Universidad de los Andes

FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA DE SISTEMAS

ITERACIÓN 2: PROYECTO 1

Sergio Pardo Gutierrez
Juan Diego Yepes Parra

1 de abril de 2022
Bogotá D.C.

Tabla de contenidos

1 Introducción

HOTELANDES

A-12

En el siguiente documento se pretende hacer un análisis de la segunda iteración del proyecto 1 Hotel-Andes; en el cual se van a documentar las reglas de negocio, decisiones tomadas y cambios hechos con respecto a la primera iteración, para poder cumplir con todos los requerimientos y asegurar los buenos principios de diseño y arquitectura con bases de datos `SMBD` definidas con `SQL` en Oracle. Para ello, vamos a mencionar los cambios que hicimos en el modelo conceptual, vamos a listar las tablas hechas para la base de datos, vamos a hacer una explicación de cómo ejecutar el programa `a12-hotelandes-jdo` y vamos a mostrar todos los requerimientos funcionales del mismo. Finalmente, vamos a discutir sobre los resultados logrados y haremos un balance del plan de pruebas.

2 Actualizaciones

Teniendo en cuenta la retroalimentación recibida en la sustentación recibida en la sustentación de la iteración 1, estos son los cambios hechos en el proceso de modelado

2.1 Modelo Conceptual

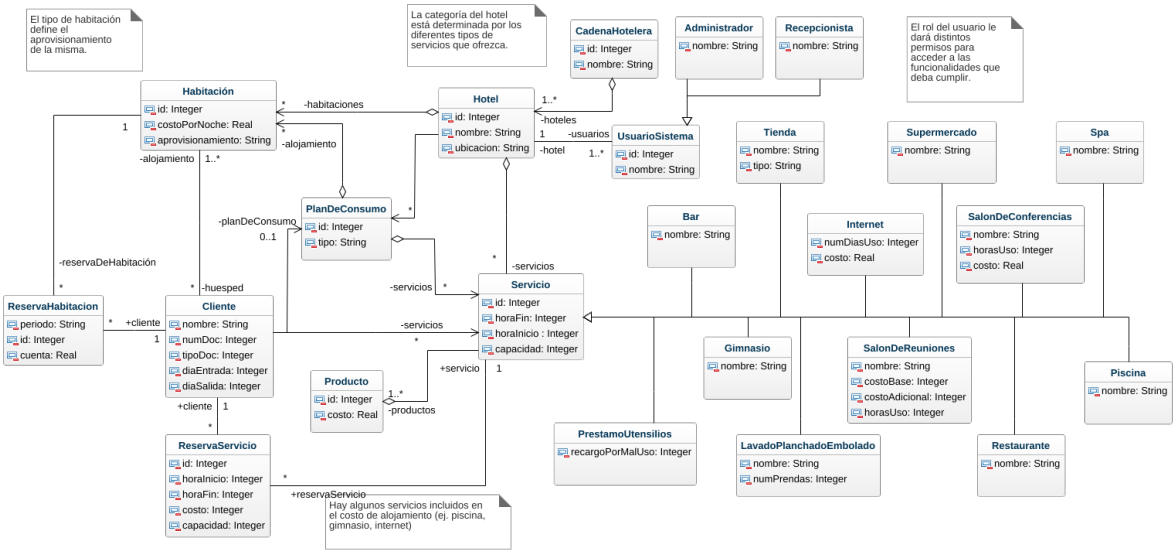

Fig. 1: Modelo conceptual actualizado

Con respecto al anterior modelo UML, en este modelo conceptual sí se están incluyendo todos los servicios, cada clase que es un servicio hereda de la clase `Servicio`, (valga la redundancia). También, en este modelo decidimos quitar la asociación de que los `id` del usuario representan el tipo de usuario, puesto que esta implementación no está utilizando el secuenciador de la base de datos y por ende pueden haber errores de concurrencia.

2.2 Modelo Relacional

Estas son las tablas de nuestro sistema:

- Hotel

id	nombre	ubicación
PK	UA	UA

- Cliente

nombre	numDoc	tipoDoc	diaEntrada	diaSalida
NN	PK, UA	PK, UA, CK in ('CC','TI','CE')	NC	NC

- TipoHabitacion

id	nombre	descripcion
PK	UA	UA

- Habitacion

id	costoPorNoche	aprovisionamiento	tipoHabitacion
PK	UA	UA	$\texttt{FK}_{TipoHabitacion}$

- RolesDeUsuario

id	nombre
PK	UA

- UsuarioSistema

id	nombre	rol
PK	UA	$\texttt{FK}_{RolesDeUsuario}$

- ReservaDeHabitacion

id	idHabitacion	tipoDocCliente	numDocCliente
PK	$\texttt{FK}_{Habitacion}$	$\texttt{FK}_{Cliente}$, CK in ('CC', 'TI' ,'CE')	UA

diaEntrada	diaSalida	completada	cuenta
UA	UA	UA	UA

- Servicio

id	horaInicio	horaFin	capacidad
PK	UA, CK between 0 and 23	UA, CK between 0 and 23	UA

