

Proyecto del curso – Iteración 3

Daniel F. Pérez Chaparro, Sofía Abadía Bermeo

Universidad de los Andes, Bogotá, Colombia

{df.perezc, s.abadia} @uniandes.edu.co

Fecha de presentación: Mayo 03 de 2020

Tabla de contenido

1 INTRODUCCIÓN	2
2 MODELO CONCEPTUAL E INFORMACIÓN DE LAS TABLAS.....	2
3 REQUERIMIENTOS FUNCIONALES	7
4 IMPLEMENTACIÓN DE REQUERIMIENTOS FUNCIONALES DE MODIFICACIÓN.....	10
4.1 REGISTRAR RESERVA COLECTIVA	10
4.1.1 Código	10
4.1.2 Resultado	13
4.1.3 Análisis	13
4.2 CANCELAR RESERVA COLECTIVA	13
4.2.1 Código	14
4.2.2 Resultado	14
4.2.3 Análisis	15
4.3 DESHABILITAR OFERTA DE ALOJAMIENTO	15
4.3.1 Código	15
4.3.2 Resultado	16
4.3.3 Análisis	16
4.4 REHABILITAR OFERTA DE ALOJAMIENTO.....	16
4.4.1 Código	16
4.4.2 Resultado	17
4.4.3 Análisis	17
5 IMPLEMENTACIÓN DE REQUERIMIENTOS FUNCIONALES DE CONSULTA	18
5.1 MOSTRAR EL USO DE ALOHANDES	18
5.1.1 Código	18
5.1.2 Resultado	18
5.1.3 Análisis	18
5.3 ANALIZAR LA OPERACIÓN DE ALOHANDES	18
5.3.2 Resultado	19
5.3.3 Análisis	19
5.4 ENCONTRAR LOS CLIENTES FRECUENTES	19
5.4.1 Código	19
5.4.2 Resultado	20
5.4.3 Análisis	20
6 CONCLUSIONES.....	20
7 BIBLIOGRAFÍA	20

1 Introducción

En este documento se presentarán los resultados y el análisis de la tercera iteración del proyecto de la clase de Sistemas Transaccionales. El objetivo de esta tercera iteración era la integración de requerimientos funcionales de modificación y de consulta en la aplicación desarrollada en la iteración 2. Como en la iteración anterior, se debía empezar con la creación de las secuencias SQL en SQL Server, para finalizar con la inserción en la aplicación en Eclipse.

2 Modelo conceptual e información de las tablas

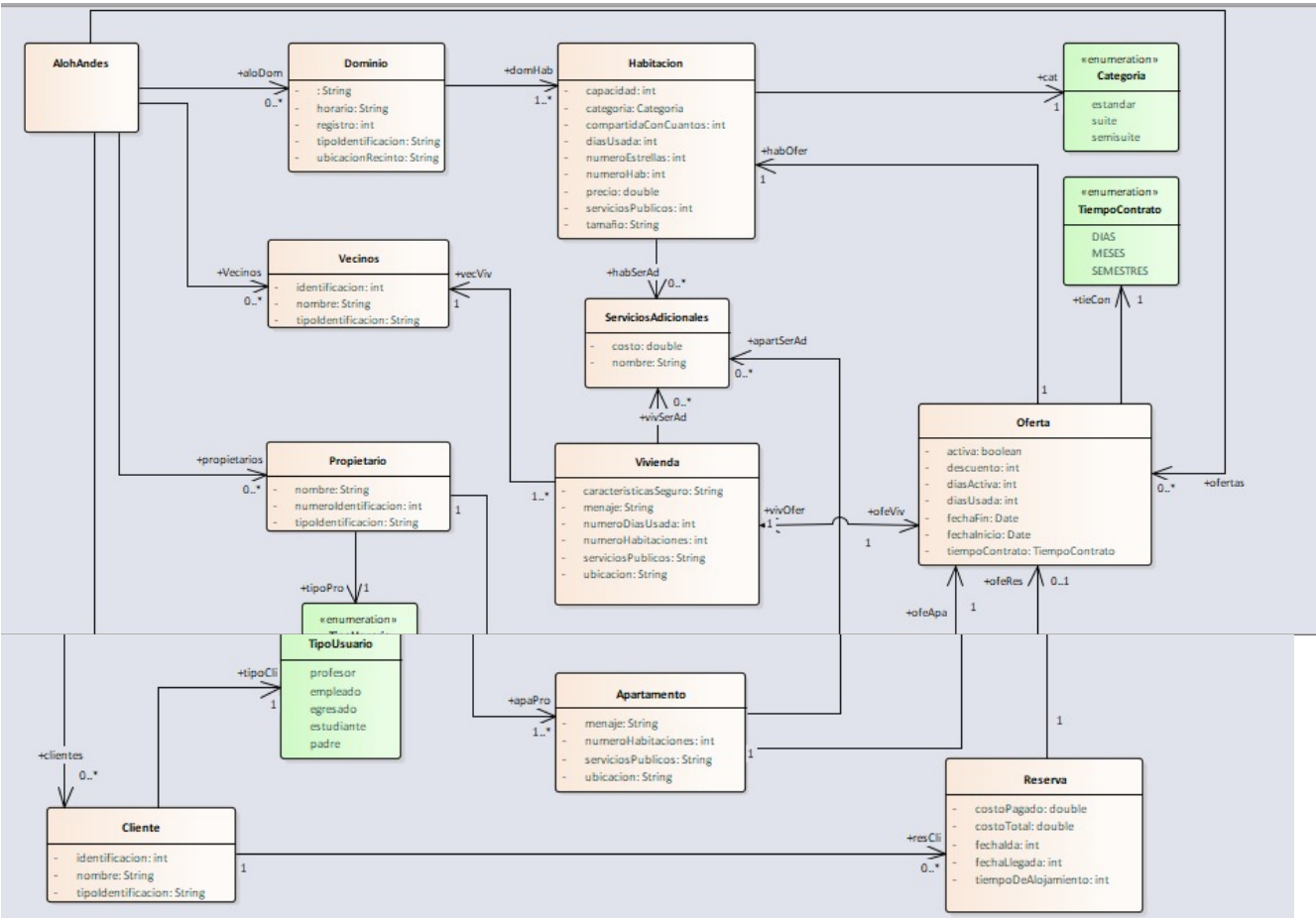


Figura 1. Modelo conceptual

APAR TAME NTO	5										
NUME RO_H ABITA CIONE S: NUMB ER	SERVI CIOS_ PUBLI COS: VARC HAR	UBICA CION: VARC HAR	PROPI ETARI O: NUMB ER								

