Engineer

Normalización de bases de datos

organizar tablas: cada hecho se guarda una vez, con su clave

Qué es

La normalización de bases de datos consiste en organizar una base de datos relacional para que cada hecho se almacene una sola vez, en la tabla a la que corresponde, y todo lo demás se refiera a él mediante su clave. En inglés británico se escribe database normalisation. La dirección de un cliente ocupa una fila de una tabla de clientes; un pedido solo contiene la clave del cliente, y una combinación reúne ambos datos cuando una consulta los necesita.

Tanto la idea como la palabra proceden de Edgar F. Codd, que propuso el modelo relacional en Communications of the ACM en junio de 1970. En aquel artículo, normalizar significaba un paso concreto: eliminar la anidación de relaciones dentro de otras relaciones, como un registro de empleado con su propia lista de empleos anteriores y su propia lista de hijos. El procedimiento de Codd copia la clave primaria del elemento padre en cada grupo anidado y extrae el grupo a una relación independiente, cuya clave combina la clave del padre con lo que identifique una fila dentro del grupo. El resultado está en forma normal, hoy llamada primera forma normal: cada valor de cada columna es un único valor simple.

Codd definió después formas normales más estrictas, entre ellas la que el artículo de Ronald Fagin de 1977 denomina tercera forma normal mejorada de Codd, también conocida como forma normal de Boyce–Codd. El artículo de Fagin añade una cuarta. Se basa en un nuevo tipo de dependencia entre columnas, la dependencia multivaluada, que incluye la conocida dependencia funcional como caso particular, y Fagin demostró que la cuarta forma es estrictamente más fuerte que la forma normal de Boyce–Codd.

Se llega a las formas más estrictas mediante descomposición. Una tabla se sustituye por dos o más de sus proyecciones, tablas formadas por algunas de sus columnas, elegidas de modo que al combinarlas se recupere exactamente la original. El artículo de Fagin demuestra que esa garantía se cumple para su propia forma: cualquier tabla puede descomponerse en tablas en cuarta forma normal sin perder información, porque su combinación recupera la original. La desnormalización hace lo contrario: conservar deliberadamente una copia de un hecho en un segundo lugar, normalmente para evitar una combinación.

Utilice el concepto en una decisión

Aplique «¿Qué es la normalización de bases de datos?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. Una estructura que refleja cómo funciona el negocio para poder responder también a las preguntas que surjan más adelante.

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

Una tabla que almacena muchas veces el mismo hecho se expone a tres tipos de problemas. Imagine una fila por empleado y proyecto, con la oficina del empleado repetida en cada fila. Cuando alguien cambia de oficina, varias filas necesitan el nuevo valor, y omitir una deja la base de datos en contradicción consigo misma. Cuando termina el último proyecto de un empleado y se elimina su fila, desaparece también el registro de dónde trabaja. Y no hay dónde registrar la oficina de una persona recién contratada que aún no participa en ningún proyecto. Son las anomalías de actualización, eliminación e inserción, y descomponer la tabla en empleado–proyecto y empleado–oficina elimina las tres.

En el trabajo diario de un equipo, estas anomalías se manifiestan como errores. Una factura muestra la dirección antigua porque se pasó por alto una copia. Un informe cuenta dos veces a un cliente porque el nombre se escribió de dos maneras. Un esquema normalizado dificulta introducir esos errores desde el principio y, si se declaran claves primarias, claves foráneas y restricciones de comprobación, la propia base de datos rechaza los datos incorrectos en vez de confiar en que cada fragmento del código de aplicación recuerde las reglas.

Cambiar también cuesta menos. Un esquema cuyas tablas describen una sola cosa cada una suele admitir nuevos tipos de datos con menos reestructuración: añadir una columna a la única tabla a la que pertenece un hecho supone una migración menor que localizar todas las tablas que lo copiaron.

El coste está en las combinaciones. Una base de datos normalizada responde a la mayoría de las preguntas volviendo a combinar tablas, y cada combinación exige trabajo al servidor. Con índices en las columnas que se combinan, ese trabajo suele ser pequeño; por eso debería ser una consulta cuya lentitud se haya medido, y no una intuición, la que decida cuándo volver a copiar un hecho.

