Engineer

データベース正規化

各事実を対応するキーの下に一度だけ保存するテーブル設計

どのようなものか

データベース正規化は、各事実を対応するテーブルに一度だけ保存し、ほかの場所からはキーで参照するように、関係データベースを設計することです。イギリス英語ではdatabase normalisationと綴ります。顧客の住所は顧客テーブルの一行に置き、注文には顧客のキーだけを持たせます。両方が必要なクエリーでは、結合して取り出します。

考え方も言葉も、1970年6月のCommunications of the ACMで関係モデルを提案したEdgar F. Coddに由来します。その論文で正規化が指したのは、一つの具体的な手順です。たとえば、従業員のレコード内に過去の職歴と子供の一覧を持つように、ほかの関係を内部に含む関係をなくすことでした。Coddの手順では、親の主キーを入れ子の各グループにコピーし、そのグループを独立した関係として取り出します。キーには、親のキーと、グループ内の行を特定するものを組み合わせます。残るものが、現在の第一正規形に当たる正規形です。各列の各値は、単一の単純な値になります。

Coddは、その後さらに厳しい正規形を定義しました。その一つを、Ronald Faginの1977年の論文は、Coddの改良第三正規形と呼び、Boyce–Codd正規形ともいいます。Faginの論文は、第四正規形を加えます。基になるのは、多値従属性という新しい種類の列間の依存関係です。よく知られた関数従属性を特殊な場合として含み、Faginは、第四正規形がBoyce–Codd正規形より真に強いことを示しました。

厳しい正規形へ進むには、分解します。テーブルを、その列の一部からなる二つ以上の射影に置き換えます。結合すると、元のテーブルを完全に復元できるように選びます。Faginの論文は、自身が示した正規形で、その保証が成り立つと示します。どのテーブルも、結合で元に戻せるため、情報を失わず第四正規形のテーブルに分解できます。非正規化は逆で、通常は結合を一つ省くため、同じ事実のコピーを意図的に別の場所にも置くことです。

概念を判断に生かす

「データベース正規化とは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 実際の業務に合った構造を設計し、将来必要になる問いにも答えられるようにします。

サービスを見る お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 ETL/ELTパイプライン6本を設計・運用 — 6 3種類のDBエンジンにまたがるETLパイプライン

ETL/ELTパイプライン6本を設計・運用 お問い合わせ →

なぜ重要なのか

同じ事実を何度も保存するテーブルは、三種類の問題を招きます。従業員とプロジェクトの組み合わせごとに一行を置き、すべての行に従業員のオフィスを繰り返す例を考えます。オフィスが変われば、複数の行を更新する必要があり、一つ忘れると、データベース内で矛盾します。従業員の最後のプロジェクトが終わり、その行を消すと、その人がどのオフィスにいるかという情報も消えます。また、まだプロジェクトに入っていない新入社員には、オフィスを記録する場所がありません。これが更新異常、削除異常、挿入異常です。テーブルを従業員とプロジェクト、従業員とオフィスの二つの関係に分けると、三つともなくなります。

チームの日々の仕事では、こうした異常がバグとして表れます。一つのコピーを更新し忘れて、請求書に古い住所が出ます。名前を二通りで入力したため、レポートで同じ顧客を二重に数えます。正規化したスキーマでは、そもそもこうしたバグを作りにくくなります。主キー、外部キー、チェック制約を宣言すれば、アプリケーションの各部分が規則を覚えていると信じるのでなく、データベース自身が不正なデータを拒否します。

変更のコストも下がります。各テーブルが一つの対象を記述するスキーマは、新しい種類のデータを、より少ない構造変更で受け入れやすくなります。一つの事実を管理するテーブルに列を足す方が、そのコピーを持つすべてのテーブルを探すより、小さな移行で済みます。

代償は結合です。正規化したデータベースは、テーブルを結合し直して大半の問いに答え、結合のたびにサーバーで処理が発生します。結合する列にインデックスがあれば、その負荷は通常小さいものです。そのため、事実をコピーし直す判断は、直感でなく、実測で遅いと分かったクエリーを基にすべきです。

実際の活用例

小さな店が、スプレッドシートからエクスポートしたデータを、一つのフラットなテーブルに読み込むところから始めます。

```
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
);
```

各行が一つの注文の明細なので、顧客の名前と都市は、すべての注文のすべての明細に繰り返され、商品名は、その商品を売ったすべての明細に繰り返されます。顧客が引っ越すと、何十行も更新しなければなりません。ある商品を売った最後の明細を消すと、その商品が存在することも忘れてしまいます。ステータスは自由入力なので、`shipped`、`Shipped`、`shipd`のどれも入り得ます。

正規化すると、各事実の保存先を一つにできます。

```
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)
);
```

顧客の都市は、一度だけ保存され、引っ越しで更新するのは一行だけです。ステータスのラベルは、`smallint`をキーとする小さな参照テーブルに置きます。各注文は、文字列を繰り返す代わりに短い数値を持ち、誤記したステータスは入りません。外部キーは、`status_id`が`order_status`にない値なら拒否します。

`unit_price`は`order_lines`に残します。`products`にすでに価格があっても同じです。手落ちではありません。顧客が払った価格は、商品の事実でなく、販売の事実なので、カタログの価格が変わっても変えてはいけません。正規化は、各事実がどこに属するかを問います。この事実は、明細に属します。

二つの`ON DELETE`句は、削除の意味を定めます。注文を削除すると、その明細も消えます。注文なしでは、明細に意味がないからです。注文を持つ顧客の削除は失敗します。注文は、店が保存すべき記録だからです。PostgreSQLの既定の`NO ACTION`も、その削除を拒否します。`RESTRICT`は意図を明示し、`NO ACTION`と違って、トランザクションの終わりまで延期できません。

