“Iteración 4: – Desarrollo del Caso de Estudio de Alohandes”

Lina María Gómez Mesa, Eduardo José Herrera Alba
Iteración 4 del Proyecto de Sistemas Transaccionales
Universidad de los Andes, Bogotá, Colombia
{l.gomez1, ej.herreraa}@uniandes.edu.co
Fecha de presentación: Mayo 28 de 2023

Tabla de contenido

1 Modelo Conceptual - Análisis

1.1 Diagrama de Clases UML

Se realizó la revisión del caso de estudio propuesto: Alohandes. A continuación, se muestra un diagrama de clases de acuerdo a lo identificado en el enunciado. El modelo conceptual tiene integradas las reglas de negocio mediante anotaciones, las clases propuestas que son elementos del mundo del negocio y las cardinalidades de las asociaciones.

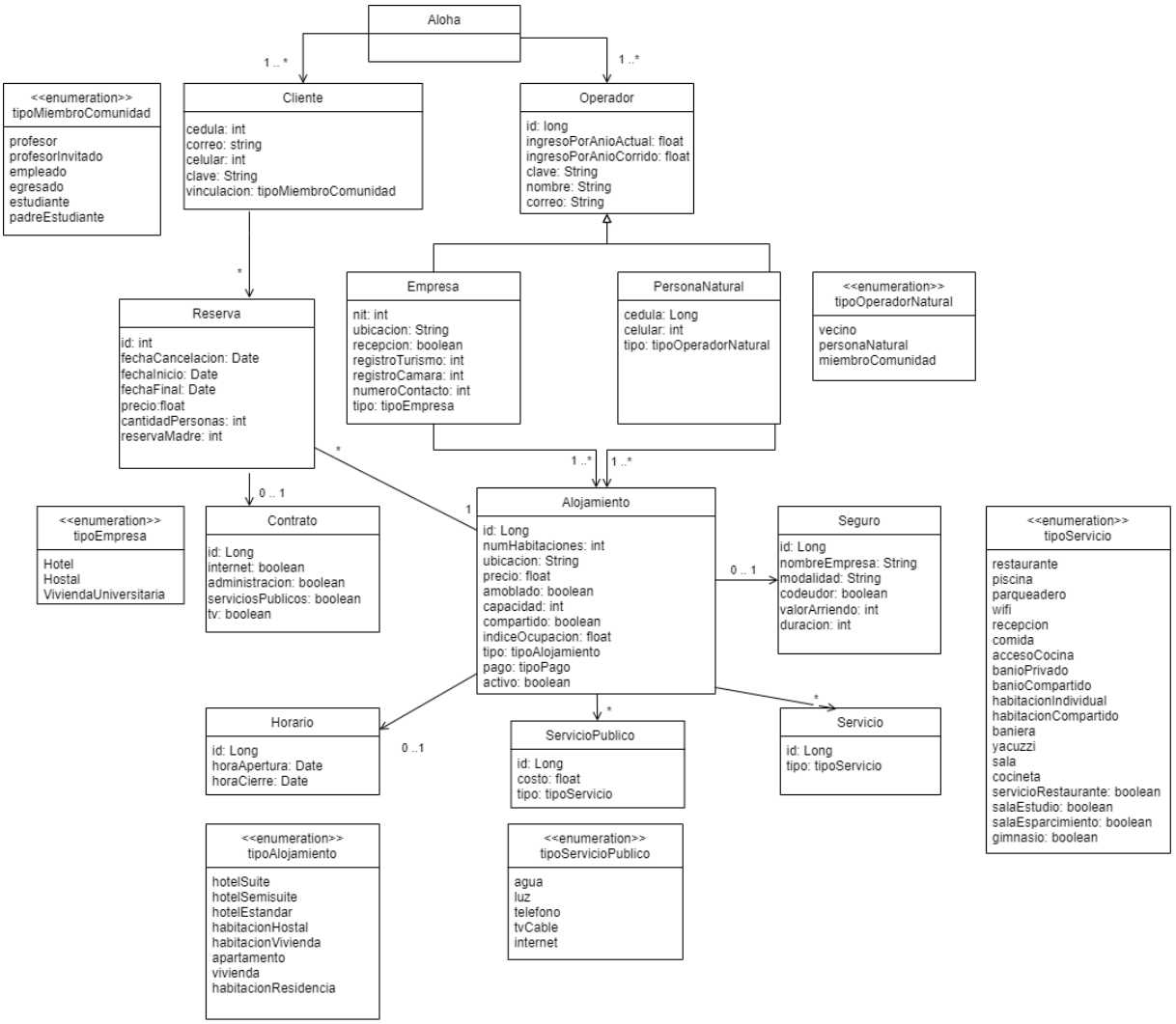


Figura 1. Modelo conceptual completo de 12 clases para el sistema transaccional de AlohaAndes [Elaborado en Diagrams.Net]

Las enumeraciones que aparecen en el diagrama tienen valores que tiene el sistema inicialmente. Sin embargo, es posible agregar nuevos valores en el modelo relacional. Teniendo en cuenta el RNF2, todas las clases deberían ser persistentes, con excepción a Aloha, la cual es una clase de carácter conceptual. Las clases del modelo del mundo que fueron actualizadas para la iteración 4 fueron: *Operador, PersonaNatural y TipoAlojamiento*.

Algunas restricciones de acuerdo al enunciado son:

1. Las empresas deben cumplir con el registro en la cámara de comercio y en la superintendencia de turismo.
2. Los clientes de Alohandes deben tener algún vínculo con la institución: estudiantes, egresados, empleados, profesores, padres de estudiantes, profesores invitados, personas registradas en eventos de Uniandes.
3. Una persona no puede reservar más de un alojamiento en un mismo día.
4. Un alojamiento no acepta reservas que superen su capacidad.
5. El alojamiento en vivienda universitaria sólo está habilitado a estudiantes, profesores, empleados y profesores visitantes.
6. El administrador de la base de datos, el gerente y el cliente son algunos de los usuarios que serán considerados para esta consulta.

1.2 Configuración de la Base de Datos

Se muestra a continuación el modelo de datos relacional que corresponde al modelo conceptual que se propuso. Las tablas de algunas relaciones son demasiado extensas para una sola línea, por lo que se partieron para conservar la legibilidad. El modelo se compone de las siguientes relaciones:

Tabla 1. Relación de TipoMiembroComunidad

TipoMiembroComunidad

Id	Nombre
PK,SA	NN, CK[profesor,empleado,egresado,estudiante,padreEstudiante], ND

Tabla 2. Relación de Cliente

Cliente

Cedula	Correo	Nombre	Celular	Vinculacion	Clave
PK,UA	NN,UA	NN,UA	CK[=10], NN,UA, ND	FK[tipoMiembroComunidad.Id],UA	NN,UA

Tabla 3. Relación de Operador

Operador

Id	IngresoPorAnioActual	IngresoPorAnioCorrido	Nombre	Clave	Correo
PK, UA	CK[>=0], UA	CK[>=0],UA	NN,UA	NN,UA	NN,UA

Tabla 4. Relación de TipoOperadorEmpresa

TipoOperadorEmpresa

Id	Nombre
SA	NN,CK[Hotel, Hostal,ViviendaUniversitaria], UA, ND

Tabla 5. Relación de Empresa

Empresa

Nit	Ubicacion	Recepcion	RegistroTurismo	Registro Camara	NumeroContacto	Tipo
PK, FK[Operador.Id],UA	NN, UA	NN, CK[0,1],UA	NN, CK[=9],UA	NN, CK[=9],UA	NN,CK[=10],UA	FK[TipoOperadorEmpresa.Id], NN, UA

Tabla 6. Relación de TipoOperadorNatural

TipoOperadorNatural

Id	Nombre
PK,SA	CK[Vecino,personaNatural,miembroComunidad], ND, UA

Tabla 7. Relación de PersonaNatural

PersonaNatural

Cedula	Celular	Tipo
PK, FK[Operador.Id],UA	CK[=10], NN, UA	FK[TipoOperadorNatural.Id], UA, NN

Tabla 8. Relación de Contrato

Contrato

Id	internet	administracion	serviciosPublicos	tv
PK, SA	CK[0,1], UA, NN	CK[0,1], UA, NN	CK[0,1], NN, UA	CK[0,1], UA,NN

Tabla 9. Relación de Seguro

Seguro

Id	NombreEmpresa	Modalidad	Codeudor	ValorArriendo	Duracion
PK, SA	NN, UA	NN, UA,	NN, CK[0,1]	NN	NN,

		CK[presencial, digital]			CK[3,6,12]

Tabla 10. Relación de Horario

Horario

Id	HoraApertura	HoraCierre
PK,SA	NN, UA	NN, UA

Tabla 11. Relación de TipoAlojamiento

TipoAlojamiento

Id	Nombre	TipoPago
PK, SA	CK[hotalSuite, hotelSemisuite, hotelEstandar,habitacionHostal,habitacion Vivienda,apartamento,vivienda,habitacion Residencia], UA, ND	CK[mes, dia]

Tabla 12. Relación de Alojamiento

Alojamiento

Id	NumHabitaciones	Ubicacion	Precio	Amoblado	Capacidad
PK	NN, CK[>0]	NN	NN, CK[>0]	NN, CK[0,1]	NN, CK[>0]

Compartido	IndiceOcupacion	Tipo	Horario	Seguro	Operador
NN, CK[0,1]	NN	FK[TipoAlojamiento.Id]	FK[Horario.Id]	FK[Contrato.Id]	FK[Seguro.Id]

Activo
NN, CK[0,1]

Tabla 13. Relación de Reserva

Reserva

Id	fechaCancelacion	fechaInicio	fechaFinal	Alojamiento	Cliente	Contrato	Precio
PK, SA	UA	NN, UA, CK[fechaInicio <fechaFinal]	NN,UA	FK[Alojamiento.Id], UA	FK[Cliente.cedula], UA	FK[Contrato.Id], UA	NN, UA, CK[>=0]

cantidad Personas	reserva Madre
NN	

Tabla 14. Relación de TipoServicioPublico

TipoServicio

Id	Nombre
PK,SA	CK[restaurante, piscina, ... (ver UML), gimnasio], UA

Tabla 15. Relación de ServicioPublico

ServicioPublico

Id	Costo	Tipo	Alojamiento
PK, SA	NN	FK[TipoServicio.Id]	FK[Alojamiento.Id]

Tabla 16. Relación de TipoServicio

TipoServicio

Id	Nombre
PK,SA	CK[restaurante, piscina, ... (ver UML), gimnasio], ND

Tabla 17. Relación de Servicio

Servicio

Id	Tipo	Alojamiento
PK, SA	FK[TipoServicio.Id]	FK[Alojamiento.Id]

Los cambios realizados al modelo fueron: en la relación modelo Operador se añadieron los atributos correo dado que este atributo es mejor que lo tengan tanto la relación Empresa como PersonaNatural.

1.3 Normalización del Modelo Relacional

El modelo relacional planteado se encuentra en el tercer nivel de normalización. A continuación, se justifica nivel por nivel:

Primera forma normal: Un modelo relacional de datos se encuentra en la primera forma normal si todos los atributos de cada una de las relaciones tienen dominios atómicos. El modelo planteado cumple con la primera forma normal dado que no hay atributos multivalor. Esto se puede observar en la sección 3.1 de este documento.

Segunda Forma Normal: Un modelo relacional de datos se encuentra en la segunda forma normal si para cada una de sus relaciones se cumple que no existen dependencias funcionales parciales desde algún atributo primo a un atributo no primo. De acuerdo a lo anterior, el modelo planteado sí cumple con la segunda forma normal porque se encuentra en primera forma normal y no existen dependencias parciales desde los atributos primos; puesto que, las llaves primarias no son compuestas y por lo tanto no hay subconjuntos propios de atributos a los cuales aplicar esta restricción.

Tercera Forma Normal: En tercer lugar, el modelo propuesto cumple con la tercera forma normal debido a que se encuentra en segunda forma normal y no existen dependencias transitivas entre atributos no primos. En el caso de este modelo, los conjuntos de llaves candidatas son de cardinalidad 1, solo tienen la llave primaria que, como se dijo antes, consta de solo un atributo, por lo general un número. Por ende, en este caso particular, todo atributo que no es la llave primaria es un atributo no primo.

Forma normal de Boyce-Codd: Un modelo relacional de datos se encuentra en la cuando no existen dependencias parciales entre atributos primos y además no hay varias llaves candidatas compuestas que no son disjuntas. En el caso de este modelo, el conjunto de llaves candidatas de cada relación consta de solo un atributo: el identificador respectivo (ya sea un id, número de identificación o NIT). Por ende, no hay dependencias parciales entre atributos primos (pues en cada relación hay un único atributo primo, el ya mencionado identificador) y además no hay llaves candidatas compuestas del todo.

2 Diseño de la Aplicación - Lógica Nuevos Requerimientos

2.1 Registrar los operadores de alojamiento para AlohAndes

En la Tabla 1 se muestra el primer requerimiento funcional de AlohAndes. Este se centra en registrar a los operadores de alojamiento para AlohAndes.

Tabla 1. Registrar los operadores de alojamiento para AlohAndes

Nombre	RF1. Registrar a los operadores de alojamiento para AlohAndes

Resumen	El sistema debe permitir el registro de los roles de usuario, tipo operador, definidos por el negocio: empresa (Hotel, Hostal, ViviendaUniversitaria) o personaNatural (vecina, personaNatural, miembroComunidad).
Entradas	
Long: Operador.id	
String: Operador.clave	
Float: Operador.IngresoPorAnioActual	
Float: Operador.IngresoPorAnioCorrido	
String: nombre	
Atributos dependiendo del tipo de Operador creado: Empresa o PersonaNatural	
Resultados	
Enviar Mensaje: String: Registro exitoso	
Creación de: nueva tupla en relación Operador	
Creación de: nueva tupla en relación Empresa o PersonaNatural	
RNF asociados	
Persistencia: AlohAndes debe garantizar que se realice de manera correcta el registro del nuevo operador de alojamientos en la base de datos. Esto es importante ya que esto debería ser poder consultado a futuro y operador no debería tener que volver a registrarse en futuras ocasiones.	
Aislamiento - Concurrencia: AlohAndes debe garantizar que el registro de un nuevo operador de alojamiento sean operaciones que deben ser realizadas aisladamente para que, en caso de que haya varios operadores inscribiéndose al tiempo, no haya problemas de integridad de datos en la base de datos.	

2.2 Registrar propuestas de alojamiento para AlohAndes

En la Tabla 2 se muestra el segundo requerimiento funcional de Alohandes. Este se centra en registrar propuestas de alojamiento para Alohandes.

