Engineer

Event sourcing

зберігання стану як послідовності подій, що його змінили

Що це таке

Event sourcing — це спосіб зберігати дані, за якого сам запис про те, що сталося, і є даними. Більшість застосунків тримає поточний стан замовлення чи рахунку в рядку таблиці й перезаписує його за кожної зміни. Застосунок, побудований на event sourcing, натомість зберігає самі зміни: кожну з них дописують у журнал як подію, а поточний стан обчислюють із цього журналу тоді, коли він потрібен. Мартін Фаулер (Martin Fowler), який описав цей патерн у 2005 році, звів його до одного рядка: «Фіксуйте всі зміни стану застосунку як послідовність подій».

Подія фіксує те, що вже відбулося. CQRS Documents Грега Янга (Greg Young) вимагають називати кожну подію дієсловом у минулому часі, наприклад `CustomerRelocated` чи `CargoShipped`, бо назва є частиною мови, якою бізнес говорить про власну роботу. Минуле не скасуєш, тож події ніколи не редагують і не видаляють. Янг порівнює журнал із бухгалтерською книгою: помилкову проводку там не витирають, а виправляють новою, і те саме правило він застосовує до видалення. Якщо покупець поклав у кошик дві пари шкарпеток, а потім їх прибрав, у журналі з’являється подія про вилучення. Янг називає такий запис сторнуванням (reversal transaction): кошик виглядає так, ніби шкарпеток там ніколи не було, а журнал і далі показує, що вони там були.

Події зберігають у потоках, по одному на кожну сутність, стан якої вони описують. Янг спирається на предметно-орієнтоване проєктування (domain-driven design) і називає таку сутність агрегатом. Кожна подія несе ідентифікатор потоку й номер версії, унікальний і послідовний у межах цього потоку, тож порядок подій ніколи не викликає сумнівів. Щоб завантажити замовлення, застосунок читає його події за версіями й застосовує їх одну за

одною. Янг наголошує, як мало вимагається від самого сховища: у найпростішому вигляді воно має дві операції — прочитати всі події одного потоку по порядку і дописати до нього нові.

Отже, журнал стає головним джерелом правди. Фаулер зазначає, що офіційним записом може бути або журнал, або поточний стан. У системі з event sourcing це журнал, а все інше — таблиця поточних залишків, пошуковий індекс, звіт — лише похідні копії, які можна викинути й побудувати заново.

Застосуйте поняття до рішення

Застосуйте «Що таке event sourcing?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Та частина, якої користувач не бачить: де зберігаються дані, які діють правила і як інші системи можуть із нею взаємодіяти.

Переглянути наші послуги Зв'язатися з нами →

Перевірте припущення за терміном

Використайте пояснення та пов'язані роботи, щоб сформулювати запитання для проєкту. Ми можемо перевірити припущення разом до вибору інструмента. Спроєктувала й підтримувала 6 пайплайнів ETL/ELT — 6 пайплайнів ETL на трьох рушіях баз даних

Спроєктувала й підтримувала 6 пайплайнів ETL/ELT Зв'язатися з нами →

Чому це важливо

Фаулер називає три можливості, які дає повний журнал. Повна перебудова: увесь стан застосунку можна відкинути й відтворити, програвши всі події на порожньому застосунку. Часовий запит: стан на будь-який момент у минулому є результатом програвання подій до цього моменту. І повторне програвання: якщо подія виявилася хибною, система може відкотити її разом з усіма наступними, а тоді програти виправлену послідовність. Той самий журнал — це ще й журнал аудиту, про який нікому не довелося пам’ятати, і Фаулер згадує підтримку та налагодження як щоденні застосування: служба підтримки бачить, що саме зробив користувач, а розробник може програти події з продакшну в тестовому середовищі.

Янг показує вигоду на прикладі двох команд великого інтернет-магазину. Обидві отримують завдання зробити звіт про товари, які покупці кладуть у кошик, а потім прибирають. Команда, що зберігає лише поточний стан, починає фіксувати такі випадки зараз, і її звіт має дані від цього дня. Команда, що зберігає події, пише ту саму логіку звіту й проганяє її по журналу від самого початку, тож її перший звіт сягає на роки назад. На запитання, про яке ніхто не думав під час побудови системи, усе одно можна відповісти, бо система зберегла те, що сталося, а не лише те, чим усе закінчилося.

