
# Informe TP Diseño

# Sistemas Distribuidos I

**Integrantes:**

Mateo Alvarez - 108666

Tomás Szejnfeld Sirkis - 107710

Martín González Prieto - 105738

# Alcance

## Requerimientos Funcionales

- Se solicita un sistema distribuido que analice la información de ventas en una cadena de negocios de Cafés en Malasia.
- Se cuenta con información transaccional por ventas (montos, items vendidos, etc), información de los clientes, de las tiendas y de los productos ofrecidos.
- Queries a hacer:
    - 1. Transacciones (Id y monto) realizadas durante 2024 y 2025 entre las 06:00 AM y las 11:00 PM con monto total mayor o igual a 75.
    - 2. Productos más vendidos (nombre y cant) y productos que más ganancias han generado (nombre y monto), para cada mes en 2024 y 2025.
    - 3. TPV (Total Payment Value) por cada semestre en 2024 y 2025, para cada sucursal, para transacciones realizadas entre las 06:00 AM y las 11:00 PM.
    - 4. Fecha de cumpleaños de los 3 clientes que han hecho más compras durante 2024 y 2025, para cada sucursal.

## Requerimientos No Funcionales

- El sistema debe estar optimizado para entornos multicomputadoras
- Se debe soportar el incremento de los elementos de cómputo para escalar los volúmenes de información a procesar
- Se requiere del desarrollo de un Middleware para abstraer la comunicación basada en grupos.
- Se debe soportar una única ejecución del procesamiento y proveer graceful quit frente a señales SIGTERM.

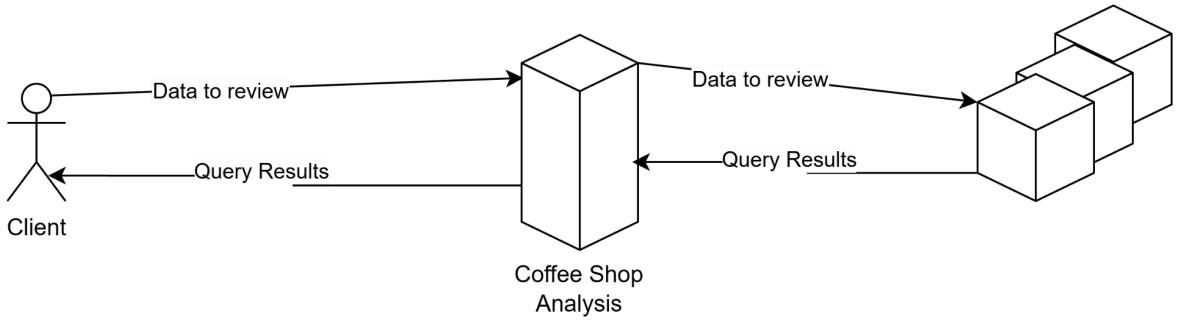
# Escenarios

## Casos de Uso

En el **Diagrama de Casos de Uso**, se identifican los siguientes elementos:

- **Actor Cliente**: Usuario que realiza las consultas al sistema, solicitando información para su análisis.
- **Servidor Coffee Shop Analyst**: Representa el sistema encargado de recibir y procesar los resultados de las consultas realizadas.
- Nodos de analisis: Estos serian los filtros a aplicar a los datos enviados
- **Casos de Uso (Pedido de Querys )**:
    - *R*epresenta los datos enviados y los distintos tipos de respuestas que el cliente puede recibir del sistema. Cada una de estas respuestas es enviada desde el sistema.

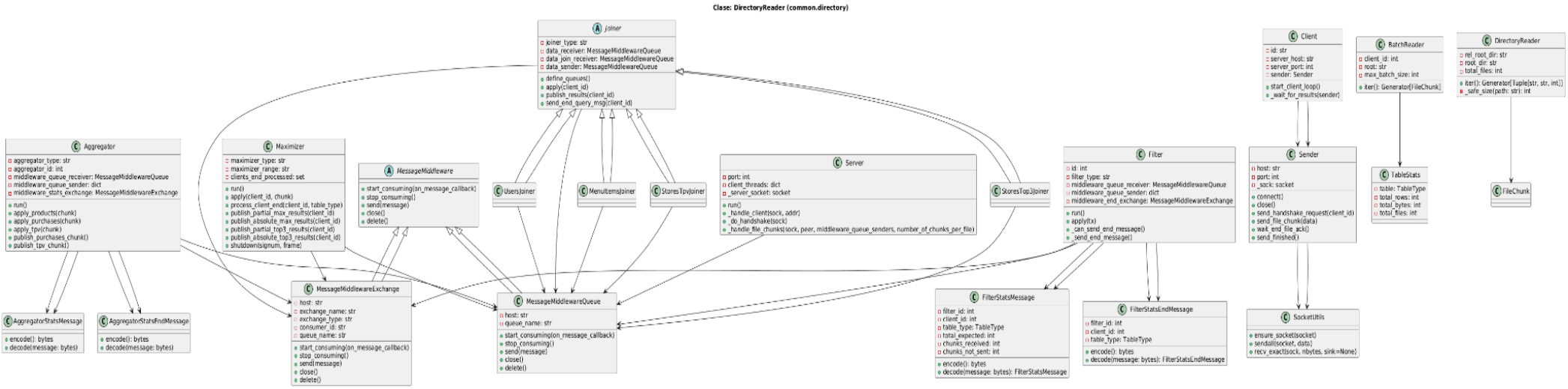
Los escenarios se centran en el flujo de información entre cliente y el sistema Coffee Shop Analysis, destacando cómo el sistema soporta múltiples tipos de consultas. Además, este modelo refuerza el rol del sistema como un **intermediario entre los datos y el análisis de negocio**, asegurando que la interacción sea clara, flexible y extensible a futuros tipos de consultas.



