MARZO 2025

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE SISTEMAS INFORMÁTICOS

GRADO EN INGENIERÍA DE SOFTWARE

PROYECTO DE FIN DE GRADO

QueryOrchestrator

Autor: Álvaro Seoane Alfonso

RESUMEN

El almacenamiento y la gestión de datos es un aspecto crítico en cualquier proyecto software. Los datos que se generan en una aplicación son muchas veces más importantes que el software en sí, puesto que el funcionamiento más básico de casi cualquier producto software consiste en recibir, procesar y almacenar datos, estos datos generalmente tienen valor tanto para el usuario como para la empresa.

Por este motivo se invierte mucho tiempo y dinero en garantizar la integridad de esos datos, normalmente haciendo copias de seguridad o replicando los datos en varias instancias de bases de datos al mismo tiempo.

Este proyecto, QueryOrchestrator, ofrece un servicio para mantener varias instancias de bases de datos igualadas en todo momento. Haciendo de intermediario entre el software y las bases de datos y generando además por cada consulta una consulta de rollback de forma completamente automática de forma que siempre que se necesite se pueda volver atrás a un punto en concreto del ciclo de vida de los datos del software.

ABSTRACT

Data storage and management is critical for any software product. The data generated in an application is oftentimes more important than the software itself, given that the basic functionality of any software is to gather, process and store data, so the generated data is valuable for both the user and the company running the software.

For this very reason a lot of time and money is invested to guarantee the integrity of all the data. Usually companies do backups or have more than one instance of a database and keep them balanced.

This project, QueryOrchestrator, offers a service oriented to maintain various database instances balanced. Acting as a middleware between the databases and the software, even generating a rollback query for every query processed. That way the software can go back to any point of its life cycle easily.

ÍNDICE

ÍNDICE DE ILUSTRACIONES

INTRODUCCIÓN

La idea para este proyecto surge a raíz de una necesidad real, tras llevar un tiempo trabajando en una empresa ayudando a mantener un software, migrarlo a diferentes tecnologías y desarrollar nuevas funcionalidades. Se me ocurrió que un producto con estas cualidades sería de gran utilidad y ahorraría mucho tiempo al equipo de desarrollo, en concreto el software con el que cuenta mi equipo ya dispone de un balanceador para gestionar dos instancias de datos en producción, pero carece de la generación automática de consultas de rollback.

QueryOrchestrator propone una solución para esta problemática, además de una integración sencilla y sin necesidad de instalar dependencias específicas para conectar el software con la base de datos directamente.

El desarrollo del proyecto me ha servido para afianzar mis conocimientos sobre bases de datos y desarrollo en Python.

En esta memoria se explicarán todas las características del software así como se detalla su planificación y el proceso de desarrollo al completo.

ALCANCE

El proyecto QueryOrchestrator tiene como objetivo principal proporcionar una solución centralizada para la gestión y orquestación de múltiples bases de datos distribuidas. Su funcionalidad principal se enfoca en permitir la ejecución de consultas simultáneas sobre un conjunto agrupado de bases de datos, facilitando la obtención de resultados agregados y simplificando la interacción con múltiples fuentes de datos desde una única interfaz API.

QueryOrchestrator está diseñado para ser extensible, seguro y eficiente, y busca resolver la complejidad operativa y de mantenimiento que surge al interactuar con múltiples bases de datos de manera manual. Este proyecto está pensado tanto para entornos de desarrollo como de producción.

METODOLOGÍAS DE TRABAJO

A continuación se detallan las principales metodologías de trabajo que se han seguido para el grueso de desarrollo del proyecto: scrumban y git-flow.

Scrumban forma parte de la familia de metodologías de desarrollo ágiles. Se trata de una metodología híbrida que combina elementos y prácticas tanto de Scrum como de Kanban. Esta forma de desarrollo permite un planificación flexible, incorporando la estructura iterativa de Scrum junto con la visualización del flujo de trabajo de Kanban, que mejora la visibilidad del estado del proyecto.

Git-flow es la estrategia de ramificación que se ha escogido para el control de versiones del proyecto. Este modelo ofrece una estructura clara y ordenada para el desarrollo del proyecto y se basa en el uso de ramas específicas para diferentes propósitos.

SCRUMBAN

Tal y como se ha explicado brevemente, Scrumban es una metodología híbrida que toma elementos de Scrum y Kanban, que forman parte de la familia de las metodologías Ágiles.

Las metodologías ágiles surgieron como una respuesta a las limitaciones de los enfoques tradicionales de desarrollo de software, que se consideraban demasiado rígidos y lentos para adaptarse a los cambios rápidos. En 2001, un grupo de 17 desarrolladores formuló el Manifiesto Ágil, estableciendo los valores y principios que guían estas nuevas metodologías (1).

Principios Fundamentales del Manifiesto Ágil

El Manifiesto Ágil se basa en cuatro valores clave:

1. Individuos e interacciones sobre procesos y herramientas.
2. Software funcionando sobre documentación extensiva.
3. Colaboración con el cliente sobre negociación contractual.
4. Respuesta ante el cambio sobre seguir un plan.

Y doce principios (2):

1. Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
2. Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
3. Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al período de tiempo más corto posible.
4. Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.
5. Construimos proyectos en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.
6. El método más eficiente y eficaz de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.

7. El software funcionando es la medida principal de progreso.

8. Los procesos ágiles promueven un desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de manera indefinida.

9. La atención continua a la excelencia técnica y al buen diseño mejora la agilidad.

10. La simplicidad es el arte de maximizar la cantidad de trabajo no realizado es esencial.

11. Las mejores arquitecturas, requisitos y diseños emergen de equipos autoorganizados.

12. A intervalos regulares, el equipo reflexiona sobre cómo ser más efectivo para, a continuación, ajustar y perfeccionar su comportamiento en consecuencia.

Las metodologías ágiles no son soluciones únicas para todos los proyectos; su aplicación debe adaptarse a las necesidades específicas del equipo y del entorno, es por esto por lo que se ha escogido scrumban para el desarrollo de QueryOrchestrator, porque esta metodología híbrida ofrece características y herramientas importantes y de gran utilidad para un desarrollo como el que se presenta.

SCRUM

Scrum es un marco de trabajo ágil diseñado para facilitar la gestión y desarrollo de proyectos complejos, especialmente en el ámbito del software. Se basa en principios iterativos e incrementales, promoviendo la colaboración y la entrega continua de valor al cliente.

Origen de Scrum

Scrum se fundamenta en la teoría del control empírico de procesos, apoyándose en tres pilares (3):

- Transparencia: Todos los aspectos significativos del proceso deben ser visibles para quienes son responsables del resultado.
- Inspección: Los miembros del equipo inspeccionan regularmente los artefactos y el progreso hacia el objetivo, detectando posibles desviaciones o problemas.
- Adaptación: Si se identifican aspectos fuera de los límites aceptables, el proceso o el producto se ajusta según sea necesario.

Estructura de Scrum

Scrum define roles, eventos y artefactos específicos que estructuran el proceso de desarrollo (4):

Roles Principales:

- Product Owner: Responsable de maximizar el valor del producto y gestionar el backlog del producto, priorizando las características según el valor para el negocio.
- Scrum Master: Facilita el proceso Scrum, asegurando que el equipo comprenda y siga las prácticas de Scrum, y elimina impedimentos que puedan surgir.
- Equipo de Desarrollo: Profesionales que trabajan en la entrega del incremento del producto, autoorganizados y multifuncionales.

Eventos Clave:

- Sprint: Iteración de trabajo con una duración fija (generalmente de dos a cuatro semanas) en la que se crea un incremento del producto potencialmente entregable.

- Sprint Planning: Reunión al inicio de cada sprint donde se planifica el trabajo a realizar.

- Daily Scrum: Reunión diaria de 15 minutos donde el equipo sincroniza actividades y discute el progreso.

- Sprint Review: Al final del sprint, se presenta el trabajo completado a los stakeholders para obtener feedback.

- Sprint Retrospective: Sesión donde el equipo reflexiona sobre el sprint pasado y propone mejoras para futuros sprints.

Artefactos Principales:

- Product Backlog: Lista priorizada de todo lo que podría ser necesario en el producto, mantenida por el Product Owner.

- Sprint Backlog: Conjunto de elementos seleccionados del Product Backlog para el sprint actual, junto con un plan para entregarlos.

- Incremento: Versión del producto con las funcionalidades completadas durante el sprint, que cumple con la definición de "hecho" acordada por el equipo.

Principales Ventajas de Scrum

- Flexibilidad y Adaptabilidad: Scrum permite adaptarse rápidamente a cambios en los requisitos del proyecto.

- Mejora de la Comunicación y Colaboración: La estructura de Scrum fomenta la comunicación constante entre los miembros del equipo y con los stakeholders.

- Entrega Rápida de Resultados: Al trabajar en sprints cortos, los equipos pueden entregar funcionalidades de manera rápida y frecuente.

- Satisfacción del Cliente: La entrega continua de incrementos funcionales aseguran que el producto final se alinee con las necesidades del cliente.

- Reducción de Costes: La identificación temprana de problemas y la capacidad de adaptarse a cambios ayudan a reducir costes asociados al desarrollo.

- Equipos Más Productivos y Motivados: La autoorganización y la participación activa en la toma de decisiones aumentan la motivación y productividad del equipo.

KANBAN

Kanban es una metodología ágil de gestión del flujo de trabajo que ayudaa visualizar y mejorar los procesos. Se centra en la entrega continua y en equilibrar las demandas con la capacidad disponible (5).

Principios Fundamentales de Kanban

- Visualizar el Trabajo: Utiliza tableros visuales para representar las tareas y su estado actual.
- Limitar el Trabajo en Proceso (WIP): Establece límites en la cantidad de tareas que pueden estar en progreso simultáneamente.
- Gestionar el Flujo.
- Hacer Explícitas las Políticas del Proceso: Define y comunica claramente las reglas y procedimientos del flujo de trabajo.
- Implementar Ciclos de Retroalimentación: Realiza revisiones periódicas.
- Mejorar Colaborativamente y Evolucionar Experimentalmente: Fomenta la colaboración.

Principales Ventajas de Kanban

- Visualización Clara del Proceso: El uso de tableros Kanban proporciona una representación visual del estado de cada tarea.
- Flexibilidad y Adaptabilidad: A diferencia de otras metodologías ágiles, Kanban permite incorporar cambios en cualquier momento.
- Mejora Continua: Al centrarse en la optimización constante del proceso, Kanban promueve la identificación y eliminación de ineficiencias.
- Reducción del Tiempo de Entrega: Al limitar el trabajo en progreso y gestionar activamente el flujo, se minimizan los tiempos de espera y se acelera la entrega.
- Mayor transparencia y colaboración: La visualización del trabajo y las políticas explícitas fomentan una comunicación abierta y una comprensión compartida.
- Prevención de la Sobrecarga de Trabajo: Al establecer límites en el trabajo en curso, Kanban ayuda a equilibrar la carga laboral.

SCRUMBAN

Scrumban es una metodología ágil que fusiona elementos de Scrum y Kanban, combinando la estructura iterativa de Scrum con la visualización del flujo de trabajo de Kanban. Concebida para facilitar la transición de equipos desde Scrum hacia Kanban, Scrumban se ha convertido en un enfoque autónomo. (6)

Características Principales de Scrumban

- Visualización del trabajo: Utiliza tableros Kanban para representar visualmente las tareas y su estado actual.
- Límites de trabajo en proceso (WIP): Restricciones en la cantidad de tareas que pueden estar en progreso simultáneamente.
- Planificación bajo demanda: A diferencia de Scrum, que tiene sprints predefinidos, Scrumban permite la planificación cuando es necesario.
- Priorización continúa: Las tareas se priorizan de manera continua.
- Mejora Continua (Kaizen): Fomenta la revisión y adaptación constante de procesos para optimizar el rendimiento y la calidad del trabajo entregado. (7)

Ventajas de Scrumban

- Flexibilidad y adaptabilidad: Permite a los equipos responder rápidamente a cambios en los requisitos o prioridades.
- Mejora de la productividad: Al limitar el trabajo en progreso y enfocarse en tareas prioritarias, se reducen los tiempos de entrega y se aumenta la eficiencia del equipo.
- Visualización clara del proceso: El uso de tableros proporciona una representación transparente del estado de cada tarea.
- Reducción de desperdicios: Al enfocarse en tareas que aportan valor y minimizar actividades innecesarias se optimiza el uso de recursos. (8)

METODOLOGÍA DURANTE EL DESARROLLO

Durante el desarrollo de QueryOrchestrator se ha optado por planificar haciendo uso de una metodología ágil puesto que ayudan con la planificación, documentación y gestión general del proyecto.

Las metodologías ágiles están pensadas para ser implementadas por equipos de desarrollo completos, en los que cada miembro de equipo asume un rol definido por la metodología. En este caso dado que el proyecto ha sido desarrollado por una única persona no se ha podido seguir la metodología completamente. Aun así, se ha escogido Scrumban como referencia por los siguientes motivos:

- Tablero: El tablero Kanban estaba definido por cada sprint y ha ayudado a la organización de las tareas.

- WIP: El límite de trabajo en progreso que aporta Kanban ha sido especialmente útil en segregar tareas en otras más pequeñas cuando ha sido necesario.

- Sprints: La división del trabajo en Sprints que facilita el desarrollo al conceder una visión general del proyecto desde el inicio.

- Story Points: La estimación de la tareas haciendo uso del método de los SP que proporciona Scrum ayuda tanto con la visualización del flujo de trabajo como con la organización de los sprints, evitando de esta forma la sobrecarga de tareas por Sprint.