Tabla 2. Registrar propuestas de alojamiento para Alohandes

Nombre	RF2. Registrar propuestas de alojamiento para Alohandes
Resumen	La aplicación debe permitir el registro de propuestas de alojamiento por parte de los distintos *operadores* (personaNatural o Empresa). Cada propuesta debe incluir información detallada sobre el alojamiento de acuerdo a los atributos solicitados.
Entradas	
Long: Operador.Id	
String: Operador.Clave	
Atributos del Alojamiento que se pueden ver en la Tabla 8 de la sección 3 en la entidad Alojamiento.	
Resultados	
Enviar Mensaje: String: Registro exitoso	
Creación de: nueva tupla en relación Alojamiento	
Creación de: nueva tupla en relaciones que lo necesiten (Horario, Seguro, ServicioPúblico, Servicio).	
RNF asociados	
Atomicidad(transaccionalidad): Se necesita que los registros de las propuestas de alojamiento se realicen por completo o de plano no se realicen. Esto se debe a que se necesita que un alojamiento no puede prescindir de información como su ubicación, capacidad, tipo de alojamiento, entre otros.	
Consistencia(transaccionalidad): Se necesita que el registro de un alojamiento se vea reflejado en la base de datos de una sola forma. No se puede tener un mismo alojamiento con, por ejemplo, 2 ubicaciones diferentes.	

Distribución(enunciado): Se necesita que el administrador de datos tenga contacto con una única base de datos centralizada con la información de los alojamientos. De esta forma, la información que se muestra es igual para los roles de usuario, en especial el administrador.

2.3 Registrar las personas habilitadas para utilizar los servicios

En la Tabla 3 se muestra el tercer requerimiento funcional de Alohandes. Este se centra en registrar las personas habilitadas para utilizar los servicios.

Tabla 3. Registrar propuestas de alojamiento para Alohandes

Nombre	RF3. Registrar a las personas habilitadas para utilizar los servicios
Resumen	El sistema debe permitir el registro de nuevos clientes para que utilicen los servicios proveídos por la aplicación. Cada persona debe incluir información personal necesaria para la inscripción de acuerdo a los atributos solicitados.
Entradas	
Atributos de la relación de cliente que se pueden ver en la Tabla 8 de la sección 3 en la entidad Cliente	
String: Cliente.Clave	
Resultados	
Enviar Mensaje: String: Registro exitoso	
Creación de: nueva tupla en relación Cliente	
RNF asociados	
Persistencia: AlohAndes debe garantizar que se realice de manera correcta el registro del nuevo operador de alojamientos en la base de datos. Esto es importante ya que esto debería ser poder consultado a futuro y el operador no debería tener que volver a registrarse en futuras ocasiones.	
Aislamiento - Concurrencia: AlohAndes debe garantizar que el registro de un nuevo operador de alojamiento sean operaciones que deben ser realizadas aisladamente para que, en caso de que haya varios operadores inscribiéndose al tiempo, no haya problemas de integridad de datos en la base de datos.	

2.4 Registrar una reserva

En la Tabla 4 se muestra el cuarto requerimiento funcional de Alohandes. Este se centra en registrar una reserva de acuerdo a las preferencias del cliente. Por ejemplo, vivienda compartida, internet incluido, cocineta … etc.

Tabla 4. Registrar una Reserva

Nombre	RF4. Registro de una reserva
Resumen	La aplicación debe permitir que los *clientes* de Alohandes puedan buscar alojamiento y reservar una estancia del de su elección. Esta debe considerar las preferencias del cliente. Por ejemplo, vivienda compartida, internet incluido, cocineta, etc.
Entradas	
Long: Cliente.Id	
String: Cliente.Clave	
Atributos de Reserva que se pueden ver en la Tabla 8 de la sección 3 en la entidad Reserva	
Atributos de entidades que se relacionan con una Reserva que lo necesiten (Alojamiento, Servicio, ServioPúblico)	
Resultados	
Enviar Mensaje: String: Registro exitoso	
Creación de: nueva tupla en relación Reserva	

RNF asociados
Durabilidad (transaccionalidad): Para el cumplimiento de este requerimiento funcional se necesita persistencia. Esto se debe a que en el momento en que se genera una reserva se necesita tener la reserva vigente durante el tiempo que dure la reserva para dicho cliente y quitar el alojamiento y su disponibilidad en ese periodo.
Isolation- Concurrencia (Enunciado): Debido a que varios usuarios pueden intentar realizar una reserva al mismo tiempo, es necesario garantizar que la aplicación maneje correctamente la concurrencia para evitar problemas de consistencia en la base de datos.
Privacidad (enunciado): La privacidad es importante para garantizar que los usuarios solo puedan manipular y consultar la información que les es propia, evitando así posibles vulneraciones de seguridad y privacidad.

2.5 Cancelar una Reserva

En la Tabla 5 se muestra el quinto requerimiento funcional de Alohandes. Este se centra en cancelar una reserva.

Tabla 5. Cancelar una Reserva

Nombre	RF5. Cancelar una reserva
Resumen	La aplicación debe permitir que un cliente que ha hecho una reserva en el sistema pueda cancelar dicha reserva antes del inicio de la misma.
Entradas	
Long: Cliente.Id	
Long: Reserva.Id	
String: Cliente.Clave	
Resultados	
Enviar Mensaje: String: Cancelación exitosa	
Deleción de: tupla en relación Reserva	
RNF asociados	
Durabilidad (transaccionalidad): Para el cumplimiento de este requerimiento funcional se necesita persistencia. Esto se debe a que en el momento en que se cancela cierta reserva se necesita que quede guardada en la base de datos para que se pueda consultar la fecha de cancelación en futuras oportunidades.	
Privacidad: Se debe garantizar que el usuario que quiere cancelar una reserva solo pueda acceder a información propia, mas no de otros usuarios, pues se debe garantizar la privacidad de estos.	

2.6 Retirar una oferta de alojamiento

En la Tabla 6 se muestra el sexto requerimiento funcional de Alohandes. Este se centra en cancelar una reserva de acuerdo a las preferencias del cliente. Por ejemplo, vivienda compartida, internet incluido, cocineta … etc.

Tabla 6. Retirar una oferta de Alojamiento

Nombre	RF6. Retirar una oferta de Alojamiento
Resumen	La aplicación debe permitir que los *operadores* de ALOHA puedan retirar una oferta de alojamiento que ya no esté disponible. Este requerimiento implica que la información de las ofertas retiradas se actualice correctamente en la base de datos de la aplicación, garantizando que no se muestran alojamientos que ya no están disponibles para reservar.
Entradas	
Long: Alojamiento.Id	

Atributos de entidades que se relacionan con la entidad Alojamiento que lo necesiten (Contrato,Seguro, Horario)
String: Operador.Clave
Resultados
Enviar Mensaje: String: Cancelación exitosa
Deleción de: tupla en relación Alojamiento
RNF asociados
Persistencia (enunciado): La acción de retirar la oferta de Alojamiento implica una actualización en la base de datos de la aplicación con tal de quitar la oferta y que no se puedan hacer reservas a futuro.
Seguridad (transaccionalidad): Se requiere que los únicos permitidos para retirar ofertas de alojamiento sean los operadores de acuerdo a sus condiciones actuales y disponibilidad. Esto se debe a que estos son los responsables de publicar ofertas.
Durabilidad (transaccionalidad): Una vez que una transacción se ha completado correctamente, los cambios realizados en la base de datos deben persistir a través de fallos del sistema y otros problemas. En especial, que los clientes realicen reservas cuando ya no está disponible el alojamiento.

2.7 Registrar una Oferta Colectiva

En la Tabla 7 se muestra el séptimo requerimiento funcional de Alohandes. Este se centra en crear una reserva colectiva de acuerdo a las preferencias del cliente. Este es únciamente válido para clientes de Alohandes. Una vez este haya ingresado con sus credenciales, se le pide la cantidad de reservas que desea realizar, las fechas de las reservas y se descompone en reservas individuales. Anteriormente, se envían las posibles opciones de alojamiento y se va ubicando cada reserva en uno de ellos. En caso de que se encuentre ocupado, se busca en el siguiente alojamiento. Si recorre todos los alojamientos y no encuentra un posible candidato retorna que no se pudo hacer la reserva.

Tabla 7. Registrar una oferta Colectiva

Nombre	RF7. Registrar una oferta Colectiva
Resumen	En casos de eventos masivos el usuario indica el tipo de alojamiento deseado y la cantidad deseada, y revisa si está en capacidad de satisfacer esa solicitud, eventualmente con varios proveedores, y en caso afirmativo realizar las reservas individuales correspondientes.
Entradas	
Long: Cliente.Id	
String: Cliente.Clave	
Long: cantidadReservas	
LocalDate: Reserva.FechaInicio	
LocalDate: Reserva.FechaFinal	
String: Alojamiento.Tipo	
List<Servicio> serviciosDeseados	
Resultados	
Enviar Mensaje: String: Reserva Exitosa / String: Reserva No Exitosa	
Inserción de: tupla en relación Alojamiento (tanto de la reserva colectiva como cada una de las reservas individuales)	
RNF asociados	
Persistencia (enunciado): La acción de retirar la oferta de Alojamiento implica una actualización en la base de datos de la aplicación con tal de quitar la oferta y que no se puedan hacer reservas a futuro.	
Seguridad/ Privacidad (transaccionalidad): Se requiere que los únicos permitidos para generar ofertas colectivas sean personas que son clientes de Alohandes.	

Durabilidad (transaccionalidad): Una vez que una transacción se ha completado correctamente, los cambios realizados en la base de datos deben persistir a través de fallos del sistema y otros problemas. En especial, que los clientes realicen reservas se conserva tanto las reservas colectivas e individuales en caso de éxito.
Concurrencia: La aplicación se ejecuta en el nivel estándar de Read Committed y utiliza tx.begin, tx.rollback y tx.commit en caso de ser necesario.

2.8 Cancelar Reserva Colectiva

En la Tabla 8 se muestra el octavo requerimiento funcional de Alohandes. Este se centra en cancelar una reserva colectiva de acuerdo a las preferencias del cliente. Inicialmente, se le pide el id de la reserva colectiva y luego, se cancela tanto la individual como la colectiva al ponerles la fecha de cancelación. Se retorna si se pudo "cancelar con éxito".

Tabla 8. Registrar una oferta Colectiva

Nombre	RF8. Cancelar una oferta Colectiva
Resumen	En casos de eventos masivos el usuario indica el id de la reserva colectiva y cancela las subreservas (les pone fecha de cancelación).
Entradas	
Long: Cliente.Id	
String: Cliente.Clave	
Long: Reserva.id (id de la reserva colectiva)	
Resultados	
Enviar Mensaje: String: Cancelación de la reserva exitosa / No exitosa.	
Cancelación de: tupla en relación Reserva (tanto de la reserva colectiva como cada una de las reservas individuales). Se les pone de fechaCancelacion.	
RNF asociados	
Persistencia (enunciado): La acción de retirar la reserva colectiva de Reserva implica una actualización en la base de datos de la aplicación con tal de ponerle fecha de cancelación.	
Seguridad/ Privacidad (transaccionalidad): Se requiere que los únicos permitidos para cancelar reservas colectivas sean personas que son clientes de Alohandes y que fueron quienes crearon dicha reserva.	
Durabilidad (transaccionalidad): Una vez que una transacción se ha completado correctamente, los cambios realizados en la base de datos deben persistir a través de fallos del sistema y otros problemas. En especial, que la reserva ya no aparezca sin fecha de cancelación en el sistema.	
Concurrencia: La aplicación se ejecuta en el nivel estándar de Read Committed y utiliza tx.begin, tx.rollback y tx.commit en caso de ser necesario.	

2.9 Deshabilitar Oferta de Alojamiento

En la Tabla 9 se muestra el noveno requerimiento funcional de Alohandes. Este se centra en deshabilitar una oferta de alojamiento por razones externas como, por ejemplo, remodelación. Inicialmente, se le pide el id del operador y su contraseña, luego, se le pide id del alojamiento que quiere deshabilitar. Como subproceso, se verifica que dicho alojamiento si sea de ese operador, y luego se comienza la transacción dándole prioridad a las reservas vigentes y próximas. Dichas reservas se pasan a alojamientos con el mismo tipoAlojamiento. Se utiliza el requerimiento descrito en la tabla 4 (RF4) para reubicar cada una de las reservas vigentes y próximas y, posteriormente, se cancelan del alojamiento deshabilitado (RF5). En caso de que no se pueda reubicar, se le indica al operador qué reserva no se logró reubicar. Además, en caso de que dicho alojamiento tuviera reservas colectivas se desagregan para poderse reubicar en otro alojamiento.

Tabla 9. Deshabilitar una oferta de alojamiento

Nombre	RF9. Deshabilitar una oferta de alojamiento
Resumen	Por motivos externos una oferta de alojamiento debe cesar su funcionamiento, entonces se reubican las reservas que en dicho momento se encuentran vigentes y próximas a realizarse. En caso de que no se pueda reubicar se le indica al usuario.
Entradas	
Long: Operador.Id	
String: Operador.Clave	
Long: Alojamiento.id (id del alojamiento a deshabilitar)	
Resultados	
Enviar Mensaje: String: Cancelación de la reserva exitosa / No exitosa.	
Reubicación de: tuplas (*) en relación Reserva que pertenecían a ese Alojamiento.id.	
RNF asociados	
Persistencia (enunciado): La acción de deshabilitar un alojamiento implica una actualización en la base de datos de la aplicación con tal de que en la Relación Alojamiento se cambie el parámetro de Alojamiento.Activo por 0 de inactivo.	
Seguridad/ Privacidad (transaccionalidad): Se requiere que los únicos permitidos para deshabilitar alojamientos sean personas que son operadores de Alohandes y que son dueños de dicho alojamiento en cuestión.	
Durabilidad (transaccionalidad): Una vez que una transacción se ha completado correctamente, los cambios realizados en la base de datos deben persistir a través de fallos del sistema y otros problemas. En especial, que el alojamiento ya esté inactivo y las reservas hayan sido cambiadas a otro alojamiento.	
Concurrencia: La aplicación se ejecuta en el nivel estándar de Read Committed y utiliza tx.begin, tx.rollback y tx.commit en caso de ser necesario.	

2.10 Rehabilitar Oferta de Alojamiento

En la Tabla 10 se muestra el décimo requerimiento funcional de Alohandes. Este se centra en rehabilitar una oferta de alojamiento que se encuentra en el momento inactiva. Inicialmente, se le pide el id del operador y su contraseña, luego, se le pide id del alojamiento que quiere volver a habilitar. Como subproceso, se verifica que dicho alojamiento si sea de ese operador, y luego se comienza la transacción para poder update el parámetro de Alojamiento.Activo a 1.

