Iteración 4 del Proyecto del Curso Sistemas Transaccionales - Caso ALOHANDES

Alonso Hernández Tavera, Cristian Bernardo Acuña Silva
Documento de la Iteración 4 – Caso ALOHANDES
Universidad de los Andes, Bogotá, Colombia
{f.hernandezt, cb.acuna}@uniandes.edu.co
Fecha de presentación: mayo 28 de 2023

Tabla de contenido

1 Introducción y Contextualización

ALOHANDES corresponde a un servicio integrado de alojamiento que, por medio de un sistema digital, busca implementar la idea del proyecto ALOHA con el objetivo de facilitar opciones de alojamiento a la comunidad de la Universidad de los Andes. Para lograr generar una oferta amplia de opciones de alojamiento, la universidad ha establecido convenios con 10 hoteles del sector, 12 hostales cercanos, un operador de residencias universitarias vecino, ha convocado a la comunidad de Fenicia (que es vecina a la universidad) y también a la comunidad universitaria (empleados, profesores y estudiantes) para que puedan ofrecer los inmuebles que tienen bajo su propiedad en cualquiera de las modalidades que ha definido ALOHA (ALOHANDES – Descripción general del caso, 2023).

La estructura base que sigue el sistema ALOHANDES es el siguiente: Un operador, que es toda persona natural o empresa de alojamientos con la que la universidad ha establecido convenios, puede ofertar los inmuebles que tenga bajo su propiedad (y se encuentren en una zona vecina a las instalaciones centrales de la universidad) a partir de su descripción y caracterización. Algunas de las características que debe incluir son: La ubicación de ese inmueble, el area del inmueble, su capacidad y a qué tipo corresponde (vivienda o habitación).

Además, también debe establecer la modalidad temporal en la que quiere que su inmueble sea reservado y ocupado (las opciones son: reserva por días, semanas, meses o semestres).

Al definir esos detalles, el operador puede inscribir su inmueble bajo una oferta de alojamiento, la cual se registra en el sistema de ALOHANDES junto a los servicios que ese operador defina pueden ser incluidos con un precio adicional o no. Un cliente registrado en el sistema (y que pertenece a la comunidad Uniandes) puede visualizar las diferentes ofertas realizadas por los operadores por medio del sistema y elegir la que mejor se ajuste a sus requerimientos y posibilidades. Una vez un cliente toma la decisión de aceptar una oferta de alojamiento, se crea un registro de la reserva. Esta reserva tiene en cuenta cualquier promoción a la que el cliente haya accedido, la fecha en la que se realizó la reserva, las fechas inicial y final de la reserva y, en caso de que se haya cancelado, también incluye la fecha de cancelación de esta.

El objetivo de esta primera parte del proyecto es analizar y diseñar el sistema ALOHANDES junto a una base de datos de tipo relacional que permita su funcionamiento correcto, teniendo en cuenta un conjunto de requerimientos funcionales y no funcionales definidos.

2 Análisis y modelo conceptual

A continuación, se puede visualizar el modelo conceptual construido con el objetivo entender el mundo del caso que se está tratando. Adjunto en el lugar de entrega de este documento, se encuentra también el mismo modelo en formato imagen para ser visualizado de mejor forma.

2.1 Modelo conceptual construido

A continuación, se puede visualizar el modelo conceptual construido con el objetivo entender el mundo del caso que se está tratando. Adjunto en el lugar de entrega de este documento, se encuentra también el mismo modelo en formato imagen para ser visualizado de mejor forma.

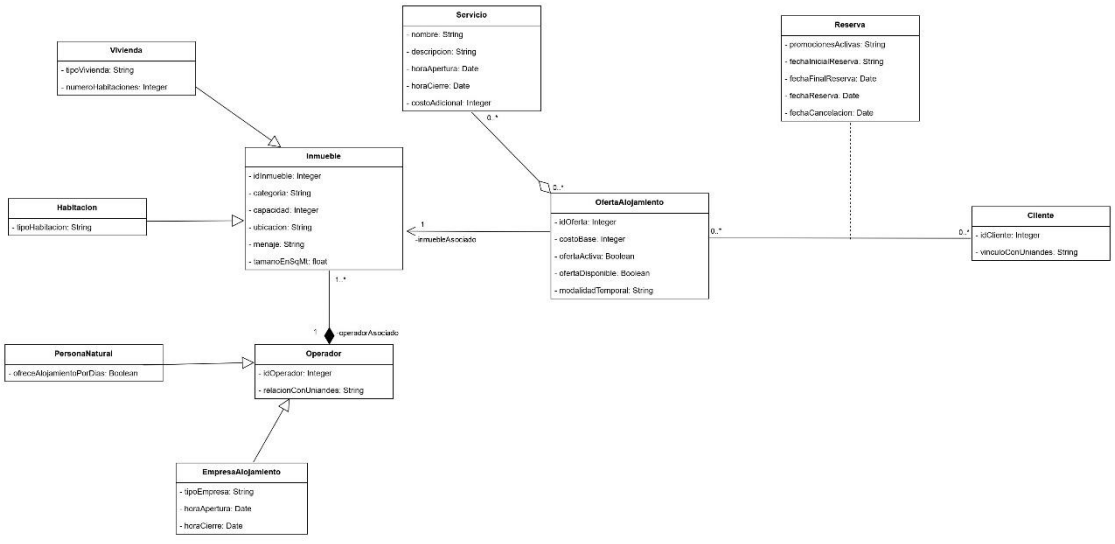

Figura 1. Diagrama conceptual construido para el caso ALOHANDES

2.2 Consideraciones y aclaraciones del modelo conceptual

Las consideraciones que se deben tener en cuenta para entender el diagrama conceptual son las siguientes:

1. Ofrecer alojamiento con días para una persona natural se relaciona con ofrecer alojamiento esporádico.
2. Se asume que todos los precios que se van a manejar van a ser de tipo entero positivo.
3. El hecho de que un inmueble se encuentre o no amoblado debe ser representado como un servicio incluido.
4. En ninguna clase se modela el atributo precio o costo total, pues este corresponde a un dato de tipo derivado (depende de otros atributos y variables para ser calculado). Los datos derivados impiden obtener la tercera forma normal (3FN) en el modelo (Huaraca, 2020) y, por tanto, no es deseable modelarlos en las relaciones de forma inicial. En su lugar, se incluye el atributo "costoBase" en la clase "OfertaAlojamiento", el cual no es un dato derivado.
5. El atributo de "modalidadTemporal" en la clase OfertaAlojamiento se refiere a si el inmueble es ofrecido por días, semanas, meses o semestres.
6. Se supone que un cliente solo puede realizar una reserva por día, pero puede tener más de una reserva activa al mismo tiempo. Cada reserva la puede hacer máximo un cliente.
7. Las clases "Vivienda" y "Habitacion" se toman como clases que heredan de la superclase "Inmueble" debido a que esta última es hasta cierto punto una abstracción de ambas y contiene atributos que comparten. De la misma forma, "PersonaNatural" y "EmpresaAlojamiento" heredan de la superclase "Operador".
8. La clase "Reserva" corresponde a una clase de asociación, la cual modela los diferentes atributos que caracterizan la relación de un cliente con una oferta de alojamiento.
9. La diferencia entre "ofertaActiva" y "ofertaDisponible" en la clase "OfertaAlojamiento" es la siguiente: Una oferta permanece activa mientras el operador que la publicó no haya decidido retirarla, en cambio, permance disponible mientras un cliente no la haya reservado.
10. En la clase "Reserva" se modela un atributo llamado "fechaCancelación" con el objetivo de saber si una reserva fue o no cancelada y aplicar la respectiva penalización en caso de que aplique.

3 Diseño de la base de datos

3.1 Modelo relacional construido con restricciones y tipos de datos

Con el objetivo de estructurar la información del caso en una base de datos, se construyó un modelo relacional que se compone de las siguientes relaciones:

Las restricciones usadas son las siguientes:

PK: Primary Key
FK: Foreign Key

SA: System Assigned
UA: User Assigned
NN: Not Null

Los tipos de dato de cada atributo está puesto entre paréntesis () antes del nombre de cada atributo.

Inmueble(idInmueble, operadorAsociado, Categoria, Capacidad, Ubicacion, Menaje, TamanoEnMtSq)

(long) idInmueble: PK
(long) operadorAsociado: FKOperador.idOperador , NN
(String) Categoria: SA
(int) Capacidad: NN, UA
(String) Ubicacion: NN, UA
(String) Menaje: NN, UA
(int) TamanoEnSqMt: NN, UA

Operador(idOperador, relacionConUniandes)

(long) idOperador: PK
(String) relacionConUniandes: NN

Servicio(idServicio, nombre, descripcion, horaApertura, horaCierre, costoAdicional)

(long) idServicio: PK
(String) nombre: NN
(String) descripcion:
(String) horaApertura:
(String) horaCierre:
(long) costoAdicional: NN

OfertaAlojamiento(idOferta, precioBase, inmuebleAsociado, ofertaActiva, ofertaDisponible, modalidadTemporal)

(long) idOferta: PK
(long) precioBase: NN
(long) inmuebleAsociado: FKinmueble.idInmueble, NN
(boolean) ofertaActiva: NN, SA
(boolean) ofertaDisponible: NN, SA
(String) modalidadTemporal: NN, UA

Cliente(idCliente, vinculoConUniandes)

(long) idCliente: PK
(String) vinculoConUniandes: NN

Reserva(idClienteAsociado, idOfertaAsociada, fechaReserva, fechaInicialReserva, fechaFinalReserva, promocionesActivas, fechaCancelacion)

(long) idClienteAsociado: FKCliente.idCliente, PK
(long) idOfertaAsociada: FKOfertaAlojamiento.idOferta, PK
(TIMESTAMP) fechaReserva: PK
(TIMESTAMP) fechaInicialReserva: NN
(TIMESTAMP) fechaFinalReserva: NN
(String) promocionesActivas: NN
(TIMESTAMP) fechaCancelacion:

EmpresaAlojamiento(idOperador, tipoEmpresa, horaApertura, horaCierre)

(long) idOperador: FKOperador.IdOperador, PK
(String) tipoEmpresa: NN
(String) horaApertura:
(String) horaCierre:

ServiciosPorOferta(idOfertaAlojamiento, idServicioOfrecido)

(long) idOfertaAlojamiento: FKOfertaAlojamiento.idOferta, PK
(long) idServicioOfrecido: FKServicio.ofertaAsociada, PK

Vivienda(idInmueble, tipoVivienda, numeroHabitaciones)

(long) idInmueble: FKInmueble.IdInmueble, PK
(String) tipoVivienda: NN, UA
(int) numeroHabitaciones: NN

Habitacion(idInmueble, tipoHabitacion)

(long) idInmueble: FKInmueble.IdInmueble, PK
(String) tipoHabitacion: NN, UA

PersonaNatural(idOperador, ofreceAlojamientoPorDias)

(long) idOperador: FKOperador.IdOperador, PK
(boolean) ofreceAlojamientoPorDias: NN, UA

Información sobre relaciones entre entidades:

- Vivienda y Habitación son clases hijas que heredan de 'Inmueble'
- PersonaNatural, EmpresaAlojamiento son clases hijas que heredan de 'Operador'
- Inmueble - Operador: Relación muchos a uno

- Inmueble - OfertaAlojamiento: Relación unidireccional de delegación sencilla, donde a una OfertaAlojamiento le corresponde un Inmueble y accede a su información.
- Servicio - OfertaAlojamiento: Relación muchos a muchos
- OfertaAlojamiento - Cliente: Relación muchos a muchos
- Reserva: Es una clase de asociación, brinda información adicional sobre la asociación entre Cliente y OfertaAlojamiento.

A continuación, se pueden visualizar cada una de las relaciones en conjunto con sus restricciones de dominio.

