Documento de informe

Joseph Steven Ortiz, Mateo Saravia Salamanca
Iteración 3
Universidad de los Andes, Bogotá, Colombia
{js.ortiz, m.saravia}@uniandes.edu.co
Fecha de presentación: Mayo 20 de 2018

Tabla de contenido

1 Modelos solicitados

1.1) Modelo conceptual

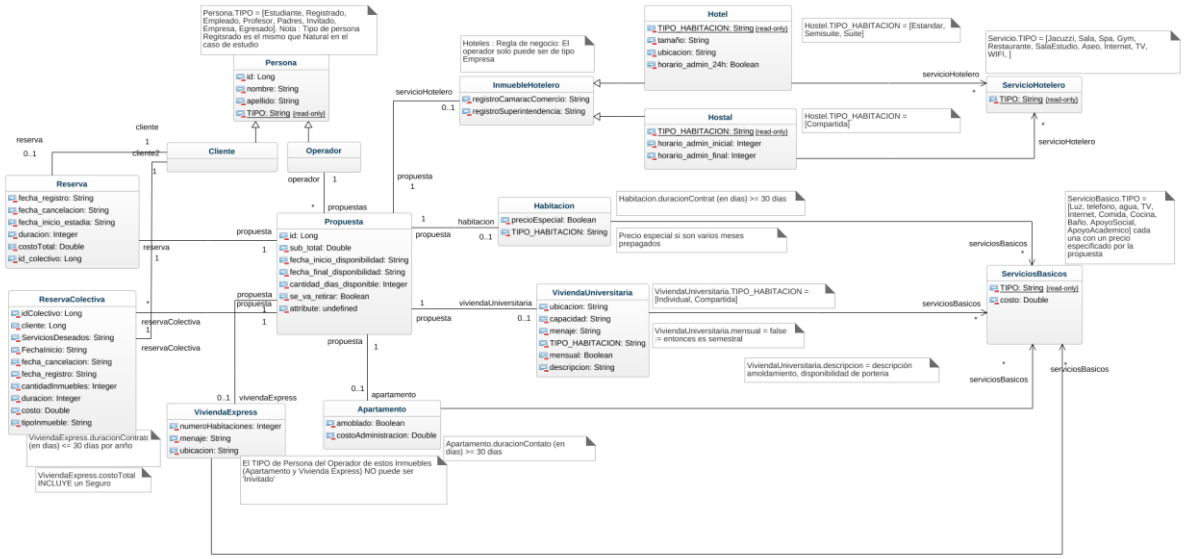

Figura 1. Modelo Conceptual de AlohAndes

Con respecto a las iteraciones anteriores, al modelo conceptual se le añadió una clase nueva llamada `ReservaColectiva`, la cual nos permite realizar de manera más simple el registro de múltiples reservas, registradas por un cliente, dado un inmueble determinado. Puesto que si se implementa con una lista de reservas puede haber complicaciones a la hora de agregar un nuevo objeto a la base de datos, debido a que objetos con atributos similares se repiten gran cantidad de veces en los datos, lo cual haría más lento el proceso de lectura en ejecución. En la clase mencionada, se registran los atributos determinados por el cliente en el inicio de la reserva, el tipo de inmueble deseado, la duración del contrato de hospedaje, la cantidad de personas a alojar, entre otros.