Tabla 10. Rehabilitar una oferta de alojamiento

Nombre	RF10. Deshabilitar una oferta de alojamiento
Resumen	Ocurre cuando la oferta de alojamiento vuelve a estar disponible y puede por lo tanto aceptar nuevas reservas.
Entradas	
Long: Operador.Id	
String: Operador.Clave	
Long: Alojamiento.id (id del alojamiento a rehabilitar)	
Resultados	
Enviar Mensaje: String: Rehabilitación del alojamiento (se debe entregar un 1 de activo)	
RNF asociados	
Persistencia (enunciado): La acción de rehabilitar reservas de un alojamiento implica una actualización en la base de datos de la aplicación con tal de que en la Relación Alojamiento se cambie el parámetro de Alojamiento.activo por el nuevo alojamiento al que pertenece cada reserva individual.	

Seguridad/ Privacidad (transaccionalidad): Se requiere que los únicos permitidos para rehabilitar alojamientos sean personas que son operadores de Alohandes y que son dueños de dicho alojamiento en cuestión.
Durabilidad (transaccionalidad): Una vez que una transacción se ha completado correctamente, los cambios realizados en la base de datos deben persistir a través de fallos del sistema y otros problemas. En especial, que la reserva ya no aparezca sin fecha de cancelación en el sistema.
Concurrencia: La aplicación se ejecuta en el nivel estándar de Read Committed y utiliza tx.begin, tx.rollback y tx.commit en caso de ser necesario.

2.11 Mostrar el dinero recibido por cada proveedor (RFC1)

En la Tabla 11 se muestra el primer requerimiento funcional de consulta de Alohandes. Este se centra en mostrar el dinero recibido por cada proveedor de alojamiento durante el año actual y el año corrido.

Tabla 11. Mostrar el dinero recibido por cada proveedor de alojamiento durante el año actual y el año corrido

Nombre	RFC1. Mostrar el dinero recibido por cada proveedor de alojamiento durante el año actual y el año corrido
Resumen	La aplicación debe poder mostrar el dinero que cada proveedor registrado en la base de datos ha recibido en el año actual y en el año corrido
Entradas	
Long: Operador.Id	
String: Operador.Clave	
Resultados	
Enviar Mensaje: String: valores de dinero del año actual y del año corrido	
RNF asociados	
Concurrencia: se debe garantizar que varios usuarios de la base de datos puedan acceder simultáneamente a la información y no haya problemas de concurrencia	
Persistencia (transaccionalidad): La información del dinero recibido en el año corrido y en el año actual de los operadores debe ser persistente para poder ser consultada en cualquier momento y por los usuarios permitidos (clientes o operadores).	
Integridad: La información de los operadores debe estar protegida contra posibles modificaciones no autorizadas para mantener la integridad de los datos.	

2.12 Mostrar las 20 ofertas más populares (RFC2)

En la Tabla 12 se muestra el segundo requerimiento funcional de consulta de Alohandes. Este se centra en mostrar las 20 ofertas más populares.

Tabla 8. Mostrar las 20 ofertas más populares

Nombre	RFC2. Mostrar las 20 ofertas más populares
Resumen	El sistema debe permitir la operación de consulta por parte de los *cliente*s y *operadores* de Alohandes. Esta se centra en consultar las 20 ofertas más populares de Alojamiento de acuerdo a la cardinalidad de la tabla Reserva.
Entradas	
Long: Operador.Id o Long: Cliente.Id	
String: Cliente.Clave o String: Operador.Clave	
Resultados	

Enviar Mensaje: List<Alojamiento>: ciertos atributos de las tuplas seleccionadas de la relación Alojamiento y el operador dueño
RNF asociados
Concurrencia (enunciado): Pueden hacerse más de una consulta de este tipo al tiempo.
Persistencia (transaccionalidad): La información de las ofertas y su popularidad debe ser persistente para poder ser consultada en cualquier momento y por los usuarios permitidos (clientes o operadores).
Integridad: La información de las ofertas y su popularidad debe estar protegida contra posibles modificaciones no autorizadas para mantener la integridad de los datos. Por lo tanto, esta debe concordar con el número de reservas que ha tenido cada alojamiento.

2.13 Mostrar el Índice de Ocupación (RFC3)

En la Tabla 13 se muestra el tercer requerimiento funcional de consulta de AlohAndes. Este se centra en mostrar el índice de Ocupación de cada una de las ofertas de alojamiento registradas.

Tabla 13. Mostrar el Índice de Ocupación de cada una de las ofertas de alojamiento registradas

Nombre	RFC3. Mostrar el índice de ocupación de cada una de las ofertas de alojamiento registradas
Resumen	El sistema debe permitir la operación de consulta por parte de los *cliente*s y *operadores* de AlohAndes. Esta se centra en consultar el índice de ocupación de las ofertas de alojamiento registradas en el sistema.
Entradas	
Long: Operador.Id o Long: Cliente.Id	
String: Cliente.Clave o String: Operador.Clave	
Resultados	
Enviar Mensaje: List<string>: valores de indiceOcupacion de todas las tuplas de la relación Alojamiento	
RNF asociados	
Concurrencia: se debe garantizar que varios usuarios de la base de datos puedan acceder simultáneamente a la información del índice de ocupación de los operadores y no haya problemas de concurrencia.	
Persistencia (transaccionalidad): La información del índice de ocupación de los operadores debe ser persistente para poder ser consultada en cualquier momento y por los usuarios permitidos (clientes o operadores).	
Integridad: La información de los operadores debe estar protegida contra posibles modificaciones no autorizadas para mantener la integridad de los datos.	

2.14 Mostrar los alojamientos disponibles (RFC4)

En la Tabla 14 se muestra el cuarto requerimiento funcional de consulta de AlohAndes. Este se centra en mostrar los alojamientos disponibles en un rango de fechas, que cumplen con un conjunto de requerimientos de dotación o servicios.

Tabla 14. Mostrar los alojamientos disponibles en un rango de fechas, que cumplen con un conjunto de requerimientos de dotación o servicios

Nombre	RFC4. Mostrar los alojamientos disponibles en un rango de fechas, que cumplen con un conjunto de requerimientos de dotación o servicios.
Resumen	La aplicación debe permitir a los *clientes* buscar alojamiento según un conjunto de criterios, como la ubicación, las fechas de estancia, la capacidad, la dotación o servicios incluidos, entre otros. La aplicación debe mostrar una lista de alojamientos disponibles que cumplan con esos criterios.

Entradas
Date: fechaInicial
Date: fechaFinal
Float: precioMinimo
Float: precioMaximo
List<Servicio> o List<ServicioPublico>
Long: Cliente.Id
String: Cliente.Clave
Resultados
Enviar Mensaje: List<Alojamiento>: ciertos atributos de las tuplas seleccionadas de la relación Alojamiento y el operador dueño (ej. Alojamiento.Ubicacion, Operador.Nombre).
RNF asociados
Concurrencia: Pueden hacerse más de una consulta de este tipo al tiempo.
Seguridad: Los usuarios deben tener permisos adecuados para acceder a la información de alojamientos y disponibilidad, y los datos deben ser protegidos contra acceso no autorizado. Es decir, personas que no sean clientes o empresas relacionadas con la Universidad de los Andes.
Usabilidad: La interfaz de usuario debe ser clara, sencilla e intuitiva para facilitar la búsqueda de alojamientos por parte de los *clientes*.

2.15 Mostrar el uso de AlohAndes para cada tipo de usuario (RFC5)

En la Tabla 15 se muestra el quinto requerimiento funcional de consulta de AlohAndes. Este se centra en mostrar el uso de alohandes para cada tipo de usuario de la comunidad. Se le pregunta al usuario por el tipoMiembroComunidad del cual le gustaría ver la actividad y se traen todas las reservas hechas por ese tipoMiembroComunidad. Entiéndase uso como el uso de los alojamientos, es decir las reservas. En este requerimiento, se realiza un inner join entre las tablas reservas y alojamiento para poder seleccionar todas las reservas que se asocian con ese tipoMiembroComunidad.

Tabla 15. Mostrar el uso de alohandes para cada tipo de usuario de la comunidad

Nombre	RFC5. Mostrar el uso de alohandes para cada tipo de usuario de la comunidad
Resumen	La aplicación debe permitir a los clientes y operadores mostrar el uso que se le da a la aplicación dependiendo del tipo de usuario de la comunidad.
Entradas	
String: TipoMiembroComunidad.Nombre	
Resultados	
Enviar Mensaje: List<Reserva>: ciertos atributos de las tuplas seleccionadas de la relación Alojamiento join Reserva para ese tipoMiembroComunidadSeleccionado.	
RNF asociados	
Concurrencia: se debe garantizar que varios usuarios de la base de datos puedan acceder simultáneamente a la información del uso de la aplicación de los usuarios según su tipo de usuario de la comunidad y no haya problemas de concurrencia.	
Persistencia (transaccionalidad): La información del uso de la aplicación de los usuarios según su tipo debe ser persistente para poder ser consultada en cualquier momento y por los usuarios permitidos (clientes o operadores).	
Integridad: La información de los operadores debe estar protegida contra posibles modificaciones no autorizadas para mantener la integridad de los datos.	

2.16 Mostrar el uso de Alohandes para un usuario dado (RFC6)

En la Tabla 16 se muestra el sexto requerimiento funcional de consulta Alohandes. Este se muestra el uso de Alohandes para un usuario dado. En primer lugar, se pide una verificación de que dicho usuario que quiere hacer esa consulta es un usuario de alohandes. Posteriormente, una vez se ha hecho la verificación, se traen todas las reservas de dicho cliente. Se le muestra el número de noches [días] por reserva que fueron contratados, dinero pagado por reserva y algunas características del alojamiento contratado como si está amoblado o no y la dirección del alojamiento. Es importante recalcar que no se consideran las reservas que fueron canceladas ya que no ocurrieron.

Tabla 16. Mostrar el uso de Alohandes para un usuario dado (número de noches o meses contratados, características del alojamiento utilizado, dinero pagado.

Nombre	RFC6. Mostrar el uso de Alohandes para un usuario dado (número de noches o meses contratados, características del alojamiento utilizado, dinero pagado.
Resumen	Este requerimiento consiste en mostrar el uso de la plataforma por parte de un usuario específico, lo que incluye el número de noches o meses que ha contratado, las características del alojamiento que ha utilizado y el dinero que ha pagado por ello. Este requerimiento tiene como objetivo brindar al usuario información detallada sobre su historial de uso de AlohAndes.
Entradas	
Long: Cliente.Id	
String: Cliente.Clave	
Resultados	
Enviar Mensaje: List<Object>: ciertos atributos de las tuplas seleccionadas de la relación Reserva,Alojamiento [Alojamiento.Id, Alojamiento.Ubicacion, Alojamiento.Amoblado,Alojamiento.Precio, Reserva.FECHAINICIO, Reserva.FECHAFIN, TotalNoches]	
RNF asociados	
Privacidad (enunciado): La información mostrada debe ser sólo accesible para el cliente que la solicita y no debe ser visible para otros usuarios. De tal manera, que el historial de reservas de un usuario solo puede ser accedido por ese usuario particular.	
Persistencia (transaccionalidad): La información mostrada debe ser obtenida de manera persistente desde la base de datos y no debe perderse en caso de fallos del sistema. Esta debe ser poder consultada en cualquier momento y únicamente por los usuarios cliente o operador.	
Concurrencia: Se debe permitir que varios usuarios puedan consultar su historial de reservas de alojamientos al mismo tiempo en Alohandes. Por tanto, no debe existir problemas de concurrencia.	

2.17 Analizar la Operación de Alohandes (RFC7)

En la Tabla 17 se muestra el séptimo requerimiento funcional de consulta Alohandes.El RFC8 involucra tres búsquedas a la base de datos que contribuyen al análisis de la operación de Alohandes. La primera involucra encontrar la fecha de mayores ingresos, mientras que las otras dos son búsquedas análogas: encontrar la fecha de mayor ocupación y la fecha de menor ocupación. Para este requerimiento se definió que la búsqueda se haría mensualmente, por lo que la respuesta del requerimiento es una fecha que incluye solamente mes y año (mm-yyyy). Por otro lado, el tipo de alojamiento se pide por consola al usuario.

La lógica de la primera búsqueda empieza buscando las reservas individuales y colectivas del tipo de alojamiento dado, para después añadir el precio del mes de inicio de cada reserva a un *HashMap<String, Long>,* donde se almacenan las fechas encontradas en el formato mencionado anteriormente y los ingresos acumulados en dicha fecha. Posteriormente, se encuentra la fecha de mayor ingreso recorriendo dicho mapa.

La lógica de las otras dos búsquedas consiste en encontrar las reservas individuales y subreservas del tipo de alojamiento solicitado, para luego recorrer estas reservas encontradas y añadirlas a un *HashMap<String, HashSet<Integer>>,* donde se almacenan las fechas en el formato y los id de los alojamientos con reservas en dicha fecha. Finalmente se busca en el mapa la fecha con mayor y menor cantidad de alojamientos.

Tabla 17. Mostrar la operación de Alohandes para fecha de mayores ingresos, mayor ocupación y menor ocupación.

Nombre	RFC7. Analizar la operación de Alohandes
Resumen	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.
Entradas	
String: TipoDeAlojamiento.Nombre	
Resultados	
Enviar Mensaje: String[]: Se retorna la fecha de mayores ingresos, mayor ocupación y menor ocupación.	
RNF asociados	
Persistencia (transaccionalidad): La información mostrada debe ser obtenida de manera persistente desde la base de datos y no debe perderse en caso de fallos del sistema. Esta debe ser poder consultada en cualquier momento y únicamente por los usuarios cliente o operador.	
Concurrencia: Se debe permitir que varios usuarios puedan consultar su historial de reservas de alojamientos al mismo tiempo en Alohandes. Por tanto, no debe existir problemas de concurrencia.	

2.18 Encontrar los clientes frecuentes (RFC8)

En la Tabla 18 se muestra el octavo requerimiento funcional de consulta Alohandes. En este se muestra los clientes frecuentes para un alojamiento (más de 3 reservas o más de 15 noches). En primer lugar, se pide el id del alojamiento de interés y posteriormente se hacen dos subconsultas anidadas en una mayor. Primero, se buscan los clientes que tienen más de tres reservas activas; es decir, no están canceladas y, en segundo lugar, los clientes que tengan una reserva cuya fechafinal menos la fecha de inicio sea mayor a 15. Se realiza una unión entre ambas consultas y se retorna la información de los clientes que cumplen con ello (a excepción de su clave).

