la consulta en una función como run_query(sql) para ejecutar sobre la base real y asegurar que los datos provienen de una fuente confiable. No se aceptan respuestas generadas desde "memoria" o inferencias sin respaldo de datos.

Evaluación funcional del sistema

Para la evaluación del sistema se utilizó el mismo conjunto de preguntas definidas para la evaluación comparativa de modelos (ver sección 4.5) como base para medir la efectividad del sistema en la generación de consultas SQL. Estas preguntas, cuidadosamente diseñadas con distintos niveles de complejidad —desde saludos básicos hasta razonamientos multitabla— permitieron evaluar de forma consistente tanto la capacidad del modelo para interpretar la intención del usuario, como la calidad estructural y semántica de las instrucciones SQL generadas. Esta metodología permitió comparar de forma directa el rendimiento del sistema bajo condiciones controladas, asegurando trazabilidad entre la evaluación del modelo y el funcionamiento real del módulo Text-to-SQL.

Cada consulta fue evaluada según:

- **Precisión SQL:** exactitud sintáctica y semántica del código generado.
- **Correcta interpretación del esquema:** uso adecuado de nombres de columnas, joins y filtros.

- **Formato y claridad del output:** legibilidad y utilidad de la respuesta para el usuario.

Impacto del prompt estructurado y modular

Uno de los elementos diferenciadores de este trabajo fue la implementación del prompt altamente estructurado.

La aplicación del prompt, no solo aumentó la precisión general del sistema, sino que redujo considerablemente la tasa de errores en consultas multitabla o con condiciones complejas. Por ejemplo, con Claude 3 Sonnet, se logró reducir el tiempo de respuesta para consultas con múltiples joins y filtros desde más de **2 minutos (Haiku)** a **menos de 40 segundos**, sin sacrificar la calidad ni la estructura de las respuestas.

Manejo de consultas complejas y corrección iterativa

Durante el proceso de desarrollo y evaluación, se identificó que la correcta interpretación de la intención del usuario depende en gran medida de la coherencia semántica y consistencia nominal de las tablas en la base de datos. Por ello, se aplicaron procesos de transformación previos para normalizar los nombres de columnas, estandarizar zonas y simplificar estructuras, facilitando así el correcto "match" entre los términos utilizados por el usuario y los campos esperados en SQL.

A lo largo de las miles de consultas realizadas durante el entrenamiento y prueba del sistema, cada instrucción SQL generada fue revisada manualmente, afinando su lógica y corrigiendo posibles errores o ambigüedades. En particular, aquellas solicitudes que requerían razonamiento multietapa —como proyecciones calendarizadas, balances negativos o transferencias entre zonas— fueron incorporadas explícitamente como ejemplos detallados dentro del prompt modular, asegurando consistencia, claridad y una generación más robusta de respuestas frente a futuras consultas similares.

5. Arquitectura y Despliegue

Una vez desarrolladas las capacidades lógicas del agente conversacional, fue necesario definir una arquitectura capaz de operar el sistema de forma estable, escalable y mantenible. Se optó por un enfoque basado en microservicios desacoplados, donde cada componente crítico se despliega y gestiona por separado. Esta decisión facilita el mantenimiento, el desarrollo paralelo y la evolución independiente de cada módulo.

El sistema fue desplegado sobre Railway, una plataforma de infraestructura como servicio (PaaS) que permite levantar contenedores con rapidez, configurar variables de entorno críticas y automatizar despliegues mediante integración con GitHub.

La arquitectura final contempla tres servicios principales:

- **Backend conversacional (n8n):** encargado del flujo lógico del agente, la ejecución de herramientas (run_query, get_schema, gen_chart) y la conexión con el modelo LLM.
- **Microservicio de visualización (Streamlit):** genera gráficos embebibles a partir de datos estructurados enviados por el agente.
- **Frontend web (React + Supabase):** interfaz donde los usuarios se autentican, consultan datos y visualizan resultados enriquecidos.

5.1 Despliegue Modular en Railway

Cada uno de estos servicios fue desplegado como un proyecto independiente dentro de Railway. Esto permitió:

- Aislar responsabilidades operativas por componente.
- Escalar cada servicio según su propia demanda.
- Actualizar módulos sin afectar la estabilidad global del sistema.

5.2 Backend Conversacional (n8n)

El backend fue implementado usando n8n, herramienta de automatización visual que permite orquestar flujos, manejar sesiones y conectar con el LLM de forma programática. Para su despliegue en Railway, se configuraron variables clave:

- **WEBHOOK_URL:** URL pública necesaria para que los nodos tipo Webhook puedan recibir consultas. Esta variable debe coincidir con la URL real que Railway asigna al servicio. Si no está correctamente definida, el agente no podrá responder.
- **SUPABASE_URL y SUPABASE_SERVICE_ROLE_KEY:** credenciales de conexión con la base de datos principal, donde se ejecutan las consultas SQL.
- Variables internas (DB_TYPE, DB_POSTGRESDB_*, N8N_BASIC_AUTH_USER/PASSWORD) para control del motor de flujos y acceso seguro.

El flujo conversacional se gestiona como un workflow visual dentro de n8n, mientras que la lógica fina se define mediante prompts estructurados y nodos de herramienta.

El código y configuraciones se versionan en GitHub, y cualquier cambio en el repositorio puede ser desplegado automáticamente gracias a GitHub Actions.

