ITERACION 4

SISTEMAS TRANSACCIONALES

CASO ALOHANDES

PRIMER SEMESTRE 2023

FELIPE NUÑEZ PINILLOS

JERONIMO VARGAS RENDON

Índice:

1. Introducción:

Continuando con el caso de estudio del curso “AlohAndes”, en esta segunda Iteración del curso se desarrolla una plataforma más realista en la cual ya hay una creación de bases de datos definida y ajustada a las necesidades de este negocio. En esta entrega, se encuentran el JDO y las sentencias SQL necesarias para cumplir con los requerimientos funcionales, requerimientos funcionales de consulta y requerimientos no funcionales fundamentales para el correcto funcionamiento de la plataforma destinada para AlohAndes.

2. Requerimientos Funcionales

Nombre	RF1. **REGISTRAR LOS OPERADORES DE ALOJAMIENTO PARA ALOHANDES**
Resumen	Este requerimiento se encarga de registrar a los proveedores (hoteles, hostales, viviendas universitarias, arriendo de una propiedad de un miembro de la comunidad o persona natural ofreciendo una habitación ensu vivienda).
Entradas	
Nombre: String	
Dirección: String	
Barrio: String	
Tipo_Vivienda: Obj (Hotel, Hostal, ViviendaUniversitaria, Particular)	
Parámetros de Creación del Objeto a crear	
Resultados	
Se crea un nuevo operador de alojamiento y se caracteriza por el tipo de alojador que es (Hotel, Hostal, ViviendaUniversitaria, Particular).	
RNF asociados	
Persistencia: La información debe ser persistente de forma que se guarde al operador registrado inmediatamente se le pueda asignar unavivienda para arrendar. (su vivienda o hotel/hostal)	
Privacidad: La información personal de los operadores y de sus viviendas debe estar asegurada seguramente para asegurar s	

Nombre	RF2. **REGISTRAR PROPUESTAS DE ALOJAMIENTOS PARA ALOHANDES**
Resumen	Este requerimiento se encarga de registrar las viviendas que se estarán publicando en AlohAndes, es decir,la oferta de vivienda que se publica en la página.
Entradas	
nombreVivienda: String	
direccionVivienda: String	
barrioVivienda: String	
tipoVivienda: Obj (Hotel, Hostal, ViviendaUniversitaria, Particular)	
Parámetros de Creación del Objeto a crear	
Resultados	
Se registra el inmueble en la aplicación de AlohAndes de forma que se pueda conocer la información de este. Datos como su precio, unaimagen, ubicación y oferta de servicios son indispensables para el arrendatario.	
El inmueble puede ser consultado con facilidad una vez que ha sido añadido a la base de datos, hay transaccionalidad y persistencia en losdatos.	
RNF asociados	
Persistencia: Los datos han de ser persistentes en la base de datos, la única forma de que no se encuentre la información de la vivienda esen el caso dado de que se haya removido.	

Nombre	RF3. **REGISTRAR LAS PERSONAS HABILITADAS PARA UTILIZAR LOS SERVICIOS**
Resumen	Este requerimiento cumple la función de realizar el registro adecuado en la base de datos de laspersonas que pueden tener acceso a los servicios ofrecidos por AlohAndes. Es decir, únicamente miembros de la comunidad universitaria (estudiantes, empleados, profesores extranjeros y profesores de la universidad)
Entradas	
nombreCliente: String	
cedulaCliente: Integer	
tipoCliente: Obj (Variable entre Estudiante, Profesor, Empleado y Miembro de la Comunidad Asociada)	
Resultados	
Se registra al cliente de forma que después podrá acceder a los servicios ofrecidos por AlohAndes	
RNF asociados	
Persistencia: El cliente ha de quedar registrado efectivamente de forma que después al ingresar tenga acceso a todos losservicios de AlohAndes	
Privacidad: Es fundamental que solo se garantice acceso a clientes que tienen vinculo con la universidad, de lo contrario sepodría poner en riesgo al usuario o simplemente acabar beneficiando a un usuario por fuera de los usuarios objetivos de AlohAndes	

Nombre	RF4. **REGISTRAR UNA RESERVA**
Resumen	Este requerimiento cumple la función de crear una nueva reserva en la base de datos añadiendotoda la información relevante de la misma.
Entradas	
cliente_reserva: Cliente	
fecha_inicio: Date	
fecha_fin: Date	
vivienda_reserva: Alojamiento	
fecha_reservacion: Date	
Resultados	
Se crea una nueva reservación en la base de datos la cual puede ser consultada o cancelada posteriormente	
Se calcula la fecha de cancelación usando la duración y fecha de inicio de esta	
RNF asociados	
Persistencia: Debido a que la información debe ser persistente en la base de datos y consultable o modificable en caso deque sea necesario.	
Concurrencia: Este requerimiento será solicitado concurrentemente mas que cualquier otro debido a que es el epicentro de laaplicación.	

Nombre	RF5. **CANCELAR UNA RESERVA**
Resumen	El usuario que realiza la reserva puede cancelar esta dentro de los plazos designados para ello.Esto se debe realizar inmediatamente ya que cualquier aumento en tiempo podría generar un cobro adicional.
Entradas	
num_reserva: Integer o en su defecto reserva: Reserva	
Resultados	
Se cancela la reserva y se realiza el cobro designado al cliente por la cancelación. Esta reserva debe eliminarse de la basede datos de forma que ya no se pueda consultar o modificar.	
RNF asociados	
Concurrencia: Esta también seguramente será un requerimiento de alta concurrencia ya que se ejecutará seguramente másde 1 vez en clientes que utilicen a AlohAndes.	
Persistencia: Ha de haber persistencia, una vez eliminada la reserva ya no podrá aparecer en la base de datos en ningunaparte.	

Nombre	RF6. **RETIRAR UNA OFERTA DE ALOJAMIENTO**
Resumen	El operador de un inmueble puede retirar dicho inmueble de oferta cuando lo considere pertinente. Una vez retirado no se podrán realizar mas reservas sobre este y ya no quedaradisponible para consulta dentro de la oferta de inmuebles.
Entradas	
operador: Operador	
inmueble: Alojamiento	
Resultados	
El inmueble en cuestión es retirado de la base de datos de forma que ya no puedan realizarse reservas de el y que noaparezca en la oferta de inmuebles disponibles.	
RNF asociados	
Concurrencia: Es uno de los requerimientos que se hará con mayor concurrencia en la base de datos de AlohAndes	
Persistencia: Ha de haber persistencia, una vez eliminado el alojamiento ya no podrá aparecer en la base de datos enninguna parte.	

Nombre	RFC1. **MOSTRAR EL DINERO RECIBIDO POR CADA PROVEEDOR DE ALOJAMIENTODURANTE EL AÑO ACTUAL Y EL AÑO CORRIDO**
Resumen	Se espera que la aplicación pueda mostrar estadísticas pasadas sobre el operador de un inmueble, en este caso el dinero que ha ganado en una cantidad variable de tiempo por elarriendo de sus inmuebles.
Entradas	
operadores: Set<Operador>	
anio: Int	
Resultados	
Se espera que la aplicación retorne la cantidad de dinero que ha sido recibido por cada uno de los operadores de alojamientoque están inscritos en la aplicación para el año que ha sido indicado.	
RNF asociados	
Privacidad: La información financiera de los operadores es delicada por ende se debe asegurar su privacidad.	
Distribución: Los datos que se precisan para el funcionamiento de este requerimiento se encuentran disueltos en variasclases de la aplicación. La centralización de la aplicación es fundamental para encontrar la información fácilmente.	

Nombre	RFC2. **MOSTRAR LAS 20 OFERTAS MÁS POPULARES**
Resumen	Se espera que la aplicación pueda mostrar estadísticas sobre las 20 ofertas de inmuebles máspopulares en la aplicación. Es decir aquellas que han tenido una mayor ocupación.
Entradas	
ofertas: Set<Alojamiento>	
Resultados	
Se espera que la aplicación retorne las 20 ofertas que más han sido ocupadas en el tiempo de funcionamiento de laaplicación.	
RNF asociados	
Privacidad: La información financiera de los operadores es delicada por ende se debe asegurar su privacidad.	
Distribución: Los datos que se precisan para el funcionamiento de este requerimiento se encuentran disueltos en variasclases de la aplicación. La centralización de la aplicación es fundamental para encontrar la información fácilmente.	

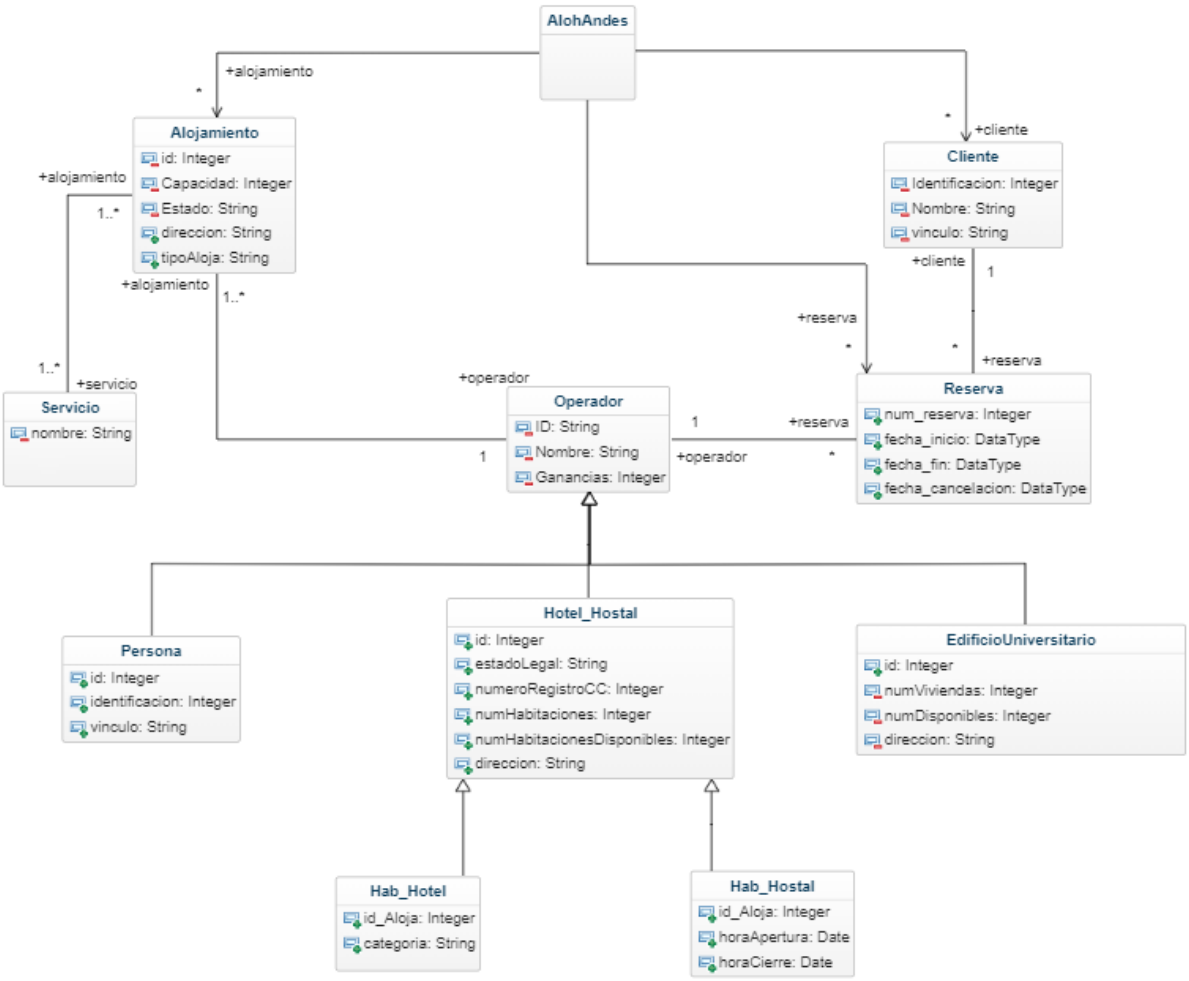
Nombre	RFC3. **MOSTRAR EL ÍNDICE DE OCUPACIÓN DE CADA UNA DE LAS OFERTAS DEALOJAMIENTO REGISTRADAS**
Resumen	Se espera que se pueda monitorear el índice de ocupación de las ofertas de alojamiento. Es decir, que parte del año, mes, u otras medidas de tiempo el inmueble se encuentra ocupado porlos usuarios de AlohAndes.
Entradas	
ofertas: Set<Alojamiento>	
Resultados	
Se espera que la aplicación retorne el indice de ocupacion (tiempo ocupada del tiempo que ha estado siendo ofrecida) paracada uno de los alojamientos siendo ofrecidos en la aplicación.	
RNF asociados	
Persistencia: Los datos a verificar pueden ser llevados en la base de datos tanto tiempo como el que la misma lleva enfuncionamiento, estos datos deben persistir y poder ser consultados.	