NN	NN	PK	FK								
CLIENTE	4										
IDENTIFICACION: NUMBER	TIPO_IDENTIFICACION: VARCHAR	NOMBRE: VARCHAR	TIPOCLIENTE: VARCHAR								
PK	PK CK IN ('TI', 'CC', 'CE', 'PA')	NN	CK IN(PROFESOR, EMPLEADO, EGRESADO, ESTUDIANTE, PADRE)								
DOMINIO	2										
REGISTRO: NUMBER	TIPO_REGISTRO: VARCHAR	NOMBRE: VARCHAR	UBICACIONRECINTO: VARCHAR	HORARIO: VARCHAR							
PK	PK, CK IN(HOTEL , HOSTAL, EMPRESA, RC, TI, CC, CE, PA)	NN	NN								
HABITACION	7										
ID	CAPACIDAD :	COMPARTIDAD_CON_C	DIAS_USADA:	NUMEROESTRELLAS:	NUMERO_HAB	PRECIO: NUMBER	SERVICIOS: VARCHAR	TAMAÑO: VARCHAR	CATEGORIA	IDDOMINIO: NUMBER	TIPOREGDOMINIO

	NUMB ER	UANT OS: NUMB ER	NUMB ER	NUMB ER							
PK	CK>0	NN>=0	CK >=0	CK>=0	NN	CK>0	NN	NN	NN	FK	FK
OFERT A	8										
ID: NUMB ER	DESCU ENTO: NUMB ER	DIAS_ ACTIV A	DIAS_ USAD A	FECHA _FIN: DATE	FECHA _INICI O: DATE	TIEMP O_CO NTRA TO: VARC HAR	ACTIV A: VARC HAR				
PK	CK<=100	CK>=0	CK>=0	NN	NN	NN	NN				
PROPI ETARI O	1										
NOMB RE: VARC HAR	NUME RO_ID ENTIFI CACIO N: NUMB ER	TIPO_I DENTI FICACI ON: VARC HAR	TIPOP ROPIE TARIO : VARC HAR								
NN	PK	PK, CK IN ('CC', 'CE', 'PA')	CK IN(PROFE SOR, EMPL EADO, EGRES ADO, ESTUD IANTE, PADR E)								
RESER VA	9										
ID: NUMB ER	COST O_PA GADO : NUMB ER	COST O_TO TAL: NUMB ER	FECHA _IDA: DATE	FECHA _LLEG ADA: DATE	TIEMP O_AL OJAMI ENTO: STRIN G	IDCLIE NTE: NUMB ER	TIPOI DCLIE NTE	OFERT A: NUMB ER			
PK	NN	NN	NN	NN	NN	FK	FK	FK			

[illegible]

[illegible]

	NUMB ER										
PK, FK	PK, FK										

Para esta iteración se requirió adicionar lo columna booleana “activa” a la tabla OFERTA. Además, se hizo necesaria la creación de la tabla RESERVACOLECTIVA y también se creó la tabla RESCOLRES para poder relacionar adecuadamente las tuplas de RESERVA con las tuplas de RESERVACOLECTIVA

3 Requerimientos funcionales

Nombre	RF7. Registrar reserva colectiva.
Resumen	El cliente informa la cantidad de habitaciones que necesita de una cierto tipo de alojamiento con cierto servicios deseados y el sistema de Alohandes confirma si tiene la capacidad de aceptar o no. Si se acepta se realiza la reserva.
Entradas	
Número de identificación del cliente	
Fecha de llegada	
Fecha de salida	
Servicios solicitados	
Resultados	
Consultar e informar al cliente si se tiene la capacidad para aceptar la reserva a las habitaciones de los alojamientos escogidos y con los servicios especificados en las fechas dadas. En caso de que se acepte, se realiza la reserva de estas habitaciones. En caso de no aceptarse, se le informa al cliente que no es posible realizar su reserva	
RNF asociados	
Concurrencia	
Privacidad	
Transaccionalidad	

Nombre	RF8. Cancelar reserva colectiva.
Resumen	El cliente cancela la reserva colectiva
Entradas	
Número de identificación del cliente	
Número de identificación de la reserva	
Resultados	
Se cancela la reserva dada por parámetro, se libera la oferta para esos alojamientos	

RNF asociados
Concurrencia
Privacidad
Transaccionalidad

Nombre	RF9. Deshabilitar oferta de alojamiento.
Resumen	El propietario de un alojamiento deshabilita una oferta; las reservas vigentes sobre esa oferta son relocalizadas de acuerdo al orden en que fueron realizadas. Si hay reservas que no pueden ser relocalizadas se deberá informar al cliente.

Entradas
Número de identificación del propietario
Número de identificación de la oferta
Resultados
Se suspende la oferta indicada, las reservas vigentes de esta oferta son relocalizadas en otras ofertas. Se le indica al cliente si su reserva fue relocalizada o si no alcanzó a reacomodarse.
RNF asociados
Concurrencia
Privacidad
Transaccionalidad

Nombre	RF10.Rehabilitar oferta de alojamiento.
Resumen	El propietario rehabilita una oferta
Entradas	
Número de identificación del propietario	
Número de identificación de la oferta	
Resultados	
Se rehabilita la oferta especificada y es accesible a nuevas reservas	
RNF asociados	
Concurrencia	
Privacidad	
Transaccionalidad	

Nombre	RFC5. Mostrar el uso de AlohaAndes de los clientes
Resumen	Informar el uso de AlohaAndes para cada tipo de usuario de la comunidad

Entradas
Resultados
Calcular e informar el uso de AlohAndes de todos los usuarios
RNF asociados
Concurrencia
Privacidad

