.UBAfiuba
FACULTAD DE INGENIERÍA

Sistemas Distribuidos I [TA050]

TP Diseño - Coffee Shop Analysis

Grupo Kafcafé

Integrante	Padrón	Email
Castro Martinez, Jose Ignacio	106957	jcastrom@fi.uba.ar
Diem, Walter Gabriel	105618	wdiem@fi.uba.ar
Gestoso, Ramiro	105950	rgestoso@fi.uba.ar

Índice

1. Alcance **3**
Diagrama de casos de uso 3
2. Objetivos arquitecturales y limitaciones **4**
Objetivos 4
Limitaciones 4
3. Vista lógica **5**
DAG (Directed Acyclic Graph) 5
4. Vista de procesos **6**
Diagramas de actividad 6
Diagramas de secuencia 11
5. Vista de desarrollo **16**
Diagrama de paquetes 16
6. Vista Física **17**
Diagrama de robustez 17
Proceso de consenso para batch terminado 22
Diagrama de despliegue 22
7. División de tareas **23**
8. Referencias **24**

1. Alcance

El trabajo comprende los siguientes casos de uso:

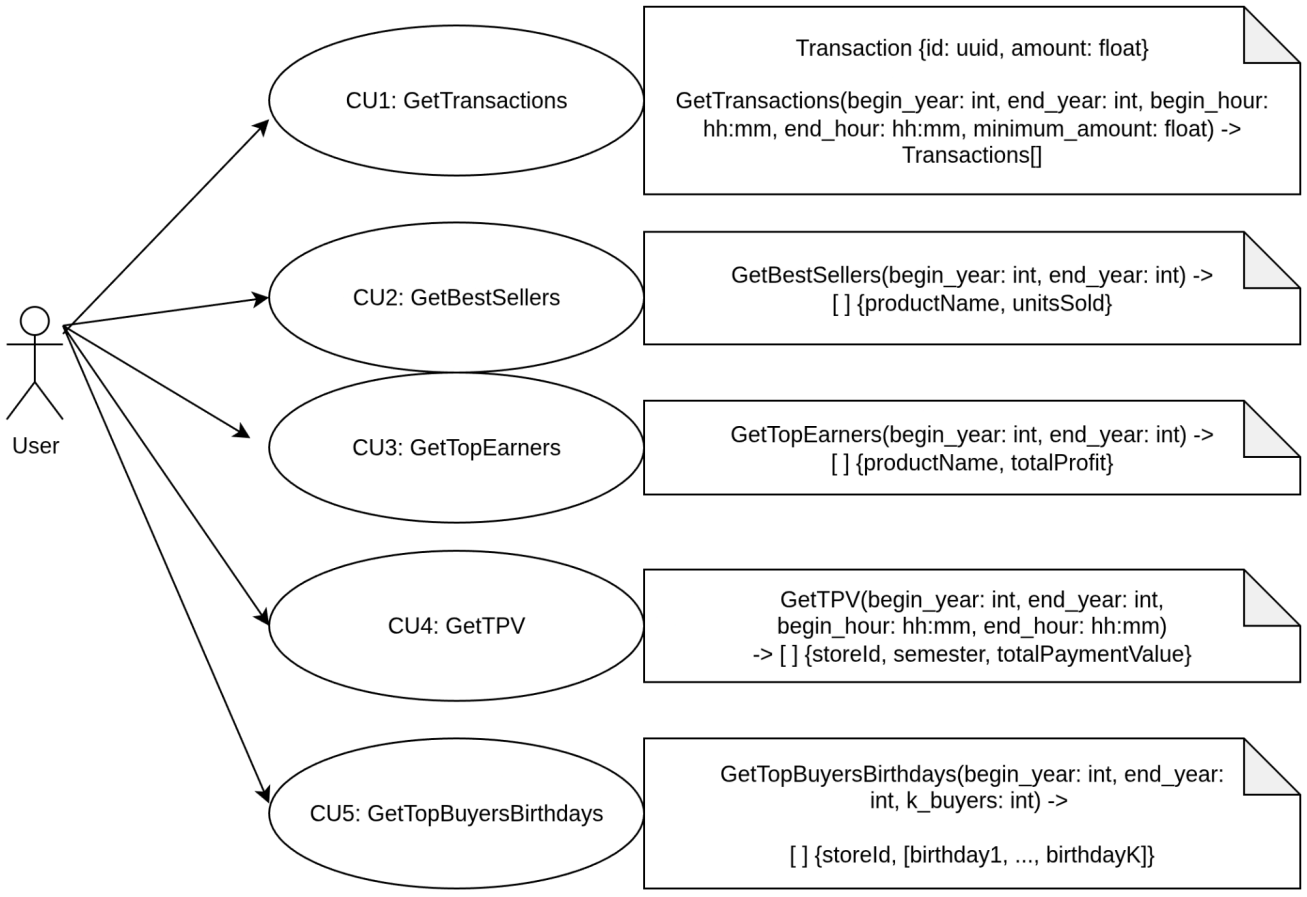
1. Obtener transacciones (id y monto) realizadas durante 2024 y 2025 entre las 06:00 AM y las 11:00 PM con monto total mayor o igual a 75.

2. Obtener productos más vendidos (nombre y cantidad) y productos que más ganancias han generado (nombre y monto), para cada mes en 2024 y 2025.

3. Obtener 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. Obtener fecha de cumpleaños de los 3 clientes que han hecho más compras durante 2024 y 2025, para cada sucursal.

Diagrama de casos de uso



Se agregó como notas a los casos de uso una notación en pseudocódigo de cómo se vería el caso de uso como una función, y cuál sería el valor de retorno de la misma.

2. Objetivos arquitecturales y limitaciones

Objetivos

Escalabilidad Horizontal Multicomputing
El sistema debe poder incorporar nuevos nodos de procesamiento en un entorno multicomputing para manejar mayores volúmenes de datos.

Procesamiento Distribuido de la Información
El sistema debe permitir procesar grandes volúmenes de datos transaccionales (ventas, clientes, productos, sucursales) de manera paralela y eficiente.

Abstracción de Comunicación entre Componente
Todo el intercambio de información entre nodos debe realizarse mediante un *MOM* (Message Oriented Middleware, el cual será RabbitMQ), asegurando desacoplamiento y transparencia de distribución.

Transparencia de Acceso y Localización
Los usuarios (consultas al sistema) no deben percibir dónde ni cómo se ejecuta el análisis: el sistema actúa como una unidad lógica homogénea.

Compatibilidad con formatos de datasets estándar
El sistema debe poder trabajar directamente con los archivos provistos (transacciones, items, usuarios, tiendas, productos) del dataset especificado en Kaggle en formato CSV.

Limitaciones

Fuente de Datos Restringida
El sistema se diseña exclusivamente sobre el formato CSV del dataset proporcionado por la cátedra.

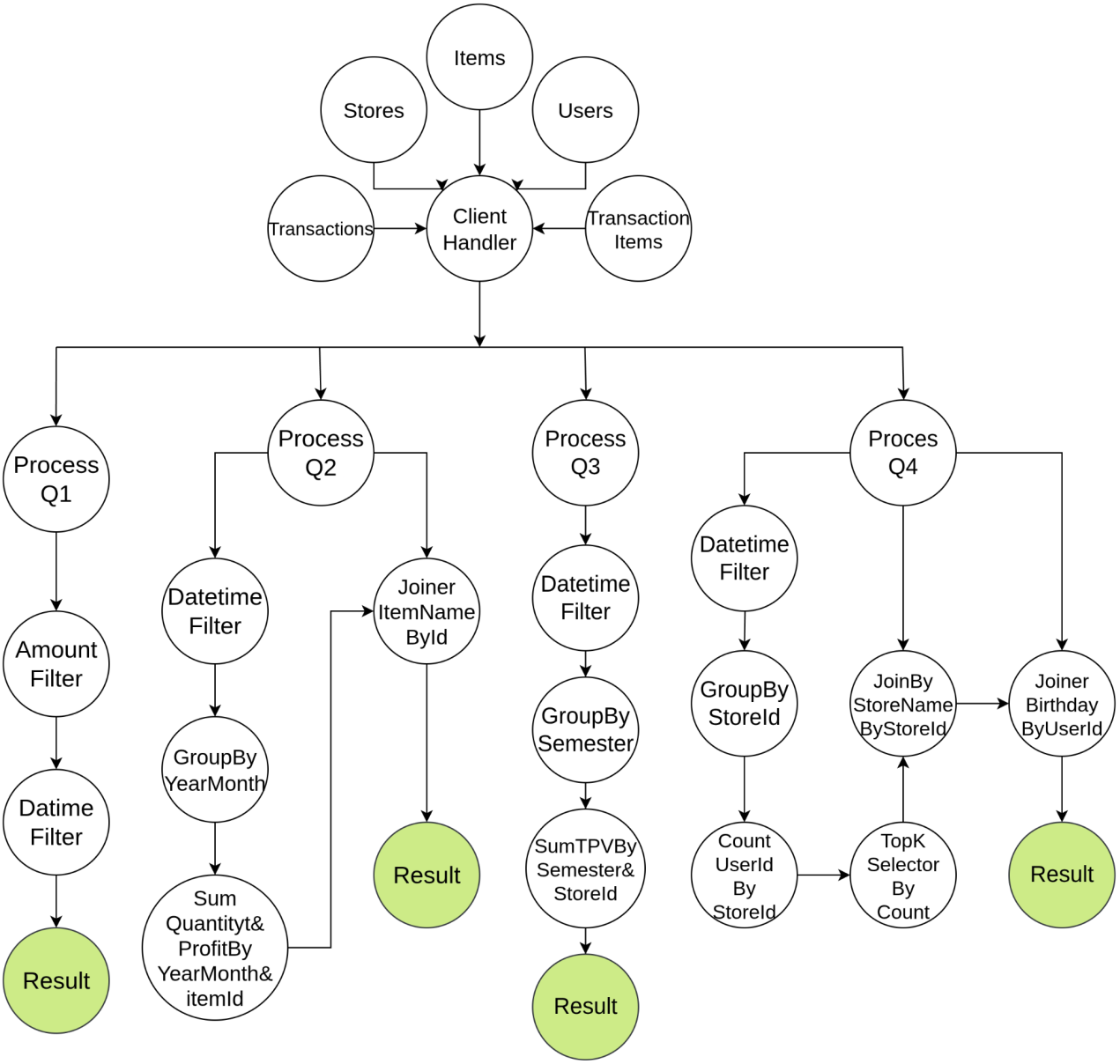
Ejecución única de procesamiento
El diseño actual comprende un único cliente que ejecuta un único procesamiento de los datos, contemplando este todas las queries en simultáneo.