Nombre	RFC4. **MOSTRAR LOS ALOJAMIENTOS DISPONIBLES EN UN RANGO DE FECHAS, QUE CUMPLEN CON UN CONJUNTO DE REQUERIMIENTOS DE DOTACIÓN O SERVICIOS. POREJEMPLO, COCINETA, TV CABLE, INTERNET, SALA**
Resumen	Se espera que se pueda filtrar en la aplicación para que el usuario pueda escoger alojamientoscon los servicios que requiere indispensablemente.
Entradas	
ofertas: Set<Alojamiento>	
Resultados	
Se espera que el usuario pueda filtrar por fecha y servicios a los alojamientos que pretende reservar. De forma que elalojamiento seleccionado cumpla con todas las expectativas del usuario.	
RNF asociados	
Distribución: Es un requerimiento que verifica muchas partes de los alojamientos para filtrarlos correctamente, por ende lacentralización de la base de datos puede facilitar el funcionamiento del requisito.	

Nombre	RFC5. **- MOSTRAR EL USO DE ALOHANDES PARA CADA TIPO DE USUARIO DE LACOMUNIDAD**
Resumen	Se espera que se pueda distinguir cuales son los usuarios más concurrentes y propensos autilizar sus servicios. Distribuido por el tipo de miembro de la comunidad que es el usuario.
Entradas	
clientes: Set<Cliente>	
Resultados	
Se espera que se conozca el uso de alohandes para cada tipo de usuario de la comunidad (preferentemente como unadistribución porcentual):	
RNF asociados	
Distribución: Es un requerimiento que verifica la información de reservaciones de los clientes para filtrarlos correctamente, porende la centralización de la base de datos puede facilitar el funcionamiento del requisito.	

Nombre	RFC6. **- MOSTRAR EL USO DE ALOHANDES PARA UN USUARIO DADO (NÚMERO DE NOCHES O MESES CONTRATADOS, CARACTERÍSTICAS DEL ALOJAMIENTO UTILIZADO,DINERO PAGADO.**
Resumen	Se espera que se pueda distinguir las tendencias e información histórica de los clientes en laaplicación. Tipo de contrataciones, dinero gastado, tiempo alojado, etc.
Entradas	
cliente: Cliente	
Resultados	
Se espera que se conozca el uso de alohandes para un cliente específico el cual muestre sus históricos en nochescontratadas y dinero pagado. Pero también qué tipo de alojamientos son de su preferencia.	
RNF asociados	
Privacidad: Este requerimiento tiene toda la información privada de los gastos y preferencias de los clientes, por ende debeser sumamente segura y privada.	

3. Modelo Conceptual Actualizado



4. Modelo relacional actualizado

Haciendo uso de la retroalimentación recibida en la anterior iteración y tras llevar el modelo relacional a la forma normal BCNF, el modelo relacional hallado y destinado para el proyecto del curso ha sido el siguiente:

Cliente		
identificación	**nombre**	**vinculo**
PK	**NN**	**NN, CK**

Ilustración 1: Tabla Cliente - Negocio AlohAndes

Alojamiento				
ID_Aloja	**Capacidad**	**Estado**	**Direccion**	**Tipo_Aloja**
PK	**NN**	**NN**	**NN**	**NN**

Ilustración 2: Tabla Alojamiento - Negocio AlohAndes

Alojamiento_Operador	
ID_Aloja	**ID_Operador**
PK	**PK**

Ilustración 3: Tabla Alojamiento_Operador - Negocio AlohAndes

Reserva							
ID	**fecha_llegada**	**fecha_salida**	**precio**	**Cliente**	**Alojamiento**	**Operador**	**Estado**
PK, FK A.ID	**NN, CK**	**NN, CK**	**NN**	**NN,FK Cliente.ID**	**NN , FK A.ID**	**NN , FK O.ID**	**CK [Activa,Cancelada]**

Ilustración 4: Tabla Reserva - Negocio AlohAndes

Servicio	
ID	**nombre**
PK	**NN**

Ilustración 5: Tabla Servicio - Negocio AlohAndes

Operador		
ID	**nombre**	**ganancias**
PK	**NN**	**NN**

Ilustración 6: Tabla Operador - Negocio AlohAndes

Edificio_Universitario			
ID	**numViviendas**	**numDisponible**	**direccion**
PK, FK O.ID	**NN**	**NN, CK [0..numViviendas]**	**NN**

Ilustración 7: Tabla Edificio_Universitario - Negocio AlohAndes

Alojamiento_Servicio	
ID_Aloja	ID_Servicio
PK, FK A.ID	PK, FK S.Nombre

Ilustración 8: Tabla Alojamiento_Servicio - Negocio AlohAndes

Hab_Hotel	
ID_Aloja	Categoria
PK, FK A.ID	NN

Ilustración 9: Tabla Hab_Hotel - Negocio AlohAndes

Hab_Hostal		
ID_Aloja	hora_apertura	hora_cierre
PK, FK A.ID	NN	NN

Ilustración 10: Tabla Hab_Hostal - Negocio AlohAndes

Persona		
ID	Identificacion	Vinculo
PK, FK O.ID	ND	NN, CK

Ilustración 11: Tabla Persona - Negocio AlohAndes

Hotel-Hostal					
ID	EstadoLegal	umeroRegistroC	numHabitaciones	HabitacionesDispor	direccion
PK, FK O.ID	NN, CK	PK	ND, CK	ND, CK	NN

Ilustración 12: Tabla Hotel_Hostal - Negocio AlohAndes

5. Normalización del modelo

El modelo se encuentra en la forma normal BCNF, ya que todas las dependencias funcionales en el modelo son elementales, las llaves candidatas de todas las tablas son irreducibles y todas las dependencias funcionales no triviales son dependencias funcionales completas. Aparte, no hay transitividad entre las tablas.

6. Diagramas de secuencia

Requerimiento Funcional 4: Registrar una Reserva

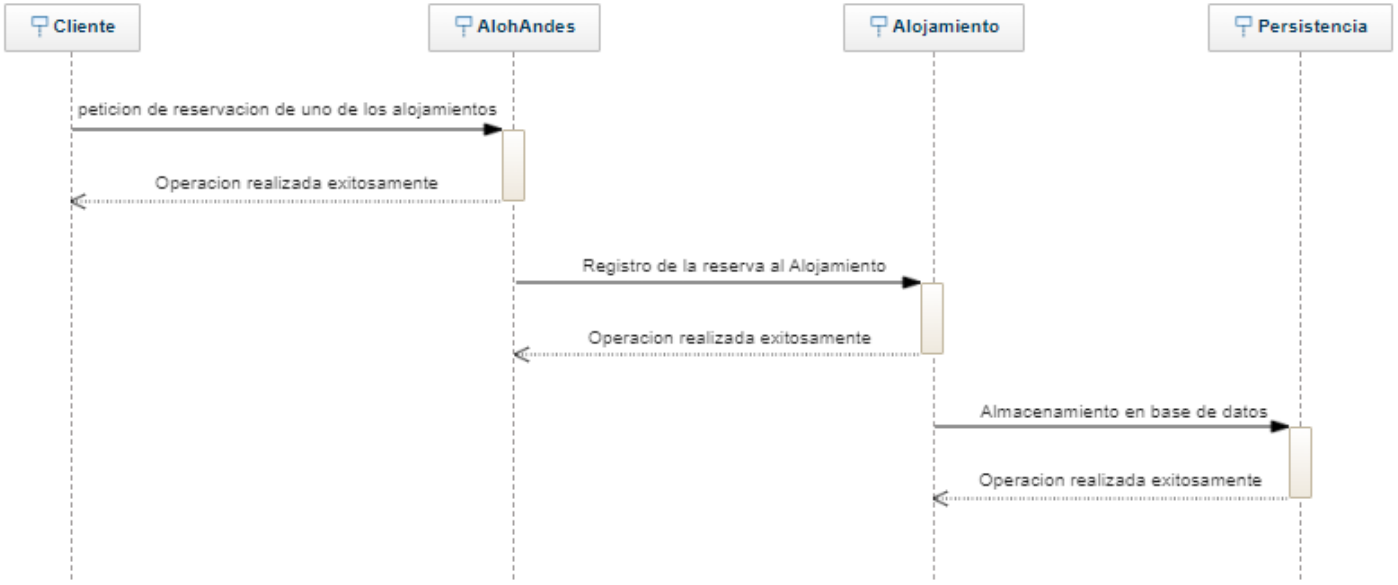


Ilustración 13: RF4 - Diagrama de Secuencia

Requerimiento Funcional 6: Retirar una oferta de alojamiento

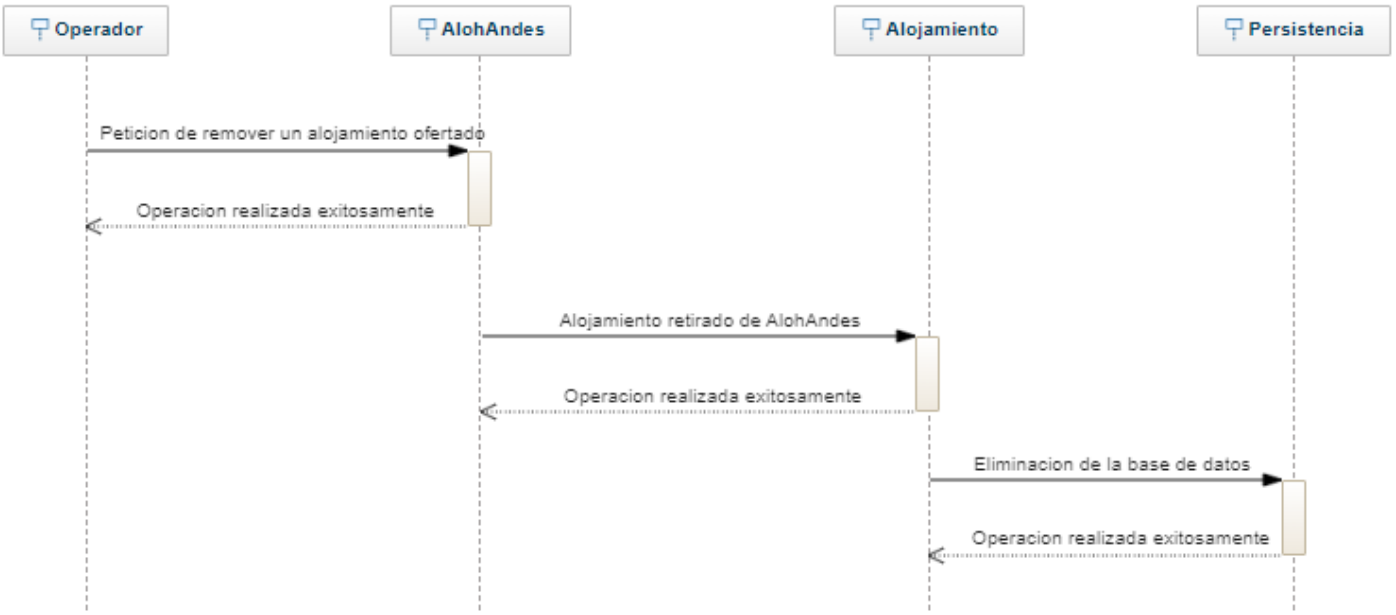


Ilustración 14: RF6 - Diagrama de Secuencia

Requerimiento Funcional de Consulta 2: MOSTRAR LAS 20 OFERTAS MÁS POPULARES

Ilustración 15: RFC2 - Diagrama de Secuencia

Requerimiento Funcional de Consulta 3: MOSTRAR EL ÍNDICE DE OCUPACIÓN DE CADA UNA DE LAS OFERTAS DE ALOJAMIENTO REGISTRADAS