Cómo se aplica en la práctica

Una pequeña tienda empieza con una sola tabla plana, cargada desde una exportación de una hoja de cálculo:

```
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
);
```

Cada fila es una línea de un pedido, así que el nombre y la ciudad de un cliente se repiten en cada línea de cada pedido, y el nombre de un producto se repite en cada línea de venta de ese producto. Si un cliente se muda, hay que actualizar decenas de filas. Eliminar la última línea correspondiente a la venta de un producto hace que se olvide que el producto existe. El estado es texto libre, de modo que pueden aparecer `shipped`, `Shipped` y `shipd`.

Con la normalización, cada hecho tiene un único lugar:

```
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)
);
```

La ciudad de un cliente se almacena ahora una sola vez, y una mudanza se resuelve actualizando una fila. Las etiquetas de estado están en una pequeña tabla de referencia con una clave `smallint`, así que cada pedido contiene un número corto en lugar de una cadena repetida, y no puede entrar un estado mal escrito: la clave foránea rechaza cualquier `status_id` que no figure en `order_status`.

`unit_price` permanece en `order_lines`, aunque `products` ya tiene un precio. No es un descuido. El precio que pagó un cliente es un hecho sobre la venta, no sobre el producto, y no debe cambiar cuando cambie el catálogo. La normalización pregunta a qué lugar pertenece cada hecho, y este pertenece a la línea.

Las dos cláusulas `ON DELETE` indican qué significa una eliminación. Eliminar un pedido elimina también sus líneas, porque una línea no tiene sentido sin su pedido. Eliminar un cliente que tiene pedidos falla, porque esos pedidos son

registros que la tienda debe conservar. El valor predeterminado de PostgreSQL, `NO ACTION`, también rechaza esa eliminación; `RESTRICT` expresa la intención y, a diferencia de `NO ACTION`, no puede diferirse hasta el final de la transacción.

PostgreSQL crea un índice para cada clave primaria, pero no para el lado que hace referencia en una clave foránea. Su documentación sugiere añadir uno, porque eliminar o actualizar una fila referenciada exige encontrar todas las filas que apuntan a ella:

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

`order_lines.order_id` no necesita nada más: es la primera columna de la clave primaria de esa tabla.

Por último, una vista recupera la estructura plana para quien la necesite:

```
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);
```

Los informes pueden leer `order_report` de forma muy parecida a como leían la tabla antigua, mientras que cada hecho subyacente se almacena una sola vez. El tutorial de PostgreSQL defiende lo mismo para las vistas en general: una vista da a una consulta un nombre que puede usarse como una tabla y permite que cambien las tablas subyacentes tras una interfaz que permanece estable.

Las mismas reglas se cumplen a una escala mucho mayor. En el trabajo de la fundadora en NextMariner, el modelo de datos marítimos se diseñó con esquemas PostgreSQL normalizados y organizados por dominio; su página lo situaba en 348 tablas repartidas entre 34 esquemas en octubre de 2026. Los conjuntos pequeños y estables, como estados, tipos y categorías, se convirtieron en tablas de referencia `SMALLINT`, lo que mantiene compactas las filas y baratas

las combinaciones, y claves foráneas y restricciones reales mantienen unidas las relaciones, de modo que la integridad es responsabilidad de la base de datos y no algo que la aplicación deba recordar.

Dónde surgen los problemas

Listas empaquetadas en una columna. Una columna `tags` que contenga `red,large,sale`, o una columna de teléfono con dos números, incumple la primera forma normal. La base de datos no puede comprobar los valores por separado, hacer que una clave foránea apunte a ellos ni utilizarlos en una combinación, y cada consulta debe dividir primero la cadena. Dé a la lista su propia tabla, con una fila por valor y una clave formada por la clave del padre más el valor: el procedimiento de Codd de 1970.