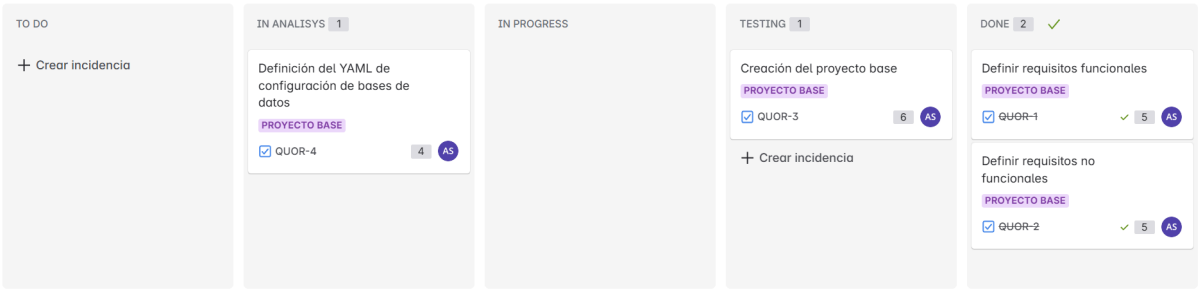
TABLERO

El tablero que se ha utilizado durante el desarrollo del proyecto es uno de los elementos de mayor utilidad y más importantes que aporta Kanban, y permite llevar un control exhaustivo del estado de todas las tareas además de proporcionar una visión general del avance del sprint en curso.

El tablero clásico de Kanban permite que las tareas pasen por tres estados esenciales: “TO DO”, “IN PROGRESS” y “DONE”. Estos tres estados definen de forma básica el ciclo de vida de cada tarea, para este desarrollo se ha decidido profundizar más en estos estados segregando el estado de “IN PROGRESS” en dos estados más “IN ANALYSIS” y “TESTING”, estos dos nuevos estados hacen las funciones de representar cuando se está analizando la mejor forma de abordar una tarea y la fase de pruebas de la nueva funcionalidad, respectivamente.

Kanban también define un límite de trabajo en progreso en el caso de este desarrollo, se ha decidido medir el “WIP” en tareas, dos tareas en progreso como máximo. Para que una sea considerada en progreso, su estado en el tablero debe estar entre los siguientes: “IN ANALYSIS”, “IN PROGRESS” y “DONE”.

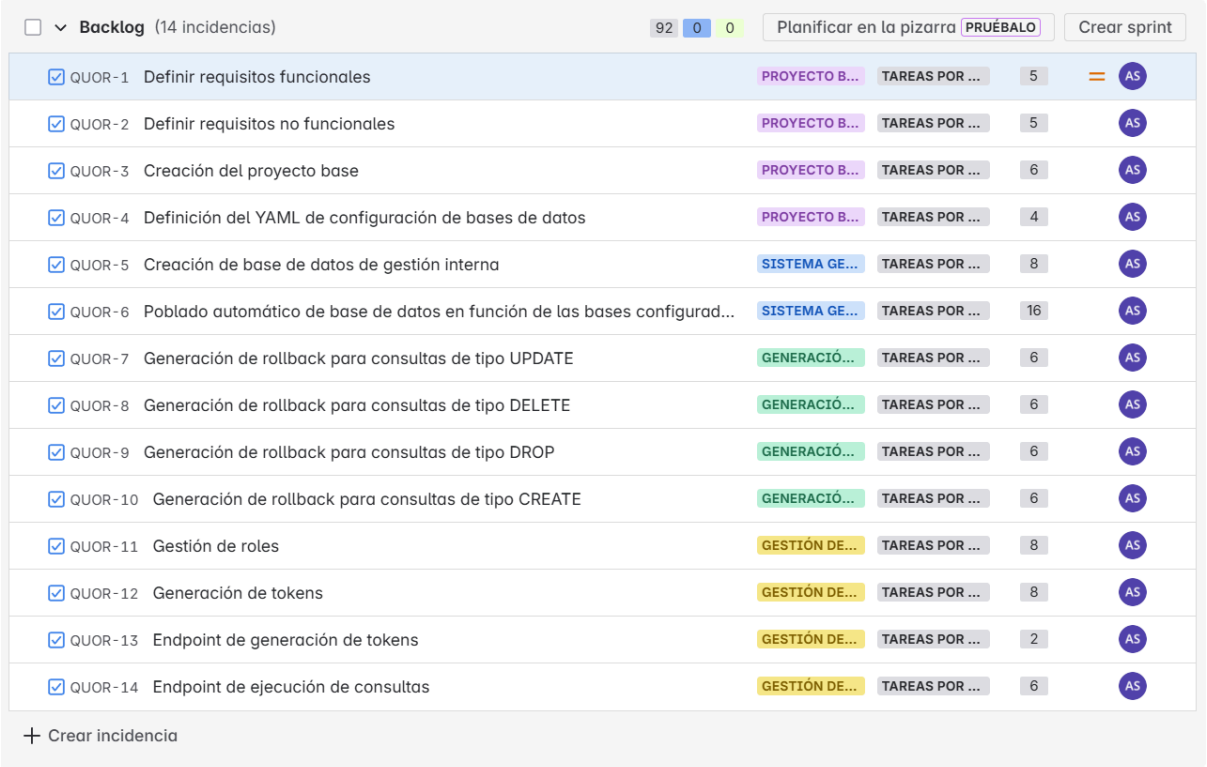
Ilustración 1. Tablero utilizado en Jira.



BACKLOG

El uso de un backlog en el proyecto es otra práctica muy común dentro de las metodologías ágiles, en este caso, el backlog total de las tareas pendientes al concebir por primera vez el proyecto constaba de 14 tareas distintas, que se segregan en Épicas y Sprints.

Ilustración 2. Backlog del proyecto en Jira.



SPRINTS

Una de las mayores aportaciones de la metodología Scrum al proyecto es el uso de “Sprints” como forma de separación de tareas en grupos de entregables.

Cada Sprint tiene una duración de dos semanas y la estimación de todas las tareas de un Sprint se encuentra en torno a los 24 SP (Story Points).

A continuación se presenta un resumen de todos los Sprints y las tareas que los componen:

1. Sprint 1: Proyecto base
    a. QUOR-1: Definición de requisitos funcionales. (5sp)
    b. QUOR-2: Definición de requisitos no funcionales. (5sp)
    c. QUOR-3: Creación del proyecto base. (6sp)
    d. QUOR-4: Definición del YAML de configuración de bases de datos. (4sp)

Ilustración 3. Tareas del Sprint 1 en Jira.

2. Sprint 2: Sistema gestor de relaciones.
    a. QUOR-5: Creación de base de datos de gestión interna. (8sp)
    b. QUOR-6: Poblado automático de base de datos en función de las bases configuradas. (16sp)

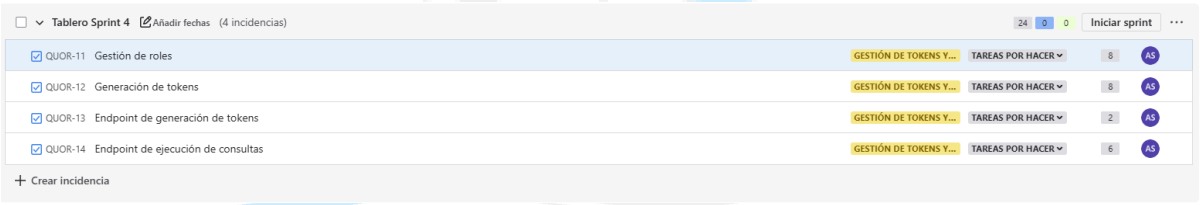
Ilustración 4: Tareas del Sprint 2 en Jira.

3. Sprint 3: Generación automática de consultas rollback.
    a. QUOR-7: Generación del rollback para consultas de tipo UPDATE. (6sp)
    b. QUOR-8: Generación del rollback para consultas de tipo DELETE. (6sp)
    c. QUOR-10: Generación del rollback para consultas de tipo INSERT. (6sp)

Ilustración 5: Tareas del Sprint 3 en Jira.

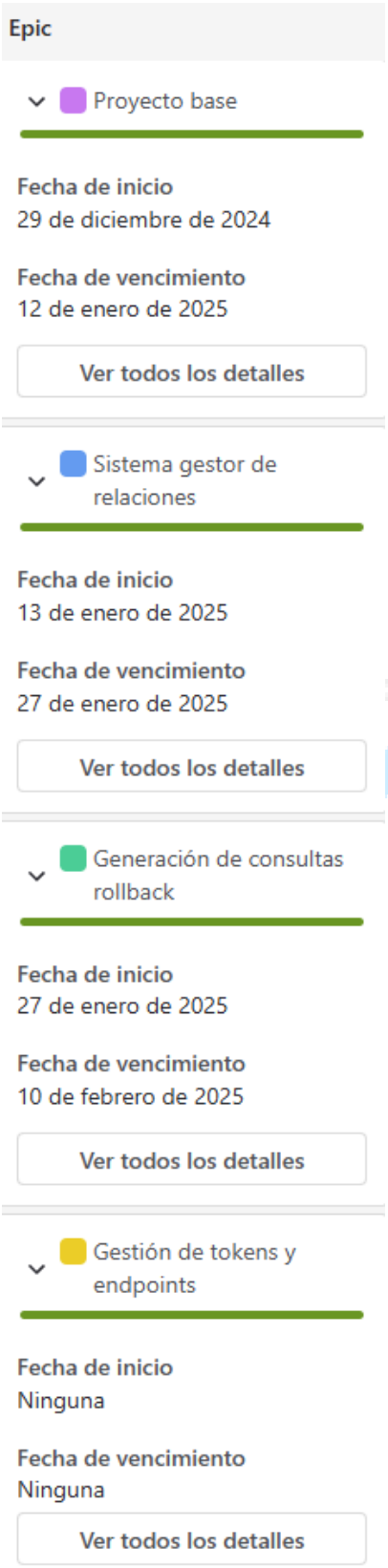
4. Sprint 4:Gestión de tokens y endpoints
    a. QUOR-11: Gestión de roles. (8sp)
    b. QUOR-12: Generación de tokens. (8sp)
    c. QUOR-13: Endpoint de generación de tokens. (2sp)
    d. QUOR-14: Endpoint de ejecución de consultas. (6sp)

Ilustración 6: Tareas del Sprint 4 en Jira.



ÉPICAS

Ilustración 7: Resumen de las Épicas en Jira.



Además de en Sprints el proyecto ha sido dividido en épicas, cada una de estas épicas marca el fin de una fase de desarrollo y a su vez el final de una funcionalidad.

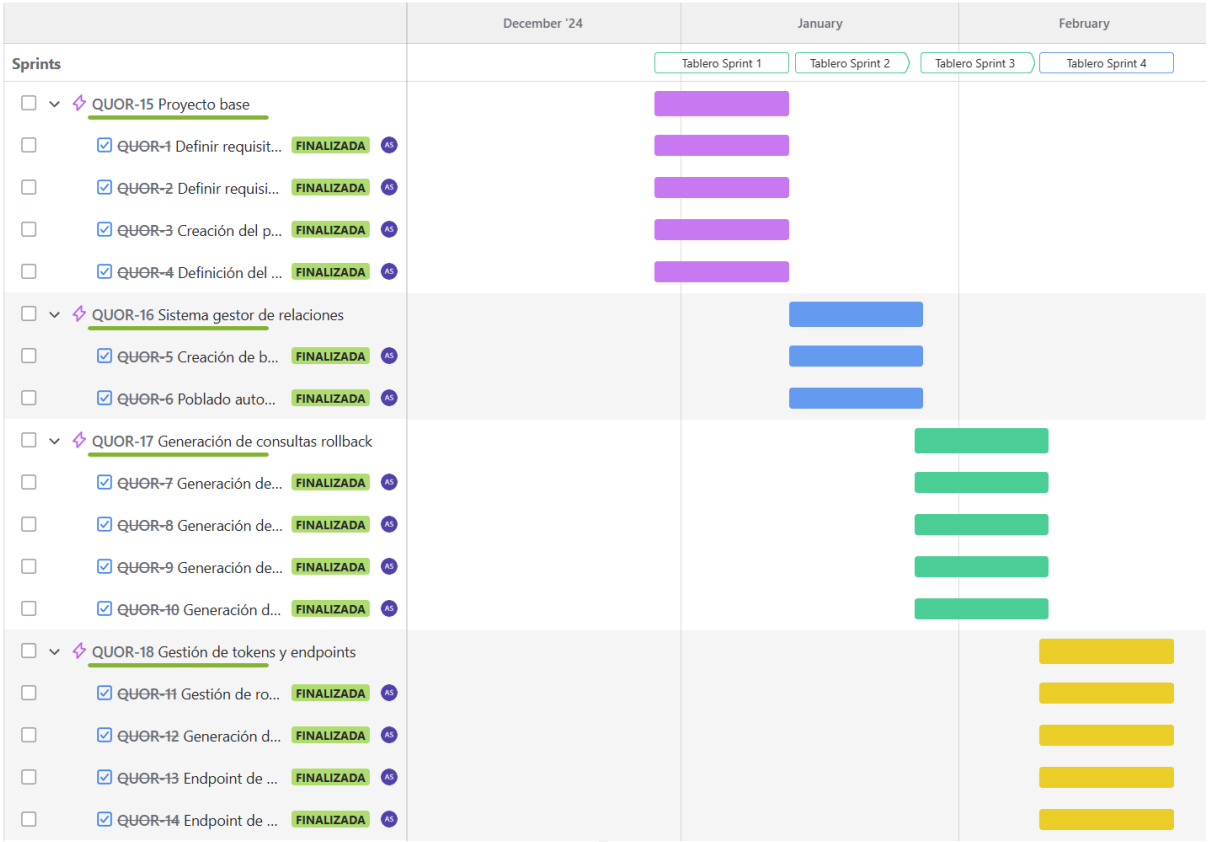
Las épicas en las que se ha dividido el proyecto son las siguientes:

- Proyecto base, esta épica contiene las tareas: QUOR-1, QUOR-2, QUOR-3 y QUOR-4.
- Sistema gestor de relaciones, esta épica contiene las tareas: QUOR-5 y QUOR-6.
- Generación de consultas rollback, esta épica contiene las tareas: QUOR-7, QUOR-8, QUOR-9 y QUOR-10.
- Gestión de tokens y endpoints, esta épica contiene las tareas: QUOR-11, QUOR-12, QUOR-13 y QUOR-14.