Nombre	RFC6. Mostrar el uso de AlohAndes de un usuario.
Resumen	Informar el uso de AlohAndes de un usuario dado
Entradas	
	Número de identificación del cliente
Resultados	
	Calcular e informar el uso de AlohAndes del usuario dado por parámetro (número de noches o meses contratados, características de alojamiento utilizado, dinero pagado, ...)
RNF asociados	
	Concurrencia
	Privacidad

Nombre	RFC7.analizar la operación de Alohandes.
Resumen	Informar las fechas de mayor demanda (mayor cantidad de alojamientos ocupados), las de mayores ingresos (mayor cantidad de dinero recibido) y las de menor ocupación
Entradas	
	Unidad de tiempo (una semana, un mes...)
	Tipo de alojamiento
Resultados	
	Calcular las fechas con mayor demanda, mayores ingresos y menor ocupación en un tiempo dado por un alojamiento especificado e indicarlo.
RNF asociados	
	Concurrencia

Nombre	RFC8.Encontrar los clientes frecuentes.
Resumen	Presenta la información de los clientes frecuentes para un alojamiento dado. Se considera frecuente a un cliente si ha utilizado (o tiene reservado) ese

	alojamiento por lo menos en tres ocasiones o por lo menos 15 noches, durante todo el periodo de operación de Alohandes.
Entradas	
Tipo de alojamiento	
Resultados	
Consulta los clientes frecuentes del tipo de alojamiento especificado y los indica	
RNF asociados	
Concurrencia	
Privacidad	

Nombre	RFC9. Encontrar las ofertas de alojamiento que no tienen mucha demanda.
Resumen	Informar las ofertas de alojamiento que no han recibido clientes en periodos superiores a 1 mes, durante todo el periodo de operación de Alohandes.
Entradas	
Resultados	
Consulta las ofertas que no tienen mucha demanda y las informa	
RNF asociados	
Concurrencia	
Privacidad	

4 Implementación de requerimientos funcionales de modificación

4.1 Registrar reserva colectiva

El usuario indica el tipo de alojamiento deseado y la cantidad deseada. ALOHANDES debe revisar si está en capacidad de satisfacer esa solicitud, eventualmente con varios proveedores, y en caso afirmativo realizar las reservas individuales correspondientes. La reserva colectiva es identificable de manera individual. Tanto en caso afirmativo como negativo debe informar de manera completa y coherente las operaciones realizadas.

4.1.1 Código

```

public ReservaColectiva registrarReservaColectiva (long idresCol, long idc, String tipoc, Timestamp lle, Timestamp ida, Timestamp fePago, int canti
{

    PersistenceManager pm = pmf.getPersistenceManager();
    Transaction tx=pm.currentTransaction();
    try
    {
        System.out.println("corriendo");
        tx.begin();
        // revisar que se pueda cubrir la demanda
        ReservaColectiva resp = null;
        switch(tipoAlojamiento) {
            case "APARTAMENTO":
                System.out.println("CORR1");
                List<Object> listaOfeApa = sqlOfertaApartamento.consultarOfertasDisponiblesApartamento(pm, cantidadRes, servicios );
                //nunca listaOfeApa.size() nunca va a ser mayor a cantidad res porque de ser mayor la recortaría
                if(listaOfeApa.size() == cantidadRes)
                {
                    System.out.println("CORR2");

                    //hay que ir guardando los ids de las reservas para luego poder hacer la relacion
                    List<Long> idsReservas = new ArrayList<Long>();
                    for( Object tupla : listaOfeApa)
                    {

                        Object[] datos = (Object[]) tupla;

                        Long tempId = nextval();
                        double tempCosPa = 0;
                        double tempCosTot = 10000;
                        Long tempIdOfe = ((BigDecimal) datos[0]).longValue();
                        idsReservas.add(tempId);
                        sqlReserva.adicionarReserva(pm, tempId, tempCosPa, tempCosTot, ida, lle, tipoAlojamiento, idc, tipoc, tempIdOfe);

                        System.out.println("CORR2.1");
                    }
                    System.out.println("CORR3");
                }
            }
    }
    catch (Exception e)
    {
        tx.rollback();
        System.out.println("Error: " + e.getMessage());
    }
}

```

```

sqlReservaColectiva.registrarReservaColectiva(pm, idresCol, tipoc, idc, lle, ida, fePago, cantidadRes, tipoAlojamiento, costo);
System.out.println("CORR3.01");

//crear la reservaColectiva
resp = new ReservaColectiva(idresCol, tipoc, idc, lle, ida, fePago, cantidadRes, tipoAlojamiento, costo);
log.trace("Inserción de reservaColectiva" + idresCol + " tuplas insertadas");
System.out.println("CORR3.1");
//relacionar las reservas con la reservaColectiva
for( Long idReservaRegistrada: idsReservas)
{
    System.out.println("CORR3.2");
    sqlResColRes.adicionarResColRes(pm, idReservaRegistrada, idresCol);
}
System.out.println("CORR4");
}
break;
case "VIVIENDA":
System.out.println("CORR1");
List<Object> listaOfeViv = sqlOfertaVivienda.consultarOfertasDisponiblesVivienda(pm, cantidadRes, servicios);
//nunca listaOfeApa.size() nunca va a ser mayor a cantidad res porque de ser mayor la recortaría
if(listaOfeViv.size() == cantidadRes)
{
    System.out.println("CORR2");

    //hay que ir guardando los ids de las reservas para luego poder hacer la relacion
    List<Long> idsReservas = new ArrayList<Long>();
    for( Object tupla : listaOfeViv)
    {
        Object[] datos = (Object[]) tupla;

        Long tempId = nextval();
        double tempCosPa = 0;
        double tempCosTot = 10000;
        Long tempIdOfe = ((BigDecimal) datos[0]).longValue();
        idsReservas.add(tempId);
        sqlReserva.adicionarReserva(pm, tempId, tempCosPa, tempCosTot, ida, lle, tipoAlojamiento, idc, tipoc, tempIdOfe);

        System.out.println("CORR2.1");
    }
    System.out.println("CORR3");

    sqlReservaColectiva.registrarReservaColectiva(pm, idresCol, tipoc, idc, lle, ida, fePago, cantidadRes, tipoAlojamiento, costo);
    System.out.println("CORR3.01");

    //crear la reservaColectiva
    resp = new ReservaColectiva(idresCol, tipoc, idc, lle, ida, fePago, cantidadRes, tipoAlojamiento, costo);
    log.trace("Inserción de reservaColectiva" + idresCol + " tuplas insertadas");
    System.out.println("CORR3.1");
    //relacionar las reservas con la reservaColectiva
    for( Long idReservaRegistrada: idsReservas)
    {
        System.out.println("CORR3.2");
        sqlResColRes.adicionarResColRes(pm, idReservaRegistrada, idresCol);
    }
    System.out.println("CORR4");
}
break;
case "HABITACION":
System.out.println("CORR1");
List<Object> listaOfeHab = sqlOfertaHabitacion.consultarOfertasDisponiblesHabitacion(pm, cantidadRes, servicios);
//nunca listaOfeApa.size() nunca va a ser mayor a cantidad res porque de ser mayor la recortaría
if(listaOfeHab.size() == cantidadRes)
{
    System.out.println("CORR2");

    //hay que ir guardando los ids de las reservas para luego poder hacer la relacion
    List<Long> idsReservas = new ArrayList<Long>();
    for( Object tupla : listaOfeHab)
    {

```

```

        Object[] datos = (Object[]) tupla;

        Long tempId = nextval();
        double tempCosPa = 0;
        double tempCosTot = 10000;
        Long tempIdOferta = ((BigDecimal) datos[0]).longValue();
        idsReservas.add(tempId);
        sqlReserva.adicionarReserva(pm, tempId, tempCosPa, tempCosTot, ida, lle, tipoAlojamiento, idc, tipoc, tempIdOferta);

        System.out.println("CORR2.1");
    }
    System.out.println("CORR3");

    sqlReservaColectiva.registrarReservaColectiva(pm, idresCol, tipoc, idc, lle, ida, fePago, cantidadRes, tipoAlojamiento, costo);
    System.out.println("CORR3.01");

    //crear la reservaColectiva
    resp = new ReservaColectiva(idresCol, tipoc, idc, lle, ida, fePago, cantidadRes, tipoAlojamiento, costo);
    Log.trace("Inserción de reservaColectiva" + idresCol + " tuplas insertadas");
    System.out.println("CORR3.1");
    //relacionar las reservas con la reservaColectiva
    for( Long idReservaRegistrada: idsReservas)
    {
        System.out.println("CORR3.2");
        sqlResColRes.adicionarResColRes(pm, idReservaRegistrada, idresCol);
    }
    System.out.println("CORR4");
}
break;
default:
}

tx.commit();

return resp;
}
catch (Exception e)
{
    e.printStackTrace();
    Log.error("Exception : " + e.getMessage() + "\n" + darDetalleException(e));
    return null;
}
finally
{
    if (tx.isActive())
    {
        tx.rollback();
    }
    pm.close();
}
}

```

4.1.2 Resultado

IDRESERVA	IDRESERVACOLECTIVA
1	61
2	62
3	63

ID	TIPOIDCLIENTE	IDCLIENTE	FECHALLEGADA	FECHAIDA	FECHADEPAGO	CANTIDAD	TIPOALOJAMIENTO	COSTO
1	30106CC	311	12/04/20	12/04/20	12/04/20	3	APARTAMENTO	1000000

ID	COSTOPAGADO	COSTOTOTAL	FECHAIDA	FECHALLEGADA	FECHADEPAGO	TIEMPOALOJAMIENTO	IDCLIENTE	TIPOIDCLIENTE	IDOFERTA
1	61	0	10000	12/04/20	12/04/20	(null)	APARTAMENTO	311 CC	262
2	62	0	10000	12/04/20	12/04/20	(null)	APARTAMENTO	311 CC	31019
3	63	0	10000	12/04/20	12/04/20	(null)	APARTAMENTO	311 CC	31022

4.1.3 Análisis

El método inicia por consultar si hay suficientes reservas para el tipo específico solicitado (apartamento, vivienda, habitación) y el servicio solicitado. En caso de que se pueda cubrir la demanda se procede a crear todas las tuplas necesarias en la tabla RESERVA. Posteriormente se crea la tupla correspondiente a la tabla RESERVACOLECTIVA. Finalmente se crean las relaciones entre las tuplas de RESERVA y la tupla de RESERVACOLECTIVA.

4.2 Cancelar reserva colectiva

El usuario indica la reserva colectiva que quiere cancelar y ALOHANDES cancela las reservas individuales correspondientes y calcula también las penalizaciones correspondientes.

4.2.1 Código

```
public long eliminarReservaColectivaPorId (long idResCol)
{
    PersistenceManager pm = pmf.getPersistenceManager();
    Transaction tx=pm.currentTransaction();
    try
    {
        tx.begin();
        System.out.println("CORE0 ");
        List<Object> listaResColRes = sqlResColRes.darReservaPorId(pm, idResCol) ;
        System.out.println("CORE02");
        System.out.println("CORE1 " + listaResColRes.size() );
        //hay que guardar los ids de reservas de rescoldes antes de borrarlos
        List<Long> idsReservas = new ArrayList<Long>();
        if(listaResColRes.size()>0)
        {

            for( Object tupla : listaResColRes)
            {

                Object[] datos = (Object[]) tupla;
                Long tempIdRes = ((BigDecimal) datos[0]).longValue();
                idsReservas.add(tempIdRes);
                System.out.println("CORE2");

            }
        }
        //eliminar las rescoldes relacionadas con la resCol
        sqlResColRes.eliminarResColResId(pm, idResCol);
        System.out.println("CORE3");
        //eliminarLasReservas
        if(idsReservas.size()>0)
        {

            for( Long idABorrar: idsReservas)
            {

                sqlReserva.eliminarReservaPorId(pm, idABorrar);
                System.out.println("CORE4");

            }
        }
        //finalmente se borra la resCol
        long resp = sqlReservaColectiva.eliminarReservaColectivaPorId(pm, idResCol);
        System.out.println("CORE5");
        tx.commit();

        return resp;
    }
    catch (Exception e)
    {
        e.printStackTrace();
        Log.error ("Exception : " + e.getMessage() + "\n" + darDetalleException(e));
        return -1;
    }
    finally
    {
        if (tx.isActive())
        {
            tx.rollback();
        }
        pm.close();
    }
}
```

4.2.2 Resultado



4.2.3 Análisis

Para realizar adecuadamente el borrado sin entrar en conflictos por llaves foraneas las primeras tuplas que deben ser eliminadas son las de la tabla COLRESCOL y a la vez que son borradas esas tuplas el método almacena la información de que tuplas de RESERVA están relacionadas con las tuplas que están siendo borradas en COLRESCOL. Luego se borran las tuplas de RESERVA que habían sido almacenadas. Finalmente, se borra la tupla correspondiente de RESERVACOLECTIVA.

4.3 Deshabilitar oferta de alojamiento

Las reservas vigentes sobre esa oferta de alojamiento deben entonces relocalizarse en las otras ofertas de alojamiento disponibles en ALOHANDES, dando prioridad a las vigentes en el momento que se realiza la operación y luego en el orden en que fueron realizadas las reservas. Siendo un caso excepcional, las reservas colectivas involucradas deben desagregarse a las reservas individuales correspondientes y puede haber reservas que no pueden ser satisfechas con la oferta disponible en el momento que se realiza la operación. ALOHANDES debe informar de manera completa y clara las operaciones realizadas, tanto el traslado exitoso de reservas como las reservas que no pudieron ser trasladadas. La oferta de alojamiento en cuestión no debe ser tenida en cuenta para reservas mientras no se haya rehabilitado

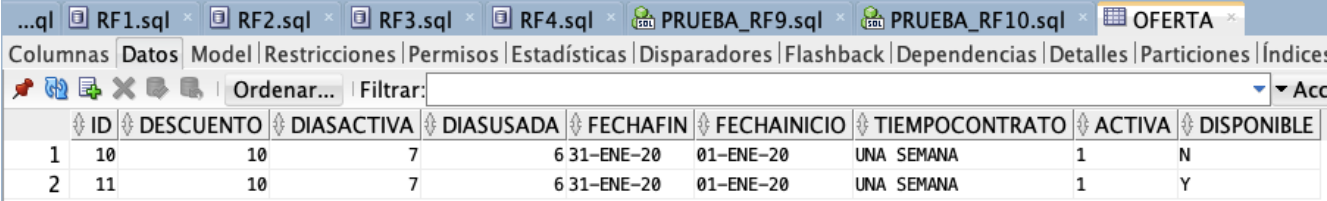
4.3.1 Código

```
--Actualiza Oferta1
UPDATE OFERTA
SET oferta.activa='0'
WHERE OFERTA.ID= 10;
COMMIT;

select distinct oferta.id as ofertas, reserva.id as cambiar
from oferta, reserva
where (oferta.id in (select distinct oferta.id
                    from oferta, reserva
                    where (oferta.id!= reserva.idoferta
                        and oferta.disponible= 'Y'
                        and reserva.id in( select reserva.id
                                        from reserva, oferta
                                        where (reserva.idoferta=oferta.id and oferta.activa='0')
                                    )
                        and oferta.fechainicio= reserva.fechallegada
                    )
                )
    and reserva.id in ( select distinct reserva.id
                        from reserva, oferta
                        where (reserva.idoferta=oferta.id and oferta.activa='0')
                    )
    )
    --and rownum<10
    )
;

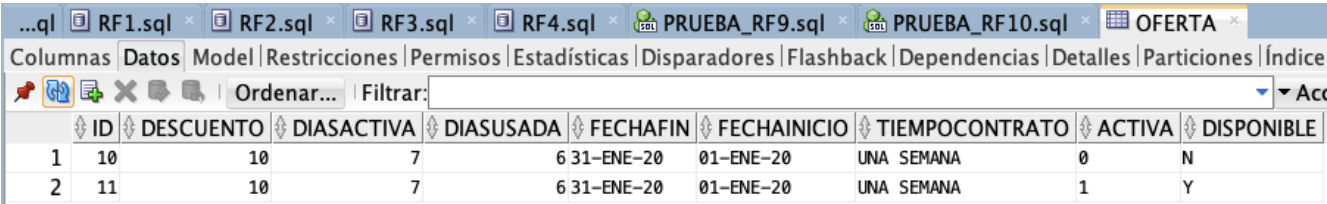
update reserva
set reserva.idoferta=( select distinct oferta.id
                      from oferta, reserva
                      where (oferta.id!= reserva.idoferta
                          and oferta.fechainicio<= reserva.fechallegada and oferta.fechafin >= reserva.fechaida
                          and reserva.id in( select reserva.id
                                          from reserva, oferta
                                          where (reserva.idoferta=oferta.id and oferta.activa='0')
                                      )
                      )
                      )
where reserva.id in ( select reserva.id
                    from reserva, oferta
                    where (reserva.idoferta=oferta.id and oferta.activa='0')
                )
    )
```

4.3.2 Resultado



	ID	DESCUENTO	DIASACTIVA	DIASUSADA	FECHAFIN	FECHAINICIO	TIEMPOCONTRATO	ACTIVA	DISPONIBLE
1	10	10	7	6	31-ENE-20	01-ENE-20	UNA SEMANA	1	N
2	11	10	7	6	31-ENE-20	01-ENE-20	UNA SEMANA	1	Y

Figura . Estado de la tabla Oferta antes de la actualización



	ID	DESCUENTO	DIASACTIVA	DIASUSADA	FECHAFIN	FECHAINICIO	TIEMPOCONTRATO	ACTIVA	DISPONIBLE
1	10	10	7	6	31-ENE-20	01-ENE-20	UNA SEMANA	0	N
2	11	10	7	6	31-ENE-20	01-ENE-20	UNA SEMANA	1	Y

Figura . Estado de la tabla Oferta después de la actualización

OFERTAS	CAMBIAR
11	10
11	11

Figura . Las reservas a modificar y las ofertas disponibles

4.3.3 Análisis

Para probar este método y el siguiente (4.4) es necesario tener un antes y un después, razón por la cual se creó una serie de casos de prueba que se puede ver en más detalle en el Excel adjunto.

4.4 Rehabilitar oferta de alojamiento

Cuando la oferta de alojamiento vuelve a estar disponible y puede por lo tanto aceptar nuevas reservas. ALOHANDES debe informar de manera completa y clara las operaciones realizadas.

4.4.1 Código

```
UPDATE OFERTA
SET oferta.activa='1'
WHERE OFERTA.ID=205;

COMMIT;
```

Figura . Código de la actualización

4.4.2 Resultado

```
-- PRUEBA RF10
-- la base de datos tiene los datos de finalizacion de PRUEBA_RF9

--Crear reserva3 (no debe ser creada)
INSERT INTO RESERVA
(ID, COSTOPAGADO,COSTOTOTAL,FECHAIDA,FECHALLEGADA,FECHADEPAGO,TEMPOALOJAMIENTO,IDCLIENTE,TIPOIDCLIENTE,IDOFACTA)
VALUES
(12, 100000, 150000, '18-ENE-2020', '11-ENE-2020', '05-ENE-2020', 'UNA SEMANA', 10, 'CC', 10);

DELETE
FROM RESERVA
WHERE RESERVA.ID=10 AND RESERVA.IDOFERTA IN( SELECT OFERTA.ID
FROM OFERTA
WHERE OFERTA.ACTIVA='0');

UPDATE OFERTA
SET OFERTA.DISPONIBLE= 'N'
WHERE OFERTA.ID= 10 AND OFERTA.ID IN (SELECT RESERVA.IDOFERTA FROM RESERVA);
COMMIT;

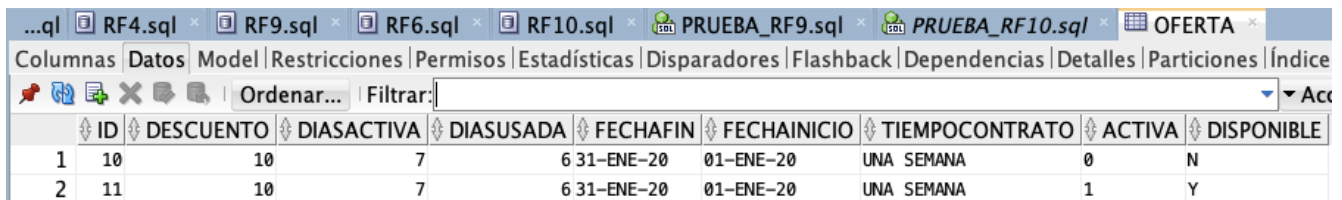
--Actualiza oferta1
UPDATE OFERTA
SET oferta.activa='1'
WHERE OFERTA.ID=10;
COMMIT;

--Crear reserva4 (debe ser creada)
INSERT INTO RESERVA
(ID, COSTOPAGADO,COSTOTOTAL,FECHAIDA,FECHALLEGADA,FECHADEPAGO,TEMPOALOJAMIENTO,IDCLIENTE,TIPOIDCLIENTE,IDOFACTA)
VALUES
(13, 100000, 150000, '18-ENE-2020', '11-ENE-2020', '05-ENE-2020', 'UNA SEMANA', 10, 'CC', 10);

DELETE
FROM RESERVA
WHERE RESERVA.ID=10 AND RESERVA.IDOFERTA IN( SELECT OFERTA.ID
FROM OFERTA
WHERE OFERTA.ACTIVA='0');

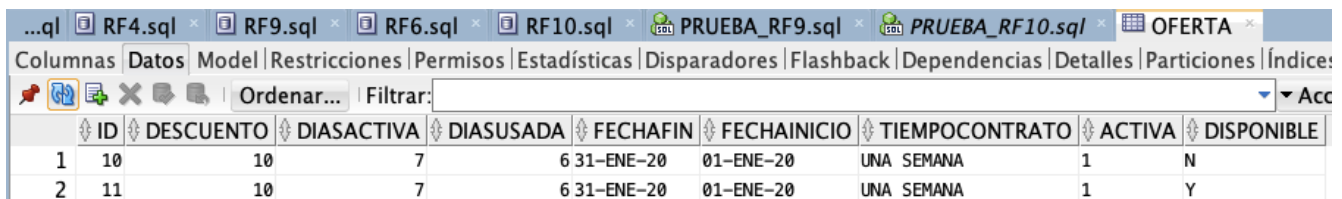
UPDATE OFERTA
SET OFERTA.DISPONIBLE= 'N'
WHERE OFERTA.ID= 10 AND OFERTA.ID IN (SELECT RESERVA.IDOFERTA FROM RESERVA);
```

Figura . Código de la prueba de funcionamiento



ID	DESCUENTO	DIASACTIVA	DIASUSADA	FECHAFIN	FECHAINICIO	TIEMPOCONTRATO	ACTIVA	DISPONIBLE
10	10	7	6	31-ENE-20	01-ENE-20	UNA SEMANA	0	N
11	10	7	6	31-ENE-20	01-ENE-20	UNA SEMANA	1	Y

Figura . Estado de la tabla Oferta antes de la actualización



ID	DESCUENTO	DIASACTIVA	DIASUSADA	FECHAFIN	FECHAINICIO	TIEMPOCONTRATO	ACTIVA	DISPONIBLE
10	10	7	6	31-ENE-20	01-ENE-20	UNA SEMANA	1	N
11	10	7	6	31-ENE-20	01-ENE-20	UNA SEMANA	1	Y

Figura . Estado de la tabla Oferta después de la actualización

4.4.3 Análisis

En la base de datos antes de correr este método hay una oferta1 que no está activa. Así pues, se prueba que no se pueda insertar una nueva reserva3 asociada a esta oferta. Posteriormente, se actualiza la oferta1, se vuelve a dejar activa y se intenta agregar una reserva4 asociada a esta oferta, en este caso, la reserva debe ser agregada.

5 Implementación de requerimientos funcionales de consulta

5.1 Mostrar el uso de Alohandes

Mostrar el uso de Alohandes para cada tipo de usuario de la comunidad

5.1.1 Código

```
SELECT CLI.NOMBRE, cli.tipocliente, SUM(RES.COSTOPAGADO) AS DINEROPAGADO, (RES.FECHAIDA - RES.FECHALLEGADA) AS DIAS, COUNT(res.id) AS RESERVAS
FROM CLIENTE CLI, RESERVA RES
WHERE(
    cli.numeroidentificacion = res.idcliente
)
GROUP BY CLI.NOMBRE, cli.tipocliente, res.fechallegada, res.fechaida, (RES.FECHAIDA - RES.FECHALLEGADA)
;
```

Figura . Código de la consulta

5.1.2 Resultado

	NOMBRE	TIPOCLIENTE	DINEROPAGADO	DIAS	RESERVAS
1	JUANA PÉREZ	PROFESOR	31000	7	2
2	ROBERTO UMANA	ESTUDIANTE	15000000	7	1
3	SOFIA ABADIA	ESTUDIANTE	1000000	7	1
4	SOFI ABADIA	PADRE	2050000	7	1
5	MAURICIO ORTIZ	EGRESADO	20000000	7	1
6	BEATRIZ GOMEZ	PADRE	2000	7	1
7	JUAN CRISTANCHO	ESTUDIANTE	20000	7	1
8	JUAN RODRIGUEZ	EGRESADO	2150000	7	2
9	DANIELA PARRA	EGRESADO	220000	7	2
10	CLAUDA PÉREZ	EMPLEADO	200000	7	1
11	JUAN CARLOS SÚNCHEZ	PROFESOR	308000	7	2
12	LAURA MARÍA VARGAS	EMPLEADO	100000	7	1
13	ADRIANA GARCIA	EMPLEADO	20200000	7	2
14	JUAN PÉREZ	PADRE	12000	7	1
15	FELIPE PARRA	PROFESOR	2000007	7	1

Figura . Resultado de la consulta

5.1.3 Análisis

Existen 5 tipos de usuario en la comunidad: Padre, Estudiante, Egresado, Profesor y Empleado. Con este método se buscaba conocer la información de cada cliente que perteneciera a esos tipos. Se devolvía el nombre, el tipo de usuario, el dinero pagado, los días reservados y la cantidad de reservas realizadas.

5.3 Analizar la operación de Alohandes

Para una unidad de tiempo definido (por ejemplo, semana o mes) y un tipo de alojamiento, considerando todo el tiempo de operación de AloHandes, indicar cuáles fueron las fechas de mayor demanda (mayor cantidad de alojamientos ocupados), las de mayores ingresos (mayor cantidad de dinero recibido) y las de menor ocupación.

5.3.1 Código

```

SELECT 'HABITACION' AS ALOJAMIENTO, MAX(SUMAS) AS MAXIMO, MEJORPAGADA, MAX(OCU) AS MAXOCU, MEJOROCUPA
FROM( SELECT SUM(RES.COSTOPAGADO) AS SUMAS, RES.FECHADEPAGO AS MEJORPAGADA, COUNT(disponible) AS OCU, res.fechallegada AS MEJOROCUPA
FROM OFERTAHAbitACION OFEHAB, OFERTA OFE, RESERVA RES
WHERE ( OFEHAB.IDOFERTA= OFE.ID AND RES.IDOFERTA= OFE.ID AND ofe.disponible='N')
GROUP BY RES.FECHADEPAGO, res.fechallegada
HAVING(RES.FECHADEPAGO <= '10-ENE-2020' AND RES.FECHADEPAGO>= '28-DIC-2019')
)
WHERE ROWNUM <=5
GROUP BY MEJORPAGADA, MEJOROCUPA

UNION

SELECT 'VIVIENDA' AS ALOJAMIENTO, MAX(SUMAS) AS MAXIMO, MEJORPAGADA, MAX(OCU) AS MAXOCU, MEJOROCUPA
FROM( SELECT SUM(RES.COSTOPAGADO) AS SUMAS, RES.FECHADEPAGO AS MEJORPAGADA, COUNT(disponible) AS OCU, res.fechallegada AS MEJOROCUPA
FROM OFERTAVIVIENDA OFEVIV, OFERTA OFE, RESERVA RES
WHERE ( OFEVIV.IDOFERTA= OFE.ID AND RES.IDOFERTA= OFE.ID AND ofe.disponible='N')
GROUP BY RES.FECHADEPAGO, res.fechallegada
HAVING(RES.FECHADEPAGO <= '10-ENE-2020' AND RES.FECHADEPAGO>= '28-DIC-2019')
)
WHERE ROWNUM<= 5
GROUP BY MEJORPAGADA, MEJOROCUPA

UNION
SELECT 'APARTAMENTO' AS ALOJAMIENTO, MAX(SUMAS) AS MAXIMO, MEJORPAGADA, MAX(OCU) AS MAXOCU, MEJOROCUPA
FROM( SELECT SUM(RES.COSTOPAGADO) AS SUMAS, RES.FECHADEPAGO AS MEJORPAGADA, COUNT(disponible) AS OCU, res.fechallegada AS MEJOROCUPA
FROM OFERTAAPARTAMENTO OFEAPT, OFERTA OFE, RESERVA RES
WHERE ( OFEAPT.IDOFERTA= OFE.ID AND RES.IDOFERTA= OFE.ID AND ofe.disponible='N')
GROUP BY RES.FECHADEPAGO, res.fechallegada
HAVING(RES.FECHADEPAGO <= '10-ENE-2020' AND RES.FECHADEPAGO>= '28-DIC-2019')
)
WHERE ROWNUM<= 5
GROUP BY MEJORPAGADA, MEJOROCUPA
ORDER BY MAXIMO DESC;