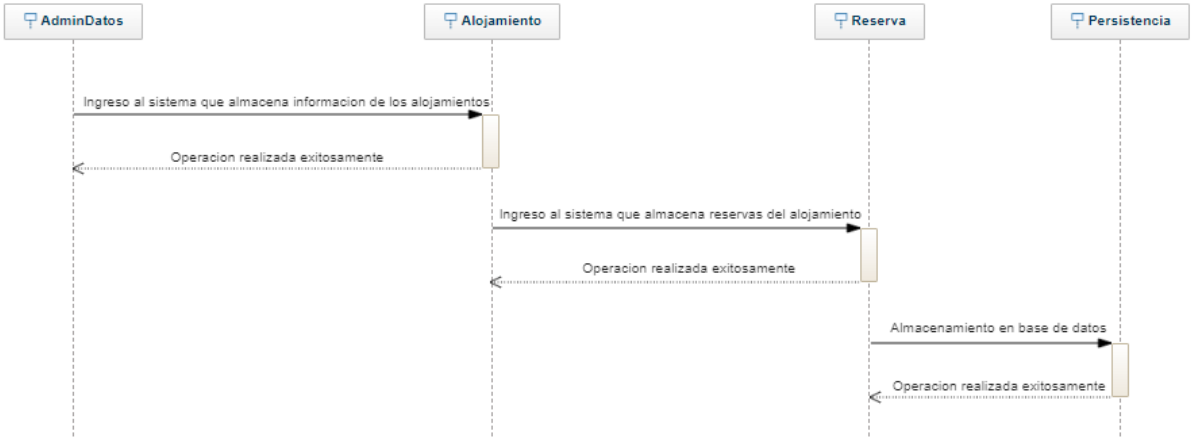


Ilustración 16: RFC3 - Diagrama de Secuencia

7. Resultados Logrados

En el desarrollo de esta iteración se realizo correctamente todo el trabajo practico esperado. El resultado del trabajo es una aplicación bien construida que se adapta perfectamente para cumplir con todos los requerimientos funcionales y requerimientos funcionales de consulta esperados. Las tablas creadas son suficientes para poder dar a conocer toda la información que el negocio requiere y se atiene correctamente a las reglas de negocio estipuladas para AlohAndes. La creación, modificación y borrado de las instancias (tuplas) se puede realizar correctamente para todas las tablas sin excepciones. De esta misma manera es evidente una preocupación por parte del equipo de trabajo en cuanto a la calidad del código entregado y las buenas practicas de desarrollo en el transcurso de la aplicación, entregando un código legible y bien comentado. Adicionalmente, se hace entrega de una base de datos sumamente centralizada lo cual favorece la distribución de la base de datos. Así mismo, es evidente suma concurrencia al realizar las operaciones solicitadas en las tablas, debido a su permisividad a tolerar la sucesión de transacciones durante su uso. Uno de los aspectos mas notables del trabajo entregado es el de la Persistencia, requerimiento no funcional en el cual se dedico bastante trabajo y organización de forma que todo en la base de datos favorezca la Persistencia de esta tras cada transacción y la Persistencia de los datos en el tiempo. Al cumplir con la mayoría de los requerimientos estipulados se puede reconocer que se lograron gran parte de los resultados previstos en la Iteración.

8. Resultados no logrados

De manera autocritica cabe también resaltar que hay algunos requerimientos o funcionalidades de la base de datos dispuesta para esta Iteración que no se cumplen de la manera esperada o en su totalidad. En primer plano, la privacidad de la base de datos no se ve tan favorecida como otros de los requerimientos no funcionales esperados en esta aplicación. No hay métodos de autenticación o restricciones de acceso a la información que demuestren una intención por ampliar la calidad de este requerimiento funcional. En segundo plano, el plan de pruebas especificado en el enunciado de la Iteración no se ve reflejado en la base de datos, ni es completamente funcional en la aplicación. Esto conlleva un alto riesgo debido a que no es posible reconocer muchos de los fallos que podría experimentar la aplicación durante su uso y manejo. Estos dos resultados fueron vistos ligeramente o como en el caso del plan de pruebas no están desarrollados en su totalidad. Únicamente para la adición, modificación y eliminación de recursos asociados a la clase Reserva.

3. Balance de plan de pruebas

9.1. Prueba de unicidad de tuplas

Tabla Cliente:

```
Error starting at line : 1 in command -
Insert into Cliente (identificacion, nombre, vinculo) values (1, 'DANIEL FERNANDEZ','ESTUDIANTE' )
Error report -
ORA-00001: unique constraint (ISIS2304C21202310.PK_CLIENTE) violated
```

Tabla Operador:

```
Error starting at line : 1 in command -
Insert into Operador (id, nombre, ganancias) values (1, 'JIMENA RODRIGUEZ',10000 )
Error report -
ORA-00001: unique constraint (ISIS2304C21202310.PK_OPERADOR) violated
```

Tabla Alojamiento:

```
Error starting at line : 1 in command -
INSERT INTO ALOJAMIENTO(ID, CAPACIDAD, ESTADO, DIRECCION, TIPO_ALOJA) VALUES (1, 1, 'DISPONIBLE', 'UNICENTRO', 'HABITACION_VIVIENDA')
Error report -
ORA-00001: unique constraint (ISIS2304C21202310.PK_ALOJAMIENTO) violated
```

Tabla Servicio:

```
Error starting at line : 1 in command -
INSERT INTO SERVICIO(ID, NOMBRE) VALUES (1, 'TINA')
Error report -
ORA-00001: unique constraint (ISIS2304C21202310.PK_SERVICIO) violated
```

Tabla Reserva:

Tabla Hab_Hotel:

```
Error starting at line : 1 in command -
INSERT INTO HAB_HOTEL(ID_ALOJA, CATEGORIA)
VALUES (1, 'SUIT')
Error report -
ORA-00001: unique constraint (ISIS2304C21202310.FK_PK_ID_HAB_HOT) violated
```

Tabla Alojamiento_Servicio:

```
Error starting at line : 1 in command -
INSERT INTO ALOJAMIENTO_SERVICIO(ID_ALOJA, ID_SERVICIO)
VALUES (1, 1)
Error report -
ORA-00001: unique constraint (ISIS2304C21202310.PK_ALOJA_SERVICIO) violated
```

Tabla Alojamiento_Operador:

```
Error starting at line : 1 in command -
INSERT INTO ALOJAMIENTO_OPERADOR(ID_ALOJA, ID_OPERADOR)
VALUES (1, 1)
Error report -
ORA-00001: unique constraint (ISIS2304C21202310.PK_ALOJA_OPERADOR) violated
```

Tabla Hotel_Hostal:

```
Error starting at line : 1 in command -
INSERT INTO HOTEL_HOSTAL(ID, ESTADO_LEGAL, NUMERO_REGISTRO_CC, NUM_HABITACIONES, habitaciones_disponibles, DIRECCION)
VALUES (1, 'VIGENTE', 1, 10, 5, 'ANDINO')
Error report -
ORA-00001: unique constraint (ISIS2304C21202310.PK_HOTEL_HOSTAL) violated
```

Tabla Edificio Universitario:

```
Error starting at line : 1 in command -
INSERT INTO EDIFICIO_UNIVERSITARIO(ID, NUMVIVIENDAS, NUMDISPONIBLE, DIRECCION)
VALUES (1, 5, 3, 'ANDINO')
Error report -
ORA-00001: unique constraint (ISIS2304C21202310.PK_EDIFICIO_UNIVERSITARIO) violated
```

Tabla Persona:

```
Error starting at line : 1 in command -
INSERT INTO PERSONA(ID, IDENTIFICACION, VINCULO)
VALUES (1, 1, 'ESTUDIANTE')
Error report -
ORA-00001: unique constraint (ISIS2304C21202310.PK_PERSONA) violated
```

4. Documentación de RF y RFC

10.1. Requerimientos Funcionales

RF7

```
tx.begin();
long tuplasInsertadas = 0;
List<Alojamiento> listAloja = sqlAlojamiento.darAlojamientosDisponiblesPorTipo(pm,fecha_llegada, fecha_salida, servicios, tipo_Aloja);
log.info("Inserción de reserva: [" + listAloja.size() + ", " + numAlojas + "]");
if (listAloja.size() >= numAlojas)
    for (int i = 0; i < numAlojas; i++) {

        Alojamiento alojamiento = listAloja.get(i);
        long Id_Res = nextval ();
        long Id_Alojamiento = alojamiento.getId();
        tuplasInsertadas += sqlReserva.adicionarReserva(pm, Id_Res, fecha_llegada, fecha_salida, precio, Id_Cliente, Id_Alojamiento, estado);

    }else
    {
        throw new Exception(message:"No hay suficientes alojamientos disponibles");
    }
    log.trace ("Inserción de reserva: [" + Id_Cliente + ", " + tuplasInsertadas + "]");

tx.commit();

return tuplasInsertadas;
```

En la imagen anterior vemos como se ve implementa el grueso de la transacción que permite cumplir con este requerimiento. La estrategia que se implementó para lograr desarrollar el requerimiento de **REGISTRAR RESERVA COLECTIVA** fue: Inicialmente pedirle al usuario los datos normales de su reserva (Fecha, Servicios que desee) y luego que indicara el tipo de alojamiento deseado y la cantidad deseada. Con esa Información se realizaba una consulta en la Bases de Datos de los Alojamiento disponibles que cumplían los requisitos para luego ver si había los suficientes Alojamientos Disponibles en esas fechas para poder hacer la reserva masiva, de lo contrario se avisaba la carencia de disponibilidad de Alojamiento para hacer la reserva. Cuando se cumplía con la capacidad se crean reservas individuales a nombre de un cliente de reserva colectiva con sus respectivos Alojamientos.

```
String servicio, String tipo_aloja )
{
    Query q = pm.newQuery(SQL, "SELECT a.id,a.capacidad,a.estado,a.direccion,a.tipo_aloja "+
    "FROM ALOJAMIENTO_SERVICIO ASER,SERVICIO S, (SELECT a.id,a.capacidad,a.estado,a.direccion,a.tipo_aloja "+
    "   FROM ALOJAMIENTO A LEFT JOIN RESERVA R "+
    "   ON a.ID = R.ID_ALOJAMIENTO "+
    "   WHERE R.fecha_llegada NOT BETWEEN TO_DATE( ?,'DD/MM/YYYY') AND TO_DATE(?,'DD/MM/YYYY') "+
    "   AND R.fecha_salida NOT BETWEEN TO_DATE(?,'DD/MM/YYYY') AND TO_DATE(?,'DD/MM/YYYY') "+
    "   OR R.fecha_salida IS NULL OR R.ESTADO = 'CANCELADA') A "+
    "WHERE aser.id_aloja = a.id AND aser.id_servicio = s.id "+
    "AND a.estado = 'DISPONIBLE' AND S.NOMBRE like ? AND a.tipo_aloja = ? "+
    "GROUP BY a.id,a.capacidad,a.estado,a.direccion,a.tipo_aloja");
```

Esta imagen muestra la sentencia en SQL que se utilizó para ver la disponibilidad de Alojamiento para reservar.

RF8

```
tx.begin();

ArrayList rta = new ArrayList<>();

List<Reserva> listarReservas = sqlReserva.darReservasPorIdCliente(pm,id_Cliente);

Date fechaActual = new Date();

SimpleDateFormat formatoFecha = new SimpleDateFormat(pattern:"dd/MM/yyyy"); // patrón de fecha

Timestamp fecha_llegada = listarReservas.get(index:0).getFecha_llegada(); // fecha en formato String

// Convertir el objeto Timestamp a una cadena de texto con el formato especificado
String Stringfecha_llegada = formatoFecha.format(fecha_llegada);
Date Datefecha_llegada = formatoFecha.parse(Stringfecha_llegada); // convertir String a Date

Timestamp fecha_salida = listarReservas.get(index:0).getFecha_Salida();; // fecha en formato String
String Stringfecha_salida = formatoFecha.format(fecha_salida);
Date Datefecha_salida = formatoFecha.parse(Stringfecha_salida); // convertir String a Date

log.info(fecha_llegada +" "+ fecha_salida);

Calendar calendar = Calendar.getInstance(); // obtener una instancia del calendario
calendar.setTime(Datefecha_llegada); // establecer la fecha actual
calendar.add(Calendar.DAY_OF_MONTH, -3); // restar tres días
Date fechaRestadatres = calendar.getTime(); // obtener la nueva fecha

double costoCancelacion = 0;
if (fechaActual.compareTo(Datefecha_llegada) > 0 && fechaActual.compareTo(Datefecha_salida) < 0 ) {
```

```
double costoCancelacion = 0;
if (fechaActual.compareTo(Datefecha_llegada) > 0 && fechaActual.compareTo(Datefecha_salida) < 0 ) {
    // La fecha está dentro del rango de la reserva
    for (Reserva reserva : listarReservas) {
        costoCancelacion += reserva.getPrecio();
        this.ActualizarReserva(estado:"CANCELADA", reserva.getId());
    }
    costoCancelacion = costoCancelacion*0.5;
}
else if(fechaActual.compareTo(fechaRestadatres) > 0 && fechaActual.compareTo(Datefecha_llegada) < 0 ){
    // La fecha está dentro de los tres dias antes de la reserva hasta la fecha
    for (Reserva reserva : listarReservas) {
        costoCancelacion += reserva.getPrecio();
        this.ActualizarReserva(estado:"CANCELADA", reserva.getId());
    }
    costoCancelacion = costoCancelacion*0.3;
}
else if (fechaActual.compareTo(fechaRestadatres) < 0) {
    // La fecha está antes de los 3 dias de lareserva
    for (Reserva reserva : listarReservas) {
        costoCancelacion += reserva.getPrecio();
        this.ActualizarReserva(estado:"CANCELADA", reserva.getId());
    }
    costoCancelacion = costoCancelacion*0.1;
}

rta.add(costoCancelacion);
rta.add(listarReservas.size());

tx.commit();

return rta;
```