Inmueble			
idInmueble	**operadorAsociado**	**Categoria**	**Capacidad**
PK	FKOperador.idOperador, PK	SA	NN, UA
192837465	987605431	"Business"	4

Ubicacion	**Menaje**	**TamanoEnSqMt**
NN, UA	NN, UA	NN, UA
"Cra 28b #45A-80, Teusaquillo, bogotá"	"Se incluye un set de cubiertos, un vaso y un plato h	60

Relación 1. Inmueble

Vivienda		
idInmueble	**tipoVivienda**	**numeroHabitaciones**
FKInmueble.IdInmueble, PK	NN, UA	NN
192837465	"Individual"	2

Relación 2. Vivienda

Habitacion	
idInmueble	**tipoHabitacion**
FKInmueble.IdInmueble, PK	NN, UA
192837462	"Compartida"

Relación 3. Habitacion

Operador	
idOperador	**relacionConUniandes**
PK	NN
987605431	"Miembro de la comunidad Fenicia"

Relación 4. Operador

PersonaNatural	
idOperador	**ofreceAlojamientoPorDias**
FKOperador.IdOperador, PK	NN, UA
987605431	TRUE

Relación 5. PersonaNatural

EmpresaAlojamiento			
idOperador	**tipoEmpresa**	**horaApertura**	**horaCierre**
FKOperador.idOperador, PK	NN		
987605432	"Vivienda Universitaria"	8:00	22:00

Relación 6. EmpresaAlojamiento

Servicio		
idServicio	**nombre**	**descripcion**
PK	NN	
1	"Acceso a Jacuzzi"	"El cliente puede acceder al jacuzzi principal del hotel"

horaApertura	**horaCierre**	**costoAdicional**
		NN
10:00	22:00	0

Relación 7. Servicio

OfertaAlojamiento		
idOferta	**precioBase**	**inmuebleAsociado**
PK	NN	FKinmueble.idInmueble, PK
132465879	12000000	192837465

ofertaActiva	**ofertaDisponible**	**modalidadTemporal**
NN, SA	NN, SA	NN, UA
TRUE	FALSE	"Plan Semestral"

Relación 8. OfertaAlojamiento

Cliente	
idCliente	**vinculoConUniandes**
PK	NN
334567898	"Estudiante de Posgrado"

Relación 9. Cliente

Reserva		
idClienteAsociado	**idOfertaAsociada**	**fechaReserva**
FKCliente.idCliente, PK	FKOfertaAlojamiento.idOferta, PK	PK
334567898	132465879	26/02/2023

fechaInicialReserva	**fechaFinalReserva**	**promocionesActivas**	**fechaCancelacion**
NN	NN	NN	
28/02/2023	15/03/2023	"No hay promociones activas"	NULL

Relación 10. Reserva

3.1.1 Cambios al modelo relacional para la Iteración 3

Debido a que el RF7 nos pide poder identificar de manera única las reservas colectivas que se realicen se decidió crear una nueva relación. Esta relación tiene dos columnas una con el id de la reserva colectiva y otra columna con el id de la oferta asociada a la reserva. A continuación se muestra la tabla creada

ReservaColectiva	
idReservaColectiva	idOfertaAsociada
PK	FK(ofertaAlojamiento.idOferta),PK
5	2

3.2 Análisis del nivel de normalización del modelo relacional

Primera Forma Normal (1FN): Debido a que todos los atributos de las relaciones son atómicos, es decir, no contienen datos multivalor o colecciones de datos (como una lista o array de elementos), es posible afirmar que se cumple con los requisitos de la primera forma normal.

En forma de aclaración, se tiene en cuenta que el tipo de dato “Date” usado para las fechas y horas del modelo construido cuenta como un solo valor a pesar de contener información de días, meses, año y hora.

Segunda Forma Normal (2FN): Para que una relación cumpla esta forma, es necesario que se encuentre en primera forma normal y que sus atributos no primos dependan de **forma completa** de las llaves candidatas.

Para la relación Inmueble: La llave {idInmueble, operadorAsociado} y la llave {idInmueble, Categoria} logran determinar de forma completa todos los atributos no primos de la relación, por tanto, cumple la 2FN.

Para la relación Habitación y Vivienda: La llave foránea {idInmueble} logra determinar de forma completa sus demás atributos, por tanto, cumple la 2FN.

Para la relación Operador: La llave idOperador puede determinar de forma completa el atributo relacionConUniandes (no primo), por tanto, cumple la 2FN.

Para la relación PersonaNatural: La llave idOperador puede determinar de forma completa el atributo ofreceAlojamientoPorDias (no primo), por tanto, cumple la 2FN.

Para la relación EmpresaAlojamiento: La llave {idOperador, tipoEmpresa} logra determinar de forma completa el resto de los atributos, por tanto, cumple la 2FN.

Para la relación Servicio: La llave candidata {ofertaAsociada, nombre} puede determinar el resto de atributos que son no primos, por tanto, cumple la 2FN.

Para la relación OfertaAlojamiento: la llave {idOferta, inmuebleAsociado} pueden determinar de forma completa todos los demás atributos que son no primos, por tanto, cumple la 2FN.

Para la relación Cliente: La llave idOperador puede determinar de forma completa el atributo relacionConUniandes (no primo), por tanto, cumple la 2FN.

Para la relación Reserva: La llave candidata {idClienteAsociado, idOfertaAsociada, fechaReserva} logra determinar de forma completa los demás atributos de la relación, por tanto, cumple la 2FN.

Conclusión: Todas las relaciones modeladas cumplen la Segunda Forma Normal.

Tercera Forma Normal (3FN): De forma general, debido a que ninguno de los datos modelados en los atributos es un dato derivado, no hay problemas relacionados con la transitividad de atributos no primos. En lugar de modelar un atributo llamado precioFinal en la relación "ofertaAlojamiento" se modeló uno llamado precioBase para que no fuese un dato derivado de ciertos cálculos.

Por otra parte, al analizar de forma detenida cada relación propuesta en el modelo, las únicas relaciones de transitividad que se identifican se dan entre atributos primos y no primos, como en el caso de la relación Inmueble, donde el idInmueble determina la Categoria del inmueble y la Categoria puede determinar el tamaño en metros cuadrados, la capacidad y el menaje. En los demás casos, no es posible que un atributo no primo llegue a determinar otro no primo por medio de inferencia transitiva.

Conclusión: Todo el modelo relacional propuesto se encuentra en Tercera Forma Normal (3FN).

La relación ReservaColectiva mantiene el nivel de normalización 3 porque sus dos únicos atributos son llaves primarias y, por tanto, no existen dependencias entre ellos.

Análisis de la Forma Normal de Boyce Codd en el modelo relacional:

La FNBC se refiere a una condición de normalización más estricta que la 3FN. Para que una relación esté en FNBC, debe cumplir la siguiente propiedad:

Para cada dependencia funcional no trivial (X → Y) en la relación, X debe ser una superclave de la relación. En otras palabras, en una relación en FNBC, todas las dependencias funcionales no triviales deben tener una clave completa como el determinante (parte izquierda de la dependencia).