Tipos de consultas acotados al alcance
El sistema puede resolver las consultas especificadas en los casos de uso, con una determinada flexibilidad para cambiar los parámetros de consulta. No se contempla un uso genérico del sistema como lo sería un DBMS SQL.

3. Vista lógica

DAG (Directed Acyclic Graph)

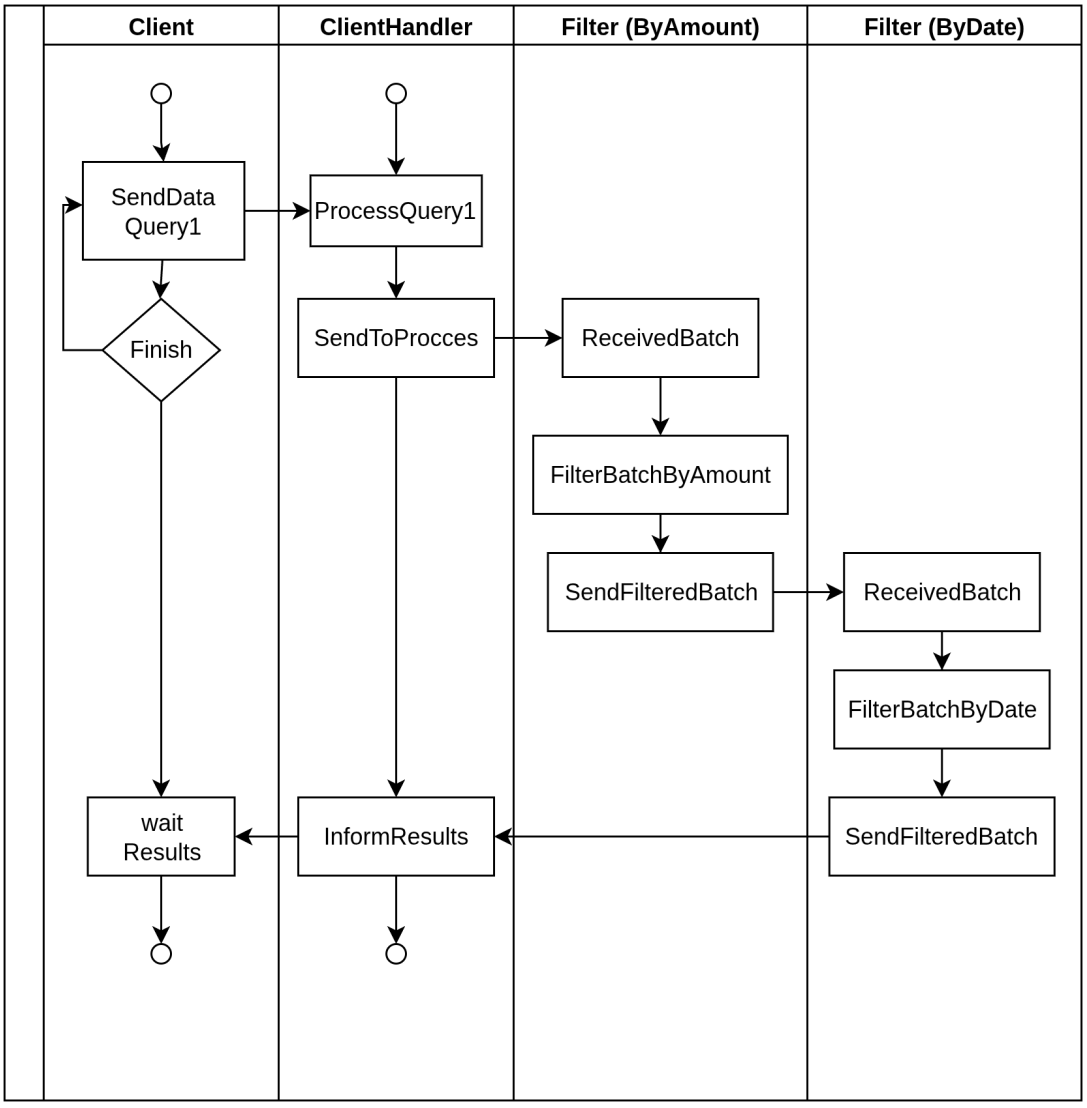
Se presenta el grafo dirigido acíclico que expone el flujo lógico de la aplicación a gran escala:



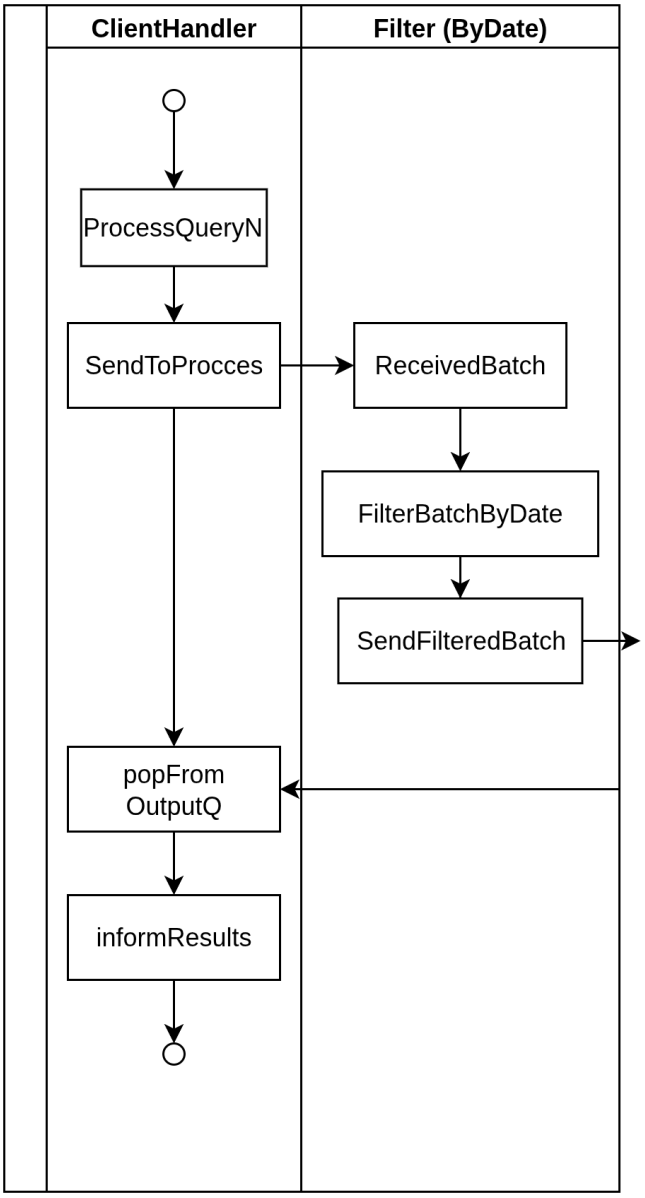
4. Vista de procesos

Diagramas de actividad

Se presenta a continuación la representación del procesamiento íntegro de la query número 1 y el flujo que tiene la información desde el cliente hasta obtener el resultado.