1.2) Modelo relacional

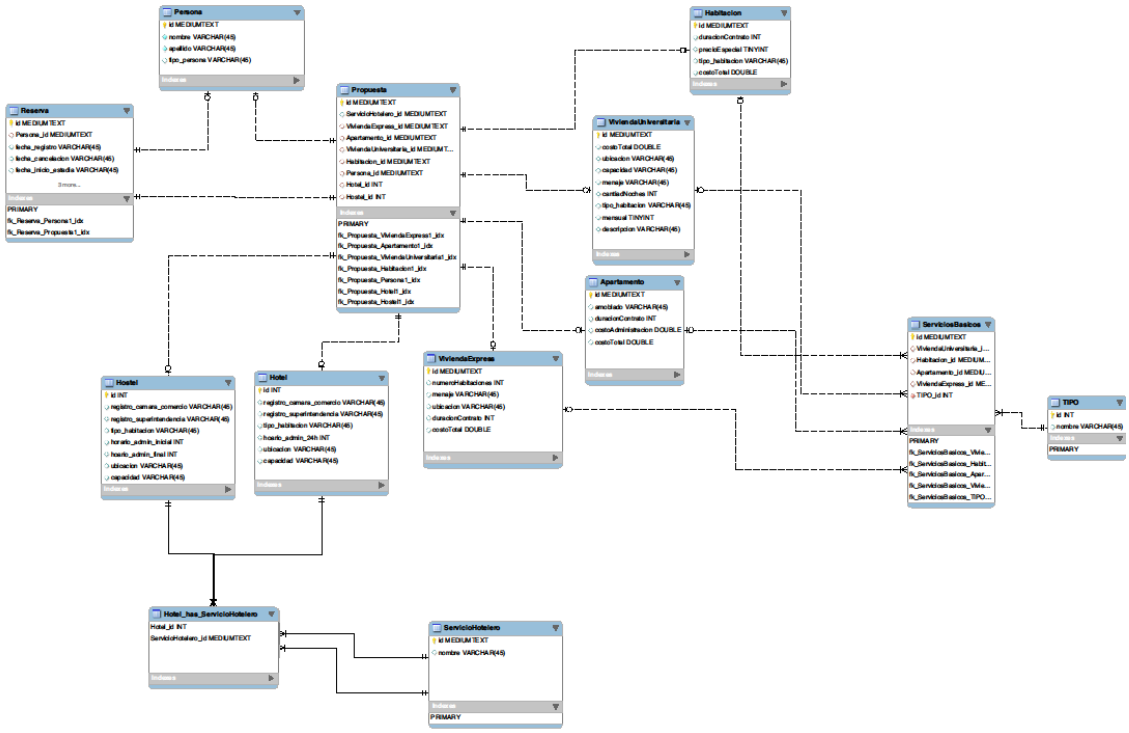

Figura 2. Modelo Relacional de AlohAndes

2 Resultados del Proyecto

Después de haber desarrollado el tercer proyecto se obtuvieron diferentes de resultados:

2.1) Resultados Logrados:

Se desarrollaron 4 nuevos requerimientos funcionales de consulta, a partir de los cuales se obtuvieron los resultados esperados. Sin embargo, algunos de ellos no se desarrollaron en consultas simples. Por tanto, al ser implementados en distintos `querys`, se tuvo que implementar más de un método para el requerimiento en el API REST. Los resultados obtenidos para cada uno de los requerimientos fueron los siguientes:

2.1.1) RFC10: Consultar consumo en AlohAndes

Para este requerimiento se evaluó que tipo de usuario tiene acceso a él, que son los clientes o los administradores. En la consulta, se preguntaba por los usuarios que realizaran una consulta en al menos en determinada oferta de alojamiento. Para ello, se hacia la selección de un conjunto en la base de datos y se preguntaba por las personas cuyo papel se “cliente”, la fecha de un registro (de la tabla registro) que se encuentre en determinada fecha por parámetro. Adicionalmente, el requerimiento decía que se debía poder filtrar la búsqueda por el tipo de inmueble, el id del operador o el tipo de persona que ponía el alojamiento. Esta consulta funciona, pero se hizo dos métodos, uno sin los filtros y otro con los filtros.

2.1.2) RFC11: Consultar consumo en AlohAndes (RFC10-v2)

Este requerimiento es igual al anterior, pero se pregunta por los usuarios que no hayan hecho una reserva en determinado tiempo pasado por parámetro, también debe permitir los filtros anteriores. En esta consulta, se hace una selección primero de un conjunto de las reservas que estén en dentro el rango dado de fechas. A este conjunto seleccionado se pregunta por los usuarios que no estén dentro de esas reservas. Y luego se permite la opción de los filtros. Este método funciona del mismo modo del anterior, se hizo un método que retorna los usuarios sin filtro y otro con filtros.

2.1.3) RFC12: Consultar funcionamiento

En este requerimiento, se pregunta para cada semana del año por la oferta de alojamiento de mayor y menor ocupación y el operador de mayor y menor demanda. En esta consulta, se toman las reservas de mayor y menor duración según la fecha de inicio, y en otra consulta se toman las semanas del año. Después se compara entre las dos consultas que esté dentro de la semana y que se sea la de mayor y menor duración. Para los operadores se hizo el mismo procedimiento, mas no se preguntaba por la duración de las reservas, sino que se contaban las reservas que se hayan hecho sobre una propuesta y se tomaba el mayor dentro de una semana determinada del año. El requerimiento funciona, pero se hizo en 4 querys diferentes, dos para las propuestas y dos para los operadores. Además, no funcionaba si se escogían todas las columnas de una tabla, así que se tuvo que llamar otro método que retornara el objeto según el id del objeto, haciendo que tome un poco más de tiempo al producir la consulta.