En el caso del modelo relacional construido para ALOHANDES, no existen relaciones de dependencia triviales, es decir, cada atributo que no corresponde a una llave primaria de cada relación solo puede ser determinado por atributos primarios y de manera no trivial. Por ejemplo, en la relación 'Servicio':

Servicio					
idServicio	**nombre**	**descripcion**	**horaApertura**	**horaCierre**	**costoAdicional**
PK	NN				NN
1	"Acceso a Jacuzzi"	"El cliente puede acceder al jacuzzi principal del hotel"	10:00	22:00	0

Relación 7. Servicio

La llave primaria es el atributo 'idServicio' y todos los demás atributos (nombre, descripcion, horaApertura, horaCierre, costoAdicional) pueden ser determinados de forma no trivial por la llave primaria únicamente.

Una dependencia funcional trivial ocurre cuando el atributo que está en la dependencia también se encuentra en la determinante. Es decir, que se autodetermina así mismo por consecuente. En

el caso de esta relación y el resto del modelo, no existen este tipo de dependencias y, además, todos los atributos que hacen parte de los determinantes en las relaciones de dependencia corresponden a llaves primarias.

---

4 Documentación de los Requerimientos Iteración 3

Requerimientos Funcionales de Modificación:

RF 7:

Para este requerimiento lo primero que se hace es preguntarle al usuario cuales son las reservas que desea realizar. Después de tener la información procedemos a verificar que las ofertas relacionadas a la reserva que quiere hacer el usuario estén disponibles. En caso de estar disponibles se registran las reservas, de lo contrario se envía un mensaje de error.

Posteriormente se añade a la relación de ofertas colectivas el id de la oferta colectiva junto con las ofertas asociadas a las reservas realizadas. Esto es posible por que una oferta solo puede ser asociada a una reserva.

RF 8:

Para este requerimiento se pregunta al usuario el id de reserva colectiva que desea eliminar. Teniendo el id de la reserva colectiva a eliminar se procede a buscar todas las ofertas asociadas al id de reserva colectiva. Una vez de se tienen los id de las ofertas asociadas se procede a eliminar de la tabla reservas estos ids. Por ultimo se elimina de la tabla de reserva colectiva todos los valores que coincidan con el id de reserva colectiva a eliminar

RF 9:

Para este requerimiento se le pide al usuario el id de la oferta que se va a deshabilitar, en caso de que la oferta no tenga reservas asociadas, simplemente se cambia la disponibilidad de la reserva al estado que muestra que no está disponible. Si sí tiene reservas asociadas se procede a eliminar las reservas asociadas y a la oferta deshabilitada y posteriormente se asignan a otra oferta que sí esté disponible

RF 10:

Para este requerimiento se pregunta al usuario por el id de la oferta que desea rehabilitar, teniendo esto se procede a cambiar el estado de la oferta de no disponible, a disponible.

Requerimientos Funcionales de Consulta:

RFC 7:

Para solucionar este requerimiento, fue necesario implementar 3 métodos diferentes que permitieran resolver las preguntas de análisis sobre la base de datos, estos métodos son:

1. CalcularFechasMayorDemanda
2. CalcularFechasMayoresIngresos
3. CalcularFechasMenorOcupación

En conjunto, estos métodos permiten resolver el requerimiento planteado.

RFC8 - Encontrar los clientes frecuentes

El método `encontrarClientesFrecuentes(long idInmueble)` en la clase `PersistenciaAlohandes` se usa para encontrar la información de los clientes frecuentes de un alojamiento específico. Un cliente se considera frecuente si ha utilizado (o tiene reservado) ese alojamiento al menos en tres ocasiones o durante al menos 15 noches.

Este método realiza una consulta SQL para seleccionar aquellos clientes que cumplen con estos criterios. La consulta se realiza en la tabla `RESERVA`, uniendo las tablas `OFERTAALOJAMIENTO` e `INMUEBLE` en el `IDINMUEBLE`. La información obtenida incluye el `IDCLIENTE`, el número de reservas realizadas y la duración total de las reservas.

RFC9 - Encontrar las ofertas de alojamiento que no tienen mucha demanda

El método `encontrarOfertasPocaDemanda()` en la clase `PersistenciaAlohandes` se utiliza para encontrar las ofertas de alojamiento que no han recibido clientes en periodos superiores a 1 mes.

Este método realiza una consulta SQL en la tabla `OFERTAALOJAMIENTO`, uniendo las tablas `RESERVA` e `INMUEBLE` en el `IDINMUEBLE`, y seleccionando aquellas ofertas que no han tenido reservas en más de un mes. La consulta devuelve una lista de ofertas de alojamiento, incluyendo el `IDOFERTA`, la última fecha de reserva y la cantidad de días desde esa fecha hasta la fecha actual.

5 Resultados logrados Iteración 4

- En la entrega de la Iteración 4, se logró completar la interfaz de la aplicación, se crearon 3 interfaces distintas (una para los clientes, otra para los operadores y otra para el GERENTE). El acceso se logra al ingresar credenciales que se encuentran en la tabla de clientes u operadores o ingresando las credenciales del GERENTE:

  usuario: gerente

  clave: alohandes

- También se logró realizar la implementación para todos los requerimientos funcionales de consulta adicionales. Estos se muestran en interfaz. Aquellos que solo debería ver el Gerente, solo los puede acceder él en su interfaz.

- Se logró preparar pruebas en Oracle SQL y revisar que las consultas se realizaran como se esperaba. Se incluyen estas pruebas en el archivo adjunto EXCEL.
- Se logró garantizar que las consultas se resuelvan en menos de 0,8 segundos. Esto se verificó por Oracle SQL Developer.
- Se pudo hacer la inserción del millón de registros en las diferentes tablas. Estos se construyeron con un script Python que los generaba. No tienen mucho sentido respecto al mundo real, pero sirven para ver la eficiencia de las consultas.

6 Resultados no logrados Iteración 4

- No fue posible terminar de implementar todos los métodos CRUD de las distintas clases de negocio por medio de la interfaz. Se hizo un enfoque principal en las consultas y modificaciones propias de cada Iteración.