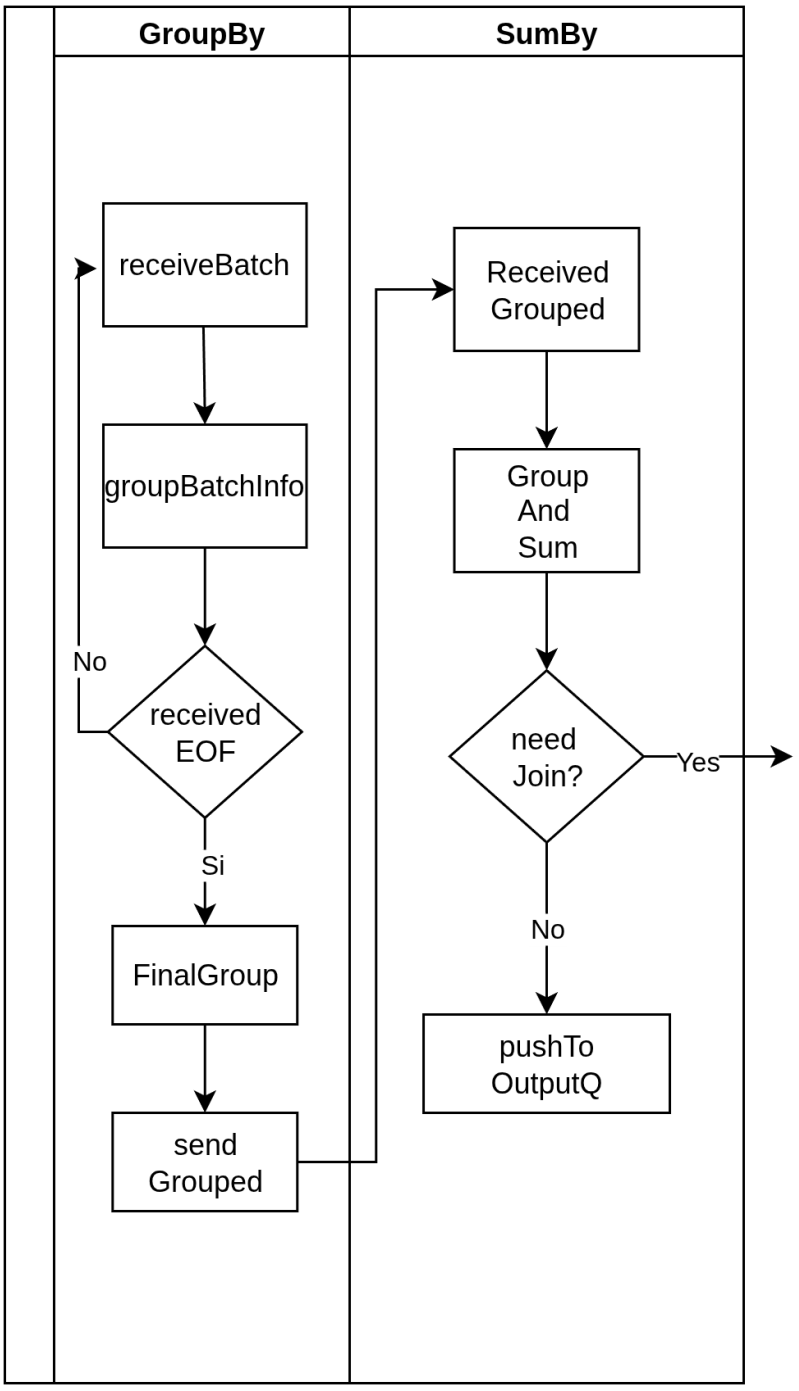


Por otro lado, se agrupa el comportamiento inicial para cada una de las consultas en donde, el Nodo **ClientHandler** se comunica inicialmente con un Nodo **Filter**, diferenciándose solamente en la Query número 1, donde el primer filtro no es una fecha si no una cantidad. Una vez pasada la información por la capa de los filtros se continúa con el procesamiento en los siguientes nodos.

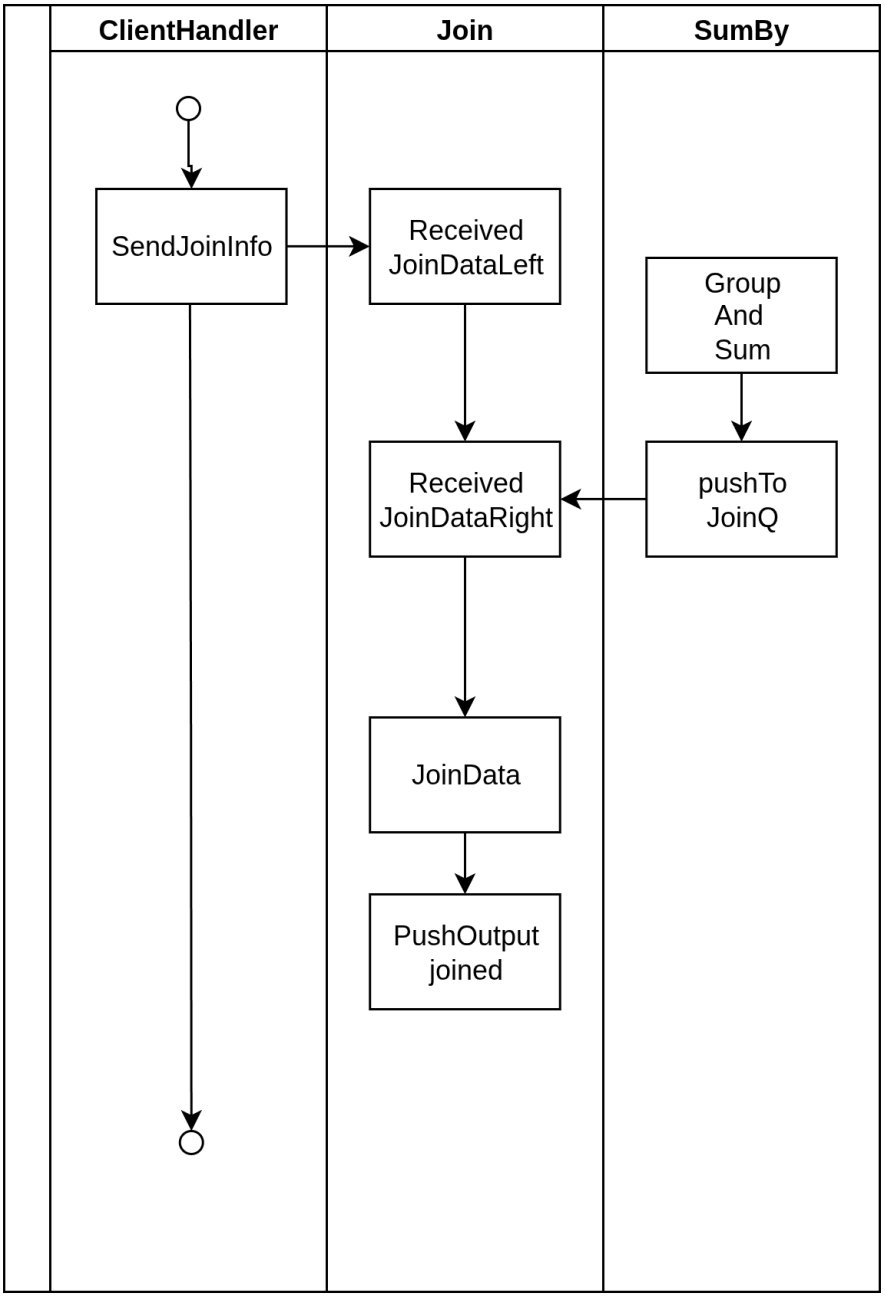


Correspondientes con todas las consultas con excepción de la primera, después de una primera etapa de filtrado se procede a realizar una etapa de doble agrupamiento, en donde en un principio se hace un "Agrupamiento simple" en donde los se separan los batches y se juntan por una condición (véase por ejemplo una agrupación por StoreId) y se prosigue a realizar una agrupación más compleja en donde se colapsan las dimensiones de los datos y donde se junta la información según dos categorías

diferentes, es decir, se considera la agrupación realizada en el paso anterior y sobre los datos agrupados se procede a realizar una segunda agrupación aplicando una transformación en los datos, véase como un ejemplo la consulta número 2, donde se agrupa por mes y año en un primer paso y posteriormente se agrupa la información según el Id del item, sumando en cada caso la cantidad y las ganancias de cada registro.

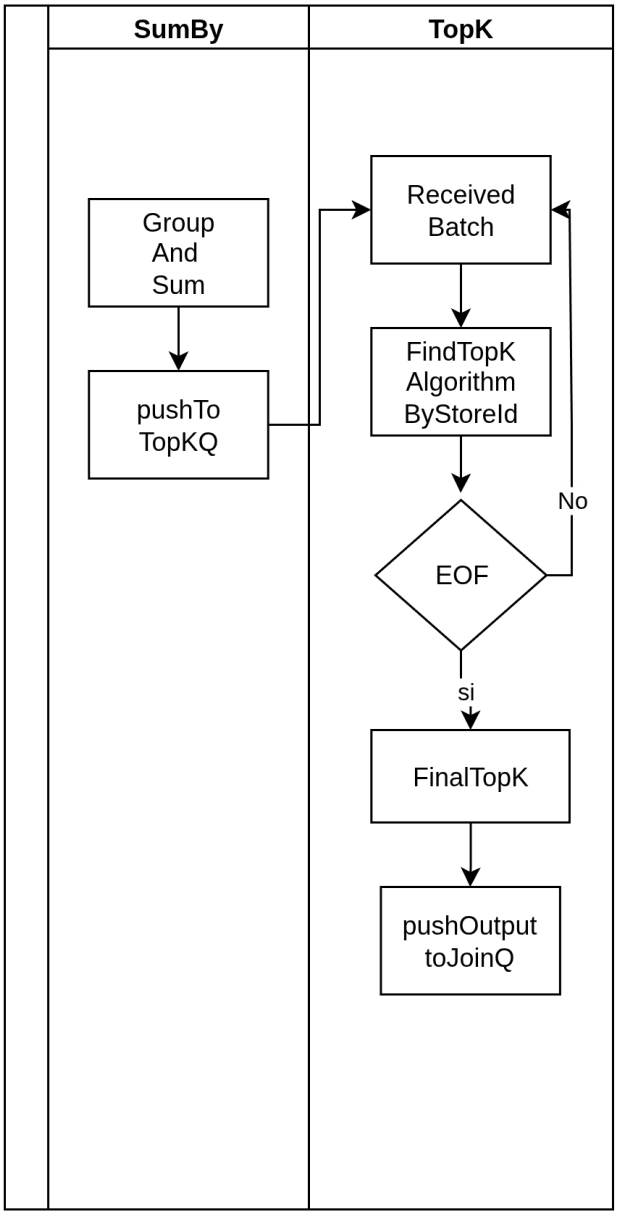


Por otro lado, el comportamiento de las operaciones de junta resulta homogéneo, necesitando en un principio la información cruda proveniente desde el cliente la cual recibe desde el principio del inicio de la transacción para posteriormente esperar el resultado del procesamiento de otros nodos (Véase el proceso GroupBy y SumBy). Una vez el nodo realice la junta entre los dos conjuntos de datos que posee procede a pushear a la cola de resultados para ser despachados al cliente.