5.3 Microservicio de Visualización (Streamlit)

Este servicio genera los gráficos que acompañan las respuestas del agente. Fue construido con Streamlit, usando Python y librerías como matplotlib y Plotly. Recibe un JSON estructurado del agente con los campos labels, values, chart_type y title, y devuelve una URL pública con el gráfico generado.

Fue desplegado como un contenedor independiente en Railway, con las siguientes variables de entorno:

- **SUPABASE_URL y SUPABASE_SERVICE_ROLE_KEY:** Permiten consultar la base de datos si el gráfico requiere validación adicional.
- **CHART_EXPIRATION_DAYS:** Define cuántos días estará disponible cada gráfico generado.

El código fuente se gestiona en GitHub, incluyendo:

- App.py: Archivo principal que define la lógica de entrada y visualización.
- Un endpoint HTTP que responde con la URL del gráfico.
- Estilos limpios y generación automatizada de iframes para integración con el frontend.

Gracias a la configuración de integración continua, cada push a GitHub actualiza automáticamente el microservicio en Railway.

5.4 Frontend Web (React + Supabase)

La interfaz de usuario fue desarrollada utilizando React, con una arquitectura modular basada en componentes reutilizables. Este frontend permite a los usuarios autenticarse, consultar al agente en lenguaje natural, visualizar respuestas enriquecidas y gestionar su historial de forma intuitiva y eficiente.

El sistema se conecta directamente a Supabase, que actúa como backend para:

- Autenticación segura mediante tokens persistentes.
- Almacenamiento del historial de conversaciones por usuario.
- Gestión de preguntas frecuentes y respuestas guardadas.

Funcionalidades principales:

- **Login y sesiones seguras:** mediante Supabase, garantizando protección de rutas sensibles.
- **Interfaz conversacional:** los usuarios pueden enviar consultas en lenguaje natural, recibir respuestas formateadas en Markdown y visualizaciones integradas sin salir del flujo de conversación.
- **Historial y gestión personalizada:** cada usuario accede a su historial de preguntas, puede marcar respuestas como útiles y revisar temas anteriores.
- **Visualización automática de gráficos:** si una respuesta contiene una URL hacia un gráfico generado, este se embebe automáticamente en la interfaz mediante un iframe, mejorando la experiencia sin requerir interacción adicional.

Configuración y despliegue:

El frontend fue desplegado como un sitio estático en Railway, utilizando un flujo de integración continua. Las variables de entorno necesarias para su funcionamiento incluyen:

- VITE_SUPABASE_URL
- VITE_SUPABASE_ANON_KEY

Estas claves permiten que el frontend interactúe directamente con Supabase desde el navegador, sin comprometer información sensible.

Repositorio y automatización:

El código fuente está alojado en GitHub, con estructura moderna:

- Conexión a Supabase mediante @supabase/supabase-js.
- Estilos implementados con Tailwind CSS para lograr una estética limpia y profesional.
- Componentes lógicos para el envío de prompts (solicitudes / consultas), renderizado de respuestas y gestión del estado conversacional.

Cada push a la rama principal del repositorio activa GitHub Actions, que ejecuta el flujo de CI/CD y despliega automáticamente la versión actualizada del frontend en Railway, sin necesidad de intervención manual. Esto garantiza que los cambios estén siempre sincronizados entre desarrollo y producción.

5.5 Beneficios de la Arquitectura Modular

Esta arquitectura distribuida, pero bien conectada, entrega:

- **Escalabilidad específica:** cada módulo se puede escalar según su demanda.
- **Mantenimiento sencillo:** fallos en un componente no comprometen el sistema completo.
- **Desarrollo en paralelo:** equipos distintos pueden trabajar en frontend, backend y visualizador sin interferencias.
- **Automatización del ciclo de vida:** CI/CD asegura fluidez desde el desarrollo hasta la operación.

6. Resultados

6.1 Producto Conversacional: Un MVP Funcional y Operativo

El resultado del proyecto fue el desarrollo de un agente conversacional orientado a datos forestales, capaz de responder preguntas en lenguaje natural mediante razonamiento estructurado sobre una base de datos relacional. El sistema, aunque concebido como un Producto Mínimo Viable (MVP), cumple con una serie de funcionalidades clave que lo hacen plenamente operable en un contexto productivo controlado.

El agente se encuentra desplegado sobre una arquitectura desacoplada, con microservicios gestionados por separado, incluyendo:

- Backend conversacional con LLM y lógica de herramientas
- Base de datos PostgreSQL en Supabase con tablas y vistas optimizadas
- Microservicio de generación de gráficos embebibles en Streamlit
- Frontend en React conectado a Supabase

Esta infraestructura permitió validar las capacidades del sistema de forma modular, asegurando su estabilidad y escalabilidad progresiva.

