Modelo Relacional Normalizado

JUSTIFICACIÓN

Se presenta el diseño de la base de datos en el modelo relacional normalizado. Se omiten las clases que están trivialmente normalizadas.

Persona

Esta clase se usa para modelar cualquier persona que tenga interacción en el restaurante; tendremos que estará dividida en Clientes y Usuarios.

Decidimos usar como identificador único el email de la persona, pues de acuerdo a las necesidades expuestas, a un cliente no se le solicita CURP o RFC, por lo que tuvimos que usar el email de las personas. Por lo que tendremos:

$$\{email\} \rightarrow \{email,\ apellido\_paterno,\ apellido\_materno,\ nombre,\ teléfono,\ id\_direccion\}$$

Ahora, como mencionamos, se divide en dos secciones:

CLIENTE

Es una nueva clase que contendrá el mismo identificador como llave -email-, por lo que tendremos lo siguiente:

$$\{email\} \rightarrow \{email,\ num\_cuenta,\ contraseña,\ puntos\_acumulados\}$$

De esta forma tendremos una extensión para determinar funcionalmente a un cliente desde persona usando el email.

Notemos que está trivialmente en 1NF[1] y 2NF[2]. Además, se encuentra en 3NF[3] pues ninguno de los atributos no llave depende de alguno otro que no sea llave. Más aún, todos los atributos dependen exclusivamente del email, por lo que se encuentra en BCNF[4].

1. 1NF se refiere a Primer Forma Normal.
2. 2NF se refiere a Segunda Forma Normal.
3. 3NF se refiere a Tercer Forma Normal.
4. BCNF se refiere a Forma Normal de Boyce-Codd.

EMPLEADO

En este caso tendremos mas atributos y el email sigue identificando de manera única a un empleado, pero por motivos prácticos, la llave en esta clase será el RFC de dicho empleado.

$$\{RFC\} \rightarrow \{RFC,\ email,\ CURP,\ num\_seg\_soc,\ fecha\_nacimiento,\ tipo\_sangre,\ tipo\_empleado\}$$

De esta forma se logra mejorar que el identificador de un empleado sea único, lo cual nos ayudará posteriormente en la clase de Contrato.

Al igual, trivialmente cumple 1NF y 2NF. Con respecto a 3NF tenemos que CURP también funciona como un identificador único, sin embargo, en ninguna parte del diseño se usa el CURP como llave primaria para algún tipo de persona, por lo que no podríamos obtener información a partir del CURP, es decir los atributos de empleado no dependen transitivamente de CURP. Más aún, dado que todos dependen directamente de la llave -RFC- tenemos que se encuentra en BCNF.

Sucursal

Se define el identificador único *id_sucursal* el cual se encargará de determinar el nombre de la sucursal, el encargado y la dirección donde se encuentra.

$$\{id\_sucursal\} \rightarrow \{id\_sucursal,\ nombre,\ supervisor,\ id\_direccion\}$$

Podemos ver que trivialmente cumple 1NF, al igual que 2NF pues la llave primaria es únicamente una columna.

Esta clase también cumple 3NF pues ningún atributo no llave determina funcionalmente a alguno otro.

Contrato

Representa una relación entre un empleado y una sucursal, de esta forma obtendremos los empleados por sucursales, de esta forma podremos tener un mejor manejo de la información de las sucursales. Tendremos la siguiente dependencia funcional:

$$\{RFC\} \rightarrow \{RFC,\ id\_sucursal,\ fecha\_inicio,\ salario,\ cuenta\_cheques\}$$

Recordemos que en Empleado decidimos usar como identificador el RFC por l que esto nos permite que solo tengamos un registro de trabajador para una solo persona en cualquier

sucursal; esto se hace pues el cliente -Tacoste- solicita que un empleado únicamente pueda trabajar en una sucursal.

Vemos que cumple 1NF y 2NF sin problema alguno, y además cumpe con 3NF pues ninguno de los atributos que no son llave podrían determinar funcionalmente algun otro atributo. Y además, se encuentra en BCNF pues todos los atributos dependen directamente de la llave.

Alimentos

Se optó por la definición de una clase que contiene un identificador único, de esta forma podremos modelar el menú que ofrece la Taquería.

Se tiene la siguiente dependencia funcional:

$$\{id\_alimento\} \rightarrow \{id\_alimento,\ id\_promocion,\ nombre,\ tipo\_alimento,\ descripción\}$$

Como podemos ver, no tenemos registrado el precio del alimento, esto se debe a que existen distintos tipos de precio que están en función de las porciones que se venden (caso de las salsas), por lo que se tienen definidos en una nueva relación. Además, esta elección era necesaria pues al cliente le interesa tener un registro histórico de los precios de los alimentos, por lo que en un momento veremos cómo solucionarlo.