Eliminar la historia al normalizar. El precio de una línea de pedido, o la dirección de entrega de un pedido, parece duplicar lo que contienen las tablas de productos y clientes. No es un duplicado: registra lo que era cierto cuando se hizo el pedido. Sustituirlo por una clave al catálogo actual hace que todas las facturas antiguas cambien la próxima vez que cambie un precio. Mantenga los hechos de un evento en el propio evento.

Claves foráneas sin índices. El lado referenciado de una clave foránea siempre está indexado, porque debe ser una clave primaria, una restricción de unicidad o un índice único. El lado que hace referencia no lo está, así que eliminar un cliente o cambiar el SKU de un producto implica recorrer toda la tabla que hace referencia para encontrar las filas que apuntan a él. Indexe las columnas que hacen referencia, como en el ejemplo anterior. PostgreSQL deja esa decisión en sus manos porque no siempre es necesario.

Cascadas que eliminan registros. `ON DELETE CASCADE` de clientes a facturas significa que una eliminación descuidada de un cliente borra silenciosamente registros que el negocio debe conservar. Aplique la cascada solo a filas que no tengan sentido sin su padre, como las líneas de pedido, y utilice `RESTRICT`, o el valor predeterminado, en todos los demás casos.

Un diseño normalizado que nada hace cumplir. Las tablas creadas a mano o cargadas desde hojas de cálculo suelen llegar sin clave primaria ni claves foráneas, por lo que nada impide una fila duplicada o un pedido de un cliente inexistente. PostgreSQL no exige una clave primaria, aunque su documentación

señala que la teoría relacional requiere una en cada tabla. Declare las claves y la base de datos rechazará la fila incorrecta con un error que identifique la restricción.

Totales en caché que se desajustan. Una columna `order_total` en `orders`, mantenida junto a las líneas que suma, es una desnormalización, y toda ruta de código que cambie una línea debe actualizarla también. Si alguna lo olvida, el total queda mal sin producir ningún error. Calcule el total en una consulta o una vista. Si resulta demasiado lento, conserve la copia, pero manténgala en un único lugar, dentro de la misma transacción que cambia las líneas.

Desnormalizar antes de medir. Las combinaciones mediante claves indexadas son baratas, y copiar columnas para evitarlas cambia un coste pequeño y conocido por las anomalías descritas arriba. Lea primero el plan de consulta. Cuando las lecturas sean realmente demasiado lentas, una vista materializada o una base de datos separada para informes puede conservar la estructura plana mientras las tablas de origen mantienen una sola copia de cada hecho. La elaboración de informes es un ámbito donde la desnormalización tiene cabida deliberadamente, como en el esquema en estrella de un almacén de datos.

Descomponer en exceso. La normalización pide un único lugar por hecho, no tantas tablas como sea posible. Descomponer una tabla cuyas columnas ya dependen de su clave y de nada más solo añade combinaciones y nombres de tablas que todo el mundo debe aprender. Un esquema que necesita una docena de combinaciones para mostrar un pedido no es más fácil de manejar que uno plano. Deténgase cuando cada hecho tenga un único lugar.

Lecturas adicionales

- Un modelo relacional de datos para grandes bancos de datos compartidos — artículo de Codd de 1970 en Communications of the ACM que introdujo el modelo relacional, la forma normal y la normalización.
- Dependencias multivaluadas y una nueva forma normal — artículo de Fagin de 1977 que define la dependencia multivaluada y la cuarta forma normal y demuestra que cualquier tabla puede descomponerse en cuarta forma normal sin perder información.
- Restricciones — PostgreSQL sobre claves primarias, claves foráneas, restricciones de comprobación y acciones `ON DELETE`, y por qué conviene indexar las columnas que hacen referencia.
- Vistas — argumentos del tutorial de PostgreSQL a favor de las vistas como interfaz estable sobre tablas cuya estructura cambia.

Glosario

Elija según sus restricciones reales

Traiga un conjunto de datos representativo, las preguntas que debe responder y el cuello de botella actual. Podemos acordar cómo medir la calidad, la capacidad de procesamiento y la recuperación.

Vea cuánto cuesta Póngase en contacto →

Haga práctica la terminología

Pregunte cómo se aplica «¿Qué es la normalización de bases de datos?» 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/normalization/