Tabla 18. Mostrar los clientes frecuentes para un alojamiento

Nombre	RFC8. Encontrar los clientes frecuentes
Resumen	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 AlohAndes
Entradas	
Long: Alojamiento.Id	
Resultados	

Enviar Mensaje: List<Cliente>: ciertos atributos de las tuplas seleccionadas de la relación Cliente.
RNF asociados
Persistencia (transaccionalidad): La información mostrada debe ser obtenida de manera persistente desde la base de datos y no debe perderse en caso de fallos del sistema. Esta debe ser poder consultada en cualquier momento y únicamente por los usuarios cliente o operador.
Concurrencia: Se debe permitir que varios usuarios puedan consultar su historial de reservas de alojamientos al mismo tiempo en Alohandes. Por tanto, no debe existir problemas de concurrencia.

2.19 Encontrar las Ofertas de Alojamiento que no tengan mucha demanda (RFC9)

En la Tabla 19 se muestra el noveno requerimiento funcional de consulta Alohandes. En este se muestra las ofertas con poca demanda. Se entiende como poca demanda a aquellos alojamientos que hayan pasado al menos un periodo de 30 días sin reservas. Para ello, se consideró tres subqueries. La primera buscaba entre las reservas de un alojamiento la última reserva y compara con la fecha actual si han pasado más de 30 días se añade ese alojamiento a la lista de poca demanda. En segundo lugar, se considera entre todas las fechas de reserva de un alojamiento si entre reservas hay más de treinta días. Si esto es afirmativo, entonces también se añade a la lista de alojamientos con poca demanda. Finalmente, si un alojamiento no tiene reservas entonces también se añade a la lista de alojamientos con poca demanda.

Tabla 19. Mostrar ofertas con poca demanda

Nombre	RFC9. Mostrar ofertas con poca demanda
Resumen	Encontrar 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	
Enviar Mensaje: List<Alojamiento>: ciertos atributos de las tuplas seleccionadas de la relación Alojamiento.	
RNF asociados	
Persistencia (transaccionalidad): La información mostrada debe ser obtenida de manera persistente desde la base de datos y no debe perderse en caso de fallos del sistema. Esta debe ser poder consultada en cualquier momento y únicamente por los usuarios cliente o operador.	
Concurrencia: Se debe permitir que varios usuarios puedan consultar su historial de reservas de alojamientos al mismo tiempo en Alohandes. Por tanto, no debe existir problemas de concurrencia.	

2.20 Consultar Consumo en Alohandes (RFC10)

En la Tabla 20 se muestra el décimo requerimiento funcional de consulta Alohandes. En este se muestra el consumo para Alohandes para los usuarios que tienen al menos una reserva para un rango de fechas. Esta se divide en dos partes. Si el usuario es el DBA (administrador de la base datos) se muestran todos los usuarios que cumplen con dicha condición. De lo contrario, solo se muestran las reservas para ese cliente de acuerdo al alojamiento seleccionado.

Tabla 20. Consultar Consumo en Alohandes

Nombre	RFC10. Consultar Consumo en Alohandes

Resumen	Conocer la información de los usuarios que realizaron al menos una reserva de una determinada oferta de alojamiento en un rango de fechas.Los resultados deben ser clasificados según un criterio deseado por quien realiza la consulta.
Entradas	
String: Cliente.Cedula	
String: Cliente.Clave/ Admin.Clave	
String: Alojamiento.id	
String: Reserva.FechaInicio	
String: Reserva.FechaFinal	
String: Criterio por el cual ordenar	
Resultados	
Enviar Mensaje: List<Cliente>: ciertos atributos de las tuplas seleccionadas de la relación Cliente.	
RNF asociados	
Persistencia (transaccionalidad): La información mostrada debe ser obtenida de manera persistente desde la base de datos y no debe perderse en caso de fallos del sistema. Esta debe ser poder consultada en cualquier momento y únicamente por los usuarios cliente o operador.	
Concurrencia: Se debe permitir que varios usuarios puedan consultar su historial de reservas de alojamientos al mismo tiempo en Alohandes. Por tanto, no debe existir problemas de concurrencia.	
Privacidad: Si es usuario Admin o cliente se muestran distintas vistas.	

2.21 Consultar Consumo en Alohandes - V2 (RFC11)

En la Tabla 21 se muestra el onceavo requerimiento funcional de consulta Alohandes. En este se muestra el consumo para Alohandes para los usuarios que no tienen ninguna reserva para un rango de fechas y un alojamiento específico. Esta se divide en dos partes. Si el usuario es el DBA (administrador de la base datos) se muestran todos los usuarios que cumplen con dicha condición. De lo contrario, se le especifica al usuario si no hizo ninguna reserva.

Tabla 21. Consultar Consumo en Alohandes - V2

Nombre	RFC10. Consultar Consumo en Alohandes
Resumen	Conocer la información de los usuarios que no realizaron ninguna reserva de una determinada oferta de alojamiento en un rango de fechas.Los resultados deben ser clasificados según un criterio deseado por quien realiza la consulta.
Entradas	
String: Cliente.Cedula	
String: Cliente.Clave/ Admin.Clave	
String: Alojamiento.id	
String: Reserva.FechaInicio	
String: Reserva.FechaFinal	
String: Criterio por el cual ordenar	
Resultados	
Enviar Mensaje: List<Cliente>: ciertos atributos de las tuplas seleccionadas de la relación Cliente.	
RNF asociados	

Persistencia (transaccionalidad): La información mostrada debe ser obtenida de manera persistente desde la base de datos y no debe perderse en caso de fallos del sistema. Esta debe ser poder consultada en cualquier momento y únicamente por los usuarios cliente o operador.
Concurrencia: Se debe permitir que varios usuarios puedan consultar su historial de reservas de alojamientos al mismo tiempo en Alohandes. Por tanto, no debe existir problemas de concurrencia.
Privacidad: Si es usuario Admin o cliente se muestran distintas vistas.

2.22 Consultar Funcionamiento en Alohandes (RFC12)

En la Tabla 22 se muestra el doceavo requerimiento funcional de consulta Alohandes. Este muestra, para cada semana del año, la oferta de alojamiento con más ocupación, la oferta de alojamiento con menos ocupación, los operadores más solicitados y los operadores menos solicitados. Las respuestas deben ser sustentadas por el detalle de las ofertas de alojamiento y operadores correspondientes. Esta operación es realizada por el gerente general de AlohAndes. Por esta razón, se dividió en cuatro subconsultas que realizan cada una de los requerimientos solicitados.

Tabla 22. Consultar Consumo en Alohandes

Nombre	RFC12. Consultar Funcionamiento en Alohandes
Resumen	Conocer la oferta de alojamiento con más ocupación, la oferta de alojamiento con menos ocupación, los operadores más solicitados y los operadores menos solicitados.
Entradas	
String: Anio	
Resultados	
Enviar Mensaje: List<String>: ciertos atributos de las tuplas seleccionadas de tanto los operadores o alojamientos.	
RNF asociados	
Persistencia (transaccionalidad): La información mostrada debe ser obtenida de manera persistente desde la base de datos y no debe perderse en caso de fallos del sistema. Esta debe ser poder consultada en cualquier momento y únicamente por los usuarios cleinte o operador.	
Concurrencia: Se debe permitir que varios usuarios puedan consultar su historial de reservas de alojamientos al mismo tiempo en Alohandes. Por tanto, no debe existir problemas de concurrencia.	
Privacidad: Solo está disponible para el CEO de Alohandes.	

2.23 Consultar Funcionamiento en Alohandes (RFC13)

En la Tabla 23 se muestra el doceavo requerimiento funcional de consulta Alohandes. Este muestra, para cada semana del año, la oferta de alojamiento con más ocupación, la oferta de alojamiento con menos ocupación, los operadores más solicitados y los operadores menos solicitados. Las respuestas deben ser sustentadas por el detalle de las ofertas de alojamiento y operadores correspondientes. Esta operación es realizada por el gerente general de AlohAndes.

Tabla 23. Consultar Consumo en Alohandes

Nombre	RFC12. Consultar Funcionamiento en Alohandes

Resumen	Conocer la oferta de alojamiento con más ocupación, la oferta de alojamiento con menos ocupación, los operadores más solicitados y los operadores menos solicitados.
Entradas	
Resultados	
Enviar Mensaje: List<Alojamiento>: ciertos atributos de las tuplas seleccionadas de la relación Alojamiento.	
RNF asociados	
Persistencia (transaccionalidad): La información mostrada debe ser obtenida de manera persistente desde la base de datos y no debe perderse en caso de fallos del sistema. Esta debe ser poder consultada en cualquier momento y únicamente por los usuarios cleinte o operador.	
Concurrencia: Se debe permitir que varios usuarios puedan consultar su historial de reservas de alojamientos al mismo tiempo en Alohandes. Por tanto, no debe existir problemas de concurrencia.	

3 Diseño Físico

	OWNER	INDEX_NAME	INDEX_TYPE	TABLE_OWNER	TABLE_NAME	TABLE_TYPE	UNIQUENESS	COMPRESSION	PREFIX_LENGTH	TABLESPACE_NAME
1	IS2304A14202310	PK_RESERVAS	NORMAL	ISIS2304A14202310	RESERVAS	TABLE	UNIQUE	DISABLED	(null)	TBSPROD
2	IS2304A14202310	PK_SILLASRESERVAS	NORMAL	ISIS2304A14202310	SILLASRESERVAS	TABLE	UNIQUE	DISABLED	(null)	TBSPROD
3	IS2304A14202310	TIPOMIEMBROCOMUNIDAD_PK	NORMAL	ISIS2304A14202310	TIPOMIEMBROCOMUNIDAD	TABLE	UNIQUE	DISABLED	(null)	TBSPROD
4	IS2304A14202310	UN_TIPO_TIPOMIEMBROCOMUNIDAD	NORMAL	ISIS2304A14202310	TIPOMIEMBROCOMUNIDAD	TABLE	UNIQUE	DISABLED	(null)	TBSPROD
5	IS2304A14202310	CLIENTE_PK	NORMAL	ISIS2304A14202310	CLIENTE	TABLE	UNIQUE	DISABLED	(null)	TBSPROD
6	IS2304A14202310	UN_TIPOCEL_CLIENTE	NORMAL	ISIS2304A14202310	CLIENTE	TABLE	UNIQUE	DISABLED	(null)	TBSPROD
7	IS2304A14202310	OPERADOR_PK	NORMAL	ISIS2304A14202310	OPERADOR	TABLE	UNIQUE	DISABLED	(null)	TBSPROD
8	IS2304A14202310	TIPOOPERADOREMPRESA_PK	NORMAL	ISIS2304A14202310	TIPOOPERADOREMPRESA	TABLE	UNIQUE	DISABLED	(null)	TBSPROD
9	IS2304A14202310	UN_TIPO_TIPOOPERADOREMPRESA	NORMAL	ISIS2304A14202310	TIPOOPERADOREMPRESA	TABLE	UNIQUE	DISABLED	(null)	TBSPROD
10	IS2304A14202310	EMPRESA_PK	NORMAL	ISIS2304A14202310	EMPRESA	TABLE	UNIQUE	DISABLED	(null)	TBSPROD
11	IS2304A14202310	TIPOOPERADORNATURAL_PK	NORMAL	ISIS2304A14202310	TIPOOPERADORNATURAL	TABLE	UNIQUE	DISABLED	(null)	TBSPROD
12	IS2304A14202310	UN_TIPO_TIPOOPERADORNATURAL	NORMAL	ISIS2304A14202310	TIPOOPERADORNATURAL	TABLE	UNIQUE	DISABLED	(null)	TBSPROD
13	IS2304A14202310	PERSONANATURAL_PK	NORMAL	ISIS2304A14202310	PERSONANATURAL	TABLE	UNIQUE	DISABLED	(null)	TBSPROD
14	IS2304A14202310	CONTRATO_PK	NORMAL	ISIS2304A14202310	CONTRATO	TABLE	UNIQUE	DISABLED	(null)	TBSPROD
15	IS2304A14202310	SEGURO_PK	NORMAL	ISIS2304A14202310	SEGURO	TABLE	UNIQUE	DISABLED	(null)	TBSPROD
16	IS2304A14202310	HORARIO_PK	NORMAL	ISIS2304A14202310	HORARIO	TABLE	UNIQUE	DISABLED	(null)	TBSPROD
17	IS2304A14202310	TIPOALOJAMIENTO_PK	NORMAL	ISIS2304A14202310	TIPOALOJAMIENTO	TABLE	UNIQUE	DISABLED	(null)	TBSPROD
18	IS2304A14202310	UN_TIPO_TIPOALOJAMIENTO	NORMAL	ISIS2304A14202310	TIPOALOJAMIENTO	TABLE	UNIQUE	DISABLED	(null)	TBSPROD
19	IS2304A14202310	ALOJAMIENTO_PK	NORMAL	ISIS2304A14202310	ALOJAMIENTO	TABLE	UNIQUE	DISABLED	(null)	TBSPROD
20	IS2304A14202310	RESERVA_PK	NORMAL	ISIS2304A14202310	RESERVA	TABLE	UNIQUE	DISABLED	(null)	TBSPROD
21	IS2304A14202310	TIPOSERVICIOPUBLICO_PK	NORMAL	ISIS2304A14202310	TIPOSERVICIOPUBLICO	TABLE	UNIQUE	DISABLED	(null)	TBSPROD
22	IS2304A14202310	UN_TIPO_TIPOSERVICIOPUBLICO	NORMAL	ISIS2304A14202310	TIPOSERVICIOPUBLICO	TABLE	UNIQUE	DISABLED	(null)	TBSPROD
23	IS2304A14202310	SERVICIOPUBLICO_PK	NORMAL	ISIS2304A14202310	SERVICIOPUBLICO	TABLE	UNIQUE	DISABLED	(null)	TBSPROD

Figura 3. Índices Iniciales de SQL Developer para la consulta 10

En la figura 3 se observan los índices predeterminados realizados por Oracle SQL Developer. Estas son las llaves primarias de cada una de las tablas creadas y las condiciones de restricción impuestas sobre el modelo relacional.

