Simulacro del segundo parcial

Seguros: Una solución posible

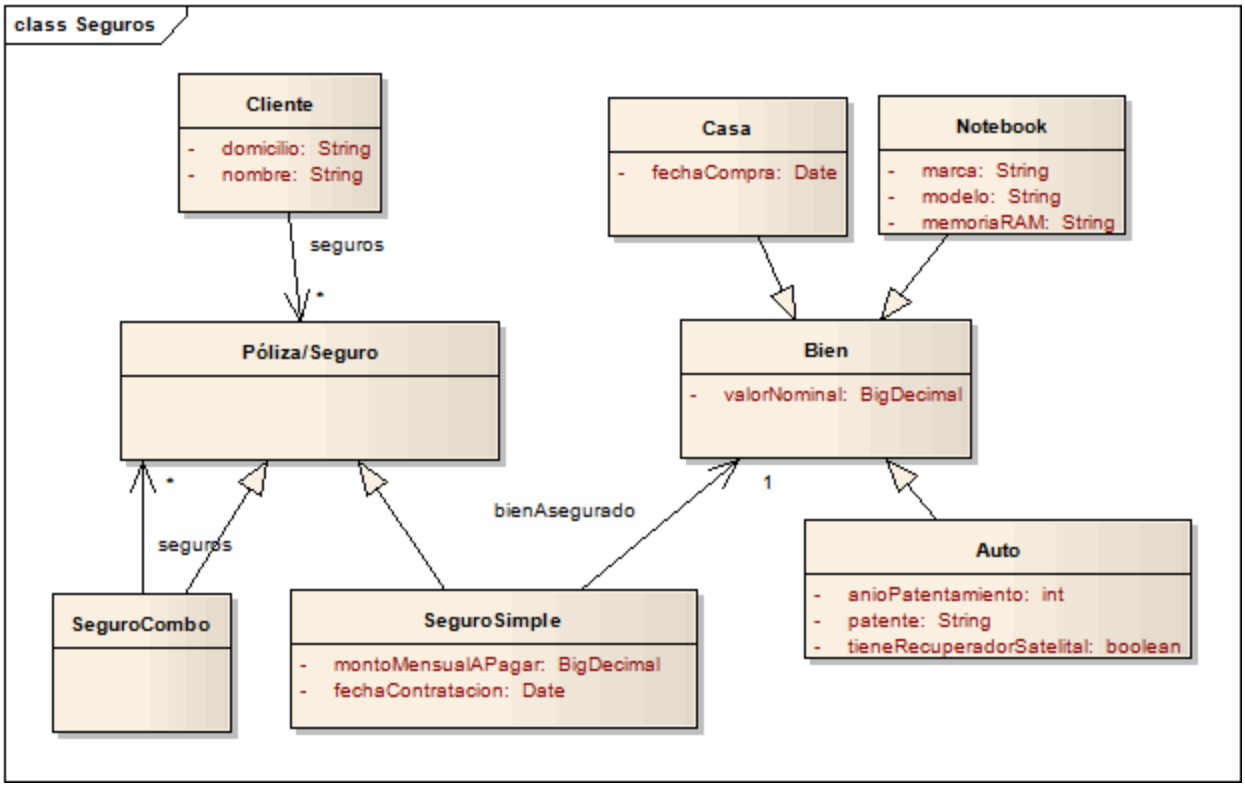
por Fernando Dodino
Octubre 2013

El enunciado

Está aquí

Punto 1

Partimos de este diagrama de objetos:



Si tenemos que trasladar esto al modelo relacional tenemos al menos dos formas de pensarlo:

- construir un DER lógico y de allí saltar al DER físico pensando en el modelo relacional
- pensar ejemplos de registros posibles para cada una de las entidades

Ambas formas de trabajar son posibles, no son antagónicas sino que se complementan: por eso no tiene sentido establecer un criterio de “mejor” o “peor” herramienta, de la misma manera que hemos visto que el diagrama de clases no es la única manera de comenzar a describir un diseño.