7 División de interfaces por rol dentro de ALOHANDES

Una funcionalidad agregada en esta iteración que se considera de suma importancia, fue la de implementar una verificación de credenciales por medio de un Jframe y Jpanel que aparecen siempre al iniciar la aplicación. Se construyó un método transaccional para recibir las credenciales del usuario y consultar en la base de datos si corresponden a las de un:

1. Cliente
2. Operador
3. El gerente general de ALOHANDES

Las credenciales asignadas al generete son:

- Usuario: gerente
- Clave: alohandes

Este fue el método transaccional construido para verificar estas credenciales:

```java
public String verificarCredenciales(String usuario, String clave) {
    log.info("Verificando credenciales: usuario = " + usuario + "y clave =  "+ clave);
    PersistenceManager pm = pmf.getPersistenceManager();
    Transaction tx = pm.currentTransaction();

    try {
        tx.begin();

        // Verificar si las credenciales corresponden a un operador
        Operador operador = sqlOperador.darOperadorPorCredenciales(pm, usuario, clave);
        if (operador != null) {
            log.info("Credenciales válidas para operador");
            return "Operador";
        }

        // Verificar si las credenciales corresponden a un cliente
        Cliente cliente = sqlCliente.darClientePorCredenciales(pm, usuario, clave);
        if (cliente != null) {
            log.info("Credenciales válidas para cliente");
            return "Cliente";
        }

        log.info("Credenciales inválidas, no existen en la base de datos.");
        return "Inexistente";

    } catch (Exception e) {
        log.error("Exception: " + e.getMessage(), e);
        return "Inexistente";
    } finally {
        if (tx.isActive()) {
            tx.rollback();
        }
        pm.close();
    }
}
```

Este método involucra los llamados a diferentes métodos de las clases SQL (persistencia) para realizar la consulta del usuario y la clave en la base de datos.

Para cada rol diferente dentro de ALOHANDES, se creó un archivo de configuración de interfaz distinto, con el objetivo de utilizar el mismo Jframe pero cargarle un Panel con botones y funcionalidades distintas.

8 Carga masiva de datos

Con el objetivo de analizar la necesidad de optimizar las consutas sobre la base de datos, fue necesario insertar una masiva cantidad de registros sobre las diferentes tablas y analizar qué índices podían ser creados para optimizar cada requerimiento funcional de consulta.

En total, se insertaron un millón de registros repartidos en todas las tablas de la base de datos de la siguiente forma:

- 400 mil registros en la tabla de Cliente
- 400 mil registros en la tabla de Operador
- 80 mil registros en la tabla de Reserva
- 20 mil registros en la tabla de OfertaAlojamiento
- 2 mil registros en la tabla de Inmueble

- 2 mil registros en la tabla de Servicio
- mil registros en la tabla EmpresaAlojamiento
- mil registros en la tabla PersonaNatural
- 2 mil registros en la tabla ServiciosPorOFerta
- mil registros en la tabla de Vivienda
- mil registros en la tabla de PersonaNatural

Esta carga masiva se realizó por medio de **inserciones** en la base de datos en el Sistema Manejador de Bases de Datos: Oracle SQL Developer. Para esto, se preparó un script en python que generase los insert para las distintas tablas. Este script se ha adjuntado en el espacio de entrega del repositorio (carpeta ITERACION4_Docs).

9 Documentación de los Requerimientos Iteración 4

Requerimientos Funcionales de Consulta:

RFC10: CONSULTAR CONSUMO EN ALOHANDES

Se quiere conocer la información de los usuarios que realizaron al menos una reserva de una determinada oferta de alojamiento en un rango de fechas. Los resultados deben ser clasificados según un criterio deseado por quien realiza la consulta. En la clasificación debe ofrecerse la posibilidad de agrupamiento y ordenamiento de las respuestas según los intereses del usuario que consulta como, por ejemplo, por los datos del cliente, por oferta de alojamiento y por tipo de alojamiento.

NOTA: Respetando la privacidad de los clientes, cuando un cliente proveedor hace esta consulta obtiene la información de su propia actividad, mientras que el administrador obtiene toda la información de cualquiera de los clientes. Ver RNF1.

```
public List<Object[]> encontrarClientesPorOfertaRangoFechas(String idOfertaAsociads
 PersistenceManager pm = pmf.getPersistenceManager();

 SimpleDateFormat f = new SimpleDateFormat("dd/MM/yy");
 String fechaInicial = f.format(fechaIniciall);
 String fechaFinal = f.format(fechaFinall);
 String sql ="";
 if (orden.equals(""))
 {
     sql = "select c.idcliente as id, c.vinculoconuniandes,c.usuario,c.clave, count
               + "from cliente c inner join reserva r on c.idcliente = r.idcliente
               + "inner join ofertaalojamiento o on r.idofertaasociada=o.idoferta
               + "inner join inmueble i on i.idinmueble=o.inmuebleasociado "
               + "where r.idofertaasociada="+idOfertaAsociads+" and r.fechainicial
               + "group by c.idcliente, c.vinculoconuniandes,c.usuario,c.clave,i.c
 }
 else
 {
```

```
public List<Object[]> encontrarClientesPorOfertaRangoFechas1(String idCliente, String
  PersistenceManager pm = pmf.getPersistenceManager();

  SimpleDateFormat f = new SimpleDateFormat("dd/MM/yy");
  String fechaInicial = f.format(fechaIniciall);
  String fechaFinal = f.format(fechaFinall);
  String sql ="";

      sql = "select c.idcliente as id, c.vinculoconuniandes,c.usuario,c.clave, r.fecha
            "from cliente c inner join reserva r on c.idcliente = r.idclienteasociado
            "inner join ofertaalojamiento o on r.idofertaasociada=o.idoferta "+
            "inner join inmueble i on i.idinmueble=o.inmuebleasociado  "+
            "where c.idcliente="+idCliente+" and r.idofertaasociada="+idOfertaAsociad

   Query q = pm.newQuery(SQL, sql);
   q.setResultClass(Object[].class);
   List<Object[]> consulta = (List<Object[]>) q.executeList();
   return consulta;
}
```

Para responder a este requerimiento de consulta, se crearon dos métodos: encontrarClientesPorOfertaRangoFechas y encontrarClientesPorOfertaRangoFechas1, ambos, en conjunto, se encargan de buscar a los clientes que realizaron reservas (al menos una) para cierta Oferta de Alojamiento determinada en un rango de fechas solicitado. Este requerimiento se puedo agilizar con la creación de índices sobre las tablas: Reservas y OfertaAlojamiento.

