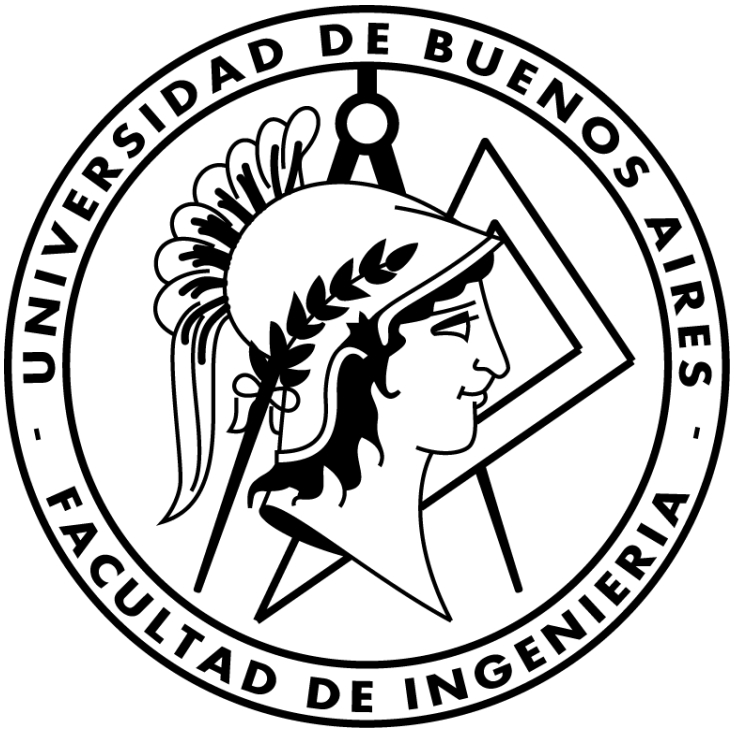


Documento de Arquitectura - Steam Analysis

Nehuén Cabibbo

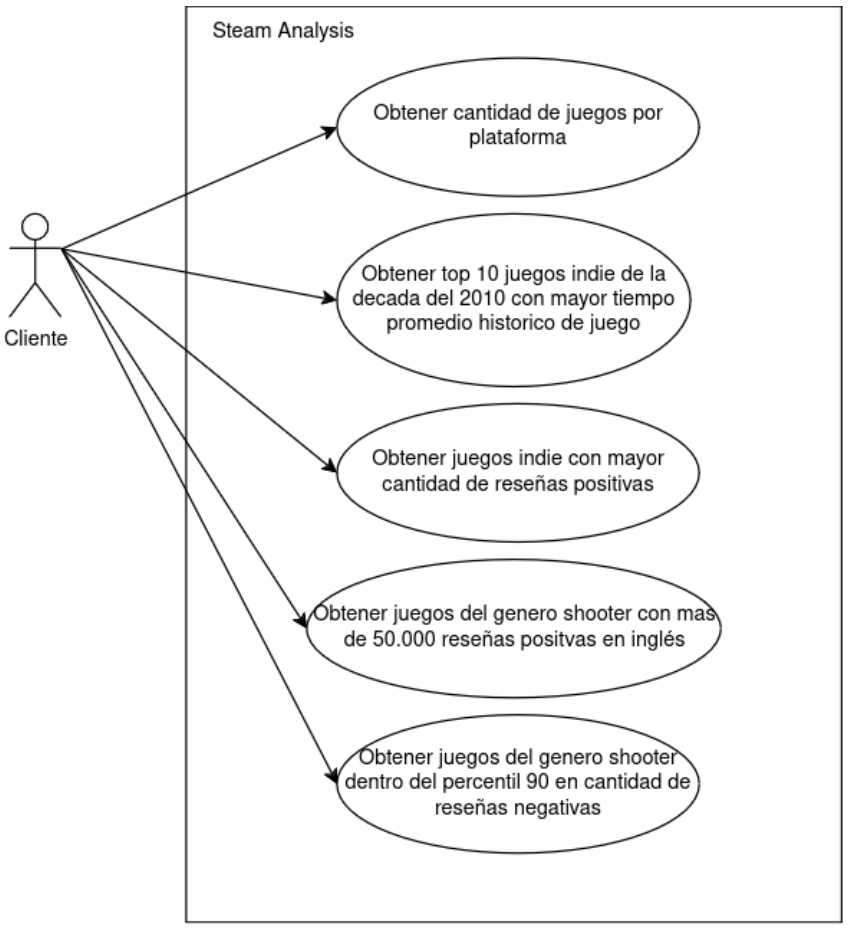
Clemente Avila

Juan Ignacio Pérez Di Chiazza

Índice

Casos de uso

- Como usuario quiero poder:
    1. Obtener la cantidad de juegos soportados en cada plataforma (Windows, Linux, MAC)
    2. Obtener el nombre de los juegos top10 del género "Indie" publicados en la década del 2010 con más tiempo promedio histórico de juego
    3. Obtener el nombre de los juegos top 5 del género "Indie" con más reseñas positivas
    4. Obtener el nombre de juegos del género "shooter" con más de 50.000 reseñas positivas en idioma inglés
    5. Obtener el nombre de juegos del género "shooter" dentro del percentil 90 en cantidad de reseñas negativas