En las dos Imágenes anteriores se muestra la implementación de la transacción del requerimiento. Para este Requerimiento de **CANCELAR RESERVA COLECTIVA** se opto por partir de una regla de negocio en la que cliente no puede tener varias reservas colectivas a un mismo identificador de cliente (Si una cliente quiere varias reservas colectivas, se asume como si fuera un cliente distinto). Luego de lo anterior se optó por implementar primero una sentencia que me retornara todas las reservas individuales asociadas a la reserva colectiva. Luego de tener todas las reservas colectivas, se analiza la fecha de llegada y salida de una de ellas ya que todas comparten el mismo intervalo de tiempo en la reserva, para luego comparar con la fecha actual para saber cual es costo que hay que asumir. Se recorren todas las reservas activas y se calcula su costo y se suma a una variable total. Finalmente se tiene el costo de la cancelación y todas las reservas individuales asociadas a al colectiva quedan canceladas.

```
public List<Reserva> darReservasPorIdCliente (PersistenceManager pm, long id_Cliente)
{
    Query q = pm.newQuery(SQL, "SELECT * FROM " + pp.darTablaReserva() + " WHERE ID_CLIENTE = ? ESTADO = 'ACTIVA'");
    q.setResultClass(Reserva.class);
    q.setParameters(id_Cliente);
    return (List<Reserva>) q.executeList();
}
```

A través de la sentencia SQL de la imagen se obtienen las reservas individuales asociadas a la reserva colectiva.

RF9

```
tx.begin();
ArrayList rta = new ArrayList<>();
SimpleDateFormat formatoFecha = new SimpleDateFormat(pattern:"dd/MM/yyyy");
Alojamiento Aloja = darAlojamientoPorId(id_Alojamiento);

if (Aloja.getEstado() == "DESHABILITADO")
{
    rta.add(e:0);
    rta.add(Aloja);
}
else
{
    long A = sqlAlojamiento.DeshabilitarAlojamiento(pm,estado:"DESHABILITADO", id_Alojamiento);
    List<Reserva> ReservasAlojaDes = sqlReserva.darReservasPorIdAlojamiento(pm, id_Alojamiento);
    Aloja = darAlojamientoPorId(id_Alojamiento);

    log.info(+A+"NumReservas: "+ReservasAlojaDes.size());

    int CuentaRelocalizados = 0;
    int CuentaNoRe = 0;

    if (ReservasAlojaDes.size() != 0)
    {
```

```
if (ReservasAlojaDes.size() != 0)
{
    for (int j = 0; j < ReservasAlojaDes.size(); j++) {

        Reserva res = ReservasAlojaDes.get(j);
        sqlReserva.ActulizarReserva(pm, estado:"CANCELADA", res.getId());

        List<Alojamiento> AlojasDisponibles = sqlAlojamiento.darAlojamientosRelocalizables(pm, res.getFecha_llegada(),res.getFecha_Salida(),Aloja.getCapacidad() );
        if (AlojasDisponibles.size() > 0){
            //Se intenta relocalizar
            Alojamiento AlojaDisponible = AlojasDisponibles.get(index:0);
            long id_new = nextval();

            Timestamp fecha_llegada = res.getFecha_llegada(); // fecha en formato String
            String Stringfecha_llegada = formatoFecha.format(fecha_llegada);

            Timestamp fecha_salida = res.getFecha_Salida(); // fecha en formato String
            String Stringfecha_salida = formatoFecha.format(fecha_salida);


            sqlReserva.adicionarReserva(pm, id_new, Stringfecha_llegada, Stringfecha_salida, res.getPrecio(),
                res.getId_cliente(),  AlojaDisponible.getId(),estado:"ACTIVA");

            CuentaRelocalizados += 1;
        }
        else
        {
            //No se pudo relocalizar
            CuentaNoRe +=1;
        }
```

```
                sqlReserva.adicionarReserva(pm, id_new, Stringfecha_llegada, Stringfecha_salida, res.getPrecio(),
                    res.getId_cliente(),  AlojaDisponible.getId(),estado:"ACTIVA");

                CuentaRelocalizados += 1;
            }
            else
            {
                //No se pudo relocalizar
                CuentaNoRe +=1;
            }
        }
    }
    rta.add(CuentaRelocalizados);
    rta.add(CuentaNoRe);
    rta.add(Aloja);

}
tx.commit();

return rta;
```

Las tres imágenes muestran el grueso de la transacción que se encargan de hacer posible el RF9. Para este requerimiento **DESHABILITAR OFERTA DE ALOJAMIENTO** inicialmente se le pide al usuario que indique el id del alojamiento para luego cambiar el estado de su atributo para indicar que esta deshabilitado (En caso de que ya está deshabilitado se le indicara al usuario y no hay que relocalizar). Luego de indicar que esta deshabilitado se procede a hacer la relocalización, para esto primero se realiza una consulta que permite saber las reservas que tiene el Alojamiento Deshabilitado asociadas, para saber cuáles hay que relocalizar. Segundo se recorren todas las reservas para indicar que ahora están canceladas y se consulta para cada una de estas que Alojamientos están disponibles. Si hay disponibles se relocaliza de lo contrario solo se cancela y se lleva un conteo de cuantas si se pudieron relocalizar y cuantas no. Finalmente, se le muestra al usuario los respectivos conteos.

```
Query q = pm.newQuery(SQL, "SELECT * FROM " + pp.darTablaReserva() + " WHERE ID_ALOJAMIENTO = ? AND ESTADO = 'ACTIVA' ORDER BY fecha_insercion ASC");
q.setResultClass(Reserva.class);
q.setParameters(id_Aloja);
return (List<Reserva>) q.executeList();
```

La anterior sentencia de consulta SQL me retorna las reservas asociadas al alojamiento y me las devuelve empezando por la mas antigua siendo este el orden de prioridad de relocalización.

```
Query q = pm.newQuery(SQL, "SELECT * "+
" FROM (SELECT a.id,a.capacidad,a.estado,a.direccion,a.tipo_aloja "+
"   FROM ALOJAMIENTO A LEFT JOIN RESERVA R "+
"   ON a.ID = R.ID_ALOJAMIENTO "+
"   WHERE R.fecha_llegada NOT BETWEEN ? AND ? "+
"   AND R.fecha_salida NOT BETWEEN ? AND ?  "+
"   OR R.fecha_salida IS NULL OR R.ESTADO = 'CANCELADA') N "+
" WHERE N.estado = 'DISPONIBLE' AND N.CAPACIDAD >= ?");
```

Y por último esta sentencia permite dar una lista de los alojamientos que puede reemplazar aquel que fue deshabilitado.

RF10

```
tx.begin();
String rta = "";
Alojamiento Aloja = darAlojamientoPorId(id_Alojamiento);
log.info(Aloja.getEstado().equals(anObject:"DESHABILITADO"));
if(Aloja.getEstado().equals(anObject:"DESHABILITADO"))
{
    sqlAlojamiento.DeshabilitarAlojamiento(pm, estado:"DISPONIBLE", id_Alojamiento);
    Aloja = darAlojamientoPorId(id_Alojamiento);
    rta = Aloja.toString();
}

tx.commit();
log.info(rta);
```

Para este requerimiento de REHABILITAR OFERTA DE ALOJAMIENTO vemos como se implementó la transacción en la imagen anterior. La estrategia fue sencilla ya que se tenia un atributo que indica que si el Alojamiento esta deshabilitado o Habilitado. Así que básicamente la transacción consiste en ver cual es el estado del Alojamiento y si es Deshabilitado lo habilita, de lo contrario se le indica al usuario que ya estaba habilitado.

10.2. Requerimientos Funcionales de Consulta

RFC5

```
SELECT VINCULO, COUNT(*) AS CANTIDAD_DE_USUARIOS
FROM (
    SELECT VINCULO FROM PERSONA
    UNION ALL
    SELECT VINCULO FROM CLIENTE
)
GROUP BY VINCULO;
```

Para este requerimiento funcional de consulta se esperaba crear una vista en la que se muestre el uso de Alohandes para cada tipo de usuario. Para ello, era fundamental separar a los usuarios en los tipos de usuario posible, los cuales se encuentran en el set de: 'ESTUDIANTE', 'PROFESOR', 'EMPLEADO', 'PADRE' y 'EGRESADO'. Estos tipos de usuario se encuentran almacenados en la variable VINCULO de las tablas PERSONA y CLIENTE. Mediante un UNION ALL, conseguimos a todos los usuarios de Alohandes que tienen algún vínculo con la universidad y los agrupamos con un GROUP BY de la variable clave VINCULO.

RFC6:

```
SELECT
    c.identificacion,
    TRUNC(MONTHS_BETWEEN(r.fecha_salida, r.fecha_llegada) * 30) AS dias_contratados,
    a.tipo_aloja,
    SUM(r.precio) AS dinero_pagado
FROM
    cliente c
    INNER JOIN reserva r ON c.identificacion = r.id_cliente
    INNER JOIN alojamiento a ON r.id_alojamiento = a.id
WHERE
    c.identificacion = 1
GROUP BY
    c.identificacion, a.tipo_aloja, TRUNC(MONTHS_BETWEEN(r.fecha_salida, r.fecha_llegada) * 30);
```

Para este requerimiento funcional de consulta se esperaba crear una vista que mostrara el uso de Alohandes para un usuario dado. Para ello, se realiza un JOIN entre las tablas RESERVA, ALOJAMIENTO Y CLIENTE. De forma que se puedan extraer los atributos importantes sobre el Tipo de Alojamiento, el ID del Cliente y El número de noches contratadas. De esta forma se recopila la información de reservas sobre el cliente. La función TRUNC retorna el número de meses entre las fechas de una reserva, que al ser multiplicado por 30 retorna el número de días.

RFC7:

```
(SELECT fecha_llegada AS fecha,
       COUNT(*) AS alojamientos_ocupados,
       SUM(PRECIO) AS ingresos_totales
FROM reserva
JOIN alojamiento ON reserva.id_alojamiento = alojamiento.id
WHERE alojamiento.tipo_aloja = 'tipo_de_alojamiento'
  AND fecha_llegada BETWEEN 'fecha_inicio' AND 'fecha_fin'
GROUP BY fecha_llegada
ORDER BY alojamientos_ocupados DESC, ingresos_totales DESC
FETCH FIRST 1 ROWS ONLY)

UNION

(SELECT fecha_llegada AS fecha,
       COUNT(*) AS alojamientos_ocupados,
       SUM(PRECIO) AS ingresos_totales
FROM reserva
JOIN alojamiento ON reserva.id_alojamiento = alojamiento.id
WHERE alojamiento.tipo_aloja = 'tipo_de_alojamiento'
  AND fecha_llegada BETWEEN 'fecha_inicio' AND 'fecha_fin'
GROUP BY fecha_llegada
ORDER BY alojamientos_ocupados ASC, ingresos_totales ASC
FETCH FIRST 1 ROWS ONLY)

UNION

(SELECT fecha_llegada AS fecha,
       COUNT(*) AS alojamientos_ocupados,
       SUM(PRECIO) AS ingresos_totales
FROM reserva
JOIN alojamiento ON reserva.id_alojamiento = alojamiento.id
WHERE alojamiento.tipo_aloja = 'tipo_de_alojamiento'
  AND fecha_llegada BETWEEN 'fecha_inicio' AND 'fecha_fin'
GROUP BY fecha_llegada
ORDER BY ingresos_totales DESC
FETCH FIRST 1 ROWS ONLY);
```

Para este requerimiento funcional de consulta se esperaba crear una vista que mostrara para una unidad de tiempo definida y para un alojamiento en específico los días de mayor demanda, los de

mayor ingreso y los de menor ocupación. Para ello creamos tres funciones distintas que están unidas a través de un JOIN, cada una para uno de los requerimientos mencionados. Cabe resaltar que 'tipo_de_alojamiento' es el tipo de alojamiento seleccionado para la consulta y que 'fecha_inicio' y 'fecha_fin' también son inputs.

