A partir del JSON entregado, diseñaría tres entidades principales: Event, que almacena toda la información contenida en cada registro; User, que representa al usuario asociado al evento; y Store, que corresponde a la tienda vinculada. Dado que los eventos pueden ser de tipo "visit" o "recharge", utilizaría un enum para restringir el atributo type a esos dos valores, haciendo el código más robusto y evitando errores. Finalmente, el campo amount sería opcional y solo estaría presente cuando el tipo de evento sea "recharge".

Parte 1:

1.1

Para abordar esta solicitud del cliente, comenzaría creando dos nuevas entidades: Beneficio, que representa el incentivo entregado al usuario, y BeneficioEvento, una entidad intermedia encargada de vincular cada beneficio con los cinco eventos que lo justifican, asignándoles un orden del 1 al 5.

Para evitar errores al procesar grandes volúmenes de eventos, implementaría una base de datos relacional bien estructurada, donde BeneficioEvento permita verificar de manera clara si un evento ya fue utilizado previamente en otro beneficio. A partir de eso, diseñaría un job periódico que procese únicamente los eventos nuevos desde la última ejecución, identificando secuencias válidas de visitas sin recargas y asegurando que no se repitan beneficios. Toda la lógica de asignación estaría encapsulada en una transacción, lo que garantiza consistencia en escenarios con múltiples ejecuciones simultáneas.

1.2

Para un manejo más eficiente de los historiales a medida que aumenta la cantidad de datos, se debería implementar paginación. De lo contrario, los procesos se volverían cada vez más lentos, el costo computacional crecería significativamente debido a la carga excesiva de trabajo, y podrían producirse fallos en otras partes del sistema al destinarse demasiados recursos al procesamiento de grandes volúmenes de información.

Parte 2:

1. Analisis de beneficios:

¿Qué limitaciones tiene tu solución actual?

- La solución actual carga todos los eventos en memoria de una sola vez mediante la función obtenerTodosLosEventos(), lo que resulta altamente ineficiente. El algoritmo tiene una complejidad $O(n^2)$, ya que itera por cada usuario y luego por cada evento asociado. Cada beneficio se crea dentro de una transacción independiente, sin una atomicidad global que garantice consistencia. No se valida si ya existen beneficios en el mismo rango temporal. Si ocurre un error durante la creación de registros en BeneficioEvento, podría quedar un beneficio huérfano sin eventos asociados.

¿Qué pasaría si esto se usa con 100.000 eventos diarios?

- El sistema comenzaría a mostrar un deterioro progresivo en su rendimiento. El análisis se volvería lento debido a la carga masiva de datos en memoria y la falta de optimizaciones, haciendo que los tiempos de procesamiento crezcan significativamente. A medida que se generan más eventos, la probabilidad de errores también aumenta: podrían aparecer beneficios duplicados por condiciones de carrera, y las conexiones a la base de datos se verían sobrecargadas, provocando fallos en la ejecución. Además, el sistema se volvería menos estable y bloquearía otras operaciones mientras se procesan los beneficios, afectando la experiencia de todos los usuarios.

2. Analisis de historiales:

¿Qué limitaciones tiene tu solución actual?

- La función obtenerEventosUsuario carga todo el historial del usuario sin ningún tipo de paginación o límite, lo que resulta poco escalable. Las estadísticas se recalculan con cada consulta, en lugar de utilizar resultados precomputados. Todo el procesamiento es síncrono y secuencial, evaluando un usuario a la vez. No se implementa ningún sistema de caché para evitar recomputaciones, y los datos permanecen en memoria durante toda la operación.

¿Qué pasaría si esto se usa con 100.000 eventos diarios?

- El sistema no sería capaz de manejar eficientemente el volumen de datos acumulado. Cada análisis tomaría más tiempo y consumiría más recursos, dificultando su ejecución para múltiples usuarios simultáneamente. La falta de paginación, caché y procesamiento asincrónico provocaría que cada análisis fuera pesado e independiente, lo que saturaría la base de datos y degradaría el rendimiento general. Como consecuencia, el sistema se volvería lento, poco responsivo y propenso a fallos durante tareas de análisis más intensivas.