Vista Lógica

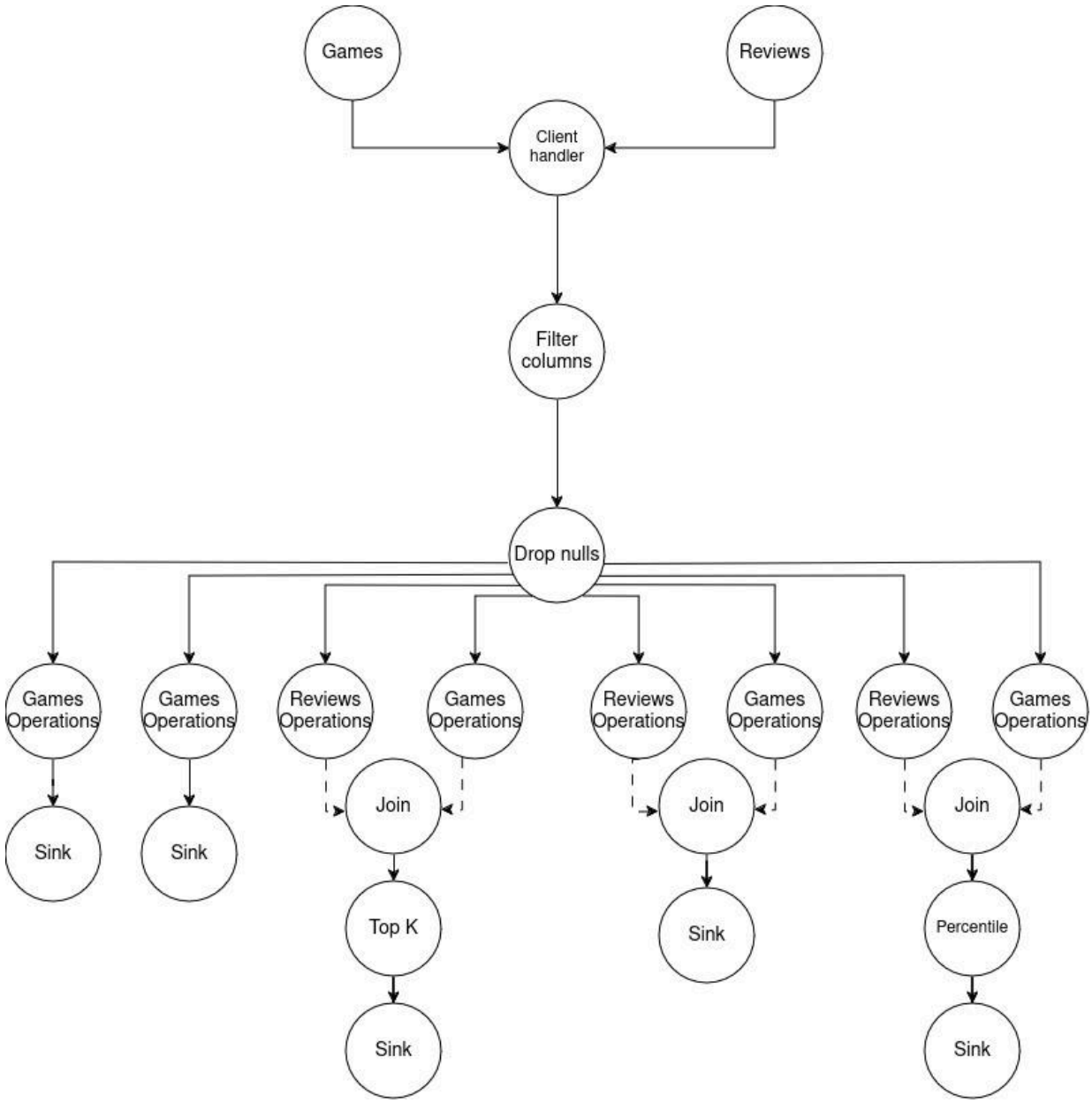
General

Se muestra un DAG del sistema completo para entender el flujo de datos que se da para resolver cada query. Como la resolución de cada query puede pensarse como un pipeline separado, también en la siguiente sección se incluye un DAG query para mayor entendimiento.

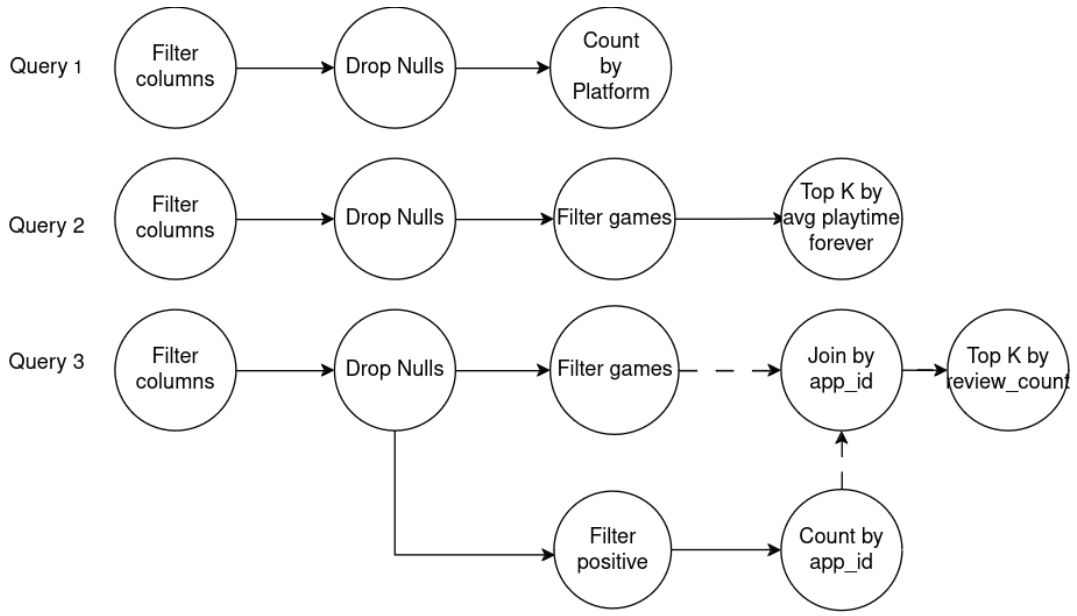
Observaciones

En aquellos lugares donde se utilicen dos flechas punteadas (que salen desde distintos nodos) y se juntan en uno, como es en los casos de join, lo que se quiere explicitar es que un nodo deberá enviar la totalidad de los datos por su parte para que se puedan empezar a recibir los datos del otro.

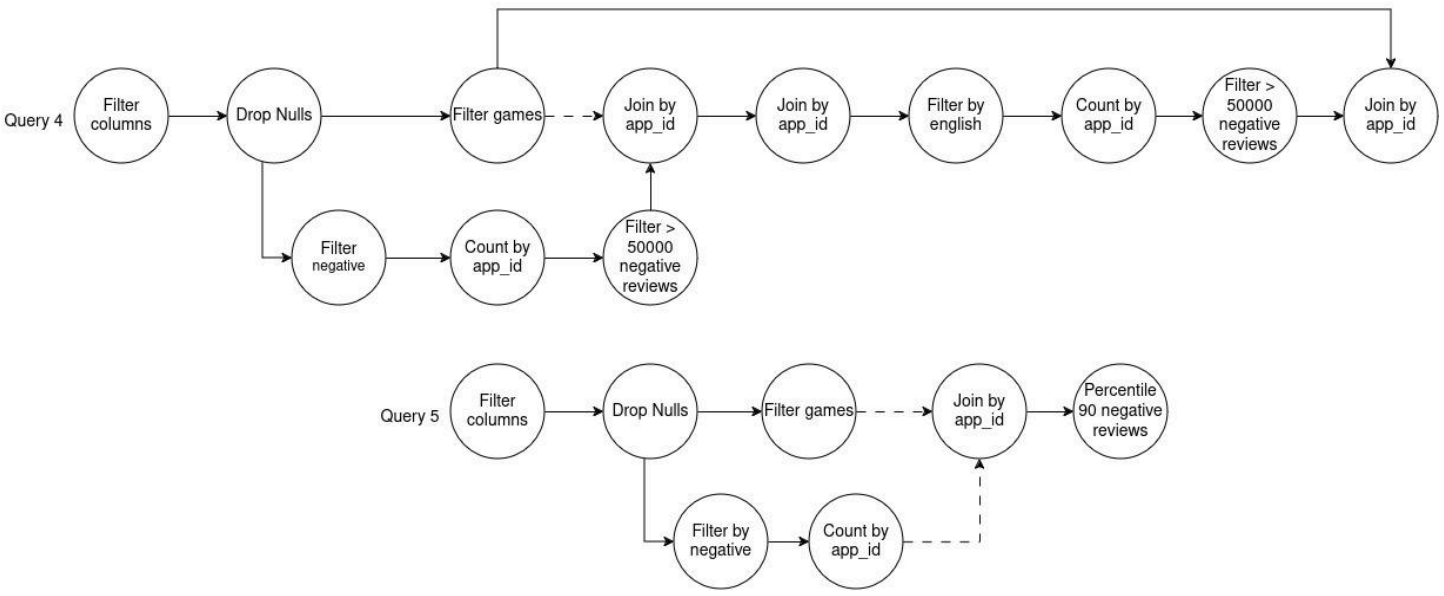
Por otro lado, en aquellos nodos que se utilice una flecha punteada (que sale de un único nodo), se busca representar que se debe obtener la totalidad de los datos por parte del mismo.