RFC11 - CONSULTAR CONSUMO EN ALOHANDES - RFC10-V2

Se quiere conocer la información de los usuarios QUE NO realizaron al menos una reserva de una determinada oferta de alojamiento en un rango de fechas. En la clasificación debe ofrecerse la posibilidad de agrupamiento y ordenamiento de las respuestas según los intereses del usuario que consulta como, por ejemplo, por los datos del cliente, por oferta de alojamiento y por tipo de alojamiento.

NOTA: Respetando la privacidad de los clientes, cuando un cliente proveedor hace esta consulta obtiene la información de su propia actividad, mientras que el administrador obtiene toda la información de cualquiera de los clientes. Ver RNF1.

```
public List<Object[]> encontrarNoClientesPorOfertaRangoFechas(String idOfertaAsociad
  PersistenceManager pm = pmf.getPersistenceManager();

  SimpleDateFormat f = new SimpleDateFormat("dd/MM/yy");
  String fechaInicial = f.format(fechaIniciall);
  String fechaFinal = f.format(fechaFinall);
  String sql ="";
  if (orden.equals(""))
  {
      sql = "select * "+
            "from cliente c "+
            "where c.idcliente not in "+
             "(select c.idcliente as id "
                + "from cliente c inner join reserva r on c.idcliente = r.idcliente
                + "inner join ofertaalojamiento o on r.idofertaasociada=o.idoferta
                + "inner join inmueble i on i.idinmueble=o.inmuebleasociado "
                + "where r.idofertaasociada="+idOfertaAsociads+" and r.fechainicial
                " fetch first 50 rows only";
  }
  else
  {
```

Este requerimiento fue similar en cuanto a su construcción respecto al RFC11. En este caso, se querían encontrar qué clientes no habían reservado cierta oferta en un rango de fechas dado, para esto, se realizaron ligeros cambios sobre el código del método que resuelve el RFC10.

RFC12: CONSULTAR FUNCIONAMIENTO

Muestra, para cada semana del año, la oferta de alojamiento con más ocupación, la oferta de alojamiento con menos ocupación, los operadores más solicitados y los operadores menos solicitados. Las respuestas deben ser sustentadas por el detalle de las ofertas de alojamiento y operadores correspondientes. Esta operación es realizada el gerente general de AlohAndes.

```
public ResultadosConsulta consultarFuncionamiento(int anio) {
    PersistenceManager pm = pmf.getPersistenceManager();
    Transaction tx = pm.currentTransaction();
    ResultadosConsulta resultados = new ResultadosConsulta();

    try {
        tx.begin();

        for (int mes = 1; mes <= 12; mes++) {
            resultados.addOfertaMaxOcupacion(encontrarOfertaMaxOcupacion(pm, mes, anio));
            resultados.addOfertaMinOcupacion(encontrarOfertaMinOcupacion(pm, mes, anio));
            resultados.addOperadoresMasSolicitados(encontrarOperadoresMasSolicitados(pm, mes, anio));
            resultados.addOperadoresMenosSolicitados(encontrarOperadoresMenosSolicitados(pm, mes, anio));
        }

        tx.commit();
    } catch (Exception e) {
        log.error("Exception: " + e.getMessage(), e);
        resultados = null;
    } finally {
        if (tx.isActive()) {
            tx.rollback();
        }
        pm.close();
    }

    return resultados;
}
```

El método construido en PersistenciaAlohandes consiste en la reunión de 4 submétodos que se encargan de realizar las 4 consultas solicitadas para revisar el funcionamiento de la aplicación.

El funcionamiento se revisa por mes para un año determinado. Haberlo hecho por semana suponía un cambio muy grande en el modelo relacional y debíamos cambiar la implementación de todos los métodos hasta la fecha construidos.