# Vista Lógica

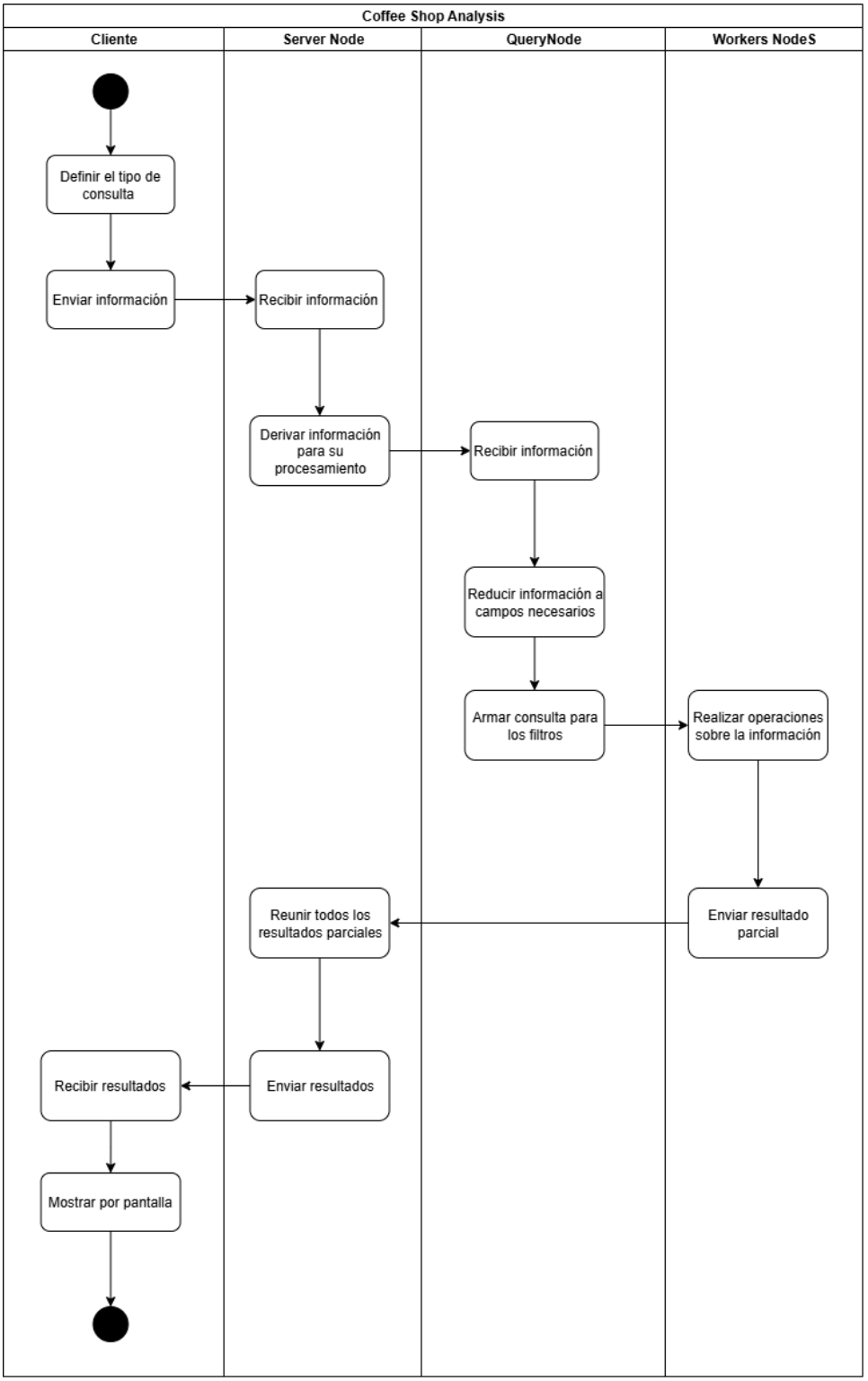
En esta vista se describen los principales módulos del sistema y sus responsabilidades dentro del procesamiento de consultas y manejo de archivos:

## Clases



# Vista de Procesos

## Actividades



El diagrama de actividad describe el flujo de ejecución a nivel de procesos distribuidos, mostrando cómo un cliente inicia una consulta, cómo el servidor central la coordina y cómo los nodos de trabajo procesan la información en paralelo para luego devolver resultados parciales que son ensamblados y enviados de vuelta al cliente. Este enfoque ilustra claramente la naturaleza cooperativa del sistema y cómo se organiza el trabajo en distintas etapas.
El diagrama se organiza en tres swimlanes: Cliente, Server Node, Query Node y Worker Nodes, representando cada uno su rol dentro del sistema.