3.1 RFC- 10: Consumo en Alohandes

Selección de Índices

```
-- RF10: CLIENTE
CREATE INDEX idx_reserva_cliente_fechainicio_fechafinal_fechacancelacion_alojamiento
ON RESERVA (CLIENTE, FECHAINICIO, ALOJAMIENTO);

-- Para la consulta RFC10: ADMIN
CREATE INDEX idx_reserva_fechainicio_fechafinal_fechacancelacion_alojamiento
ON RESERVA (FECHAINICIO, ALOJAMIENTO);

CREATE INDEX idx_alojamiento
ON RESERVA (ALOJAMIENTO);
```

Figura 4. Índices Iniciales de SQL Developer para la consulta 10

Para esta requerimiento funcional de consulta se decidieron crear tres índices compuestos basados en cliente, fechaInicio y alojamiento. Estos son los atributos que se encuentran en las sentencias tipo WHERE y de ORDENAMIENTO por lo que crear índices sobre estos atributos disminuye el tiempo de búsqueda en la base de datos. Inicialmente, por defecto, en Oracle SQL Developer los índices se generan como índices tipo B+. Estos son efectivos dado que los árboles B+ crecen a lo ancho (de manera horizontal) por lo que son eficientes a la hora de llegar a un argumento en particular. Además, este tipo de índice no reorganiza totalmente el archivo por lo que automáticamente se reorganiza con cambios pequeños y locales en las inserciones y borrados.

Índices Creados Automáticamente Por Oracle

Los índices creados se encuentran en la parte superior en la figura 4. Estos son la llave primaria Cliente.Cedula de la relación Cliente. En general, al ejecutar esta consulta Oracle utiliza por defecto el índice sobre la llave primaria Cliente.id dado que es el índice primario sobre la tabla Cliente. Esto se muestra en el plan de ejecución abajo.

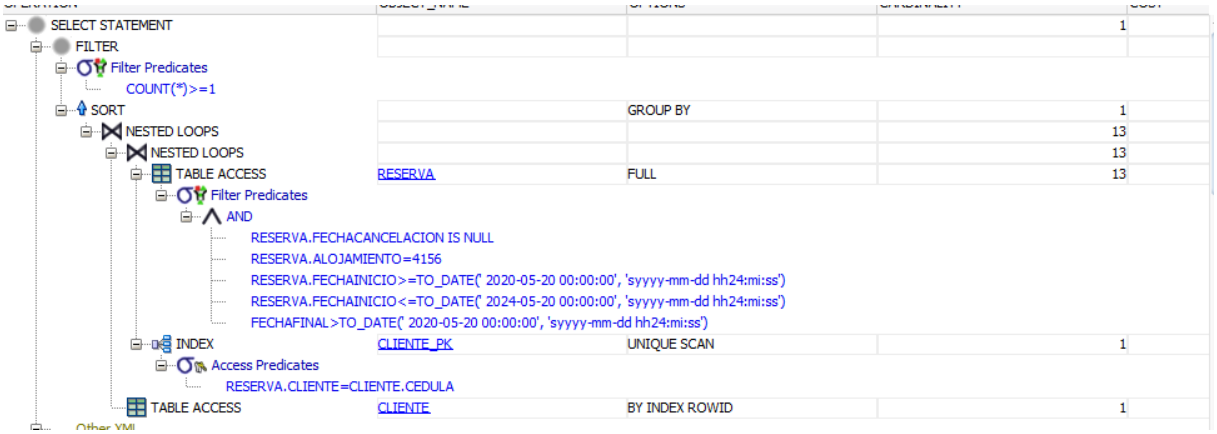


Figura 4. Índices Iniciales y Plan de Ejecución de SQL Developer para la consulta 10

Tal como se puede ver Oracle filtra primero por los clientes que sí tienen reservas y posteriormente ejecuta nested loops de acuerdo a las sentencias del tipo WHERE.

Análisis de La Eficacia

Para esta prueba se consideraron 3 casos de prueba variando el parámetro de fechadeinicio. Se dejaron fijos los parámetros de alojamiento (4156) y fechaFinal (2024-05-20). Los índices creados se encuentran en la parte superior en la figura 4. Estos son la llave primaria Cliente.Cedula de la relación Cliente.

Tabla 24. Análisis de la eficiencia para Requerimiento Funcional de Consulta

	tiempo [s]	**tiempo [s]-índice**	**selectividad [%]**	**fechaInicio**
caso1	0,064	0,018	33/200.000	2020-05-20
caso2	0,061	0,048	28/200.000	2022-05-20
caso3	0,052	0,038	13/200.000	2023-05-20

El índice más efectivo fue el índice sobre el alojamiento dado que la consulta filtra los resultados de acuerdo a un alojamiento particular. Por tanto, se puede ver, en la figura 5 que es el índice que Oracle utiliza para filtrar los resultados. Luego de aplicar el idx_alojamiento se observa que se reduce el tiempo de ejecución de cada una de las sentencias. Con el id_alojamiento el plan de ejecución para esta consulta sería:

1. Usar el índice en la columna "alojamiento.id" para realizar una búsqueda rápida de las tuplas que cumplen con la condición de igualdad "ALOJAMIENTO = 4156" en la cláusula WHERE. Esto ayudaría ya que ya no se tendrían que examinar todas las filas de la tabla.
2. Una vez que se obtienen los registros relacionados con el alojamiento específico, se busca la condición de rango de fechas en la cláusula WHERE "FECHAINICIO between FECHAINICIAL Y FECHAFINAL.
3. Se verifica posteriormente que la fechaDeCancelacion sea null dado que se quiere buscar las reservas que sí están activas.
4. Posteriormente, se realizaría la unión con la tabla cliente para buscar la información de los clientes que cumplen con dichas restricciones de la fecha y reserva en un alojamiento particular. Luego se agrupan los resultados.
5. Se ordenan los resultados y se traen las columnas deseadas.

Asimismo, con respecto al plan de ejecución al comparado con Oracle, se observa que ambos modelos difieren en que primero se debe hacer el filtro de que si se tenga al menos una reserva, luego se unen las tablas y posteriormente sí se aplican los filtros.

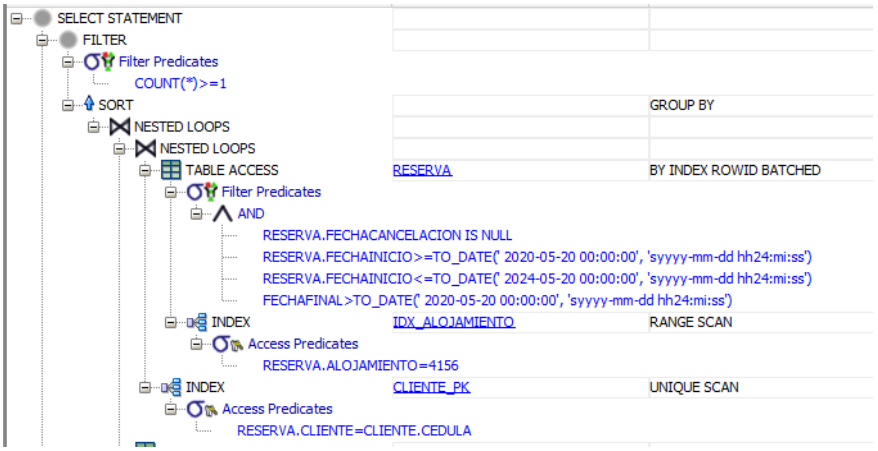


Figura 5. Índices IDX_Alojamiento y Plan de Ejecución de SQL Developer para la consulta 10

3.2 RFC- 11: Consumo en Alohandes - V2

Selección de Índices

Para este requerimiento funcional de consulta, al igual que en el RFC10, se decidieron crear tres índices compuestos basados en cliente, fechaInicio y alojamiento. Estos son los atributos que se encuentran en las sentencias tipo WHERE y de ORDENAMIENTO por lo que crear índices sobre estos atributos disminuye el tiempo de búsqueda en la base de datos. Inicialmente, por defecto, en Oracle SQL Developer los índices se generan como índices tipo B+. Estos son efectivos dado que los árboles B+ crecen a lo ancho (de manera horizontal) por lo que son eficientes a la hora de llegar a un argumento en particular. Además, este tipo de

índice no reorganiza totalmente el archivo por lo que automáticamente se reorganiza con cambios pequeños y locales en las inserciones y borrados.

Índices Creados Automáticamente Por Oracle

Los índices creados se encuentran en la parte superior en la figura 6. Estos son la llave primaria Cliente.Cedula de la relación Cliente. En general, al ejecutar esta consulta Oracle utiliza por defecto el índice sobre la llave primaria Cliente.id dado que es el índice primario sobre la tabla Cliente. Esto se muestra en el plan de ejecución abajo.

Figura 7. Índices Iniciales y Plan de Ejecución de SQL Developer para la consulta 11

Tal como se puede ver Oracle filtra primero por los clientes que sí tienen reservas y posteriormente ejecuta nested loops de acuerdo a las sentencias del tipo WHERE.

Análisis de La Eficacia

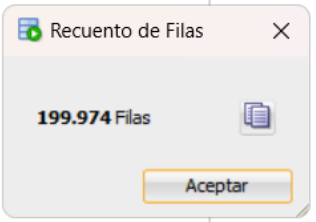
Para esta prueba se consideraron 3 casos de prueba variando el parámetro de fechadeinicio. Se dejaron fijos los parámetros de alojamiento (4156) y fechaFinal (2024-05-20). Los índices creados se encuentran en la parte superior en la figura 6. Estos son la llave primaria Cliente.Cedula de la relación Cliente.

Tabla 24. Análisis de la eficiencia para Requerimiento Funcional de Consulta

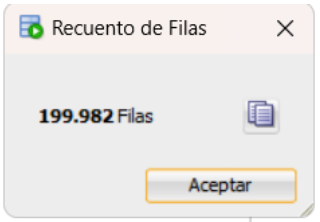
	tiempo [ms]	**tiempo [ms]-índice**	**selectividad [%]**	**fechaInicio**
caso1	8	N.D	199967/200000	2020-05-20
caso2	8	N.D	199972/200000	2022-05-20
caso3	7	N.D	199987/200000	2023-05-20

Se obtienen resultados similares para diferentes alojamientos y diferentes periodos. Esto ocurre por la cantidad de tuplas en nuestro poblamiento masivo de las tablas involucradas (Cliente, Reservas y Alojamiento). Por la selectividad encontrada, consideramos que para este requerimiento funcional de consulta no era adecuado crear índices secundarios.

```
SELECT cliente.CEDULA AS CEDULA, 0 AS NUMRESERVAS,
CLIENTE.NOMBRE, CLIENTE.VINCULACION, CLIENTE.CORREO
 FROM CLIENTE
 WHERE CLIENTE.CEDULA NOT IN (
SELECT RESERVA.CLIENTE
FROM reserva
INNER JOIN CLIENTE ON reserva.cliente = cliente.cedula
WHERE FECHAINICIO between to_date('2020-01-01','yyyy-mm-dd') and to_date('2026-01-01','yyyy-mm-dd')
AND ALOJAMIENTO = 666
GROUP BY reserva.cliente, CLIENTE.NOMBRE,CLIENTE.VINCULACION, CLIENTE.CORREO
HAVING COUNT(reserva.cliente) >= 1)
ORDER BY CLIENTE.CEDULA;
```



```sql
SELECT cliente.CEDULA AS CEDULA, 0 AS NUMRESERVAS,
CLIENTE.NOMBRE, CLIENTE.VINCULACION, CLIENTE.CORREO
 FROM CLIENTE
 WHERE CLIENTE.CEDULA NOT IN (
SELECT RESERVA.CLIENTE
FROM reserva
INNER JOIN CLIENTE ON reserva.cliente = cliente.cedula
WHERE FECHAINICIO between to_date('2020-01-01','yyyy-mm-dd') and to_date('2023-01-01','yyyy-mm-dd')
AND ALOJAMIENTO = 777
GROUP BY reserva.cliente, CLIENTE.NOMBRE,CLIENTE.VINCULACION, CLIENTE.CORREO
HAVING COUNT(reserva.cliente) >= 1)
ORDER BY CLIENTE.CEDULA;
```



En el plan de ejecución se puede ver que Oracle solamente utiliza los índices primarios predeterminados para las tres tablas involucradas en la consulta.

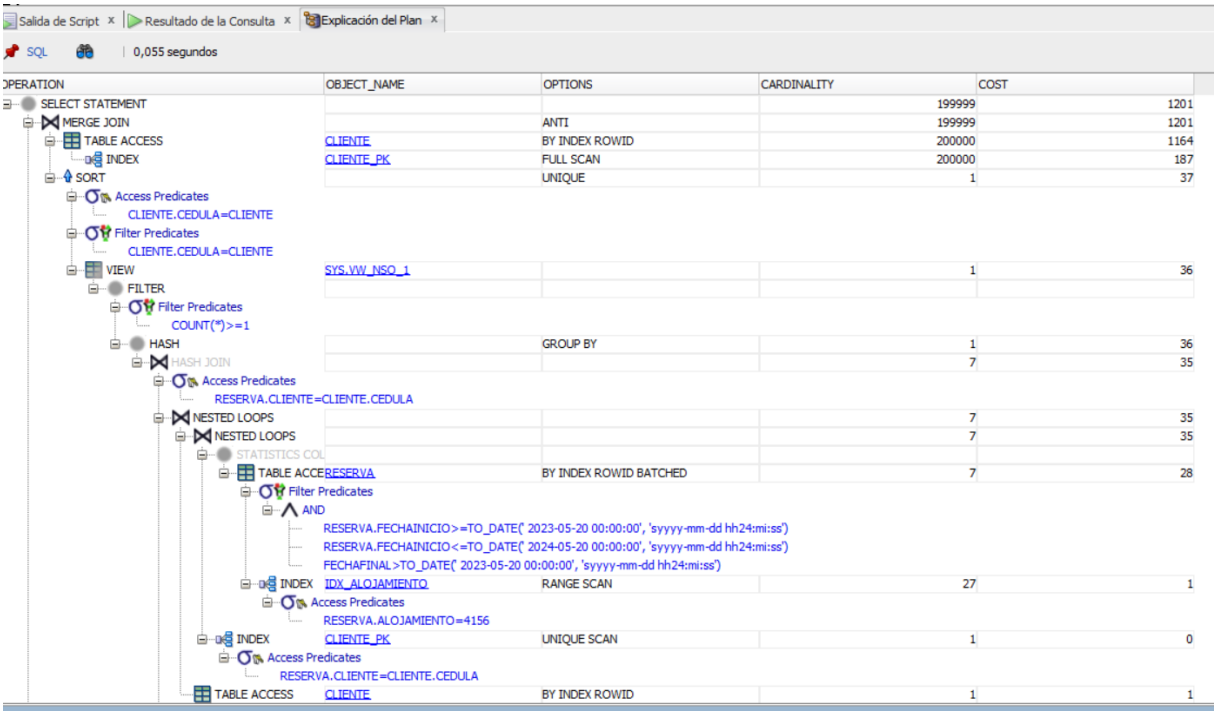


Figura 8. Plan de Ejecución de SQL Developer para la consulta 11

3.3 RFC- 12: Consultar Funcionamiento

Selección de Índices

```sql
CREATE INDEX idx_alojamiento_operador
ON ALOJAMIENTO (OPERADOR);

CREATE INDEX idx_reserva_fechainicio_fechafinal_fechacancelacion_alojamiento2
ON RESERVA (FECHAINICIO,FECHAFINAL, ALOJAMIENTO);

CREATE INDEX idx_reserva_fechainicio_fechafinal_fechacancelacion
ON RESERVA (FECHAINICIO,FECHAFINAL);
```

Figura 9. Índices Iniciales de SQL Developer para la consulta 12

Para esta requerimiento funcional de consulta se decidieron crear tres índices compuestos basados en operador, fechaInicio, fechafinal y alojamiento. Estos son los atributos que se encuentran en las sentencias tipo WHERE y de ORDENAMIENTO por lo que crear índices sobre estos atributos disminuye el tiempo de búsqueda en la base de datos. Inicialmente, por defecto, en Oracle SQL Developer los índices se generan como índices tipo B+ y son de tipo secundario y densos. Estos son efectivos dado que los árboles B+ crecen a lo ancho (de manera horizontal) por lo que son eficientes a la hora de llegar a un argumento en particular. Además, este tipo de índice no reorganiza totalmente el archivo por lo que automáticamente se reorganiza con cambios pequeños y locales en las inserciones y borrados. Asimismo, se espera que en los momentos en los que se debe hacer join sea más fácil buscar las tuplas que hacen *match* entre ambas tablas.