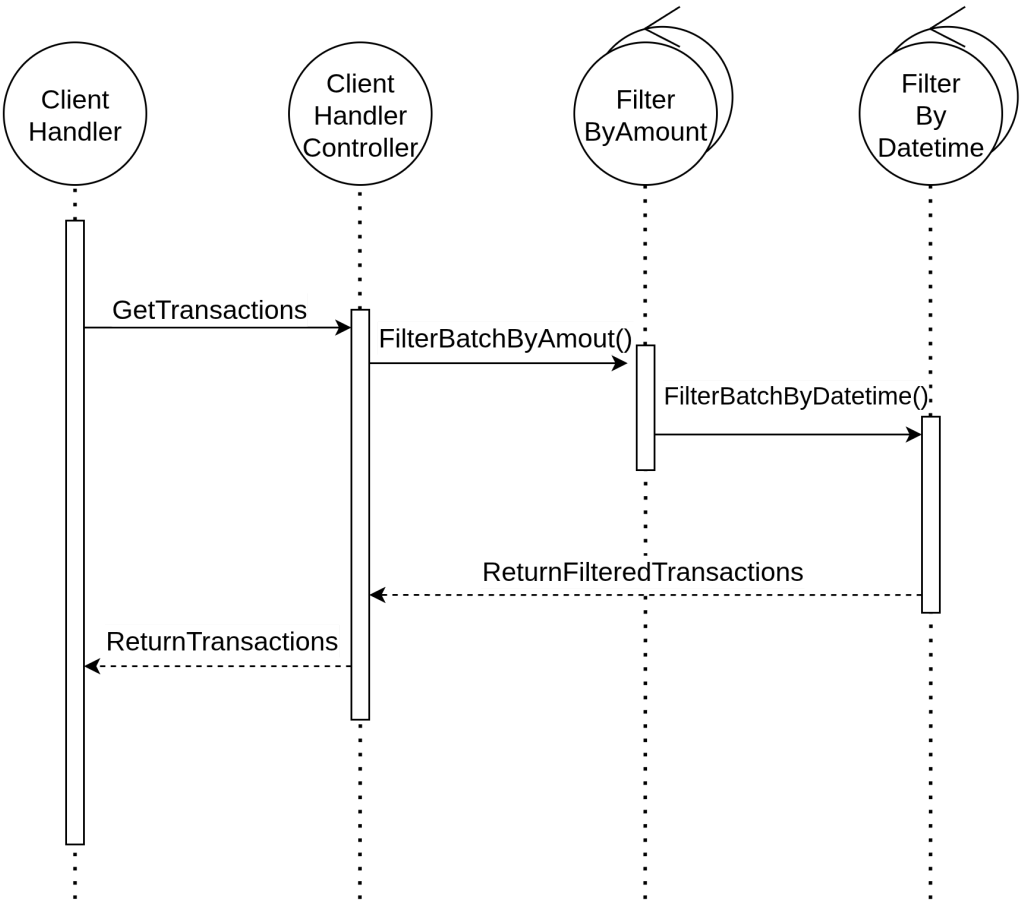
Finalmente, se grafica el comportamiento esperado para realizar las operaciones Top, este comportamiento solo se da en la consulta número 4 y el nodo que realice esta

operación siempre tendrá provista su información por los nodos SumBy que realicen la operación previa.

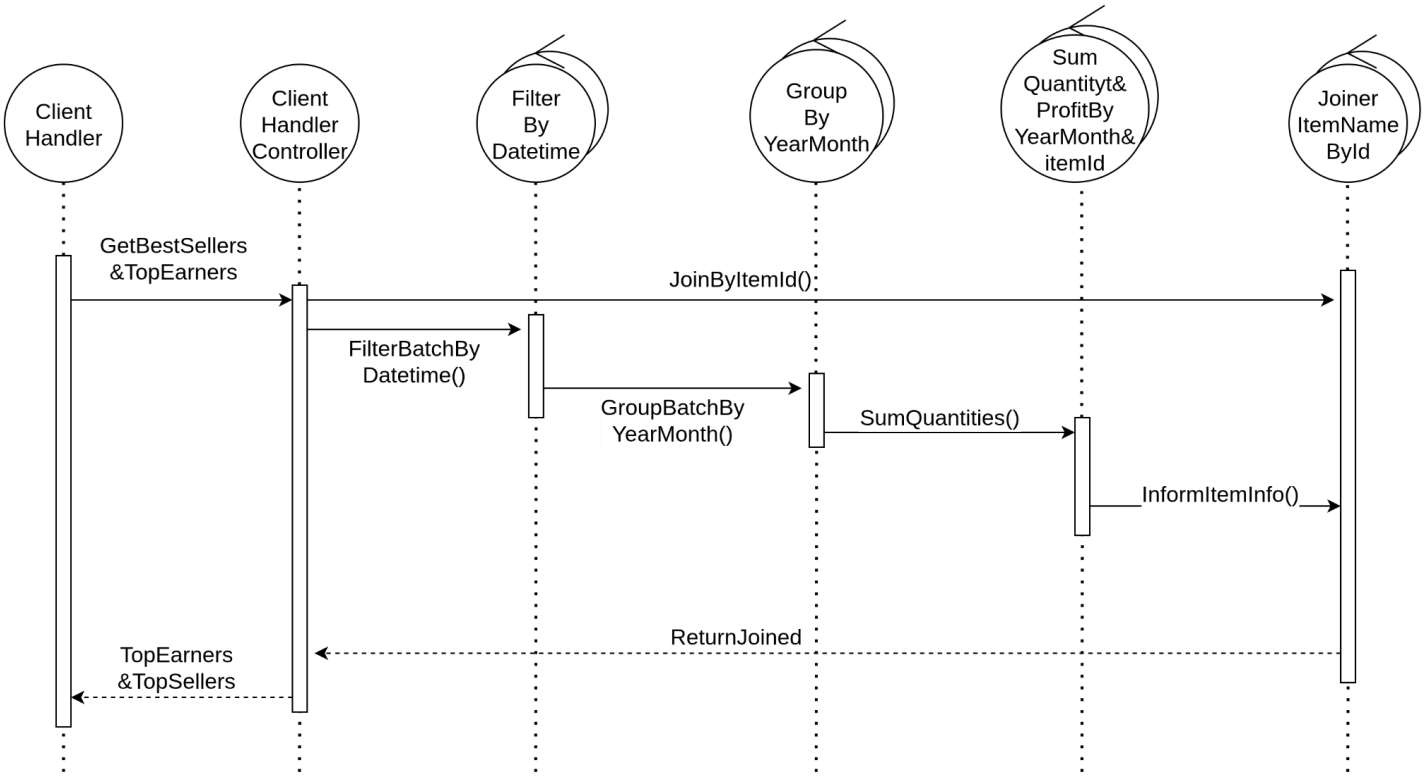


Diagramas de secuencia

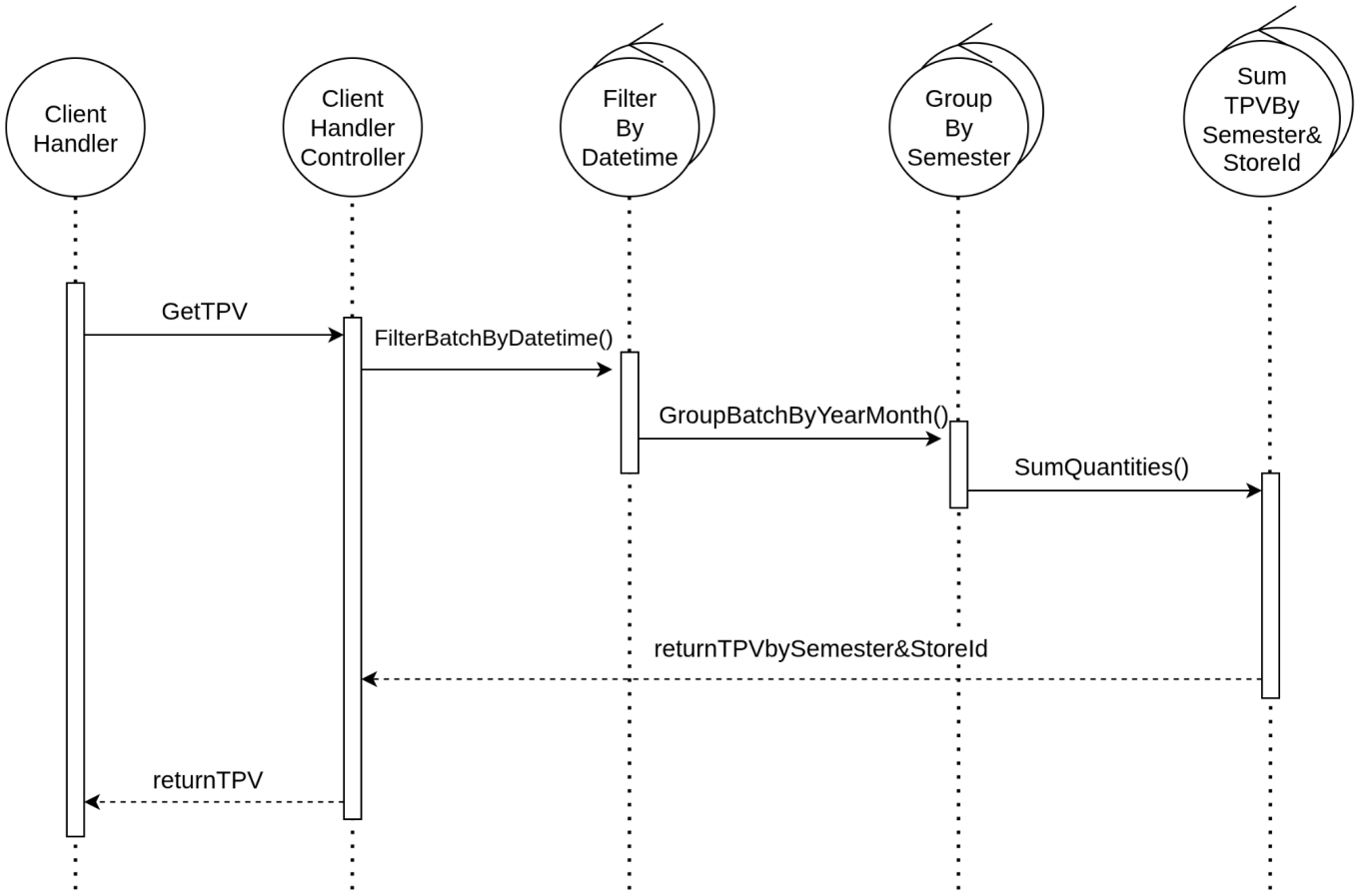
A continuación, se presentan los diagramas de secuencia que muestran el flujo y la interacción de los nodos para llevar a cabo el desarrollo de todas las operaciones:



Consulta 1

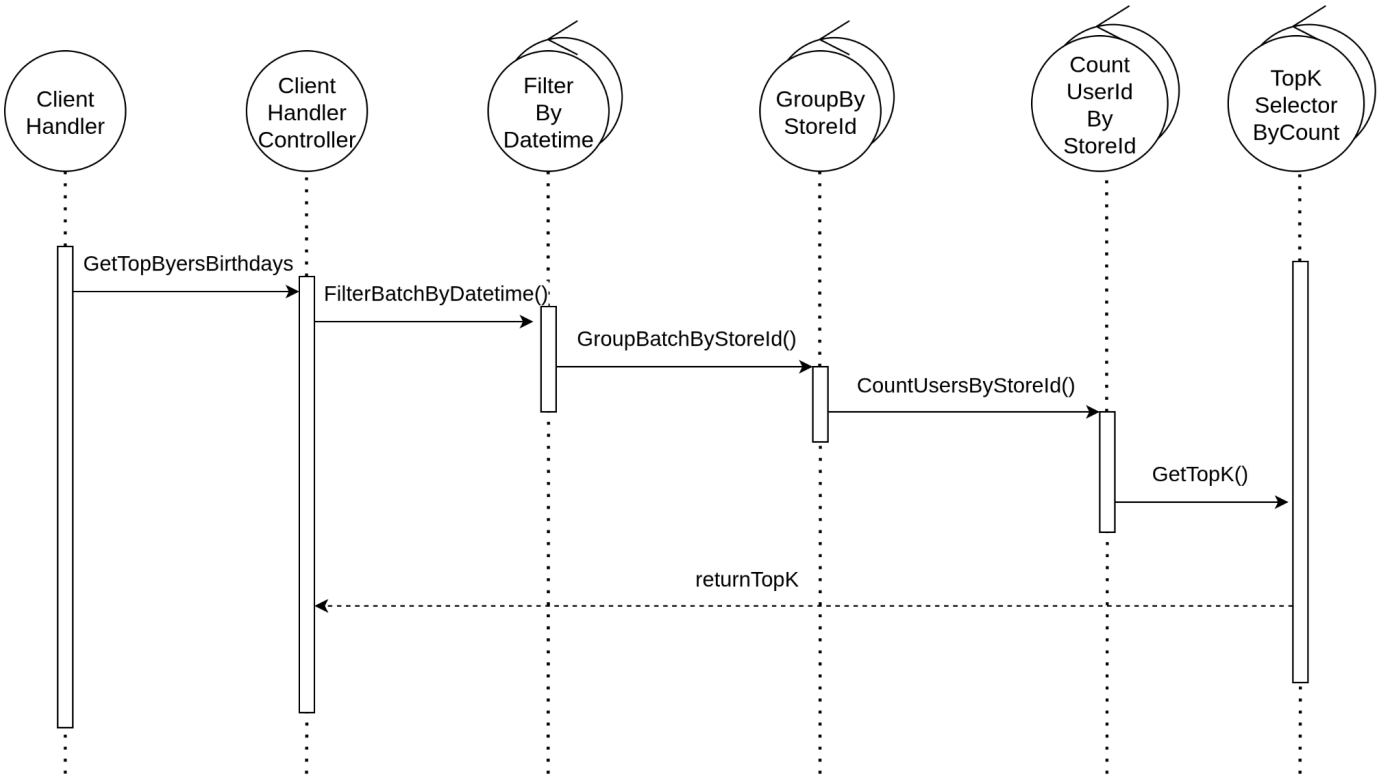

Client Handler
Client Handler Controller
Filter By Datetime
Group By YearMonth
Sum Quantityt& ProfitBy YearMonth& itemId
Joiner ItemName ById
GetBestSellers &TopEarners
JoinByItemId()
FilterBatchBy Datetime()
GroupBatchBy YearMonth()
SumQuantities()
InformItemInfo()
ReturnJoined
TopEarners &TopSellers

Consulta 2

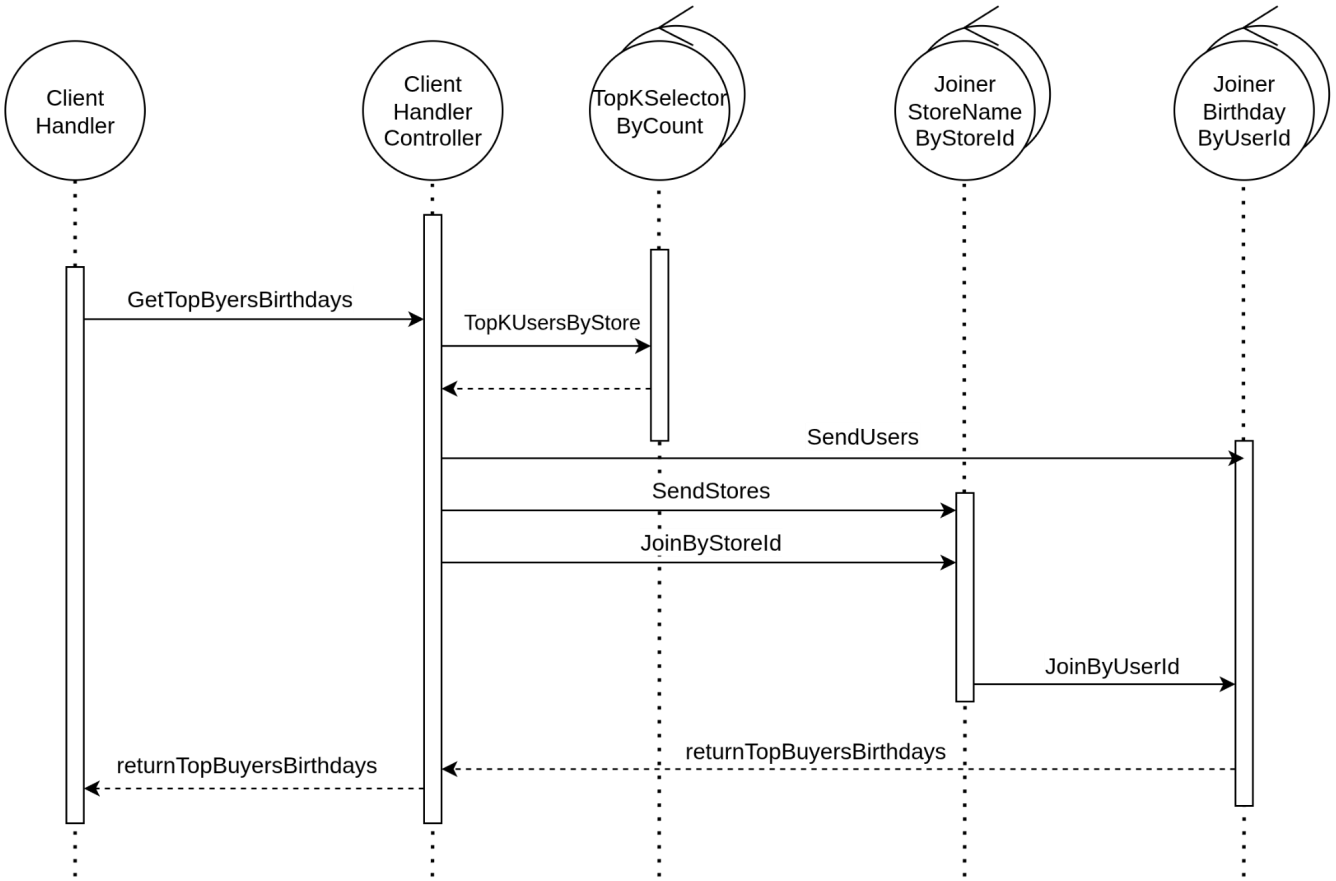

Client Handler
Client Handler Controller
Filter By Datetime
Group By Semester
Sum TPVBy Semester& StoreId
GetTPV
FilterBatchByDatetime()
GroupBatchByYearMonth()
SumQuantities()
returnTPVbySemester&StoreId
returnTPV

Consulta 3

Para la última consulta se particionan los gráficos para mayor claridad:



Consulta 4.1


Client Handler
Client Handler Controller
TopKSelector ByCount
Joiner StoreName ByStoreId
Joiner Birthday ByUserId
GetTopByersBirthdays
TopKUsersByStore
SendUsers
SendStores
JoinByStoreId
JoinByUserId
returnTopBuyersBirthdays
returnTopBuyersBirthdays

