Engineer

Event Sourcing

Zustand als geordnetes Log seiner Änderungsereignisse halten

Was es ist

Event Sourcing ist eine Art der Datenspeicherung, bei der die Aufzeichnung des Geschehenen selbst die Daten bildet. Die meisten Anwendungen halten den aktuellen Zustand einer Bestellung oder eines Kontos in einer Zeile und überschreiben ihn bei jeder Änderung. Eine Anwendung mit Event Sourcing hält stattdessen die Änderungen fest: Jede wird als Ereignis an ein Log angehängt. Der aktuelle Zustand wird bei Bedarf aus diesem Log ermittelt. Martin Fowler beschrieb das Muster 2005 in einem Satz: „Alle Änderungen am Zustand einer Anwendung als eine Folge von Ereignissen erfassen."

Ein Ereignis hält etwas fest, das bereits geschehen ist. Greg Young verlangt in seinen CQRS Documents, jedes Ereignis mit einem Verb in der Vergangenheitsform zu benennen, etwa `CustomerRelocated` oder `CargoShipped`. Der Name ist Teil der Sprache, mit der das Unternehmen über seine eigene Arbeit spricht. Eine Tatsache aus der Vergangenheit lässt sich nicht ungeschehen machen. Ereignisse werden daher nie bearbeitet oder entfernt. Young vergleicht das Log mit einem Buchhaltungsjournal: Ein falscher Eintrag wird nicht ausradiert, sondern durch einen neuen korrigiert. Dieselbe Regel wendet er auf das Löschen an. Nimmt ein Kunde zwei Paar Socken wieder aus dem Warenkorb, erhält das Log ein Ereignis „entfernt". Young nennt einen solchen Eintrag eine Umkehrung: Der Warenkorb ist anschließend so, als wären die Socken nie hinzugefügt worden, während das Log weiterhin zeigt, dass es geschehen ist.

Die Ereignisse liegen in Streams, jeweils einem pro Gegenstand, dessen Zustand sie beschreiben. Young baut auf Domain-Driven Design auf und nennt diesen Gegenstand ein Aggregat. Jedes Ereignis trägt die ID des Streams und eine darin eindeutige, fortlaufende Versionsnummer. Über die Reihenfolge besteht deshalb nie Zweifel. Um eine Bestellung zu laden, liest die Anwendung deren Ereignisse

nach Version und wendet sie nacheinander an. Young betont, wie wenig der Speicher selbst tun muss: Im einfachsten Fall bietet er zwei Operationen, alle Ereignisse eines Streams in Reihenfolge zu lesen und neue daran anzuhängen.

Damit wird das Log zur maßgeblichen Datenquelle. Fowler merkt an, dass entweder das Log oder der aktuelle Zustand der offizielle Datenbestand sein kann. In einem System mit Event Sourcing ist es das Log. Alles andere, eine Tabelle aktueller Kontostände, ein Suchindex oder ein Bericht, ist eine daraus abgeleitete Kopie, die verworfen und neu aufgebaut werden kann.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist Event Sourcing?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Der Teil, den Nutzer nie sehen: wo die Daten liegen, welche Regeln gelten und wie andere Systeme darauf zugreifen dürfen.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. 6 ETL/ELT-Pipelines entworfen und betrieben — 6 ETL-Pipelines über drei Datenbankengines hinweg

6 ETL/ELT-Pipelines entworfen und betrieben Kontakt aufnehmen →

Warum es wichtig ist

Fowler nennt drei Möglichkeiten, die ein vollständiges Log eröffnet. Ein vollständiger Neuaufbau: Der Anwendungszustand lässt sich verwerfen und durch das erneute Abspielen aller Ereignisse in einer leeren Anwendung wiederherstellen. Eine zeitbezogene Abfrage: Der Zustand zu einem früheren Zeitpunkt ergibt sich aus dem Abspielen der Ereignisse bis dahin. Und das erneute Abspielen von Ereignissen: Stellt sich eines als falsch heraus, kann das System dieses und die nachfolgenden rückgängig machen und dann die korrigierte Folge erneut abspielen. Dasselbe Log ist auch eine Prüfspur, an deren Erstellung niemand extra denken musste. Fowler nennt Support und Fehlersuche als alltägliche Anwendungen: Ein Supportteam kann genau sehen, was ein Nutzer getan hat, und eine Entwicklerin oder ein Entwickler kann Produktionsereignisse in einer Testumgebung erneut abspielen.

Young zeigt den Nutzen an zwei Teams eines Onlinehändlers. Beide sollen einen Bericht über Artikel erstellen, die Kunden aus ihren Warenkörben entfernen. Das Team, das nur den aktuellen Zustand speichert, beginnt jetzt mit dem Aufzeichnen der Entnahmen. Sein Bericht enthält Daten ab diesem Tag. Das Team, das Ereignisse speichert, schreibt dieselbe Berichtsfunktion und lässt sie das Log von Anfang an verarbeiten. Schon sein erster Bericht reicht dadurch

Jahre zurück. Eine Frage, an die beim Bau des Systems niemand dachte, lässt sich dennoch beantworten, weil das System das Geschehene statt nur seines Endzustands festhielt.