Games
Reviews
Client handler
Filter columns
Drop nulls
Games Operations
Games Operations
Reviews Operations
Games Operations
Reviews Operations
Games Operations
Reviews Operations
Games Operations
Sink
Sink
Join
Join
Join
Top K
Sink
Percentile
Sink
Sink

Data Pipeline Queries 1, 2 y 3



Data Pipeline Queries 4 y 5

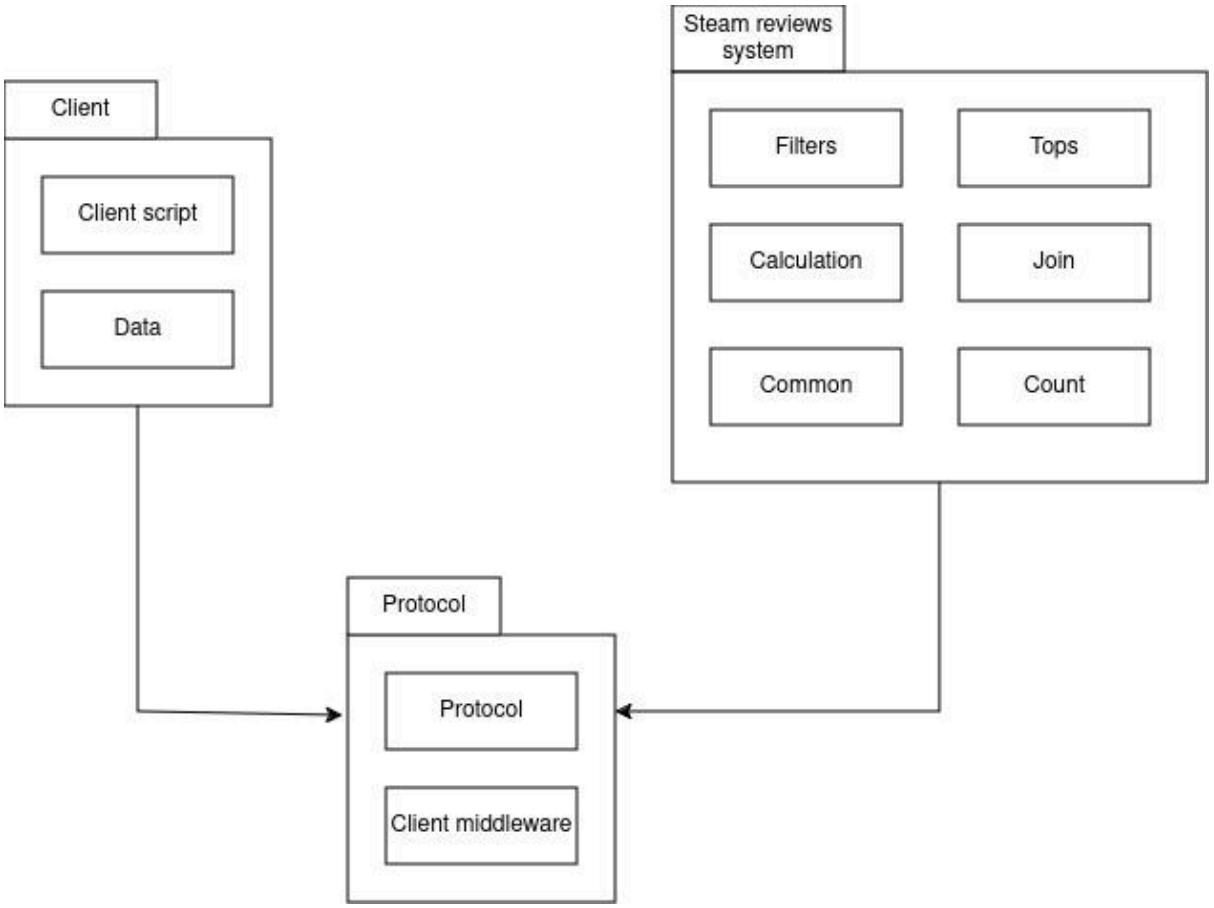


Vista de desarrollo

Diagrama de paquetes

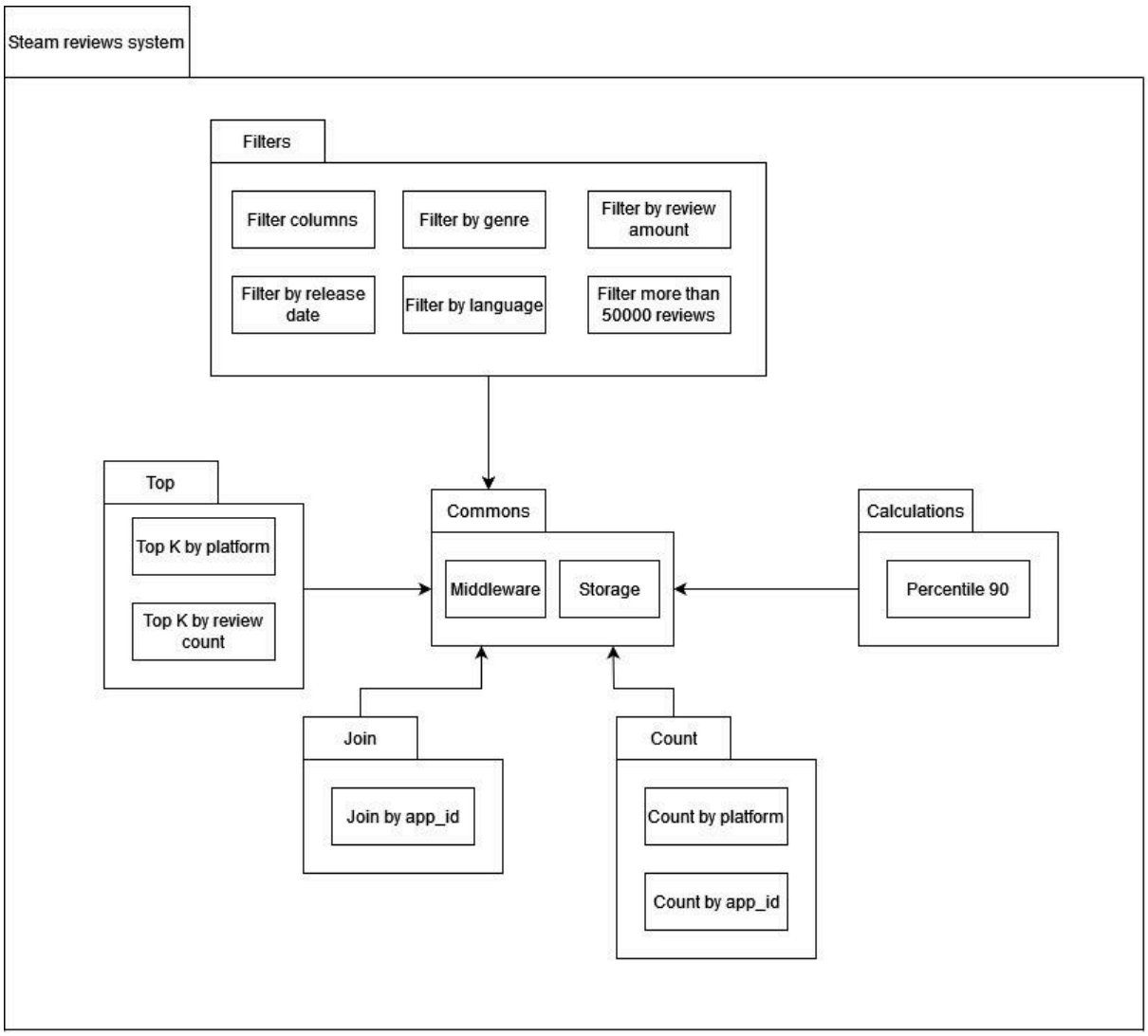
Se expone el diagrama de paquetes para denotar la relación entre ciertos elementos del sistema de similar funcionalidad, y a nivel conjunto como estos se relacionan con otros grupos de elementos.