CRONOGRAMA

El siguiente diagrama de Gantt muestra el cronograma seguido durante el desarrollo de QueryOrchestrator segregado en tareas y Sprints:

Ilustración 8: Cronograma del proyecto en Jira.



GIT-FLOW

Para el control de versiones de QueryOrchestrator se ha utilizado Git-Flow, este modelo de ramificación para Git define una estructura para la gestión de las ramas durante el desarrollo y mantenimiento de los proyectos software. (9)

ESTRUCTURA DE GIT-FLOW

Git-Flow utiliza varios tipos de ramas, cada uno con un propósito diferente y cada rama cumpliendo una función específica durante el desarrollo.

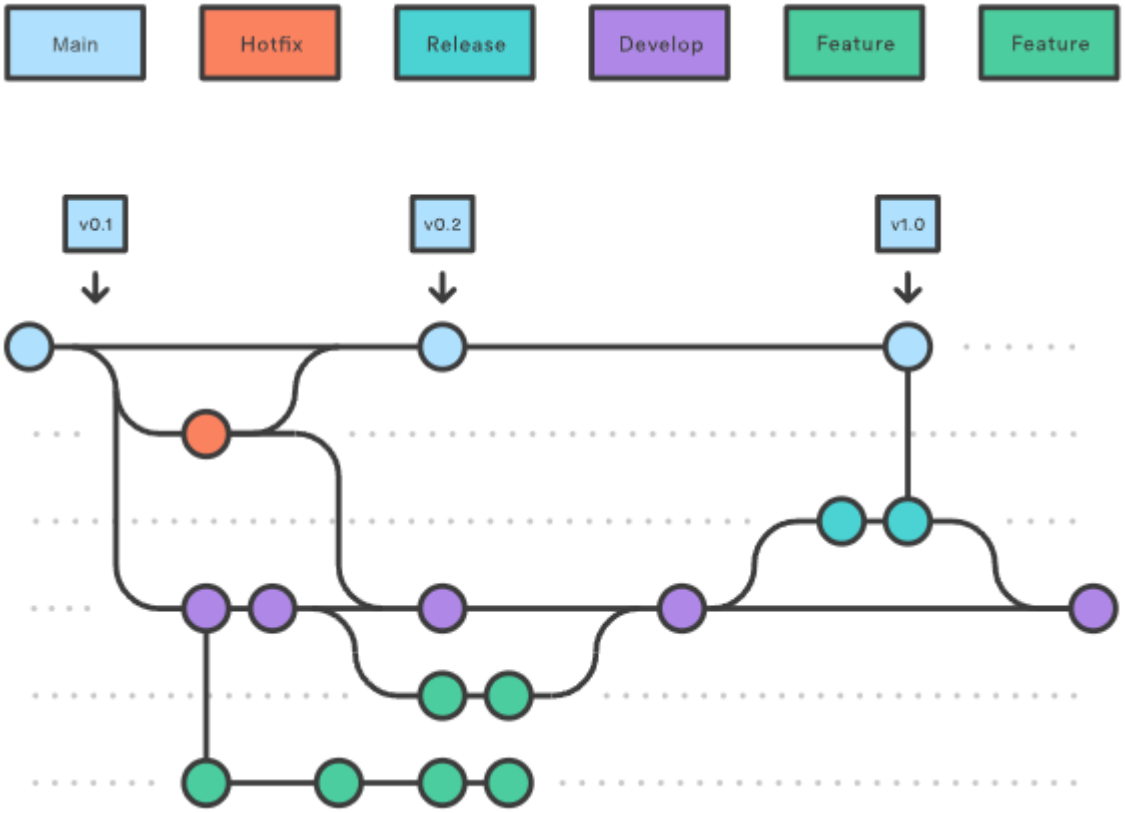
Ramas principales:

- main: La rama main contiene el historial de versiones definitivas del proyecto y su última versión constituye la versión productiva actual.
- development: esta rama es la rama de integración principal, necesaria para el desarrollo continuo. Las nuevas funcionalidades se integra primero a esta rama para ser validadas antes de pasar a la rama main.

Ramas de propósitos específicos:

- Ramas feature/: Se crean a partir de development y en estas ramas se desarrollan nuevas funcionalidades.
- Ramas release/: Estas ramas nacen de development cuando se alcanza un un conjunto de funcionalidades nuevas suficientes como para generar un nueva versión, estas ramas representan cada versión del software.
- Ramas hotfix/: Estas ramas se crean a partir de main para corregir problemas y “bugs” encontrados en la última versión productiva. Una vez resuelto el problema estas ramas se vuelven a integrar en main y en development.

Ilustración 9. Esquema de git-flow. (9)



En definitiva, las principales ventajas que aporta git-flow al desarrollo son: una estructura clara y bien definida del proyecto, el aislamiento de funcionalidades, el aislamiento de correcciones de errores, una gestión eficiente de “releases” y un historial del desarrollo del proyecto bien ordenado.

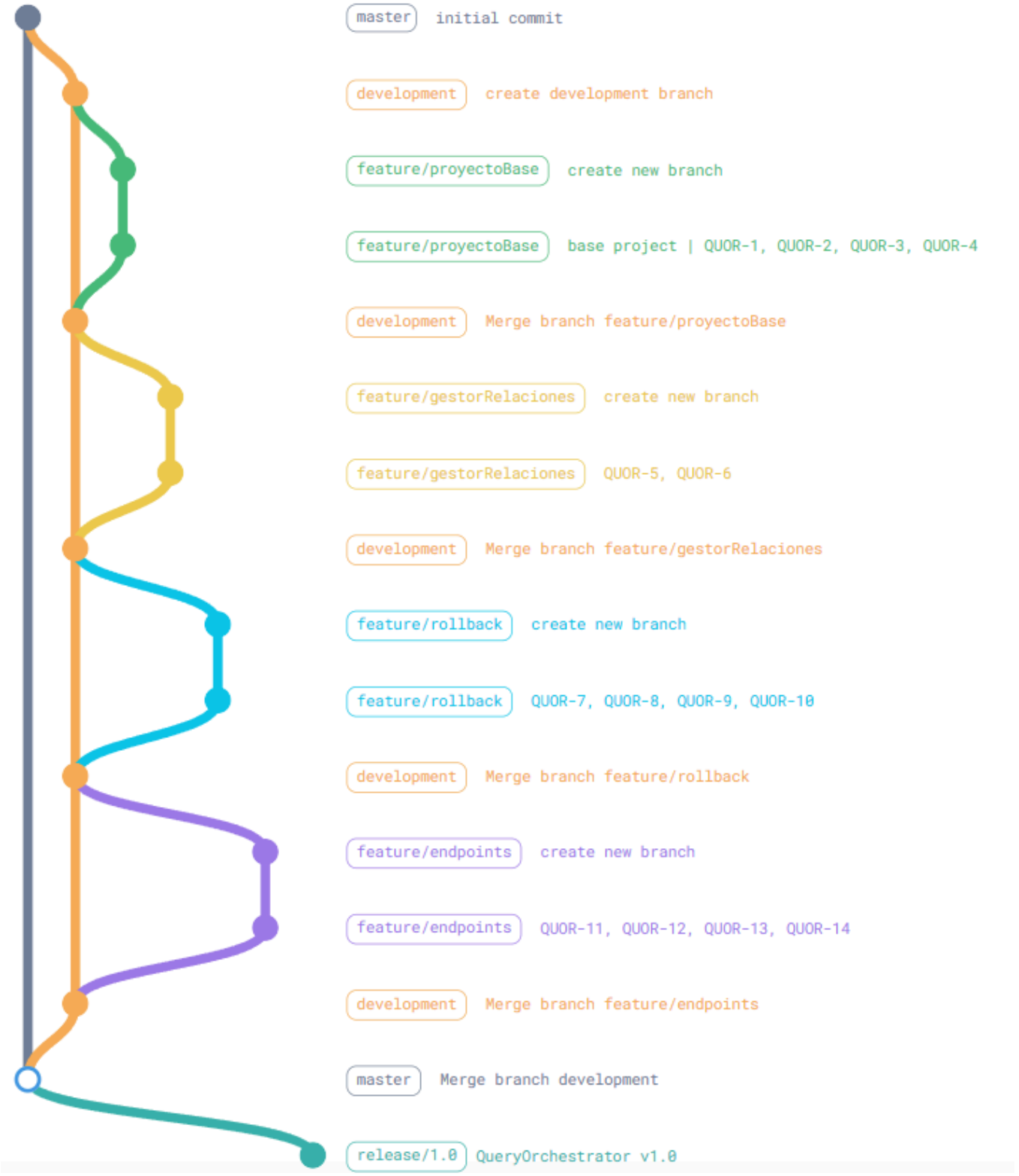
GIT-FLOW EN EL PROYECTO

El uso de esta metodología durante el desarrollo de QueryOrchestrator ha dado lugar las siguientes ramas:

- master: rama principal del proyecto.
- development: rama de integración del proyecto.
- feature/proyectoBase: En esta rama se ha incluido el desarrollo de la tareas QUOR-1, QUOR-2, QUOR-3 y QUOR-4.
- feature/gestorRelaciones: En esta rama se han desarrollado la tareas QUOR-5 y QUOR-6.
- feature/rollback: En esta rama se han desarrollado las tareas QUOR-7, QUOR-8, QUOR-9 y QUOR-10.
- feature/endpoints: En esta rama se han desarrollado las tareas QUOR-11, QUOR-12, QUOR-13 y QUOR-14.
- release/1.0: rama que constituye la primera versión del software.

Puesto que durante el desarrollo del proyecto solo había una persona a la vez trabajando, se ha optado por incluir varias funcionalidades en cada rama feature, agrupando estas ramas por épica en lugar de por tareas.

Ilustración 10. Git Graph del proyecto



TECNOLOGÍAS Y HERRAMIENTAS

Para el desarrollo y el posterior correcto funcionamiento de QueryOrchestrator es necesario el uso de ciertas herramientas y tecnologías que cumplen funciones de pruebas, desarrollo y funcionamiento.

Además del IDE que se ha utilizado para el desarrollo y el sistema de control de versiones (Github), cuya importancia en el proyecto ya ha sido detallada extensivamente anteriormente, se ha utilizado las siguientes herramientas:

- Python: lenguaje de programación utilizado en el desarrollo.
- Flask: Dependencia del backend que se encarga de manejar las solicitudes API.
- JWT: Sistema de producción y gestión de tokens en la aplicación.
- PostgreSQL: Sistema gestor de bases de datos utilizado tanto como base de datos principal del proyecto como para pruebas.
- DBeaber: Herramienta de interacción con PostgreSQL utilizada para pruebas y comprobaciones.
- Postman: Herramienta de gestión de peticiones API utilizada para las pruebas.

PYTHON

Python es un lenguaje de programación de alto nivel e interpretado que se estructura utilizando indentación para definir los bloques de código, orientado a facilitar la lectura y por lo tanto el mantenimiento del código.

El lenguaje es interpretado y dinámico, por lo que el código se ejecuta línea por línea sin necesidad de una compilación previa. Además Python tiene un enfoque multiparadigma que permite el desarrollo de código tanto orientado a objetos como procedimental.

Python también cuenta con un amplia biblioteca estándar y al ser interpretado y compatible con una gran variedad de sistemas operativos lo convierte en un lenguaje muy portable y escalable. (10)

Python en el proyecto

Entre los principales motivos por los cuales se ha escogido python para el desarrollo de QueryOrchestrator se encuentran:

- Compatibilidad con PostgreSQL y la mayoría de sistemas gestores de bases de datos.
- Portabilidad y compatibilidad con una gran variedad de sistemas operativos que amplía las opciones de despliegue y casos de uso del proyecto.
- Lenguaje interpretado, lo que facilita las pruebas y la depuración del código.
- Lenguaje no tipado, lo que facilita el procesamiento de datos y la ejecución de consultas.

FLASK

Flask es un microframework web escrito en python, es simple y flexible a la par que potente. Es un sistema ligero, que permite a los desarrolladores añadir solo las funcionalidades necesarias mediante extensiones.

También cuenta con un sistema de enrutamiento sencillo que vincula de forma directa las URLs con las funciones correspondientes, utilizando Jinja2 para separar la lógica de la presentación. Flask incluye además un servidor de desarrollo y depurador integrado, lo que facilita la fase de pruebas durante el desarrollo.

Flask en el proyecto

Durante el desarrollo de QueryOrchestrator, Flask se ha utilizado para definir las rutas de API REST como medio principal para interactuar con el software. Se ha escogido Flask para esta tarea por su compatibilidad con Python, simplicidad y facilidad de integración.

JSON WEB TOKEN

JWT es un estándar abierto utilizado para la transmisión de información de forma segura en formato JSON en forma de token (12).

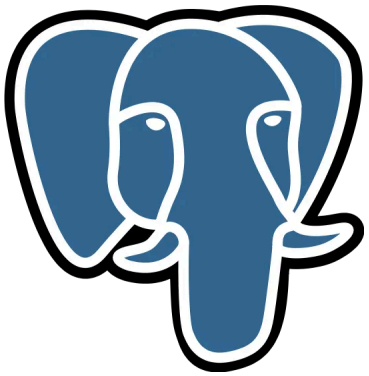
Un token JWT consta de tres partes codificadas en base64:

1. Header: indica el tipo de token y el algoritmo de firma.
2. Payload: contiene los datos que se desean transmitir.
3. Signature: garantiza que el contenido no ha sido alterado con una clave secreta.

JWT en el proyecto

El papel que desempeña este estándar en el proyecto es el de garantizar la seguridad de los endpoints a través de los cuales los usuarios interactúan con las bases de datos a gestionar. Además se ha hecho uso del formato JSON de los token utilizados en la autenticación para crear un sistemas de roles a través del cual se pueden identificar permisos diferentes para los usuarios permitiendo de esta forma realizar unas acciones u otras.

POSTGRESQL



PostgreSQL es un sistema gestor de bases de datos relacional y de código abierto. Este sistema cumple con los estándares SQL y es utilizado en multitud de aplicaciones. PostgreSQL destaca por su rendimiento, robustez y escalabilidad (13).