6.2 Capacidades Habilitadas por el MVP

Durante las pruebas del agente, se comprobaron las siguientes capacidades funcionales:

Interpretación semántica y lógica estructurada:

- Detección de intención del usuario (consulta directa, proyección, comparación, visualización, etc.)
- Identificación de entidades como zonas, especies, métricas, fechas y filtros compuestos
- Traducción de lenguaje natural a SQL con múltiples condiciones y operaciones agregadas

Consulta dinámica sobre múltiples tablas:

- Capacidad para acceder y cruzar información desde más de 10 tablas distintas
- Uso de vistas modeladas para reducir la complejidad lógica en el prompt
- Ejecución directa de SQL sobre Supabase mediante herramienta run_query

Generación de visualizaciones:

- Creación de gráficos de barra, línea y multiserie a partir de datos reales
- Proyecciones iterativas calendar-aware con representación visual
- Uso del microservicio Streamlit para renderizado externo y retorno de URLs embebibles

Respuestas enriquecidas y presentadas profesionalmente:

- Generación de respuestas en formato Markdown, legibles y estructuradas
- Inclusión automática de tablas, listas y visualizaciones dentro del flujo conversacional
- Separación entre lógica de razonamiento y presentación visual

Proyecciones Calendar-Aware y Escenarios Simulados:

Una de las capacidades más destacadas del agente es su habilidad para proyectar escenarios futuros basándose en datos históricos y ajustándose al calendario laboral. Las proyecciones de stock, producción o despacho se construyen aplicando reglas como:

- Despacho nulo en domingos y feriados
- Reducción de despacho en sábados o días preferenciales
- Producción constante o ajustada, según instrucciones del usuario
- Cálculo iterativo donde el valor del día siguiente depende del anterior

Además, el sistema permite ajustar variables y simular escenarios, como por ejemplo:

- Aumento o reducción del despacho
- Cambios en el ritmo de producción
- Cálculo del impacto acumulado sobre el stock en un período determinado

Estas proyecciones se visualizan automáticamente mediante gráficos embebidos, lo que facilita la toma de decisiones y la planificación operativa.

Trazabilidad y transparencia del razonamiento

- Uso de reglas estrictas definidas de forma modular en el prompt (ANTIMEMORIA, herramientas obligatorias)
- Prohibición de uso de memoria conversacional como fuente de respuesta
- Trazabilidad completa de cada respuesta mediante los logs y consultas generadas

Autenticación y personalización por usuario

- Login seguro mediante Supabase
- Historial de conversaciones persistente por sesión
- Posibilidad de guardar respuestas y visualizar preguntas frecuentes

6.3 Validación Operativa

El sistema fue sometido a un proceso de pruebas durante el mes de junio, registrando aproximadamente 1500 interacciones que cubrieron una amplia gama de consultas representativas del entorno forestal. Las pruebas incluyeron:

- Reportes de stock segmentados por zona, especie y calidad.
- Comparativas entre planificación y despacho real, tanto para terceros como para plantas propias.
- Proyecciones de stock calendar-aware, ajustadas por domingos, feriados y escenarios modificados.
- Simulación de escenarios con variación de producción o despacho.
- Detección de transferencias entre zonas, aplicando lógica SQL avanzada.
- Análisis multiespecie y multizona con generación automática de visualizaciones.
- Identificación de métricas críticas como balances negativos o tendencias de acumulación.

En un 85% de los casos (≈1.275 respuestas), el agente fue capaz de entregar resultados correctos, completos y comprensibles, cumpliendo con los criterios funcionales definidos:

- Exactitud sintáctica (SQL bien formado y ejecutable).
- Relevancia semántica (respuesta coherente con la intención del usuario).
- Trazabilidad total (uso explícito de herramientas y datos en tiempo real).
- Visualización enriquecida, cuando correspondía.

El 15% restante, aunque no entregó respuestas totalmente correctas, fue crucial para afinar el sistema. Estas fallas expusieron errores de razonamiento o relaciones mal definidas, lo que permitió ajustar herramientas, ampliar alias y fortalecer la lógica del agente, mejorando su desempeño de forma continua.

6.3.1 Costos y rendimiento

La validación no solo evaluó la calidad de las respuestas, sino también la eficiencia operativa. En la tabla 6.1 se resumen los principales indicadores registrados durante junio:

Tabla 6.1: Métricas de Uso y Costos Operativos del Sistema.

Métrica	**Valor aproximado (junio 2025)**
Consultas totales estimadas	~1.500 consultas
Tiempo promedio por consulta	1 min (simples) / 3 min (proyecciones)
Costo total modelo IA	$17.91 USD
Costo total infraestructura	$13.00 USD (Railway)
Costo por consulta	$0.0206 USD
Modelo utilizado	Claude 3.5 Haiku

Fuente: Elaboración Propia

Este desempeño validó no solo la funcionalidad, sino también la viabilidad técnica y económica del sistema como MVP. El agente demostró ser capaz de interpretar lenguaje natural, ejecutar lógica compleja sobre una base real y generar visualizaciones útiles, todo con un costo operativo que bordea los $30 USD/mes, lo que lo posiciona como una herramienta escalable y accesible para equipos que requieran apoyo analítico sin depender de conocimientos técnicos especializados.

6.4 Proyección de Uso y Valor Potencial

A pesar de ser un MVP, el sistema presenta un alto valor potencial para su uso en operaciones reales, especialmente en contextos donde:

- Se requiere acceso rápido y flexible a información operativa sin depender de analistas técnicos
- Las respuestas deben estar trazadas, basadas en datos actuales y visualmente comprensibles
- La escalabilidad modular sea una ventaja (e.g., nuevos usuarios, nuevas vistas, nuevos módulos)

El enfoque adoptado puede extenderse fácilmente a otras áreas de la empresa (logística, abastecimiento, planificación, etc.) o replicarse en industrias similares con estructura de datos compleja.