Con este primer diagrama se busca denotar las dos entidades que existen, el cliente y el *Steam reviews system*. Con este se busca denotar como ambos se relacionan utilizando un paquete llamado *protocol*, que encapsula la comunicación entre ambos.



En este siguiente diagrama se busca entrar en detalle en los componentes del *Steam reviews system*, y cómo se relacionan entre ellos.

A destacar está el paquete *common*, el cual contiene el *middleware* a utilizar para comunicación, al igual que un elemento *storage*, el cual se encarga de encapsular toda la lógica relacionada a la persistencia del sistema.


Steam reviews system
Filters
Filter columns
Filter by genre
Filter by review amount
Filter by release date
Filter by language
Filter more than 50000 reviews
Top
Top K by platform
Top K by review count
Commons
Middleware
Storage
Calculations
Percentile 90
Join
Join by app_id
Count
Count by platform
Count by app_id

Vista de procesos

Diagrama de Actividad

Diagrama de actividad de la comunicación cliente-servidor

El siguiente diagrama se muestra la comunicación a grandes rasgos entre el cliente y el servidor

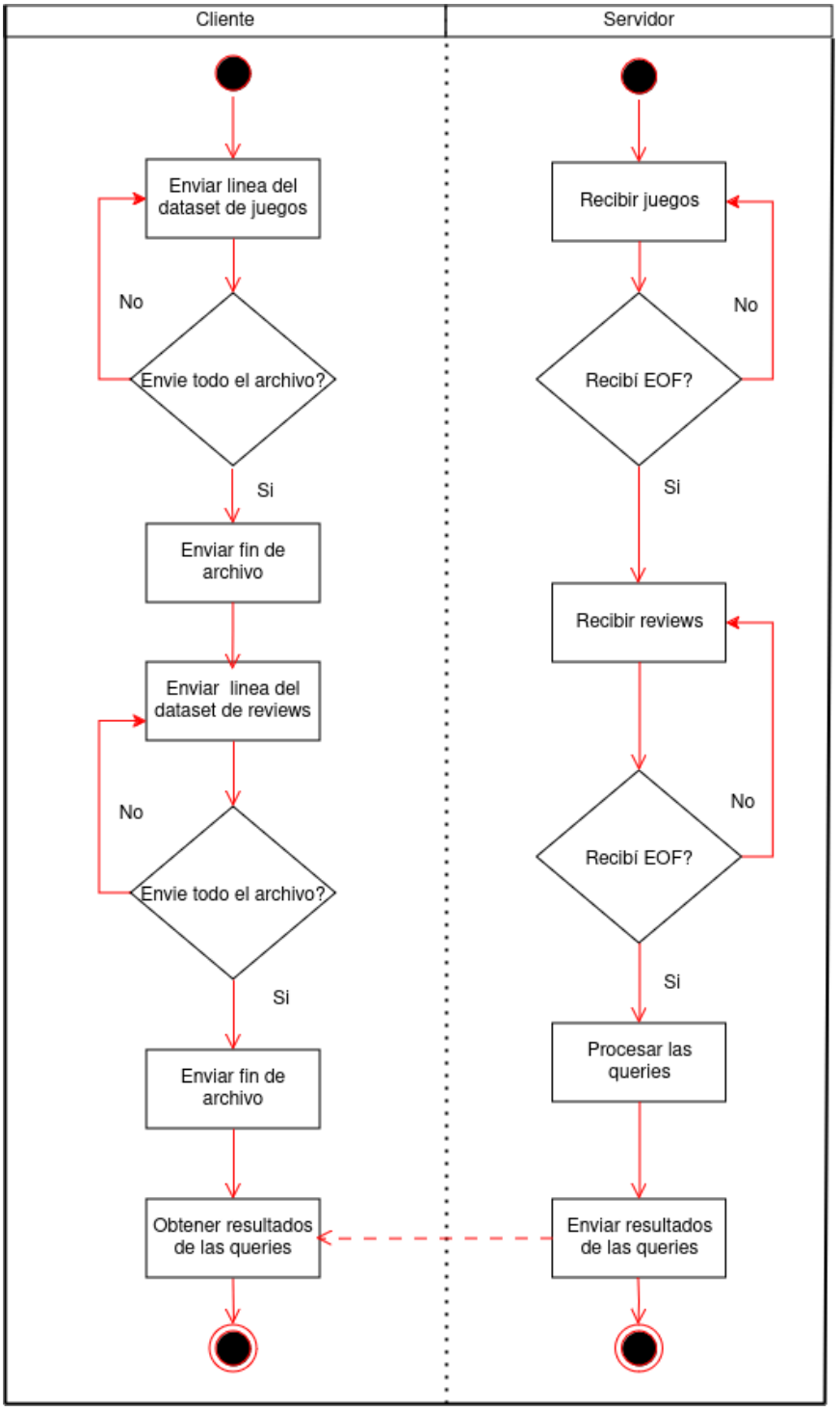


Diagrama de actividad del pre-filtrado de la query 3

El diagrama de actividad muestra el filtrado que es común para todas las queries, solo mostrandolo para el caso específico de la siguiente query: “Nombre de los juegos Indie top 5 en cantidad de reseñas positivas”