PostgreSQL en el proyecto

Este sistema gestor de bases de datos ha sido elegido para su uso en el desarrollo de QueryOrchestrator por las siguientes razones:

- Soporte completo de ACID.
- Gran capacidad de escalado.
- Soporte nativo para objetos complejos.
- Compatibilidad con Python.

DBEAVER

DBeaver es una herramienta de administración de bases de datos multiplataforma y de código abierto. La aplicación permite a los desarrolladores establecer conexiones con múltiples bases de datos y permite la ejecución de consultas y visualización de datos, sin limitar la experiencia a un solo sistema gestor (14).

DBeaver en el proyecto

En el desarrollo de QueryOrchestrator DBeaver ha demostrado ser un herramienta de gran utilidad al permitir hacer pruebas y comprobaciones sobre varias bases de datos gestionadas desde la aplicación de forma simultánea.

POSTMAN

Postman es una herramienta de desarrollo de APIs que permite probar, documentar y automatizar peticiones HTTP mediante una interfaz gráfica (15).

Postman en el proyecto

Postman es otra herramienta que ha resultado de gran utilidad durante el desarrollo ya que permite de un forma sencilla validar las peticiones HTTP que se realizan a QueryOrchestrator, de una forma repetible y predecible puesto que la herramienta tiene una función de archivado de peticiones.

REQUISITOS Y DIAGRAMAS

En este apartado se muestran, explican y discuten los requisitos y diagramas que conciben el desarrollo del proyecto, así como el efecto que estos han tenido en el desarrollo.

A pesar de que el proyecto haya sido gestionado mediante metodologías ágiles, puesto que los requisitos iniciales del proyecto no iban a ser modificados una vez propuesto al no existir la figura del cliente durante el desarrollo, se ha optado por una definición de requisitos clásica, funcionales y no funcionales, de las cuales se han derivados las tareas posteriormente.

REQUISITOS

REQUISITOS FUNCIONALES

Gestión y Configuración de Clusters

- **RF1.1:** Permitir definir grupos (clusters) de bases de datos PostgreSQL mediante un archivo de configuración en formato YAML.
- **RF1.2:** Cada cluster debe tener una identificación única y contener la lista de bases de datos que lo integran.

Balanceo de Operaciones

- **RF2.1:** Ejecutar de forma simultánea cualquier cambio (insert, update, delete, create, drop) en todas las bases de datos que conforman un cluster.
- **RF2.2:** Garantizar que la operación se aplique de manera consistente en cada una de las bases configuradas en el cluster.

Interfaz API para Consultas

- **RF3.1:** Exponer una API a través de la cual se puedan enviar consultas y comandos al balanceador.
- **RF3.2:** Permitir a los usuarios seleccionar, mediante la API, a qué cluster de bases de datos se dirigirá la consulta.

Seguridad y Autorización

- **RF4.1:** Requerir un token JWT en cada solicitud a la API para validar la identidad y permisos del usuario.
- **RF4.2:** Validar que el token JWT otorgue los permisos necesarios para el tipo de consulta u operación solicitada en el cluster especificado.

Registro y Auditoría de Consultas

- **RF5.1:** Registrar cada consulta ejecutada.
- **RF5.2:** Almacenar estos registros en una base de datos interna para facilitar auditorías y seguimiento de operaciones.

Gestión de Rollbacks

- **RF6.1:** Para cada operación de modificación de datos (update, delete, insert), generar automáticamente una consulta de rollback que permita revertir los cambios.
- **RF6.2:** Almacenar las consultas de rollback en la base de datos interna junto con la consulta original.
- **RF6.3:** Permitir, vía API, ejecutar un rollback de una consulta determinada.
- **RF6.4:** Utilizar la información de las relaciones entre tablas (obtenida durante el escaneo de las bases) para asegurar que el rollback se ejecute sin errores y mantenga la integridad referencial.

Exploración y Actualización de Metadatos

- **RF7.1:** Escanear las bases de datos configuradas en cada cluster para extraer y actualizar metadatos: esquemas, tablas y relaciones.
- **RF7.2:** Almacenar esta información en una base de datos interna que sirva como repositorio de la estructura y relaciones de los datos.

REQUISITOS NO FUNCIONALES

Rendimiento

- **RNF1.1:** El sistema debe procesar múltiples consultas concurrentes con baja latencia.

Escalabilidad

- **RNF2.1:** El sistema debe ser escalable para soportar el incremento en la cantidad de clusters y volumen de consultas.
- **RNF2.2:** Permitir la adición de nuevas bases de datos y clusters sin necesidad de reestructuraciones mayores.

Disponibilidad y Fiabilidad

- **RNF3.1:** Garantizar alta disponibilidad, minimizando tiempos de inactividad incluso durante operaciones de actualización de configuración o mantenimiento.
- **RNF3.2:** Implementar mecanismos de tolerancia a fallos y recuperación en caso de error en alguna de las bases de datos o en la ejecución de operaciones de balanceo y rollback.
- **RNF3.3:** Asegurar la integridad y consistencia de los datos a través de todas las operaciones distribuidas en los clusters.

Seguridad

- **RNF4.1:** Utilizar prácticas de seguridad robustas en la gestión de tokens JWT y en la comunicación a través de la API (por ejemplo, uso de HTTPS).
- **RNF4.2:** Proteger la información sensible y los registros de operaciones contra accesos no autorizados.

Usabilidad y Configuración

- **RNF5.1:** El formato YAML de configuración debe ser intuitivo y fácil de modificar, permitiendo a los administradores realizar cambios sin necesidad de conocimientos profundos en programación.

Portabilidad y Despliegue

- **RNF6.1:** El sistema, desarrollado en Python, debe ser compatible con las principales versiones y entornos (Linux, Windows, contenedores Docker).

Auditoría y Trazabilidad

- **RNF7.1:** Los registros de todas las operaciones (consultas, cambios, rollbacks) deben ser almacenados de forma que permitan su posterior auditoría.

Extensibilidad

- **RNF8.1:** Diseñar el sistema de manera que permita la incorporación de nuevas funciones o la extensión a otros tipos de bases de datos en el futuro.
- **RNF8.2:** Asegurar que la arquitectura facilite la integración con otros sistemas o servicios, mediante API u otros mecanismos de comunicación.

RELACIÓN ENTRE TAREAS Y REQUISITOS

Las épicas y las tareas del backlog definidas anteriormente en el apartado de “Scrumban” están directamente relacionadas con los requisitos funcionales y no funcionales descritos.

A continuación, se explica a partir de qué requisitos se han extraído cada una de las tareas puesto que la correlación entre requisitos y tareas no es de uno a uno. A excepción de las tareas QUOR-1 y QUOR-2 que son la definición de requisitos funcionales y no funcionales, dos tareas necesarias antes de comenzar el desarrollo y de las cuales parten los requisitos y por lo tanto el resto de tareas definidas en el proyecto.

Épica 1: Proyecto base

- QUOR-3: Creación del proyecto base.
    - RNF6.1: Durante este paso se consolida la base del proyecto en el lenguaje seleccionado (Python) y se asegura la compatibilidad con diferentes entornos.
    - RNF8.1: La concepción inicial del proyecto es tal que permite la agregación de otros tipos de bases de datos en un futuro.

- QUOR-4: Definición del YAML de configuración de base de datos.
    - RF1.1: La definición del YAML de configuración permite definir grupos dentro del mismo.
    - RF1.2: La definición del YAML de configuración permite exponer una lista de bases de datos pertenecientes a un mismo cluster identificable.
    - RNF2.2: La definición del YAML de configuración permite agregar un número indefinido de clusters de bases de datos.
    - RNF5.1: La definición del YAML de configuración está pensada para ser intuitivo y fácil de modificar, evitando configuraciones innecesarias y reduciendo el número de elementos a configurar al mínimo sin perder funcionalidad.

Épica 2: Proyecto base

- QUOR-5: Creación de base de datos de gestión interna.
    - RF5.2: La base de datos debe contener tablas de registro de operaciones.
    - RF6.2: La base de datos debe almacenar los rollbacks generados por las consultas.
    - RF7.2: La base de datos debe tener una estructura cual que permita almacenar la estructura y las relaciones de las bases de datos que se gestionan.
- QUOR-6: Poblado automático de base de datos en función de las bases configuradas.
    - RF7.1: El sistema escanea las bases de datos configuradas y obtiene sus metadatos.
    - RF7.2: Los metadatos obtenidos se almacenan en la base de datos de gestión interna.

Épica 3: Generación de consultas rollback

- QUOR-7: Generación del rollback para consultas de tipo UPDATE.
    - RF6.1: Se genera una consulta de rollback para las operaciones de tipo UPDATE.
    - RF6.4: Se generan las consultas de rollback de tipo UPDATE haciendo uso de los metadatos de las bases de datos.

- QUOR-8: Generación del rollback para consultas de tipo DELETE.
    - RF6.1: Se genera una consulta de rollback para las operaciones de tipo DELETE.
    - RF6.4: Se generan las consultas de rollback de tipo DELETE haciendo uso de los metadatos de las bases de datos.

- QUOR-10: Generación del rollback para consultas de tipo INSERT.
    - RF6.1: Se genera una consulta de rollback para las operaciones de tipo INSERT.
    - RF6.4: Se generan las consultas de rollback de tipo INSERT haciendo uso de los metadatos de las bases de datos.

Épica 4: Gestión de token y endpoints

- QUOR-11: Gestión de roles.
    - RF4.1: La definición de los roles permite la identificación de los permisos de los usuarios.
    - RF4.2: Los permisos se encuentran embebidos en un token JWT para la autenticación.

- QUOR-12: Generación de tokens.
    - RF4.1: Se requiere un token JWT para validar los permisos.
    - RF4.2: Se requiere un método de validación de token JWT con sus permisos.
    - RNF4.1: Prácticas de seguridad robustas en la gestión de tokens.

- QUOR-13: Endpoint de generación de tokens.
    - RF4.1: Endpoint de generación de tokens necesario para la posterior autenticación de los usuarios.
    - RNF4.1: Medidas de seguridad en la implementación del endpoint de generación de tokens JWT.

- QUOR-14: Endpoint de ejecución de consultas.
    - RF2.1: Ejecutar de forma simultánea las consultas de en un cluster.
    - RF2.2: Se garantiza que las consultas se hayan aplicado con éxito.
    - RF3.1: Se expone un endpoint mediante el cual se permita la ejecución de consultas.
    - RF3.2: El endpoint de ejecución de consultas cuenta con herramientas que permiten seleccionar el cluster sobre el que se realizará la consulta.
    - RF5.1: Se registran las consultas ejecutadas.
    - RF6.2: Se almacena el rollback en el registro de auditoría.
    - RF6.3: Se permite la ejecución de consultas de rollback vía API.

DIAGRAMAS

DIAGRAMA DE FLUJO

Ilustración 11. Diagrama de flujo de ejecución de consulta INSERT.

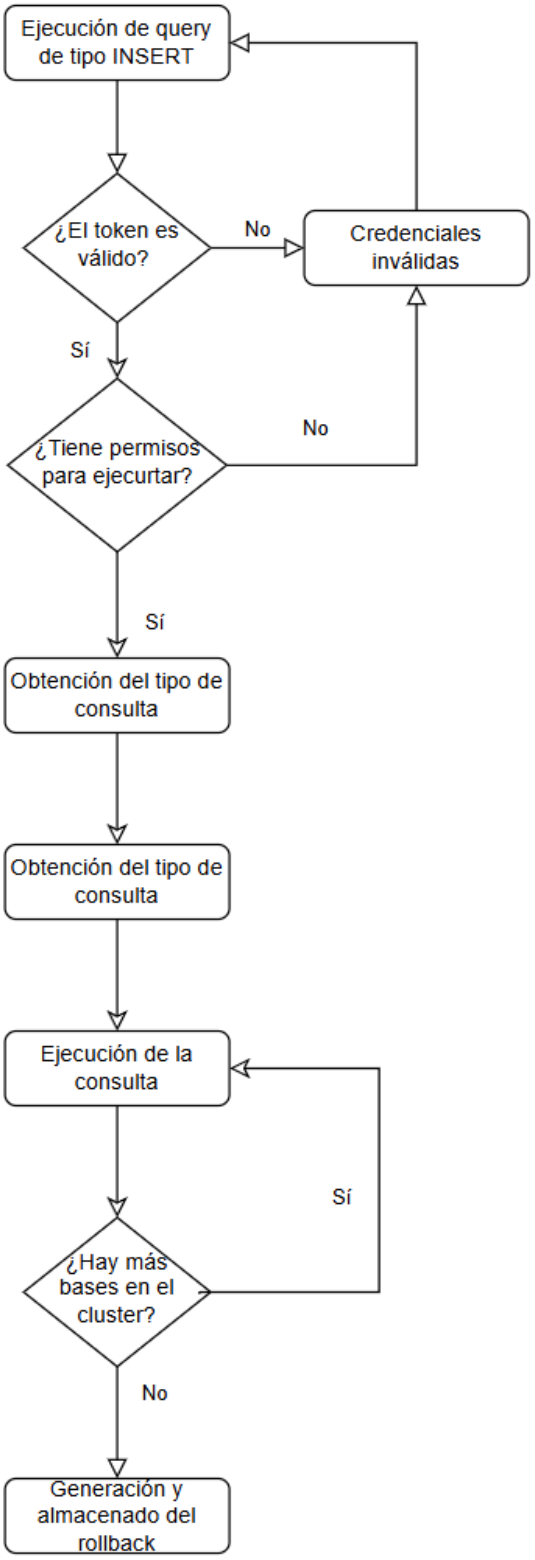


DIAGRAMA DE SECUENCIA

Ilustración 12. Diagrama de secuencia de ejecución de consulta INSERT.