Índices Creados Automáticamente Por Oracle

Los índices creados se encuentran en la parte superior en la figura 4. Estos son la llave primaria Operador.Id de la relación Operador. En general, al ejecutar esta consulta Oracle utiliza por defecto el índice sobre la llave primaria Operador.id dado que es el índice primario sobre la tabla Operador. Esto se muestra en el plan de ejecución abajo.

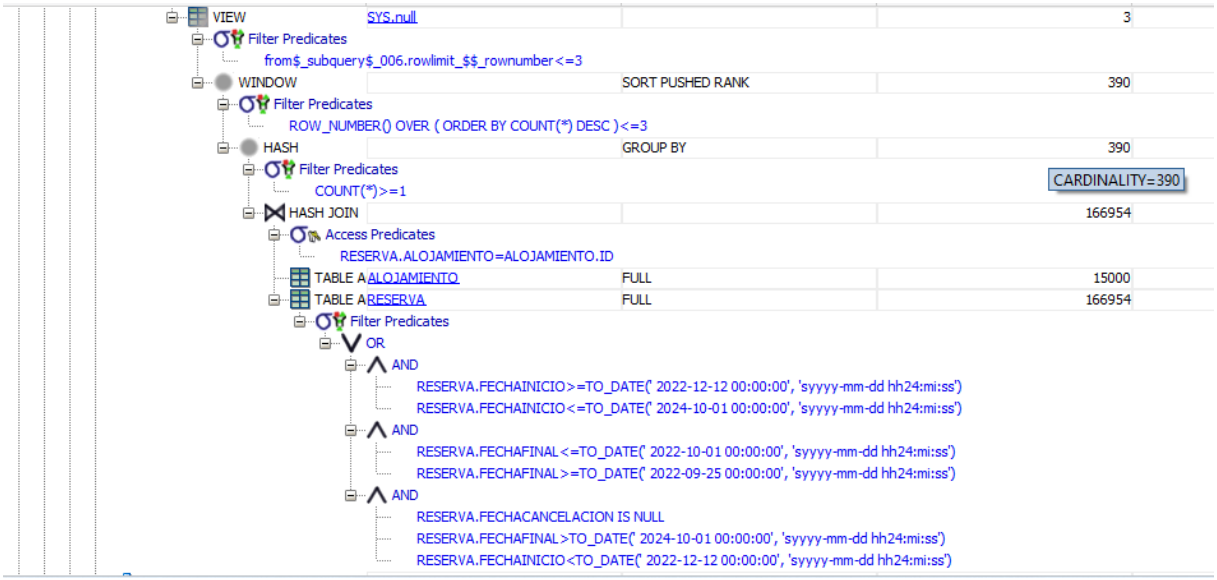


Figura 10. Índices Iniciales y Plan de Ejecución de SQL Developer para la consulta 10

Tal como se puede ver Oracle filtra primero por los operadores que sí tienen reservas y posteriormente ejecuta nested loops de acuerdo a las sentencias del tipo WHERE.

Análisis de La Eficacia

Para esta prueba se consideraron 3 casos de prueba variando el año para el cual se realiza la consulta total. Los años considerados fueron 2021, 2022 y 2023. De las cuatro subconsultas que tiene se utilizó específicamente la de los mejores operadores.

Tabla 24. Análisis de la eficiencia para Requerimiento Funcional de Consulta

	tiempo [s]	**tiempo [s]-índice**	**selectividad [%]**	**Año**

caso1	0,1002	0,118	7789/399714	2021
caso2	0,11	0,104	7789/399714	2022
caso3	0,135	0,113	7789/399714	2023

El índice más efectivo fue el índice sobre el operador_id y alojamiento_id dado que la consulta filtra los resultados de acuerdo a un operador particular o alojamiento para cada semana del año. Por tanto, se puede ver, en la figura 10 que es el índice que Oracle utiliza para filtrar los resultados. A pesar de haber creado índices para filtrar por las cláusulas where y el ordenamiento sigue siendo más efectivo el índice primario de la llave primaria de cada tabla. Esto se debe dado que al final se hacen consultas de agrupamiento, ordenamiento y join por esta columna. De acuerdo a lo anterior, se espera que el plan de ejecución sea el mismo que el de Oracle, es decir:

1. Se revisa la tabla RESERVA de acuerdo a los filtros de búsqueda de las columnas FECHAINICIO, FECHAFINAL y FECHACANCELACION.

2. Se hace el proceso de Join entre las tablas RESERVA y ALOJAMIENTO mediante la regla de que: reserva.alojamiento = alojamiento.id y luego se agrupan de acuerdo al operador o alojamiento.
3. Se realiza el ordenamiento por los grupos restantes se ordenarán en orden descendente según el COUNT calculado.
4. Se hace un fetch para que solo traiga las tuplas que se encuentran en las primeras tres posiciones.

En relación al tiempo de búsqueda este tiempo fue tomado desde la aplicación pasando por todas las capas hasta llegar al sql, ir a la base de datos y retornar resultados.

3.4 RFC- 13: Consultar los buenos clientes

Selección de Índices

```
55  CREATE INDEX idx_alojamiento_tipo ON Alojamiento (Tipo);
```

Figura 11. Índices Iniciales de SQL Developer para la consulta 13

Para este requerimiento funcional de consulta se decidió crear un índice secundario simple sobre el atributo Tipo de la tabla Alojamiento. Este es el atributo que se encuentra en la sentencia tipo WHERE de la subconsulta 3, por lo que crear índices sobre estos atributos disminuye el tiempo de búsqueda en la base de datos. Inicialmente, por defecto, en Oracle SQL Developer los índices se generan como índices tipo B+. Estos son efectivos dado que los árboles B+ crecen a lo ancho (de manera horizontal) por lo que son eficientes a la hora de llegar a un argumento en particular. Además, este tipo de índice no reorganiza totalmente el archivo por lo que automáticamente se reorganiza con cambios pequeños y locales en las inserciones y borrados.

Índices Creados Automáticamente Por Oracle

En este caso, Oracle utiliza en el plan de ejecución el índice primario sobre la tabla Alojamiento.

Análisis de La Eficacia

Para esta prueba se consideraron 3 casos de prueba variando el parámetro de fechadeinicio. Se dejaron fijos los parámetros de alojamiento (4156) y fechaFinal (2024-05-20). Los índices creados se encuentran en la parte superior en la figura 6. Estos son la llave primaria Cliente.Cedula de la relación Cliente.

Tabla 24. Análisis de la eficiencia para Requerimiento Funcional de Consulta

	tiempo [ms]	**tiempo [ms] - índice**	**selectividad [%]**
SUBCONSULTA 1	300	N.D.	172.710/200.000
SUBCONSULTA 2	43	N.D.	189.109/200.000
SUBCONSULTA 3	13	12	37.557/200.000

Debido a la selectividad de las dos primeras subconsultas, se decidió que no era conveniente crear índices adicionales a los predeterminados de Oracle. Por su parte, para la subconsulta 3, consideramos que el índice más efectivo es el índice sobre el atributo de tipo en la tabla alojamiento dado que la consulta busca sobre un alojamiento específico (hotelSuite). Por tanto, se puede ver que es el índice que Oracle utiliza para filtrar los resultados. Luego de aplicar el idx_alojamiento_tipo se observa que se reduce el tiempo de ejecución de cada una de las sentencias.

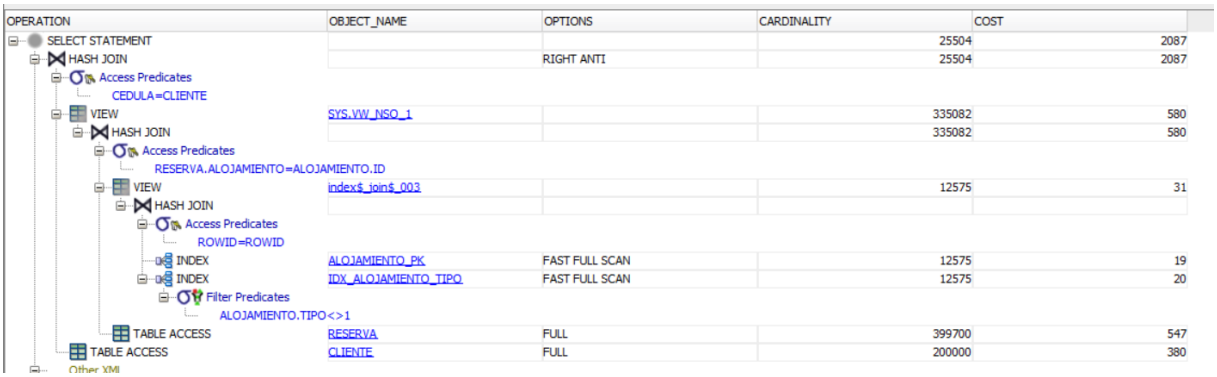


Figura 13. Índices IDX_Alojamiento y Plan de Ejecución de SQL Developer para la consulta 13

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

4.1 Diseño de los Datos

La creación masiva de datos se realizó usando Faker, una biblioteca de Python que permite generar datos ficticios (o "fake data"), la cual se usa ampliamente en pruebas y en la generación de datos para el desarrollo de software cuando se necesitan datos realistas pero no se pueden utilizar datos reales por razones de privacidad o seguridad. Además de esta biblioteca, usamos la biblioteca random, la cual es estándar en Python. Estas bibliotecas nos permitieron poblar las tablas de nuestra base de datos. Un fragmento del código que utilizamos usando Faker y random es:

```python
from faker import Faker
import random

# Instanciar Faker
fake = Faker()

def cargarCliente(fake : Faker, N_clientes : int = 0):

    with open('inserts_cliente.txt', 'w') as f:
        # Generar N_clientes sentencias INSERT
        for id in range(1, N_clientes+1):
            cedula = id
            correo = fake.email()
            nombre = fake.name()
            celular = fake.unique.random_int(min=3000000000, max=3999999999)  # Número de celular único de 10 dígitos que empieza por 3
            vinculacion = random.randint(1, 5)  # ID aleatoria de TIPOMIEMBROCOMUNIDAD entre 1 y 5
            clave = fake.password(length=12, special_chars=False)

            f.write(f"INSERT INTO CLIENTE (CEDULA, CORREO, NOMBRE, CELULAR, VINCULACION, CLAVE) "
                    f"VALUES ({cedula}, '{correo}', '{nombre}', {celular}, {vinculacion}, '{clave}');\n")
```

Figura 14. Extracto del código utilizado para el poblamiento masivo de la tabla Cliente.

La distribución de los datos se hizo pensando en cómo se distribuirían en la vida real. La mayor cantidad de tuplas se concentran en la tabla Reserva. Hay 10.000 operadores y 15.000 alojamientos (en promedio, cada operador tiene 1.5 alojamientos). Cada alojamiento tiene en promedio 10 servicios posibles y 2.5 servicios públicos. La mitad de todas las reservas tiene asociada un contrato. Por último, la mitad de los alojamientos tiene seguro y la mitad de alojamientos tiene horario. Entre todas las tablas, se crearon un poco más de un millón de datos, distribuidos de la siguiente manera:

Tabla 25. Distribución del número de tuplas para las tablas pobladas masivamente.

Tabla	Número de tuplas
Cliente	200.000
Operador	10.000
Alojamiento	15.000
Reserva	400.000
Servicio	150.000
ServicioPublico	37500
Contrato	200.000
seguro	7.500
Horario	7.500

4.2 Carga Masiva de Datos

Luego de generar los archivos .txt para cada tabla de Alohandes, estos se cargaron de manera manual en Oracle SQL Server copiando cada archivo .txt de la carpeta Data/CargaMasiva. En total se insertaron 1029917 datos distribuidos de manera uniforme en las tablas. La relación con más datos fue Reserva y con la de menos datos TipoOperadorEmpresa. A continuación, se muestra la distribución en la figura de la parte inferior.

	TABLE_NAME	ROW_COUNT
1	RESERVA	399700
2	CONTRATO	200000
3	CLIENTE	200000
4	SERVICIO	142889
5	SERVICIOPUBLICO	37287
6	ALOJAMIENTO	15000
7	OPERADOR	10000
8	HORARIO	7500
9	SEGURO	7500
10	PERSONANATURAL	5000
11	EMPRESA	5000
12	TIPOSERVICIO	19
13	TIPOALOJAMIENTO	6
14	TIPOMIEMBROCOMUNIDAD	5
15	TIPOSERVICIOPUBLICO	5
16	TIPOOPERADORNATURAL	3
17	TIPOOPERADOREMPRESA	3
18	TOTAL	1029917

Figura 14. Estado final de la BD [Elaborado en OracleSQLDeveloper]

4.3 Análisis del proceso de optimización y el modelo de ejecución de consultas

Se pudo observar a lo largo de la iteración que hay una gran diferencia entre la ejecución de consultas delegada al manejador de bases de datos como Oracle y la aplicación. En el primer caso, con respecto al procesamiento y rendimiento, la base de datos se encarga de optimizar y analizar la consulta y determina el plan de ejecución más eficiente a través del uso de índices. Asimismo, en la BD a pesar de añadir grandes volúmenes de datos gracias al uso de índices no aumenta exponencialmente el tiempo de ejecución.