7. Conclusión

Este proyecto nació desde una problemática concreta y urgente: la dificultad de acceder, consultar y utilizar de forma eficiente los datos operacionales que sostienen la cadena logística forestal de Forestal Arauco. En la Central Integrada de Operaciones (CIO), donde se planifican y monitorean diariamente los movimientos que abastecen a las plantas de celulosa y aserraderos de la compañía, la información crítica se encontraba fragmentada, dispersa en planillas Excel, con estructuras inconsistentes, sin trazabilidad ni estandarización. Esto no solo ralentizaba la toma de decisiones, sino que generaba dependencia técnica, errores en el análisis y una brecha entre lo que ocurre en terreno y lo que se podía visualizar desde los sistemas.

Frente a este desafío, el desarrollo del sistema conversacional del agente SQL no se limitó a incorporar inteligencia artificial: fue necesario construir desde cero una base de datos relacional. Esto implicó diseñar un pipeline de transformación robusto que procesó más de diez fuentes distintas, normalizando formatos, corrigiendo miles de inconsistencias, eliminando ruido operativo y modelando estructuras que permitieran consultas SQL eficientes, interpretables y auditables.

Además, se desarrolló una arquitectura modular basada en microservicios, que incluye:

- Un backend conversacional automatizado (n8n) con lógica propia.
- Un microservicio gráfico que permite generar visualizaciones interactivas embebidas.
- Un frontend web profesional donde cada usuario puede consultar, guardar, y revisar sus interacciones.

Todo esto integrado con Supabase como base de datos central, desplegado en Railway y versionado con GitHub Actions, asegurando trazabilidad y replicabilidad completa.

En términos de resultados, el impacto fue claro y medible:

- Se habilitó la posibilidad de consultar stock, producción, despachos o proyecciones por zona, especie o calidad en lenguaje natural, sin necesidad de conocimientos técnicos.
- El agente fue capaz de responder correctamente más del 80 % de las consultas en pruebas reales, con tiempos de respuesta variables según la consulta del usuario.
- Se generaron visualizaciones automáticas de alta calidad, integradas directamente en el flujo conversacional, mejorando la comprensión de los datos y facilitando la toma de decisiones.

Este sistema representa un cambio de paradigma en la forma en que la empresa puede relacionarse con sus datos operacionales: ya no a través de filtros estáticos ni dashboards predefinidos, sino mediante diálogo, lógica y entendimiento.

El valor de este proyecto no está solo en la tecnología, sino en la solución integral que se logró construir: una plataforma robusta, alineada con los flujos operativos reales, capaz de escalar, adaptarse y mantenerse en el tiempo. El agente SQL no es solo un asistente; es una nueva forma de pensar el acceso a la información forestal en Arauco: más ágil, más inteligente y verdaderamente al servicio de quienes toman decisiones en el bosque y en la sala de control.

8. Referencias

Brown, T. (2009). *Change by Design: How Design Thinking Transforms Organizations and Inspires Innovation*. HarperBusiness.

CRISP-DM. (2000). *Cross Industry Standard Process for Data Mining*. The CRISP-DM Consortium.

Kimball, R., & Caserta, J. (2004). *The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data*. Wiley.

McTear, M. (2016). *Conversational AI: Dialogue Systems, Conversational Agents, and Chatbots*. Synthesis Lectures on Human Language Technologies.

Radziwill, N. M., & Benton, M. C. (2017). Evaluating quality of chatbots and intelligent conversational agents. *Quality Engineering*, 29(4), 624-634.

Rahm, E., & Do, H. H. (2000). Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin*, 23(4), 3-13.

Schwaber, K., & Sutherland, J. (2020). *The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game*. Scrum.org.

Segel, E., & Heer, J. (2010). Narrative visualization: Telling stories with data. *IEEE Transactions on Visualization and Computer Graphics*, 16(6), 1139-1148.

Streamlit. (2024). *Streamlit Documentation*. Streamlit Inc.

Supabase. (2024). *Supabase Documentation*. Supabase Inc.

Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., ... & Radev, D. (2018). Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql tasks. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 3911-3921.

Amershi, S., Weld, D., Vorvoreanu, M., Fourney, A., Nushi, B., Collisson, P., ... & Horvitz, E. (2019). *Guidelines for human-AI interaction*. In Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (pp. 1–13). ACM. https://doi.org/10.1145/3290605.3300233

Helicone. (2025). *LLM API providers: Comparison and leaderboard*. https://www.helicone.ai/blog/the-complete-llm-model-comparison-guide

Shakudo. (2025). *Top 9 large language models as of July 2025*. https://www.shakudo.io/blog/top-9-large-language-models

Upmarket. (2025). *Best AI chatbots & LLMs of Q1 2025: Complete comparison*. https://www.upmarket.co/blog/the-best-ai-chatbots-llms-of-q1-2025

Wang, Z., Wang, W., Zhang, S., Liu, Q., & Xiong, C. (2023). *SQLEval: A comprehensive evaluation benchmark for text-to-SQL models*. arXiv. https://arxiv.org/abs/2305.13406

Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., ... & Radev, D. (2018). *Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL tasks*. In Proceedings of EMNLP 2018 (pp. 3911–3921). https://doi.org/10.18653/v1/D18-1425

LM Arena. (2025). *Leaderboards for Language Model Evaluation*. https://lmarena.org

