Engineer

Нормалізація баз даних

побудова таблиць, у яких кожен факт зберігається лише раз

Що це таке

Нормалізація баз даних — це спосіб будувати реляційну базу даних так, щоб кожен факт зберігався в одному місці, у таблиці, яку він описує, а все інше посилалося на нього за ключем. Кажуть і в однині — нормалізація бази даних, а коли контекст очевидний, просто нормалізація. Адреса клієнта лежить в одному рядку таблиці клієнтів; замовлення містить лише ключ клієнта, а з'єднання (JOIN) зводить їх докупи, коли запиту потрібне і те, і те.

І ідея, і саме слово походять від Едгара Кодда, який запропонував реляційну модель у журналі Communications of the ACM у червні 1970 року. У тій статті нормалізація означала один конкретний крок: позбутися відношень, усередині яких є інші відношення, як-от запису про працівника з власним списком попередніх посад і власним списком дітей. Процедура Кодда копіює первинний ключ батьківського запису в кожну вкладену групу й виносить групу в окреме відношення, ключем якого стає ключ батька плюс те, що визначає рядок усередині групи. Те, що лишається, перебуває в нормальній формі, яку нині називають першою нормальною формою: кожне значення в кожному стовпці є одним простим значенням.

Згодом Кодд визначив суворіші нормальні форми, серед них ту, яку стаття Рональда Фейгіна 1977 року називає покращеною третьою нормальною формою Кодда, або нормальною формою Бойса–Кодда. Стаття Фейгіна додає четверту. Вона спирається на новий вид залежності між стовпцями, багатозначну залежність, яка охоплює добре відому функціональну залежність як окремий випадок, і Фейгін показав, що четверта форма строго сильніша за нормальну форму Бойса–Кодда.

До суворіших форм доходять розбиттям. Таблицю замінюють двома чи більше її проєкціями, тобто таблицями з частини її стовпців, дібраними так, щоб їхнє з’єднання давало рівно початкову таблицю. Стаття Фейгіна показує, що ця гарантія діє і для його форми: будь-яку таблицю можна розкласти на таблиці в четвертій нормальній формі без втрати інформації, бо з’єднання відновлює початкову таблицю. Денормалізація — зворотний процес: свідомо зберігати копію факту ще в одному місці, здебільшого щоб уникнути з’єднання.

Застосуйте поняття до рішення

Застосуйте «Що таке нормалізація баз даних?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Структура, що відповідає реальній роботі бізнесу, тож потрібні згодом запитання до даних усе ще можна буде поставити.

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

Таблиця, яка зберігає той самий факт багато разів, напрошується на три види проблем. Уявіть таблицю з рядком на кожну пару працівник–проєкт, де в кожному рядку повторюється кабінет працівника. Коли хтось переїжджає в інший кабінет, нове значення потрібне в кількох рядках, і якщо пропустити один, база даних суперечитиме сама собі. Коли останній проєкт працівника завершується і його рядок видаляють, разом із ним зникає запис про те, де людина сидить. А новому працівникові, який не бере участі в жодному проєкті, ніде записати свій кабінет. Це аномалії оновлення, видалення та вставлення, і розбиття таблиці на пари працівник–проєкт і працівник–кабінет усуває всі три.

У щоденній роботі команди ці аномалії приходять як баги. Рахунок показує стару адресу, бо одну копію пропустили. Звіт рахує клієнта двічі, бо ім’я ввели двома способами. Нормалізована схема ускладнює появу таких багів із самого початку, а коли оголошено первинні ключі, зовнішні ключі та обмеження CHECK, база даних сама відкидає погані дані, замість того щоб покладатися на те, що весь код застосунку пам’ятає правила.

Зміни теж дешевшають. Схема, де кожна таблиця описує одну річ, зазвичай приймає нові види даних із меншою перебудовою: додати стовпець до однієї таблиці, якій належить факт, — менша міграція, ніж знайти кожну таблицю, що його скопіювала.

Ціна — з'єднання. Нормалізована база даних відповідає на більшість запитань, знову з'єднуючи таблиці, і кожне з'єднання — це робота для сервера. З індексами на стовпцях, за якими з'єднують, цієї роботи зазвичай мало, тому рішення скопіювати факт назад має ухвалювати виміряний повільний запит, а не відчуття.

Як це виглядає на практиці

Невеликий інтернет-магазин починає з однієї пласкої таблиці, завантаженої з експорту електронної таблиці:

```
CREATE TABLE order_flat (
    order_no       integer,
    ordered_on     date,
    customer_email text,
    customer_name  text,
    customer_city  text,
    sku            text,
    product_name   text,
    unit_price     numeric,
    quantity       integer,
    status         text
);
```

Кожен рядок — це одна позиція одного замовлення, тож ім'я та місто клієнта повторюються в кожній позиції кожного замовлення, а назва товару — у кожній позиції, де його продано. Клієнт, який переїхав, має десятки рядків для оновлення. Якщо видалити останню позицію, в якій продано товар, таблиця забуде, що такий товар існує. Статус — довільний текст, тож можуть трапитися і `shipped`, і `Shipped`, і `shipd`.