- ReservaDeServicio

idReserva	idServicio	fechaInicio	fechaFin
PK, $\texttt{FK}_{ReservaHabitacion}$	$\texttt{FK}_{Servicio}$	UA	UA

- Piscina

idServicio	nombre
$\texttt{FK}_{Servicio}$, PK	UA

- Gimnasio

idServicio	nombre
$\texttt{FK}_{Servicio}$, PK	UA

- Internet

idServicio	idReserva	numeroDiasUso	costo
$\texttt{FK}_{Servicio}$, PK	$\texttt{FK}_{Reserva}$, PK	UA	UA

- Bar

idServicio	nombre
$\texttt{FK}_{Servicio}$, PK	UA

- Restaurante

idServicio	nombre	estilo
FK$_{Servicio}$, PK	UA	UA

- Supermercado

idServicio	nombre
FK$_{Servicio}$, PK	UA

- Tienda

idServicio	nombre	tipo
FK$_{Servicio}$, PK	UA	UA

- Spa

idServicio	nombre
FK$_{Servicio}$, PK	UA

- Lavado/planchado/embolado

idServicio	idReserva	tipoPrenda	numeroPrendas
FK$_{Servicio}$, PK	FK$_{Reserva}$, PK	UA	UA

- PrestamoUtensilios

idServicio	idReserva	recargoPorMalUso
FK$_{Servicio}$, PK	FK$_{Reserva}$, PK	UA

- SalonReuniones

idServicio	idReserva	horasUso	costoBase	costoAdicionalPorEquipos
FK$_{Servicio}$, PK	FK$_{idReserva}$, PK	UA	UA	UA

- SalonConferencias

idServicio	idReserva	horasUso	costo
FK$_{Servicio}$, PK	FK$_{Reserva}$, PK	UA	UA

- Producto

id	costo
NN, ND, PK	UA

- PlanDeConsumo

id	tipo
PK	UA

- LargaEstadía

idPlanDeConsumo	descuento	idHotel	tiempoEstadia
PK	UA	FK$_{Hotel}$	UA

- TiempoCompartido

idPlanDeConsumo	idServicioAsociado
PK, FK$_{PlanDeConsumo}$	FK$_{Servicio}$

- TodoIncluido

idPlanDeConsumo	idServicioAsociado	cuentaHabitacion	costoFijoTotal
PK, $\mathrm{FK}_{PlanDeConsumo}$	$\mathrm{FK}_{Servicio}$	UA	UA

- ProductosTodoIncluido

idPlanDeConsumo	idProductoAsociado	descuento
PK, $\mathrm{FK}_{PlanDeConsumo}$	$\mathrm{FK}_{Producto}$	UA

- PromocionParticular

idPlanDeConsumo	descripcion
PK, $\mathrm{FK}_{PlanDeConsumo}$	UA

En comparación con las tablas anteriores, hemos adicionado y hemos borrado algunas, sin embargo no vale la pena detallar cada una sino en general por qué tomamos estas decisiones. En principio, encontramos que teníamos varias tablas innecesarias que nos complicaban el proceso y finalmente vimos que era útil definir en general los clientes porque los podemos asociar a servicio.

3 ¿Cómo ejecutar el jdo?

Instalación del proyecto

Es posible acceder al código fuente de dos formas: con el archivo `.zip` o haciendo `clone` del siguiente repositorio: **link**. Posteriormente es necesario importar este proyecto en Eclipse.

Creación de la base de datos

Primero diríjase al archivo `src/main/resources/META-INF/persistence.xml` e inserte sus credenciales, ya que no va a poder el proyecto de otra forma.

Abra el archivo `data/secuenciasFinal.sql`, copie todo su contenido e insértelo en SQLDeveloper. Esto va a crear las bases de datos en su usuario.

Ejecución

Preferiblemente desde Eclipse, diríjase a `src/main/uniandes/isis2304/a12hotelandes/interfazApp/InterfazA12HotelAndesApp.java` y corra el archivo como una apicación de Java.

Si todo el proceso fue correcto, debería encontrar una ventana así:

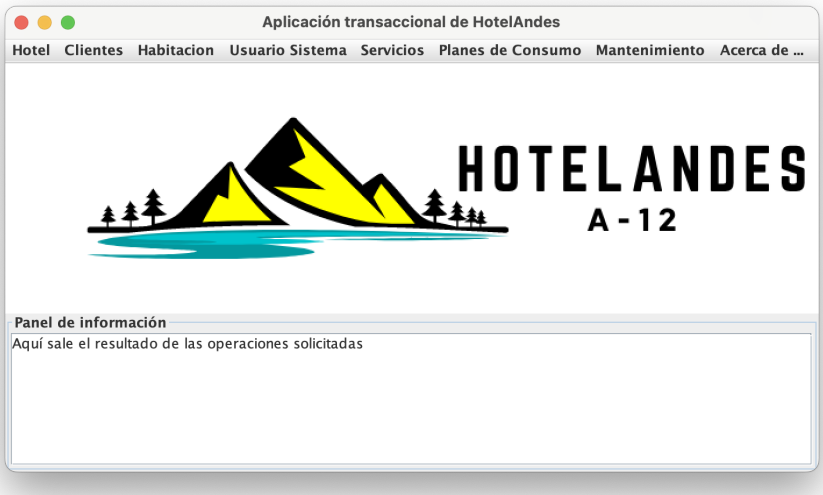

Fig. 2: Vista de la interfaz

Ejecución de sentencias SQL

Para las sentencias de los requerimientos funcionales de consulta, es necesario poblar la base de datos primero con datos ficticios y luego hacer las consultas.

Para poblar la base de datos, utilice las sentencias que se encuentran en `/a12-hotelandes-jdo/data/SentenciasPoblarTablas.sql`. Abra este archivo y ejecútelo como un script. Posteriormente, en `/a12-hotelandes-jdo/data/Sentencias de Consultas.sql` encontrará las sentencias para hacer las consultas que desee desde SQLDeveloper, teniendo en cuenta lo expuesto en la sección 4.2

4 Implementación

Para poder desarrollar los requerimientos que se detallan en el enunciado, realizamos las siguientes instrucciones.

4.1 Requerimientos funcionales

Sin explicar los pormenores de cómo funciona el `jdo` (porque esto ya está explicado), nos encargaremos de mostrar el patrón de implementación que seguimos para cada tabla.

- **Creación de la clase con el nombre de la tabla en el paquete de negocio**:

  Para cada tabla hay una clase en Java que tiene cada columna de la tabla como un atributo, un constructor vacío y otro con todos los atributos, *getters* y *setters* y un método `toString()`. Por ejemplo, la clase `Cliente.java`:

```
package uniandes.isis2304.a12hotelandes.negocio;
import java.sql.Date;

public class Cliente implements VOCliente {
    public String nombre;
    public String tipoDoc;
    public Integer numDoc;

    @Override
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    // ... demas getters y setters
    public Cliente() {
        super();
        this.nombre = "";
        this.tipoDoc = "";
        this.numDoc = 0;
    }
    public Cliente(String nombre, String tipoDoc, Integer numDoc) {
        super();
        this.nombre = nombre;
        this.tipoDoc = tipoDoc;
        this.numDoc = numDoc;
    }
    @Override
    public String toString() {
        return "Cliente [nombre=" + nombre + ", tipoDoc=" + tipoDoc + ",
            numDoc=" + numDoc + "]";
    }
}
```

Como podemos ver, esta clase extiende de la interfaz `VOCliente.java`. Esta interfaz es un *ValueObject* que pretende asegurar la integridad de los datos, ya que de esta manera no se pueden modificar estas clases desde la interfaz por error. La interfaz contiene los *getters* y el método `toString()`.

- **Creación de las clases `SQL` para cada tabla**

  Siguiendo con el ejemplo anterior, para la tabla Cliente se creó la clase `SQLCliente.java` en el paquete de persistencia, que contiene todas las sentencias `SQL` para hacer las operaciones básicas de C.R.U.D. a la tabla. Esto se logra de la siguiente manera:

  Para explicarlo solo pondremos algunos de los métodos relevantes. Primero, el constructor recibe un objeto de tipo `PersistenciaA12HotelAndes` para poder instanciarse.

```
public SQLCliente(PersistenciaA12HotelAndes pp) {
    super();
    this.pp = pp;
}
```

  Para adicionar a la tabla, es necesario un método que reciba todos los atributos de la misma y genere la sentencia `SQL`, luego tenemos el método

```
public long adicionarCliente (PersistenceManager pm, String nombre,
    String tipoDoc, Integer numDoc, Date diaEntrada, Date diaSalida)
{
    Query q = pm.newQuery(SQL, "INSERT INTO " + pp.darTablaCliente () + "
        (nombre, numdoc, tipodoc, diaentrada, diasalida) values (?, ?, ?,
         ?, ?)");
    q.setParameters(nombre, numDoc, tipoDoc, diaEntrada, diaSalida);
    return (long) q.executeUnique();
}
```

  Los métodos del Retrieve, Update y Delete funcionan muy parecido todos, así que mostraremos solo uno:

```
public long eliminarClientePorId (PersistenceManager pm, long idCliente)
{
    Query q = pm.newQuery(SQL, "DELETE FROM " + pp.darTablaCliente () + "
         WHERE id = ?");
    q.setParameters(idCliente);
    return (long) q.executeUnique();
}
```

- **Integración con los *singleton* de persistencia y clase principal**

  Como la clase `PersistenciaA12HotelAndes.java` funciona como un controlador de las clases SQL, esta clase hace un llamado de los métodos de las mismas, crea instancias de las SQL, y las conecta con las tablas de la DB.