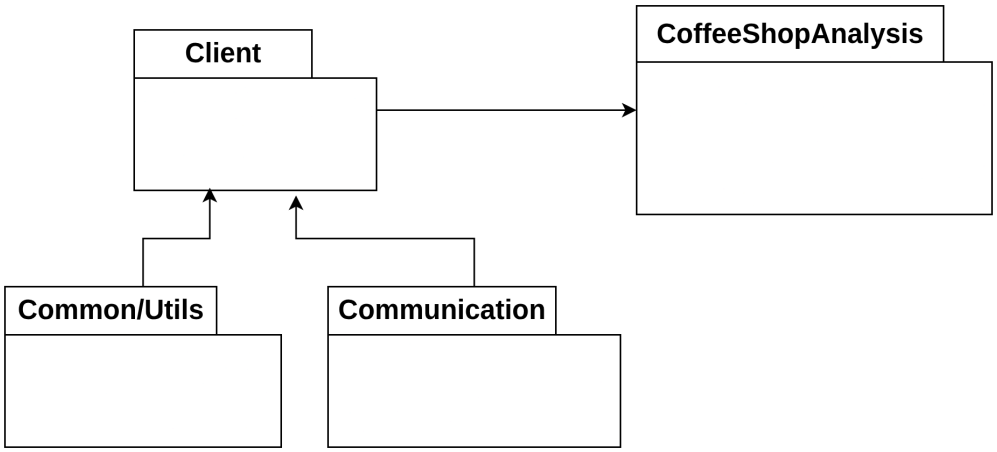
Consulta 4.2

5. Vista de desarrollo

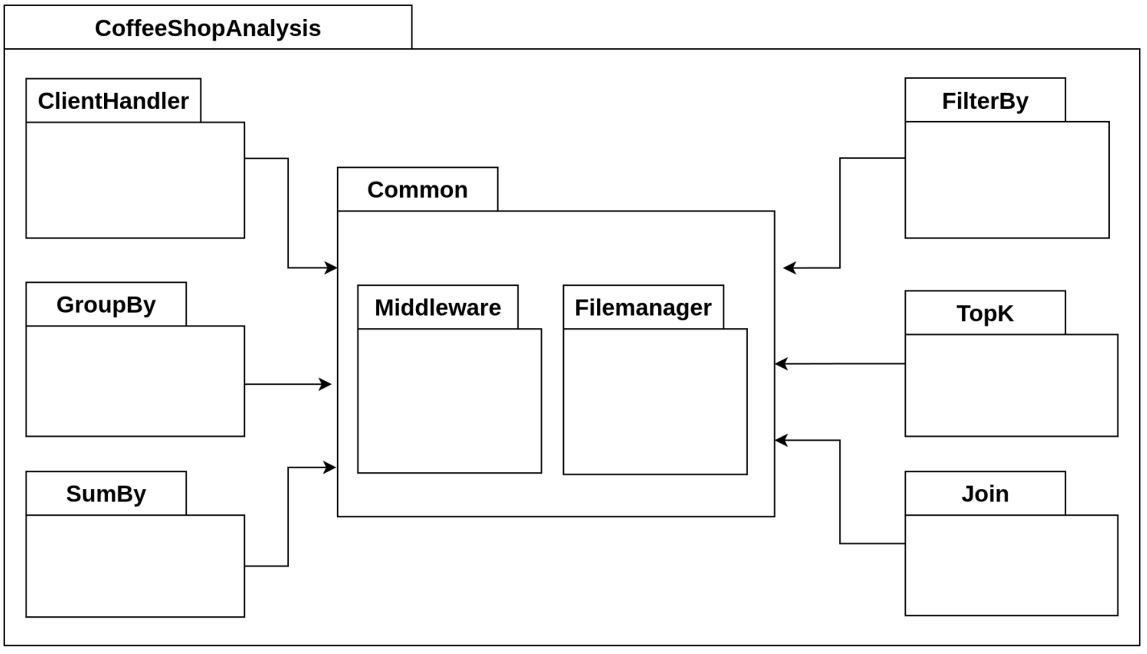
Diagrama de paquetes

Los paquetes requeridos para desarrollar el proyecto se pueden dividir en dos grandes grupos:

En un principio se tiene el cliente, externo a nuestro sistema distribuido el cual se encarga de subir la información necesaria para poder realizar las consultas:



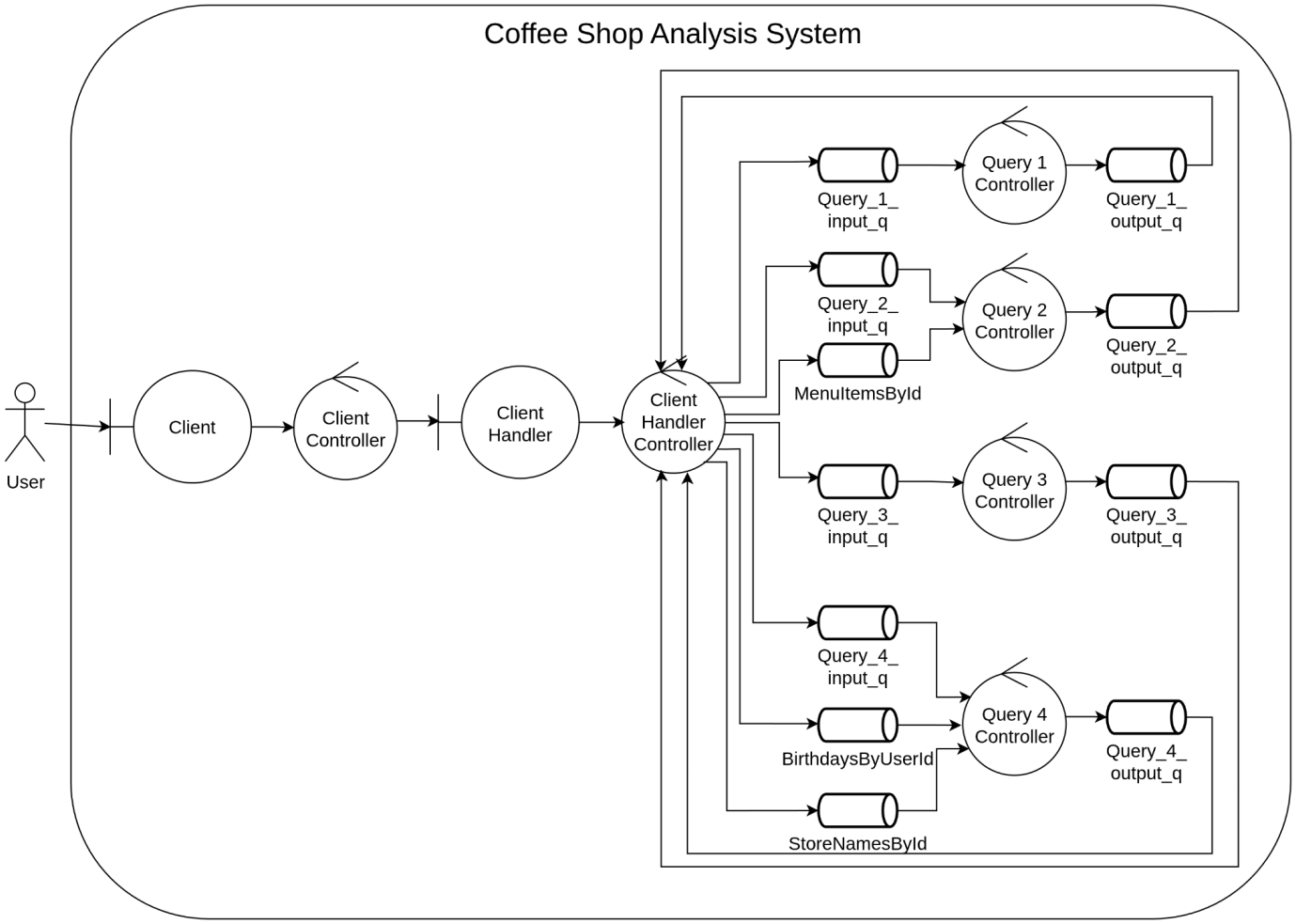
Finalmente se tiene el diagrama de paquetes correspondiente al sistema distribuido:



6. Vista Física

Diagrama de robustez

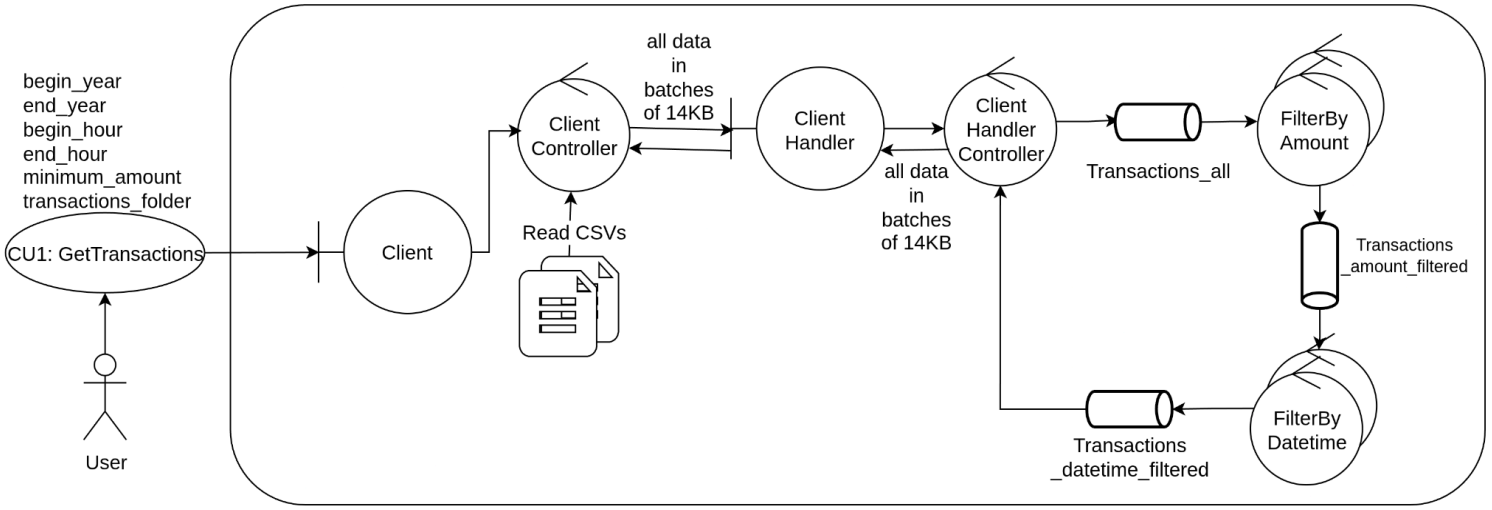
Se presenta un diagrama de robustez general del sistema distribuido Coffee Shop Analysis System, abstrayendo el flujo interno de cada query que luego se mostrará con más detalle.