RFC13 - CONSULTAR LOS BUENOS CLIENTES

Los buenos clientes son de tres tipos: aquellos que hacen reservas en AlohAndes al menos una vez al mes, aquellos que siempre reservan alojamientos costosos (Entiéndase costoso, por ejemplo, como mayor a USD 150 por noche) y aquellos que siempre reservan suites. Esta consulta retorna toda la información de dichos clientes, incluyendo la que justifica su calificación como buenos clientes. Esta operación es realizada únicamente por el gerente general de AlohAndes */

```java
public List<Object[]> consultarBuenosClientes() {
    PersistenceManager pm = pmf.getPersistenceManager();
    Transaction tx = pm.currentTransaction();

    List<Object[]> buenosClientes = new ArrayList<>();

    try {
        tx.begin();

        // Query 1
        String sql1 = "SELECT c.*, " +
                      "(SELECT COUNT(*) FROM Reserva r WHERE c.idCliente = r.idClienteAsociado AND EXT
                      "AS NumeroReservasAlAño " +
                      "FROM Cliente c " +
                      "WHERE (SELECT COUNT(DISTINCT EXTRACT(MONTH FROM r.fechaReserva)) " +
                      "FROM Reserva r WHERE c.idCliente = r.idClienteAsociado AND EXTRACT(YEAR FROM r.
        Query q1 = pm.newQuery(SQL, sql1);
        q1.setResultClass(Object[].class);
        List<Object[]> resultadoQuery1 = (List<Object[]>) q1.executeList();
        buenosClientes.addAll(resultadoQuery1);

        // Query 2
        String sql2 = "SELECT c.*, " +
                      "(SELECT COUNT(*) FROM Reserva r, OfertaAlojamiento o " +
                      "WHERE c.idCliente = r.idClienteAsociado AND r.idOfertaAsociada = o.idOferta " +
                      "AND o.precioBase >= 700000 AND EXTRACT(YEAR FROM r.fechaReserva) = 2022) " +
                      "AS Cantidad_Reservas_Costosas_AlAño " +
                      "FROM Cliente c " +
                      "WHERE (SELECT COUNT(DISTINCT EXTRACT(MONTH FROM r.fechaReserva)) " +
                      "FROM Reserva r WHERE c.idCliente = r.idClienteAsociado AND EXTRACT(YEAR FROM r.
```

```java
        // Query 2
        String sql2 = "SELECT c.*, " +
                      "(SELECT COUNT(*) FROM Reserva r, OfertaAlojamiento o " +
                      "WHERE c.idCliente = r.idClienteAsociado AND r.idOfertaAsociada = o.idOferta "
                      "AND o.precioBase >= 700000 AND EXTRACT(YEAR FROM r.fechaReserva) = 2022) " +
                      "AS Cantidad_Reservas_Costosas_AlAño " +
                      "FROM Cliente c " +
                      "WHERE (SELECT COUNT(DISTINCT EXTRACT(MONTH FROM r.fechaReserva)) " +
                      "FROM Reserva r WHERE c.idCliente = r.idClienteAsociado AND EXTRACT(YEAR FROM
        Query q2 = pm.newQuery(SQL, sql2);
        q2.setResultClass(Object[].class);
        List<Object[]> resultadoQuery2 = (List<Object[]>) q2.executeList();
        buenosClientes.addAll(resultadoQuery2);

        // Query 3
        String sql3 = "SELECT c.*, " +
                      "(SELECT COUNT(*) FROM Reserva r, OfertaAlojamiento o, Inmueble i " +
                      "WHERE c.idCliente = r.idClienteAsociado AND r.idOfertaAsociada = o.idOferta "
                      "AND o.inmuebleAsociado = i.idInmueble AND i.Categoria = 'SUITE' " +
                      "AND EXTRACT(YEAR FROM r.fechaReserva) = 2022) " +
                      "AS Numero_Reservas_Suite_Al_Año " +
                      "FROM Cliente c " +
                      "WHERE (SELECT COUNT(DISTINCT EXTRACT(MONTH FROM r.fechaReserva)) " +
                      "FROM Reserva r WHERE c.idCliente = r.idClienteAsociado AND EXTRACT(YEAR FROM
        Query q3 = pm.newQuery(SQL, sql3);
        q3.setResultClass(Object[].class);
        List<Object[]> resultadoQuery3 = (List<Object[]>) q3.executeList();
        buenosClientes.addAll(resultadoQuery3);
```

Para responder a este requerimiento de consulta, se dividió la respuesta en 3 queries independientes, los cuales luego se reunen en una lista para mostrarlos en la interfaz.

El análisis de los buenos clientes se hace sobre el año 2022, pues es el año inmediatamente anterior al actual y contiene los registros completos en sus meses.

10 ¿Cómo se garantiza la Transaccionalidad a lo largo de todo el proyecto?

La transaccionalidad, que se puede garantizar por medio del cumplimiento de los 4 principios ACID:

1. Atomicidad
2. Coherencia
3. Aislamiento
4. Durabilidad

Se garantizan a lo largo del proyecto de la siguiente forma:

- **La Atomicidad** se posibilita porque en todos los requerimientos de modificación e incluso los de consulta, se realiza un ROLLBACK a la base de datos en caso que ocurra una SQLException o una Excepción ocasionada por el incumplimiento de alguana regla de negocio, falla de la conexión con la base de datos o falla en la ejecución de los requerimientos transaccionales.
- **La Coherencia** se garantiza porque todo el tratamiento de los requerimientos en la base de datos se realiza con datos reales, aquellos dentro del sistema relacional de ALOHANDES y todas las modificaciones realizadas siempre toman la información más actualizada para cada momento, pues se realizan en tiempo real. Por lo tanto, siempre se está tratando con datos que reflejan el mundo real.
- **El Aislamiento** se logra garantizar gracias a los CANDADOS de lectura y escritura que el SISTEMA MANEJADOR DE BASES DE DATOS (DBMS) Oracle SQL Developer pone sobre las tablals y tuplas que se están consultando. Estos candados corresponden al nivel de aislamiento SERIALIZABLE que maneja por defecto y que incluye siempre candados de escritura de LARGA DURACIÓN.
- **La Durabilidad** se posibilita porque en la estructura de todos los requerimientos de modificación sobre la base de datos, se realiza un commit luego de terminar satisfactoriamente una transacción. De esta forma, siempre se persisten los datos modificados para su posterior uso.

11 Optimización de las consultas para la Iteración 4

Con la finalidad de mejorar el tiempo de respuesta de la base de datos para responder a los requerimientos propuestos, se crearon los siguientes índices por medio de Oracle SQL Developer:

Se toma en consideración que el tipo de estructura de dato por defecto para los índices es el **árbol B+** (BTREE) y que a la hora de crear un índice sobre un atributo o atributos que no son la llave primaria, se crea un **índice secundario** sobre esa tabla.

Índices creados:

Para optimizar la consulta del RFC10:

1. (Ya había sido creado por el DBMS) Índice PRIMARIO SIMPLE en la tabla `Reserva` para el atributo `idClienteAsociado`:

```
CREATE INDEX idx_reserva_id_cliente_asociado
ON Reserva(idClienteAsociado);
COMMIT;
```

- **Justificación:** Al utilizar el atributo `idClienteAsociado` en la cláusula de join y en la cláusula de group by de la consulta, este índice mejorará la velocidad de las operaciones de join y agrupamiento al buscar y agrupar los registros de la tabla `Reserva` por el campo `idClienteAsociado`.

2. Índice PRIMARIO SIMPLE en la tabla `Reserva` para el atributo `idOfertaAsociada`:

```
CREATE INDEX idx_reserva_id_oferta_asociada
ON Reserva(idOfertaAsociada);
```

- **Justificación:** Al utilizar el atributo `idOfertaAsociada` en la cláusula de join de la consulta, este índice ayudará a mejorar el rendimiento al buscar y unir los registros de la tabla `Reserva` con la tabla `OfertaAlojamiento` de manera más eficiente.

3. (Ya había sido creado por el DBMS) Índice PRIMARIO SIMPLE en la tabla `Inmueble` para el atributo `idInmueble`:

```
CREATE INDEX idx_inmueble_id_inmueble
ON Inmueble(idInmueble);
```

- **Justificación:** Al utilizar el atributo `idInmueble` en la cláusula de join de la consulta, este índice mejorará la velocidad de las operaciones de join al buscar y unir los registros de la tabla `Inmueble` con la tabla `OfertaAlojamiento` de manera más eficiente.