Por otro lado, en la aplicación dado que no hay optimización la aplicación debe traer los datos desde la base de datos a memoria principal y procesarlos de forma local lo que hace que haya un rendimiento más lento y mayor consumo de recursos. También vale la pena considerar que la aplicación, al ingresar más datos tiene como restricción la memoria y capacidad de procesamiento local. Finalmente, es importante considerar que la aplicación resuelve con instrucciones de control (if, while, etc.) lo que puede aumentar la complejidad temporal.

5 Documentación de Cambios en el Diseño

5.1 Cambios en las Clases SQL

Entre los cambios considerados para las tres clases se incluye la modificación y adición de los atributos de las siguientes relaciones:

1. Operador: Se modificó las para que el operador fuera quien tuviera el atributo de correo en vez de las clases PersonaNatural o empresa. Esto significó cambiar tanto el esquema como las clases asociadas en el JDO. Se consideró este cambio dado que era redundante que lo tuviera tanto la empresa y/o personaNatural cuando ambas entidades heredan de Operador.
2. Se realizaron cambios a la tabla tipoAlojamiento para que ahora tuviera también las opciones de hotelSuite, hotelSemisuite y hotelEstandar. Estas hacen referencia a los tipos de habitaciones las cuales puede reservar un cliente cuando se trata de hoteles.

5.2 Desarrollo y/o ajustes en la interfaz y en el control de la aplicación

En relación, a los usuarios que pueden acceder a ALOHANDES por medio de la interfaz de la aplicación, se decidió añadir dos nuevos tipos de roles. El primero, es el administrador de base de datos ADMIN quien puede acceder tanto al RFC10 y RFC11 con la clave *password* y el CEO quien puede consultar semana por semana los mejores y peores alojamientos y operadores. La clave de este también es *password.*

5.3 Desarrollo y/o ajustes en la interfaz y en el control de la aplicación

Además, de añadir en cada capa del JDO los métodos necesarios para poder ejecutar los requerimientos funcionales de consulta RFC10, RFC11, RFC12 y RFC13 no se realizaron cambios en estas clases.

6 Resultados Logrados

De forma general, se logró poblar las tablas, los requerimientos de RF1 a RF10 y los requerimientos de consulta RFC1-RFC8. Asimismo se cumplió con:

1. **Privacidad**: El sistema tiene incorporado la autenticación de usuario (ya sea cliente o empresa). Por lo que, se necesita que se haga una autenticación y especificación de la información para poder hacer los requerimientos funcionales. Esto se adicionó en Registrar una Reserva, cancelar una Reserva, adicionar un Alojamiento y Retirar una oferta de Alojamiento.

2. **Persistencia**: Dado que es una base de datos, es persistente el manejo de la información para todas las clases a excepción de Alohandes.

3. **Concurrencia**: Para este requerimiento no funcional se mencionaba que cada una de las solicitudes podían ser hechas de manera múltiple y simultánea. Esto es particularmente importante para el caso de las consultas de información y registros. La carencia de información duplicada garantiza la unicidad de la información. De esta manera, cada consulta o registro se maneja como si fuera secuencial, siguiendo el concepto de transaccionalidad *Isolation.* Además, es posible que dos usuarios distintos hagan uso de la aplicación en máquinas diferentes accediendo a la misma base de datos, y aun así podrán realizar sus operaciones de manera simultánea.

4. **Distribución**: Para la garantía de la información en la base de datos de manera centralizada, se maneja el concepto de Alohandes, que tiene conexión directa o indirecta con todas las entidades del modelo, y es a partir de donde se inicia el análisis para cada uno de los requerimientos funcionales una vez se ejecuta la autenticación de Usuario (empresa/cliente). Además, está centralizada haciendo uso del SMBD de Oracle SQL Developer.

5. **Transaccionalidad:** La aplicación cumple con transaccionalidad ya que considera que puede haber solicitudes simultáneas. Esto se consideró la capa de persistencia, en especial, al considerar tx.begin, tx.commit y el nivel de aislamiento como se discutió en la sección anterior.

6. **Eficiencia en las consultas:** Con respecto a la eficiencia de las consultas, se puede observar en la sección 9 que todas las consultas se encuentran por debajo de 0,8 s. Esto es, independientemente de las tuplas insertadas. Por lo tanto, la aplicación cumple con estre RNF.

7. **Eficiencia en la Actualización:** Al ejecutar los criterios de actualización y eliminación de las consultas desde el RF1 al RF10 se puede observar que los tiempos de espera son bajos por lo que sí se presenta este RNF.

8. Esquema físico de la base de datos: En general, dado que se realizó el proceso de análisis de los índices y tanto para los criterios de consulta y actualización, se evidencia balance global de eficiencia de la aplicación.

7 Resultados No Logrados

En general, con respecto a esta iteración, hizo falta lograr una mejor optimización de los tiempos de ejecución de la aplicación. Dado que mientras en la BD sí se cumplió con el criterio de la eficiencia, en el JDO los tiempos aumentaron significativamente debido al diseño de los algorítmos implementados, estructuras de datos seleccionadas y conexión entre las capas en conjunto con la base de datos.

8 Supuestos Adicionales

Con respecto a anteriores iteraciones algunos de los supuestos adicionales que se tuvieron en cuenta fueron:

- El mejor operador o mejor alojamiento es el que tiene más reservas asociadas a un id específico. Lo mismo ocurre con el peor operador o alojamiento es el que tiene un menor número de reservas.
- En el caso del usuario de cliente, este únicamente podrá ver sus reservas asociadas a algún alojamiento particular. En caso de que tenga reservas las podrá ver. Sin embargo si no tiene reservas se le indicará que no tiene reservas para las fechas dadas.

9 Balance de Pruebas

9.1 RFC- 10: Consumo en Alohandes

Los casos de éxito para el requerimiento funcional son:

Estado Inicial de la BD

	ID	FECHACANCELACION	FECHAINICIO	FECHAFINAL	ALOJAMIENTO	CLIENTE	CONTRATO	PRECIO	CANTIDADPERSONAS	RESERVAMADRE
1	213798	(null)	20/12/21	14/12/24	5445	153698	106899	5634495,18	1	(null)

Figura 15. Datos involucrados en RFC10 en el caso del cliente [Elaborado en OracleSQLDeveloper]

	CEDULA	NUMRESERVAS	NOMBRE	VINCULACION	CORREO
1	474	1	Devin Jacobson	5	theresa04@example.org
2	4180	1	Dana Robinson	2	mcrawford@example.net
3	13733	1	Megan Cooper	5	david41@example.net
4	14581	1	Erica Smith	4	lukebridges@example.org
5	17004	1	David Zhang	1	andrew26@example.org
6	20105	1	Leslie Hall	5	jturner@example.com
7	22822	1	Victor Nelson Jr.	2	youngsarah@example.net
8	38580	1	Laura Byrd	1	brandy55@example.com
9	56388	1	Kenneth Delgado	2	meyerrobin@example.com
10	63048	1	Samuel Hutchinson	2	wmcmahon@example.org

Figura 16. Datos involucrados en RFC10 en el caso del ADMIN[Elaborado en OracleSQLDeveloper]

Datos involucrados en la operación transaccional solicitada

```
SELECT * FROM RESERVA
WHERE CLIENTE = 153698
AND FECHAINICIO between to_date('2020-07-01','yyyy-mm-dd') and to_date('2026-02-20','yyyy-mm-dd')
AND FECHACANCELACION IS NULL
AND ALOJAMIENTO = 5445
ORDER BY PRECIO;
```

Figura 17. Datos involucrados en RFC10 [Elaborado en OracleSQLDeveloper]

```
SELECT reserva.cliente AS CEDULA, COUNT(reserva.alojamiento) as NUMRESERVAS, CLIENTE.NOMBRE,CLIENTE.VINCULACION, CLIENTE.CORREO
FROM reserva
INNER JOIN CLIENTE ON reserva.cliente = cliente.cedula
WHERE FECHAINICIO between to_date('2020-05-20','yyyy-mm-dd') and to_date('2024-05-20','yyyy-mm-dd')
AND FECHACANCELACION IS NULL
AND ALOJAMIENTO = 4156
GROUP BY reserva.cliente, CLIENTE.NOMBRE,CLIENTE.VINCULACION, CLIENTE.CORREO
HAVING COUNT(reserva.cliente) >= 1
ORDER BY CEDULA;
```

Figura 18. Datos involucrados en RFC10 en el caso del ADMIN [Elaborado en OracleSQLDeveloper]

Los casos de fallo para el requerimiento funcional de consulta se consideran a continuación. En el caso de falla se considera la opción cuando el usuario cliente solicita en un alojamiento en el que nunca ha hecho reservas o en el caso del admin en un alojamiento que no existe. Otro posible caso de fallo para ambos usuarios es solicitar en fechas en las que no hay reservas.

Datos involucrados en la operación transaccional solicitada

```
SELECT * FROM RESERVA
WHERE CLIENTE = 153698
AND FECHAINICIO between to_date('2020-07-01','yyyy-mm-dd') and to_date('2026-02-20','yyyy-mm-dd')
AND FECHACANCELACION IS NULL
AND ALOJAMIENTO = 1
ORDER BY PRECIO;
```

Figura 19. Datos involucrados en RFC10 en el caso del CLIENTE fallo [Elaborado en OracleSQLDeveloper]

```
SELECT reserva.cliente AS CEDULA, COUNT(reserva.alojamiento) as NUMRESERVAS, CLIENTE.NOMBRE,CLIENTE.VINCULACION, CLIENTE.CORREO
FROM reserva
INNER JOIN CLIENTE ON reserva.cliente = cliente.cedula
WHERE FECHAINICIO between to_date('2020-05-20','yyyy-mm-dd') and to_date('2024-05-20','yyyy-mm-dd')
AND FECHACANCELACION IS NULL
AND ALOJAMIENTO = 100000
GROUP BY reserva.cliente, CLIENTE.NOMBRE,CLIENTE.VINCULACION, CLIENTE.CORREO
HAVING COUNT(reserva.cliente) >= 1
ORDER BY CEDULA;
```

Figura 20. Datos involucrados en RFC10 en el caso del CLIENTE fallo [Elaborado en OracleSQLDeveloper]

9.2 RFC- 11: Consumo en Alohandes - V2

Los casos de éxito para el requerimiento funcional son:

Estado Inicial de la BD

Salida de Script | Explicación del Plan | Resultado de la Consulta

SQL | Se han recuperado 50 filas en 0,059 segundos

	CEDULA	NUMRESERVAS	NOMBRE	VINCULACION	CORREO
1	1	0	Kenneth Padilla	5	nancytaylor@example.org
2	2	0	Alexandra White	1	christopher32@example.org
3	3	0	Gary Rhodes	2	benjamin93@example.net
4	4	0	Tonya Fernandez	3	david35@example.com
5	5	0	Vanessa Murphy	3	mariagoodwin@example.com
6	6	0	Jaime Snow	1	suzanne20@example.com
7	7	0	Raymond Perez	2	greenchristopher@example.net
8	8	0	Angela Lewis	2	ewebster@example.net
9	9	0	Brianna Williams	1	greenjason@example.org
10	10	0	April Crane	2	julia48@example.com
11	11	0	Brooke Wilson	3	marshallaaron@example.com
12	12	0	Philip Davis	4	georgemontoya@example.org

Figura 21. Datos involucrados en RFC10 en el caso del ADMIN [Elaborado en OracleSQLDeveloper]

Datos involucrados en la operación transaccional solicitada

```
SELECT cliente.CEDULA AS CEDULA, 0 AS NUMRESERVAS,
CLIENTE.NOMBRE, CLIENTE.VINCULACION, CLIENTE.CORREO
 FROM CLIENTE
 WHERE CLIENTE.CEDULA NOT IN (
SELECT RESERVA.CLIENTE
FROM reserva
INNER JOIN CLIENTE ON reserva.cliente = cliente.cedula
WHERE FECHAINICIO between to_date('2020-05-20','yyyy-mm-dd') and to_date('2024-05-20','yyyy-mm-dd')
AND FECHACANCELACION IS NULL
AND ALOJAMIENTO = 4156
GROUP BY reserva.cliente, CLIENTE.NOMBRE,CLIENTE.VINCULACION, CLIENTE.CORREO
HAVING COUNT(reserva.cliente) >= 1)
ORDER BY CLIENTE.CEDULA;
```

Figura 22. Datos involucrados en RFC10 en el caso del ADMIN [Elaborado en OracleSQLDeveloper]

Los casos de fallo para el requerimiento funcional de consulta se consideran a continuación. En el caso de falla se considera la opción cuando el admin consulta los clientes en un alojamiento que no existe.