Über Ereignisse erfahren auch andere Systeme von einer Änderung. Young beobachtet, dass viele Systeme ihren Zustand speichern und dann ein Ereignis in einer Queue veröffentlichen. Streng genommen erfordere das einen Zwei-Phasen-Commit zwischen beiden, weil ein Absturz dazwischen widersprüchliche Zustände hinterlässt. Seine Alternative ist, den Ereignisspeicher selbst als Queue zu behandeln. Eine Sequenznummer an jedem Ereignis ermöglicht einem separaten Prozess, sie in Reihenfolge zu veröffentlichen; damit bleibt nur ein Schreibvorgang. Weil der Speicher nur Anhängen erlaubt, lässt er sich zudem leichter skalieren. Young merkt an, dass ein solcher Speicher deutlich weniger Sperren braucht als einer, der bestehende Daten aktualisiert. Die Aggregat-ID ist außerdem der einzige Schlüssel, nach dem ein Ereignisspeicher jemals partitioniert werden muss.

Die Kosten sind real, und Fowler benennt sie. Jede Änderung als Ereignis zu verpacken ist ein Programmierstil, mit dem sich nicht alle wohlfühlen. Das Muster lässt sich nur sehr schwer nachträglich in ein System einbauen, das nicht darauf ausgelegt wurde. Der Speicher beantwortet außerdem nur eine Art von Frage: „Gib mir diesen Stream." Youngs Beispiel: Ein Ereignisspeicher kann nicht sagen, welche Nutzer Greg heißen. Solche Fragen brauchen ein separates, aus den Ereignissen aufgebautes Lesemodell. Deshalb behandelt er Event Sourcing und CQRS, die Trennung der Modelle zum Schreiben und Lesen, als zusammengehörig.

Wie es in der Praxis aussieht

Der kleinste funktionsfähige Ereignisspeicher ist eine einzelne PostgreSQL-Tabelle, nahe an Youngs Aufbau mit Aggregat-ID, Ereignisdaten und Version. Hier hält sie den zum Verkauf stehenden Bestand eines Artikels fest. Jedes Ereignis enthält eine Änderung der angebotenen Stückzahl statt der neuen Gesamtsumme. Fowler nennt diese Form die am einfachsten umkehrbare: „Zehn hinzufügen" wird durch das Abziehen von zehn rückgängig gemacht, während „auf 110 setzen" den ersetzten Wert verloren hat.

```
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
```

Der Primärschlüssel übernimmt die von Young vorgesehene optimistische Nebenläufigkeitsprüfung. Lesen zwei Kaufabschlüsse den Stream bei Version 5 und entscheiden beide, ein Stück zu reservieren, versuchen beide, Version 6 anzuhängen:

```
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
```

Die PostgreSQL-Dokumentation beschreibt einen Primärschlüssel über mehrere Spalten so, dass er diese Kombination eindeutig macht. Ein Einfügevorgang, der sie duplizieren würde, löst einen Fehler aus. Der unterlegene Kaufabschluss lädt den Stream erneut, prüft, ob noch ein Stück zum Verkauf steht, und versucht es als Version 7 noch einmal. Keiner hat je ein nicht vorhandenes Stück verkauft, und das Log zeigt beide Entscheidungen in ihrer Reihenfolge.

Die Notiz Das letzte Ticket genau einmal verkaufen behandelt dasselbe Bestandsproblem und zeigt die Grenzen dieses Aufbaus. Ihre erste Lösung hält die Anzahl in einer Zeile und ändert sie mit einem einzelnen bedingten `UPDATE`. Sie rät bei einem stark umkämpften Zähler von optimistischen Sperren ab, weil eine Versionsspalte Konflikte in Wiederholungsversuche verwandelt. Für den am stärksten nachgefragten Artikel legt sie jede Anfrage auf eine Kafka-Partition. Ein einzelner Konsument liest sie, hält den Bestand im Speicher und schreibt seine

Entscheidungen gesammelt fest. Die Notiz nennt das den LMAX-Aufbau. Fowlers Artikel über die LMAX-Architektur, eine ihrer Quellen, beschreibt Event Sourcing in Reinform: eine Handelsengine ohne Datenbank, deren gesamter Zustand sich aus ihren Eingabeereignissen ableiten lässt. Ein Journalprozess speichert diese dauerhaft, damit sie erneut abgespielt werden können. Der Kaufabschluss in der Notiz verwendet Ereignisse anderer Art. Jeder Schritt schreibt eine Nachricht in eine Outbox-Tabelle, Change Data Capture veröffentlicht sie, und die Tabellen bleiben die maßgebliche Datenquelle. Diese Ereignisse kündigen Änderungen an; bei Event Sourcing sind die Ereignisse selbst der maßgebliche Datenbestand.