  Como cada método debe completarse transaccionalmente, se hizo de la siguiente manera:

```
public long eliminarClientePorNombre (String nombreCliente)
{
    PersistenceManager pm = pmf.getPersistenceManager();
    Transaction tx=pm.currentTransaction(); // Aqui se asegura la
        transaccionalidad
    try
    {
        tx.begin();
        long resp = sqlCliente.eliminarClientesPorNombre(pm,
            nombreCliente);
        tx.commit();
        return resp;
    }
    catch (Exception e)
    {
        e.printStackTrace();
        log.error ("Exception : " + e.getMessage() + "\n" +
            darDetalleException(e));
        return -1;
    }
    finally
    {
        if (tx.isActive()) tx.rollback();
        pm.close();
    }
}
```

Luego en la clase principal `A12HotelAndes` se llaman los métodos escribiendo en el log:

```
public Cliente adicionarCliente(String nombreCliente, String tipoDoc,
    Integer numDoc) {
    log.info ("Adicionando cliente: " + nombreCliente);
    Cliente cliente = pp.adicionarCliente (nombreCliente, tipoDoc, numDoc
        );
    log.info ("Adicionando cliente: " + cliente);
    return cliente;
}
```

- **Integración en la interfaz gráfica**

  En la interfaz gráfica se crean los métodos que en el `interfaceConfigApp.json` se llaman por evento, de forma que se vea así en la interfaz:

  Como vemos en la figura 3, al hacer clic sobre cualquier menú de la barra de menús podemos acceder a las funcionalidades del programa. Después de hacer click se pueden ingresar entradas al programa mediante cajas de texto:

  Finalmente el usuario puede ver el estado de su transacción en el panel de información:

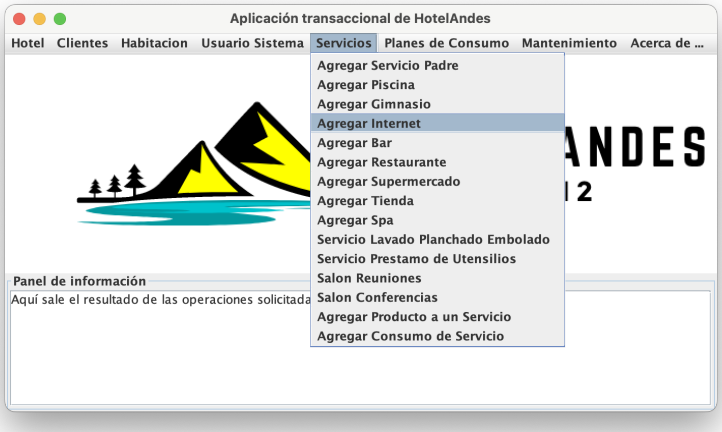

Fig. 3: Métodos para agregar servicios desde la interfaz gráfica

4.1.1 Satisfacción de los requerimientos

Después de haber explicado cómo hicimos para integrar cada tabla, pasamos a explicar la ruta para ejecutarlo

- **RF1:** REGISTRAR ROLES DE USUARIOS Usuario Sistema> Adicionar Roles
- **RF2:** REGISTRAR USUARIO Usuario Sistema> Adicionar Usuario
- **RF3:** REGISTRAR TIPO DE HABITACIÓN Habitacion> Adicionar tipo de habitación
- **RF4:** REGISTRAR HABITACIÓN Habitacion> Adicionar habitación
- **RF5:** REGISTRAR SERVICIO DE HOTEL Servicios> Agregar [servicio deseado]
- **RF6:** REGISTRAR PLAN DE CONSUMO Planes de Consumo > Agregar Plan de Consumo
- **RF7:** REGISTRAR RESERVA DE ALOJAMIENTO Habitacion > Hacer una reserva
- **RF8:** REGISTRAR RESERVA SERVICIO DE HOTEL Servicios > Agregar Reserva Servicio
- **RF9:** REGISTRAR LLEGADA CLIENTE AL HOTEL Clientes > Adicionar
- **RF10:** REGISTRAR CONSUMO DE UN SERVICIO Servicios > Agregar Consumo de servicio
- **RF11:** REGISTRAR SALIDA DE UN CLIENTE Clientes > Borrar por [nombre | id]

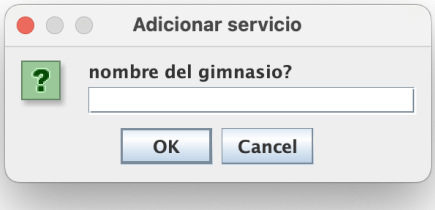

Fig. 4: Interacción con el método agregar gimnasio de Servicio.