BirdSQL. (2025). *Benchmarking In-the-Wild Reasoning for Databases*. https://birdsql.github.io

Anthropic. (2025). *Claude 3 Model Card*. https://www.anthropic.com/news/claude-3-family

OpenAI. (2025). *GPT-4 Turbo Technical Report*. https://platform.openai.com/docs/models/gpt-4-turbo

9. Anexos

Anexo N°1: Pseudocódigo de transformación tabla "stock_canchas"

Algoritmo 3: Pseudocódigo Limpieza Stock Operativo Canchas

Input: CSV Stock Operativo (SO_*.csv) delimitado por ";
Output: CSV limpio (SO_*_ready_oficial.csv)

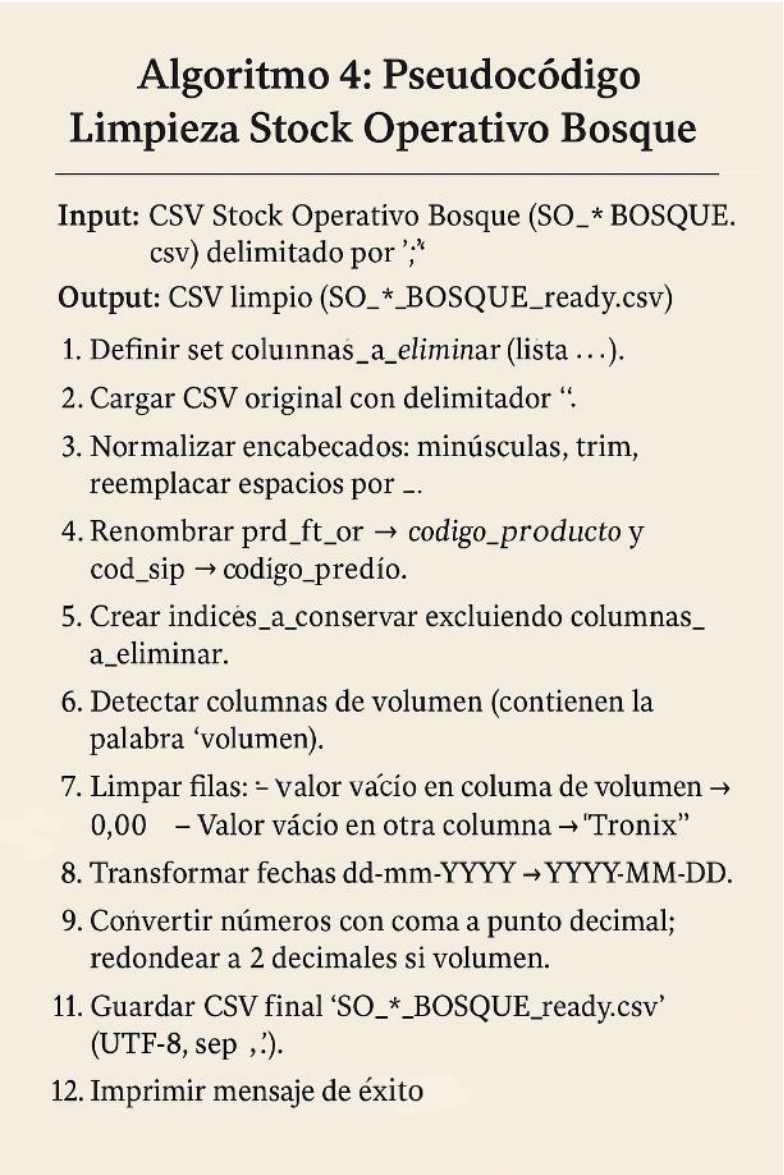
1. Definir lista columnas_a_eliminar.
2. Cargar CSV original con delimitador ",'.
3. Normalizar encabezados: minúsculas, trim, reemplazar espacios por –.
4. Renombrar prd_ft_or → codigo_producto y cod_sap → codigo_origen.
5. Construir indices_a_conservar excluvendo columnas_a_eliminar y variantes de key_volumen.
6. Filtrar encabezados para conservar solo indices_a_conservar.
7. Detectar columnas de volumen (contienen la palabra 'volumen).
8. Limpiar filas:
   – Si valor vacío en columna de volumen → 0.
   – Si valor vacío en otra columna → "Tronix".
9. Transformar fechas dd mm-YYYY → YYYVM-DD.
10. Gonvertir nũmeros con coma a punto decimal.
11. Guardar CSV final 'SO_*_ready_oficial.csv' (UTF-8, sep =').
12. Imprimir mensaje de éxito

Figura 9.1: PseudoCódigo tratamiento tabla Stock en Canchas.

Fuente: Elaboración Propia.

.