Datos involucrados en la operación transaccional solicitada

```
SELECT cliente.CEDULA AS CEDULA, 0 AS NUMRESERVAS,
CLIENTE.NOMBRE, CLIENTE.VINCULACION, CLIENTE.CORREO
 FROM CLIENTE
 WHERE CLIENTE.CEDULA NOT IN (
SELECT RESERVA.CLIENTE
FROM reserva
INNER JOIN CLIENTE ON reserva.cliente = cliente.cedula
WHERE FECHAINICIO between to_date('2020-05-20','yyyy-mm-dd') and to_date('2024-05-20','yyyy-mm-dd')
AND FECHACANCELACION IS NULL
AND ALOJAMIENTO = 666666
GROUP BY reserva.cliente, CLIENTE.NOMBRE,CLIENTE.VINCULACION, CLIENTE.CORREO
HAVING COUNT(reserva.cliente) >= 1)
ORDER BY CLIENTE.CEDULA;
```

Figura 23. Datos involucrados en RFC11 en el caso fallo del ADMIN [Elaborado en OracleSQLDeveloper]

9.3 RFC- 12: Consultar Funcionamiento

Los casos de éxito para el requerimiento funcional son:

Estado Inicial de la BD

ID	NUMHABITACIONES	UBICACION	PRECIO	AMOBLADO	CAPACIDAD	COMPARTIDO	INDICEOCUPACION	TIPO	HORARIO	SEGURO
4023	9	817 Sophia Circle Apt. 935, East Charleneport, ID 79592	1536081,4182253804	0	11	0	97	3	(null)	2012

Figura 21. Tupla retornada en RFC12 en el caso de mejores alojamiento [Elaborado en OracleSQLDeveloper]

ID	NUMHABITACIONES	UBICACION	PRECIO	AMOBLADO	CAPACIDAD	COMPARTIDO	INDICEOCUPACION	TIPO	HORARIO	SEGURO	OPERADOR
5547	9	95255 Erica Summit Apt. 000, North Amandaburgh, FM 60649	8229562,771043359	1	16	1	53	4	(null)	2774	1663

Figura 22. Tupla retornada en RFC12 en el caso de peores alojamiento [Elaborado en OracleSQLDeveloper]

ID	INGRESOPORANIOACTUAL	INGRESOPORANIOCORRIDO	NOMBRE	CLAVE	CORREO
4856	31854468	478873067	Timothy Nash	GUkUlvZZ8CKO	garciajulie@example.com
747	166485790	233822540	Matthew Miller	MBSv3Yie3YP0	riversjennifer@example.net
218	317157956	363496028	Andrew Campos	86SNPCGer9yX	maria58@example.com

Figura 23. Tupla retornada en RFC12 en el caso de peores operadores [Elaborado en OracleSQLDeveloper]

ID	INGRESOPORANIOACTUAL	INGRESOPORANIOCORRIDO	NOMBRE	CLAVE	CORREO
931	270508365	431012457	Elizabeth Smith	20yWdntxKZLQ	robertdodson@example.com
7856	812875482	920769152	Brianna Stout	h8ovTQJe4p41	bcombs@example.net
1781	534828538	821598799	Brianna Fuentes	34wQ2ekkq8i2	ryanvictor@example.net

Figura 24. Tupla retornada en RFC12 en el caso de mejores operadores [Elaborado en OracleSQLDeveloper]

Datos involucrados en la operación transaccional solicitada

```
select alojamiento.* from alojamiento where id=(select reserva.alojamiento
from reserva
WHERE FECHAINICIO between to_date('2022-01-01','yyyy-mm-dd') and to_date('2022-12-12','yyyy-mm-dd')
OR FECHAFINAL between to_date('2022-01-01','yyyy-mm-dd') and to_date('2022-10-01','yyyy-mm-dd')
AND FECHACANCELACION IS NULL
GROUP BY reserva.alojamiento
HAVING COUNT(reserva.alojamiento) >= 1
ORDER BY count(ALOJAMIENTO) desc
FETCH FIRST 1 ROWS ONLY);
```

Figura 25. Datos involucrados en RFC12 en el caso de mejores alojamiento [Elaborado en OracleSQLDeveloper]

```
select alojamiento.* from alojamiento where id=(select reserva.alojamiento
from reserva
WHERE FECHAINICIO between to_date('2022-12-12','yyyy-mm-dd') and to_date('2024-10-01','yyyy-mm-dd')
OR FECHAFINAL between to_date('2022-09-25','yyyy-mm-dd') and to_date('2022-10-01','yyyy-mm-dd')
AND FECHACANCELACION IS NULL
GROUP BY reserva.alojamiento
HAVING COUNT(reserva.alojamiento) >= 1
ORDER BY count(ALOJAMIENTO) asc
FETCH FIRST 1 ROWS ONLY);
```

Figura 26. Datos involucrados en RFC12 en el caso de mejores alojamiento [Elaborado en OracleSQLDeveloper]

```
select operador.* from operador where id in (select a.operador from(
select reserva.*,alojamiento.operador as operador from reserva
inner join alojamiento
on reserva.alojamiento = alojamiento.id
WHERE FECHAINICIO between to_date('2022-12-12','yyyy-mm-dd') and to_date('2024-10-01','yyyy-mm-dd')
OR FECHAFINAL between to_date('2022-09-25','yyyy-mm-dd') and to_date('2022-10-01','yyyy-mm-dd')
OR (FECHAINICIO <to_date('2022-12-12','yyyy-mm-dd') and FECHAFINAL > to_date('2024-10-01','yyyy-mm-dd'))
AND FECHACANCELACION IS NULL) a
GROUP BY a.operador
HAVING COUNT(a.operador) >= 1
ORDER BY count(a.operador) asc
FETCH FIRST 3 ROWS ONLY);
```

Figura 27. Datos involucrados en RFC12 en el caso de peores operadores [Elaborado en OracleSQLDeveloper]

```
select operador.* from operador where id in (select a.operador from(
select reserva.*,alojamiento.operador as operador from reserva
inner join alojamiento
on reserva.alojamiento = alojamiento.id
WHERE FECHAINICIO between to_date('2022-12-12','yyyy-mm-dd') and to_date('2024-10-01','yyyy-mm-dd')
OR FECHAFINAL between to_date('2022-09-25','yyyy-mm-dd') and to_date('2022-10-01','yyyy-mm-dd')
OR (FECHAINICIO <to_date('2022-12-12','yyyy-mm-dd') and FECHAFINAL > to_date('2024-10-01','yyyy-mm-dd'))
AND FECHACANCELACION IS NULL) a
GROUP BY a.operador
HAVING COUNT(a.operador) >= 1
ORDER BY count(a.operador) desc
FETCH FIRST 3 ROWS ONLY);
```

Figura 28. Datos involucrados en RFC12 en el caso de mejores alojamiento [Elaborado en OracleSQLDeveloper]

Los casos de fallo para el requerimiento funcional de consulta se consideran a continuación. En el caso de falla se considera la opción cuando el usuario cliente solicita en un alojamiento que no existe o un año que no existe o del que no hay reservas. Esto pasa tanto para operador como para alojamiento.

```
select alojamiento.* from alojamiento where id=(select reserva.alojamiento
from reserva
WHERE FECHAINICIO between to_date('2027-01-01','yyyy-mm-dd') and to_date('2027-12-12','yyyy-mm-dd')
OR FECHAFINAL between to_date('2027-01-01','yyyy-mm-dd') and to_date('2027-10-01','yyyy-mm-dd')
AND FECHACANCELACION IS NULL
GROUP BY reserva.alojamiento
HAVING COUNT(reserva.alojamiento) >= 1
ORDER BY count(ALOJAMIENTO) desc
FETCH FIRST 1 ROWS ONLY);
```

Figura 29. Datos involucrados en RFC12 en el caso de mejores alojamiento: error [Elaborado en OracleSQLDeveloper]

```
select alojamiento.* from alojamiento where id=(select reserva.alojamiento
from reserva
WHERE FECHAINICIO between to_date('2027-12-12','yyyy-mm-dd') and to_date('2027-10-01','yyyy-mm-dd')
OR FECHAFINAL between to_date('2027-09-25','yyyy-mm-dd') and to_date('2027-10-01','yyyy-mm-dd')
AND FECHACANCELACION IS NULL
GROUP BY reserva.alojamiento
HAVING COUNT(reserva.alojamiento) >= 1
ORDER BY count(ALOJAMIENTO) asc
FETCH FIRST 1 ROWS ONLY);
```

Figura 30. Datos involucrados en RFC12 en el caso de peores alojamiento: error [Elaborado en OracleSQLDeveloper]

```
select operador.* from operador where id in (select a.operador from(
select reserva.*,alojamiento.operador as operador from reserva
inner join alojamiento
on reserva.alojamiento = alojamiento.id
WHERE FECHAINICIO between to_date('2027-12-12','yyyy-mm-dd') and to_date('2027-10-01','yyyy-mm-dd')
OR FECHAFINAL between to_date('2027-09-25','yyyy-mm-dd') and to_date('2027-10-01','yyyy-mm-dd')
OR (FECHAINICIO <to_date('2027-12-12','yyyy-mm-dd') and FECHAFINAL > to_date('2027-10-01','yyyy-mm-dd'))
AND FECHACANCELACION IS NULL) a
GROUP BY a.operador
HAVING COUNT(a.operador) >= 1
ORDER BY count(a.operador) asc
FETCH FIRST 3 ROWS ONLY);
```

Figura 31. Datos involucrados en RFC12 en el caso de peores operadores: error [Elaborado en OracleSQLDeveloper]

```
select operador.* from operador where id in (select a.operador from(
select reserva.*,alojamiento.operador as operador from reserva
inner join alojamiento
on reserva.alojamiento = alojamiento.id
WHERE FECHAINICIO between to_date('2027-12-12','yyyy-mm-dd') and to_date('2027-10-01','yyyy-mm-dd')
OR FECHAFINAL between to_date('2027-09-25','yyyy-mm-dd') and to_date('2027-10-01','yyyy-mm-dd')
OR (FECHAINICIO <to_date('2027-12-12','yyyy-mm-dd') and FECHAFINAL > to_date('2027-10-01','yyyy-mm-dd'))
AND FECHACANCELACION IS NULL) a
GROUP BY a.operador
HAVING COUNT(a.operador) >= 1
ORDER BY count(a.operador) desc
FETCH FIRST 3 ROWS ONLY);
```

Figura 32. Datos involucrados en RFC12 en el caso de mejores operadores: error [Elaborado en OracleSQLDeveloper]

9.4 RFC- 13: Consultar los buenos clientes

Los casos de éxito para el requerimiento funcional son:

Estado Inicial de la BD

	CEDULA	CORREO	NOMBRE	CELULAR	VINCULACION	CLAVE
1	426	nicoleweber@example.com	Tamara Jones	3096012725	5	1GDYpmqCVkdV
2	427	nathan90@example.org	Devin Thomas	3304144599	3	99OPBkLYFmet
3	429	karlsmith@example.net	Gabriel Smith	3044405431	4	F5jQPext0F0T
4	430	lindsay01@example.net	Anne Nelson	3071416941	4	CEUvJo8J9JmT
5	431	michael73@example.com	Scott Conley	3536461582	1	MsQQqlemQ2xg
6	432	sarah86@example.com	Anna Smith	3306650722	5	T8PWtfZhdhcS
7	433	elizabeth41@example.org	Nicholas Cortez	3911327562	5	3EHbcdVNr4kn
8	435	dennismarshall@example.com	Margaret Gonzales	3002587595	2	ulyQirzzR4CR
9	436	christopherwheeler@example.org	Rebecca Combs	3904505549	2	RDDwNiBCTo9h
10	437	caitlin92@example.org	Christian Lewis	3766804390	1	ClYR95cRK8vK

Figura 33. Tuplas retornada en RFC13 en la primera subconsulta de buenos clientes [Elaborado en OracleSQLDeveloper]

	CEDULA	CORREO	NOMBRE	CELULAR	VINCULACION	CLAVE
1	426	nicoleweber@example.com	Tamara Jones	3096012725	5	1GDYpmqCVkdV
2	427	nathan90@example.org	Devin Thomas	3304144599	3	99OPBkLYFmet
3	428	ztanner@example.net	Aaron Carlson	3035527635	3	s3pPe6ZCc7Kd
4	429	karlsmith@example.net	Gabriel Smith	3044405431	4	F5jQPext0F0T
5	430	lindsay01@example.net	Anne Nelson	3071416941	4	CEUvJo8J9JmT
6	431	michael73@example.com	Scott Conley	3536461582	1	MsQQqlemQ2xg
7	432	sarah86@example.com	Anna Smith	3306650722	5	T8PWtfZhdhcS
8	433	elizabeth41@example.org	Nicholas Cortez	3911327562	5	3EHbcdVNr4kn
9	434	justin98@example.com	Jonathan Schmidt	3396713177	3	XJJCHuCwzey0
10	435	dennismarshall@example.com	Margaret Gonzales	3002587595	2	ulyQirzzR4CR

Figura 34. Tuplas retornada en RFC13 en la segunda subconsulta de buenos clientes [Elaborado en OracleSQLDeveloper]

	CEDULA	CORREO	NOMBRE	CELULAR	VINCULACION	CLAVE
1	428	ztanner@example.net	Aaron Carlson	3035527635	3	s3pPe6ZCc7Kd
2	431	michael73@example.com	Scott Conley	3536461582	1	MsQQqlemQ2xg
3	434	justin98@example.com	Jonathan Schmidt	3396713177	3	XJJCHuCwzey0
4	444	wilsonbrandy@example.net	Lori Dixon	3705079422	5	MvFWfajo3Qkl
5	445	mirandaangela@example.net	Adam Jones	3223130138	3	74gwQrA4aYGm
6	451	daniellel1@example.net	Joel Long	3308529989	1	6B9qXFef6fBA
7	458	suegilbert@example.com	Shirley Ortiz	3693796755	2	oQuvhDRU7RGn
8	465	newmanrandy@example.net	Franklin Ramsey	3959599144	5	TEaeBSUp4nyu
9	476	davidbennett@example.net	Vicki Ferguson MD	3591778796	5	3DNVXrarCFnV
10	478	mark00@example.org	Whitney Jones	3956937691	5	IjGxurWMg8Li

Figura 35. Tuplas retornada en RFC13 en la tercera subconsulta de buenos clientes [Elaborado en OracleSQLDeveloper]

Datos involucrados en la operación transaccional solicitada

```
SELECT Cliente.*
FROM Cliente
WHERE Cedula IN (
SELECT Cliente
FROM Reserva
GROUP BY Cliente
HAVING COUNT(DISTINCT(EXTRACT(MONTH FROM FechaInicio))) >= 1);
```

Figura 36. Datos involucrados en RFC13 en la primera subconsulta de buenos clientes [Elaborado en OracleSQLDeveloper]

```
SELECT Cliente.*
FROM Cliente
WHERE Cedula NOT IN (
SELECT Cliente
FROM Reserva
WHERE Precio <= 1500000
);
```

Figura 37. Datos involucrados en RFC13 en la segunda subconsulta de buenos clientes [Elaborado en OracleSQLDeveloper]

```
SELECT Cliente.*
FROM Cliente
WHERE Cedula NOT IN (SELECT Reserva.Cliente FROM Reserva
JOIN Alojamiento ON Reserva.Alojamiento = Alojamiento.ID
WHERE Alojamiento.Tipo != '1');
```

Figura 38. Datos involucrados en RFC13 en la tercera subconsulta de buenos clientes [Elaborado en OracleSQLDeveloper]

```
select operador.* from operador where id in (select a.operador from(
select reserva.*,alojamiento.operador as operador from reserva
inner join alojamiento
on reserva.alojamiento = alojamiento.id
WHERE FECHAINICIO between to_date('2022-12-12','yyyy-mm-dd') and to_date('2024-10-01','yyyy-mm-dd')
OR FECHAFINAL between to_date('2022-09-25','yyyy-mm-dd') and to_date('2022-10-01','yyyy-mm-dd')
OR (FECHAINICIO <to_date('2022-12-12','yyyy-mm-dd') and FECHAFINAL > to_date('2024-10-01','yyyy-mm-dd'))
AND FECHACANCELACION IS NULL) a
GROUP BY a.operador
HAVING COUNT(a.operador) >= 1
ORDER BY count(a.operador) desc
FETCH FIRST 3 ROWS ONLY);
```

Figura 28. Datos involucrados en RFC12 en el caso de mejores alojamiento [Elaborado en OracleSQLDeveloper]

Los casos de fallo para este requerimiento funcional de consulta ocurren cuando el director general ingresa una contraseña incorrecta (único input de este requerimiento).