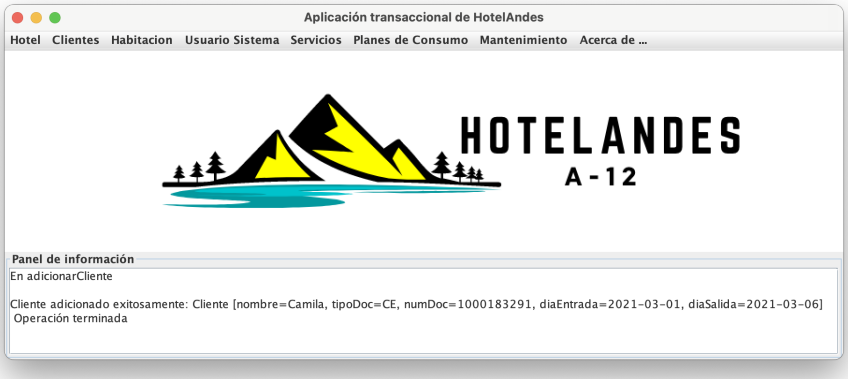

Fig. 5: Interfaz con respuesta en el panel después de una transacción exitosa

4.2 Requerimientos funcionales de consulta

Para satisfacer estos requerimientos era necesario hacer sentencias SQL sobre las que se hicieran búsquedas sobre nuestras tablas. Según el enunciado estas no se tenían que implementar dentro del jdo así que las vamos a enunciar aquí:

Nota: Para poder ejecutar cada sentencia es necesario cambiar los valores de `FECHA INICIO`, `FECHA FIN` y `COSTO`.

- **RFC1:** MOSTRAR EL DINERO RECOLECTADO POR SERVICIOS EN CADA HABITACIÓN DURANTE UN PERIODO DE TIEMPO Y EN EL AÑO CORRIDO

```
SELECT H.ID ID_HABITACION,SUM(P.COSTO* CS.CANTIDAD) CONSUMO
FROM HTA_CONSUMO_SERVICIO CS, HTA_RESERVA_HABITACION RH, HTA_HABITACION H
    , HTA_PRODUCTO P
WHERE RH.IDHABITACION = H.ID
```

```
    AND RH.ID = CS.IDRESERVA
    AND CS.IDPRODUCTO = P.IDPRODUCTO
    AND RH.DIAENTRADA > (DATE'FECHAINICIO')
    AND RH.DIASALIDA < (DATE'FECHAFIN')
GROUP BY H.ID;
```

- **RFC2:** MOSTRAR LOS 20 SERVICIOS MÁS POPULARES.

```
SELECT CS.IDSERVICIO, COUNT(DISTINCT CS.IDFACTURA)NUM_CONUMOS
FROM HTA_CONSUMO_SERVICIO CS, HTA_RESERVA_HABITACION RH
WHERE RH.ID = CS.IDRESERVA
    AND RH.DIAENTRADA > (DATE'FECHAINICIO')
    AND RH.DIASALIDA < (DATE'FECHAFIN')
GROUP BY IDSERVICIO
ORDER BY COUNT(DISTINCT IDFACTURA) DESC,IDSERVICIO;
```

- **RFC3:** MOSTRAR EL ÍNDICE DE OCUPACIÓN DE CADA UNA DE LAS HABITACIONES DEL HOTEL

```
SELECT H.ID, COUNT(RH.ID) NUM_RESERVAS
FROM HTA_RESERVA_HABITACION RH, HTA_HABITACION H
WHERE RH.IDHABITACION = H.ID
    AND RH.DIAENTRADA > (DATE'FECHAINICIO')
    AND RH.DIASALIDA < (DATE'FECHAFIN')
GROUP BY H.ID
ORDER BY H.ID;

\bigskip



\item \textbf{RFC4:} MOSTRAR LOS SERVICIOS QUE CUMPLEN CON CIERTA
    CARACTER STICA

\begin{lstlisting}[language = SQL]
SELECT *
FROM HTA_SERVICIO
WHERE CAPACIDAD < 'CAPACIDAD MAXIMA'
    AND CAPCIDAD > 'CAPACIDAD MINIMA';

SELECT *
FROM HTA_SERVICIO S
WHERE HORAINICIO> 'HORAINICIO'
    AND HORAFIN < 'HORAFIN'
```

- **RFC5:** MOSTRAR EL CONSUMO EN HOTELANDES POR UN USUARIO DADO, EN UN RANGO DE FECHAS INDICADO.

```
SELECT IDfACTURA, IDRESERVA, IDSERVICIO, IDPRODUCTO, CANTIDAD, C.NOMBRE
FROM HTA_CONSUMO_SERVICIO CS, HTA_RESERVA_HABITACION RH, HTA_CLIENTE C
WHERE RH.ID = CS.IDRESERVA
    AND RH.NUMDOCCLIENTE = C.NUMDOC
    AND RH.TIPODOCCLIENTE = C.TIPODOC
    AND RH.DIAENTRADA > (DATE'FECHAINICIO')
    AND RH.DIASALIDA < (DATE'FECHAFIN')
    AND C.TIPODOC = 'TIPODOC'
    AND C.NUMDOC = 'NUMDOC';
```

4.3 Requerimientos no funcionales

Para los requerimientos no funcionales, se tienen en cuenta los siguientes aspectos:

- **RNF1:** Seguridad
  Para identificarse en el sistema es importante tener un rol definido, algo que implementamos con el identificador de cada posible usuario del sistema. Asimismo, aspectos importantes en el cálculo de procesos que generan valor para la organización, como la fecha de entrada y de salida de un cliente, son atributos estáticos no modificables. De igual forma, para acceder a la base de datos es necesario ingresar las credenciales en el sistema.