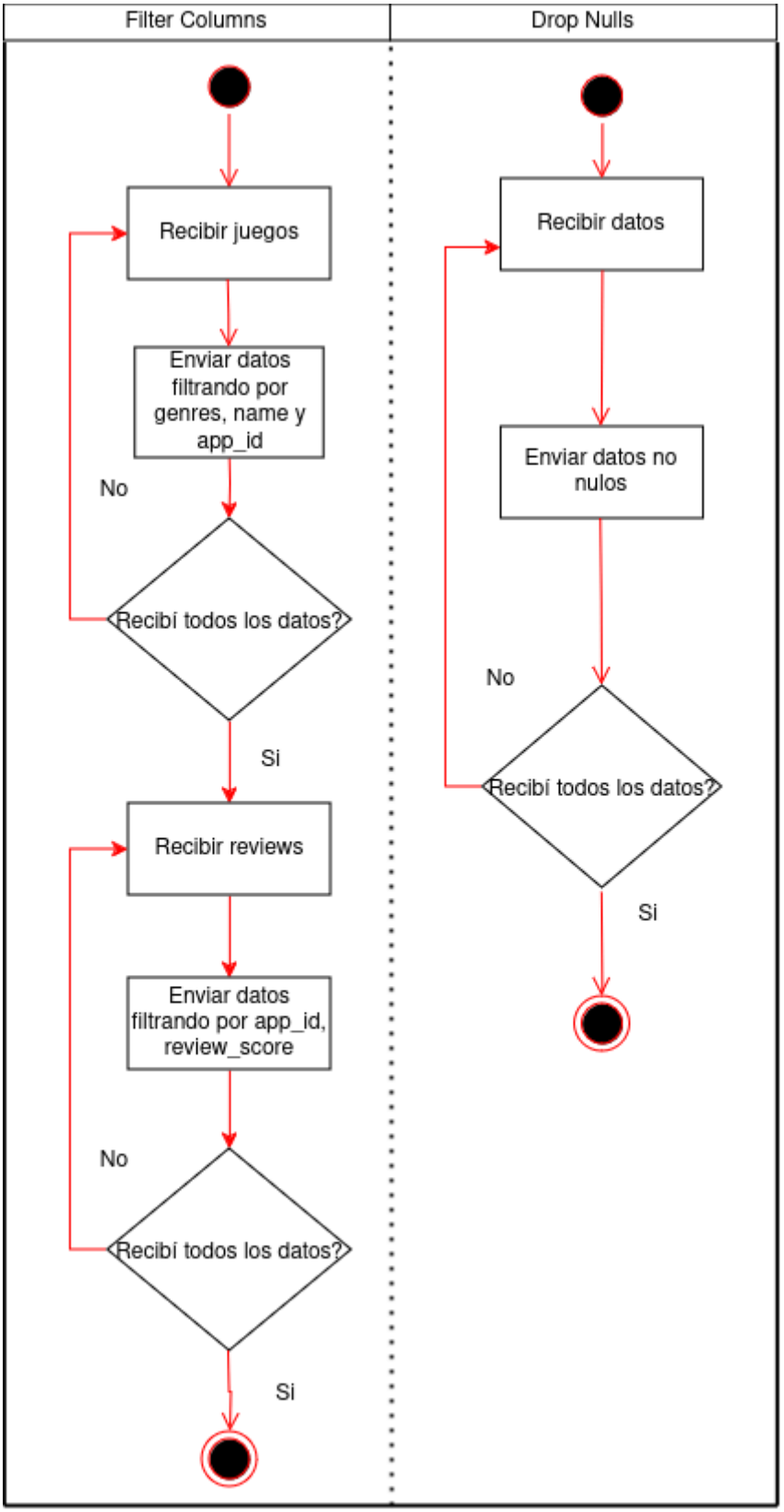


Diagrama de actividad de la query 3

Este diagrama muestra cómo se ejecuta la query 3 luego del filtrado inicial. En este caso se simplificaron algunas de las actividades para poder hacerlo más legible

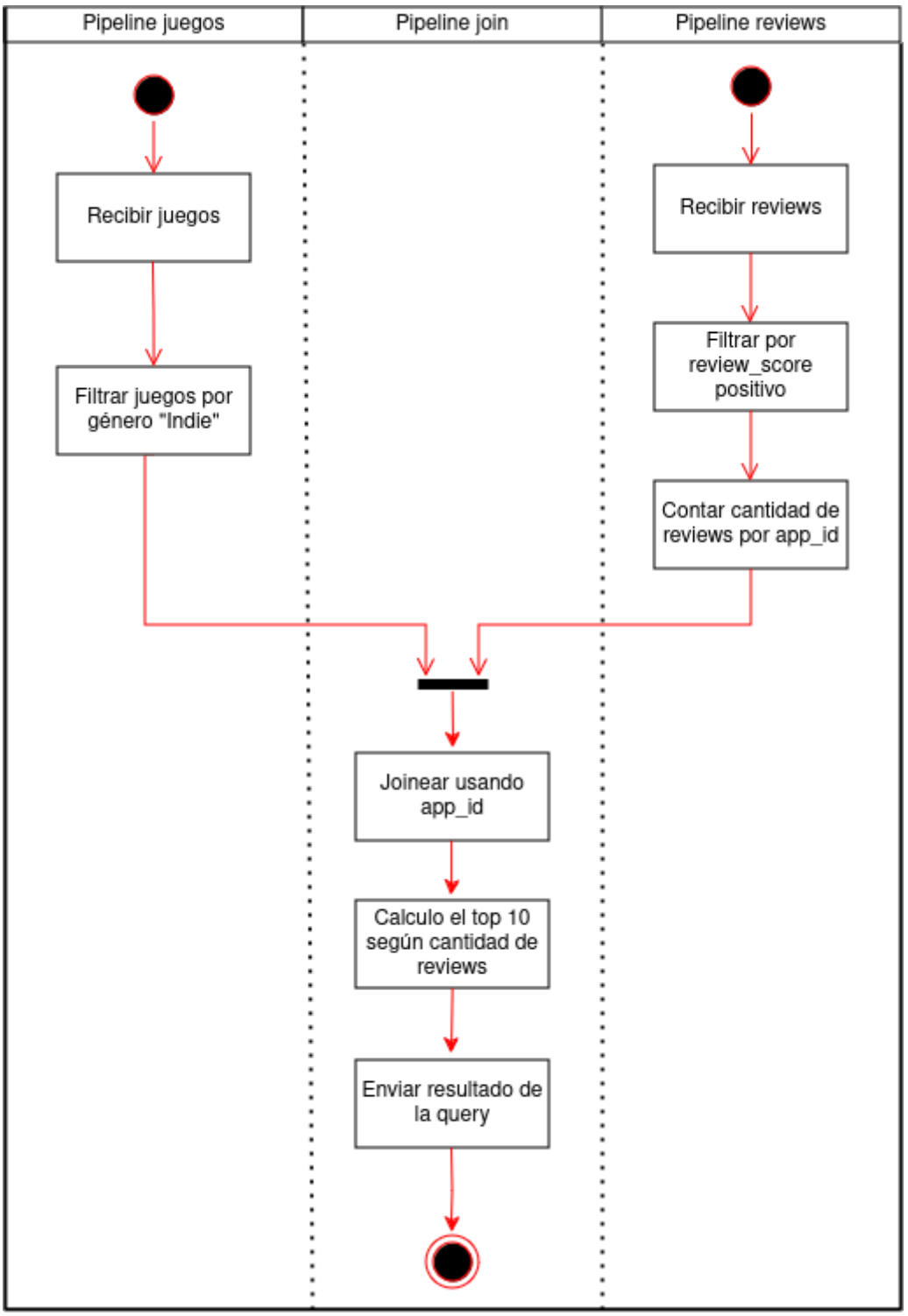
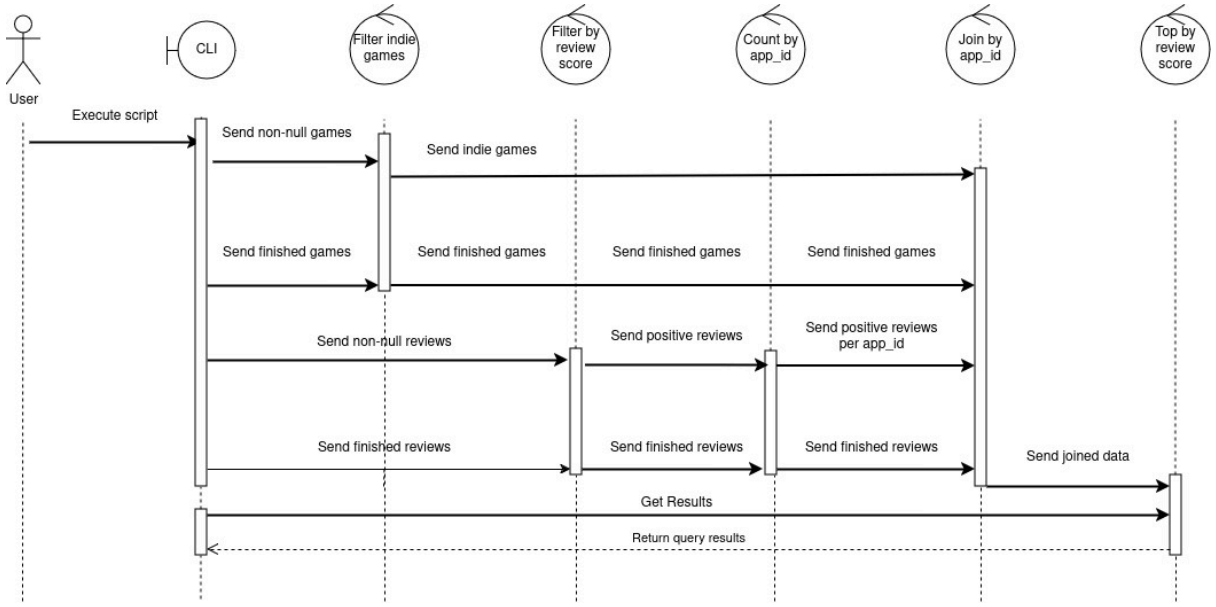