Para optimizar la consulta del RFC11:

Para optimizar esta consulta, no fue necesario añadir nuevos índices, los que habían sido creados para el RFC10 resultan ser convenientes para esta consulta también.

Para optimizar la consulta del RFC12:

1. Índice SECUNDARIO SIMPLE en la tabla `Reserva` para el atributo `fechaInicialReserva`:

```
CREATE INDEX idx_reserva_fecha_inicial
ON Reserva(fechaInicialReserva);
```

- **Justificación:** El índice en `fechaInicialReserva` ayudará a acelerar las consultas que involucran la condición `r.fechaInicialReserva BETWEEN TO_DATE('" + fechaInicial + "', 'DD-MM-YYYY') AND TO_DATE('" + fechaFinal + "', 'DD-MM-YYYY')`. Al tener un índice en este atributo, se mejorará la eficiencia de búsqueda y filtrado de las reservas que caen dentro del rango de fechas especificado.

2. (Ya había sido creado por el DBMS) Índice PRIMARIO SIMPLE en la tabla `Operador` para el atributo `idOperador`:

```
CREATE INDEX idx_operador_id_operador
ON Operador(idOperador);
COMMIT;
```

- **Justificación:** El índice en `idOperador` mejora el rendimiento de las consultas que filtran registros por este atributo. Tanto `encontrarOperadoresMasSolicitados` como `encontrarOperadoresMenosSolicitados` realizan una serie de uniones y agrupaciones en las tablas `Operador`, `Inmueble`, `OfertaAlojamiento` y `Reserva` para obtener los operadores más y menos solicitados. Un índice en `idOperador` agilizará la búsqueda y agrupación de los operadores según las condiciones de fecha establecidas en las consultas.

Para optimizar la consulta del RFC13

1. Índice PRIMARIO SIMPLE en la tabla `Reserva` para el atributo `fechaReserva`:

```
CREATE INDEX idx_Reserva_fechaReserva
ON Reserva (fechaReserva);
COMMIT;
```

- **Justificación:** El índice en `fechaReserva` mejora el rendimiento de las consultas que involucran la condición de igualdad o rango en este atributo. En este caso, el índice ayudará a acelerar las consultas que filtran registros por año, como la cláusula `EXTRACT(YEAR FROM r.fechaReserva) = 2022`. Al tener un índice en `fechaReserva`, el motor de la base de datos puede realizar búsquedas más rápidas y evitará explorar todos los registros de la tabla.

2. (Ya había sido creado por el DBMS) Índice PRIMARIO SIMPLE en la tabla `OfertaAlojamiento` para el atributo `idOferta`:

```
CREATE INDEX idx_OfertaAlojamiento_idOferta
ON OfertaAlojamiento (idOferta);
COMMIT;
```

- **Justificación:** El índice en `idOferta` mejora el rendimiento de las consultas que necesitan buscar ofertas específicas. En este caso, se utiliza en la subconsulta de la Query 2, donde se compara `r.idOfertaAsociada` con `o.idOferta`. Al tener un índice en `idOferta`, se acelerará la búsqueda de ofertas correspondientes y se reducirá el tiempo de ejecución de la consulta.

3. Índice SECUNDARIO SIMPLE en la tabla `OfertaAlojamiento` para el atributo `inmuebleAsociado`:

```
CREATE INDEX idx_OfertaAlojamiento_inmuebleAsociado
ON OfertaAlojamiento (inmuebleAsociado);
```

- **Justificación:** El índice en `inmuebleAsociado` mejora el rendimiento de las consultas que buscan ofertas asociadas a un inmueble específico. En este caso, se utiliza en la subconsulta de la Query 3, donde se compara `o.inmuebleAsociado` con `i.idInmueble`. El índice en `inmuebleAsociado` facilitará la búsqueda y recuperación eficiente de las ofertas relacionadas con un inmueble determinado.

4. Índice SECUNDARIO SIMPLE en la tabla `Inmueble` para el atributo `Categoria`:

```
CREATE INDEX idx_Inmueble_Categoria
ON Inmueble (Categoria);
```

- **Justificación:** El índice en `Categoria` mejora el rendimiento de las consultas que filtran registros por este atributo. En el contexto de la subconsulta de la Query 3, se realiza una búsqueda de inmuebles con la condición `i.Categoria = 'SUITE'`. El índice en `Categoria` permitirá una búsqueda rápida de los inmuebles que cumplen con esa condición, lo que agilizará la ejecución de la consulta.

12 Conclusiones del proyecto

Se tuvo en cuenta el modelo conceptual y el modelo relacional que se planteó previamente con el objetivo de implementar el negocio Alohandes. Para esto se usaron conocimientos como SQL con el objetivo de realizar diferentes requerimientos funcionales planteados relacionados con el negocio Alohandes. Se creó una base de datos con datos artificiales con el objetivo de verificar el funcionamiento de la base de datos. Se creó un proyecto en java con el objetivo de plasmar el negocio a través de una aplicación.

La aplicación transaccional mantiene un buen rendimiento, lo cual se ve reflejado en el rapido tiempo de respuesta de las consultas, los cuales se logran con los índices construidos. La interfaz de la aplicación muestra un ejemplo de una aplicación real para un sistema transaccional. Las consultas respetan los 4 principios de la Transaccionalidad, pues mantienen Atomicidad en todas las ejecuciones de los métodos de consulta y modificación.

La transaccionalidad, que es un concepto central del curso y del proyecto, se garantiza al materializar cada uno de sus cuatro componentes ACID: Atomicidad, Coherencia, Aislamiento y Durabilidad. El Sistema Manejador de Bases de Datos Relacionales Oracle SQL Developer tiene un rol fundamental a la hora de garantizar la transaccionalidad y algunas de sus tareas como el establecimiento de candados de lectura y escritura, permiten el correcto funcionamiento del sistema ALOHANDES.

13 Bibliografía

1. *Enunciado del caso Iteración 1. Curso Sistemas Transaccionales 2023-10. Tomado de los contenidos del curso en Bloque Neón.*

2. NORMALIZACIÓN DE BASES DE DATOS RELACIONALES. (n.d.-b). CODIDEEP. Tomado de https://codideep.com/blogpost/normalizacion-de-bases-de-datos-relacionales