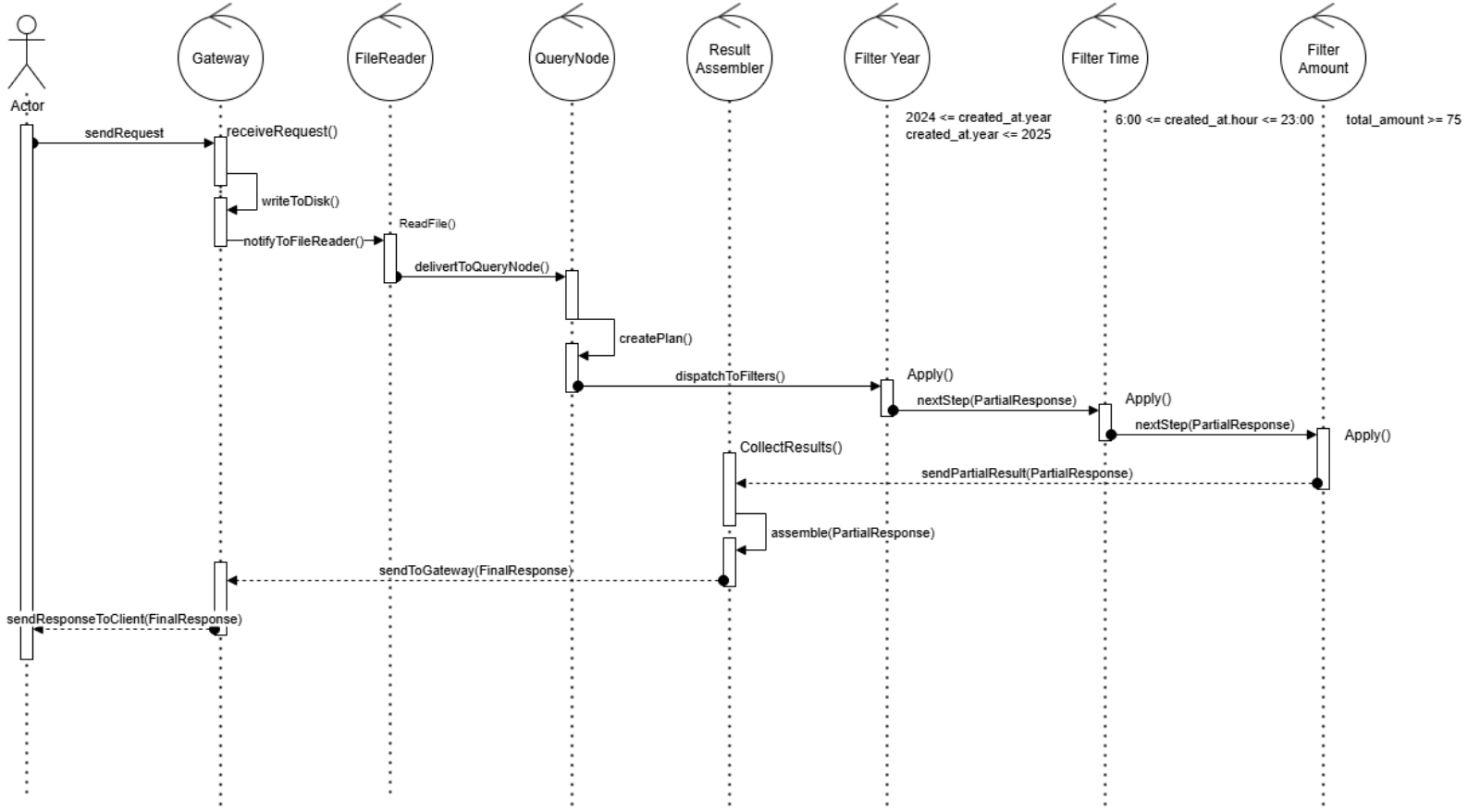
- **Cliente**:
    - Define el tipo de consulta (Query 1, Query 2, Query 3 o Query 4).
    - Envía la información y los parámetros de la query al servidor.
    - Finalmente, recibe los resultados procesados y los visualiza en pantalla.
- Server Node
    - Recibe la información proveniente del cliente.
    - Se la deriva al *QueryNode*
    - Una vez que los workers envían los resultados parciales, el servidor los reúne y los ensambla en un único resultado.
    - Devuelve los resultados finales al cliente.
- **Query Node**
    - Determina el tipo de consulta y sus operaciones
    - Deriva la información hacia los nodos de trabajo encargados del procesamiento.
- **Worker Nodes**:
    - Reciben los fragmentos de información enviados por el servidor.
    - Reducen los datos a los campos estrictamente necesarios según la query.
    - Ejecutan las operaciones específicas (filtros, agregaciones, joins, etc., dependiendo de cada query):
        - Query 1: filtrar por fechas, horarios y montos, luego agregar montos.
        - Query 2: calcular productos más vendidos y más rentables por mes.
        - Query 3: calcular TPV semestral por sucursal en franjas horarias válidas.
        - Query 4: contar compras por cliente, obtener top 3 y cruzar con datos de cumpleaños.
    - Envían los resultados parciales al servidor.

El servidor reúne todos los resultados parciales, los consolida y los envía al cliente, quien los recibe y los presenta en pantalla, finalizando así el proceso.

# Secuencias

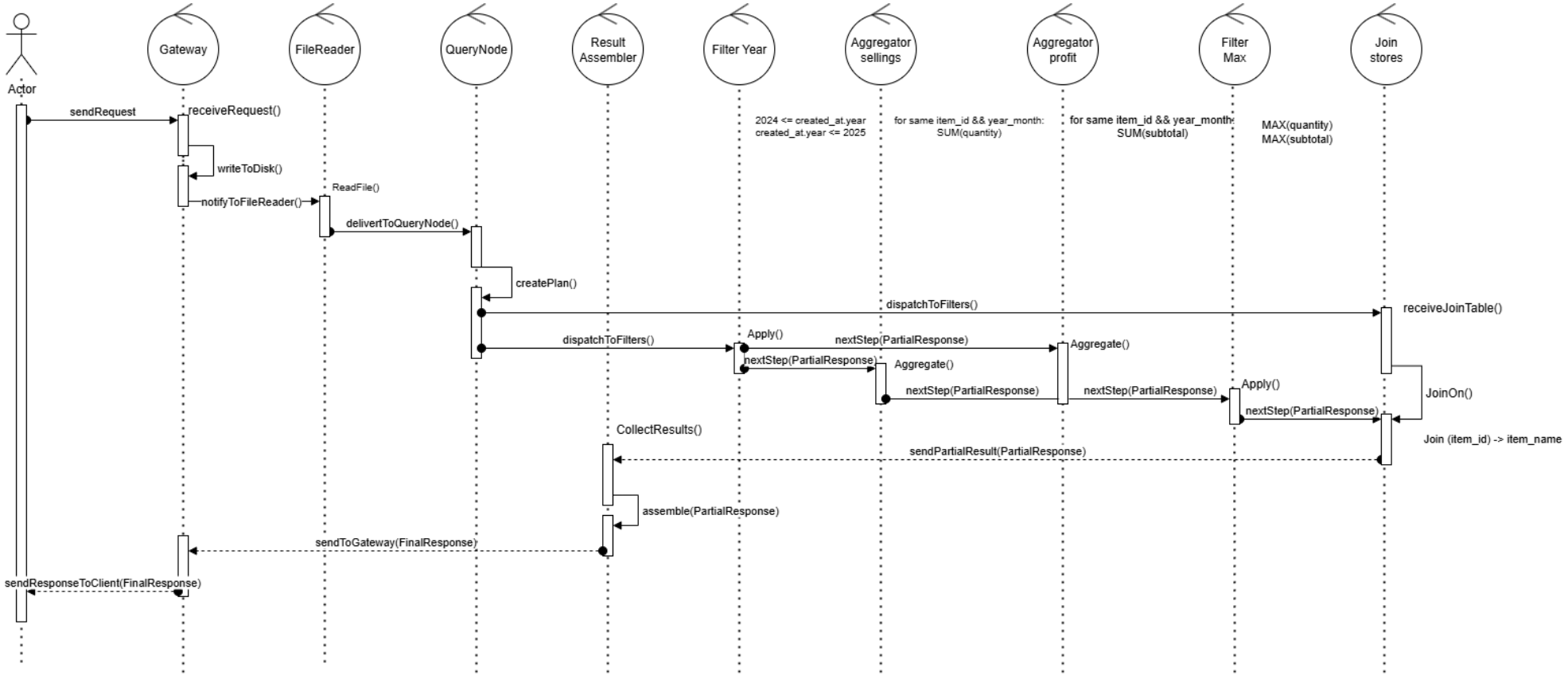
**Query 1 – Transacciones filtradas por año, hora y monto**

El flujo inicia con la recepción de la consulta por el *Gateway*, que delega la lectura al *FileReader*. El *QueryNode* aplica en secuencia los filtros de año, rango horario y monto mínimo. Los resultados parciales se envían al *Result Assembler*, que arma la respuesta final y la devuelve al cliente.