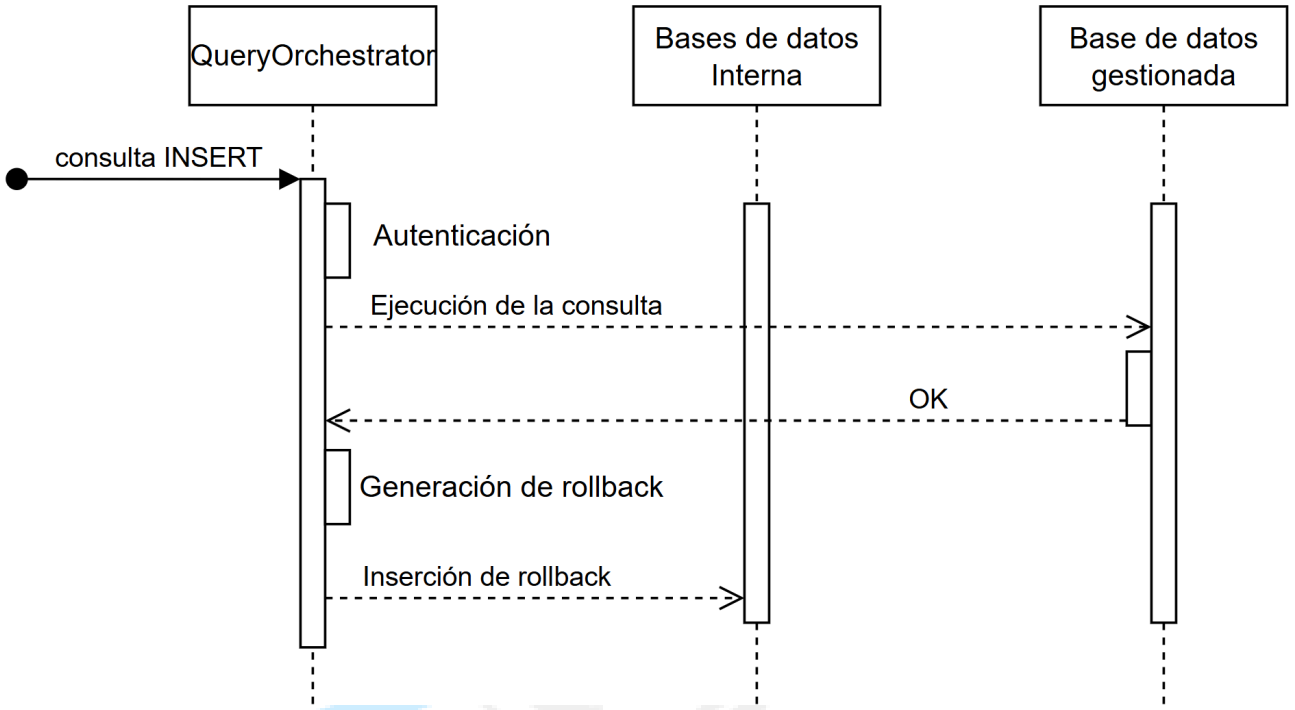
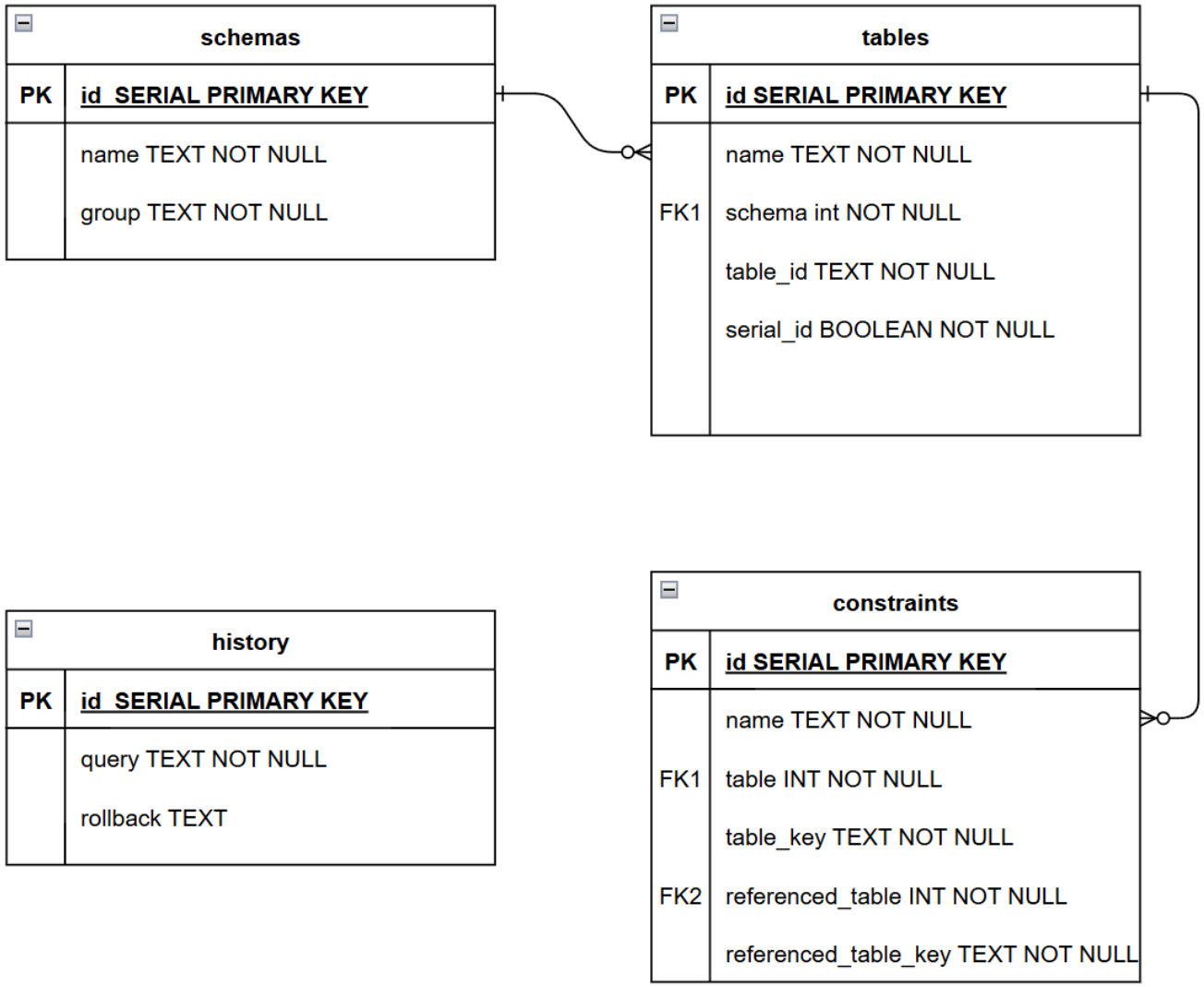


DIAGRAMA ENTIDAD-RELACIÓN

Ilustración 13. Diagrama entidad-relación de la base de datos de gestión interna.



RESUMEN DE SPRINTS

SPRINT 1

El sprint 1 constituye el inicio del proyecto y está enfocado en completar la la Épica de “Proyecto base”, cuyo objetivo fundamental es sentar las bases de desarrollo sobre las cuales se va construir el software.

El primer sprint tiene una duración de dos semanas y durante ese tiempo se abordan las siguientes tareas:

- QUOR-1: Definición de requisitos funcionales. Esta tarea trata de redactar una lista de requisitos que consolidan y segregan la idea del proyecto en partes más manejables. Sobre el resultado de esta tarea se podrán obtener el resto de tareas del proyecto. (5SP)

- QUOR-2: Definición de requisitos no funcionales. Esta tarea tiene el mismo propósito que la tarea anterior y consiste en redactar una lista de requisitos no funcionales sobre los que fundamentar el proyecto. (5SP)

- QUOR-3: Creación del proyecto base. El objetivo de esta tarea es el de la creación de la estructura para el proyecto, configuración del entorno de trabajo, instalación de dependencias y la configuración inicial del control de versiones. (6SP)

- QUOR-4: Definición del YAML de configuración de bases de datos. El principal objetivo a abordar con esta tarea es el de definir la estructura que tendrá el fichero de configuración del proyecto. De forma que sea legible y sencillo de montar y editar. Principalmente el fichero de configuración debe contener información sobre la base de datos del proyecto y todas las demás bases de datos que gestiona, por lo que por cada una de las bases de datos es necesaria alguna información básica, así como la configuración para la conexión. (4SP)

SPRINT 2

El sprint 2 aborda las tareas del proyecto necesarias para que QueryOrchestrator funciona con una base de datos en la que se almacenan los registros de auditoría de las consultas que realizan los usuarios además de los datos de interés de las bases de datos que se gestionan con el programa. Estas tareas están recogidas en la Épica de “Sistema gestor de relaciones”.

El segundo sprint tiene también una duración de dos semanas y consta de las siguientes tareas:

- QUOR-5: Creación de la base de datos de gestión interna. Esta base de datos consta de dos partes principales, una dinámica con los registros de auditoría de las consultas que se realizan y otra estática, que se crea cada vez que se ejecuta el proyecto y contiene información sobre las bases de datos que se gestionan, sus esquemas y sus tablas. El objetivo de la tarea es establecer la estructura y proporcionar las consultas de creación de esta base de datos. (8SP)

- QUOR-6: Poblado automático de base de datos en función de las bases de datos configuradas. Puesto que QueryOrchestrator necesita información sobre las bases de datos a gestionar y esta información varía en función de las conexiones configuradas. Es necesario un sistema que obtenga de manera automática información relevante sobre las bases de datos y la centralice en la base de datos gestión interna. Esta tarea por tanto tiene como objetivo definir las consultas necesarias para obtener esa información y desarrollar una lógica que a partir del fichero de configuración obtenga los datos necesarios de las conexiones definidas y los afiance. (16SP)

SPRINT 3

El sprint 3 tiene como objetivo desarrollar la lógica de una de los elementos principales del proyecto, la Épica de “Generación automática de consultas rollback”.

El tercer sprint tiene una duración de tres semanas y está compuesto por las siguientes tareas:

- QUOR-7: Generación del rollback para consultas de tipo UPDATE. Esta tarea tiene como objetivo desarrollar un método que detecte si la consulta que se le ofrece es de tipo UPDATE y si es es el caso, generar una consulta de tipo UPDATE que deshaga el cambio en caso de que el usuario lo desee.

- QUOR-8: Generación del rollback para consultas de tipo DELETE. Esta tarea tiene como objetivo desarrollar un método que detecte si la consulta que se le ofrece es de tipo DELETE y si es es el caso, generar una consulta de tipo INSERT que deshaga el cambio en caso de que el usuario lo desee.

- QUOR-7: Generación del rollback para consultas de tipo INSERT. Esta tarea tiene como objetivo desarrollar un método que detecte si la consulta que se le ofrece es de tipo INSERT y si es es el caso, generar una consulta de tipo DELETE que deshaga el cambio en caso de que el usuario lo desee.

SPRINT 4

El sprint 4 es el último y tiene como objetivo cerrar el proyecto cohesionando y encapsulando toda la funcionalidad desarrollada en un paquete seguro, accesible vía API. Esta funcionalidad está recogida dentro de la Épica de “Gestión de tokens y endpoints”.

El cuarto sprint tiene también una duración de dos semanas y consta de las siguientes tareas:

- QUOR-11: Gestión de roles. Esta tarea está definida con el objetivo de encontrar una solución de permisos en la aplicación que permita controlar los accesos y la ejecución de consultas a las bases de datos gestionadas, todo en una paquete simple y con capacidad de ser embebido en un token JWT como medida de seguridad. (8SP)

- QUOR-12: Generación de tokens. El objetivo de esta tarea es el de crear un sistema de generación y decodificación de tokens JWT mediante el cual se pueden comprobar los permisos definidos anteriormente. (8SP)

- QUOR-13: Endpoint de generación de tokens. Esta tarea se plantea con el objetivo de desarrollar un endpoint mediante el cual se pueda acceder a la lógica de la tarea QUOR-12. (2SP)

- QUOR-14: Endpoint de ejecución de consultas. El objetivo de esta tarea es el de proporcionar un endpoint mediante el cual, los usuarios con permisos puedan ejecutar consultas sobre los cluster de bases de datos. Además esta tarea también combina los métodos de generación de rollback y de inserción en la base de datos de gestión interna para dejar registro de auditoría y la posibilidad de deshacer los cambios realizados por la consulta. (6SP)

BURNDOWN CHARTS

A continuación se muestran las gráficas de *Burndown* de los *Story Points* (SP) de cada sprint. Además de una gráfica de todo el proyecto combinado.

Ilustración 14: Burndown chart del sprint 1.

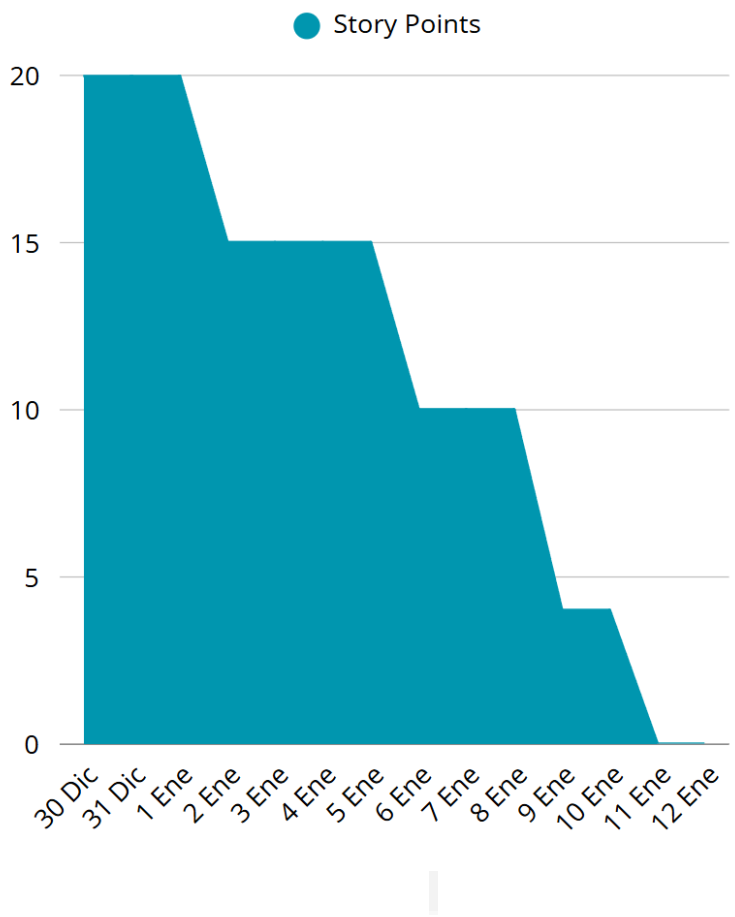


Ilustración 15: Burndown chart del sprint 2.