Anexo N°2: Pseudocódigo de transformación tabla "Stock en Predios"



Algoritmo 4: Pseudocódigo Limpieza Stock Operativo Bosque

Input: CSV Stock Operativo Bosque (SO_* BOSQUE. csv) delimitado por ';'

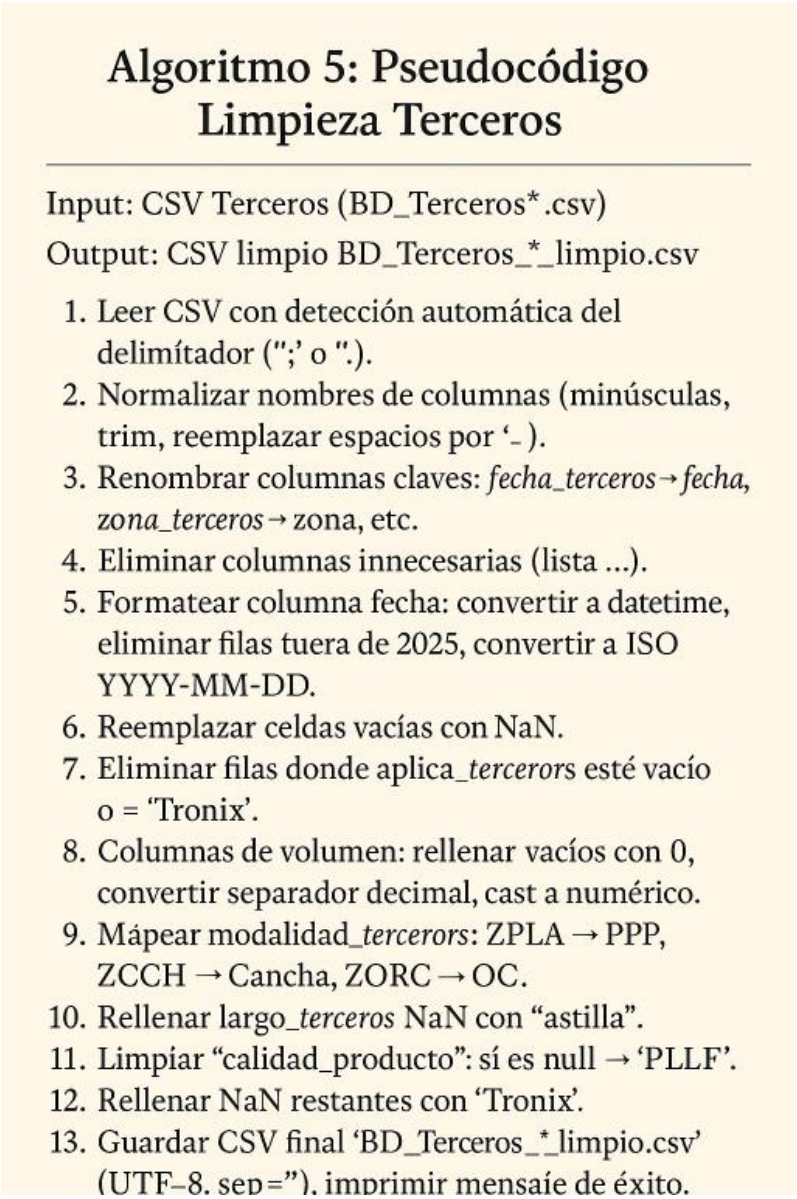
Output: CSV limpio (SO_*_BOSQUE_ready.csv)

1. Definir set columnas_a_*eliminar* (lista ...).
2. Cargar CSV original con delimitador ''.
3. Normalizar encabecados: minúsculas, trim, reemplacar espacios por _.
4. Renombrar prd_ft_or → *codigo_producto* y cod_sip → codigo_predio.
5. Crear indices_a_conservar excluiendo columnas_ a_eliminar.
6. Detectar columnas de volumen (contienen la palabra 'volumen).
7. Limpar filas: - valor vacío en columa de volumen → 0,00 – Valor vácio en otra columna → 'Tronix"
8. Transformar fechas dd-mm-YYYY → YYYY-MM-DD.
9. Convertir números con coma a punto decimal; redondear a 2 decimales si volumen.
11. Guardar CSV final 'SO_*_BOSQUE_ready.csv' (UTF-8, sep ,').
12. Imprimir mensaje de éxito

Figura 9.2: PseudoCódigo tratamiento tabla Stock en Predios.

Fuente: Elaboración Propia.

Anexo N°3: Pseudocódigo de transformación tabla "Terceros"



Algoritmo 5: Pseudocódigo Limpieza Terceros

Input: CSV Terceros (BD_Terceros*.csv)