- **RNF2:** Privacidad
  El sistema funciona mediante las clases VO que aseguran que las clases no van a poder modificarse por interfaces de manera indeseada.

- **RNF3:** Persistencia
  La base de datos funciona de manera transaccional, luego cada tupla insertada debería permanecer en el DB.

- **RNF4:** Concurrencia
  De nuevo, la base de datos funciona de manera transaccional, luego cada operación debe poder hacerse sin afectar otros procesos.

- **RNF5:** Distribución
  La base de datos de la aplicación está centralizada, ya que nos basamos en los conceptos de normalización vistos. Nuestra base de datos pretende estar en una forma normalizada FN3.

5 Cierre

5.1 Resultados logrados

En cuanto a los requerimientos funcionales (RF1-RF11) podemos decir que se cumplieron todos a cabalidad, ya que se pueden agregar todos los datos de forma correcta. De igual forma, estos cambios son transaccionales y se reflejan en las tablas de la base de datos. Por otro lado, las sentencias SQL también muestran los resultados esperados, y se puede consultar la información que piden los requerimientos.

5.2 Escenarios de prueba

Para cada tabla se pueden hacer las siguientes pruebas, que aquí se exponen como pantallazos en SQLDeveloper.

Pruebas de unicidad

Aquí se puede ver cómo probamos poner dos tablas con el mismo `PK` y no es posible:

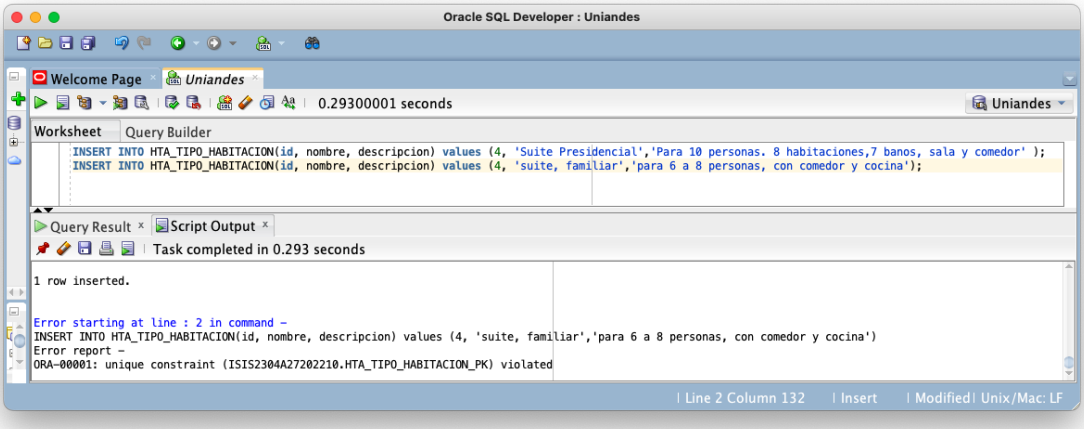

Fig. 6: Prueba de unicidad con mensaje de error correcto

Intentamos ingresar dos habitaciones con el mismo id (que es el `PK` y no fue posible.

Pruebas de integridad con Foreign Key

Aquí estamos probando que no se puede insertar un `FK` que no se haya creado antes:

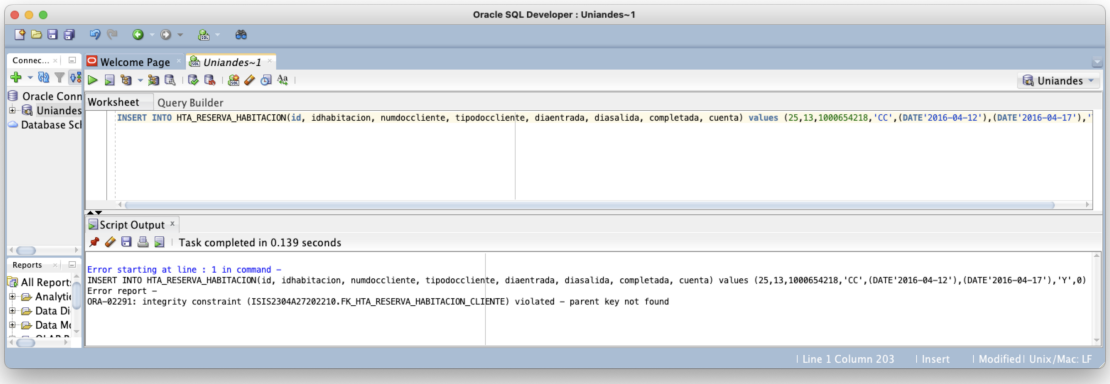

Fig. 7: Prueba de integridad con mensaje de error correcto

El foreign key en este caso es incorrecto, y se está encontrando el constraint `FK_HTA_RESERVA_HABITACION_CLIENTE`

