Proyecto 2 Sistrans documentación:

Luis Pinilla

Juan Esteban Quiroga

Mateo Rincón

El proyecto 2 se Sitrans es una continuación del proyecto 1 por lo que tener un buen primer proyecto era fundamental para tener un buen segundo proyecto. Pero este no fue nuestro caso y en realidad tuvimos muchos problemas con la implementación del proyecto 1 Revisando la primera entrega notábamos que teníamos problemas desde el modelo relacional. Esto provocó que tuviéramos relaciones de herencia erradas ya que hay entidades que a pesar de tener atributos similares se comportaban de maneras muy distintas. Además de esto nos dimos cuenta de que en algunos casos acceder a la información era muy complicado. Por ejemplo, entre la tabla de oficinas y direcciones había 2 entidades o tablas intermedias lo que hizo la base de datos muy confusa. Otro error que notamos es que tomamos a una operación como una entidad cuando se puede reemplazar por un log que sea mucho más simple y así también simplificamos el modelo.

Entre otras cosas, estos fueron los errores principales que nos llevaron a decidir partir desde ceros de nuevo y tener un modelo mucho más sólido. El partir desde ceros nos sirvió porque ya contábamos con la experiencia de nuestros errores y ahora en nuestra segunda versión si tuviésemos una implementación completamente funcional cómo estaba estipulado en el proyecto 1, donde además de todos los errores y complicaciones que tuvimos, no alcanzamos a implementar todo el front de la aplicación.

Comparación con proyecto 1:

Diagrama UML proyecto 1:

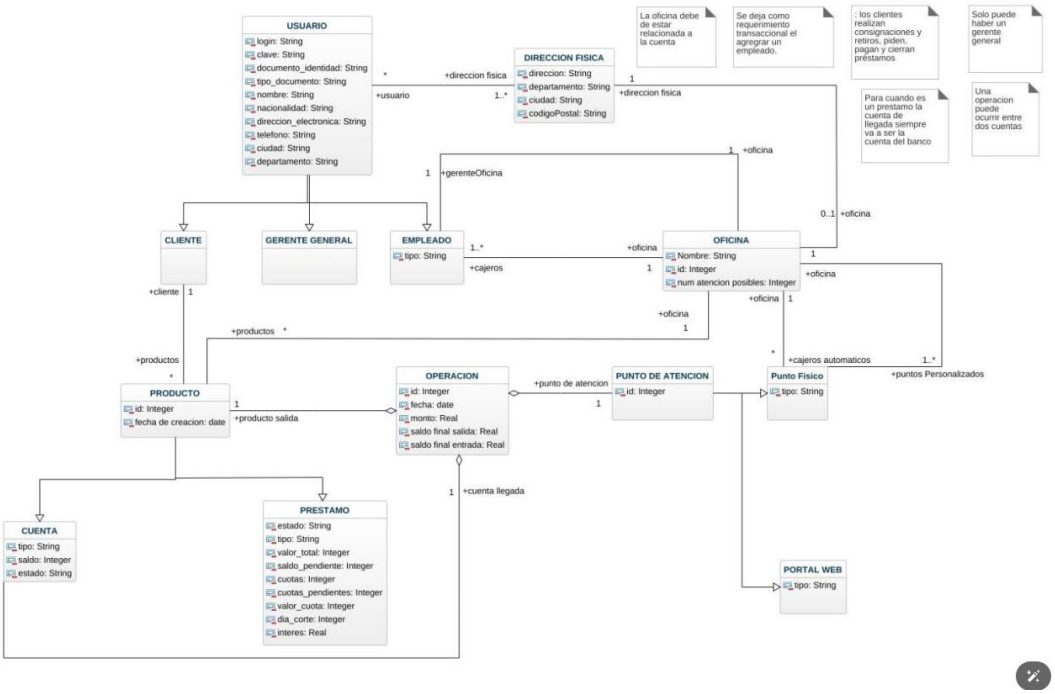
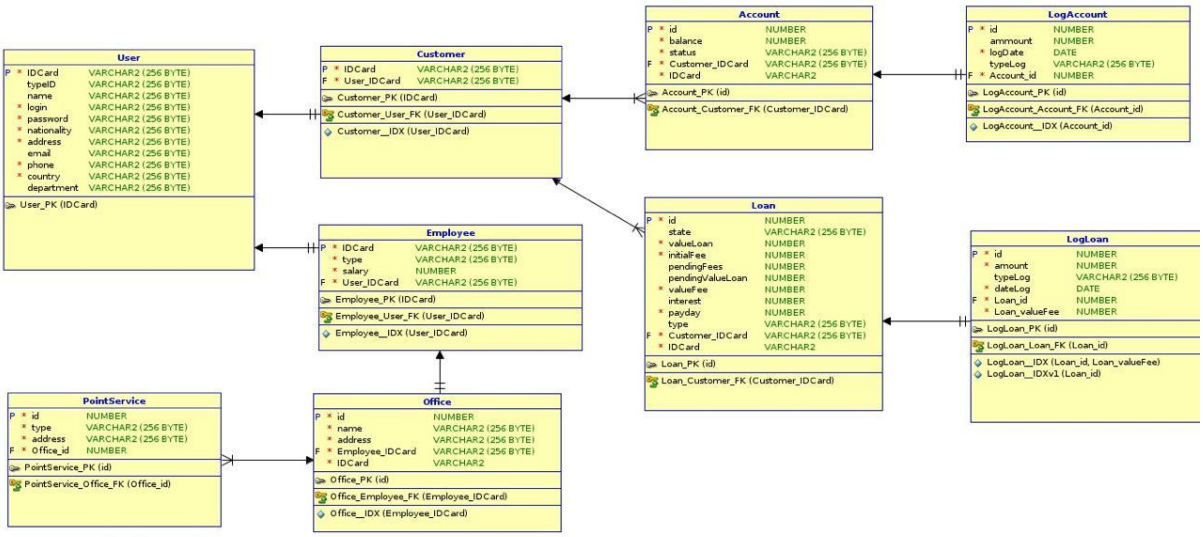


Diagrama proyecto 2:



Cómo se puede ver en los diagramas, es evidente que el segundo es mucho más compacto. Esto pasa porque en el primer proyecto definitivamente pasamos por alto el verdadero propósito de una base de datos que es que sea dinámica y permita fácilitar el acceso a información. En otras palabras, se debe buscar una base de datos que sea lo más simple posible y que sea dinámica para priorizar el buen funcionamiento de esta. Lo que hicimos en el primer proyecto fue básicamente listar todo lo que vimos en el enunciado y posteriormente intentamos implementarlo y ahí fue cuándo nos dimos cuenta de que teníamos una base de datos muy mal planteada.

Así quedaron organizadas nuestras tablas en el SQL developer de Oracle:

REQ 1: Crear usuario

En este requerimiento realmente la sentencia SQL es idéntica a la de la primera entrega y aunque en la segunda intervinimos mucho las relaciones, las tablas se mantuvieron muy similares entonces una vez implementada la base, este requerimiento ya lo teníamos.

REQ 2: Crear oficina

En este requerimiento, así como en el primero conservamos gran parte de la tabla original, así que en SQL no cambiamos nada, pero las direcciones físicas son más fáciles de consultar, ya que fue un arreglo notable, hay una relación directa entre la tabla de direcciones físicas y oficinas.

REQ 3: Crear y borrar punto de atención

Para esta nueva implementación del proyecto decidimos ligar a las oficinas y los puntos de atención que se pueden dividir en puntos o cajeros automáticos. La división es necesaria porque cada uno cumple funciones distintas y se comportan de distintas maneras. Esto también simplifico la manera de acceder a las direcciones de cada punto de atención sin importar du tipo.

REQ 4: Crear cuenta

Para crear una cuenta nos sostuvimos con la sentencia SQL que plantemos en la primera implementación y esta tabla no sufrió cambios más que unas relaciones así que fue cuestión de poner las llaves foráneas faltantes y quedó listo.

REQ 5: Cambio de estado de cuenta

Este requerimiento es realmente bastante simple ya que una vez creada la tabla solo era necesario hacer un update del valor del estado de la cuenta, lo que es una columna en la tabla de cuentas. Solo había que determinar si la cuenta se desactivará o cerrará y se aplica el cambio.

REQ 6: Registrar operación sobre cuenta

Este requerimiento es el que sufrió el mayor cambio en comparación con la primera implementación. Lo que hicimos es que la entidad operación dejó de existir y ahora convertimos el registro en un log. De esta manera nos evitamos estar buscando información por toda la aplicación y logramos simplificar mucho el proceso de registrar la información requerida del registro de cada operación.

REQ 7: Crear préstamo

Los préstamos son entidades que antiguamente guardábamos con una relación de herencia, pero en nuestro nuevo modelo nos dimos cuenta de que era un despropósito. Ahora solo usamos una sentencia SQL para crear un préstamo.

REQ 8: Registro de operación sobre préstamo

Este requerimiento sufrió un cambio muy grande, similar a lo que hicimos con el requerimiento 6. La tabla de registro de operaciones sobre cuentas pasó a ser un log y lo mismo ocurrió con el requerimiento 8 donde cambiamos una entidad por un registro en log.

REQ 9: Actualizar préstamo a cerrado

El requerimiento 9 es de nuevo muy parecido al requerimiento que pedía lo mismo, pero para una cuenta que en este caso es el requerimiento número 7. Al tener una columna dedicada el estado de cada préstamo nos es posible actualizar el atributo mediante un update en SQL para registrar el cambio de estado del préstamo para saber que ahora está cerrado.

Requermientos de consulta:

Para todos estos requerimientos hay que notar que sólo consultar una base de datos no pretende alterarla entonces mediante selects podremos obtener toda la información que necesitamos. Dicho esto, podemos enfatizar en cada requerimiento.

REQ de cosulta 1: Consultar las cuentas en BancAndes

El proceso de consultar una cuenta es bastante simple ya que contamos con una tabla y de ahí podemos traer toda la información o filtrarla según sea necesario con SQL ya que los filtros son, entre otros, tipo de cuenta, rango de saldos, fecha de creación, fecha del último movimiento, cliente podemos dar la opción de todos esos filtros para mandarlos como condición en sentencia SQL y traer la información que es requerida.

REQ de consulta 2: Consultar un cliente

Para obtener la información de un cliente hemos determinado que la llave primaria sea el id de cada cliente así que al incluir el id en la sentencia SQL podremos tener la información del cliente. Es importante aclarar que solo los empleados y gerentes generales podrán tener acceso a esta información para cumplir con los lineamentos de privacidad.

REQ de consulta 3: Extracto bancario para una cuenta

La sentencia SQL de este requerimiento es especial porque no nos piden toda la información de una cuenta, pero esto en sencillo porque en vez de poner un asterisco en la sentencia SQL, indicando que queremos toda la información, podemos poner solo los atributos que nos interesan que son: saldo inicial antes de empezar el mes, la lista de operaciones de consignación, retiro y transferencia sobre la cuenta, y, por último, el saldo al final del mes. Aparte de eso no es muy distinto a cualquier búsqueda en SQL.

2. Nivel de Aislamiento:

SESION 1	T	SESION 2
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;	T1	
-- Crear una cuenta para el cliente 1 INSERT INTO account (id, balance, status, customer_idcard) VALUES (1, 1000.00, 'Active', '1234567890');	T2	
-- Actualizar el LOG agregando la cuenta 1 INSERT INTO logaccount (id, ammount, logdate, typelog, account_id) VALUES (1, 0, SYSDATE,'Consignación',1);	T3	
-- Actualizar el saldo de la cuenta de Juan Pérez sumándole 1 millón de pesos UPDATE account SET balance = balance + 1000000 WHERE id = 1;	T4	SET TRANSACTION ISOLATION LEVEL READ COMMITED;
-- Actualizar el LOG sumándole 1 millón de pesos INSERT INTO logaccount (id, ammount, logdate, typelog, account_id) VALUES (1, 1000000,SYSDATE,'Consignación',1);	T5	
	T6	-- Crear otra cuenta para el cliente 2

		INSERT INTO account (id, balance, status, customer_idcard) VALUES (2, 500.00, 'Active', '0987654321');
COMMIT;	T7	
	T8	-- Actualizar el saldo de la cuenta 2 restando 50 unidades monetarias UPDATE account SET balance = balance - 50 WHERE id = 2;
	T9	-- Actualizar el LOG quitandole 50 de pesos INSERT INTO logaccount (id, ammount, logdate, typelog, account_id) VALUES (2, -50,SYSDATE,'Retiro',2);
	T10	INSERT INTO logaccount (id, ammount, logdate, typelog, account_id) VALUES (3,5,SYSDATE,'Intereses',2);
-- Consulta del saldo de la cuenta CUENTA_1) SELECT balance FROM account WHERE id = 1; -- Consulta del saldo de la cuenta CUENTA_2 SELECT balance FROM account WHERE id = 2;	T11	
	T12	COMMIT;
-- Consulta del saldo de la cuenta CUENTA_1) SELECT balance FROM account WHERE id = 1; -- Consulta del saldo de la cuenta CUENTA_2 SELECT balance FROM account WHERE id = 2;	T13	-- Consulta del saldo de la cuenta CUENTA_1) SELECT balance FROM account WHERE id = 1; -- Consulta del saldo de la cuenta CUENTA_2 SELECT balance FROM account WHERE id = 2;

Las operaciones en S1 se ejecutan con normalidad hasta T11, y en cambio, en S2 para las primeras operaciones (T4, T6) tiene que esperar hasta que S1 suelta los recursos, sin embargo, hasta el momento no se han causado problemas relacionados con la concurrencia.

Para T11, S2 ya ha añadido los datos de la transacción, otra vez sin problemas, sin embargo, para T11 sucede una consulta sucia, pues se lee la información de la cuenta id = 2, pero esta información no ha sido confirmada (commit), por lo que ocurre una lecutra sucia.

Finalmente en T12, T13 no se modifican los datos, tampoco se agregan, por lo que simplemente se muestran los datos que ambas transacciones han modificado.