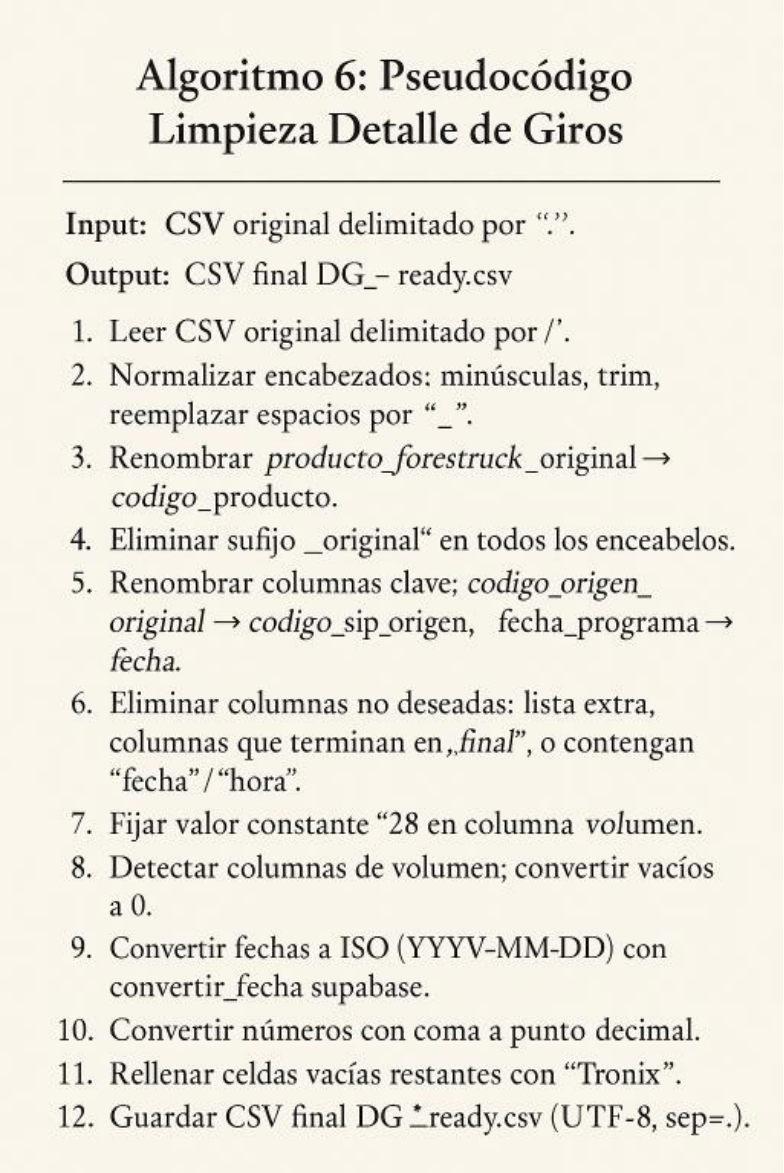
Output: CSV limpio BD_Terceros_*_limpio.csv

1. Leer CSV con detección automática del delimítador (';' o ',').
2. Normalizar nombres de columnas (minúsculas, trim, reemplazar espacios por '_').
3. Renombrar columnas claves: *fecha_terceros* → *fecha*, *zona_terceros* → zona, etc.
4. Eliminar columnas innecesarias (lista ...).
5. Formatear columna fecha: convertir a datetime, eliminar filas tuera de 2025, convertir a ISO YYYY-MM-DD.
6. Reemplazar celdas vacías con NaN.
7. Eliminar filas donde aplica_*tercero*rs esté vacío o = 'Tronix'.
8. Columnas de volumen: rellenar vacíos con 0, convertir separador decimal, cast a numérico.
9. Mápear modalidad_*tercerors*: ZPLA → PPP, ZCCH → Cancha, ZORC → OC.
10. Rellenar largo_*terceros* NaN con "astilla".
11. Limpiar "calidad_producto": sí es null → 'PLLF'.
12. Rellenar NaN restantes con 'Tronix'.
13. Guardar CSV final 'BD_Terceros_*_limpio.csv' (UTF-8, sep=','), imprimir mensaje de éxito.

Figura 9.3: PseudoCódigo tratamiento tabla Terceros.

Fuente: Elaboración Propia.

Anexo N°4: Pseudocódigo de transformación tabla "Detalle de Giros"



Algoritmo 6: Pseudocódigo
Limpieza Detalle de Giros

Input: CSV original delimitado por ".".

Output: CSV final DG_– ready.csv

1. Leer CSV original delimitado por /'.
2. Normalizar encabezados: minúsculas, trim, reemplazar espacios por "_".
3. Renombrar *producto_forestruck*_original → *codigo*_producto.
4. Eliminar sufijo _original" en todos los enceabelos.
5. Renombrar columnas clave; *codigo_origen_ original* → *codigo*_sip_origen, fecha_programa → *fecha.*
6. Eliminar columnas no deseadas: lista extra, columnas que terminan en „*final*", o contengan "fecha" / "hora".
7. Fijar valor constante "28 en columna *vo*lumen.
8. Detectar columnas de volumen; convertir vacíos a 0.
9. Convertir fechas a ISO (YYYV-MM-DD) con convertir_fecha supabase.
10. Convertir números con coma a punto decimal.
11. Rellenar celdas vacías restantes con "Tronix".
12. Guardar CSV final DG _ready.csv (UTF-8, sep=.).

Figura 9.4: PseudoCódigo tratamiento tabla Giros Programados.

Fuente: Elaboración Propia.

Anexo N°5: Pseudocódigo de transformación tabla "gruas"

Algoritmo 7: Pseudocódigo Limpieza Gruas

Input: CSV Gruas (BD_Gruas*.csv) delimitado por ';' o '.,.'.

Output: CSV limpio Gruas_-_limpio.csv (UTF-8, sep='').

1. Leer CSV original l y auto-dectar delimitador ('' o ';').
2. Normalizar encabezados: minúsculas, trim, reeplacar espacos por _', eliminar BOM y tildes.
3. Renombrar columnas: dia_natural → fecha; volumen_m3 → ıolumen.
4. Definir columnas_a_eliminar [tipo, dia_mes, key_turno, no_considerar, aplica_descarguio,
5. Construir lista indices_a_matener excluiendo columnas_a_eliminar.
6. Filtrar encabezados para conservar lo indices_a_matener.
7. Procesar filas:
   - Si header és fecha → convertir de YYYMDD a VYYY-M-DD.
   - Si valor vacío en columa volumen → 0, otra columna vaciá →'Tronix'.
8. Guardar CSV final 'Gruas_limpio.csv' (UF-8;).
9. Imprimir mensaje de éxito

Figura 9.5: PseudoCódigo tratamiento tabla Grúas.