Pruebas de integridad con Check

Aquí estamos probando que no se puede insertar un valor incorrecto gracias al constraint `CK`:

Fig. 8: Prueba de integridad con mensaje de error correcto

fff El valor `'CEX'` no está dentro de los valores posibles del tipo de documento, luego arroja el error correcto.

5.3 Resultados de las consultas

Para las consultas se utilizarán los siguientes datos como ejemplo (solo se muestran las tablas de mayor relevancia):

Habitacion

ID	CostoPorNoche	tipoHabitacion	aprovisionamiento
13	2000	4	Nevera llena, Wh...
14	2000	4	Nevera llena, Wh...
15	1200	5	Nevera llena, go...
16	1300	5	Nevera llena, go...
17	700	6	Nevera con bebi...
18	750	6	Nevera con bebi...
19	300	7	Televisor basico

TipoHabitacion

ID	nombre	descripcion
4	Suite Presidencial	Para 10 persona...
5	Suite Familiar	Para 6 a 8 perso...
6	Doble	Para 2 o 3 perso..
7	Sencilla	Para 1 persona...

Consumo Servicio

IDFactura	IDReserva	IDServicio	IDProducto	Cantidad
100	25	37	66	5
101	25	40	67	2
102	26	49	70	2
103	65	46	68	1
104	26	55	71	2
105	65	43	69	10
106	26	49	70	3

Reserva Habitacion

ID	IDHabitacion	Numero documento	TipoDocumento	DiaEntrada	DiaSalida	Completada	Cuenta
##	13	1000654218	CC	12/04/2016	17/04/2016	Y	0
##	16	1000985283	CC	25/11/2021	3/12/2021	Y	0
##	18	1000416291	CE	5/08/2020	12/08/2020	Y	0
##	15	10000985283	CC	28/12/2020	4/01/2020	N	0
##	18	1000416291	CE	12/04/2021	12/04/2021	Y	0

- Para consultar el dinero recolectado por servicios en cada habitacion entre 2015-04-12 y 2022-04-12:

```
SELECT H.ID ID_HABITACION,SUM(P.COSTO* CS.CANTIDAD) CONSUMO
FROM HTA_CONSUMO_SERVICIO CS, HTA_RESERVA_HABITACION RH, HTA_HABITACION H,
    HTA_PRODUCTO P
WHERE RH.IDHABITACION = H.ID
    AND RH.ID = CS.IDRESERVA
    AND CS.IDPRODUCTO = P.IDPRODUCTO
    AND RH.DIAENTRADA > (DATE'2015-04-12')
    AND RH.DIASALIDA < (DATE'2022-04-12')
GROUP BY H.ID;
```

Resultado:

	ID_HABITACION	CONSUMO
1	15	90
2	13	170
3	16	200

- Para consultar el dinero recolectado por servicios en cada habitacion entre 2021-04-12 y 2022-04-12:

```
SELECT H.ID ID_HABITACION,SUM(P.COSTO* CS.CANTIDAD) CONSUMO
FROM HTA_CONSUMO_SERVICIO CS, HTA_RESERVA_HABITACION RH, HTA_HABITACION H,
    HTA_PRODUCTO P
WHERE RH.IDHABITACION = H.ID
```

```
    AND RH.ID = CS.IDRESERVA
    AND CS.IDPRODUCTO = P.IDPRODUCTO
    AND RH.DIAENTRADA > (DATE'2021-04-12')
    AND RH.DIASALIDA < (DATE'2022-04-12')
GROUP BY H.ID;
```

	ID_HABITACION	CONSUMO
1	16	200

- Mostrando los 20 servicios más populares entre 2015-04-1 y 2022-04-1

```
SELECT CS.IDSERVICIO, COUNT(DISTINCT CS.IDFACTURA)NUM_CONUMOS
FROM HTA_CONSUMO_SERVICIO CS, HTA_RESERVA_HABITACION RH
WHERE RH.ID = CS.IDRESERVA
    AND RH.DIAENTRADA > (DATE'2015-04-12')
    AND RH.DIASALIDA < (DATE'2022-04-12')
GROUP BY IDSERVICIO
ORDER BY COUNT(DISTINCT IDFACTURA) DESC,IDSERVICIO;
```