```

Figura . Código de la consulta

5.3.2 Resultado

	ALOJAMIENTO	MAXIMO	MEJORPAGADA	MAXOCU	MEJOROCUPA
1	APARTAMENTO	22050000	01-ENE-20	2	01-ENE-20
2	VIVIENDA	15000000	05-ENE-20	1	11-ENE-20
3	HABITACION	2001000	03-ENE-20	2	11-ENE-20
4	APARTAMENTO	2000007	06-ENE-20	1	11-ENE-20
5	HABITACION	300000	01-ENE-20	2	01-ENE-20
6	HABITACION	262000	05-ENE-20	3	11-ENE-20
7	HABITACION	30000	06-ENE-20	1	11-ENE-20
8	HABITACION	20000	30-DIC-19	1	01-ENE-20
9	VIVIENDA	2000	01-ENE-20	1	01-ENE-20

Figura . Resultado de la consulta

5.3.3 Análisis

En este caso lo que se buscaba era la información general de un alojamiento. Se busca la fecha de mayor obtención de dinero y la de mejor ocupación. Para el desarrollo de esta consulta se debía entrar a la oferta de cada alojamiento y se buscaban las ofertas con reservas. Para la máxima cantidad de dinero pagado, se buscaba la fecha de pago de las reservas y el costo pagado; y para la mejor ocupación se buscaba la fecha de llegada de la reserva y la disponibilidad de las ofertas.

5.4 Encontrar los clientes frecuentes

Para un alojamiento dado, encontrar la información de sus clientes frecuentes. se considera frecuente a un cliente si ha utilizado (o tiene reservado) ese alojamiento por lo menos en tres ocasiones o por lo menos 15 noches, durante todo el periodo de operación de AlohaAndes.

5.4.1 Código

```

public List<Cliente> darClientesHabituales ()
{
    PersistenceManager pm = pmf.getPersistenceManager();
    List<Cliente> respuesta = new LinkedList<Cliente> ();
    List<Object> tuplas = sqlResColRes.darClientesHabituales(pm);
    System.out.println(tuplas.size());
    for ( Object tupla : tuplas)
    {
        System.out.println("C02");
        Object [] datos = (Object []) tupla;
        long identificacion = ((BigDecimal) datos [0]).longValue ();
        String tipoid = (String) datos[1];
        String nom = (String) datos[2];
        String tipoc = (String) datos[3];

        Cliente tempCli = new Cliente(identificacion, tipoid, nom, tipoc);
        respuesta.add(tempCli);
    }

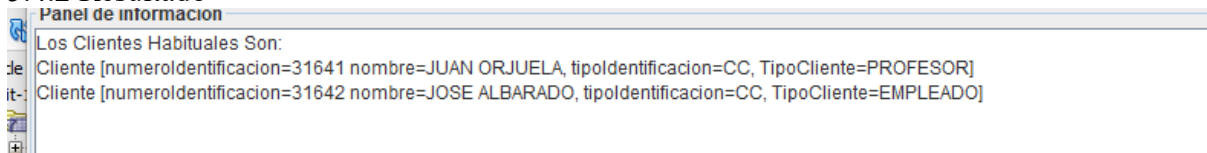
    return respuesta;
}

public List<Object> darClientesHabituales (PersistenceManager pm)
{
    String sql= "";
    sql+=" SELECT SEL1.IDCLIENTE, SEL1.TIPOIDCLIENTE, CLI.NOMBRE, CLI.TIPOCLIENTE ";
    sql+=" FROM ( ";
    sql+=" SELECT IDCLIENTE, TIPOIDCLIENTE, COUNT(IDCLIENTE) ";
    sql+=" FROM RESERVA ";
    sql+=" GROUP BY IDCLIENTE, TIPOIDCLIENTE, IDOFERTA ";
    sql+=" HAVING COUNT(IDCLIENTE)>=3)SEL1, ";
    sql+=" (SELECT IDCLIENTE, TIPOIDCLIENTE ";
    sql+=" FROM RESERVA ";
    sql+=" WHERE FECHADA-FECHALLEGADA <= 15) SEL2, CLIENTE CLI ";
    sql+=" WHERE SEL1.IDCLIENTE = SEL2.IDCLIENTE ";
    sql+=" AND SEL1.TIPOIDCLIENTE = SEL2.TIPOIDCLIENTE ";
    sql+=" AND SEL1.IDCLIENTE = CLI.NUMEROIDENTIFICACION ";
    sql+=" AND SEL1.TIPOIDCLIENTE= CLI.TIPOIDENTIFICACION ";
    sql+=" GROUP BY SEL1.IDCLIENTE, SEL1.TIPOIDCLIENTE, CLI.NOMBRE, CLI.TIPOCLIENTE ";

    Query q = pm.newQuery(SQL, sql);
    System.out.println("CORE01");
    return q.executeList();
}

```

5.4.2 Resultado



5.4.3 Análisis

Para encontrar a los clientes habituales se suman todas las tuplas de RESERVA de un cliente reaccionadas con una misma oferta y se selecciona a aquellos clientes cuyo conteo sea superior a 3. Posteriormente se añaden los datos de los clientes como el nombre y el tipo de cliente mediante la unión con la tabla CLIENTE.

6 Conclusiones

Para el desarrollo de los requerimientos funcionales y los requerimientos funcionales de consulta se continuó haciendo uso de los comandos básicos de SQL, es decir, la inserción, la actualización, la eliminación y la búsqueda. Sin embargo, en esta iteración los requerimientos eran más complejos y en consecuencia las secuencias SQL también fueron más complejas.

7 Bibliografía

[1] Universidad de los Andes. [En línea] [Citado el 03 de mayo de 2020] <https://learn-us-east-1-prod-fleet01-xythos.s3.us-east-1.amazonaws.com/5cdee82dbf7b1/10640679?response-content-disposition=inline%3B%20filename%2A%3DUTF-8%27%27isis2304-201-Iteracion3.pdf&response-content-type=application%2Fpdf&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20200501T175144Z&X-Amz-SignedHeaders=host&X-Amz-Expires=21599&X-Amz->

Credential=AKIAZH6WM4PLTYPZRQMY%2F20200501%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Signature=a80d0ddc26a670d2ebf860152bc0183bf851bf0372e758befe529369c39e722f