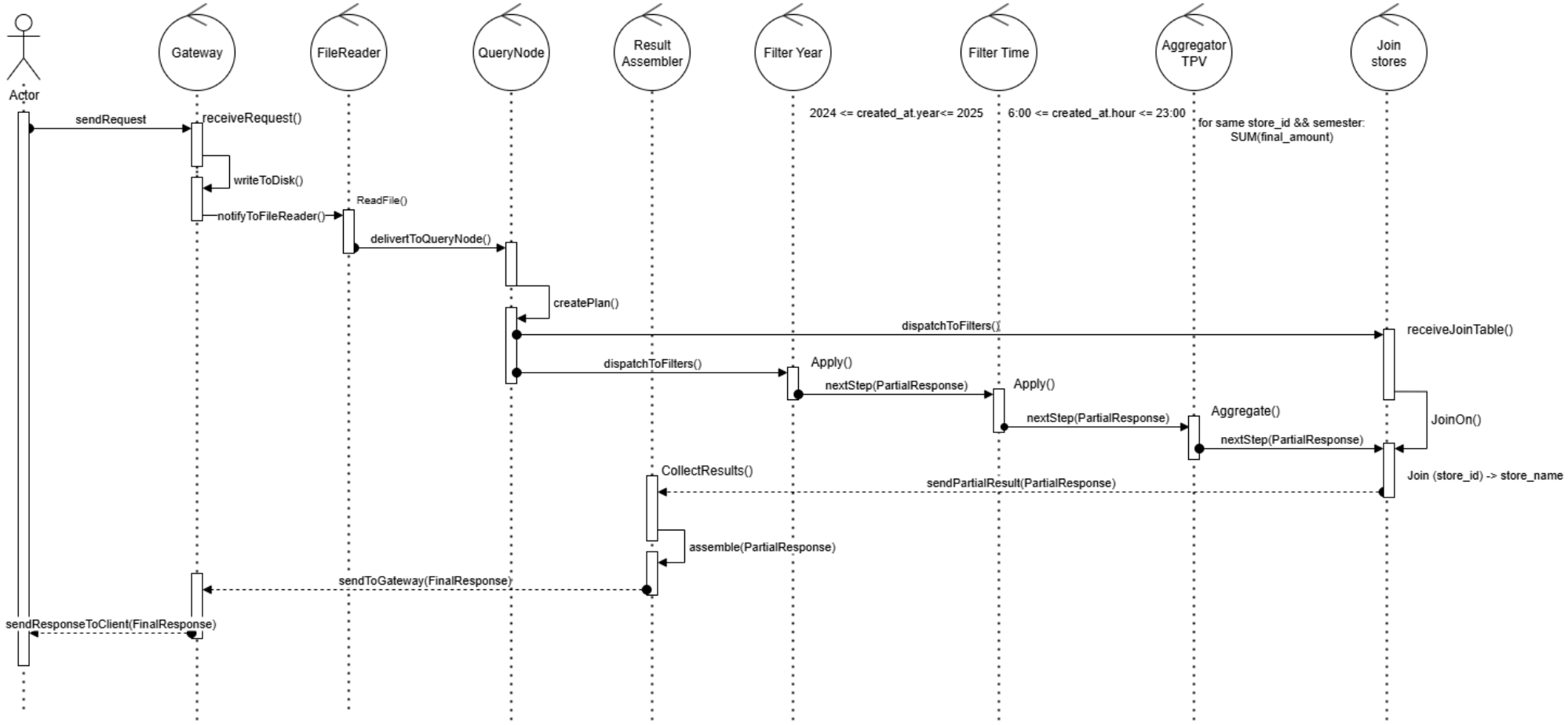
**Query 2 – Productos más vendidos y con mayores ganancias**

Luego de la lectura inicial, el *QueryNode* envía los datos a un filtro de años y luego a dos *Aggregators*: uno calcula las cantidades vendidas y el otro las ganancias por producto y mes. Posteriormente, se aplica un filtro para obtener los máximos y se realiza un *Join* con la tabla de productos para obtener los nombres. Los resultados se integran y devuelven al cliente.