	IDSERVICIO	NUM_CONUMOS
1	37	1
2	40	1
3	43	1
4	46	1
5	49	1
6	55	1

- Ahora agregamos un consumo al servicio 49.

```
SELECT H.ID, COUNT(RH.ID) NUM_RESERVAS
FROM HTA_RESERVA_HABITACION RH, HTA_HABITACION H
WHERE RH.IDHABITACION = H.ID
    AND RH.DIAENTRADA > (DATE'2015-04-12')
    AND RH.DIASALIDA <(DATE'2022-04-12')
GROUP BY H.ID
ORDER BY H.ID;
```

	IDSERVICIO	NUM_CONUMOS
1	49	2
2	37	1
3	40	1
4	43	1
5	46	1
6	55	1

- Para consultar el índice de ocupación de las habitaciones entre 2015-04-12 y 2022-04-12

```
SELECT H.ID, COUNT(RH.ID) NUM_RESERVAS
FROM HTA_RESERVA_HABITACION RH, HTA_HABITACION H
WHERE RH.IDHABITACION = H.ID
    AND RH.DIAENTRADA > (DATE'2015-04-12')
    AND RH.DIASALIDA < (DATE'2022-04-12')
GROUP BY H.ID
ORDER BY H.ID;
```

	ID	NUM_RESERVAS
1	13	1
2	15	1
3	16	1
4	18	1

- Ahora insertamos una nueva reserva a la habitacion 16 el 2021-04-12 y cambiamos la fecha de inicio para que la consulta vaya desde el 2019-04-12 hasta 2022-04-12

	ID	NUM_RESERVAS
1	15	1
2	16	1
3	18	2

- Consultando los servicios que tienen una capacidad entre 0 y 100 personas

```
SELECT *
FROM HTA_SERVICIO
WHERE CAPACIDAD < 100
    AND CAPACIDAD > 0;
```

- Consultando los servicios que tienen una capacidad entre 50 y 150 personas

```
SELECT *
FROM HTA_SERVICIO
WHERE CAPACIDAD < 150
    AND CAPACIDAD > 50;
```

- Conultar servicios que cumplan con tener un horario entre 8 de la mañana y 6 de la tarde.

```
SELECT *
FROM HTA_SERVICIO S
WHERE HORAINICIO> 8
    AND HORAFIN < 18
```

- Consultar consumos entre 2015-04-12 y 2022-04-12 del cliente con cedula 1000654218

	ID	HORAINICIO	HORAFIN	CAPA...
1	28	10	20	50
2	29	10	20	20
3	30	10	20	45
4	31	10	22	20
5	32	10	22	15
6	33	10	22	40
7	38	14	23	50
8	40	10	22	70
9	41	10	23	80
10	42	10	22	50
11	46	10	20	30
12	47	10	19	15
13	48	10	9	20
14	49	10	20	10
15	50	10	20	10
16	51	10	20	10
17	58	7	0	12
18	59	7	0	12
19	60	7	0	12

	ID	HORAINICIO	HORAFIN	CAPACIDAD
1	37	17	2	120
2	40	10	22	70
3	41	10	23	80
4	45	10	0	130

```
SELECT IDfACTURA, IDRESERVA, IDSERVICIO, IDPRODUCTO, CANTIDAD, C.NOMBRE
FROM HTA_CONSUMO_SERVICIO CS, HTA_RESERVA_HABITACION RH, HTA_CLIENTE C
WHERE RH.ID = CS.IDRESERVA
    AND RH.NUMDOCCLIENTE = C.NUMDOC
    AND RH.TIPODOCCLIENTE = C.TIPODOC
    AND RH.DIAENTRADA > (DATE'2015-04-12')
    AND RH.DIASALIDA < (DATE'2022-04-12')
    AND C.TIPODOC = 'CC'
    AND C.NUMDOC = 1000654218;
```

	ID	HORAI...	HORAFIN	CAPACIDAD
1	37	17	2	120
2	39	21	5	200
3	43	10	0	200
4	44	10	0	150
5	45	10	0	130
6	48	10	9	20
7	52	10	17	0
8	53	10	17	0
9	54	10	17	0

	IDFACTURA	IDRESERVA	IDSERVICIO	IDPRODUCTO	CANTIDAD	NOMBRE
1	100	25	37	66	5	Juan Diego Pardo
2	101	25	40	67	2	Juan Diego Pardo

6 Referencias

Oracle (2022) *Database SQL Reference*. Desde https://docs.oracle.com/cd/B19306_01/server.102/b14200/toc.htm

Oracle (2003) *Regexp_Like*. Desde https://docs.oracle.com/cd/B12037_01/server.101/b10759/conditions018.htm

Oracle (S.A) *4 Using Regular Expressions y Oracle Database*. Desde https://docs.oracle.com/cd/B19306_01/B1425101/adfnsregexp.htm

Hebbrecht, J. (2022). *Drop all tables in Oracle DB (scheme)* | JOCHEN HEBBRECHT. Recuperado de http://www.jochenhebbrecht.be/site/2010-05-10/database/drop-all-tables-in-oracle-db-scheme