Wo es schiefgeht

Das erneute Abspielen des Logs wiederholt auch seine Nebenwirkungen. Eine Verarbeitungsfunktion, die eine Bestätigungsmail sendet oder eine Karte belastet, tut das beim erneuten Abspielen wieder. Fowler weist darauf hin, dass externe Systeme eine Wiederholung nicht von echter Arbeit unterscheiden können. Der Neuaufbau eines Lesemodells kann dann jedem Kunden die E-Mails des letzten Jahres erneut schicken. Setzen Sie vor jedes externe System ein Gateway, das weiß, wann die Anwendung Ereignisse erneut abspielt, und dabei ausgehende Aufrufe verwirft. Die Domänenlogik muss davon dann nichts wissen. Abfragen brauchen dieselbe Sorgfalt: Fowlers Beispiel ist eine Währungsumrechnung, bei der ein erneut abgespieltes Ereignis den Wechselkurs seines eigenen Tages braucht, nicht den heutigen. Das Gateway sollte deshalb empfangene Antworten aufzeichnen und beim erneuten Abspielen zurückgeben.

Ein viel genutzter Stream erzeugt Wiederholungsversuche. Optimistische Nebenläufigkeitskontrolle setzt seltene Konflikte voraus. Fügen viele Schreiber gleichzeitig Ereignisse an einen Stream an, bedeutet jeder Konflikt erneutes Laden und einen weiteren Versuch. Bei einer Lastspitze werden die Wiederholungsversuche selbst zur Last. Genau davor warnt die obige Notiz bei Versionsspalten. Schneiden Sie Streams so zu, dass sie selten zwei gleichzeitige Schreiber haben: ein Stream pro Bestellung statt einer für eine ganze Produktlinie. Ist ein einzelner Stream tatsächlich stark umkämpft, geben Sie ihm wie im LMAX-Entwurf einen einzigen Schreiber. Dann kann nichts mehr kollidieren.

Lange Streams laden langsam. Tausende Ereignisse zum Laden eines Objekts erneut abzuspielen kostet bei jeder Anfrage Zeit. Fowler zählt langsamen Neuaufbau zu den Gründen, eine Kopie des aktuellen Zustands zu halten. Youngs Antwort ist der fortlaufende Snapshot: der serialisierte Zustand eines Streams bei einer bekannten Version, in einer eigenen Tabelle gehalten und im

Hintergrund von einem separaten Prozess geschrieben. Zum Laden braucht es dann den neuesten Snapshot und die Ereignisse nach dessen Version. Halten Sie Snapshots aus dem Ereignislog selbst heraus. Young beschreibt, wie ein Snapshot-Prozess bei einem viel genutzten Stream immer wieder an seiner Nebenläufigkeitsprüfung scheitert. Er empfiehlt außerdem, zunächst ohne Snapshots zu bauen, weil sie sich später jederzeit ergänzen lassen, um das Laden zu beschleunigen.

Leser hinken dem Schreiber hinterher. Jedes Lesemodell entsteht erst nach dem Schreiben aus den Ereignissen und liegt daher hinter dem Log zurück. Fowler warnt, dass Systeme, die das Log lesen, leicht nicht auf dessen Stand sind. Wer ein Formular abschickt und danach eine Liste ohne die Änderung sieht, schickt es erneut ab. Lassen Sie jedes Lesemodell das zuletzt angewendete Ereignis festhalten, damit seine Verzögerung messbar wird und es von Anfang an neu aufgebaut werden kann. Zeigen Sie Nutzern das Ergebnis ihres eigenen Befehls aus dessen Antwort statt aus einem möglicherweise noch nicht aktuellen Lesemodell.

Weiterführende Lektüre

- Event Sourcing — Martin Fowlers Beschreibung des Musters: vollständiger Neuaufbau, zeitbezogene Abfragen, erneutes Abspielen und Gateways, die Wiederholungen von anderen Systemen fernhalten.
- CQRS Documents von Greg Young — Ereignisse als Tatsachen in der Vergangenheitsform, Umkehrung statt Löschen, ein Ereignisspeicher aus zwei Tabellen, fortlaufende Snapshots und der Ereignisspeicher als Queue.
- Die LMAX-Architektur — eine Handelsengine im Arbeitsspeicher, deren gesamter Zustand aus protokollierten Eingabeereignissen abgeleitet wird.
- PostgreSQL: Integritätsbedingungen — Eindeutigkeits- und Primärschlüsselbedingungen über mehrere Spalten, die Datenbankseite einer optimistischen Nebenläufigkeitsprüfung.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Arbeitsablauf, die vorhandenen Schnittstellen und die Nutzer, die die Veränderung benötigen. Wir können eine prüfbare Lieferung definieren und die Übergabe so gestalten, dass Ihr Team die gelieferten Bestandteile warten kann.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist Event Sourcing?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

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

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/event-sourcing/