En este diagrama se observa el boundary Client y el ClientController que conforman el cliente que ejecutará el envío de datos al servidor. El cliente interactúa con el boundary ClientHandler del lado del servidor, por el cual se conecta mediante un socket TCP para el envío de datos en batches de 14KB (cantidad seleccionada teniendo en cuenta el MTU histórico y la ventana de slow start de TCP).

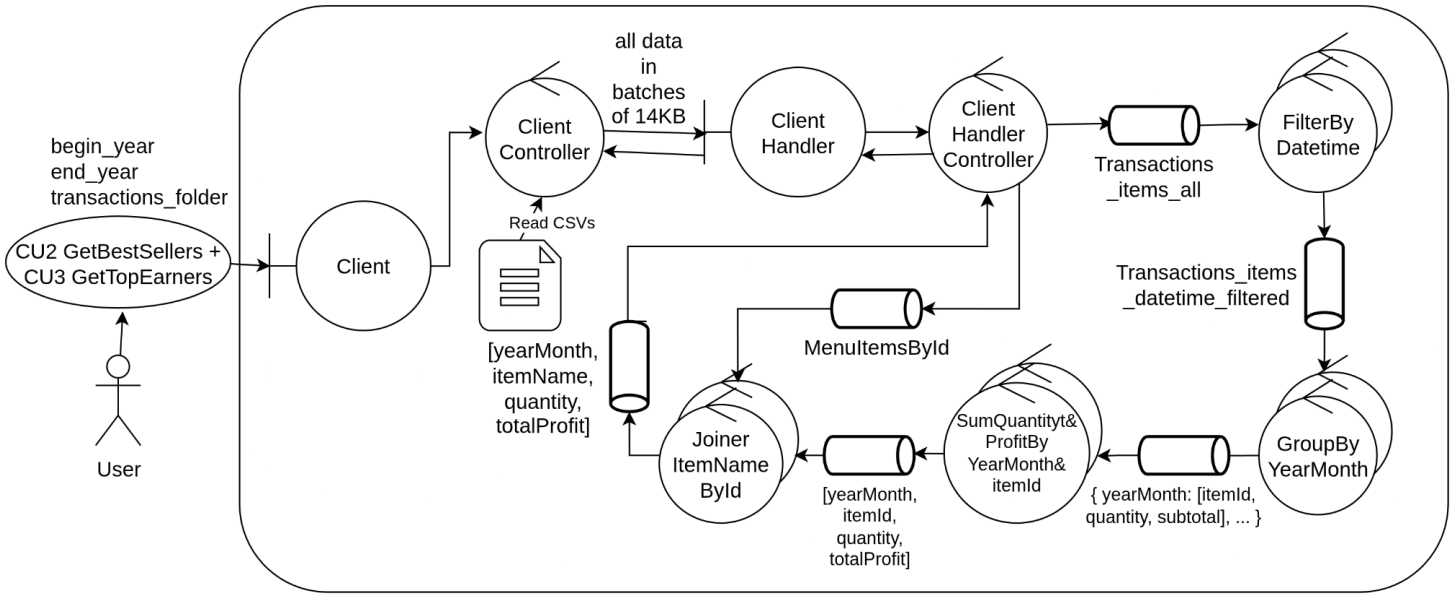
Una vez llegada toda la data al ClientHandlerController, este envía la información adecuada mediante el middleware a las queues de input de cada query, esbozadas en el diagrama anterior de manera abstracta para detallar su funcionamiento interno más adelante. Una vez terminado el procesamiento de las queries, cada uno deposita el output en una queue de output del cual el ClientHandlerController sacará los resultados finales para enviar al Client mediante el socket TCP.

Se procede a mostrar el diagrama de robustez en más detalle para la **query número 1** con su caso de uso asociado:



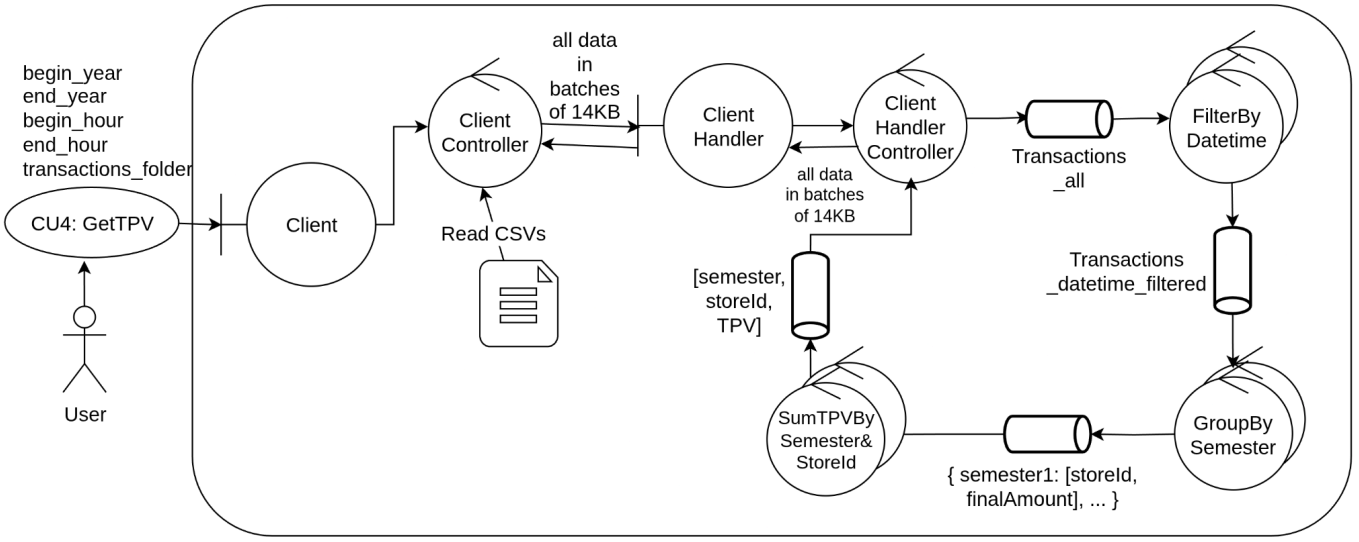
En este diagrama de la query 1 se ve cómo todas las transacciones se insertan en una cola, luego se aplica un filtro que puede dejar pasar o no a la transacción, en este caso considerando un mínimo del costo final de la transacción. De haber pasado el filtro, éste deposita la transacción en una cola intermedia para que el siguiente filtro que opera revisando que la transacción esté entre las fechas solicitadas. Finalmente se insertan los elementos que cumplen la condición en una queue de output final.

Se procede a mostrar el diagrama de robustez en más detalle para la **query número 2** con sus dos casos de uso asociados:



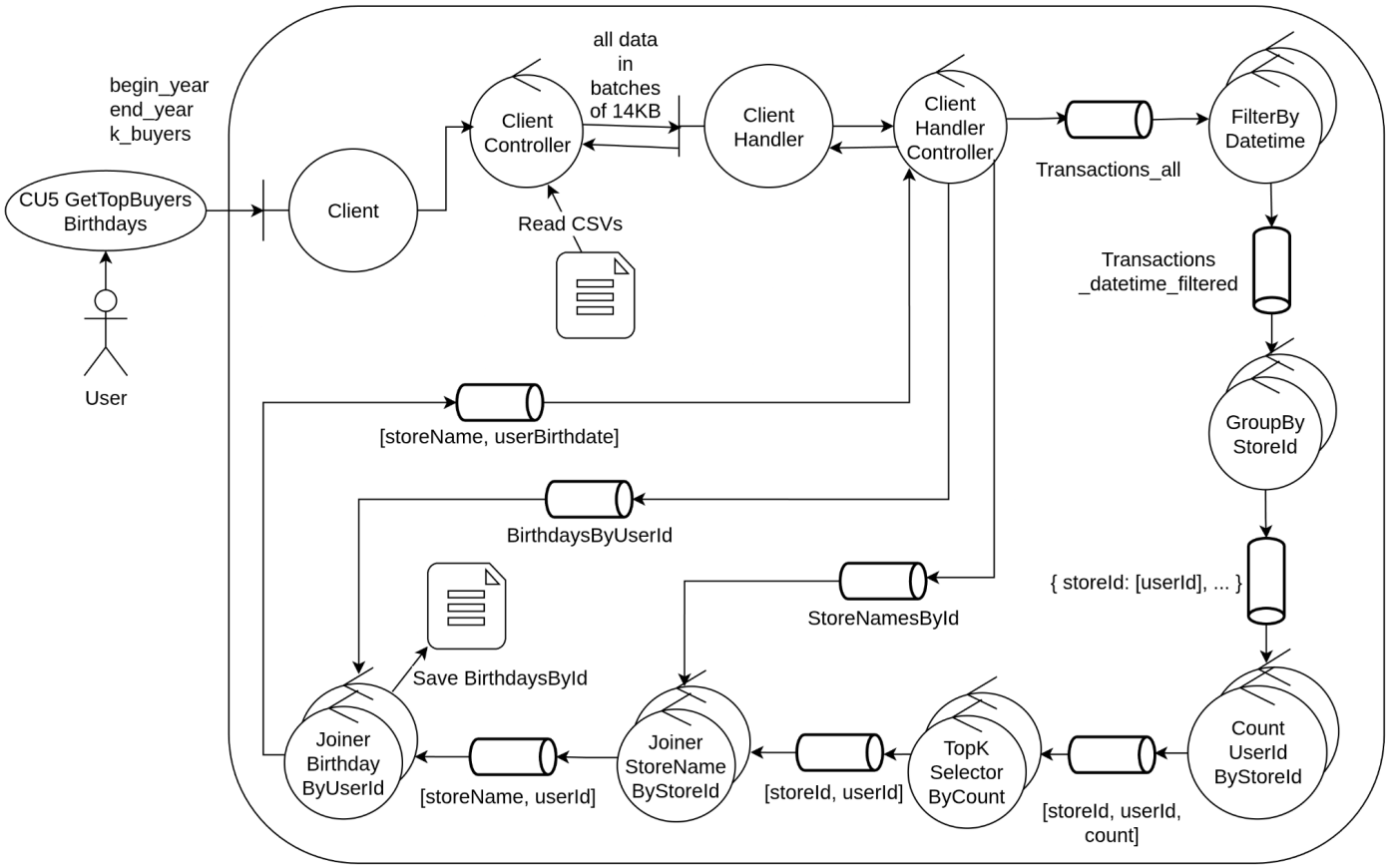
En este diagrama de la query 2 se ve cómo todos los ítems-transacciones se insertan en una cola, luego se aplica un filtro revisando que el ítem-transacción esté entre las fechas solicitadas. De haber pasado el filtro, éste deposita la transacción en una cola intermedia para que los nodos de agrupamiento agrupen los ítems-transacciones por mes de un año determinado (e.g. 2025-01 es distinto de 2024-01).
Cada nodo de agrupamiento toma un mensaje de la cola (que puede contener uno o varios ítem-transacción) y hace una agrupación interna, luego toma otro mensaje y agrupa. Eventualmente, se llega al último mensaje (mensaje EOF), el nodo que procese ese mensaje será el que reconcilie los resultados parciales de los demás nodos comunicándose mediante el middleware y depositará el output en la cola.
Luego, los nodos de suma procesarán de la misma manera los datos agrupados (resultados parciales y luego consolidación de resultados) y los depositarán en la cola con la información sumada. Se eligió combinar las operaciones de suma, es decir, sumar quantity y totalProfit a la vez, en lugar de tener dos etapas de suma, ya que se aprovecha la localidad de la información porque ambas operaciones comparten las mismas keys en el CSV.
Una vez realizada la suma, para intercambiar el nombre por el ID del ítem, el nodo joiner consume los mensajes de la cola de MenuItems para reemplazar el ID y pushear el resultado en la cola de output.

