Universidad de los Andes Colombia | Departamento de Ingeniería de Sistemas y Computación

SISTEMAS TRANSACCIONALES - ISIS2304

PROYECTO 1 ENTREGA 2

Franklin Smith Fernandez Romero *Código: 202215103*

f.fernandezr@uniandes.edu.co

Andres Mateo Chilito Avella *Código: 202214992*

a.chilitoa@uniandes.edu.co

Este documento presenta la solución para el proyecto número uno entrega dos de la materia ***Sistemas Transaccionales***.

Universidad de los Andes

Bogotá - Colombía

19 de marzo de 2024

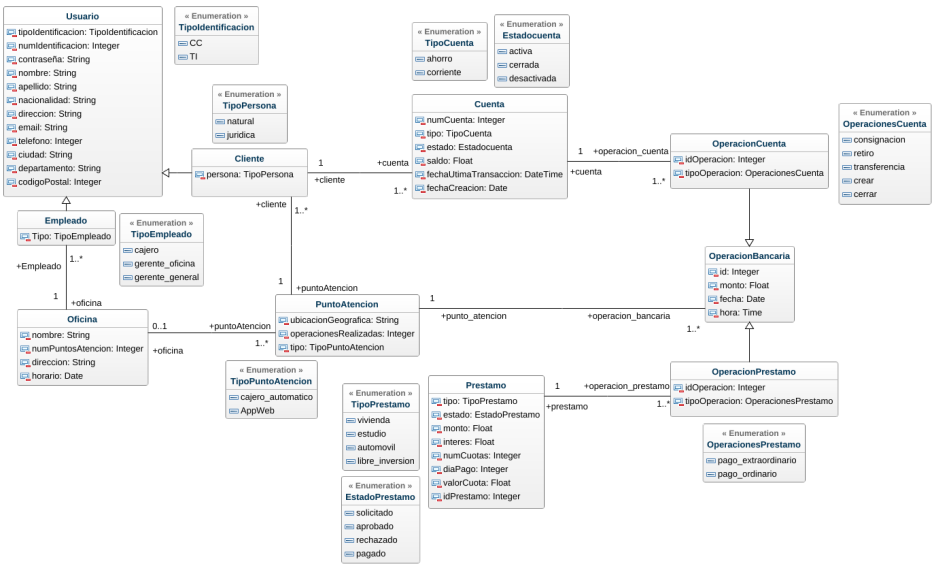

Figura 1: UML corregido conforme a la retroalimentación de la entrega uno

BANCO DE LOS ALPES

1. **Ajuste del modelo conceptual UML**

   En la revisión de la entrega anterior, se identificaron confusiones relacionadas con la representación incorrecta de las cardinalidades, específicamente entre los símbolos (*) y (1..*), dentro de nuestro modelo de diagrama UML. Este problema emergió principalmente debido a algunas confusiones en la coordinación entre los miembros del equipo, lo que resultó en una falta de uniformidad al documentar la relación de cardinalidad üno o muchos". Para abordar estas deficiencias de manera efectiva y mejorar la calidad de nuestra documentación técnica, hemos realizado una evaluación de nuestras prácticas de comunicación y documentación interna. A través de esta introspección, hemos reconocido la importancia crítica de adherirnos a convenciones y estándares uniformes en la representación de relaciones de cardinalidad en diagramas UML, especialmente para expresar relaciones de manera precisa e inequívoca. Finalmente, para esta entrega, hemos tomado medidas correctivas estratégicas, implementando un protocolo de documentación mejorado que estipula (1..*) como la cardinalidad predeterminada para representar la relación üno o muchos". Esta decisión se fundamenta en el objetivo de armonizar nuestra representación de relaciones de cardinalidad, promoviendo así una mayor coherencia, claridad y precisión técnica en nuestro modelo UML. Este enfoque estandarizado no solo facilitará una interpretación más clara y unificada por parte de los stakeholders del banco de los andes, sino que también reforzará la integridad y la fiabilidad de nuestro diseño de sistema en fases subsecuentes de desarrollo y análisis.

2. Se implementa un proceso de evaluación y normalización detallado, enfocado primordialmente en asegurar que cada relación dentro de la base de datos cumpla con los estándares de normalización que hemos visto en clase. Este proceso se centra en la verificación de que todas las relaciones alcancen la Forma Normal de Boyce-Codd (BCNF). La BCNF está diseñada para resolver posibles anomalías en la inserción, actualización, y eliminación de datos, que podrían surgir incluso después de haber alcanzado la 3NF. Para que una relación se considere en BCNF, debe satisfacer dos criterios fundamentales: primero, debe estar en 3NF; y segundo, para cada una de sus dependencias funcionales, el lado izquierdo debe ser una llave candidata. Este enfoque no solo mejora la integridad y la consistencia de los datos, sino que también optimiza el rendimiento de la nuestra base de datos al minimizar la redundancia y facilitar el mantenimiento. El procedimiento para alcanzar la

BCNF implicó identificar y analizar cada dependencia funcional dentro de las relaciones, y aplicar descomposición en caso necesario, de manera que se preserven las dependencias funcionales y se garantice la pérdida de información.

Partiendo de lo anterior se ha llevado a cabo una revisión exhaustiva de las dependencias funcionales y las formas normales hasta la Forma Normal de Boyce-Codd (BCNF). Este proceso de normalización ha garantizado la eliminación de redundancias, la minimización de anomalías de inserción, actualización y eliminación, y el mantenimiento de la integridad de los datos en la base de datos.

El modelo de base de datos ha sido diseñado con una especial atención a las restricciones de integridad, tales como claves foráneas y únicas, y la implementación de reglas de negocio a nivel de la lógica de la aplicación. A continuación, se describen brevemente las etapas de normalización evaluadas:

a) Primera Forma Normal (1NF): Cada entidad presenta atributos con valores atómicos y únicos en cada instancia, asegurando la ausencia de grupos repetitivos.

b) Segunda Forma Normal (2NF): Todos los atributos no clave dependen funcionalmente de la totalidad de la clave primaria, eliminando las dependencias funcionales parciales.

c) Tercera Forma Normal (3NF): Se identifico que los códigos postales por lo general sirven para la identificación de la ubicación geográfica regional, es decir, suplen el papel de los datos de ciudad y departamento y sus homónimos. por lo anterior se genero la tabla ciudad-departamento que tiene los valores de ciudad y departamento asociados a un único código postal. ya con esto arreglado el resto de tablas cumplen con 3FN

d) Forma Normal de Boyce-Codd (BCNF): Cada determinante es una llave candidata, y no existen dependencias funcionales que violen esta condición. Este nivel de normalización asegura la integridad estructural y lógica de nuestro diseño.

Aunque la base de datos cumple con las formas normales hasta BCNF, ciertos controles y validaciones se han delegado a la lógica de la aplicación. Esto incluye restricciones sobre quién puede realizar ciertas operaciones bancarias, la asociación entre empleados y oficinas, y el manejo de la privacidad en las consultas.

3. Se realiza el archivo esquemaSQL que crea nuestras tablas, además se adjunta otro archivo que llena las tablas llamado poblacionSQL.
   - ◇ los archivos se encuentran en el directorio del proyecto: //implementacion/archivos.sql/*
4. Puede verificar la pagina web en los archivos adjuntos para la carpeta proyecto uno. Cabe resaltar que en cada clase se encuentra la documentación respectiva de su funcionamiento y demás. Aunque documentamos acá un poco de la estructura de la base de datos:

Resumen de la Estructura de la Base de Datos de BancAndes

Tabla `USUARIOS`

- ◇ **Clave primaria**: `NUMIDENTIFICACION`
- ◇ **Campos principales**: `TIPO_IDENTIFICACION`, `CONTRASENA`, `NOMBRE`, `APELLIDO`, `NACIONALIDAD`, `DIRECCION`, `EMAIL`, `TELEFONO`, `CIUDAD`, `DEPARTAMENTO`, `CODIGO_POSTAL`

Tabla `OFICINAS`

- ◇ **Clave primaria**: `NOMBRE`
- ◇ **Campos principales**: `NUM_PUNTOS_ATENCION`, `DIRECCION`, `HORARIO`

Tabla EMPLEADOS

- **Clave primaria**: NUM_IDENTIFICACION
- **Campos principales**: TIPO_EMPLEADO, OFICINA

Tabla PUNTOSATENCION

- **Clave primaria**: UBICACION_GEOGRAFICA
- **Campos principales**: OPERACIONES_REALIZADAS, TIPO_PUNTO_ATENCION, OFICINA_NAME

Tabla CLIENTES

- **Clave primaria**: NUM_IDENTIFICACION
- **Campos principales**: TIPO_PERSONA, UBICACION_GEOGRAFICA

Tabla CUENTAS

- **Clave primaria**: NUMCUENTA
- **Campos principales**: TIPO_CUENTA, ESTADO_CUENTA, SALDO, FECHA_UTIMA_TRANSACCION, FECHA_CREACION, NUM_IDENTIFICACION

Tabla PRESTAMOS

- **Clave primaria**: ID
- **Campos principales**: TIPO_PRESTAMO, ESTADO_PRESTAMO, MONTO, INTERES, NUM_CUOTAS, DIA_PAGO, VALOR_CUOTA