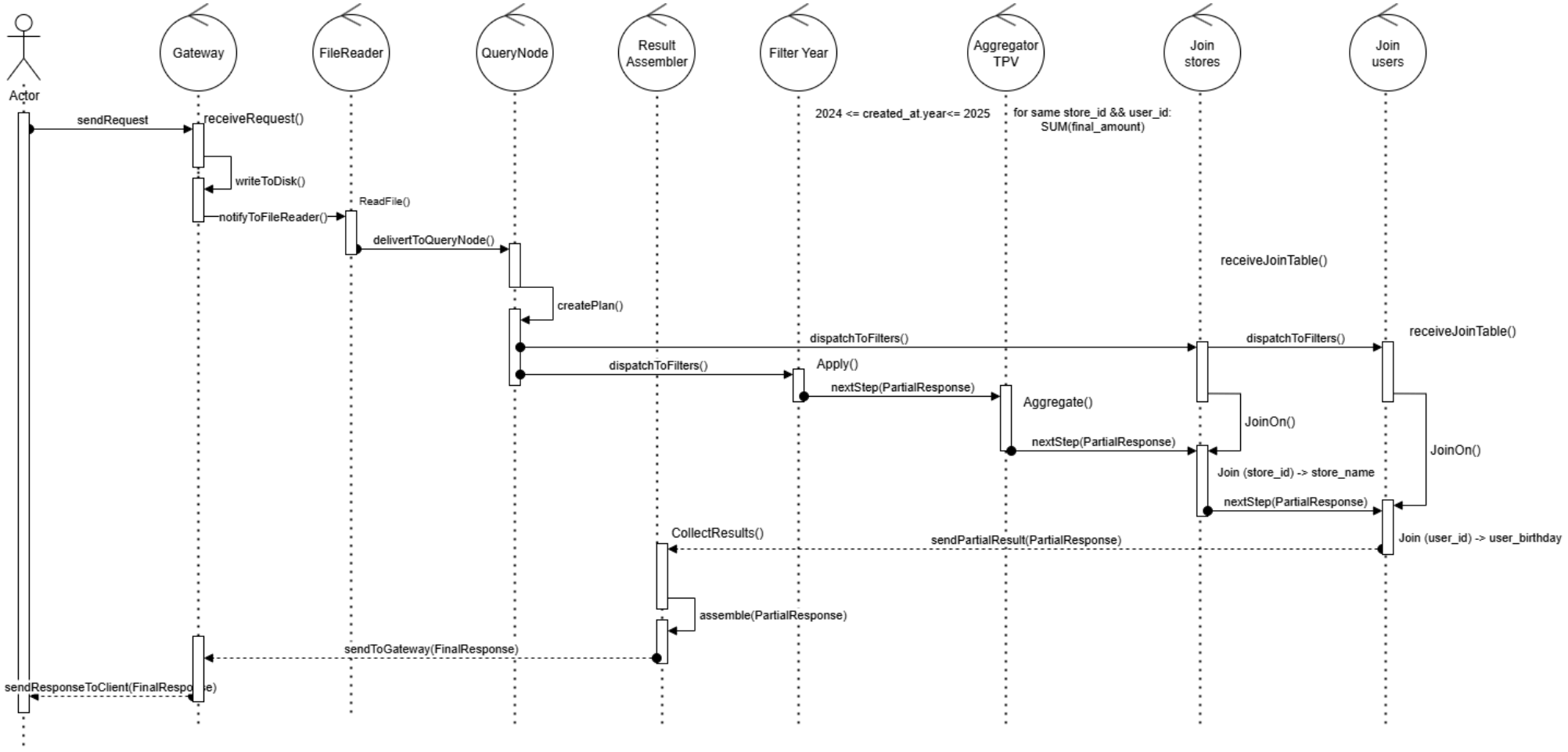
**Query 3 – TPV por semestre y sucursal**

Luego de la lectura inicial, el *QueryNode* aplica filtros de año y rango horario, y luego un *Aggregator* calcula la suma de montos por semestre y sucursal. Finalmente, se hace un *Join* con la tabla de sucursales para obtener los nombres, y se retorna la respuesta procesada.



**Query 4 – Cumpleaños de los 3 clientes con más compras por sucursal**

Tras recibir y leer la información, el *QueryNode* filtra por años válidos. Luego, un *Aggregator* cuenta las compras por cliente y sucursal, y los resultados se combinan con un *Join* sobre sucursales y otro sobre usuarios, de donde se obtiene la fecha de cumpleaños. El *Result Assembler* compone la respuesta final para el cliente.



# Comunicación entre nodos homólogos para reenvío de END

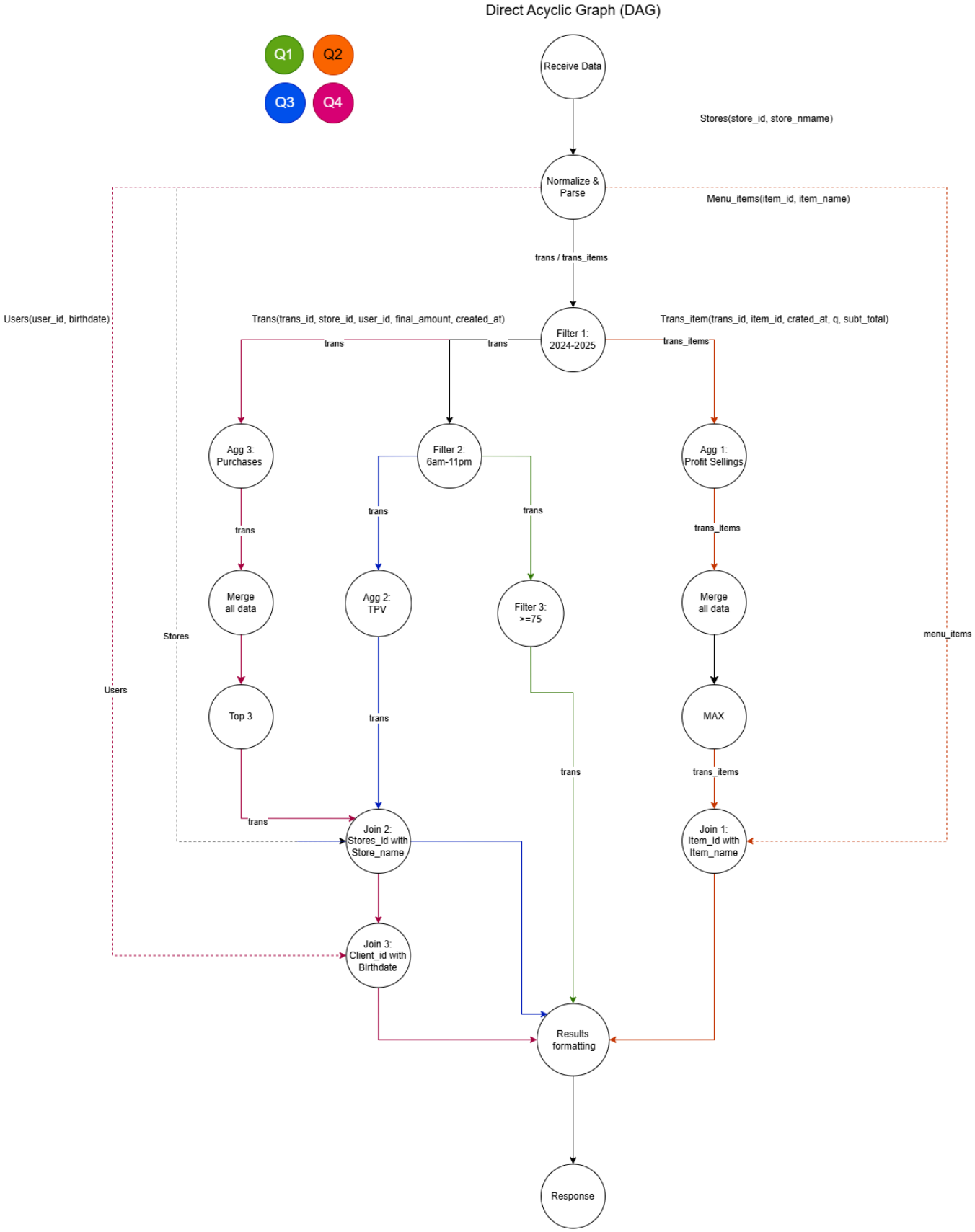
Cuando un nodo recibe un mensaje **END**, este indica que un cliente ha terminado de enviar todos sus datos. El filtro actualiza sus contadores internos, y genera un **mensaje de** estadísticas, como **FilterStatsMessage** por ejemplo. Este se publica en un *exchange* de tipo *fanout*, permitiendo que otros nodos del mismo tipo sepan que el mensaje END ya llegó, enviar su estado actual y actualizar su estado interno.