Diagrama de Secuencia

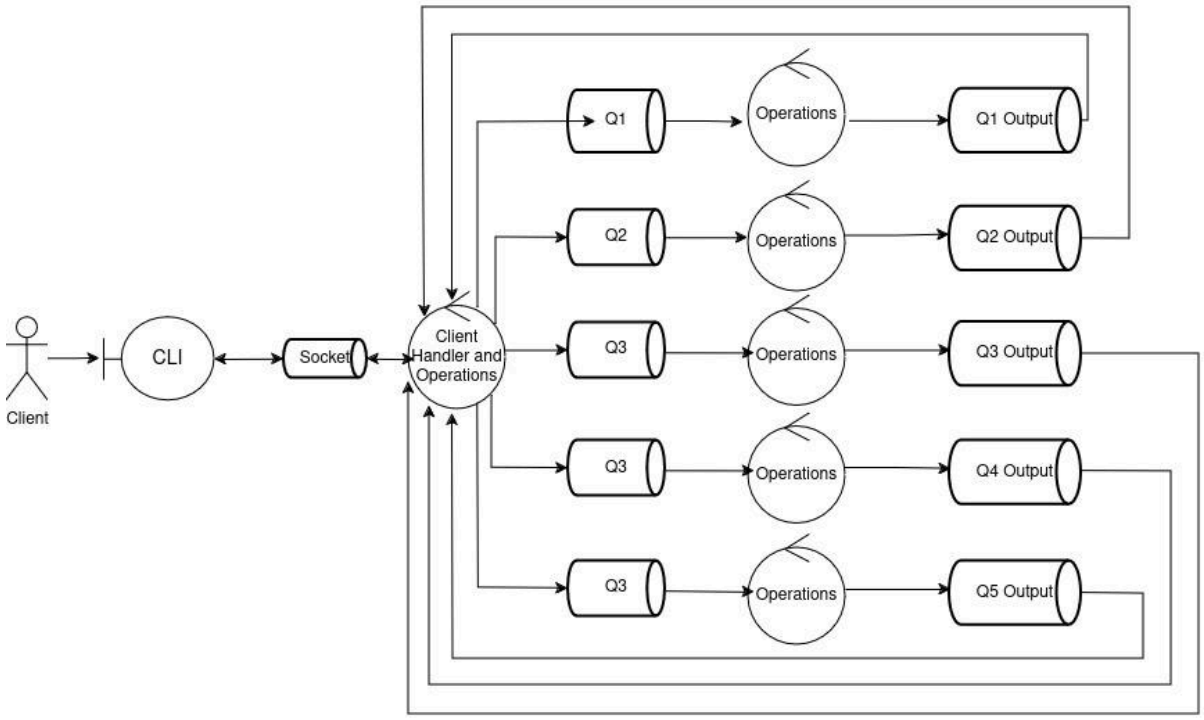
El diagrama de secuencia muestra la ejecución de la la query “Nombre de los juegos Indie top 5 en cantidad de reseñas positivas”, por la misma razón descrita en el de actividad, este “engloba” a las demás queries.



El cliente al momento de enviar los datos correspondientes se mantendrá haciendo *polling* para pedir los resultados que pueda obtener con los datos enviados hasta el momento. Esto no se explicito en todo el gráfico por motivos de legibilidad, en cambio, se optó por dejar, a modo de ilustración, al final del envío de datos

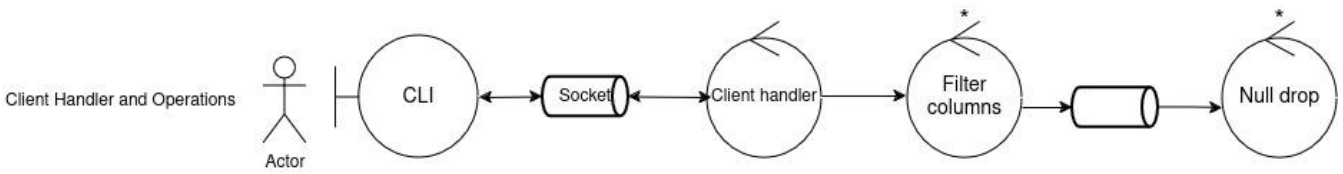
Vista física

Diagrama de robustez general

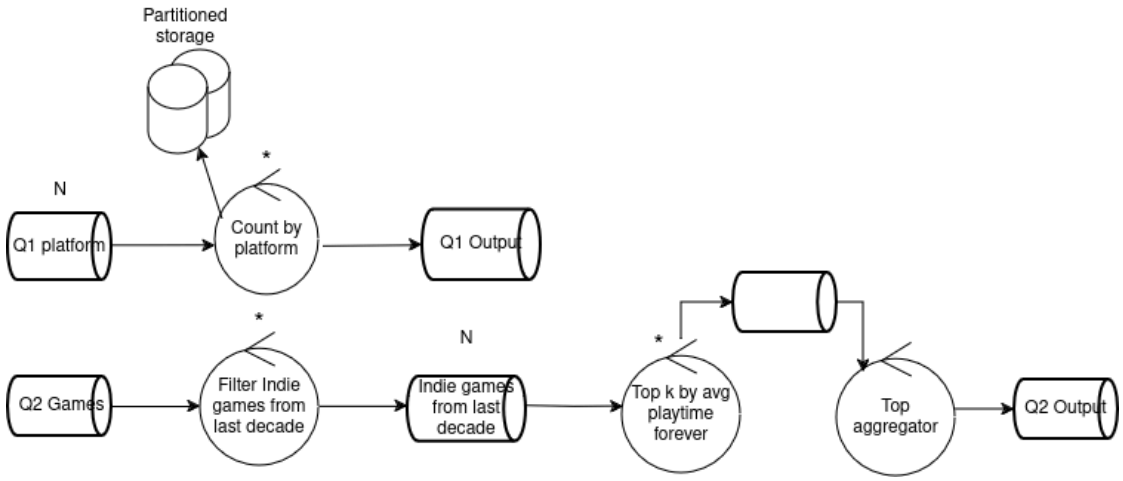


- Q1, Q2, …, Q5 son abreviaturas de QueryN (N el número de la *query*)

