**Iteración 1**

Juan Sebastián Espitia Acero

Daniel Perilla

Índice:

* Contexto general (descripción del caso de uso)
* Modelado del proyecto
  + Diagrama de clases
  + Modelo relacional de la base de datos
  + Comparación del modelo propuesto con el modelo generado automáticamente por *Enterprise Architect*
* Resultados del trabajo y la implementación
  + Aspectos logrados
  + Aspectos pendientes y observaciones
  + Pruebas

Contexto general del caso:

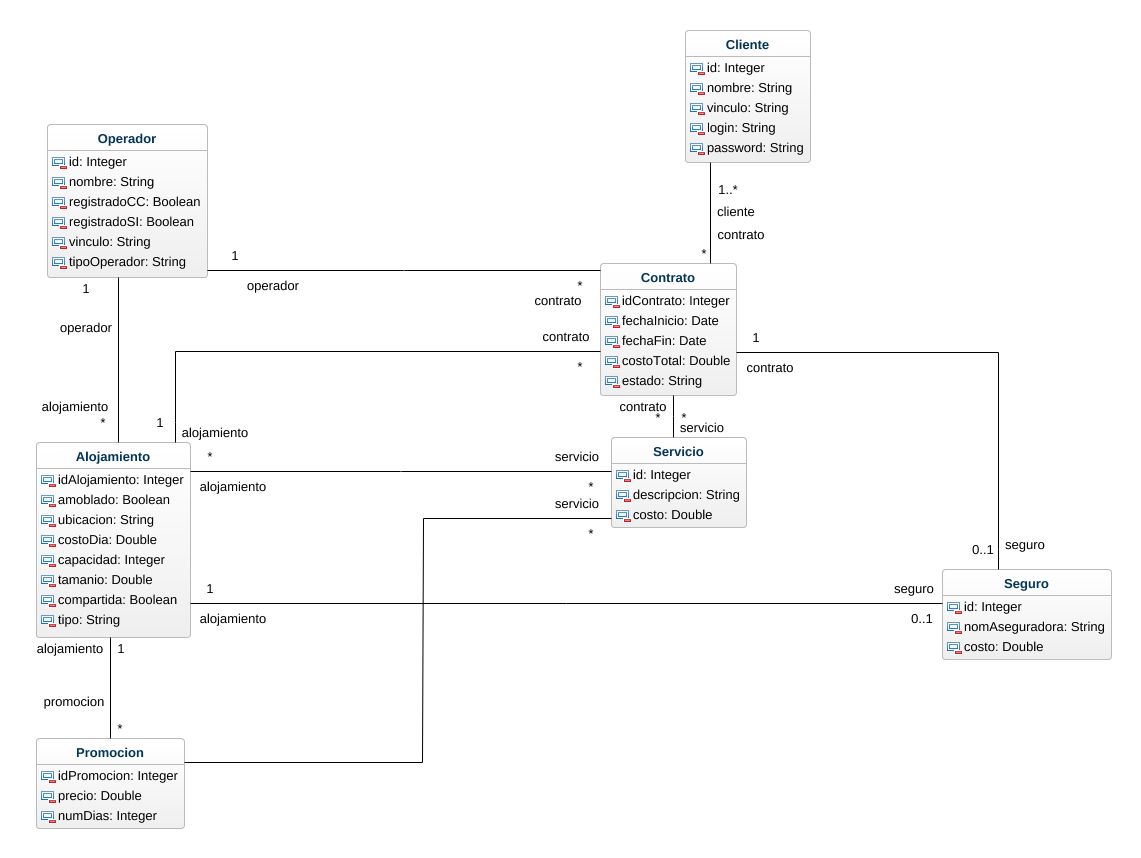
Se propone entregar “AlohAndes” como una solución que permite acordar contratos y reservas entre los clientes, que son miembros de la comunidad uniandina, y los operadores aprobados en el sistema, que son entidades que ofrecen servicios de alojamiento y vivienda temporal. Los dos usuarios anteriormente mencionados son los principales actores que interactúan con el programa. Un operador puede alterar su oferta de vivienda en cualquier momento siempre y cuando no haya clientes afectados, y también es informado de las solicitudes de alojamiento que crean los clientes. Un cliente puede elegir la vivienda que desea, el tiempo en que la ocupará, y los servicios que piensa adquirir, que varían según el tipo de vivienda ofertada.