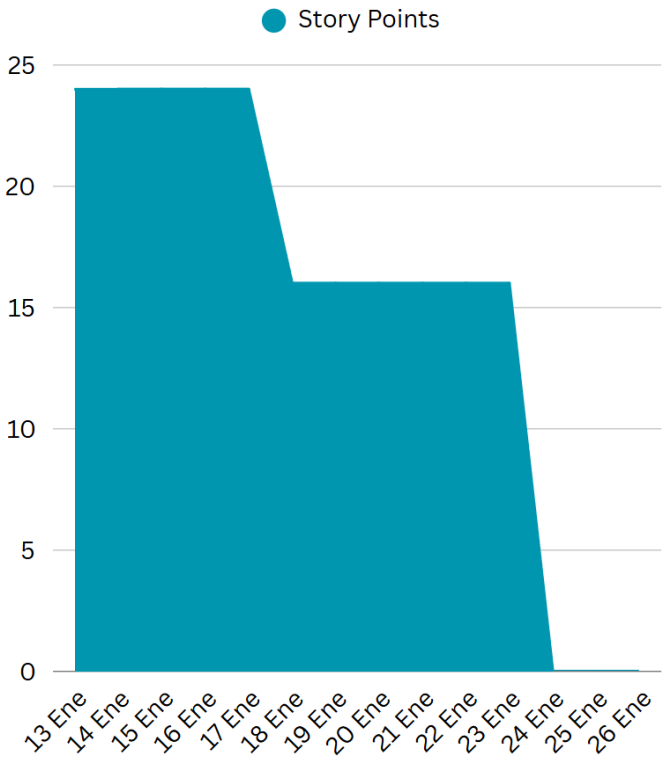


Ilustración 16: Burndown chart del sprint 3.

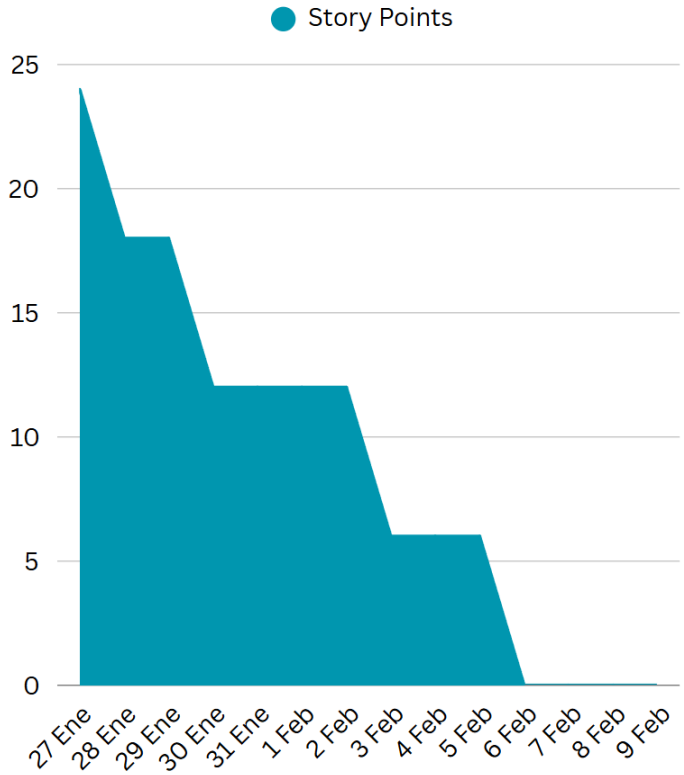


Ilustración 17: Burndown chart del sprint 4.

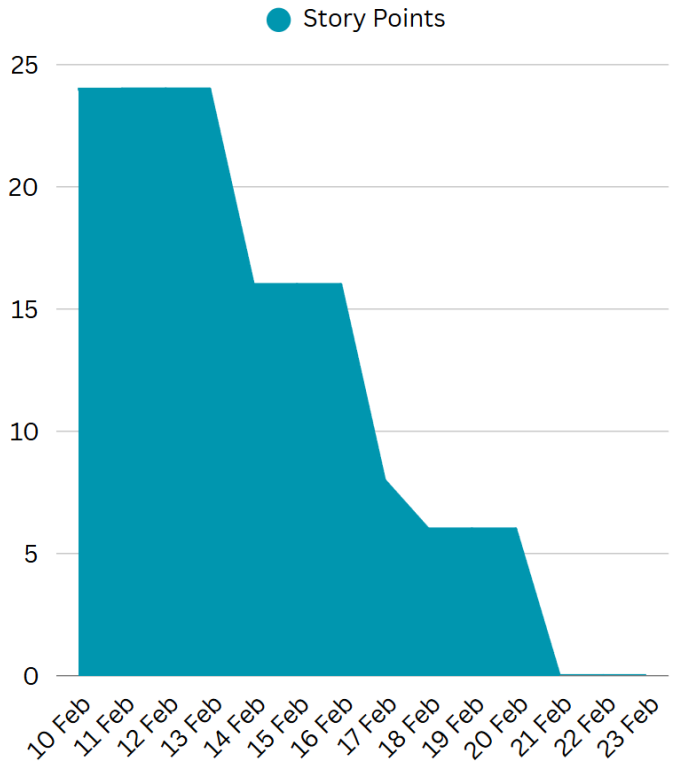
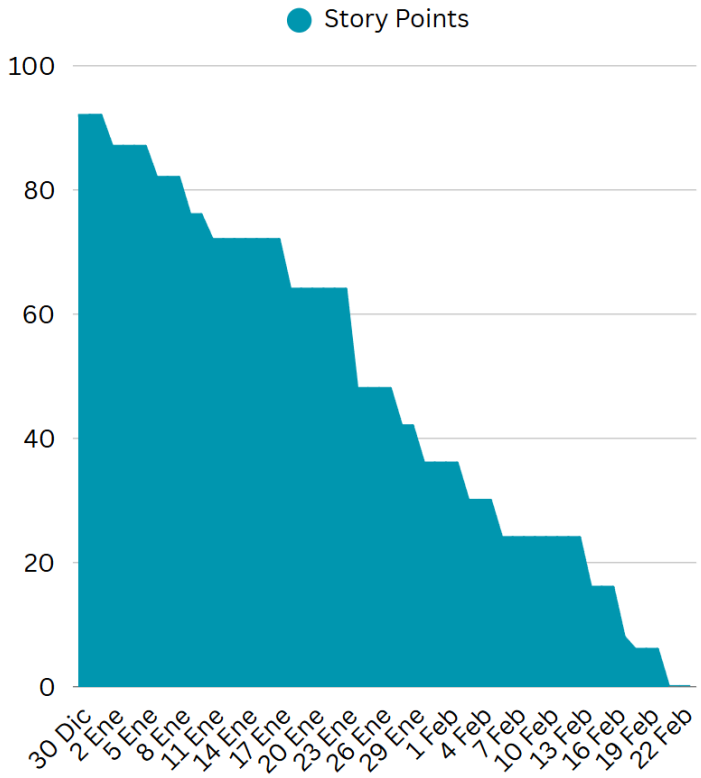


Ilustración 18: Burndown chart del proyecto.



PROYECTO

En el siguiente apartado se exploran los datos de interés relativos al funcionamiento del programa, la funcionalidad, los casos de uso, el modo de operación, endpoints y diagramas de interés.

A grandes rasgos, QueryOrchestrator es un sistema de balanceo y gestión de bases de datos, planteado para integrarse con otras aplicaciones y ser usado como punto de acceso para la conexión con bases de datos. El sistema opera mediante la ejecución de consultas vía API para una mejor integración con todo tipo de sistemas y puede ser configurado para que las consultas se ejecuten en varias bases de datos al mismo tiempo, obteniendo de esa forma redundancia de una forma sencilla.

FUNCIONALIDAD

QueryOrchestrator es un módulo responsable de la coordinación y ejecución de operaciones SQL distribuidas sobre los clusters de bases de datos configurados. Actúa como el núcleo lógico del sistema, gestionando tanto la ejecución de queries como la generación de mecanismos de reversión y auditoría.

Funcionalidad principal

- Ejecución de queries: A partir de una solicitud API, QueryOrchestrator identifica el cluster de destino y ejecuta la consulta SQL proporcionada en todas las instancias de base de datos registradas en dicho cluster.
- Generación automática de rollback: Para cada operación de tipo INSERT, UPDATE, o DELETE, el sistema genera dinámicamente una consulta SQL de rollback que permite revertir los cambios realizados.
- Persistencia para auditoría: Tanto la consulta original como la de rollback se almacenan en una base de datos interna de auditoría.

Casos de uso

- Cambios transaccionales distribuidos: Aplicar modificaciones consistentes sobre múltiples réplicas o nodos de base de datos en un entorno distribuido.
- Sincronización de datos en entornos.
- Auditoría completa de operaciones SQL con capacidad de reversión.
- Automatización de procesos de despliegue o migración de datos con rollback seguro.

FUNCIONAMIENTO

CONFIGURACIÓN

A continuación se explica el modelo que se sigue para configurar el software.

La configuración de QueryOrchestrator se encuentra en el fichero “config.yml” del proyecto y en este se encuentran los datos necesarios para el funcionamiento del programa, como la conexión a la base de datos interna, la clave para la generación y decodificación de tokens y también las bases de datos que se gestionan.

La estructura de la configuración del proyecto es la siguiente:

Ilustración 19: Fichero de configuración.

```yaml
---
secret: jwt_secret
native_database:
  dbname: "dbname"
  user: "user"
  password: "password"
  host: "host"
  port: 0000
groups:
  - - schema: schema1
      dbname: dbname1
      user: user1
      password: password1
      host: host1
      port: 0000
      name: name1
    - schema: schema2
      dbname: dbname2
      user: user2
      password: password2
      host: host2
      port: 0000
      name: name1
  - - schema: schema3
      dbname: dbname3
      user: user3
      password: password3
      host: host3
      port: 0000
      name: name3
```

Estos datos se pueden segregar en tres grupos principales.

Secret

```
secret: jwt_secret
```

El valor “secret” es la clave utilizada para la generación y decodificación de los token JWT utilizados para la autenticación.

Native Database

```
native_database:
  dbname: "dbname"
  user: "user"
  password: "password"
  host: "host"
  port: 0000
```

El siguiente fragmento de configuración “native_database” se utiliza para establecer la conexión con la base de datos de gestión interna de la aplicación, donde se recogen los registros de auditoría y los datos de interés de las bases de datos configuradas.

La variable “dbname” corresponde con el nombre de la base de datos.
La variable “user” corresponde con el usuario de acceso a la base de datos.
La variable “password” corresponde con la contraseña de acceso a la base de datos.
La variable “host” corresponde con la dirección IP o el DNS de la conexión.
La variable “port” corresponde con el puerto de conexión de la base de datos.

Groups

```
groups:
  - - schema: schema1
      dbname: dbname1
      user: user1
      password: password1
      host: host1
      port: 0000
      name: name1
    - schema: schema2
      dbname: dbname2
      user: user2
      password: password2
      host: host2
      port: 0000
      name: name1
  - - schema: schema3
      dbname: dbname3
      user: user3
      password: password3
      host: host3
      port: 0000
      name: name3
```

El fragmento de grupos es en el que se describen y configuran las conexiones las las bases de datos a gestionar.

En el ejemplo, sustituyendo cada conexión a base de datos por un nombre, se puede observar que están configuradas tres bases de datos en listas anidadas de esta forma:

groups: [[DB1,DB2],[DB3]]

Esto indica que hay dos clusters o grupos de bases de datos configuradas, [BD1,DB2] y [DB3], el primero con dos bases de datos y el segundo con una.

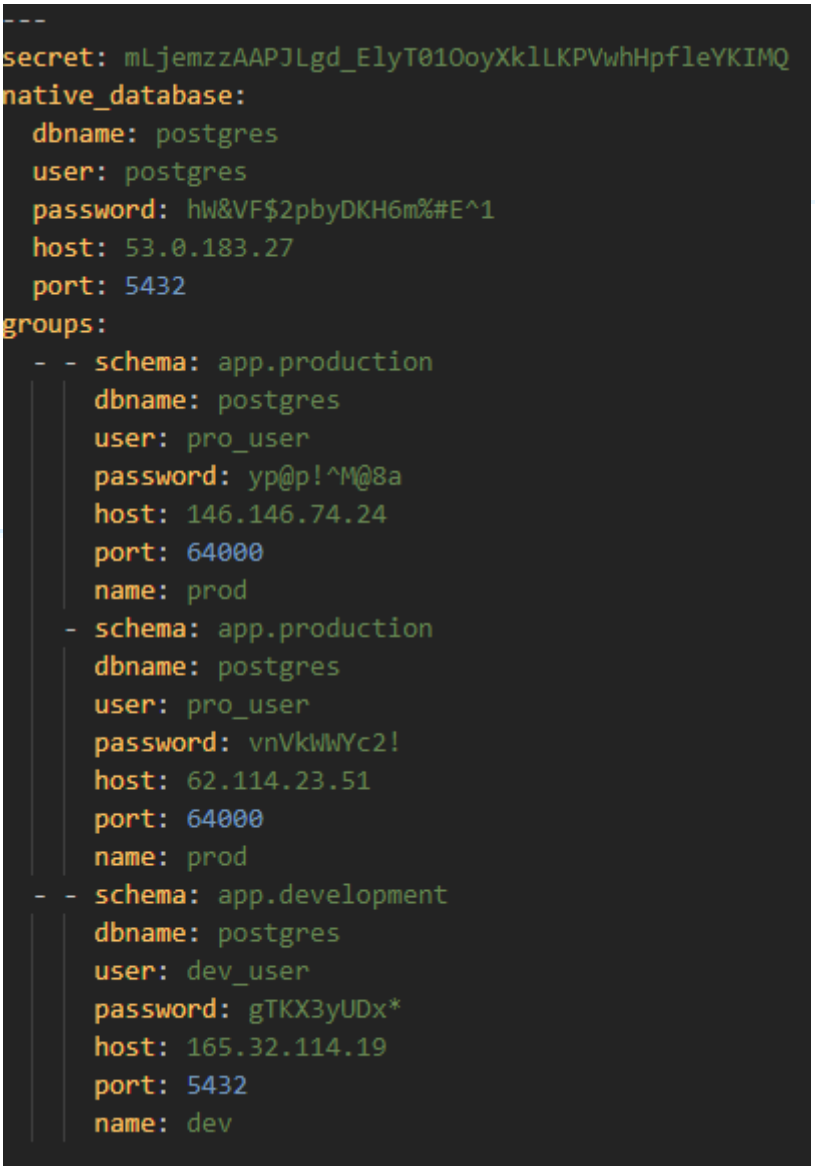
Al ser un modelo recursivo y la lógica de ejecución de consultas también, se puede registrar un número ilimitado de grupos con un número también ilimitado de bases de datos por cada grupo.