Notemos que cumple con 1NF y 2NF trivialmente, y también con 3NF pues cumple que todos los atributos dependen totalmente de la llave, lo que a su vez significa que está en BCNF.

SALSAS

Se crea la clase Salsa como una especialización para los alimentos que se venden; esto con el fin de poder asignarles picor y recomendación (cosa que no se pide para cualquier otro tipo de alimentos). Decidimos que picor se quedaría en esta clase como atributo dependiendo directamente de id_salsa que es la versión de llave foránea de id_alimento, por lo que se preserva la unicidad de este identificador.

$$\{id\_salsa\} \rightarrow \{id\_salsa,\ picor\}$$

Para las recomendaciones de las salsas se hizo uso de una nueva tabla la cual tendrá registrado el id_salsa y id_recomendación, donde id_recomendación es una referencia a un

alimento. Con el fin de tener múltiples recomendaciones para una misma salsa se hace uso de una llave primaria compuesta, por lo que tendremos la siguiente dependencia funcional.

$$\{id\_salsa,\ id\_recomendación\} \rightarrow \{id\_salsa; id\_recomendación\}$$

Vemos que estas dos clases están trivialmente normalizadas en BCNF.

Históricos de Precios

El cliente pide que se cree un registro de todos los precios de los alimentos en cualquier cambio nuevo.

Se deciden crear dos tipos de registros históricos pues tenemos que para las salsas necesitaremos un formato distinto pues tiene hasta 3 precios dependiendo de la porción.

HISTÓRICO DE PRECIOS DE SALSAS

Cuenta con un identificador único id_precio. Los atributos son una referencia -FK- al alimento en cuestión, su precio, la fecha de asignación del precio, entre otros. Por lo que tendremos la siguiente dependencia funcional.

$$\{id\_precio\} \rightarrow \{id\_precio,\ id\_alimento,\ inicio\_vigencia,\ ml,\ medio\_kg,\ lt\}$$

Esta relación se encuentra en BCNF pues todos los atributos definidos dependen totalmente del id_precio. Se podría pensar que dependen únicamente de id_alimento violando 2NF, sin embargo, recordemos que la idea es almacenar múltiple precios para un mismo alimento, por lo que no podríamos obtener una tupla en particular usando únicamente el id_alimento.

HISTÓRICO DE PRECIOS

Análogamente al histórico de precios para salsas, tenemos un identificador único llamado id_precio, y la diferencia entre esta relación y la anterior es que para este tipo de alimentos hay solo una porción definida, por lo que igualmente solo hay un precio.

$$\{id\_precio\} \rightarrow \{id\_precio,\ id\_alimento,\ inicio\_vigencia,\ precio\_porción\}$$

Por la misma razón que el anterior, esta clase se encuentra en BCNF.

Promociones

Tenemos definida una clase que engloba los tipos de promociones que se usan en Tacoste. Se diseñaron como especialización, una de ellas es Paquete y Descuentos, donde la primera se encarga de promociones que incluyen algún tipo de alimento (Como 1 refresco en compra de 5 tacos) y en Descuentos que se consideran como descuento en porcentaje directamente al total de la cuenta y no como tal a algún alimento gratuito bajo ciertas condiciones.

$$\{id\_promoción\} \rightarrow \{id\_promoción, inicio\_vigencia, fin\_vigencia\}$$

La cual se encuentra en 2NF pues fin_vigencia depende de alguna forma de inicio_vigencia (depende en el sentido de que la fecha de fin debe ser mayor que la de inicio), sin embargo, con el uso de triggers en este apartado podemos mejorar la seguridad de este caso.

DESCUENTOS

Se definen con la siguiente dependencia funcional

$$\{id\_promoción\} \rightarrow \{id\_promoción, porcentaje\_de\_descuento\}$$

Trivialmente está en BCNF.

PAQUETES

Esta clase es un poco mas elaborada y se presenta de la siguiente forma

$$\{id\_promoción\} \rightarrow \{id\_promoción, alimento\_paquete, cantidad\_necesarios, cantidad\_paquete\}$$

Una breve explicaición para los atributos:

1. alimento_paquete representa el alimento que se da en caos de reunir los requerimientos pedidos.
2. cantidad_necesarios es la cantidad de alimentos que se deben de comprar para dar alimento_paquete. Por ejmplo, si queremos dar un refresco en la compra de 5 tacos, tendremos que cantidad_necesarios = 5.
3. cantidad_paquete es la cantidad de elementos que se darán si cumple la restricción. Retomando el ejemplo anterior, tendríamos que cantidad_paquete = 1 pues únicamente daremos un refresco.