Після нормалізації кожен факт має один дім:

```
CREATE TABLE customers (
    customer_id integer PRIMARY KEY,
    email       text NOT NULL UNIQUE,
```

```
    name        text NOT NULL,
    city        text NOT NULL
);

CREATE TABLE products (
    sku         text PRIMARY KEY,
    name        text NOT NULL,
    list_price numeric NOT NULL CHECK (list_price >= 0)
);

CREATE TABLE order_status (
    status_id smallint PRIMARY KEY,
    label     text NOT NULL UNIQUE
);

CREATE TABLE orders (
    order_id    integer PRIMARY KEY,
    customer_id integer NOT NULL REFERENCES customers ON DELETE RESTRICT,
    status_id   smallint NOT NULL REFERENCES order_status,
    ordered_on  date NOT NULL
);

CREATE TABLE order_lines (
    order_id   integer REFERENCES orders ON DELETE CASCADE,
    sku        text NOT NULL REFERENCES products,
    quantity   integer NOT NULL CHECK (quantity > 0),
    unit_price numeric NOT NULL CHECK (unit_price >= 0),
    PRIMARY KEY (order_id, sku)
);
```

Місто клієнта тепер зберігається один раз, і переїзд — це оновлення одного рядка. Назви статусів лежать у маленькій довідковій таблиці з ключем типу `smallint`, тож кожне замовлення несе коротке число замість повторюваного рядка, а статус із помилкою не пройде: зовнішній ключ відкидає будь-який `status_id`, якого немає в `order_status`.

`unit_price` лишається в `order_lines`, хоча в `products` уже є ціна. Це не недогляд. Ціна, яку заплатив клієнт, — факт про продаж, а не про товар, і вона не повинна змінюватися разом із каталогом. Нормалізація питає, де місце кожного факту, і цей належить позиції замовлення.

Дві фрази `ON DELETE` кажуть, що означає видалення. Видалення замовлення забирає з собою його позиції, бо позиція без замовлення нічого не означає. Видалити клієнта, в якого є замовлення, не вдасться, бо ці замовлення

магазин мусить зберігати. Типова поведінка PostgreSQL, `NO ACTION`, теж відкидає таке видалення; дія `RESTRICT` робить намір явним, і, на відміну від `NO ACTION`, її не можна відкласти до кінця транзакції.

PostgreSQL створює індекс для кожного первинного ключа, але не для стовпців, які посилаються через зовнішній ключ. Документація радить додати його, бо видалення чи оновлення рядка, на який посилаються, мусить знайти кожен рядок, що на нього вказує:

```
CREATE INDEX ON orders (customer_id);
CREATE INDEX ON order_lines (sku);
```

`order_lines.order_id` нічого додаткового не потребує: це перший стовпець первинного ключа цієї таблиці.

Наостанок подання (view) повертає пласку форму тим, кому вона потрібна:

```
CREATE VIEW order_report AS
SELECT o.order_id, o.ordered_on, c.email, c.name AS customer_name, c.city,
       l.sku, p.name AS product_name, l.unit_price, l.quantity, s.label
FROM   orders o
JOIN   customers c    USING (customer_id)
JOIN   order_status s USING (status_id)
JOIN   order_lines l  USING (order_id)
JOIN   products p     USING (sku);
```

Звіти можуть читати `order_report` майже так само, як читали стару таблицю, а кожен факт за ним зберігається один раз. Вступний посібник у документації PostgreSQL обстоює подання загалом із тих самих міркувань: подання дає запиту ім’я, яким можна користуватися як таблицею, і дозволяє таблицям під ним змінюватися за інтерфейсом, який лишається незмінним.

Ті самі правила діють і в набагато більшому масштабі. У межах роботи засновниці в NextMariner морську модель даних організовано як нормалізовані схеми PostgreSQL, згруповані за доменами, і, за даними її сторінки, у жовтні 2026 року модель налічувала 348 таблиць у 34 схемах. Дрібні стабільні переліки, як-от статуси, типи й категорії, стали довідковими

таблицями `SMALLINT`, що тримає рядки компактними, а з'єднання дешевими, а справжні зовнішні ключі й обмеження тримають зв'язки, тож за цілісність відповідає база даних, а не застосунок, який мав би про неї пам'ятати.