La interacción de los usuarios y el sistema se detalla en el documento adjunto en este mismo directorio, con el nombre “Casos de uso iteracion 1.pdf”

Modelado del proyecto:

* Descripción del modelo de clases UML

En la siguiente figura se muestra el diseño del programa que se implementó, en un diagrama de clases UML. En principio se pensó realizar un diagrama complejo que tuviera en cuenta todos los posibles detalles que naturalmente se pueden relacionar a las entidades involucradas en el mundo real (por ejemplo, incluir el número de cuenta bancaria del cliente, o su documento de identidad), pero se intentó simplificar el modelo para que correspondiera con el caso de estudio sin mucha extensión. Así mismo, la herencia de clases que se podía tener en cuenta se modeló con los atributos “tipo” y “vínculo” en varias de las clases, los cuales solo hay que confirmar que pertenezcan a un conjunto específico de valores.



**Figura 1. Diagrama de clases UML generado con *genmymodel***

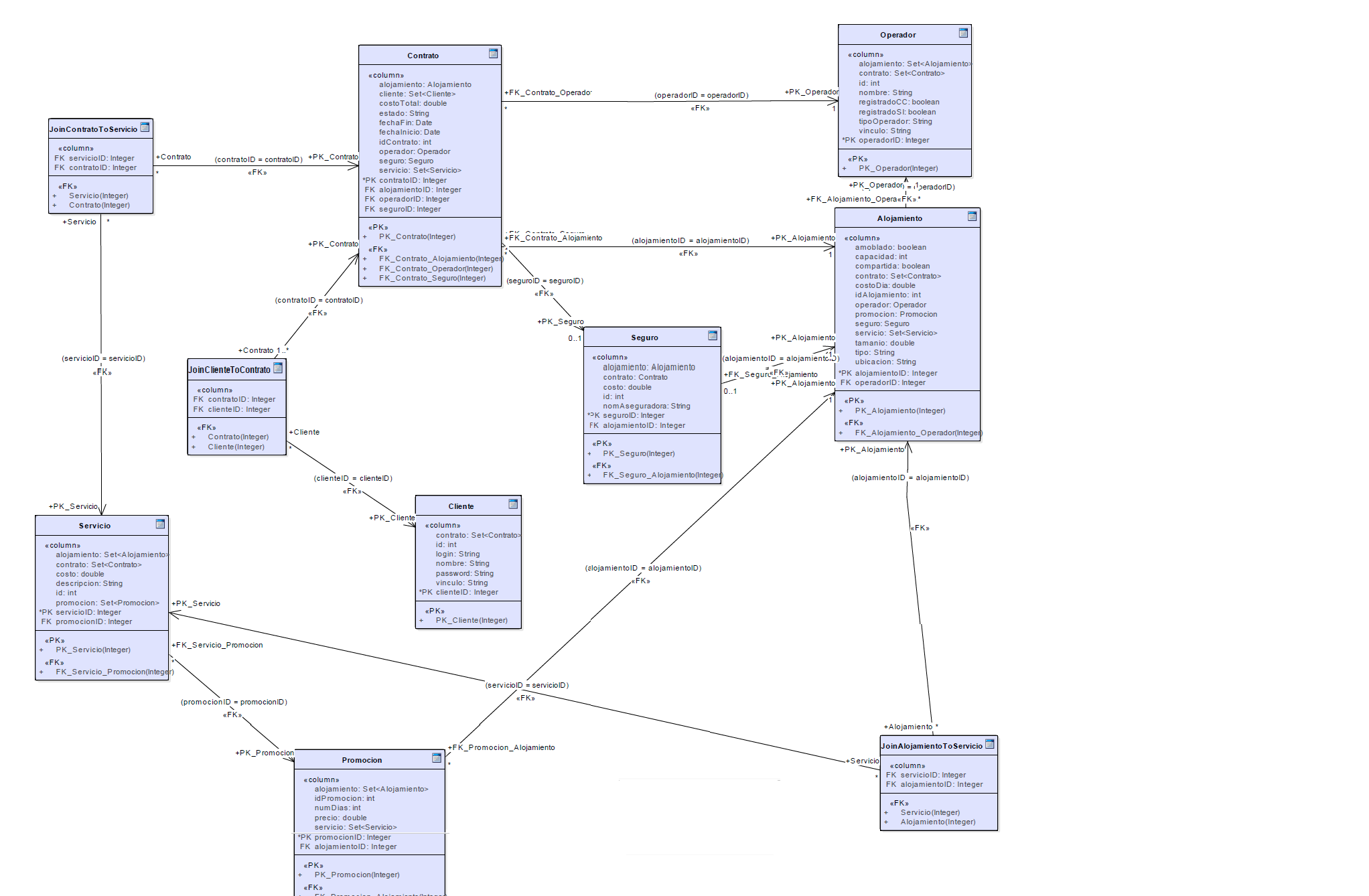
En la figura 1 se puede ver que el concepto de una vivienda ofertada se unificó como un alojamiento (que puede ser habitación de hotel con sus categorías, de hostal, de vivienda, una casa o un apartamento), que tiene de forma inherente unos servicios asociados, que son los que se ofrecen desde el momento en que el operador registra la nueva opción de alojamiento.