PostgreSQLは、主キーごとにインデックスを作りますが、外部キーの参照元側には作りません。参照される行を削除、更新すると、それを指すすべての行を探す必要があるため、ドキュメントは参照元にもインデックスを追加することを勧めます。

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

`order_lines.order_id`は、このテーブルの主キーの先頭列なので、追加のインデックスは不要です。

最後に、必要な人のために、ビューでフラットな形に戻します。

```
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);
```

レポートは、以前のテーブルとほぼ同じように `order_report` を読めますが、その背後の事実は、それぞれ一度だけ保存します。PostgreSQLのチュートリアルも、ビュー一般について同じ利点を挙げます。クエリーにテーブルのように使える名前を付け、変わらないインターフェースの裏で、基のテーブルを変更できます。

同じ規則は、はるかに大きい規模でも成り立ちます。NextMarinerでの創業者の取り組みでは、海事データモデルを、ドメインごとに整理した、正規化したPostgreSQLスキーマとして設計しました。そのページは、2026年10月時点で34スキーマ、348テーブルと記します。ステータス、種別、分類などの小さく安定した集合は、 `SMALLINT` の参照テーブルにしました。行を小さく保ち、結合を軽くし、実際の外部キーと制約で関係を維持します。整合性は、アプリケーションが覚えておくことではなく、データベースの仕事になります。

問題が起きるところ

一列にリストを詰め込みます。 `tags` 列に `red,large,sale` を詰めたり、電話番号の列に二つの番号を入れたりすると、第一正規形に反します。データベースは、個々の値を検査したり、外部キーで指したり、結合に使ったりできず、各クエリーで先に文字列を分割する必要があります。リストには、値ごとに一行を持つ専用テーブルを与え、親のキーと値を組み合わせてキーにします。1970年のCoddの手順です。

正規化で履歴まで消します。 注文明細の価格や、注文の配送先住所は、商品や顧客のテーブルにある値の重複に見えます。しかし、注文時点で正しかったことの記録であり、重複ではありません。現在のカタログへのキーに置き換えると、次の価格変更で、過去のすべての請求書が変わります。イベントに関する事実は、そのイベントに残します。

外部キーの参照元にインデックスがありません。 外部キーの参照先は、主キー、一意制約、または一意インデックスでなければならないため、必ずインデックスがあります。参照元にはないため、顧客を削除したり、商品のSKUを変更したりすると、それを指す行を探して、参照元テーブル全体を走査します。先の例のように、参照元の列にインデックスを作ります。必ず必要なわけではないため、PostgreSQLは判断を利用者に任せます。

カスケード削除で、保存すべき記録も消えます。 顧客から請求書への `ON DELETE CASCADE` は、顧客をうっかり一件消すだけで、業務で保存すべき記録も、気付かないうちに消すという意味です。カスケードは、注文明細のように、親なしでは意味がない行だけに使います。それ以外には、 `RESTRICT` か既定の動作を使います。

正規化した設計を、強制する仕組みがありません。 手作業で作ったり、スプレッドシートから読み込んだりしたテーブルは、主キーも外部キーもないことがよくあります。重複行や、存在しない顧客の注文を防ぐものがありません。PostgreSQLは主キーを必須にしませんが、ドキュメントは、関係理論ではすべてのテーブルに必要だと記します。キーを宣言すれば、データベースは不正な行を拒否し、制約名を含むエラーを返します。

キャッシュした合計がずれます。 `order_total` 列を `orders` に置き、合計の基になる明細と別に保持するのは非正規化です。明細を変えるすべてのコード経路で、合計も更新しなければなりません。一か所でも忘れると、エラーも出ずに合計が間違います。クエリーかビューで計算します。遅すぎると分かった場合はコピーを残せますが、明細を変えるのと同じトランザクション内の、一つの場所で維持します。

測る前に非正規化します。 インデックス付きのキーによる結合は軽く、避けるために列をコピーすると、小さく既知のコストと引き換えに、先に挙げた異常を抱えます。まず実行計画を読みます。読み取りが実際に遅すぎる場合は、マテリアライズドビューや、レポート専用の別データベースにフラットな形でデータを持たせ、基のテーブルには各事実を一つだけ保存できます。データウェアハウスのスタースキーマのように、レポートは、意図的な非正規化が適した領域です。

分解しすぎます。 正規化が求めるのは、一つの事実に一つの保存先であり、テーブルをできる限り増やすことではありません。すでに各列がキーだけに依存するテーブルを分けても、結合と、全員が覚えるべきテーブル名が増えるだけです。一つの注文を示すのに十二回の結合が必要なスキーマは、フラットなものより扱いやすくはありません。各事実の保存先が一つになったら、そこで止めます。

参考資料

- 大規模な共有データバンクのための関係データモデル — 関係モデル、正規形、正規化を導入した、Communications of the ACM掲載のCoddの1970年の論文です。
- 多値従属性と新しい正規形 — 多値従属性と第四正規形を定義し、どのテーブルも情報を失わず第四正規形に分解できると示した、Faginの1977年の論文です。
- 制約 — PostgreSQLの主キー、外部キー、チェック制約、`ON DELETE` の動作と、参照元列にインデックスを作る価値の説明です。
- ビュー — 構造の変わるテーブルの上に、安定したインターフェースを置くという、PostgreSQLチュートリアルの説明です。

用語集

実際の制約に合わせて選ぶ

代表的なデータセット、そこから答えを得たい問い、現在のボトルネックをお知らせください。品質、処理量、復旧の測り方を合意できます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「データベース正規化とは？」がお客様の組織にどう当てはまるかをご相談ください。現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く

連絡先カード

心を込めて丁寧に制作： D.

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

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/normalization/