Luego, el nodo verifica si ya recibió todos los *chunks* esperados de ese cliente. Si es así, el nodo con ID 1, envía un nuevo **mensaje END** a las colas de los siguientes nodos (otros filtros o agregadores), indicando el cierre del flujo para ese cliente. Finalmente, emite un mensaje de fin de estadisticas, como **FilterStatsEndMessage** por ejemplo, al *exchange* para notificar que el procesamiento ha finalizado y limpia sus estructuras internas.

En resumen, el mensaje END activa un proceso de sincronización y propagación que marca el fin de una etapa de filtrado y garantiza la continuidad ordenada del flujo de datos hacia los siguientes componentes del sistema.

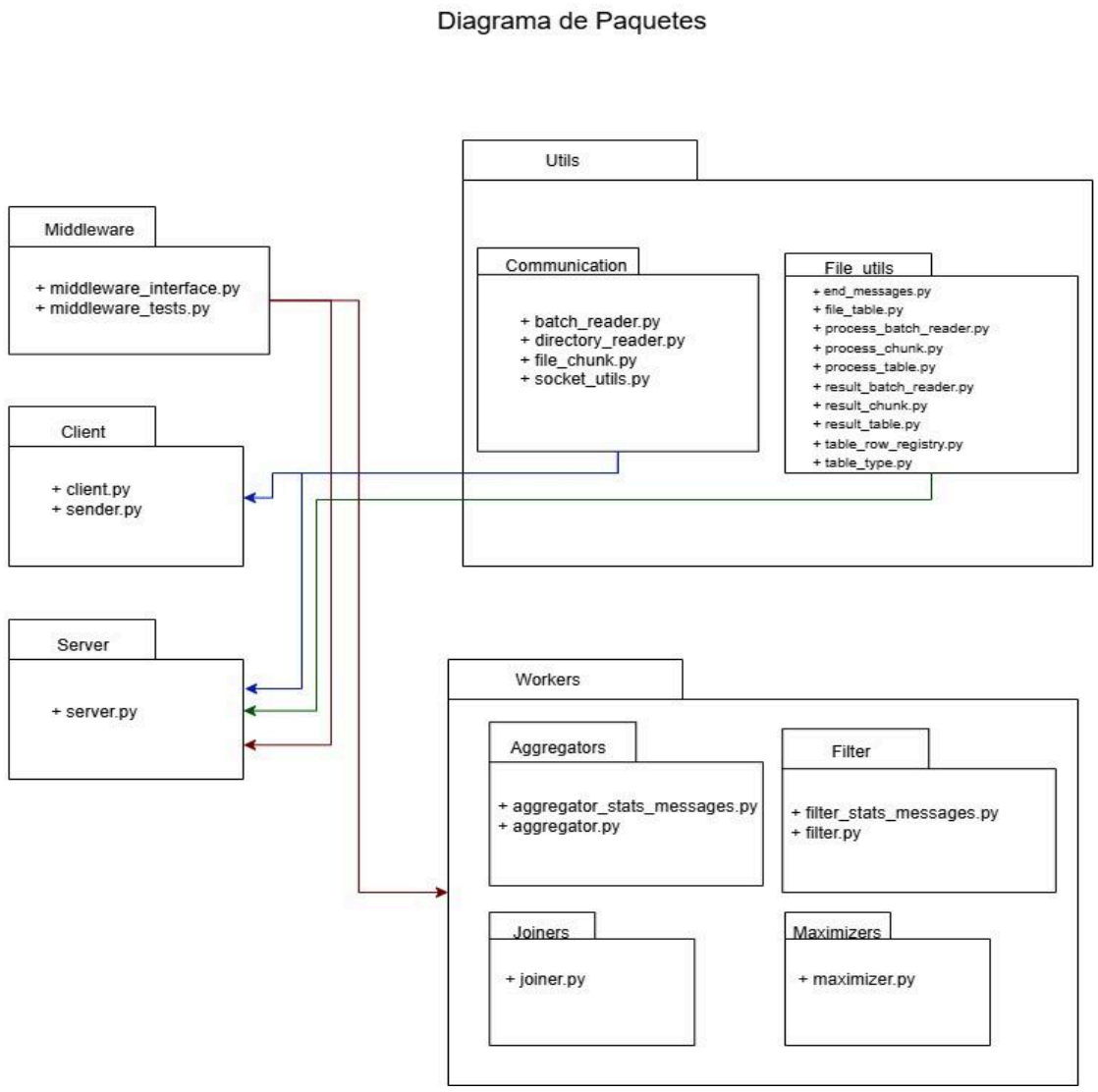
# DAG

Los **DAGs (Direct Acyclic Graphs)** representan la ejecución de consultas como un flujo de pasos secuenciales y ramificados. Cada consulta se modela como una cadena de transformaciones de datos: primero se recibe y normaliza la información, luego se aplican filtros, agregaciones o uniones, y finalmente se formatea la respuesta.


Direct Acyclic Graph (DAG)
Q1
Q2
Q3
Q4
Receive Data
Stores(store_id, store_nmame)
Normalize & Parse
Menu_items(item_id, item_name)
trans / trans_items
Users(user_id, birthdate)
Trans(trans_id, store_id, user_id, final_amount, created_at)
Trans_item(trans_id, item_id, crated_at, q, subt_total)
Filter 1: 2024-2025
trans
trans
trans_items
Agg 3: Purchases
Filter 2: 6am-11pm
Agg 1: Profit Sellings
trans
trans
trans
trans_items
Merge all data
Agg 2: TPV
Filter 3: >=75
Merge all data
Stores
menu_items
Users
Top 3
trans
MAX
trans
trans
trans_items
Join 2: Stores_id with Store_name
Join 1: Item_id with Item_name
Join 3: Client_id with Birthdate
Results formatting
Response

# Vista de Desarrollo

## Paquetes




En la parte superior se encuentra el paquete **Utils**, que contiene dos submódulos principales: **Communication** y **File_utils**, encargados de la comunicación TCP y el manejo de archivos y consultas respectivamente.

