Engineer

Event sourcing

estado guardado como registro ordenado de eventos de cambio

Qué es

El event sourcing es una forma de almacenar datos en la que los propios datos son el registro de lo sucedido. La mayoría de las aplicaciones guarda el estado actual de un pedido o una cuenta en una fila y lo sobrescribe con cada cambio. Una aplicación con event sourcing conserva, en cambio, las modificaciones: añade cada una al final de un registro como un evento y calcula el estado actual a partir de ese registro cuando lo necesita. Martin Fowler, que describió el patrón en 2005, lo resumió en una frase: «Capturar todos los cambios del estado de una aplicación como una secuencia de eventos».

Un evento registra algo que ya ha ocurrido. Los *CQRS Documents* de Greg Young piden que cada evento se nombre con un verbo en pasado, como `CustomerRelocated` o `CargoShipped`, porque su nombre forma parte del lenguaje con que el negocio habla de su propio trabajo. Un hecho del pasado no se puede deshacer, así que los eventos nunca se editan ni se eliminan. Young compara el registro con un libro contable, donde una entrada incorrecta no se borra, sino que se corrige con otra nueva; aplica la misma regla a la eliminación. Cuando un cliente quita dos pares de calcetines del carrito, el registro recibe un evento «retirado». Young llama reversión a esa entrada: el carrito queda como si nunca se hubieran añadido los calcetines y el registro sigue mostrando que sí se añadieron.

Los eventos se guardan en flujos, uno por cada elemento cuyo estado describen. Young, que parte del diseño guiado por el dominio, llama agregado a ese elemento. Cada evento lleva el identificador del flujo y un número de versión único y secuencial dentro de él, por lo que nunca hay dudas sobre el orden. Para cargar un pedido, la aplicación lee sus eventos por versión y los aplica de uno en uno. Young señala lo poco que tiene que hacer el almacenamiento: en su forma más sencilla ofrece dos operaciones, leer todos los eventos de un flujo en orden y añadirle otros nuevos al final.

Eso convierte al registro en la fuente de referencia del sistema. Fowler señala que el registro oficial puede ser el registro de eventos o el estado actual. En un sistema con event sourcing lo es el registro de eventos; todo lo demás, una tabla de saldos actuales, un índice de búsqueda o un informe, es una copia derivada que se puede descartar y volver a construir.

Utilice el concepto en una decisión

Aplique «¿Qué es el event sourcing?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. La parte que el usuario no ve: dónde se guardan los datos, qué reglas se aplican y cómo pueden comunicarse otros sistemas con ella.

Explore nuestros servicios Póngase en contacto →

Compruebe los supuestos detrás del término

Utilice la explicación y los trabajos enlazados para formular preguntas sobre su proyecto. Podemos examinar los supuestos con usted antes de elegir una herramienta. Diseño y operación de 6 flujos ETL/ELT — 6 flujos ETL en tres motores de bases de datos

Diseño y operación de 6 flujos ETL/ELT Póngase en contacto →

Por qué importa

Fowler enumera tres cosas que permite un registro completo. La reconstrucción completa: se puede descartar el estado de la aplicación y recrearlo reproduciendo todos los eventos en una aplicación vacía. La consulta temporal: el estado en cualquier momento pasado es el resultado de reproducir los eventos hasta ese instante. Y la reproducción de eventos: cuando se descubre que uno era incorrecto, el sistema puede revertirlo junto a los posteriores y después reproducir la secuencia corregida. El mismo registro también es una pista de auditoría que nadie tuvo que acordarse de escribir; Fowler cita el soporte y la depuración como usos cotidianos: un equipo de soporte puede ver exactamente qué hizo un usuario y un desarrollador puede reproducir eventos de producción en un entorno de pruebas.

Young muestra el beneficio con dos equipos de un comercio en línea a los que se pide un informe sobre los artículos que los clientes quitan de sus carritos. El que solo almacena el estado actual empieza a registrar eliminaciones ahora, y su informe tiene datos desde ese día. El que guarda eventos escribe el mismo manejador de informes y lo ejecuta sobre el registro desde el principio, así que su primer informe ya se remonta años atrás. Una pregunta que nadie pensó en hacer al construir el sistema se puede responder porque este conservó lo ocurrido, en lugar de solo su resultado final.