Vamos entonces a imaginar dos clientes posibles:

CLIE_DOMICILIO	**CLIE_NOMBRE**	**CLIE_ID (PK)**
San Blas 2671 CABA	Francis Underwood	6
Bartolomé Mitre 47 6°B CABA	Zoe Barnes	15

Lo primero que surge es que ni el nombre ni el domicilio sirven como identificadores unívocos de la relación Clientes. Entonces generamos una clave subrogada autoincremental (CLIE_ID) que cumple este objetivo. Sabemos que en objetos la identidad es resuelta por la VM + algunos trucos de redefinir equals y hashCode para cierto tipo de colecciones.

Un cliente conoce muchas pólizas o seguros. A su vez cada póliza se relaciona con un cliente(y sólo uno). En la relación “one-to-many” de Cliente a Póliza no podemos representar esa información del lado del cliente porque la definición del modelo relacional *excluye los atributos multivaluados*, entonces debemos incorporar un campo a la tabla Póliza que referencie al cliente. La forma de referenciar un registro de otra entidad es a partir del identificador unívoco, generando una foreign key (clave foránea) hacia la primary key (clave primaria) de Cliente:

POLI_MONTO_ MENSUAL	**POLI_FECHA_ CONTRATACION**	**POLI_ID (PK)**	**POLI_CLIENTE_ID (FK)**
450	09/09/2009	5	6

Claro, nos falta definir algunas cosas más todavía:

- ¿Cómo resolver la subclasificación de póliza en seguro simple y el combo? Vemos que no hay muchos atributos diferenciales entre uno y otro, por otra parte hay una relación autorreferencial o recursiva para construir el composite. Y definitivamente en un trabajo deberíamos completar nuestra visión de diseñador con algunas preguntas al usuario que permitan saber si en los casos de uso vamos a necesitar realizar queries polimórficos entre las diferentes pólizas. Por todo esto la estrategia elegida va a ser tener una sola tabla para modelar las pólizas, agregando un campo que sirva para discriminar los combos (“C”) de las pólizas o seguros simples (“S”)..
- ¿Qué tipo de colección vamos a usar para representar que un combo tiene muchas pólizas? Las preguntas son: ¿necesitamos mantener ordenadas las pólizas? no. ¿Queremos evitar tener elementos repetidos en esa colección? seguramente. Entonces el Set es el tipo de colección que queremos representar en el modelo relacional.
- Por otra parte hay una relación “x-to-one” entre póliza y bien asegurado. Un buen diseñador debería resolver la incógnita “x” preguntando al usuario si es posible que el mismo bien esté asegurado múltiples veces. Como no tenemos al usuario, vamos a

asumir que no y la relación será “one-to-one”: la póliza tiene exactamente un bien asegurado y el bien sólo puede participar de esa póliza y de ninguna otra más. Esto nos permite referenciar al bien desde la tabla Pólizas, aunque podríamos referenciar a la póliza desde la tabla Bienes.

Vemos cómo quedan los registros de la tabla Póliza, donde vamos a definir:

1. Que Francis Underwood tiene un combo de
    a. una póliza que asegura un auto BMW, contratado el 09/09/2009, paga 450 rupias por mes.
    b. otra póliza por la casa, contratada el 17/06/2010, paga 1580 rupias por mes.
2. Que Zoe Barnes tiene una póliza común por su notebook, paga 140 rupias por mes y la contrató el 13/06/2012.

POLI_ID (PK)	**POLI_ CLIENTE_ ID (FK)**	**POLI_ MONTO...**	**POLI_ FECHA...**	**POLI_ BIEN_ID (FK)**	**POLI_ TIPO_ SEGURO**	**POLI_ POLIZA_ PADRE_ID**
5	NULL (*)	450	09/09/2009	4	S	9
8	NULL (*)	1580	17/06/2010	7	S	9
9	6	NULL	NULL	NULL	C	NULL
11	15	140	13/06/2012	8	S	NULL

Aparece una gran cantidad de campos que deben admitir nulos:

- monto a pagar mensual y fecha de contratación son campos que no están definidos para el combo, de la misma manera que el bien
- de la misma manera el seguro simple no trabaja con identificador del padre
- (*) a las pólizas que participan de combos preferimos podríamos relacionarlas con el cliente, pero preferimos dejar ese campo nulo para marcar cuáles son las pólizas que participan en la colección que el cliente tiene en el modelo de objetos. De otra manera al tratar de generar las pólizas de Francis Underwood podríamos pensar que tiene 3 elementos (dos pólizas simples y un combo, identificadores 5, 8 y 9), cuando en realidad en la colección de pólizas sólo debemos referenciar al combo.

Al estar pensando nuestro modelo, permitimos relajar algunas reglas del modelo relacional:

- no podríamos insertar un registro de una póliza cuyo PADRE_ID referencie a un registro que no se haya generado todavía (ej: ID de póliza 5 cuyo padre es el ID 9)
- de la misma manera tendríamos que pensar primero en crear el bien asegurado para luego asociarlo a la póliza.

Ahora tenemos que definir una notebook, un auto y una casa como bienes asegurados de las

pólizas que escribimos.

¿Qué estrategia vamos a adoptar para trabajar la subclasificación? En este caso tenemos una gran cantidad de campos disjuntos entre sí. Lamentablemente no sabemos de la necesidad de generar consultas polimórficas: “cuántos bienes tienen un valor nominal de más de 50.000 rupias”, “cuántos bienes de lujo están asegurados” (definiendo luego qué significa que sean de lujo), etc. Si los casos de uso se plantean más en términos de conocer primero la póliza para luego capturar la información del bien asegurado, nuestra estrategia puede ser a) tener una tabla por subclase concreta o b) utilizar (n+ 1) tablas. En este ejemplo mostraremos la opción b) aunque podríamos haber elegido la a) de la misma manera.

- El auto vale 100.000 rupias
- La casa: 4.700.000 rupias
- La notebook: 2.300 rupias

Tabla Bienes

BIEN_ID (PK)	**BIEN_VALOR_NOMINAL**
4	100000
7	4700000
8	2300

Ahora definimos 3 tablas, una por subclase:

Tabla Autos

AUTO_ID (PK)	**AUTO_ BIEN_ID (FK)**	**AUTO_ANIO**	**AUTO_PATENTE**	**AUTO_ RECUPERADOR**
1	4	2007	BGU 726	SI

Tabla Casas

CASA_ID (PK)	**CASA_BIEN_ID (FK)**	**CASA_FECHA_COMPRA**
1	7	16/06/2010

Tabla Notebooks

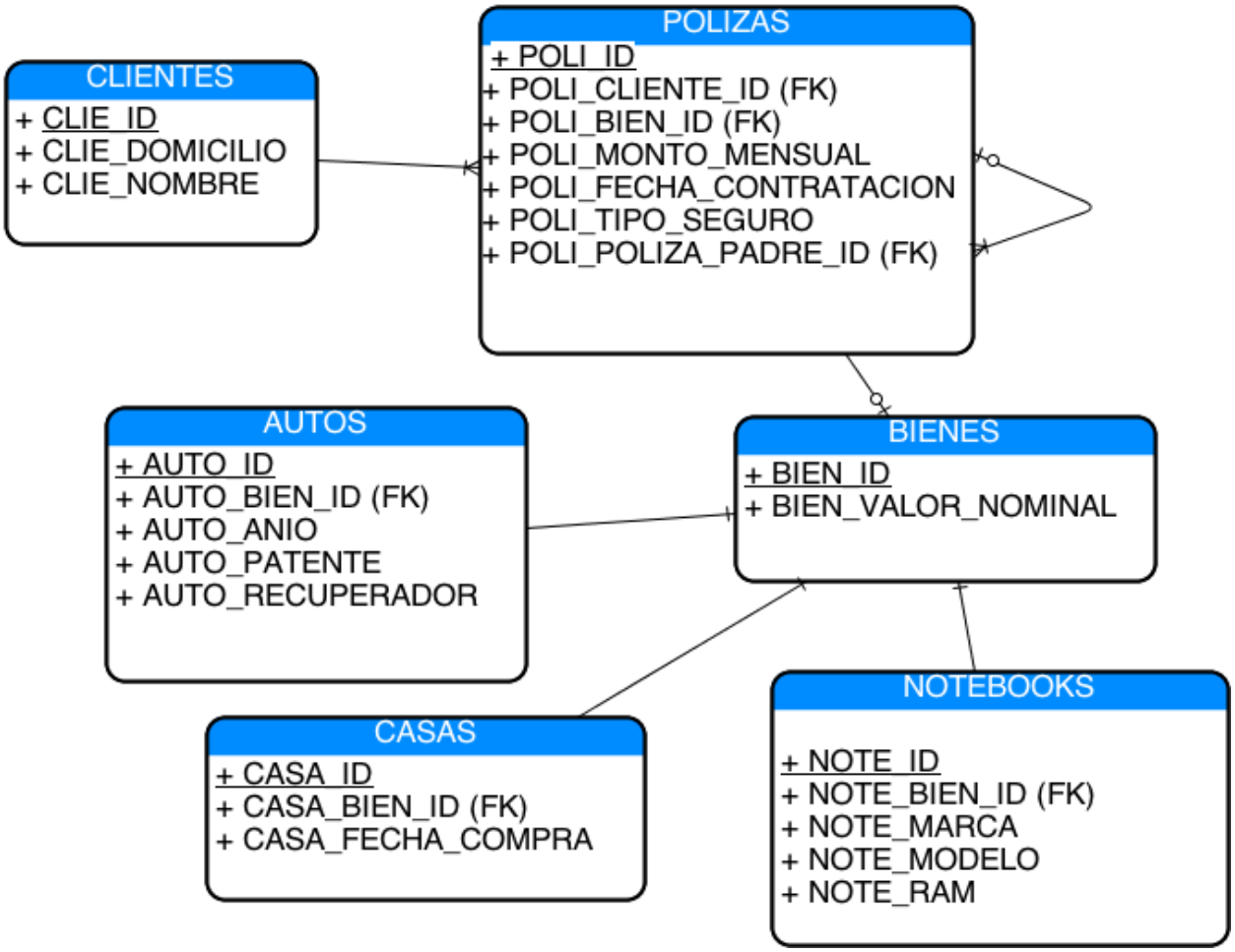
NOTE_ID (PK)	**NOTE_BIEN_ID**	**NOTE_MARCA**	**NOTE_MODELO**	**NOTE_RAM**

	(FK)			
2	8	ASUS	Taichi 21	8

Nuestra validación del modelo se completaría:

- escribiendo un script con la definición de la estructura de datos (DDL)
- luego generando un fixture o juego de datos (DML)
- e incorporando consultas que testeen el modelo.

Pero como tenemos fines didácticos más que laborales, nos alcanza con un DER:



Y con explicaciones acerca de algunas conversiones de tipos de dato:

- String vs. varchar: el domicilio se representa en objetos como un String, mientras que debemos especificar la longitud máxima para el RDBMS.

- En el caso del auto, estamos modelando el Boolean recuperadorSatelital de objetos con un varchar(2) “SI”/”NO” (podríamos trabajarlo con un tinyint o un int a secas, 0 ó 1)
- Los dates del motor tampoco coinciden con la representación de java.util.Date, aunque esto depende de la capacidad que tenga el componente OR/M para resolverlo.

> Nota de implementación: JPA/Hibernate en realidad por defecto se utilizará una sóla columna id en la tabla BIENES, que es PK, y una sóla columna ID, en CASAS, NOTEBOOKS y AUTOS, que sea, en cada una de ellas, al mismo tiempo PK y FK hacia BIENES. Es decir, no es necesario tener, por ejemplo en AUTOS, dos columnas AUTO_ID y AUTO_BIEN_ID.

Punto 2

Ahora incorporamos los siniestros. Las decisiones que debemos tomar son:

- *normalizar o mantener desnormalizados los datos del cliente*: las preguntas que deberíamos hacer es
    - ¿necesito almacenar la información del cliente en el momento de tener el siniestro?
    - ¿necesitamos hacer consultas sobre todas las pólizas? ¿hay casos de uso que involucren este tipo de consultas? ¿hay una cantidad enorme de pólizas (+ de 10.000.000) y de clientes de manera de que el JOIN de ambas tablas resulte costoso y pueda degradarse la performance?

  En base a lo que conocemos, que no es mucho, creemos que no es necesario desnormalizar los datos del cliente.
- si aplicamos la forma canónica el atributo cantidadSiniestros es calculable recorriendo la lista de siniestros asociados a un cliente. Pero 150.000 consultas diarias vs. el costo de mantener ese dato actualizado nos dan un indicio de que sí puede ser útil guardar esta información en la tabla de clientes.
- y está claro que si bien podemos normalizar los datos del bien, el valor nominal está asociado al momento en el que ocurre el siniestro. Entonces vamos a trabajar ese campo tanto en la entidad Siniestro como en el Bien.
- los siniestros conocen dos entidades
    - el empleado que sigue el expediente
    - y la póliza involucrada

  No tenemos reglas de negocio que induzcan a conservar estos datos desnormalizados.

Creamos nuestro modelo en base a estas decisiones, de nuevo con ejemplos concretos:

CLIE_DOMICILIO	**CLIE_NOMBRE**	**CLIE_ID (PK)**	**CLIE_CANT_SINIESTROS (PRECALCULADO)**

San Blas 2671 CABA	Francis Underwood	6	1
Bartolomé Mitre 47 6°B CABA	Zoe Barnes	15	4
Barugel Azulay 39 Villa Crespo	Nicolás Eliaschev	20	2

Esto permite conocer la cantidad de siniestros de un cliente accediendo a un único registro. Modelamos el siniestro del ejemplo:

Siniestros

SINI_ID (PK)	**SINI_POLIZA_ID (FK)**	**SINI_MONTO_ PAGADO**	**SINI_EMPLEADO_ID**	**SINI_BIEN_VALOR**
2	1	145000	2	300000
3	15	NULL	2	4000

Si necesitamos conocer “los clientes que tuvieron siniestros”, o consultas similares en las que necesitamos obtener rápidamente el siniestro y el cliente relacionado, podría considerarse como opción redundar el campo CLIENTE_ID en la tabla de siniestros. Pero esta decisión de diseño surge cuando tenemos más en claro los casos de uso de nuestra aplicación y un volumen de datos considerable.

Las pólizas no sufren cambios en su conformación.

Y por último modelamos los empleados:

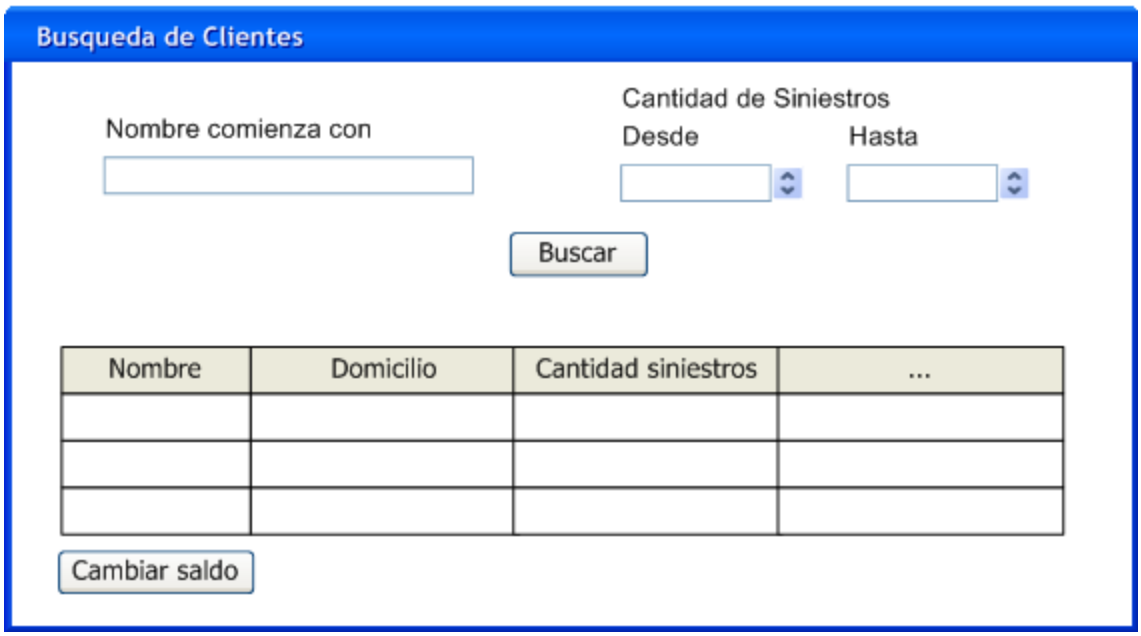
Empleados

EMPL_ID (PK)	**EMPL_NOMBRE**	**EMPL_LEGAJO**
2	Juan Domínguez	21429
...		