El paquete **Middleware** incluye la lógica intermedia de comunicación, con los archivos middleware_interface.py y middleware_tests.py.

Por otro lado, el paquete **Client** representa la capa que envía los datos para hacer la consulta, mientras que el paquete Server actúa como receptor o procesador principal.

Finalmente, el paquete Workers agrupa diferentes tipos de tareas: Aggregators (para consolidar estadísticas), Filter (para filtrar datos innecesarios), Joiners (para unir tablas) y Maximizers (para obtener valores máximos como Top3 o Top1).

Las flechas entre los paquetes indican dependencias: cómo cada módulo utiliza funciones o clases de otros, destacando la interacción entre Client, Server y Utils para el procesamiento y transmisión de datos.

# Componentes

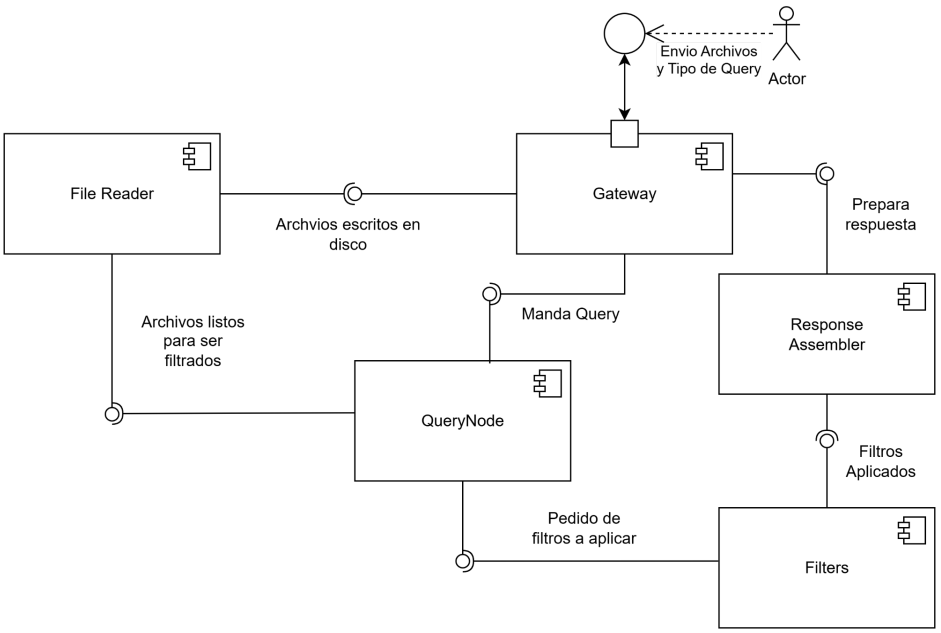
El **Diagrama de Componentes** detalla la interacción operativa:

1. El **Actor** interactúa con el sistema enviando archivos y el tipo de consulta a través del Gateway.
2. El **File Reader** gestiona la lectura de los archivos y prepara los datos para su procesamiento.
3. 3. El **QueryNode** recibe las solicitudes de consulta desde el Gateway y solicita a Filters la aplicación de los filtros requeridos.
4. Una vez aplicados los filtros y operaciones, el Response Assembler construye la respuesta que será entregada al actor.

# Consideraciones de Diseño

La arquitectura se basa en una separación modular de responsabilidades, donde cada paquete concentra una funcionalidad específica. El Gateway centraliza la entrada de datos y solicitudes, facilitando la trazabilidad y control del flujo. Además, la existencia de Utils como paquete independiente fomenta la reutilización de código y la reducción de acoplamiento. También, el Query Node se establece como el componente clave en la ejecución de consultas, actuando como intermediario entre la lógica de negocio (filtros y agregaciones) y los datos de entrada.
Esta organización permite un sistema escalable, mantenible y con roles claramente definidos, asegurando que cada cambio futuro pueda concentrarse en un paquete sin afectar de manera significativa a los demás.

**Diagrama de Componentes**



# Vista Física

## Arquitectura de Nodos

En la parte izquierda se observan los **clusters de procesamiento**, donde se realizan tareas específicas:

- **Filters Cluster** aplica filtros sobre los datos (por año, tiempo y cantidad).
- **Joiners Cluster** combina información de diferentes fuentes, como ítems y productos.
- **Aggregators Cluster** resume datos, por ejemplo, calculando ventas mensuales.

Estos clusters envían los resultados hacia el **Messaging Node**, que utiliza un sistema de mensajería **RabbitMQ** para gestionar la comunicación entre los distintos módulos.

Del lado derecho se encuentra el **Server Node**, que contiene tres componentes principales:

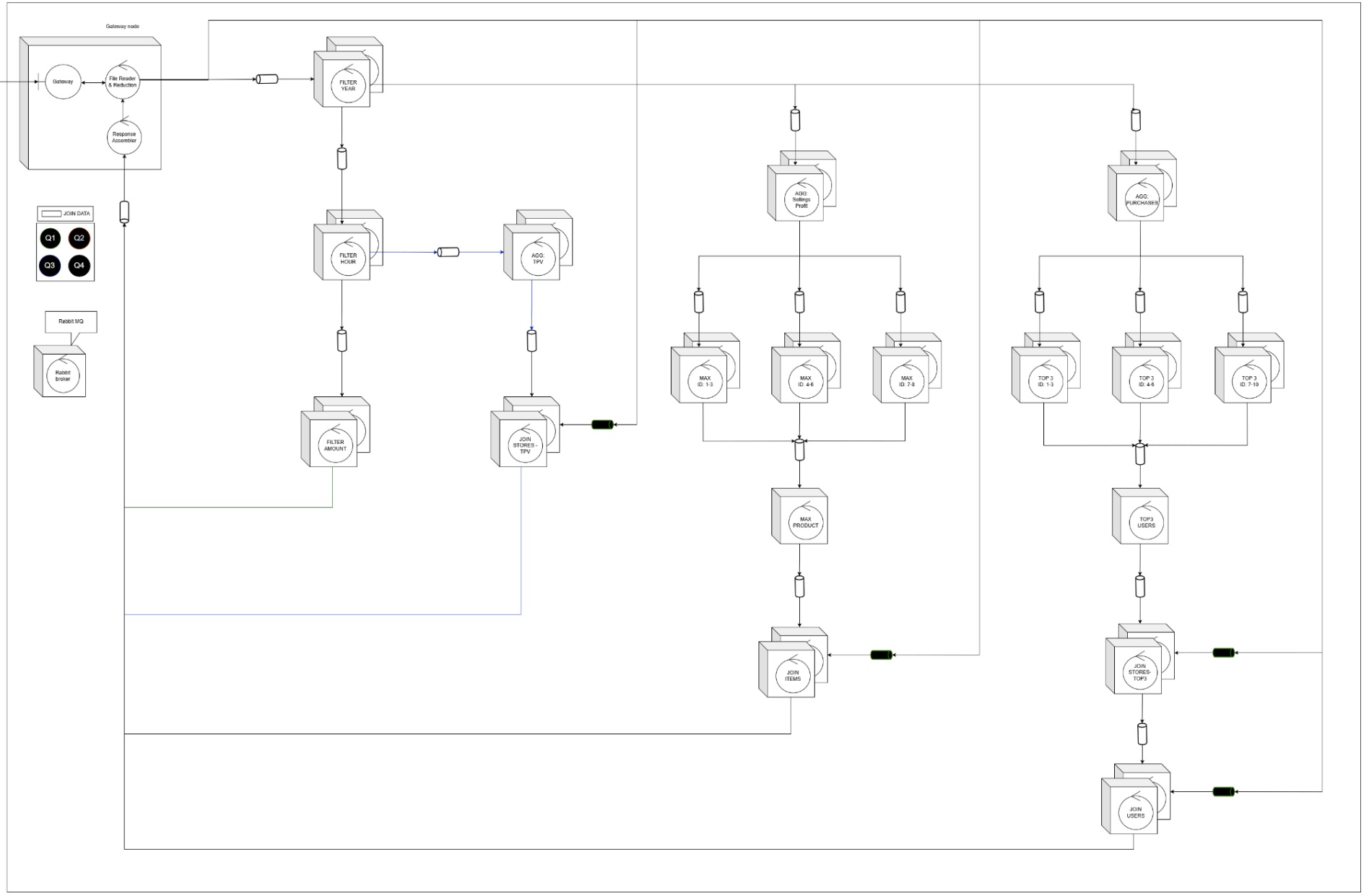
- **Results Assembler**, que organiza y prepara los resultados,
- **Gateway**, que actúa como punto de acceso al servidor.

Finalmente, el **cliente** (por ejemplo, una laptop o PC) se conecta al sistema a través del gateway para enviar solicitudes y recibir resultados procesados.

## Middleware de Mensajería

El sistema emplea **RabbitMQ** como bus de comunicación asíncrona. Este middleware desacopla la interacción entre los clusters, permitiendo escalabilidad y tolerancia a fallos. Los mensajes enviados incluyen instrucciones de procesamiento y resultados intermedios, lo que asegura una ejecución distribuida eficiente.

# Diagrama de Robustez



# Listado de Tareas - TP Diseño

1. **Protocolo de comunicación (Gateway – Socket Protocol)**
    - Documento técnico con el diseño del protocolo cliente - servidor Implementación de headers de comandos: begin_connection, upload, start_job, submit_results, ping.
    - Módulo de sistema de pings para mantener conexión persistente.

      **Asignado**: Mateo

2. **Middleware de Mensajería (RabbitMQ – MOM)**

    - Configuración de RabbitMQ y sus canales internos.
    - Implementación de la API de interacción: publish, subscribe, ack/nack, pause/resume, emit_barrier, graceful_shutdown, etc
    - Módulo de **retry handling** con reintentos automáticos.

      **Asignado**: Tomás

3. **Gateway (Socket Server)**
    - Servidor de sockets que acepte múltiples conexiones simultáneas.
    - Parser y validador de mensajes de cliente → publicación a canal raw en chunks.
    - Diccionario request_id → client_conn para hacer push de resultados.
    - Manejo de conexiones persistentes y re-conexiones de clientes.

      **Asignado**: Mateo

4. **File Reader**
    - Input de de chunks de archivo .csv.
    - Eliminación de redundancias y parseo de datos.
    - Normalización opcional. Depeniendo de query.
    - Publicación en topic normalized.

      **Asignado**: Martín

5. **Query Node**
    - Módulo que genere “planes de ejecución de queries”.
    - Definición de qué colas/etapas participan, filtros, agregaciones y joins.
    - Estado persistente de cada job.
    - Cancelación de jobs en ejecución.
    - Cola FIFO interna para queries entrantes.

      **Asignado**: Tomás

6. **Módulo de Filtros (Filters Workers)**
    - Implementación de filtros parametrizados:
        - year in {2024,2025}
        - hour between 06:00–23:00
        - amount ≥ 75
    - Publicación en filtered.
    - Logging de operaciones.

    **Asignado**: Tomás

7. **Módulo de Joiners (Join Workers)**
    - Join entre item_norm y product_norm.
    - Enriquecimiento con product_name, unit_price, unit_cost.
    - Publicación en joined.

    **Asignado**: Mateo

8. **Módulo de Aggregators (Aggregator Workers)**
    - Implementación de sumatorias sobre campos configurables.
    - Agrupación parametrizada.
    - Publicación en aggregated.

    **Asignado**: Mateo

9. **Query Result Messages**
    - Respuesta de los nodos hacia el servidor con la respuestas de las querys

    **Asignado**: Tomás

10. **End Protocol para Escalado**
    - Protocolo de Comunicación entre nodos del mismo tipo para saber cuando termino de recibir información entre sí

    **Asignado**: Tomás

11. **Preparación de Multiclient**
    - Que todos los nodos puedan manejar múltiples clientes

    **Asignado:** Martín, Tomás y Mateo

12. **Preparación de Escalado**
    - Que todos los nodos puedan manejar ser escalados si necesario

    **Asignado:** Martín, Tomás y Mateo

13. **Manejo de Señales y Graceful Shutdown**
    - Implementación uniforme en todos los procesos:
        - Al recibir SIGTERM: pausar consumo → ack → cerrar conexión.

      **Asignado**: Martín

14. **Retry Handler - A posteriori**
    - Módulo central de logging de errores.
    - Reintentos configurables tras fallos.
    - Recuperación de resultados intermedios si es posible.

      **Asignado**: