ITERACIÓN 2

María Alejandra Pabón, Julian Eduardo Jaimes
Reporte de la iteración #2, sistemas transaccionales
Universidad de los Andes, Bogotá, Colombia
{ma.pabon, je.jaimes}@uniandes.edu.co
Fecha de presentación: Marzo 25 de 2020

Table of Contents

1. Introducción

En el presente informe se presentan los resultados del trabajo realizado para realizar la iteración 2 con todo lo que se exigía dentro del enunciado de la iteración. En esta se evidencian todo lo relacionado a los pasos seguidos, condiciones y resultados obtenidos para los puntos propuestos. A demás, se visualiza un manejo de sentencias SQL y manejo de base de datos haciendo uso de los lenguajes DDL y DML.

2. Modelo de negocio utilizado

Para la correcta implementación de la base de datos, se hizo uso del modelo de negocio utilizado para la primera iteración. Se hizo una ligera modificación: se agregó una clase PROVEEDOR la cual manejará los datos de los proveedores existentes en la plataforma y los

alojamientos que estos pueden manejar. En este orden de ideas, en nuevo modelo de clases se expone en la figura 1.

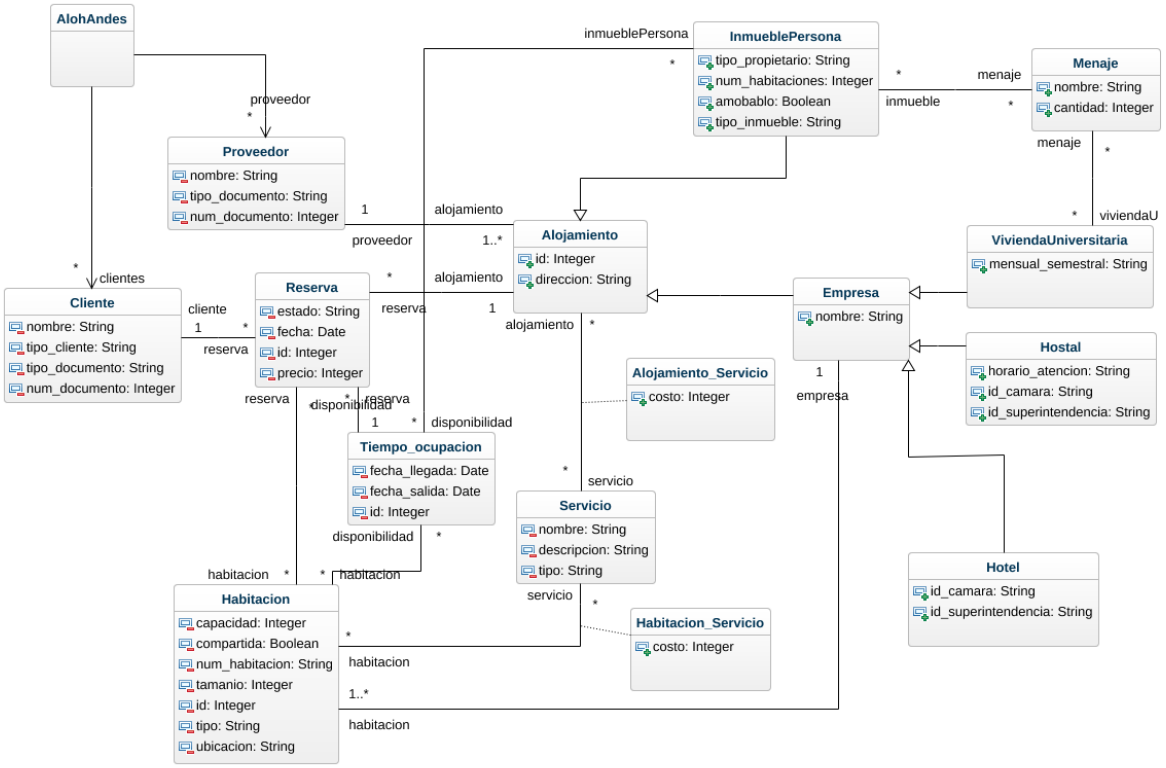

Figura 1. Modelo conceptual modificado

3. Diseño y construcción de la base de datos

A continuación, se muestra los pasos seguidos para poder llegar a la construcción de la base de datos pertinente y correcta.

3.1 Modelo el BCNF

Primero, se verificó que el modelo planteado estuviese en BCNF. Para lograr esto, en cada tabla se observaron las dependencias funcionales de los atributos que la conforman. Para lograr BCNF, tienen que estar en todas las formas normales anteriores a esta (primera, segunda y tercera forma normal) además de cumplir Boyce-Codd. A continuación, se muestran las dependencias funcionales de los atributos para cada tabla:

- Cliente (Nombre, Tipo_Persona, Tipo_Documento, Num_documento)

  Num_documento, Tipo_Documento -> Nombre

  Num_documento, Tipo_Documento -> Tipo_Persona

- Reserva (Estado, Fecha, Id, Cliente, Alojamiento, Id_Tiempo_Ocupacion,Precio)

  Id -> Estado, Fecha, Cliente, Alojamiento, Id_Tiempo_Ocupacion,Precio

- Tiempo_Ocupacion (Fecha_Llegada, Fecha_Salida, Id)

  Id -> Fecha_Llegada, Fecha_Salida

- Habitacion (ID, Empresa, Num_Habitacion, Tipo, Ubicacion, Capacidad, Compartida, Tamanio)

  Id -> Empresa, Num_Habitacion, Tipo, Ubicacion, Capacidad, Compartida, Tamanio,

- Servicio (Nombre, Descripcion, Tipo)

  Nombre -> Descripcion, Tipo

- Reserva_Habitacion (Id_Reserva, Id_Habitacion)

  Id_Reserva -> Id_habitacion

- Habitacion_Tiempo_Ocupada (Id_Habitacion, Id_Tiempo_Ocupacion)

  No hay dependencias funcionales.

- Habitacion_Servicio (Nombre_Servicio, Id_Habitacion, Costo)

  (Nombre_Servicio, Id_Habitacion) -> Costo

- Menaje (Nombre, Cantidad)

  No hay dependencias funcionales.

- Alojamiento (Id, Direccion)

  Id -> Direccion

- Inmueble_Persona (TipoPropietario, Num_Habitaciones, Amoblado, Tipo_Inmueble, Id_alojamiento)

  Id_alojamiento -> TipoPropietario, Num_Habitaciones, Amoblado, Tipo_Inmueble

- Alojamiento_Servicio(Nombre_Servicio, Id_Alojamiento, Costo)

  (Nombre_Servicio, Id_Alojamiento) -> Costo

- Empresa (Id, Nombre)

  Id -> Nombre

- Vivienda_Universitaria (Id)

  No hay dependencias funcionales.

- Hostal (Id, Horario_Atencion, Id_Superintendencia, Id_Camara)
  Id -> Id_Superintendencia, Id_Camara, Horario_Atencion

- Hotel (Id, Id_Superintendencia, Id_Camara)
  Id -> Id_Superintendencia, Id_Camara

- Menaje_ViviendaU (Id_ViviendaU, Nombre_Menaje)
  No hay depencencias funcionales.

- Menaje_Inmueble (Id_Inmueble, Nombre_Menaje)
  No hay depencencias funcionales.

- Inmueble_Tiempo_Ocupada (Id_Inmueble, Id_Tiempo_Ocupacion)
  No hay depencencias funcionales.

- Proveedor (Nombre, Tipo_Documento, Num_documento)

Num_documento, Tipo_Documento -> Nombre

3.2 Creación de tablas, con sus restricciones, en SQL Developer

En este apartado se utilizó la herramienta SQL Developer para la creación de las tablas utilizadas para la base de datos. En el archivo *CreacionTablas.sql* se encuentran las sentencias SQL respectivas para la correcta creación de las tablas; este .sql se divide en 3 secciones. La primera sección crea todas las tablas haciendo uso del comando CREATE TABLE (nombre de la tabla) seguido por los atributos de esta y el tipo de dato (se manejó Number, Varchar2 y Date). Para cada tabla, en su respectiva creación, se asignó la PK como constraint, siguiendo así el modelo propuesto en Parranderos-jdo. La segunda sección fue la asignación de restricciones para cada tabla haciendo uso de ALTER TABLE (nombre de la tabla) seguido por las restricciones a colocar. Se creo una sentencia con ALTER TABLE para cada restricción. Se manejaron restricciones de no nulidad, no duplicado, llaves foráneas y restricciones de chequeo. En la tercera sección se establecieron los triggers; los cuales fueron necesarios para condiciones de chequeo donde se verificaba que la fecha ingresada fuese mayor a la fecha actual.

4. Población de las tablas

En este apartado se utilizó la herramienta SQL Developer para poblar las tablas necesarias que satisficieran los RF1, RF2 y RF3. Estos tres requerimientos funcionales implican agregar toda la información relacionada a los operadores, las propuestas de alojamiento y las personas habilitadas hará hacer uso de los alojamientos. Para el esquema trabajado, esto sería poblar las tablas: PROVEEDOR, ALOJAMIENTO, CLIENTE, EMPRESA, HABITACION, VIVIENDA_UNIVERSITARIA, HOTEL, HOSTAL, INMUEBLE_PERSONA, MENAJE,

MENAJE_INMUEBLE, MENAJE VIVIENDA_U, SERVICIO, ALOJAMIENTO_SERVICIO, HABITACION_SERVICIO. Se usan estas tablas ya que son las necesarias para mostrar la información completa de las propuestas de alojamiento, los clientes y los proveedores. Para un mayor rendimiento, en lugar de hacer INSERT a cada una de las tuplas, se utilizó la sentencia: INSERT ALL INTO (nombre de la tabla) (atributos) VALUES (valores a insertar) SELECT * FROM DUAL; esta última sentencia permite ingresar de manera más eficiente varias tuplas en una sola sentencia. Sin embargo, para las tablas donde el ID (pk) es asignado por el sistema, no se puede utilizar esta sentencia ya que genera error de violación de primary key. El archivo en el cual se presenta la población de los datos se llama *PoblarTablas.sql.*

5. Implementación de RF4, RF5, RF6, RFC1, RFC2

Para Esta parte de la iteración, se trabajó desde Java Eclipse, siguiendo el formato propuesto en Parranderos-jdo.

Se comenzó trabajando en el paquete src/main/java. En este se comenzaron a implementar las clases, donde cada clase representa una tabla de la base de datos. Cada una de estas clases implementa una interfaz VO la cual es utilizada para una correcta protección de los datos. A cada clase se le añadieron los atributos y métodos necesarios para representar correctamente su respectiva tabla de la base de datos. Todo esto fue realizado dentro del paquete de negocio.

Seguido a esto, se comenzó a trabajar sobre el paquete de persistencia. Se crearon diferentes clases SQL las cuales son las encargadas de modificar los datos de la base de datos. Estas fueron creadas una para cada tabla de la base de datos, así hay un mayor orden y protección de datos a la hora de la modificación. En estas clases se implementaron sentencias SQL INSERT y DELETE, para agregar o eliminar una tupla en una tabla respectivamente. En la clase PersistenciaAlohandes se implementaron el llamado a los métodos de cada clase SQL, así una sola clase es capaz de manejar los métodos con sentencias SQL para la base de datos.

La clase principal que maneja todas las conexiones y la relación entre los métodos que modifican o consulta la base de datos con las clases que representan las tablas se llama ALOHANDES. A demás esta clase será llamada para los test implementados y para el manejo desde la consola.

RF4. El requerimiento funcional 4 hace referencia a la creación SIMPLE de una reserva. Este fue implementado en la clase SQLReserva. Haciendo uso de la sentencia INSERT INTO RESERVA + (nombre de los atributos de Reserva) VALUES + (los valores que fueron ingresados por parámetro para cumplir la correcta creación), este método retorna un long con la cantidad de tuplas insertadas, el cual debería ser siempre 1. Este método es llamado en PersistenciaAlohandes, cuando se empieza a ejecutar, crea en la tabla Tiempo_ocupada un nuevo tiempo que representa una fecha de llegada y otra de salida junto con un id, el cual se utilizará para crear la reserva. Se convierten los tiempos en el formato Timestamp necesario y se crea un objeto Reserva que representa a la reserva una vez fue añadida a la base de datos. En caso de que haya un error con algún dato mal ingresado o que se viole algún constraint del negocio, no se crea ninguna reserva

RF5. Para eliminar una reserva, se utiliza el id de esta, ya que es el PK y es único lo que evitaría que se elimine una reserva diferente o que se elimine más de una. Este método, que elimina directamente de la base de datos fue implementado en la clase SQLReserva y retorna el número de tuplas eliminadas. Luego cuando este método es llamado, se rectifica la correcta

eliminación si el valor retornado es diferente de 0 es que se eliminó correctamente, de lo contrario hubo algún error al ingresar algún dato o en las restricciones de chequeo.

RF6. Para eliminar una oferta de alojamiento, se utiliza el id de esta, ya que es el PK y es único lo que evitaría que se elimine un alojamiento diferente o que se elimine más de uno. Este método, que elimina directamente de la base de datos fue implementado en la clase SQLAlojamiento y retorna el número de tuplas eliminadas. Luego cuando este método es llamado, se rectifica la correcta eliminación si el valor retornado es diferente de 0 es que se eliminó correctamente, de lo contrario hubo algún error al ingresar algún dato o en las restricciones de chequeo. Además de esto, en la clase PersistenciaAlohandes, antes de llamar al método mencionado anteriormente, se verifica que no existan reservas asociadas a el alojamiento que se quiere eliminar. Si existen reservas asociadas al alojamiento, se retorna –1, para indicar que no se pudo eliminar el alojamiento. En caso contrario, se procede a llamar el método eliminar alojamiento de la clase SQLAlojamiento.

RFC1. Para consultar el dinero recibido por cada proveedor durante el año actual, se toma la información de la tabla reserva y se une con la de la tabla alojamiento para y se filtra para que solo se tomen en cuenta las reservas registradas en el año actual. Luego se agrupan los resultados por el proveedor de alojamiento y se suman el costo de la reserva para encontrar la ganancia de cada proveedor.

RFC2. Para consultar las 20 ofertas de alojamiento más populares se consulta la tabla reserva y se cuenta el número de tuplas agrupadas por alojamiento, para saber cuántas reservas se hacen a cada alojamiento, después se junta esta información con la de la tabla alojamiento, para mostrar la información del alojamiento, posteriormente se ordena por la cantidad de reservas y finalmente se limita el número de tuplas mostrado a 20.

6. Implementación de pruebas

En este apartado se crearon los casos de prueba especificados en el enunciado de la iteración. Este apartado se divide en dos partes: una primera parte en java y una segunda parte en sql.

6.1 Implementación de pruebas en java

En este apartado se crearon las pruebas implementadas en java y se tienen 2 clases. La clase ConexionTest que se encarga de probar el acceso a la base de datos. Y la clase TablasTest que implementa las pruebas a los requerimientos funcionales 4,5 y 6 y los requerimientos de consulta 1 y 2.

En la clase TablasTest se realizan las pruebas bajo el supuesto de que las tablas ya han sido creadas y pobladas previamente utilizando los archivos de creación y población de tablas mencionados en los numerales 3.2 y 4 de este documento y no se han realizado más cambios.

Dentro de la clase TablasTest podemos encontrar 4 métodos que son los test que prueban todos los requerimientos. El método AgregarYEliminarReserva prueba el correcto funcionamiento del requerimiento RF4 al verificar la creación de una reserva y del requerimiento RF5 al eliminar una reserva correctamente, para ambos requerimientos se verifica el número de reservas antes y después de ejecutarlos y se verifica que efectivamente se haya creado o eliminado en cada caso. El método EliminarAlojamiento prueba el funcionamiento correcto del requerimiento RF6 al verificar el número de alojamientos antes y después de eliminar un alojamiento. El método verificarRFC1 verifica el funcionamiento requerimiento RFC1 al verificar el número de proveedores que aparecen y su ganancia después de agregar una reserva. En el método verificarRFC2 se verifica el funcionamiento

requerimiento RFC2, al verificar el número de alojamientos que aparecen en la respuesta y el orden en que aparecen, después de añadir una cantidad de reservas.

6.2 Implementación de pruebas en sql

En este apartado se crearon los escenarios de prueba para la corrección y calidad del modelo en la base de datos. Para esto, se crearon 3 archivos sql llamados pruebasUnicidad.sql, pruebasFKInsercion.sql y pruebasRestriccionesChequeo.sql. Cuando se vaya a ejecutar este archivo las tablas deben estar vacías para que no haya conflictos, para asegurar que esto puede ejecutar el archivo BorrarTuplas.sql.

En el primer archivo se realizan las pruebas de unicidad para cada tabla de acuerdo con lo especificado en el lineal de los escenarios de prueba del enunciado. El resultado de ejecutar este archivo como script es el nombre de la tabla a la que se está haciendo la prueba, Después de esto, una tupla insertada correctamente. Y finalmente, un error producido por tratar de insertar una tupla con la misma llave primaria.

En el segundo archivo se realizan las pruebas de integridad de FK al realizar inserciones. El resultado de ejecutar este archivo como script es primero la inserción de algunas tuplas necesarias para probar la inserción correcta de las tuplas. Posteriormente, el nombre de la tabla a la que se está haciendo la prueba, Después de esto, una tupla insertada correctamente. Y finalmente, uno o varios errores debido a que se trató de insertar una tupla que tiene una referencia a una tupla que no existe en la tabla a la que se referencia. Dependiendo del número de atributos que tengan constraint de foreign key en la tabla se mandan el correspondiente número de errores.

En el tercer archivo se realizan las pruebas de integridad de restricciones de chequeo al realizar inserciones. El resultado de ejecutar este archivo como script es primero la inserción de algunas tuplas necesarias para probar la inserción correcta de las tuplas. Posteriormente, el nombre de la tabla a la que se está haciendo la prueba, Después de esto, una tupla insertada correctamente. Y finalmente, uno o varios errores debido a que se trató de insertar una tupla que tiene una referencia a una tupla que no cumplía con las restricciones para probar que no se debería poder insertar ninguna tupla que incumpla con alguna de las restricciones de chequeo establecidas en el esquema.