Cada base de datos configurada tiene las siguientes variables:

La variable “schema” que indica el nombre del esquema al que realizar las consultas.
La variable “dbname” corresponde con el nombre de la base de datos.
La variable “user” corresponde con el usuario de acceso a la base de datos.
La variable “password” corresponde con la contraseña de acceso a la base de datos.
La variable “host” corresponde con la dirección IP o el DNS de la conexión.
La variable “port” corresponde con el puerto de conexión de la base de datos.
La variable “name” que indica el nombre del grupo al que pertenece la base de datos.

Ejemplo

Ilustración 20: Fichero de configuración de ejemplo con datos ficticios.



POBLADO AUTOMÁTICO DE BASE DE DATOS

Para el correcto funcionamiento de la generación automática de consultas rollback a partir de los comandos recibidos por API, una de las principales funcionalidades de QueryOrchestrator, es imprescindible saber ciertas características sobre las bases de datos y esquemas que se gestionan puesto que en los modelos de datos construídos siguiendo buenas prácticas de diseño, las tablas que los componen suelen estar fuertemente relacionadas, existiendo relaciones de dependencia entre ellas. Por lo que para construir una consulta de rollback en la mayor parte de las ocasiones es necesario tener en cuenta estas relaciones entre las tablas puesto que es necesaria información que no está presente en la propia consulta.

Para abordar este problema se ha construído un sistema que de forma automática clasifica y da de alta en la base de datos interna toda la información de interés acerca de los esquemas que se gestionan. Esta información se obtiene a partir de una serie de consultas predefinidas que se ejecutan sobre la base de datos a gestionar.

Datos de interés

A continuación se muestra un resumen de las tablas y los datos de interés que se almacenan en la base de datos interna:

- Schemas:
  La tabla *schemas* almacena información relacionada a los esquemas de las bases de datos que se gestionan y sirve como punto de partida para identificar a qué cluster de base de datos pertenecen el resto de elementos.

  Esta tabla tiene los siguientes campos:
  - *id*: Identificador único.
  - *name*: Nombre del esquema.
  - *group*: Nombre del grupo o cluster al que pertenece ese esquema.

- Tables:
  La tabla *tables* contiene información sobre las tablas pertenecientes a los clusters de bases de datos que se gestionan, además de la información relevante de las tablas, esta tabla también indica el esquema al que pertenece.

  Esta tabla tiene los siguientes campos:
  - *id*: Identificador único.
  - *name*: Nombre de la tabla.
  - *schema*: Id del esquema al que pertenece la tabla.
  - *table_id*: Nombre del campo id de la tabla.
  - *serial_id*: Indica si el id de la tabla es de tipo *SERIAL*.

- Constraints:
  La tabla *constraints* muestra información acerca de las relaciones entre tablas. Además de estos datos, la tabla también indica a qué tabla pertenece esa relación y por consiguiente a qué esquema pertenece.

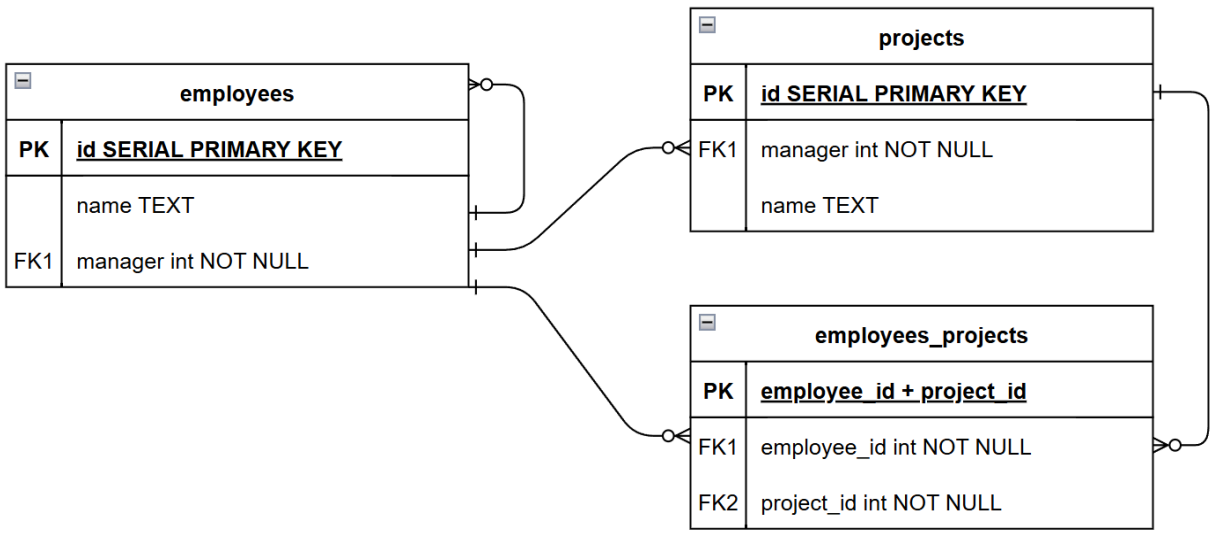
  Esta tabla tiene los siguientes campos:
  - *id*: Identificador único.
  - *name*: Nombre de la relación.
  - *table*: Id de la tabla donde se define la relación.
  - *referenced_table:* Id de la tabla referenciada en la relación.
  - *table_key*: Nombre del campo sobre el que parte la relación.
  - *referenced_table_key*: Nombre del campo sobre el que se ejecuta la relación.

Ejemplo

Durante el desarrollo se utilizó una base de datos para hacer pruebas en la que se describe una gestión sencilla de usuarios y proyectos en una empresa ficticia.

Este es el diagrama entidad-relación de la base de datos de prueba:

Ilustración 21: Base de datos de pruebas.



Consulta para la creación del esquema de ejemplo *test1*:

```
CREATE SCHEMA IF NOT EXISTS test1;

CREATE TABLE IF NOT EXISTS test1.employees (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100),
  manager INTEGER,
  FOREIGN KEY(manager) REFERENCES test1.employees(id)
);


CREATE TABLE IF NOT EXISTS test1.projects (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100),
  manager INTEGER REFERENCES test1.employees(id)
);

CREATE TABLE IF NOT EXISTS test1.employees_projects (
  employee_id INTEGER REFERENCES test1.employees(id),
  project_id INTEGER REFERENCES test1.projects(id),
  CONSTRAINT employees_projects_pk PRIMARY KEY(employee_id, project_id)
);
```

Tras el poblado automático de las base de datos interna con los datos de la base de datos de prueba, las tablas quedan de la siguiente forma:

Ilustración 22: Ejemplo de tabla schemas.

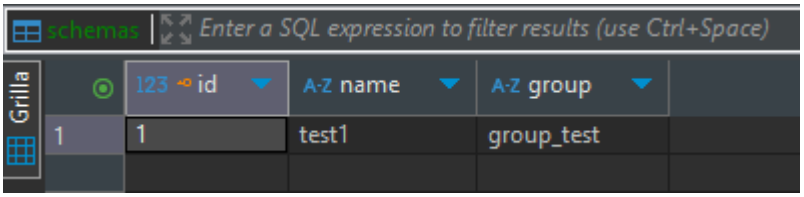


Ilustración 23: Ejemplo de tabla tables.

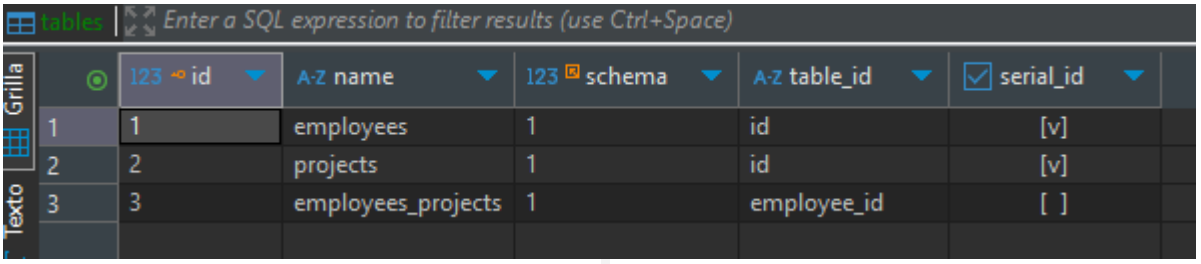
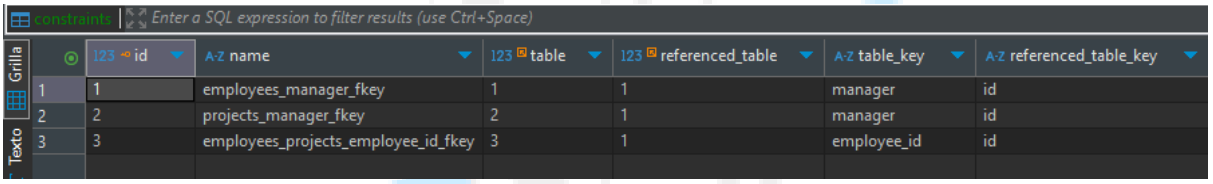


Ilustración 24: Ejemplo de tabla constraints.



Como se puede observar en la imágenes, los registros muestran de forma clara y concisa los datos de interés del esquema de ejemplo así como las relaciones entre sus tablas y las columnas que forman parte de esas asociaciones entre tablas.

GENERACIÓN DE ROLLBACK

La principal diferencia entre QueryOrchestrator y otros balanceadores de bases de datos con un propósito similar es la generación automática de consultas rollback para todas las operaciones de borrado, actualización e inserción de datos realizadas por el usuario.

En el software, todas las consultas quedan registradas en una tabla de gestión interna de auditoría y todas las consultas que cumplen con los requisitos necesarios para generar una consulta de rollback quedan registradas junto a esa nueva consulta, de tal forma que si en algún momento un usuario ha realizado una operación no deseada por equivocación puede revertir los cambios de esa consulta indicando el id de la consulta en una nueva petición API.

Las consultas elegibles para generar un rollback automático son de los siguientes tipos:

Insert

Para generar consultas de tipo INSERT, el software realiza los siguientes pasos:

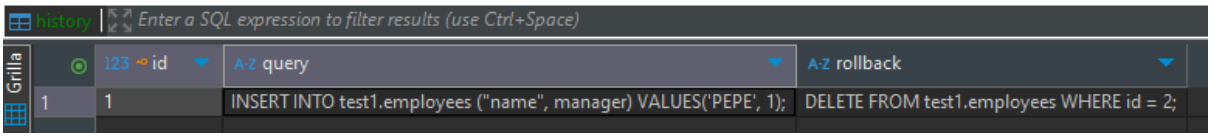
1. Primero reconoce que la consulta en cuestión es del tipo indicado.

2. Acto seguido el sistema evalúa los campos que se van a insertar y los valores que se insertan.

3. Se obtiene a partir de la base de datos interna la columna id de la tabla a la que se pretende insertar.
    a. Si el id es de tipo serial se calcula el id que se generará con la inserción.
    b. Si el id no es de tipo serial se obtiene a partir de la consulta de inserción.

4. Se genera la consulta de borrado de la inserción a partir del id obtenido en el paso anterior.

Ilustración 25: Ejemplo de rollback de insert en base de datos.



Update

Para genera las consultas de tipo UPDATE el software realiza los siguientes pasos:

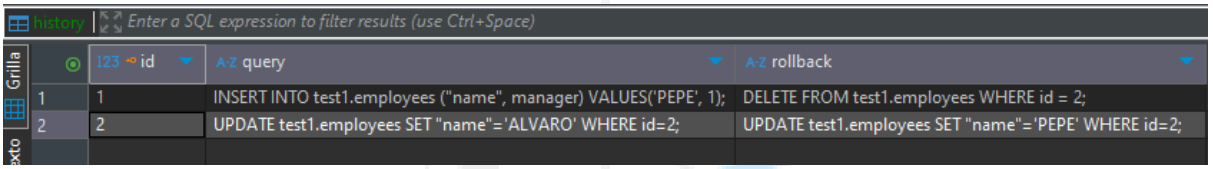
1. Se reconoce que la consulta en cuestión es de tipo UPDATE.

2. Se obtienen los campos que se van a modificar a partir de la consulta de entrada.

3. Se realiza una consulta a la base de datos para obtener los valores de los campos antes del cambio.

4. Se utiliza la misma estructura de la consulta de entrada para generar un rollback con los valores que se obtuvieron en el paso anterior.

Ilustración 26: Ejemplo de rollback de update en base de datos.