Події — це також спосіб, у який інші системи дізнаються про зміну. Янг помічає, що багато систем зберігають свій стан, а потім публікують подію в чергу, і строго кажучи, це потребує двофазної фіксації (two-phase commit) між ними, бо збій посередині лишає їх у незгоді. Його альтернатива — вважати чергою саме сховище подій: порядковий номер на кожній події дає окремому процесу змогу публікувати події по порядку, тож запис лише один. А оскільки сховище тільки дописує, його легше масштабувати. Янг зауважує, що сховищу, яке лише дописує, потрібно набагато менше блокувань, ніж тому, що оновлює дані на місці, і що ідентифікатор агрегату — єдиний ключ, за яким сховище подій узагалі доводиться розділяти на партиції.

Ціна теж реальна, і Фаулер її називає. Подавати кожну зміну як подію — стиль програмування, з яким комфортно не всім, а додати цей патерн згодом до системи, яку будували без нього, дуже важко. До того ж сховище відповідає лише на один тип запитань: «дай мені цей потік». Власний приклад Янга: сховище подій не скаже, яких користувачів звуть Грег. Для таких запитань потрібна окрема модель читання, побудована з подій, і тому Янг розглядає event sourcing і CQRS, тобто розділення моделей для запису й читання, як пару.

Як це виглядає на практиці

Найменше робоче сховище подій — одна таблиця в PostgreSQL, близька до схеми, яку дає Янг: ідентифікатор агрегату, дані події та версія. Тут вона тримає залишок одного товару в продажу. Кожна подія несе зміну кількості одиниць у продажу, а не новий підсумок, бо саме таку форму Фаулер називає найпростішою для скасування: «додай десять» скасовується відніманням десяти, а «встанови 110» уже втратило значення, яке замінило.

```
CREATE TABLE stock_events (
    item_id  text    NOT NULL,
    version  integer NOT NULL,
    event    text    NOT NULL,
    order_id text,
    units    integer NOT NULL,  -- зміна кількості в продажу
    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);

-- поточний стан: згорнути весь потік
SELECT sum(units) AS on_sale FROM stock_events
WHERE item_id = 'item-881';                    -- 99

-- часовий запит: товар таким, яким він був після версії 2
SELECT sum(units) AS on_sale FROM stock_events
WHERE item_id = 'item-881' AND version <= 2;   -- 98
```