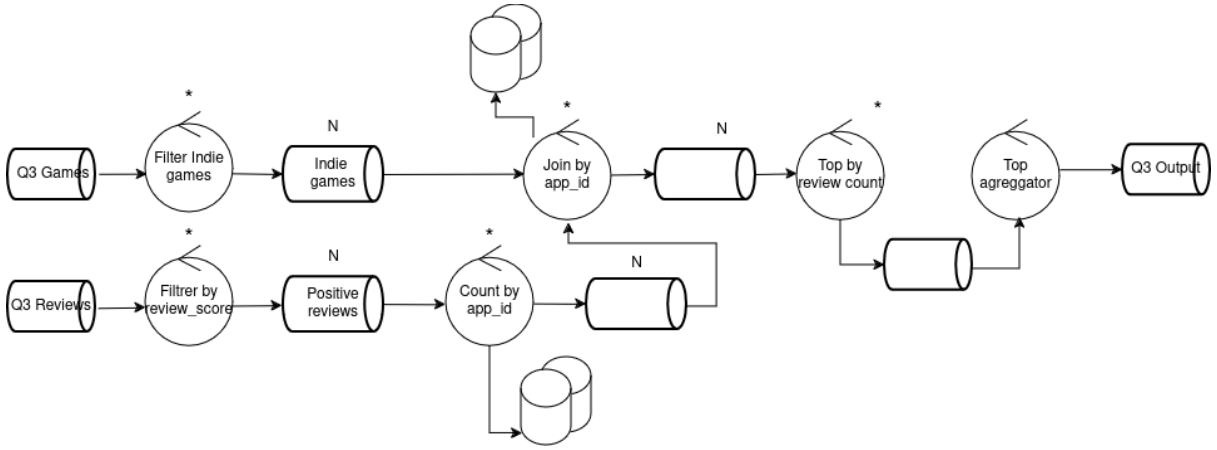
Client Handler and Operations (Acercamiento)



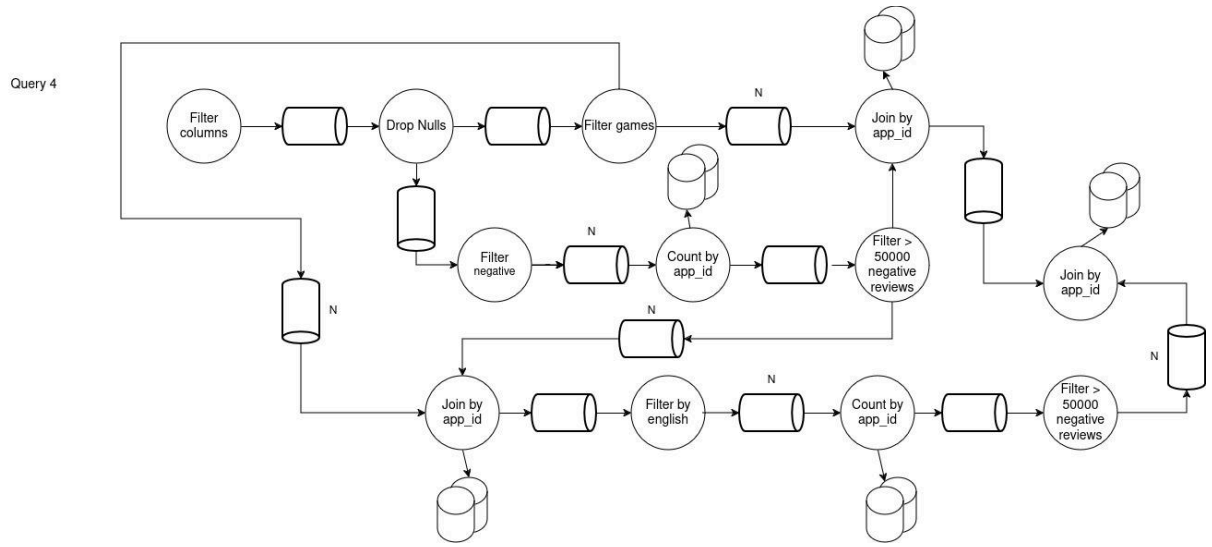
Queries 1 y 2 (Acercamiento)



Query 3 (Acercamiento)



Query 4 (Acercamiento)

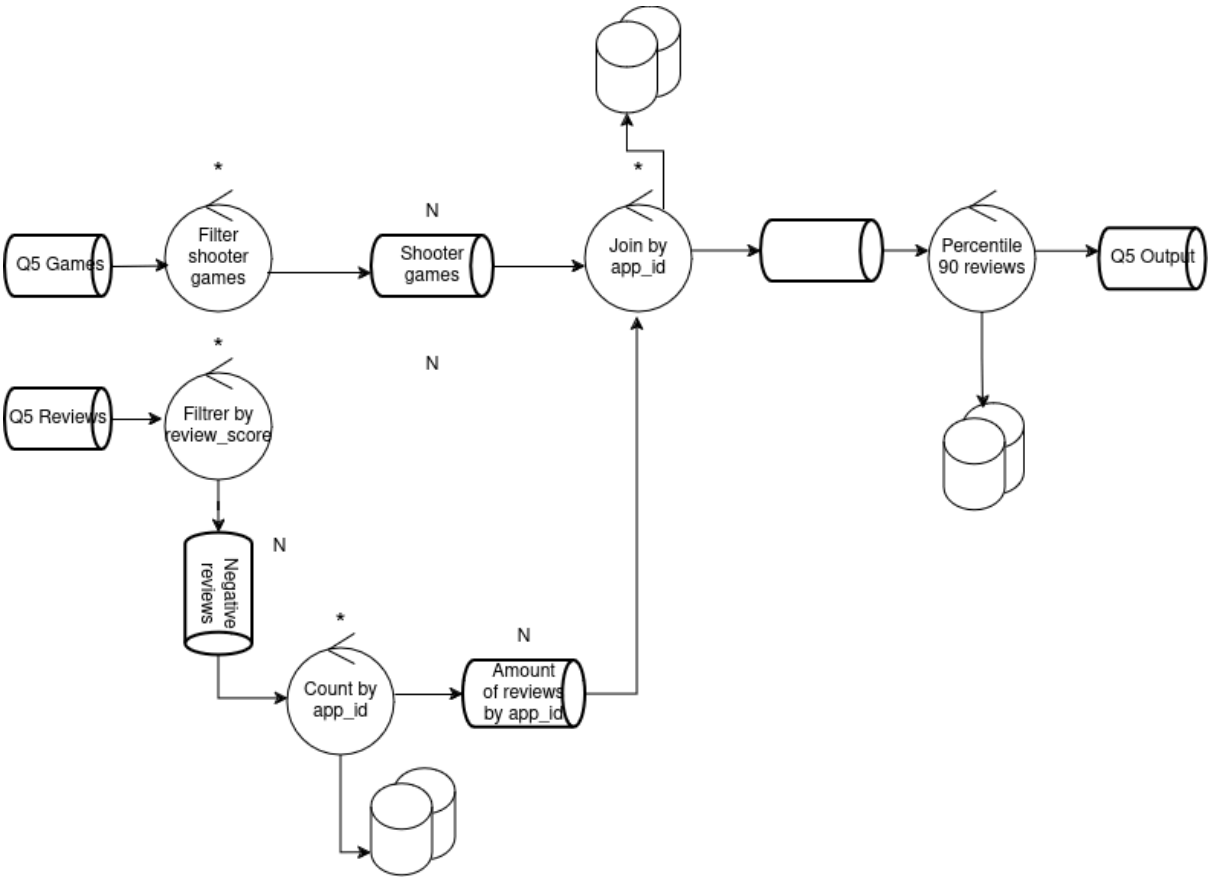


Observaciones

- Para un determinado client_id y app_id se manda a un determinado nodo de join by app_id

- Para un determinado cliente se manda a un determinado nodo de top

Query 5 (Acercamiento)



Observaciones

- Para un determinado cliente se manda a un determinado nodo de percentile 90 reviews
- Para un determinado client_id y app_id se manda a un determinado nodo de join by app_id

Observaciones:

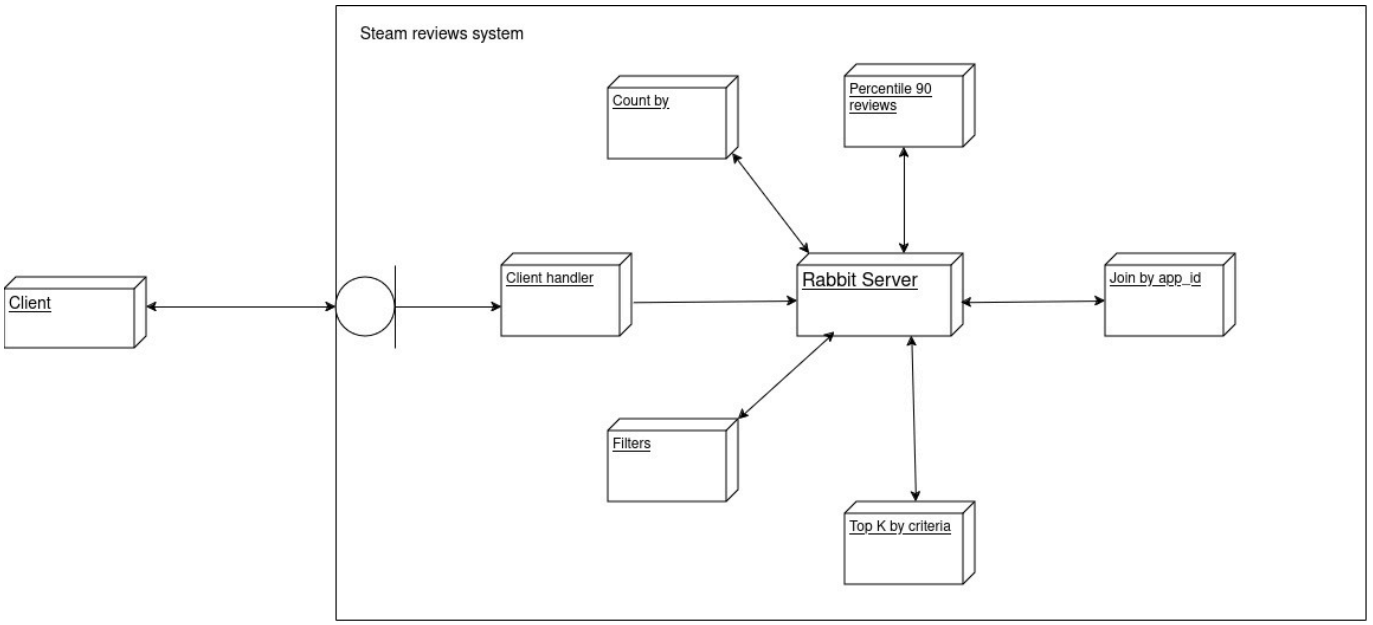
- En aquellos lugares que se utilice la notación "*Partitioned Storage"* se referira a que ese node deberá persistir temporalmente datos de forma particionada, de forma tal de que se pueda cumplir con la query sin tomar supuestos en cuanto a la memoria principal del mismo (véase los casos de *join*, *count*, o aquellos en los que se lleven a cabo funciones de agregación).
- Las particiones se harán según sea conveniente en cada nodo.
- En aquellos nodos en los que se explicite, los mismos serán categorizados utilizando una *sharding key*, siguiendo un patrón de *sharding*.
    - Las colas que tienen un "N" arriba, representan que son para cada *sharding key.*
    - Las *sharding keys* se harán en base al client_id, y un atributo más que dependerá del nodo. Dependiendo de la *query* este se tomará en rangos, por ejemplo app_id (que como se ve en el *dataset* son ids incrementales).
    - La ventaja de esto es que cada nodo, para cada cliente, tendrá los valores absolutos de dicho atributo por el que se busca almacenar, evitando así la necesidad de un agregador o un nodo líder que junte los resultados.
    - La pregunta que surge naturalmente es "qué pasa si la distribución de aquellos atributos elegidos no es uniforme"? En ese caso se podrían tomar métricas del sistema, y crear más instancias de aquellos nodos que tengan más carga.
- La complejidad inherente a la comunicación para el *sharding* será encapsulada por el *middleware*.

Diagrama de despliegue

A continuación se presenta el diagrama de despliegue del sistema.

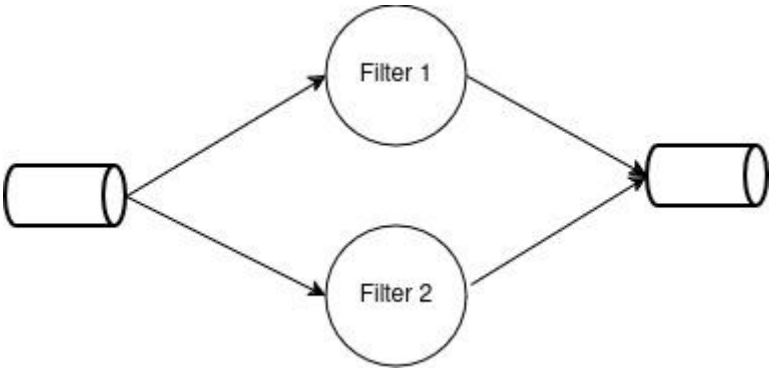
Se busca mostrar en este diagrama los componentes físicos del sistema, y cuáles son estos.

Dado que tiene mucha similitud con el de robustez, se engloban elementos similares que preservan las mismas relaciones.



Manejo de END

A la hora de enviar un mensaje para informar sobre la finalización del dataset para un determinado cliente (vamos a llamarlo END), puede surgir el problema presentado a continuación:



Si se tiene la situación anterior, donde múltiples workers de un tipo de nodo (en este caso de un filtro genérico) consumen de una misma cola, puede ocurrir que uno reciba el END y lo propague, mientras el otro siga procesando data, lo cual podría ocasionar que se crea que se terminó la data antes de lo esperado, perdiendo así datos en el camino.

Para que esto no ocurra, tiene que haber un consenso entre los *workers* para mandar el END, solamente uno de estos debería propagar el END, una vez todos los demás ya lo hayan recibido. Para solucionar este problema, se propone el siguiente algoritmo:

Se asume que todos los *workers* saben la totalidad de workers que hay, para este caso, sería dos.

1) Si se recibe un mensaje de END, se chequea el *payload* del mismo
2) Si no hay tantos ids como *workers,* y el id del *worker* no se encuentra en el payload del mensaje, entonces se agrega al mismo el id del *worker*
3) Si hay tantos ids como *workers*, entonces todos los *workers* recibieron el mensaje, por lo tanto este *worker* propaga el mensaje hacia la cola correspondiente

División de tareas (Tentativo)

Todos: Middleware y storage

Nehuén:
Semana 1 (26/9 - 01/10) :

- Drop columns
- Null drops
- Filter by column

Semana 2 (02/10 - 09/10):

- Filter by language
- Top

Clemente
Semana 1(26/9 - 01/10):

- Envio de juegos desde el cliente
- Envío de reviews desde el cliente
- Polling para obtener resultados desde el cliente

Semana 2(02/10 - 09/10):

- Count sin sharding
- Count usando sharding
- Percentile 90

Juani
Semana 1 (26/9 - 01/10)**:**

- Join
- Hash by client_id, app_id

Semana 2 (02/10 - 09/10)**:**

- Guardar de manera particionada
- Obtención de info que está particionada