Delete

Para la generación de consultas rollback en las operaciones de tipo DELETE se realizan los siguientes pasos:

1. Se reconoce que la consulta en cuestión es de tipo DELETE.

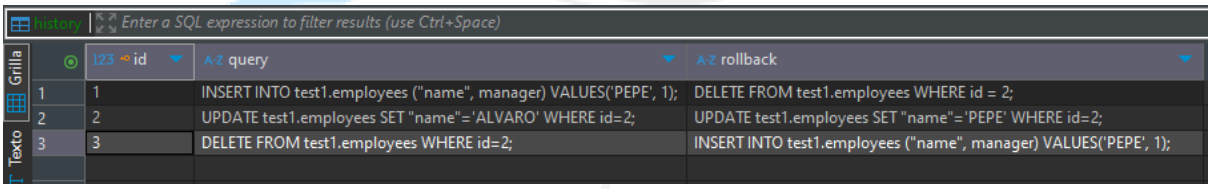
2. Se obtiene la parte WHERE de la consulta de entrada.

3. Se realiza una consulta a la base de datos con la parte WHERE de la consulta de entrada.

4. Se realiza una consulta a la base de datos interna para obtener las relaciones entre las tablas, para obtener posibles borrados en cascada.

5. Por cada entrada recibida:
    a. Se genera una consulta de tipo INSERT.
        i. Si la tabla tiene un id de tipo serial se ignora el id.
        ii. Si la tabla no tiene un id de tipo serial se conserva el id en la consulta INSERT generada.
    b. Si la tabla tiene una relación con otra tal que provoque borrado en cascada:
        i. Se vuelven a ejecutar estos pasos a partir del 4 para los posibles datos borrados en la tabla de la relación.
        ii. Se concatenan las consultas generadas.

Ilustración 26: Ejemplo de rollback de delete en base de datos.



BALANCEO Y EJECUCIÓN DE QUERIES

El principal pilar de QueryOrchestrator es el balanceo de operaciones entre bases de datos, de forma que sea posible la redundancia a través de un punto de acceso centralizado, que sería este software.

La forma en la que se realiza este balanceo es muy sencilla y a pesar de tener espacio de mejora, cumple su función sin problemas. la lógica que se sigue para el balanceo y ejecución de operaciones en dos diferentes clusters configurados es la siguiente:

1. Se recibe la consulta vía API y se comprueban los permisos del usuario.

2. Una vez validados los permisos se obtiene la categoría de la consulta, entre las cuales se encuentran: *INSERT, UPDATE, DELETE* y *OTHER*.

3. Generación de rollback
    a. Si la categoría de la consulta es elegible para rollback (*INSERT, UPDATE, DELETE*) se genera la consulta de rollback.
    b. Si el cuerpo petición tiene el campo rollback, se utiliza el rollback indicado por el usuario.

4. Se obtiene de la configuración de las bases de datos que pertenecen al cluster sobre el que se quiere ejecutar la consulta.

5. Se ejecuta la consulta en todas las bases de datos, en serie.
    a. Si falla la operación en alguna de las consultas se aborta la operación y se notifica al usuario.
    b. Si la consulta tiene un rollback asociado y se ha llegado a ejecutar en alguna de las instancias, se ejecuta en esa instancia el rollback.

6. Se inserta en el histórico la consulta y su rollback (si lo tiene).

7. Se devuelve al usuario el resultado de la operación.

ENDPOINTS

Este apartado constituye un resumen de los endpoints del proyecto y sus posibles respuestas y características.

Todos los endpoints del sistema están protegidos mediante autenticación por token JWT, incluído el endpoint de generación de tokens, siendo necesario para la ejecución de este un token válido con permisos de administración.

Endpoints del sistema

- **GET /status:** Este endpoint sirve como comprobación de que el software se encuentra desplegado correctamente.
  - Respuestas:
    - 200 - OK

```
{
  "status": "ok"
}
```

Endpoints de autenticación

- **POST /generate_token:** Este endpoint requiere permisos de administración para ser ejecutado y genera un token con los permisos indicados en el cuerpo de la petición. La autenticación se realiza con la etiqueta “Authorization” de la cabecera de la petición y un token válido en esa cabecera.
  - Cuerpo:

```
{
  "data": {
    "db1": ["adm"],
    "db2": ["read", "write"]
  },
  "user": "username",
  "is_admin": true
}
```

  - Respuestas:
    - 200 - OK

```
{
  "token": "Bearer token"
}
```

    - 403

```
{
  "error": "Authorization header missing or invalid"
}
```

- 403

```
{
  "error": "Invalid token"
}
```

- 403

```
{
  "error": "Admin role required"
}
```

- 400

```
{
  "error": "Missing one of the required keys: 'user', 'data', 'is_admin'"
}
```

- 400

```
{
  "error": "'data' must be a dictionary"
}
```

- 400

```
{
  "error": "Value for key '{key}' must be a list of strings"
}
```

- 400

```
{
  "error": "'user' must be a string"
}
```

- 400

```
{
  "error": "'is_admin' must be a boolean"
}
```

- 400

```
{
  "error": "Missing JSON payload"
}
```

- 500 : Internal error

Endpoints de ejecución de consultas

- **POST** **/execute_query:** Este sirve para ejecutar consultas en la base de datos seleccionada. La autenticación se realiza con la etiqueta “Authorization” de la cabecera de la petición y un token válido en esa cabecera.
  - Cuerpo:

```
{
  "query": "query",
  "group": "group",
  "rollback": "rollback"
}
```

  - Respuestas:
    - 200 - OK

```
{
  "message": "OK",
  "query_id": 1
}
```

    - 403

```
{
  "error": "Invalid token"
}
```

    - 403

```
{
  "error": "Authorization header missing or invalid"
}
```

    - 403

```
{
  "error": "Unauthorized access to database"
}
```

    - 400

```
{
  "error": "Something went wrong while generating the rollback",
  "message": "Error message"
}
```

    - 400

```
{
  "error": "Something went wrong while executing the query",
  "message": "Error message"
}
```

- POST **/execute_rollback:** Este sirve para ejecutar consultas rollback en la base de datos seleccionada. La autenticación se realiza con la etiqueta “Authorization” de la cabecera de la petición y un token válido en esa cabecera.
    - Cuerpo:

```
{
  "query_id": 1
}
```

    - Respuestas:
        - 200 - OK

```
{
  "message": "OK",
  "query_id": 2
}
```

        - 403

```
{
  "error": "Invalid token"
}
```

        - 403

```
{
  "error": "Authorization header missing or invalid"
}
```

        - 403

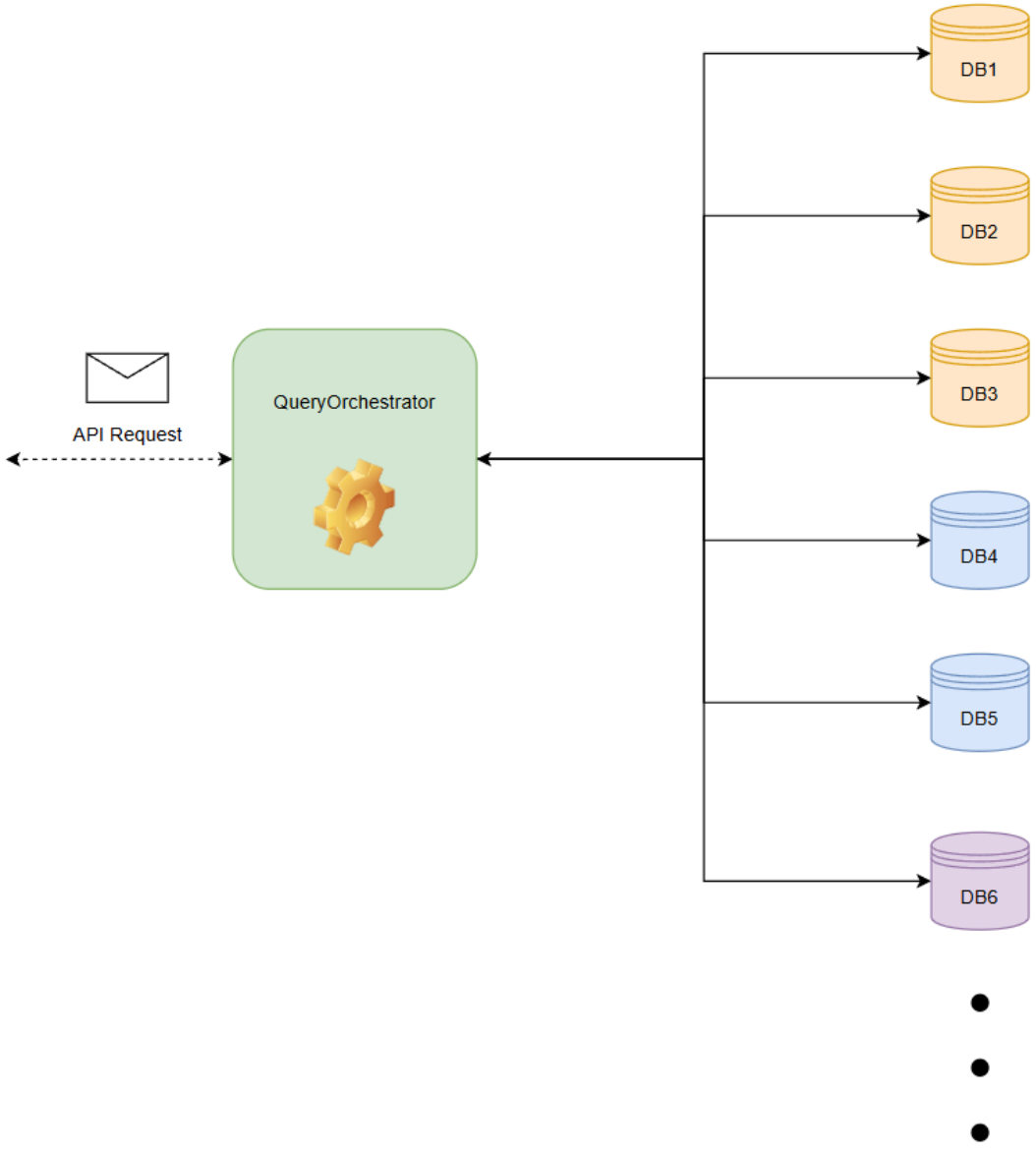
```
{
  "error": "Unauthorized access to database"
}
```

        - 400

```
{
  "error": "Something went wrong while executing the query",
  "message": "Error message"
}
```

ARQUITECTURA

Ilustración 28: Arquitectura del proyecto



MEJORAS

A continuación se enumeran las posibles mejoras para futuras versiones de QueryOrchestrator, orientadas a ampliar su compatibilidad, mejorar el rendimiento y fortalecer otros aspectos clave.

Compatibilidad con otros sistemas de bases de datos

Actualmente, QueryOrchestrator es compatible exclusivamente con bases de datos PostgreSQL. Se planea extender esta compatibilidad a otros sistemas gestores de bases de datos como MySQL, SQL Server y Oracle, entre otros, lo que permitirá una adopción más amplia y resultará en un software más versátil.

Ejecución de operaciones en paralelo

En la versión actual, las operaciones definidas sobre un clúster se ejecutan de forma secuencial. Se implementará la posibilidad de ejecutar consultas en paralelo, haciendo uso de hilos, mediante un parámetro opcional en la solicitud API.

Generación del rollback para otras operaciones

Actualmente, QueryOrchestrator genera instrucciones de rollback automáticas únicamente para operaciones INSERT, UPDATE y DELETE. Se incluirá soporte para la generación automática de consultas rollback para operaciones como DROP, CREATE, ALTER.

Mejora en el sistema de autenticación

La autenticación actual se basa en tokens JWT sin caducidad. Se planea introducir un sistema de autenticación basado en tokens con expiración y con la posibilidad de renovación. Lo que permitirá una gestión más segura de sesiones y usuarios.

OTRAS APLICACIONES

Además de su función principal como gestor de operaciones en clusters de bases de datos. QueryOrchestrator puede ser utilizado en otros contextos:

Pruebas de consistencia de datos

El software permite ejecutar la misma operación sobre bases de datos diferentes, lo que permite evaluar la evolución del mismo dato y su coherencia entre diferentes entornos.

Auditoría centralizada

QueryOrchestrator se puede utilizar como una capa de auditoría centralizada para operaciones SQL, proporcionando mayor trazabilidad en la evolución de los datos en otros proyectos.

Orquestación de actualizaciones masivas de datos

En procesos de migración o de despliegue, se puede utilizar QueryOrchestrator para garantizar los cambios en todas las instancias de bases de datos, manteniendo la posibilidad de rollback.

Interfaz de automatización en procesos CI/CD

Se puede integrar dentro de pipelines existentes como método para ejecutar actualizaciones de datos automáticas durante los procesos de despliegue.

CONCLUSIÓN

QueryOrchestrator nace como una solución flexible y extensible orientada a la gestión de operaciones SQL distribuidas. Su enfoque en la ejecución coordinada de consultas en múltiples bases de datos con posibilidad de rollback en operaciones de inserción, actualización y borrado de datos, junto con un sistema robusto de auditoría, lo convierte en un herramienta a considerar cuando sea necesario en mantenimiento y la gestión de múltiples bases de datos en un proyecto.

BIBLIOGRAFÍA

1. https://learn.microsoft.com/es-es/devops/plan/what-is-agile-development
2. https://agilemanifesto.org/iso/es/principles.html
3. https://www.innk.cl/caracteristicas-de-la-metodologia-scrum/
4. https://scrumguides.org/docs/scrumguide/v1/Scrum-Guide-ES.pdf
5. https://www.isdi.education/es/blog/ventajas-de-la-metodologia-kanban
6. https://asana.com/es/resources/scrumban
7. https://openwebinars.net/blog/que-es-scrumban
8. https://www.iebschool.com/hub/que-es-scrumban-agile-scrum/
9. https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow
10. https://docs.python.org/3/faq/general.html
11. https://flask.palletsprojects.com/en/stable/
12. https://auth0.com/docs/secure/tokens/json-web-tokens
13. https://www.postgresql.org/docs/
14. https://dbeaver.com/docs/dbeaver/
15. https://learning.postman.com/docs/introduction/overview/