Punto 3

Definimos la UI:

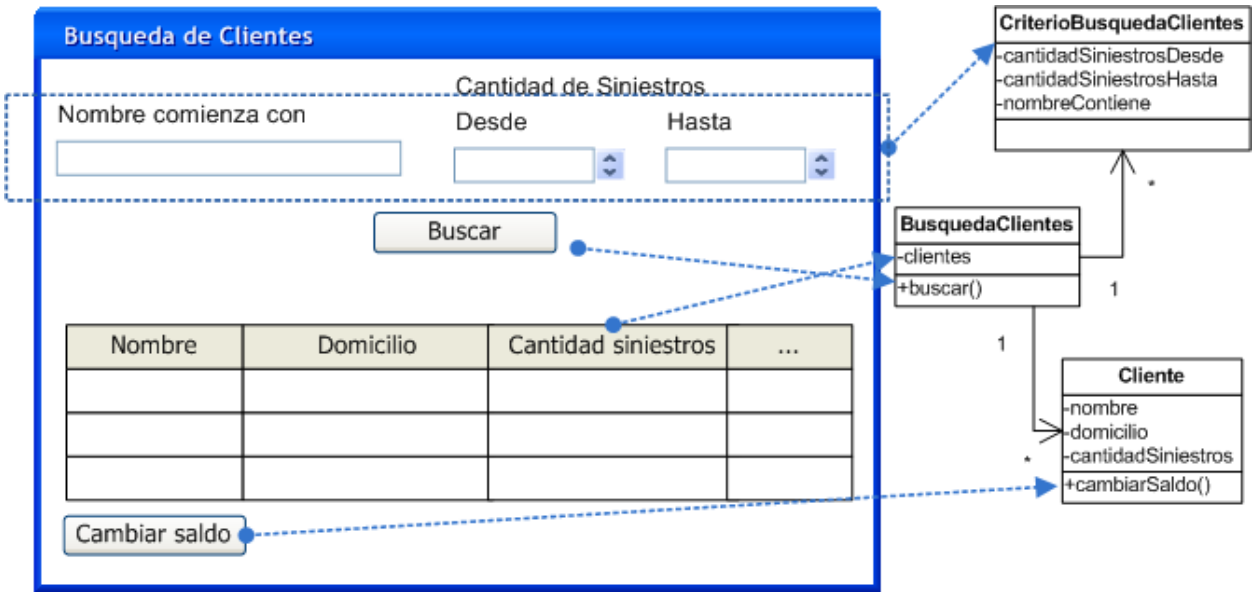
1)



2) Definimos un objeto BusquedaClientes que tendría como atributos

- el criterio de búsqueda:
  - nombre contiene
  - rango desde y
  - rango hasta de cantidad de siniestros
- lista de resultados encontrados (objetos de la clase Cliente)
- cliente seleccionado
- propiedad habilitaCambiarSaldo de sólo lectura booleana (cuando el cliente seleccionado no es nulo)
- y un método buscar que delega al home

Mostramos el binding:



(hay que definir cómo se resolvería el hecho de cambiar el saldo, si se abre otra ventana para ingresar un número, etc.)

3) La ausencia de este objeto nos obligaría a que la vista tome esa responsabilidad:

- en diseño esto equivale a bajar su cohesión (hacer más cosas implica una oportunidad perdida para dividir trabajo)
- se entremezclan cuestiones no-tecnológicas con lo estrictamente tecnológico de presentación
- con el objeto BusquedaClientes podemos hacer testing unitario automatizado simulando el ingreso por pantalla de un criterio y la posterior búsqueda. Sin este objeto todo queda en la vista y se dificulta mucho más esta posibilidad.