Fuente: Elaboración Propia

Anexo N°6: Pseudocódigo de transformación tabla "Plan despachos"

Algoritmo 8: Pseudocódigo Limpieza Plan de Abastecimiento

Input: archivo Excel

Output: datos limpios

1 Leer archivo Excel (Libro1.xlsx)
2 Renombrar columnas base (cod_destino → codigo destino)
3 Transformar a formato largo (melt)
4 Eliminar filas sin volumen
5 Normalizar strings en columnas
6 Filtrar filas con 'QUEMADO' en producto
8 Filtrar filas con codigo_destino = 'C035)
9 Detectar es_tercero por preséncia de "PPP'
10 Eliminar 'PPP'
11 Separar código y zona
12 Mapear zona_origen (BC01 → ARAUCO, FT01 → CONSTITUCION, FC00 → CHILLAN, FS01 → VALDIVIA, otro → DESCONOCIDO)
13 Limpiar palabra 'MANCHADO
14 Eliminar columna 'codigo_base_match' (si existe)
15 Eliminar columnas zona_operacion, codigo_ producto_original, descripción_destino, MAYUSCL
16 Normalizar zona_origen a MAYUSCULAS
17 Guardar CSV final 'plan_abastecimiento_ transformado.csv' (UTF-8-SIG)
18 Imprimir mensaje de éxito

Figura 9.6: PseudoCódigo tratamiento tabla Plan despachos.

Fuente: Elaboración Propia

Anexo N°7: Pseudocódigo de transformación tabla "Proyecciones"

Algoritmo 9: Pseudocódigo Limpieza Proyecciones

Input: Leer archivo Excel (proyecciones.xlsx)

Output: CSV limpio con proyecciones

1 Renombrar columnas: lower-case, spaces to _ map volssc → volumen_proyectado, tipo_prod → calidad, diatexto → fecha

2 Eliminar columnas no deseadas (list)

3 Renombrar columnas adicionales: nombre_dest → destino, interv → intervencion, cod → codigo_predio_origen, est_madera → estado_madera, esp → especie, emsefor → empresa serv́cios forestales, ep → encargado_de_produccion

4 Detectar columnas de volumen (contengan 'volumen')

5 Limpiar celdas vacias: si columna de volumen → 0, si otras → 'Trenix'

6 Si columna = 'fecha': aplicar *transformar_fecha*; else: aplicar *transformar_numero*

7 Guardar CSV final proyecciones_ready.csv (UTF-8, sep='')

8 Imprimir mensaje de éxito

Figura 9.7: PseudoCódigo tratamiento tabla Proyecciones.

Fuente: Elaboración Propia

Anexo N°8: Pseudocódigo de transformación tabla "Plan Producción OptiCort"

Algoritmo 10 : Pseudocódigo Limpieza Plan de Producción Opticort

Input: archivo Excel (205066.xlsx)
Output: CSV limpio (20506_limpio.csv)

1 Leer archivo Excel (205066.xlsx)
2 Normalizar nombres de columnas (minúsculas, _)
3 Eliminar columnas no deseadas (*sap_ems, emsefor, predio, pendi, pertenencia_máquina, tipo_maquina, temporada, opticort*)
4 Renombrar columnas: total ssc → volumen_proyetado, producto → calidad, zona2 → zona, id_predio → codigo
5 Fijar parámetros: año_actual ← 2025, mes_actual ← 6
6 Para cada columna con 'fecha': convertir a datetime
7 Filtrar filas donde *fecha* ∈ junio 2025
8 Formatear fecha a ISO (YYYY-MM.DD
9 Guardar CSV final '202056_*limpio.csv*' (UTF-8, sep=')
10 Imprimir mensaje de éxito

Figura 9.8: PseudoCódigo tratamiento tabla Plan producción OptiCort.

Fuente: Elaboración Propia

Anexo N°9: Glosario Técnico y Conceptos Clave

Concepto	Definición breve
SQL	Lenguaje utilizado para consultar y manipular bases de datos relacionales. Permite obtener, filtrar, agrupar o modificar información de forma estructurada.
API	Interfaz que permite la comunicación entre diferentes sistemas o servicios. En este proyecto se usan para conectar el agente con Supabase y el microservicio de gráficos.
Prompt	Instrucción estructurada que guía el comportamiento del modelo de lenguaje. En Tronix, define reglas, herramientas disponibles y formato de respuestas.
n8n	Plataforma de automatización visual que permite orquestar flujos entre sistemas como el modelo IA, Supabase o el generador de gráficos.
Supabase	Plataforma de base de datos en la nube que funciona como backend para almacenar información estructurada (stock, producción, historial de preguntas, etc).
Streamlit	Herramienta para construir interfaces visuales y dashboards. En este sistema se usa como microservicio para generar gráficos embebidos.
JSON	Formato estructurado de texto usado para enviar datos entre sistemas. Por ejemplo, el agente construye un JSON con datos para graficar.
LLM	Large Language Model, Tipo de inteligencia artificial capaz de entender y generar lenguaje natural, similar al humano.
Embebido	Integración directa de un gráfico o componente dentro de otro sistema (como un chat), sin necesidad de abrir otra página.

Tabla 9.1: Glosario de Términos Importantes.

Fuente: Elaboración Propia.