Де виникають проблеми

Списки, запаковані в один стовпець. Стовпець `tags` зі значенням `red,large,sale` або стовпець телефону з двома номерами порушує першу нормальну форму. База даних не може перевірити окремі значення, спрямувати на них зовнішній ключ чи з'єднувати за ними, і кожен запит мусить спершу розбирати рядок. Дайте списку власну таблицю, по рядку на значення, з ключем із ключа батька плюс значення: це процедура Кодда 1970 року.

Історія, знищена нормалізацією. Ціна в позиції замовлення чи адреса доставки в замовленні схожі на дублікат того, що лежить у таблицях товарів і клієнтів. Але це не дублікат: вони фіксують те, що було правдою на момент замовлення. Замініть їх ключем на поточний каталог, і кожен старий рахунок зміниться наступного разу, коли зміниться ціна. Факти про подію зберігайте разом із подією.

Зовнішні ключі без індексів. Сторона, на яку посилаються, завжди має індекс, бо мусить бути первинним ключем, обмеженням унікальності чи унікальним індексом. Сторона, що посилається, його не має, тож видалення клієнта чи зміна артикула товару означає перегляд усієї таблиці, яка посилається, у пошуках рядків, що вказують на нього. Проіндексуйте стовпці, що посилаються, як показано вище. PostgreSQL залишає це вам, бо таке потрібно не завжди.

Каскади, що видаляють важливі записи. `ON DELETE CASCADE` від клієнтів до рахунків означає, що одне необережне видалення клієнта тихо знищує дані, які бізнес мусить зберігати. Дозволяйте каскад лише для рядків, які нічого не означають без батьківського, як-от позиції замовлення, а всюди інде використовуйте `RESTRICT` або типову поведінку.

Нормалізований дизайн, який ніщо не забезпечує. Таблиці, створені вручну чи завантажені з електронних таблиць, часто не мають ні первинного ключа, ні зовнішніх ключів, тож ніщо не зупинить дубльований рядок чи замовлення для клієнта, якого не існує. PostgreSQL не вимагає первинного

ключа, хоча документація зазначає, що реляційна теорія вимагає його в кожній таблиці. Оголосіть ключі, і база даних відкине поганий рядок із помилкою, яка називає обмеження на ім'я.

Збережені суми, що розходяться. Стовпець `order_total` у `orders`, що зберігається поряд із позиціями, які він підсумовує, — це денормалізація, і кожен шлях у коді, що змінює позицію, мусить оновити й його. Шлях, який про це забуде, залишить неправильну суму без жодної помилки. Обчислюйте суму в запиті чи поданні. Якщо це виявиться надто повільним, залиште копію, але підтримуйте її в одному місці, у тій самій транзакції, що змінює позиції.

Денормалізація до вимірювань. З'єднання за проіндексованими ключами дешеві, а копіювання стовпців, щоб їх уникнути, міняє малу відому ціну на аномалії, описані вище. Спершу прочитайте план запиту. Коли читання справді надто повільні, пласку форму може тримати матеріалізоване подання чи окрема база даних для звітів, а вихідні таблиці зберігатимуть по одній копії кожного факту. Звітність — саме те місце, де денормалізація доречна свідомо, як у схемі «зірка» сховища даних.

Надмірне дроблення. Нормалізація вимагає одного місця для кожного факту, а не якомога більше таблиць. Розбиття таблиці, чиї стовпці вже залежать від ключа і більше ні від чого, додає лише з'єднання й назви таблиць, які всім доведеться вивчити. Схема, якій потрібно дюжину з'єднань, щоб показати одне замовлення, не зручніша за пласку. Зупиніться, щойно кожен факт має одне місце.

Що почитати далі

- A Relational Model of Data for Large Shared Data Banks — стаття Кодда 1970 року в Communications of the ACM, яка запровадила реляційну модель, нормальну форму та нормалізацію.
- Multivalued Dependencies and a New Normal Form — стаття Фейгіна 1977 року, яка визначає багатозначну залежність і четверту нормальну форму та показує, що будь-яку таблицю можна розкласти до четвертої нормальної форми без втрати інформації.
- Constraints — PostgreSQL про первинні та зовнішні ключі, обмеження CHECK, дії `ON DELETE` і про те, чому стовпці, що посилаються, варто індексувати.
- Views — вступний посібник PostgreSQL про подання як стабільний інтерфейс над таблицями, структура яких змінюється.

Словник

Обирайте за фактичними умовами

Надайте репрезентативний набір даних, запитання до нього та поточне вузьке місце. Узгодимо вимірювання якості, пропускної здатності та відновлення.

Дивитися, скільки це коштує Зв'язатися з нами →

Зробіть термінологію практичною

Запитайте, як «Що таке нормалізація баз даних?» 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/normalization/