Se procede a mostrar el diagrama de robustez en más detalle para la **query número 3** con su caso de uso asociado:



En este diagrama de la query 3 se ve cómo todas las transacciones se insertan en una cola, luego se aplica un filtro revisando que el ítem-transacción esté entre las fechas solicitadas. De haber pasado el filtro, éste deposita la transacción en una cola intermedia para que los nodos de agrupamiento agrupen las transacciones por semestre (2024-H1, 2024-H2, 2025-H1, etc).
Luego los nodos de suma ejecutan el proceso de suma sobre el finalAmount generando el TPV (Total Payment Value), con el mismo funcionamiento como se detalló anteriormente, y luego de consolidar el resultado final dejan el resultado final en la cola de outputs.

Se procede a mostrar el diagrama de robustez en más detalle para la **query número 4** con su caso de uso asociado:



En este diagrama de la query 4 se ve cómo todas las transacciones se insertan en una cola, luego se aplica un filtro revisando que el ítem-transacción esté entre las fechas solicitadas. De haber pasado el filtro, éste deposita la transacción en una cola intermedia para que los nodos de agrupamiento agrupen las transacciones por ID de la sucursal.

Luego se ejecuta un conteo de la cantidad de veces que aparecen los usuarios vinculados a una determinada sucursal. Esto en esencia es hacer el mismo proceso de suma donde cada entrada tiene una quantity de 1 (uno), y se cuentan los usuarios porque esto representa contar la cantidad de veces que el usuario ejecutó una transacción en esa sucursal.

Los resultados anteriores son colocados en una cola intermedia donde los nodos TopK consumen un mensaje (que puede tener uno o varios ítems), encuentran el top k resultados ordenados por la cantidad de ocurrencias count, luego toman otro mensaje y actualizan ese top k en base a la información previa y la nueva. Finalmente, el que procese el último mensaje (marcado con un EOF) va a ser el que inicie la consolidación de resultados ejecutando el algoritmo TopK sobre los resultados parciales.

Una vez encontrados los TopK resultados estos se colocan en una cola intermedia donde todos los nodos Joiner utilizarán ese resultado para procesar por batches todos los usuarios hasta agotar los usuarios. Sería análogo a la búsqueda de los itemNames previamente, pero con el input intercambiado (en este caso el input de gran tamaño es el

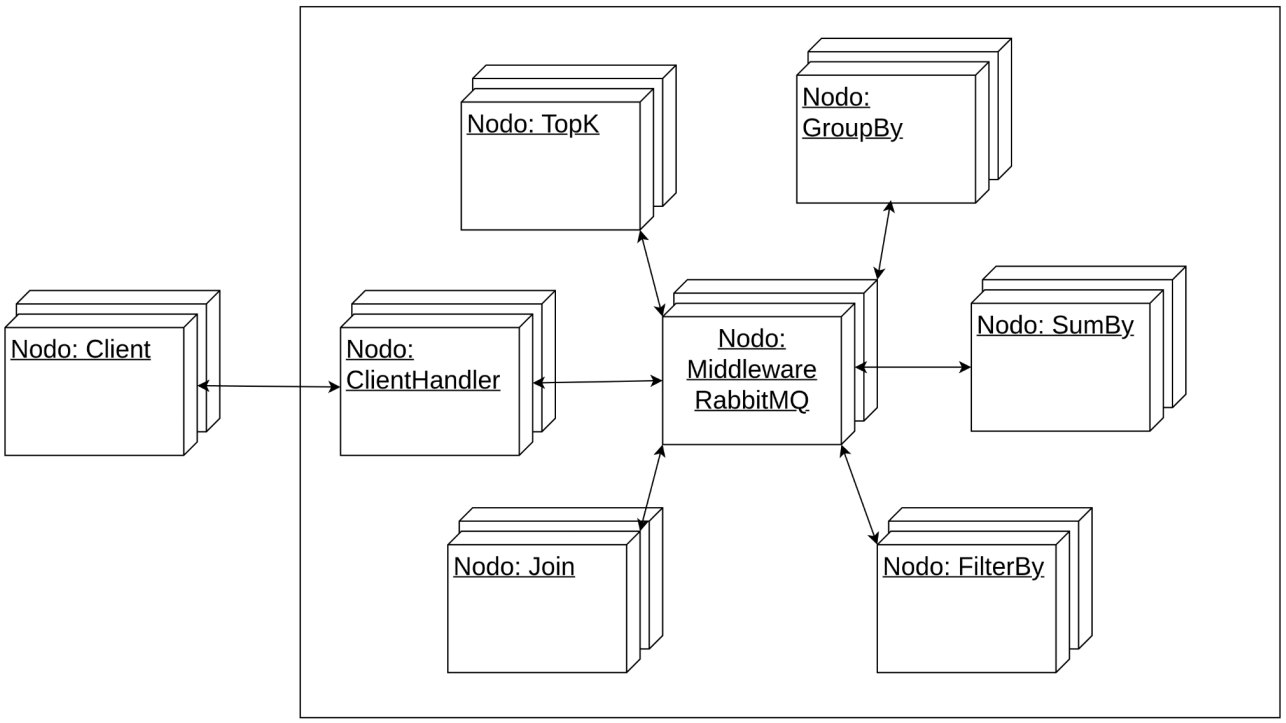
que viene dado por el ClientHandlerController y el input pequeño es el que pasa por el pipeline).

Proceso de consenso para batch terminado

Para los nodos Filter y Joiner es necesario un mecanismo de sincronización final para asegurar que el mensaje de fin de batch (EOF) para esa determinada etapa en el pipeline sea enviada de manera correcta siempre al final, y sin un mensaje pendiente de procesamiento detrás.
Para cumplir este objetivo se optó por un enfoque de consenso entre los nodos de la etapa del pipeline (e.g. consenso entre los nodos de FilterByDatetime para un determinado clientId). Una vez alcanzado el consenso, se emite el EOF para que continúe el flujo de datos.

Diagrama de despliegue

Se presenta el diagrama de despliegue del sistema distribuido, mostrando explícitamente las posibilidades de escalabilidad horizontal con multicomputing:



7. División de tareas

Se provee una distribución de tareas de alto nivel para el desarrollo del proyecto:

Tareas	Integrante Responsable
Implementación del módulo de **ingesta de datos en el cliente** que lea y prepare mensajes (batch) para ser enviados al servidor	José
Implementación del **ClientHandler** y el **routing** de información para ejecutar todas las queries.	José
Implementación del pipeline de procesamiento para la **consulta 1** que involucra: **FilterByAmount** y **FilterByDatetime**	Gabriel
Implementación de parte del pipeline de procesamiento para la **consulta 2** que involucra: **GroupByYearMonth** y **SumQuantityt&ProfitByYearMonth&itemId**	Ramiro
Implementación de parte del pipeline de procesamiento para la **consulta 2** que involucra: **JoinerItemNameById**	Gabriel
Implementación de parte del pipeline de procesamiento para la **consulta 3** que involucra: **GroupBySemester** y **SumTPVBySemester&StoreId**	Gabriel
Implementación de parte del pipeline de procesamiento para la **consulta 4** que involucra: **GroupByStoreId, CountUserIdByStoreId** y **TopKSelectorByCount**	José
Implementación de parte del pipeline de procesamiento para la **consulta 4** que involucra: **JoinerStoreNameByStoreId** y **JoinerBirthdayByUserId**	Ramiro
Implementación de **tolerancia básica de fallos** en workers (reintentos por falla de conexión, graceful quit con SIGTERM)	Ramiro

8. Referencias

- *RabbitMQ Documentation* (s. f.). RabbitMQ. Recuperado en septiembre de 2025, de https://www.rabbitmq.com/docs