RFC 8:

```
SELECT C.IDENTIFICACION, COUNT(*) AS NUM_RESERVAS, SUM(TRUNC(MONTHS_BETWEEN(R.FECHA_SALIDA, R.FECHA_LLEGADA)*30)) AS NUM_NOCHES
FROM RESERVA R
INNER JOIN CLIENTE C ON R.ID_CLIENTE = C.IDENTIFICACION
WHERE R.ID_ALOJAMIENTO = 1
GROUP BY C.IDENTIFICACION
HAVING COUNT(*) >= 3 OR SUM(TRUNC(MONTHS_BETWEEN(R.FECHA_SALIDA, R.FECHA_LLEGADA)*30)) >= 15
ORDER BY NUM_RESERVAS DESC;
```

Para este RFC se buscaba crear una vista en la cual se detallarán los clientes más frecuentes de un alojamiento dado. Un cliente frecuente es aquel que ha reservado el alojamiento en 3 diferentes ocasiones o aquel que se haya hospedado en este 15 noches. Para ello creamos un INNER JOIN con las reservas del alojamiento con aquel ID en específico. Para posteriormente contar el número de veces que se ha hospedado o si alguna de las reservas fue superior a 30 dias.

RFC 9:

```
SELECT A.ID, A.TIPO_ALOJA
FROM ALOJAMIENTO A
WHERE NOT EXISTS (
    SELECT *
    FROM RESERVA R
    WHERE R.ID_ALOJAMIENTO = A.ID
    AND R.FECHA_LLEGADA >= (SELECT MAX(FECHA_LLEGADA) FROM RESERVA WHERE ID_ALOJAMIENTO = A.ID) - INTERVAL '1' MONTH);
```

Para este requerimiento funcional se creó una vista que muestra a los alojamientos sin reservas durante un mes. Para ello se hizo un NOT EXISTS de la tabla reserva con intervalo de 1 mes sobre la fecha de llegada.

RFC 10

```
--RFC 10:

SELECT CLIENTE.*
FROM CLIENTE, RESERVA
WHERE CLIENTE.IDENTIFICACION = RESERVA.ID_CLIENTE
    AND RESERVA.FECHA_LLEGADA >= TO_DATE('01/05/2023','DD/MM/YYYY')
    AND RESERVA.FECHA_SALIDA <= TO_DATE('01/12/2023','DD/MM/YYYY')
    AND RESERVA.ID_ALOJAMIENTO = 1
GROUP BY CLIENTE.IDENTIFICACION, CLIENTE.NOMBRE, CLIENTE.VINCULO
;
--
SELECT C.*
FROM CLIENTE C, RESERVA R, ALOJAMIENTO A
WHERE C.IDENTIFICACION = R.ID_CLIENTE AND A.ID = R.ID_ALOJAMIENTO
    AND R.FECHA_LLEGADA >= TO_DATE('01/05/2023','DD/MM/YYYY')
    AND R.FECHA_SALIDA <= TO_DATE('01/12/2023','DD/MM/YYYY')
    AND A.TIPO_ALOJA = 'HABITACION_HOSTAL'
GROUP BY C.IDENTIFICACION, C.NOMBRE, C.VINCULO
;
```

Para el requerimiento funcional de consulta 10 que pedía que se diera la información sobre los usuarios que realizaron al menos una reserva de una determinada oferta de alojamiento en un rango de fechas. Con la finalidad de cumplir con este requerimiento se comienza por realizar una conexión entre las

tablas cliente y reserva, posteriormente se confirma que se encuentre en el rango de fechas seleccionado, para finalmente confirmar que la reserva pertenece al alojamiento con el id deseado. Esto retornaría el id, nombre y vinculo de los clientes que reservaron el alojamiento seleccionado durante las fechas seleccionadas.

En segundo lugar, se modifica la sentencia ligeramente de forma que se retorne los clientes que hayan realizado reservas en un tipo de alojamiento específico para así cumplir con el requisito de que se pueda consultar basado en diferentes atributos de la tabla alojamiento. En este caso sería en base al tipo de alojamiento y retornaría las mismas columnas que la consulta anterior.

RFC 11

```
--RFC 11:
SELECT C.IDENTIFICACION, C.NOMBRE, C.VINCULO
FROM CLIENTE C
WHERE C.IDENTIFICACION NOT IN (
  SELECT R.ID_CLIENTE
  FROM RESERVA R, ALOJAMIENTO A
  WHERE R.ID_ALOJAMIENTO = A.ID
  AND A.TIPO_ALOJA = 'VIVIENDA_U'
  AND R.FECHA_LLEGADA >= TO_DATE('01/05/2023','DD/MM/YYYY')
  AND R.FECHA_SALIDA <= TO_DATE('01/12/2023','DD/MM/YYYY')
)
ORDER BY C.IDENTIFICACION, C.VINCULO;
--
```

El requerimiento funcional de consulta número 11 ha de retornar lo contrario del RFC10, los usuarios que no han reservado en dicho rango de fechas. Por ende, su implementacion es muy similar a la del RFC10. Una vez más se selecciona el id del cliente en la reserva y se confirma que se encuentre3 una reserva entre el rango de fechas seleccionado. El cambio se encuentra en la cláusula de: WHERE C. IDENTIFICACION NOT IT, con el uso de esta negación se extraen únicamente los clientes que no hacen parte del grupo de clientes que han reservado en los alojamientos de interés durante las fechas seleccionadas.

RFC 12

```
--RFC 12:
SELECT
    O1.SEMANA,
    O1.ID_ALOJAMIENTO AS OFERTA_MAYOR_OCUPACION,
    O2.ID_ALOJAMIENTO AS OFERTA_MENOR_OCUPACION,
    OP1.NOMBRE AS OPERADOR_MAS_SOLICITADO,
    OP2.NOMBRE AS OPERADOR_MENOS_SOLICITADO
FROM
    (
        SELECT SEMANA, MAX(NUM_RESERVAS) AS MAX_OCUPACION
        FROM
            (
                SELECT
                    ID_ALOJAMIENTO,
                    TO_CHAR(FECHA_LLEGADA, 'IYYY-IW') AS SEMANA,
                    COUNT(*) AS NUM_RESERVAS
                FROM
                    RESERVA
                WHERE
```

```
                    RESERVA
                WHERE
                    ESTADO = 'ACTIVA'
                GROUP BY
                    ID_ALOJAMIENTO,
                    TO_CHAR(FECHA_LLEGADA, 'IYYY-IW')
            ) ocupacion_semanal
        GROUP BY SEMANA
    ) MAX_OCUPACION_SEMANAL
INNER JOIN
    (
        SELECT
            ID_ALOJAMIENTO,
            TO_CHAR(FECHA_LLEGADA, 'IYYY-IW') AS SEMANA,
            COUNT(*) AS NUM_RESERVAS
        FROM
            RESERVA
        WHERE
            ESTADO = 'ACTIVA'
        GROUP BY
            ID_ALOJAMIENTO,
            TO_CHAR(FECHA_LLEGADA, 'IYYY-IW')
```

```
            TO_CHAR(FECHA_LLEGADA, 'IYYY-IW')
    ) O1
    ON MAX_OCUPACION_SEMANAL.SEMANA = O1.SEMANA
    AND MAX_OCUPACION_SEMANAL.MAX_OCUPACION = O1.NUM_RESERVAS
INNER JOIN
    (
        SELECT SEMANA, MIN(NUM_RESERVAS) AS MIN_OCUPACION
        FROM
            (
                SELECT
                    ID_ALOJAMIENTO,
                    TO_CHAR(FECHA_LLEGADA, 'IYYY-IW') AS SEMANA,
                    COUNT(*) AS NUM_RESERVAS
                FROM
                    RESERVA
                WHERE
                    ESTADO = 'ACTIVA'
                GROUP BY
                    ID_ALOJAMIENTO,
                    TO_CHAR(FECHA_LLEGADA, 'IYYY-IW')
            ) ocupacion_semanal
        GROUP BY SEMANA
```

```
        GROUP BY SEMANA
    ) MIN_OCUPACION_SEMANAL
    ON O1.SEMANA = MIN_OCUPACION_SEMANAL.SEMANA
INNER JOIN
    (
        SELECT
            ID_ALOJAMIENTO,
            TO_CHAR(FECHA_LLEGADA, 'IYYY-IW') AS SEMANA,
            COUNT(*) AS NUM_RESERVAS
        FROM
            RESERVA
        WHERE
            ESTADO = 'ACTIVA'
        GROUP BY
            ID_ALOJAMIENTO,
            TO_CHAR(FECHA_LLEGADA, 'IYYY-IW')
    ) O2
    ON MIN_OCUPACION_SEMANAL.SEMANA = O2.SEMANA
    AND MIN_OCUPACION_SEMANAL.MIN_OCUPACION = O2.NUM_RESERVAS
INNER JOIN
    (
        SELECT
```

```
        SELECT
            AO.ID_OPERADOR,
            COUNT(*) AS NUM_SOLICITUDES
        FROM
            ALOJAMIENTO_OPERADOR AO
            INNER JOIN RESERVA R ON AO.ID_ALOJA = R.ID_ALOJAMIENTO
        WHERE
            R.ESTADO = 'ACTIVA'
        GROUP BY
            AO.ID_OPERADOR
    ) SO1
    ON O1.ID_ALOJAMIENTO = SO1.ID_OPERADOR
INNER JOIN
    (
        SELECT
            AO.ID_OPERADOR,
            COUNT(*) AS NUM_SOLICITUDES
        FROM
            ALOJAMIENTO_OPERADOR AO
            INNER JOIN RESERVA R ON AO.ID_ALOJA = R.ID_ALOJAMIENTO
        WHERE
            R.ESTADO = 'ACTIVA'
```

```
            R.ESTADO = 'ACTIVA'
        GROUP BY
            AO.ID_OPERADOR
    ) SO2
    ON O2.ID_ALOJAMIENTO = SO2.ID_OPERADOR
INNER JOIN OPERADOR OP1 ON SO1.ID_OPERADOR = OP1.ID
INNER JOIN OPERADOR OP2 ON SO2.ID_OPERADOR = OP2.ID;
```

Con el fin de resolver los requerimientos de este requerimiento funcional de consulta que nos pide retornar para cada semana de un año dado los alojamientos de mayor/menor ocupacion y los operadores más/menos solicitados. Comenzamos por declarar las variables que representaran a la semana y a los alojamientos de mayor/menor ocupacion y los operadores más/menos solicitados. Después, se declaran 3 INNER JOIN los cuales cumplen la función de unir las 4 consultas que retornan el resultado a cada una de estas preguntas utilizando un contador global asociado a cada alojamiento u operador para la semana seleccionada. Finalmente se realiza la unión de los resultados de las 4 consultas independientes unidas a través de los INNER JOIN, para finalmente retornar cada semana y el valor de id que corresponde al alojamiento/operador de interés en cada caso.