Los eventos también permiten que otros sistemas conozcan un cambio. Young observa que muchos guardan su estado y después publican un evento en una cola, lo que, estrictamente, necesita una confirmación en dos fases entre ambos, porque una caída entre los dos pasos los deja en desacuerdo. Su alternativa es tratar el propio almacén de eventos como la cola: un número de secuencia por evento permite que otro proceso los publique en orden, así que solo hay una escritura. Además, como el almacenamiento solo permite añadir al final, es más fácil escalarlo. Young señala que necesita muchos menos bloqueos que un almacenamiento que actualiza los datos existentes, y que el identificador del agregado es la única clave por la que un almacén de eventos necesita particionarse.

Los costes son reales y Fowler los expone. Empaquetar cada cambio como evento es un estilo de programación con el que no todo el mundo se siente cómodo, y el patrón es muy difícil de incorporar después a un sistema que no se construyó a su alrededor. El almacenamiento también responde a un solo tipo de pregunta: «dame este flujo». Young lo ilustra así: un almacén de eventos no puede decir qué usuarios se llaman Greg. Esas preguntas necesitan un modelo de lectura aparte, construido a partir de los eventos; por eso trata el event sourcing y CQRS, la separación de los modelos de escritura y lectura, como una pareja.

Cómo se aplica en la práctica

El almacén de eventos funcional más pequeño es una tabla de PostgreSQL, cercana al diseño de Young: un identificador de agregado, los datos del evento y una versión. Aquí conserva las existencias de un artículo a la venta. Cada evento lleva un cambio en la cantidad de unidades a la venta en lugar del nuevo total, la forma que Fowler considera más sencilla de revertir: «añadir diez» se deshace restando diez, mientras que «fijar en 110» ha perdido el valor que sustituye.

```
CREATE TABLE stock_events (
    item_id  text    NOT NULL,
    version  integer NOT NULL,
    event    text    NOT NULL,
    order_id text,
    units    integer NOT NULL,  -- change to the units on sale
    PRIMARY KEY (item_id, version)
);

INSERT INTO stock_events VALUES
    ('item-881', 1, 'ItemStocked',   NULL,   100),
    ('item-881', 2, 'UnitsHeld',     'A-17',  -2),
    ('item-881', 3, 'HoldExpired',   'A-17',   2),
```

```
    ('item-881', 4, 'UnitsHeld',     'B-02',  -1),
    ('item-881', 5, 'HoldConfirmed', 'B-02',   0);

-- current state: fold the whole stream
SELECT sum(units) AS on_sale FROM stock_events
WHERE item_id = 'item-881';                    -- 99

-- temporal query: the item as it stood after version 2
SELECT sum(units) AS on_sale FROM stock_events
WHERE item_id = 'item-881' AND version <= 2;   -- 98
```

La clave primaria hace el trabajo que Young atribuye a una comprobación de concurrencia optimista. Dos procesos de compra que leen ambos el flujo en la versión 5 y deciden reservar una unidad intentan añadir cada uno la versión 6:

```
INSERT INTO stock_events VALUES ('item-881', 6, 'UnitsHeld', 'C-40', -1);
INSERT INTO stock_events VALUES ('item-881', 6, 'UnitsHeld', 'D-11', -1);
-- the second fails: (item-881, 6) already exists
```

La documentación de PostgreSQL explica que una clave primaria sobre un grupo de columnas hace única esa combinación, y una inserción que la duplicaría genera un error. El proceso de compra que perdió vuelve a cargar el flujo, comprueba que aún haya una unidad a la venta y lo intenta otra vez como versión 7. Ninguno vendió una unidad inexistente y el registro muestra ambas decisiones en el orden en que se tomaron.

La nota Vender el último billete exactamente una vez recorre el mismo problema de existencias y muestra dónde se complica esta estructura. Su primera respuesta mantiene el contador en una fila y lo cambia con un único `UPDATE` condicional; desaconseja el bloqueo optimista en un contador muy disputado, porque una columna de versión convierte la contención en reintentos. Para el artículo más solicitado coloca cada solicitud en una partición de Kafka, leída por un solo consumidor que conserva el contador en memoria y confirma sus decisiones por lotes; lo llama estructura LMAX. El artículo de Fowler sobre la arquitectura LMAX, una de sus fuentes, describe el event sourcing en su forma más pura: un motor de negociación sin base de datos cuyo estado entero se obtiene de sus eventos de entrada, que un proceso de registro guarda de forma duradera para poder reproducirlos. El proceso de compra de la nota usa eventos de otra clase. Cada paso escribe un mensaje en una tabla de salida (outbox), la

captura de datos modificados lo publica y las tablas siguen siendo la fuente de referencia. Esos eventos anuncian cambios; en event sourcing, los eventos son el registro.