En cuanto a los acuerdos de renta de un alojamiento, se condensaron en la clase “Contrato”, la cual incluye la información necesaria del contrato: entre quiénes se firma, cuál es el bien que se contrata, la vigencia del mismo, el costo y un estado, que indica si está activo, en reserva, se canceló o ya terminó.

Los diferentes tipos de alojamiento se condensaron en la clase “Alojamiento”, la cual tiene en cuenta los posibles atributos particulares de estos tipos de alojamiento, un indicador que lo ubica en una categoría (tipo), y el costo simplificado a un costo diario.

- Modelo relacional y comparación

Como se utilizó una herramienta diferente a *Enterprise*, fue necesario corregir errores para poder exportar el modelo a un formato XMI, que luego fue posible importar en *Enterprise* y desde ahí transformar por un algoritmo del programa*.* El siguiente es el modelo que se generó.



**Figura 2: El modelo relacional generado de manera automática por *Enterprise Architect***

La documentación del modelo relacional que desarrollamos a mano está en esta misma carpeta en el archivo “FormatoDocEsqRelacional.xlsx”. En dicho archivo documentamos las tablas que creamos en la base de datos, con sus respectivas restricciones.

Las diferencias más notables son, que el modelo generado de manera automática establece las relaciones como atributos de objetos dentro de la clase padre, mientras que en el diseño consideramos que bastaba con que el padre conociera los identificadores de los objetos a los que se relaciona y no a la totalidad de los objetos. Esto simplifica el modelado de las relaciones puesto que no se corre el riesgo de entrar en un ciclo en el que se crea en la clase padre, a un hijo que a su vez necesitaría conocer al objeto concreto de su padre/dueño. Otras diferencias menos significativas se encuentran en el nombramiento de las relaciones y los atributos llave.

Resultados del trabajo y la implementación

* Aspectos logrados
* Aspectos pendientes y observaciones
* Pruebas

En cuanto a los escenarios de prueba, se omitió la prueba del numeral 1, puesto que las tablas de la base de datos fueron configuradas para que la llave de identidad se generara automáticamente de 1 en 1, y por este motivo no se pueden hacer inserciones con algún valor para esta columna.

Para el numeral 2 de las pruebas, vimos que funcionaba perfectamente una inserción con llaves foráneas existentes en sus correspondientes tablas, pero cuando no existen dichas llaves, no permite realizar la inserción. Se arrojan Excepciones al intentar ejecutar esas sentencias. En cuanto al borrado de las tuplas con columnas que funcionan como llave foránea en alguna otra tabla, se vio que hay varios métodos de solución, entre los cuales el más adecuado con las necesidades del programa, es el de cascada.

Para el numeral 3 de las pruebas, vimos que las restricciones de chequeo se revisan siempre que se intenta insertar una tupla, y que, al intentar insertar una tupla que viole estas restricciones, se rechaza la transacción.