RFC 13

```
SELECT DISTINCT C.IDENTIFICACION, C.NOMBRE, C.VINCULO,
       CASE WHEN MONTLY_RESERVAS.ID_CLIENTE IS NOT NULL THEN 'Realiza reservas mensuales' ELSE NULL END AS Calificacion_Reservas_Mensuales,
       CASE WHEN COSTOSO_RESERVAS.ID_CLIENTE IS NOT NULL THEN 'Reserva alojamientos costosos' ELSE NULL END AS Calificacion_Reservas_Costosas,
       CASE WHEN SUITE_RESERVAS.ID_CLIENTE IS NOT NULL THEN 'Reserva suites' ELSE NULL END AS Calificacion_Reservas_Suites
FROM CLIENTE C
LEFT JOIN (
    SELECT ID_CLIENTE
    FROM (
        SELECT ID_CLIENTE, EXTRACT(MONTH FROM FECHA_LLEGADA) AS MES, COUNT(*) AS NUM_RESERVAS
        FROM RESERVA
        GROUP BY ID_CLIENTE, EXTRACT(MONTH FROM FECHA_LLEGADA)
    ) RESERVAS_MENSUALES
    WHERE NUM_RESERVAS > 0
) MONTLY_RESERVAS ON C.IDENTIFICACION = MONTLY_RESERVAS.ID_CLIENTE
LEFT JOIN (
    SELECT ID_CLIENTE
    FROM (
        SELECT ID_CLIENTE, COUNT(*) AS NUM_RESERVAS_COSTOSAS
        FROM RESERVA
        WHERE PRECIO > 150
        GROUP BY ID_CLIENTE
    ) RESERVAS_COSTOSAS
```

```
) COSTOSO_RESERVAS ON C.IDENTIFICACION = COSTOSO_RESERVAS.ID_CLIENTE
LEFT JOIN (
    SELECT ID_CLIENTE
    FROM (
        SELECT ID_CLIENTE, COUNT(*) AS NUM_RESERVAS_SUITE
        FROM RESERVA R
        JOIN HAB_HOTEL HH ON R.ID_ALOJAMIENTO = HH.ID_ALOJA
        WHERE HH.CATEGORIA = 'SUITE'
        GROUP BY ID_CLIENTE
    ) RESERVAS_SUITE
) SUITE_RESERVAS ON C.IDENTIFICACION = SUITE_RESERVAS.ID_CLIENTE
WHERE MONTLY_RESERVAS.ID_CLIENTE IS NOT NULL
    OR COSTOSO_RESERVAS.ID_CLIENTE IS NOT NULL
    OR SUITE_RESERVAS.ID_CLIENTE IS NOT NULL
;
```

El requerimiento de consulta mostrado en las imágenes ha de retornar a los buenos clientes de AlohAndes, se reconoce a un cliente como bueno si cumple con alguna de las 3 condiciones: reserva alojamientos costosos (+150USD), reserva al menos una vez al mes y reserva habitaciones SUITE. Para cumplir con dichos requisitos se comenzó creando las 3 condiciones y manejando su clasificación al cumplirse o no. Posteriormente se realiza un LEFT JOIN entre las tablas cliente y reservas para tener la información sobre las reservas del cliente y se confirma que sean mayor a 0 (al menos una reserva por mes). Después se realiza otro LEFT JOIN para la segunda condición uniendo las tablas de CLIENTE y RESERVA una vez más y confirmando que el precio de las reservas sea superior a 150 USD, agrupando por el id del cliente y contando el número de reservas que cumplen con la condición con la variable NUM_RESERVAS_COSTOSAS. Finalmente, para confirmar la última condición se hace otro LEFT JOIN entre CLIENTE y RESERVA, esta vez se confirma que la categoría del alojamiento que se reserva sea SUITE y esto se almacena en la variable NUM_RESERVAS_SUITE. Finalmente se confirma que el cliente con el id x cumpla con una de las tres condiciones o no para ver si hace parte del grupo de clientes que son buenos.

10. 3 Resultados esperados RFC:

Cliente:

Error:

```
20:57:38,470 (AWT-EventQueue-0) INFO  [uniandes.isis2304.parranderos.negocio.Alohandes] - Adicionando Cliente:
20:57:38,724 (AWT-EventQueue-0) ERROR [uniandes.isis2304.parranderos.persistencia.PersistenciaAlohandes] - Exception : Error executing SQL query "INSERT INTO
CLIENTE(identificacion, nombre, vinculo) values (?, ?, ?)".
ORA-01400: cannot insert NULL into ("ISIS2304C32202310"."CLIENTE"."NOMBRE")
```

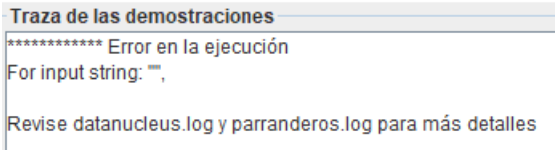
Operación Exitosa:

```
21:00:55,069 (AWT-EventQueue-0) INFO  [uniandes.isis2304.parranderos.negocio.Alohandes] - Adicionando Cliente: Felipe Nunez
21:00:55,122 (AWT-EventQueue-0) INFO  [uniandes.isis2304.parranderos.negocio.Alohandes] - Adicionando Cliente:Felipe Nunez
```

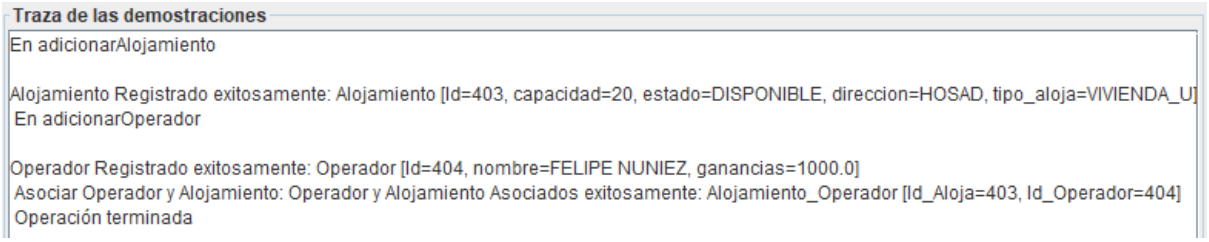
Alojamiento/Operador:

-Registrar Alojamiento nuevo con operador nuevo

Error:

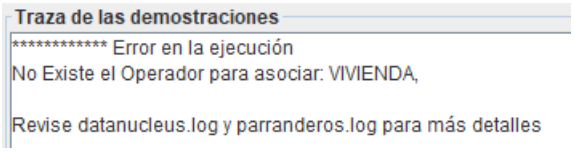


Operación Exitosa:

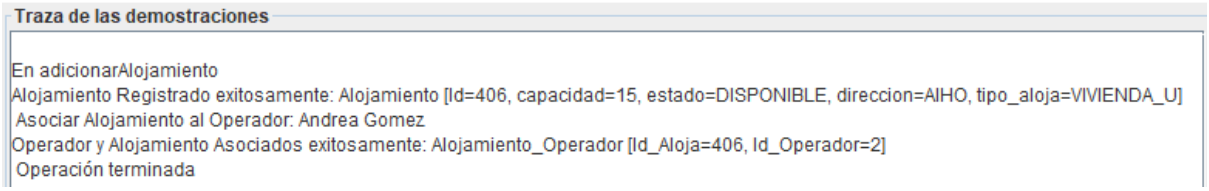


-Registrar Alojamiento nuevo con operador existente

Error:



Operación exitosa:



-Retirar oferta de alojamiento:

Error:

```
21:12:08,195 (AWT-EventQueue-0) INFO  [uniandes.isis2304.parranderos.negocio.Alohandes] - Eliminando Reserva
21:12:08,479 (AWT-EventQueue-0) INFO  [uniandes.isis2304.parranderos.negocio.Alohandes] - Eliminando Reserva: 0 tuplas eliminadas
21:12:08,479 (AWT-EventQueue-0) INFO  [uniandes.isis2304.parranderos.negocio.Alohandes] - Eliminando Alojamiento_Operador
21:12:08,495 (AWT-EventQueue-0) INFO  [uniandes.isis2304.parranderos.negocio.Alohandes] - Eliminando Alojamiento_Operador: 0tuplas eliminadas
21:12:08,495 (AWT-EventQueue-0) INFO  [uniandes.isis2304.parranderos.negocio.Alohandes] - Eliminando Alojamiento por id: 165485624
21:12:08,512 (AWT-EventQueue-0) INFO  [uniandes.isis2304.parranderos.negocio.Alohandes] - Eliminando Alojamiento por Id: 0 tuplas eliminadas
```

Operación Exitosa:

```
21:11:37,382 (AWT-EventQueue-0) INFO  [uniandes.isis2304.parranderos.negocio.Alohandes] - Eliminando Alojamiento por Id: -1 tuplas eliminadas
```

Reserva:

-Registrar Reserva:

Error:

```
21:14:23,580 (AWT-EventQueue-0) INFO  [uniandes.isis2304.parranderos.negocio.Alohandes] - Consultando Cliente por ID
21:14:23,603 (AWT-EventQueue-0) INFO  [uniandes.isis2304.parranderos.negocio.Alohandes] - Consultando Cliente por ID: Cliente [Identificacion=1, nombre=Jeronimo Vargas,
vinculo=ESTUDIANTE] existentes
21:14:23,603 (AWT-EventQueue-0) INFO  [uniandes.isis2304.parranderos.negocio.Alohandes] - Consultando Operadores
21:14:23,624 (AWT-EventQueue-0) INFO  [uniandes.isis2304.parranderos.negocio.Alohandes] - Consultando Operadores: Operador [Id=1, nombre=Jimena Garcia, ganancias=10000.0]
bebidas existentes
21:14:23,624 (AWT-EventQueue-0) INFO  [uniandes.isis2304.parranderos.negocio.Alohandes] - Adicionando Reserva [1, 1, 1000.0]
21:14:23,653 (AWT-EventQueue-0) ERROR [uniandes.isis2304.parranderos.persistencia.PersistenciaAlohandes] - Exception : Error executing SQL query "INSERT INTO RESERVA(Id,
fecha_llegada, fecha_salida, precio, Id_Cliente, Id_Alojamiento, Id_Operador, estado) values (?, TO_DATE(?, 'DD/MM/YYYY'), TO_DATE(?, 'DD/MM/YYYY'), ?, ?, ?, ?, ?)".
ORA-00904: "ID_OPERADOR": invalid identifier

21:14:23,663 (AWT-EventQueue-0) INFO  [uniandes.isis2304.parranderos.negocio.Alohandes] - Adicionando Reserva: null tuplas insertadas
```

Operación Exitosa:

-Cancelar Reserva:

Error:

```
21:17:22,926 (AWT-EventQueue-0) INFO  [uniandes.isis2304.parranderos.negocio.Alohandes] - Adicionando Reserva: null tuplas insertadas
21:17:53,367 (AWT-EventQueue-0) INFO  [uniandes.isis2304.parranderos.negocio.Alohandes] - Actualizando Reserva
21:17:53,413 (AWT-EventQueue-0) INFO  [uniandes.isis2304.parranderos.negocio.Alohandes] - Actualizando Reserva: 0 tuplas actualizadas
```

11 Diseño Físico

11.1 Documentación Diseño Físico

Para optimizar la consulta RFC 10, 11 y 13 que busca información de clientes basada en ciertas condiciones de reserva, se pueden crear los siguientes índices:

1. Índice en la columna **IDENTIFICACION** de la tabla **CLIENTE**:
    - Tipo de índice: Índice primario o único (B+ o similar).
    - Justificación: Este índice permitirá una búsqueda eficiente de clientes por su identificación. Dado que la consulta realiza una comparación en la columna **IDENTIFICACION**, un índice en esta columna mejorará el rendimiento de la búsqueda y la unión con la tabla **RESERVA**.
2. Índice en la columna **ID_CLIENTE** de la tabla **RESERVA**:
    - Tipo de índice: Índice secundario (B+ o similar).
    - Justificación: Como la consulta realiza una comparación y una unión basada en el **ID_CLIENTE**, un índice en esta columna agilizará la búsqueda y la unión entre las tablas **CLIENTE** y **RESERVA**.
3. Índice en la columna **FECHA_LLEGADA** de la tabla **RESERVA**:
    - Tipo de índice: Índice secundario (B+ o similar).
    - Justificación: La consulta filtra las reservas basándose en la fecha de llegada. Un índice en la columna **FECHA_LLEGADA** mejorará el rendimiento de las consultas que realizan comparaciones y restricciones basadas en esta columna.
4. Índice en la columna **FECHA_SALIDA** de la tabla **RESERVA**:

- Tipo de índice: Índice secundario (B+ o similar).
- Justificación: Similar al índice en **FECHA_LLEGADA**, un índice en la columna **FECHA_SALIDA** optimizará las consultas que filtran por esta condición.

5. Índice en la columna **ID_ALOJAMIENTO** de la tabla **RESERVA**:
    - Tipo de índice: Índice secundario (B+ o similar).
    - Justificación: Dado que la consulta incluye una restricción en **ID_ALOJAMIENTO**, un índice en esta columna mejorará la búsqueda y la unión con la tabla **ALOJAMIENTO**.
6. Índice en la columna **TIPO_ALOJA** de la tabla **ALOJAMIENTO**:
    - Tipo de índice: Índice secundario (B+ o similar).
    - Justificación: La consulta filtra los alojamientos por el tipo de alojamiento (**HABITACION_HOTEL**). Un índice en la columna **TIPO_ALOJA** facilitará la búsqueda de alojamientos que coincidan con esta condición.
7. Índice en la columna PRECIO de la tabla RESERVA:
    - Tipo de índice: Índice secundario (B+ o similar).
    - Justificación: La subconsulta que cuenta las reservas costosas utiliza la columna **PRECIO** para filtrar las reservas con un precio mayor a 150. Un índice en esta columna mejorará el rendimiento de la búsqueda y el filtrado de las reservas costosas.

A continuación, se justifica la selección de índices desde el punto de vista de cada uno de los requerimientos funcionales mencionados, teniendo en cuenta el tipo de índice utilizado y el costo asociado.

Búsqueda eficiente por estado de reserva y fecha de llegada (RFC12):

1. Búsqueda eficiente por estado de reserva y fecha de llegada:
    - Índice utilizado: Índice compuesto (B+ o similar) en las columnas **ESTADO** y **FECHA_LLEGADA** de la tabla **RESERVA**.
    - Justificación: Este índice permite una búsqueda rápida y eficiente de registros de reserva según el estado y la fecha de llegada. El índice compuesto combina ambas columnas en una estructura de índice, lo que mejora el rendimiento de las consultas que filtran por estas condiciones. El costo de almacenamiento y mantenimiento asociado a este índice es mayor debido a su naturaleza compuesta, pero se justifica por la mejora en el rendimiento de las consultas de búsqueda.
2. Unión eficiente entre las tablas **ALOJAMIENTO_OPERADOR** y **RESERVA** por ID de alojamiento:
    - Índice utilizado: Índice (B+ o similar) en la columna **ID_ALOJA** de la tabla **ALOJAMIENTO_OPERADOR**.
    - Justificación: Al crear un índice en la columna **ID_ALOJA**, se agiliza la búsqueda y unión de registros entre las tablas **ALOJAMIENTO_OPERADOR** y **RESERVA** basándose en el ID de alojamiento. Esto es especialmente útil para la consulta que involucra la unión de estas tablas en la cláusula **JOIN**. El costo de almacenamiento y mantenimiento asociado a este índice es moderado, ya que es un índice secundario adicional que se debe mantener.

	OWNER	INDEX_NAME	INDEX_TYPE	TABLE_OWNER	TABLE_NAME	TABLE_TYPE	UNIQUENESS
1	ISIS2304C21202310	PK_ALOJAMIENTO	NORMAL	ISIS2304C21202310	ALOJAMIENTO	TABLE	UNIQUE
2	ISIS2304C21202310	IDX_ALOJAMIENTO_OPERADOR_ID_ALOJA	NORMAL	ISIS2304C21202310	ALOJAMIENTO_OPERADOR	TABLE	NONUNIQUE
3	ISIS2304C21202310	PK_ALOJA_OPERADOR	NORMAL	ISIS2304C21202310	ALOJAMIENTO_OPERADOR	TABLE	UNIQUE
4	ISIS2304C21202310	PK_ALOJA_SERVICIO	NORMAL	ISIS2304C21202310	ALOJAMIENTO_SERVICIO	TABLE	UNIQUE
5	ISIS2304C21202310	PK_CLIENTE	NORMAL	ISIS2304C21202310	CLIENTE	TABLE	UNIQUE
6	ISIS2304C21202310	PK_EDIFICIO_UNIVERSITARIO	NORMAL	ISIS2304C21202310	EDIFICIO_UNIVERSITARIO	TABLE	UNIQUE
7	ISIS2304C21202310	FK_PK_ID_HAB_HOS	NORMAL	ISIS2304C21202310	HAB_HOSTAL	TABLE	UNIQUE
8	ISIS2304C21202310	FK_PK_ID_HAB_HOT	NORMAL	ISIS2304C21202310	HAB_HOTEL	TABLE	UNIQUE
9	ISIS2304C21202310	PK_HOTEL_HOSTAL	NORMAL	ISIS2304C21202310	HOTEL_HOSTAL	TABLE	UNIQUE
10	ISIS2304C21202310	PK_OPERADOR	NORMAL	ISIS2304C21202310	OPERADOR	TABLE	UNIQUE
11	ISIS2304C21202310	PK_PERSONA	NORMAL	ISIS2304C21202310	PERSONA	TABLE	UNIQUE
12	ISIS2304C21202310	ID_RESERVA	NORMAL	ISIS2304C21202310	RESERVA	TABLE	UNIQUE
13	ISIS2304C21202310	IDX_RESERVA_ESTADO_FECHA_LLEGADA	NORMAL	ISIS2304C21202310	RESERVA	TABLE	NONUNIQUE
14	ISIS2304C21202310	PK_SERVICIO	NORMAL	ISIS2304C21202310	SERVICIO	TABLE	UNIQUE

1. **PK_ALOJAMIENTO**:
    - Tipo de índice: PK (Primary Key) - Índice primario.
    - Tabla asociada: ALOJAMIENTO.
    - Propósito: Creado por Oracle para respaldar la clave primaria de la tabla ALOJAMIENTO.
    - Contribución al rendimiento: Ayuda a la eficiencia en las operaciones que involucran búsquedas por la clave primaria de la tabla ALOJAMIENTO, como las consultas de actualización y eliminación basadas en la clave primaria.
2. **IDX_ALOJAMIENTO_OPERADOR_ID_ALOJA**:
    - Tipo de índice: NORMAL - Índice no único.
    - Tabla asociada: ALOJAMIENTO_OPERADOR.
    - Propósito: Creado por Oracle para respaldar las consultas que involucran la columna ID_ALOJA de la tabla ALOJAMIENTO_OPERADOR.
    - Contribución al rendimiento: Mejora la eficiencia en las operaciones de búsqueda y unión basadas en la columna ID_ALOJA de la tabla ALOJAMIENTO_OPERADOR.
3. **PK_ALOJA_OPERADOR**, **PK_ALOJA_SERVICIO**, **PK_CLIENTE**, **PK_EDIFICIO_UNIVERSITARIO**, **PK_HOTEL_HOSTAL**, **PK_OPERADOR**, **PK_PERSONA**, **ID_RESERVA**, **PK_SERVICIO**:
    - Tipo de índice: PK (Primary Key) - Índice primario.
    - Propósito: Creados por Oracle para respaldar las claves primarias de las respectivas tablas.
    - Contribución al rendimiento: Ayudan a la eficiencia en las operaciones que involucran búsquedas por las claves primarias de las tablas asociadas.
4. **FK_PK_ID_HAB_HOS**, **FK_PK_ID_HAB_HOT**:
    - Tipo de índice: NORMAL - Índice no único.
    - Propósito: Creados por Oracle para respaldar las restricciones de clave externa (FK) de las respectivas tablas.
    - Contribución al rendimiento: Mejoran la eficiencia en las operaciones de búsqueda y unión basadas en las columnas de clave externa.

5. **IDX_RESERVA_ESTADO_FECHA_LLEGADA**:
    - Tipo de índice: NORMAL - Índice no único.
    - Tabla asociada: RESERVA.
    - Propósito: Creado por Oracle para respaldar las consultas que involucran las columnas ESTADO y FECHA_LLEGADA de la tabla RESERVA.
    - Contribución al rendimiento: Mejora la eficiencia en las operaciones de búsqueda y filtro basadas en las columnas ESTADO y FECHA_LLEGADA de la tabla RESERVA.

11.2 Documentación Diseño Requerimientos Funcionales de Consulta

- Sentencias SQL que responden el requerimiento y que fueron analizadas.
- Distribución de los datos con respecto a los parámetros de entrada utilizados en el requerimiento funcional. En particular se quiere un análisis de distribución que permita ver cómo puede cambiar
- el tamaño de la respuesta según el valor de los parámetros utilizados y la configuración de los datos de prueba.
- Valores de los parámetros utilizados en el análisis y que constituyen diferenciadores en los planes de ejecución obtenidos.
- Planes de consulta obtenidos en Oracle para la ejecución del requerimiento. Para ello,documente con una foto de pantalla los planes de consulta obtenidos en SQLDevelopper.
- Tiempos obtenidos con la ejecución de cada uno de los planes. Estos tiempos son medidos

RFC 10

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------
SQL_ID  671p30qdpxj5w, child number 0
-------------------------------------
SELECT C.* FROM CLIENTE C, RESERVA R, ALOJAMIENTO A WHERE
C.IDENTIFICACION = R.ID_CLIENTE AND A.ID = R.ID_ALOJAMIENTO     AND
R.FECHA_LLEGADA >= TO_DATE('01/05/2016','DD/MM/YYYY')      AND
R.FECHA_SALIDA <= TO_DATE('01/12/2023','DD/MM/YYYY')     AND
A.TIPO_ALOJA = 'HABITACION_HOTEL' GROUP BY C.IDENTIFICACION, C.NOMBRE,
C.VINCULO

Plan hash value: 3283810953
```

```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
| Id  | Operation             | Name         | E-Rows |  OMem |  1Mem | Used-Mem |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |              |        |       |       |          |
|*  1 |  HASH JOIN            |              |  85366 |  4172K|  2474K| 4282K (0)|
|   2 |   VIEW                | VW_GBF_17    |  85366 |       |       |          |
|   3 |    HASH GROUP BY      |              |  85366 |  3779K|  2749K| 3409K (0)|
|*  4 |     HASH JOIN         |              |  85868 |  4000K|  3201K| 3648K (0)|
|*  5 |      TABLE ACCESS FULL| ALOJAMIENTO  |  45000 |       |       |          |
|*  6 |      TABLE ACCESS FULL| RESERVA      |    163K|       |       |          |
|   7 |   TABLE ACCESS FULL   | CLIENTE      |    100K|       |       |          |

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - access("C"."IDENTIFICACION"="ITEM_1")
   4 - access("A"."ID"="R"."ID_ALOJAMIENTO")
   5 - filter("A"."TIPO_ALOJA"='HABITACION_HOTEL')
   6 - filter(("R"."FECHA_LLEGADA">=TO_DATE(' 2016-05-01 00:00:00',
              'syyyy-mm-dd hh24:mi:ss') AND "R"."FECHA_SALIDA"<=TO_DATE(' 2023-12-01
              00:00:00', 'syyyy-mm-dd hh24:mi:ss')))

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
```

TIEMPOS

SQL | Se han recuperado 50 filas en 0,085 segundos

	IDENTIFICACION	NOMBRE	VINCULO
1	462	Jaimie Shinn	EMPLEADO
2	463	Craig Rice	PADRE
3	465	Mai Fletcher	INVITADO
4	466	Susan Jordan	PROFESOR

Resultado de la Consulta

SQL | Todas las Filas Recuperadas: 1 en 0,019 seg

	IDENTIFICACION	NOMBRE	VINCULO
1	7673	Georgina Newsome	EGRESADO

RFC 11

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------
SQL_ID  bmkcrrhjmpq94, child number 0
-------------------------------------
SELECT C.IDENTIFICACION, C.NOMBRE, C.VINCULO FROM CLIENTE C WHERE
C.IDENTIFICACION NOT IN (   SELECT R.ID_CLIENTE   FROM RESERVA R,
ALOJAMIENTO A   WHERE R.ID_ALOJAMIENTO = A.ID    AND A.TIPO_ALOJA =
'VIVIENDA_U'   AND R.FECHA_LLEGADA >=
TO_DATE('01/05/2023','DD/MM/YYYY')    AND R.FECHA_SALIDA <=
TO_DATE('01/12/2023','DD/MM/YYYY') ) ORDER BY C.IDENTIFICACION,
C.VINCULO

Plan hash value: 394754091
```

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------
| Id  | Operation             | Name         | E-Rows |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |              |        |       |       |          |
|   1 |  SORT ORDER BY        |              |  92747 |  5439K|   959K| 4834K (0)|
|*  2 |   HASH JOIN RIGHT ANTI|              |  92747 |  2402K|  2402K| 1564K (0)|
|   3 |    VIEW               | VW_NSO_1     |   7253 |       |       |          |
|*  4 |     HASH JOIN SEMI    |              |   7253 |  1856K|  1856K| 2168K (0)|
|*  5 |      TABLE ACCESS FULL| RESERVA      |   7253 |       |       |          |
|*  6 |      TABLE ACCESS FULL| ALOJAMIENTO  |  30000 |       |       |          |

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------
|   7 |    TABLE ACCESS FULL  | CLIENTE      |    100K|       |       |          |
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("C"."IDENTIFICACION"="ID_CLIENTE")
   4 - access("R"."ID_ALOJAMIENTO"="A"."ID")
   5 - filter(("R"."FECHA_LLEGADA">=TO_DATE(' 2023-05-01 00:00:00',
              'syyyy-mm-dd hh24:mi:ss') AND "R"."FECHA_SALIDA"<=TO_DATE(' 2023-12-01
              00:00:00', 'syyyy-mm-dd hh24:mi:ss')))

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------
   6 - filter("A"."TIPO_ALOJA"='VIVIENDA_U')
```

TIEMPOS

Resultado de la Consulta

SQL | Se han recuperado 50 filas en 0,092 segundos

	IDENTIFICACION	NOMBRE	VINCULO
1	1	Albert Westbrooks	EGRESADO
2	2	Petra Friou	EMPLEADO
3	3	Jo Newcombe	COLECTIVA
4	4	Myrtle Jones	PADRE

RFC 12

```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
SQL_ID  4w9pg2u61pn44, child number 0
-------------------------------------
SELECT DISTINCT C.IDENTIFICACION, C.NOMBRE, C.VINCULO,        CASE WHEN
MONTLY_RESERVAS.ID_CLIENTE IS NOT NULL THEN 'Realiza reservas
mensuales' ELSE NULL END AS Calificacion_Reservas_Mensuales,
CASE WHEN COSTOSO_RESERVAS.ID_CLIENTE IS NOT NULL THEN 'Reserva
alojamientos costosos' ELSE NULL END AS Calificacion_Reservas_Costosas,
       CASE WHEN SUITE_RESERVAS.ID_CLIENTE IS NOT NULL THEN 'Reserva
suites' ELSE NULL END AS Calificacion_Reservas_Suites FROM CLIENTE C
LEFT JOIN (     SELECT ID_CLIENTE     FROM (         SELECT ID_CLIENTE,
EXTRACT(MONTH FROM FECHA_LLEGADA) AS MES, COUNT(*) AS NUM_RESERVAS

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
   FROM RESERVA          GROUP BY ID_CLIENTE, EXTRACT(MONTH FROM
FECHA_LLEGADA)     ) RESERVAS_MENSUALES     WHERE NUM_RESERVAS > 0 )
MONTLY_RESERVAS ON C.IDENTIFICACION = MONTLY_RESERVAS.ID_CLIENTE LEFT
JOIN (     SELECT ID_CLIENTE     FROM (         SELECT ID_CLIENTE,
COUNT(*) AS NUM_RESERVAS_COSTOSAS         FROM RESERVA         WHERE
PRECIO > 150         GROUP BY ID_CLIENTE

Plan hash value: 717031352
```

```
-------------------------------------------------------------------------------
| Id  | Operation                | Name      | E-Rows |  OMem |  1Mem | Used-Mem |

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |           |        |       |       |          |
|   1 |  HASH UNIQUE             |           |    100K|    11M|  2528K|          |
|*  2 |   FILTER                 |           |        |       |       |          |
|*  3 |    HASH JOIN RIGHT OUTER |           |    100K|  5598K|  3201K| 4878K (0)|
|   4 |     VIEW                 |           |  86053 |       |       |          |
|   5 |      HASH GROUP BY       |           |  86053 |  5151K|  2749K| 4633K (0)|
|*  6 |       TABLE ACCESS FULL  | RESERVA   |    170K|       |       |          |
|*  7 |     HASH JOIN RIGHT OUTER|           |    100K|  2801K|  2801K| 2212K (0)|
|   8 |      VIEW                |           |  22894 |       |       |          |
|   9 |       HASH GROUP BY      |           |  22894 |  2406K|  2406K| 1887K (0)|

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
|* 10 |        HASH JOIN         |           |  22894 |  2402K|  2402K| 2190K (0)|
|* 11 |         TABLE ACCESS FULL| HAB_HOTEL |   9997 |       |       |          |
|  12 |         TABLE ACCESS FULL| RESERVA   |    200K|       |       |          |
|* 13 |      HASH JOIN RIGHT OUTER|          |    100K|  9793K|  3201K|   11M (0)|
|  14 |       VIEW               |           |  10000 |       |       |          |
|* 15 |        FILTER            |           |        |       |       |          |
|  16 |         HASH GROUP BY    |           |  10000 |    11M|  3648K| 9597K (0)|
|  17 |          TABLE ACCESS FULL| RESERVA   |    200K|       |       |          |
|  18 |       TABLE ACCESS FULL  | CLIENTE   |    100K|       |       |          |
-------------------------------------------------------------------------------
```

```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------

   2 - filter(("MONTLY_RESERVAS"."ID_CLIENTE" IS NOT NULL OR
              "COSTOSO_RESERVAS"."ID_CLIENTE" IS NOT NULL OR
              "SUITE_RESERVAS"."ID_CLIENTE" IS NOT NULL))
   3 - access("C"."IDENTIFICACION"="COSTOSO_RESERVAS"."ID_CLIENTE")
   6 - filter("PRECIO">150)
   7 - access("C"."IDENTIFICACION"="SUITE_RESERVAS"."ID_CLIENTE")
  10 - access("R"."ID_ALOJAMIENTO"="HH"."ID_ALOJA")
  11 - filter("HH"."CATEGORIA"='SUIT')

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
  13 - access("C"."IDENTIFICACION"="MONTLY_RESERVAS"."ID_CLIENTE")
  15 - filter(COUNT(*)>0)
```

TIEMPOS

Resultado de la Consulta

SQL | Se han recuperado 50 filas en 0,45 segundos

	SEMANA	OFERTA_MAYOR_OCUPACION	OFERTA_MENOR_OCUPACION	OPERADOR_MAS_SOLICITADO	OPERADOR_MENOS_SOLICITADO
1	2021-23	22395	5248	Anita Miller	William Hartford
2	2021-23	22395	20983	Anita Miller	Ricky James
3	2021-23	22395	4909	Anita Miller	Luciano Watson
4	2021-23	22395	22429	Anita Miller	Tisha Spain

RFC 13

```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
SQL_ID  4w9pg2u6lpn44, child number 0
-------------------------------------
SELECT DISTINCT C.IDENTIFICACION, C.NOMBRE, C.VINCULO,        CASE WHEN
MONTLY_RESERVAS.ID_CLIENTE IS NOT NULL THEN 'Realiza reservas
mensuales' ELSE NULL END AS Calificacion_Reservas_Mensuales,
CASE WHEN COSTOSO_RESERVAS.ID_CLIENTE IS NOT NULL THEN 'Reserva
alojamientos costosos' ELSE NULL END AS Calificacion_Reservas_Costosas,
       CASE WHEN SUITE_RESERVAS.ID_CLIENTE IS NOT NULL THEN 'Reserva
suites' ELSE NULL END AS Calificacion_Reservas_Suites FROM CLIENTE C
LEFT JOIN (     SELECT ID_CLIENTE     FROM (         SELECT ID_CLIENTE,
EXTRACT(MONTH FROM FECHA_LLEGADA) AS MES, COUNT(*) AS NUM_RESERVAS

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
   FROM RESERVA         GROUP BY ID_CLIENTE, EXTRACT(MONTH FROM
FECHA_LLEGADA)     ) RESERVAS_MENSUALES     WHERE NUM_RESERVAS > 0 )
MONTLY_RESERVAS ON C.IDENTIFICACION = MONTLY_RESERVAS.ID_CLIENTE LEFT
JOIN (     SELECT ID_CLIENTE     FROM (         SELECT ID_CLIENTE,
COUNT(*) AS NUM_RESERVAS_COSTOSAS         FROM RESERVA         WHERE
PRECIO > 150         GROUP BY ID_CLIENTE

Plan hash value: 717031352
```

```
PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT          |           |       |       |       |          |
|   1 |  HASH UNIQUE              |           |   100K|    11M|  2528K|          |
|*  2 |   FILTER                  |           |       |       |       |          |
|*  3 |    HASH JOIN RIGHT OUTER  |           |   100K|  5598K|  3201K| 4878K (0)|
|   4 |     VIEW                  |           | 86053 |       |       |          |
|   5 |      HASH GROUP BY        |           | 86053 |  5151K|  2749K| 4633K (0)|
|*  6 |       TABLE ACCESS FULL   | RESERVA   |   170K|       |       |          |
|*  7 |     HASH JOIN RIGHT OUTER |           |   100K|  2801K|  2801K| 2212K (0)|
|   8 |      VIEW                 |           | 22894 |       |       |          |
|   9 |       HASH GROUP BY       |           | 22894 |  2406K|  2406K| 1887K (0)|

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------
|* 10 |        HASH JOIN          |           | 22894 |  2402K|  2402K| 2190K (0)|
|* 11 |         TABLE ACCESS FULL | HAB_HOTEL |  9997 |       |       |          |
|  12 |         TABLE ACCESS FULL | RESERVA   |   200K|       |       |          |
|* 13 |       HASH JOIN RIGHT OUTER|          |   100K|  9793K|  3201K|   11M (0)|
|  14 |        VIEW               |           | 10000 |       |       |          |
|* 15 |         FILTER            |           |       |       |       |          |
|  16 |          HASH GROUP BY    |           | 10000 |    11M|  3648K| 9597K (0)|
|  17 |           TABLE ACCESS FULL| RESERVA   |   200K|       |       |          |
|  18 |        TABLE ACCESS FULL  | CLIENTE   |   100K|       |       |          |
-------------------------------------------------------------------------------
PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------

   2 - filter(("MONTLY_RESERVAS"."ID_CLIENTE" IS NOT NULL OR
              "COSTOSO_RESERVAS"."ID_CLIENTE" IS NOT NULL OR
              "SUITE_RESERVAS"."ID_CLIENTE" IS NOT NULL))
   3 - access("C"."IDENTIFICACION"="COSTOSO_RESERVAS"."ID_CLIENTE")
   6 - filter("PRECIO">150)
   7 - access("C"."IDENTIFICACION"="SUITE_RESERVAS"."ID_CLIENTE")
  10 - access("R"."ID_ALOJAMIENTO"="HH"."ID_ALOJA")
  11 - filter("HH"."CATEGORIA"='SUIT')

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------
  13 - access("C"."IDENTIFICACION"="MONTLY_RESERVAS"."ID_CLIENTE")
  15 - filter(COUNT(*)>0)
```

TIEMPOS

Resultado de la Consulta ×

SQL | Se han recuperado 50 filas en 0,33 segundos

	IDENTIFICACION	NOMBRE	VINCULO	CALIFICACION_RESERVAS_MENSUALES	CALIFICACION_RESERVAS_COSTOSAS	CALIFICACION_RESERVAS_SUITES
1	473	Barry Willis	PADRE	Realiza reservas mensuales	Reserva alojamientos costosos	(null)
2	494	Frank Lee	ESTUDIANTE	Realiza reservas mensuales	Reserva alojamientos costosos	(null)
3	515	Nathan Gilbert	PROFESOR	Realiza reservas mensuales	Reserva alojamientos costosos	(null)
4	540	Nancy Asbill	EGRESADO	Realiza reservas mensuales	Reserva alojamientos costosos	(null)

12 Construcción de la aplicación, ejecución de pruebas y análisis de resultados

12.1 Documentación carga de datos

1. Población de la Base de Datos

Para el proceso de población de la base de datos se crearon siete documentos individuales en Python que siguen una estructura muy similar. Cada uno de los documentos crea alrededor de 100.000 datos

dependiendo de la tabla sobre la que se va a trabajar. Para la tabla de cliente y operador (en los referentes a personas naturales que rentan su vivienda, o una habitación) fue fundamental el uso de la librería "names" de Python la cual emula nombres aleatorios, los cuales son de suma importancia para que estos atributos de las tablas tengan sentido en un ámbito real. Por otro lado, para la generación de Alojamientos se realizó de forma que cada persona natural registrada tuviese publicado por lo menos un alojamiento y que los operadores mayoritarios como hoteles o viviendas universitarias tuviesen alrededor de 7500 alojamientos a su nombre. En cuanto a las reservas se crearon aproximadamente 200,000 de forma que todos los alojamientos tuviesen reservas registradas históricamente y que las fechas fuesen coherentes en cuanto a que la fecha de salida fuese posterior a la de entrada. Los servicios eran muy pocos debido a que no hay una gran variedad de servicios a ofrecer, por ende, se crearon de forma tradicional haciendo uso de la herramienta de INSERT de Oracle simplemente creando 20 servicios diferentes que se ofrecen en la mayoría de los lugares de estadía. Una vez se crearon estas tablas mencionadas ya se pasó a la segunda parte de la carga de datos, la cual se encargaría de realizar conexiones entre tablas que ya tienen tuplas y que, por ende, podrían relacionarse bien entre sí. Estas tablas serian Alojamiento_Servicio, que relaciona cada alojamiento con los servicios que presta, en el programa de Python encargado de esta parte se daba para cada alojamiento un numero aleatorio de servicios (entre 1 y 5), donde los servicios son aleatorios entre las opciones creadas en la tabla servicios. Por su parte en la tabla Alojamiento_Operador se utilizó una modalidad similar, sin embargo, es importante hacer énfasis en que al haber más alojamientos que operadores (debido a los de alto tamaño como hoteles, hostales y vivienda universitaria) se modificó ligeramente el código de forma que a cada uno de estos grandes operadores se le asignaran específicamente los alojamientos de ese tipo en grandes cantidades (como mencionado antes 7500 para cada uno). Finalmente, las habitaciones de hoteles y hostales se relacionaron con su hotel/hostal padre de forma que cada uno quedara con las 7500 habitaciones mencionadas. Para la creación de datos aleatorios como direcciones, cedula, numero de registro y la conexión entre servicios y alojamientos se utilizó la librería random y la función randint() de esta, para generar enteros entre rangos definidos.