2.1.4) RFC13: Consultar los buenos clientes

En este requerimiento, primero se preguntaba por los clientes que hagan reservas al menos una vez al mes, los que hagan reservas en alojamientos costosos (más de 150USD por noche) y los que solo reservan suites. En esta consulta solo fue necesario un query, donde se tomaban los objetos de la tabla personas unida con propuesta y reserva. Se preguntaba si reservaba hotel y en el hotel reservaba una “suite” (porque las suites son de los hoteles), o si el costo de la reserva dividido al duración era mayor a 150USD, o si la fecha del sistema menos la fecha de reserva era menor a una duración dad por parámetro, para saber si hizo al menos una reserva al mes.

2.2) Resultados no logrados

Con la carga masiva, al realizar los requerimientos no se obtenían ciertos datos que si cumplían las funciones, además de alentar el tiempo de ejecución.

3 Supuestos adicionales de las reglas de negocio

Cuando se pregunta por los buenos clientes se consideran buenos clientes si hacen reservas de al menos 150USD la noche, pero se podría redondear desde 145USD la noche, puesto que esta información puede beneficiar a los clientes para poder brindarles ofertas con respecto a las propuesta de alojamiento. También se le hace descuento a los clientes que hacen reservas en promedio una vez al mes, y no necesariamente los que hagan una reserva cada mes estrictamente.

4 Balance de plan de pruebas

Para el balance de plan de pruebas, en `SQLDeveloper` se seleccionaban todas las personas o propuestas (según lo que especificaba el requerimiento), y se miraban los datos que servían para verificar. Después, se desarrollaba la sentencia que resolvía el requerimiento y se comparaban los datos con los tomados en la primera selección. Después de que funcione adecuadamente, se insertaban datos que cumplieran y que no cumplieran

con las condiciones estipuladas, al comparar que tome los datos que cumplen las condiciones se procedía a pasar el código en java. En este ya se verificaba que retorne el objeto solicitado y se comparaba con el `SQLDeveloper` para que sean los datos adecuados.

5 Indices Utilizados

	OWNER	INDEX_NAME	INDEX_TYPE	TABLE_OWNER	TABLE_NAME	TABLE_TYPE	UNIQUENESS	COMPRESSION	PREFIX_LENGTH
1	ISIS2304A901810	PROPUESTA_PK	NORMAL	ISIS2304A901810	PROPUESTA	TABLE	UNIQUE	DISABLED	(null
2	ISIS2304A901810	I_TIPO_INMUEBLE	NORMAL	ISIS2304A901810	PROPUESTA	TABLE	NONUNIQUE	DISABLED	(null

Figura 3. Index tabla Propuesta

	OWNER	INDEX_NAME	INDEX_TYPE	TABLE_OWNER	TABLE_NAME	TABLE_TYPE	UNIQUENESS	COMPRESSION	PREFIX_LENGTH
1	ISIS2304A901810	RESERVA_PK	NORMAL	ISIS2304A901810	RESERVA	TABLE	UNIQUE	DISABLED	(null) TB
2	ISIS2304A901810	I_FECHA_REG	NORMAL	ISIS2304A901810	RESERVA	TABLE	NONUNIQUE	DISABLED	(null) TB
3	ISIS2304A901810	I_COSTO	NORMAL	ISIS2304A901810	RESERVA	TABLE	NONUNIQUE	DISABLED	(null) TB
4	ISIS2304A901810	I_DURACION	NORMAL	ISIS2304A901810	RESERVA	TABLE	NONUNIQUE	DISABLED	(null) TB
5	ISIS2304A901810	I_FECHA_IN	NORMAL	ISIS2304A901810	RESERVA	TABLE	NONUNIQUE	DISABLED	(null) TB

Figura 4. Index tabla Reserva

Para la óptima ejecución de las consultas requeridas, se añadieron los índices presentados anteriormente, puesto que estos mejoraban notoriamente el rendimiento del tiempo de búsqueda. Debido a que el número de datos manejados por el programa es bastante alto, las recursiones sobre las listas de datos obtenidas, tomaba tiempo significativo. En el caso del índice `I_FECHA_REG`, se hizo para los requerimientos RFC10 y RFC11 los cuales solicitaban búsqueda y selección de datos, noción que se puede simplificar con la implementación de un `Clustered Index` y su ordenamiento ascendente; debido a que éste permite una búsqueda simple entre rangos. El índice `I_FECHA_INICIO`, se usó para los requerimientos `RFC12 y RFC13`, los cuales, de igual formaba requerían búsquedas entre rangos de fecha. Y por último, los índices `I_COSTO` e `I_DURACION` se usaron únicamente en el último requerimiento. Para estos índices se implementaron como “no únicos” debido a la selectividad de los elementos en la base de datos, ya que las fechas están en rangos no definidos y el costo y la duración no van a ser únicos dentro de los datos seleccionados.