Dónde surgen los problemas

Reproducir el registro repite sus efectos secundarios. Un manejador que envía un correo de confirmación o realiza un cargo en una tarjeta vuelve a hacerlo cuando se reproduce el registro; Fowler señala que los sistemas externos no distinguen una reproducción del trabajo real. Reconstruir un modelo de lectura puede entonces reenviar a todos los clientes los correos del año pasado. Sitúe cada sistema externo detrás de una pasarela que sepa cuándo la aplicación está reproduciendo eventos y descarte las llamadas salientes durante la reproducción, para que la lógica de dominio nunca tenga que saberlo. Las consultas necesitan el mismo cuidado: Fowler pone el ejemplo de una conversión de moneda, donde el evento reproducido necesita el tipo de cambio de su propio día, no el de hoy, así que la pasarela debería registrar las respuestas que recibió y devolverlas durante la reproducción.

Un flujo con mucha actividad acaba generando reintentos. La concurrencia optimista supone que los conflictos son poco frecuentes. Si muchos escritores añaden eventos a un mismo flujo a la vez, cada conflicto obliga a recargarlo y volver a intentarlo; ante una ráfaga de tráfico, los reintentos se convierten en la carga. Es la advertencia de la nota anterior sobre las columnas de versión. Delimite los flujos para que rara vez tengan dos escritores simultáneos: uno por pedido en lugar de uno para toda una línea de productos. Cuando un flujo sí esté muy disputado, asígnele un solo escritor, como en el diseño LMAX, para que ya no haya nada que entre en conflicto.

Los flujos largos tardan en cargarse. Reproducir miles de eventos para cargar un objeto cuesta tiempo en cada solicitud; Fowler incluye las reconstrucciones lentas entre los motivos para conservar una copia del estado actual. La respuesta de Young es la instantánea que se va renovando: el estado serializado de un flujo en una versión conocida, guardado en una tabla propia y escrito en segundo plano por otro proceso, de modo que cargarlo requiere la última instantánea y los eventos posteriores a su versión. Mantenga las instantáneas fuera del propio registro de eventos: Young describe un proceso de instantáneas que, en un flujo con mucha actividad, falla una y otra vez su comprobación de concurrencia. También recomienda construir primero sin instantáneas, porque siempre se pueden añadir después para acelerar la carga.

Los lectores se quedan atrás respecto al escritor. Cada modelo de lectura se construye a partir de los eventos después de escribirlos, por lo que va detrás del registro. Fowler advierte de que los sistemas que leen el registro pueden estar desincronizados con él, y un usuario que envía un formulario y después ve una lista sin el cambio lo enviará otra vez. Haga que cada modelo de lectura registre el último evento que aplicó, para medir su retraso y poder reconstruirlo desde el principio; muestre a los usuarios el resultado de su propia orden a partir de la respuesta de esta, en lugar de usar un modelo de lectura que quizá aún no se haya puesto al día.

Lecturas adicionales

- Event Sourcing — La descripción del patrón de Martin Fowler: reconstrucción completa, consulta temporal, reproducción de eventos y pasarelas que impiden que las reproducciones lleguen a otros sistemas.
- Documentos de CQRS de Greg Young — Eventos como hechos en pasado, reversión en vez de eliminación, un almacén de eventos de dos tablas, instantáneas que se van renovando y el almacén como cola.
- La arquitectura LMAX — Un motor de negociación en memoria cuyo estado entero se obtiene de eventos de entrada registrados de forma duradera.
- PostgreSQL: restricciones — Restricciones de unicidad y clave primaria sobre un grupo de columnas: la parte de la base de datos en una comprobación de concurrencia optimista.

Glosario

Elija según sus restricciones reales

Describa el flujo de trabajo, las interfaces existentes y los usuarios que necesitan el cambio. Podemos definir una entrega comprobable y un traspaso que permita a su equipo mantener lo entregado.

Vea cuánto cuesta Póngase en contacto →

Haga práctica la terminología

Pregunte cómo se aplica «¿Qué es el event sourcing?» a su organización. Basta un ejemplo concreto de su sistema actual para iniciar la conversación.

Explore nuestros servicios

welcome@engineer.company +45 50 10 61 84 Copenhague, Dinamarca

Llévese esta página

Tarjeta de contacto

Creado con esmero y cariño por D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.

Impreso desde https://engineer.company/es/glossary/event-sourcing/