Órdenes

Para realizar el modelado de las órdenes necesitaremos ayuda de otra relación. Primero definamos la clase Orden, la cual está dada por la siguiente dependencia funcional.

$$\{id\_orden\} \rightarrow \{id\_orden,\ id\_sucursal,\ cliente,\ fecha\}$$

Es fácil de ver que ninguno de los atributos depende de alguno otro -excepto por la llave-, de esta forma tenemos que todos dependen únicamente de la llave primaria, por lo que cumple estar en BCNF pues los atributos son atómicos y la llave es unitaria.

PEDIDOS

Ahora, crearemos un pedido, el cual contendrá un identificador que será id_pedido. Decidimos que contendrá id_orden pues un pedido corresponde a la orden de algún cliente, id_alimento que es el alimento que se pide, cantidad pues podemos pedir directamente mas de una porción de ese mismo alimento (de esta forma ahorramos añadir la misma tupla tantas veces como cantidad, y evitamos redundancia). Además podremos calcular directamente el precio a pagar por los alimentos pedidos en este apartado.

Se tiene la siguiente dependencia funcional

$$\{id\_pedido\} \rightarrow \{id\_pedido,\ id\_orden,\ id\_alimento,\ cantidad\}$$

Vemos que esta relación se encuentra en BCNF pues todos los atributos dependen totalmente de la llave. No podríamos obtener únicamente alguno de los atributos con alguno otro que no sea la llave. Y tenemos que los atributos son atómicos.

ÓRDEN POR INTERNET

Ya que podemos crear una órden queremos clasificarlas. En este caso nos enfocaremos en aquellas que se venden por internet. Creamos una clase llamada Reparto, la cual cuenta como identificador único id_orden, es decir, será una llave foránea. Además agregaremos información acerca del reparto como el RFC del Repartidor, hora de salida, y un indicador si ya fue entregado. Algo importante de notar es que cuando una orden por internet se paga en efectivo -en persona- se asume que la orden ya fue pagada cuando el identificador de entregado es verdadero; si es falso es porque no ha sido pagada -únicamente para órdenes por internet para

pagar en efectivo-. También notemos que no necesitamos directamente la dirección del cliente pues es calculable pues la orden tiene el identificador único del cliente como atributo.

Tenemos la siguiente dependencia funcional que se encuentra en BCNF pues cumple trivialmente 1NF y 2NF y además todos los atributos de esta clase dependen totalmente del identificado id_orden.

$$\{id\_orden\} \rightarrow \{id\_orden,\ RFC\_Repartidor,\ hora\_salida,\ entregado\}$$

ORDEN EN SUCURSAL

Esta clase es mas sencilla, de nuevo necesitaremos el identificador único de la orden para saber que es lo que se pidió y agregaremos el RFC del mesero que atendió, y se almacenará también la mesa que se usó -en caso de que no hayan usado mesa, se asignó 0 como identificador-.

Nuevamente, esta relación cumple muy fácil BCNF pues id_orden es un buen identificador único.

$$\{id\_orden\} \rightarrow \{id\_orden,\ RFC\_atendió,\ mesa\}$$

Ventas

El apartado de ventas es bastante sencillo, usaremos nuevamente como llave primaria id_orden, guardaremos el empleado que cobró -que no es necesariamente el mesero o repartidor-, hora de pago, y tipo de pago -que puede ser en efectivo, tarjeta, etc.-.

Notemos que no tenemos la cantidad a pagar, esto se debe a que en orden podemos calcular el total a pagar obteniéndose a partir de la suma de los cobros de cada pedido que tenga relacionado. Y como la promoción se considera en cada uno de estos pedidos, tampoco tendremos que hacer revisión de las ofertas en este apartado pues ya se hizo. Sin embargo, esto no quiere decir que o podamos obtener el costo real en caso de que haya aplicado alguna promoción.

Recordemos que por medio de id_orden podemos obtener el cliente por lo que si requiere pagar con tarjeta no será necesario una columna para este atributo.

Igualmente, en la promoción de *Taquero Mucho,* se podrá calcular el 10% directamente en este apartado, y con el uso de disparadores actualizaremos automáticamente los puntos del cliente. De la misma forma si decide pagar con los puntos que tiene acumulados.

Finalmente, se muestra la dependencia funcional.

$$\{id\_orden\} \rightarrow \{id\_orden,\ RFC\_cobrador,\ hora\_pago,\ tipo\_de\_pago\}$$

Podemos ver que trivialmente cubre 1NF, 2NF y cubre 3NF pues no hay forma de obtener un atributo a partir de otro que no sea id_orden pues podemos tener la misma hora_pago para distintas tuplas, y lo mismo pasa con los demás atributos que no son id_orden.

De esta forma se ha diseñado la base de datos, la cual gracias a que la mayoría de las clases cuentan con normalización en 3NF y BCNF no tendremos mayor problema en expandir el diseño en el futuro si así lo desea Tacoste.