Первинний ключ виконує ту роботу, яку Янг доручає оптимістичній перевірці конкурентності. Два оформлення замовлення, що обидва прочитали потік на версії 5 і обидва вирішили зарезервувати одиницю, намагаються дописати версію 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);
-- друга вставка падає: (item-881, 6) вже існує
```

Документація PostgreSQL описує первинний ключ на кількох стовпцях як вимогу унікальності їхньої комбінації, а вставка, що повторила б її, завершується помилкою. Оформлення, яке програло, перечитує потік, перевіряє, чи одиниця ще є в продажу, і пробує знову як версія 7. Жодне з них не продало одиниці, якої не було, а журнал показує обидва рішення в тому порядку, в якому їх ухвалили.

Нотатка Продати останній квиток рівно один раз розбирає ту саму задачу із залишками й показує, де ця схема стає тісною. Її перша відповідь тримає лічильник в одному рядку і змінює його одним умовним `UPDATE`, а оптимістичне блокування на гарячому лічильнику вона не радить, бо стовпець версії перетворює конкуренцію на повтори. Для найгарячішого товару вона кладе кожен запит в один розділ Kafka, який читає єдиний споживач: він тримає лічильник у пам’яті й фіксує рішення пакетами, і нотатка називає це архітектурою в стилі LMAX. Стаття Фаулера про архітектуру LMAX, на яку посилається нотатка, описує event sourcing у найчистішому вигляді: торговий рушій зовсім без бази даних, увесь стан якого можна вивести з вхідних подій, а ці події надійно зберігає журналювальник, щоб їх можна було програти знову. Оформлення замовлення в нотатці використовує події іншого роду. Кожен крок записує

повідомлення в таблицю outbox, change data capture публікує його далі, а таблиці лишаються головним записом. Там події оголошують про зміни; в event sourcing події і є записом.

Де виникають проблеми

Повторне програвання повторює побічні ефекти. Обробник, що надсилає підтвердження замовлення чи списує гроші з картки, зробить це знову, коли журнал програють, і Фаулер наголошує, що зовнішні системи не відрізнять програвання від справжньої обробки. Перебудова моделі читання може тоді розіслати кожному клієнтові торішні листи. Ставте кожну зовнішню систему за шлюз, який знає, коли застосунок програє журнал, і на цей час відкидає вихідні виклики, тож доменній логіці не треба про це знати. Запити потребують тієї ж уваги: приклад Фаулера — конвертація валют, де програна подія має взяти курс свого дня, а не сьогоднішній, тож шлюз повинен зберігати отримані відповіді й повертати їх під час програвання.

Завантажений потік тоне в повторах. Оптимістична конкурентність припускає, що конфлікти рідкісні. Коли багато записувачів дописують в один потік водночас, кожен конфлікт означає перечитати потік і спробувати ще раз, а під час сплеску трафіку самі повтори стають навантаженням — саме від цього застерігає нотатка вище щодо стовпців версії. Нарізайте потоки так, щоб в одного рідко було два записувачі одночасно: потік на замовлення, а не один на цілу товарну групу. Якщо окремий потік справді гарячий, дайте йому єдиного записувача, як у дизайні LMAX, і конфліктувати стане нікому.

Довгі потоки завантажуються повільно. Програвати тисячі подій, щоб завантажити один об’єкт, означає витрачати час на кожен запит, і Фаулер називає повільну перебудову серед причин зберігати копію поточного стану. Відповідь Янга — ковзний знімок (rolling snapshot): серіалізований стан потоку на відомій версії, що лежить в окремій таблиці й записується у фоні окремим процесом, тож завантаження означає найновіший знімок плюс події після його версії. Не тримайте знімки в самому журналі подій: Янг описує, як процес, що їх робить, на завантаженому потоці знову й знову провалює перевірку конкурентності. Він також радить починати без знімків, бо їх завжди можна додати пізніше, щоб пришвидшити завантаження.

Моделі читання відстають. Кожну модель читання будують із подій уже після того, як їх записано, тож вона йде позаду журналу. Фаулер попереджає, що системи, які читають журнал, легко розходяться з ним, а користувач, який

надіслав форму й побачив список без своєї зміни, надішле її вдруге. Нехай кожна модель читання записує останню застосовану подію, щоб відставання можна було виміряти, а модель перебудувати з нуля, і показуйте користувачеві результат його власної команди з відповіді на цю команду, а не з моделі читання, яка ще могла її не наздогнати.

Що почитати далі

- Event Sourcing — опис патерну від Мартіна Фаулера: повна перебудова, часові запити, повторне програвання і шлюзи, що не пускають програвання до інших систем.
- CQRS Documents by Greg Young — події як факти в минулому часі, сторнування замість видалення, сховище подій із двох таблиць, ковзні знімки і сховище подій як черга.
- The LMAX Architecture — торговий рушій у пам’яті, увесь стан якого виводиться з журнальованих вхідних подій.
- PostgreSQL: Constraints — обмеження унікальності й первинного ключа на кількох стовпцях, тобто половина оптимістичної перевірки конкурентності на боці бази даних.

Словник

Обирайте за фактичними умовами

Опишіть робочий процес, наявні інтерфейси та користувачів, яким потрібна зміна. Ми можемо визначити перевірюваний результат і спосіб передачі, який дасть вашій команді змогу підтримувати його.

Дивитися, скільки це коштує Зв'язатися з нами →

Зробіть термінологію практичною

Запитайте, як «Що таке event sourcing?» стосується вашої організації. Конкретного прикладу з поточної системи достатньо для початку розмови.

Переглянути наші послуги

welcome@engineer.company +45 50 10 61 84 Копенгаген, Данія

Візьміть цю сторінку із собою

Візитівка

Дбайливо зроблено з любов'ю — D.

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

Авторське право © 2025 – дотепер · Engineer ApS · Усі права захищено.

Надруковано з https://engineer.company/uk/glossary/event-sourcing/