Tablas de Operaciones

- OPERACIONESBANCARIAS, OPERACIONESCUENTA, OPERACIONESPRESTAMO
- **Campos principales**: MONTO, FECHA, HORA, UBICACION_GEOGRAFICA

Para hablar un poco de lo que hicimos en el código, vamos a hablar un poco de la arquitectura que usamo y en cada parte hablaremos de las generalidades:

a) Modelo: En nuestro proyecto, hemos definido clases de modelo para representar entidades como Usuarios, Cuentas Bancarias y Préstamos. Estas clases contienen atributos que representan los datos relevantes, como nombres, saldos y tipos de cuenta. Utilizamos anotaciones de JPA para mapear estas clases a tablas en una base de datos relacional. Implementamos métodos en estas clases para acceder y modificar los datos, como guardar un nuevo usuario o recuperar el saldo de una cuenta.

b) Vista: Para la interfaz de usuario de nuestra aplicación bancaria, creamos páginas HTML y archivos de plantillas Thymeleaf. Estas vistas muestran al usuario información como su saldo actual, historial de transacciones y opciones para realizar operaciones bancarias. Utilizamos HTML, CSS para diseñar interfaces intuitivas y receptivas que funcionen bien en todos los dispositivos.

c) Controlador: Nuestros controladores Spring Boot manejan las solicitudes HTTP entrantes y las dirigen al código adecuado para su procesamiento. Por ejemplo, tenemos un controlador usuario que maneja las solicitudes relacionadas con la gestión de usuarios, como el registro. Otro controlador, cuenta, se encarga de las operaciones relacionadas con las cuentas bancarias, como consultar saldos. Los métodos en estos controladores están anotados con @RequestMapping para asociarlos con rutas específicas de URL, como /user/register.º /account/transfer".

d) Capa de Servicio: Hemos introducido una capa de servicio entre nuestros controladores y modelos para la lógica de negocio más compleja. Por ejemplo, tenemos un UsuarioService que gestiona la lógica de registro de usuarios, autenticación y gestión de sesiones. Un CuentaService se encarga de operaciones bancarias como consignaciones, retiros y transferencias, aplicando reglas de negocio y verificaciones de seguridad.

e) Configuración y Estructura del Proyecto Nuestro proyecto Spring Boot está estructurado en paquetes separados para modelos, controladores, servicios y configuración. Seguimos las convenciones de configuración de Spring Boot, colocando archivos de configuración en ubicaciones predefinidas como src/main/resources. Utilizamos Maven o Gradle para gestionar dependencias y construir nuestro proyecto en un artefacto ejecutable, como un archivo JAR.

5. Utilizando pruebas unitarias, simulamos la inserción de tuplas en las tablas relevantes, como Oficinas y Usuarios, y verificamos la respuesta del sistema.

   ◇ Al intentar insertar una tupla 1 con una nueva clave primaria (PK), registramos el resultado de la operación.

   ◇ Posteriormente, tratamos de insertar una tupla 2 con la misma PK que la tupla 1 y observamos la respuesta del sistema.

   ◇ Los resultados indicaron que el sistema rechaza correctamente la inserción de la tupla 2, confirmando así la unicidad de las claves primarias y garantizando la integridad de los datos.

   Por otra parte para las Pruebas de Integridad Referencial con Claves Foráneas (FK)

   - Mediante pruebas de integración, evaluamos la relación entre las tablas que tienen llaves foráneas, como Cuentas y Clientes, para garantizar la coherencia de los datos.
   - Intentamos insertar una tupla en la tabla de Cuentas que hace referencia a un cliente existente en la tabla de Clientes y registramos el resultado.
   - Luego, intentamos insertar una tupla en la tabla de Cuentas que referencia a un cliente inexistente y evaluamos la respuesta del sistema.
   - Las pruebas confirmaron que el sistema acepta la inserción de la tupla que referencia a un cliente existente y rechaza la inserción de la tupla que hace referencia a un cliente inexistente, asegurando así la integridad referencial de los datos.

   Por ultimo para las Pruebas de Integridad de acuerdo con Restricciones de Chequeo:

   - Empleando casos de prueba específicos, verificamos el cumplimiento de las restricciones de chequeo de valores establecidas en las tablas.
   - Intentamos insertar tuplas que violan estas restricciones, como valores fuera de rango o formato incorrecto, y evaluamos el resultado de cada intento.
   - Los resultados de las pruebas indicaron que el sistema rechaza adecuadamente la inserción de tuplas que violan las restricciones de chequeo, mientras que acepta las inserciones que cumplen con las restricciones, asegurando la consistencia y validez de los datos almacenados.

   Estas pruebas las ejecutamos como parte de nuestro proceso de integración continua, lo que nos permitio detectar y corregir anomalías en el modelo de base de datos de manera proactiva.