UNIVERSIDAD
DE GRANADA

PRÁCTICA 3

Análisis y especificación de requisitos en un Sistema de Información

Diseño y Desarrollo de Sistemas de Información

GIIMADE - 2024/2025

Autores

Carmen Azorín Martí
Adrián Jaén Fuentes
Rafael Luque Framit
Pablo Luque Salguero
Jaime Martínez Bravo
Juan Pedro Moreno Ruiz

ÍNDICE

1. Descripción general 2
2. Requisitos funcionales 5
    2.1. Gestión de habitaciones - Carmen Azorín Martí 5
    2.2. Facturación y pago - Pablo Luque Salguero 9
    2.3. Trabajadores - Jaime Martínez Bravo 13
    2.4. Suministros - Adrián Jaén Fuentes 16
    2.5. Gestión de clientes - Rafael Luque Framit 19
    2.6. Servicios - Juan Pedro Moreno Ruiz 22
3. DFD Esquema Caja Negra 25
4. DFD Armazón 26
5. DFD de los Subsistemas 27
    5.1. Gestión de habitaciones 27
    5.2. Facturación y pago 28
    5.3. Trabajadores 28
    5.4. Suministros 29
    5.5. Gestión de Clientes 29
    5.6. Servicios 30
6. Esquemas Externos 30
    6.1. Gestión de habitaciones 30
    6.2. Facturación y pago 33
    6.3. Trabajadores 35
    6.4. Suministros 36
    6.5. Gestión de Clientes 36
    6.6. Servicios 38
7. Esquema E/R del Sistema 39
8. Conjunto de Tablas (paso a tablas) 40
9. Dependencias funcionales y normalización 42

1. Descripción general

Deseamos crear un sistema de información para la gestión de un hotel.
Por un lado, de cada usuario vamos a almacenar su nombre (cadena de caracteres de 20), sus apellidos (cadena de caracteres de 50), su NIF o DNI (cadena de caracteres de 9), su número de teléfono (cadena de caracteres de 20), su domicilio (cadena de caracteres de 50) y por último su email (cadena de caracteres de 30 con un símbolo @).

Los usuarios del hotel pueden tener el rol de Cliente o de Trabajador. En cuanto a los Clientes vamos a almacenar sobre ellos los puntos de cliente que tenga (número entero mayor o igual que 0), el rango (ENUM= {“INICIAL”, “AVANZADO”, “VIP”, “PLATINO”}) y la tarjeta de crédito o débito del cliente. En el caso de los Trabajadores se almacenará su domicilio (cadena de caracteres de 50), puesto (ENUM={“ADMINISTRADOR”, “RECEPCIONISTA”, …}) y nómina (cadena de caracteres de 10).

Para **dar de alta** a un Cliente el usuario deberá proporcionar el nombre, los apellidos, el NIF o DNI, el número de teléfono y el correo electrónico. Confirmando la inserción y dándole 0 puntos de cliente y el tipo INICIAL como rango. El NIF o DNI solo puede pertenecer a un usuario. Para **dar de baja** a un Cliente el usuario deberá proporcionar el NIF o DNI del Cliente, si existe una reserva o factura asociada a este DNI/NIF del cliente, no se puede borrar al cliente dando un error. Para **consultar un Cliente** debemos proporcionar el NIF o DNI del usuario y se nos devolverá todos los datos del Cliente. Por otro lado, si queremos **modificar la información de un Cliente** debemos pasar los mismos datos que cuando se da de alta al cliente: DNI/NIF, nombre, apellidos, número de teléfono, correo electrónico, puntos de cliente y rango. Si existe el DNI/NIF dado en la BD se deberá devolver la confirmación del cambio o sino, se devolverá un error. Por último, para **consultar el rango** del Cliente debemos pasar el NIF o DNI del usuario y nos devolverá el valor ENUM que poseé ese Cliente como rango.

El subsistema de facturación se encargará de gestionar los pagos. Para poder realizar pagos, el cliente deberá **añadir un método de pago**. Para ello debe introducir un número de tarjeta de crédito o débito (cadena de 20 caracteres numéricos). Esta tarjeta se asocia al cliente en un nuevo atributo con el nombre de “tarjeta”. Un cliente no puede tener más de una tarjeta asociada. Un método de pago se **elimina** si el cliente lo solicita. Para ello, el cliente debe introducir su NIF y el número de la tarjeta.

Una vez el cliente tenga un método de pago asociado y realiza el pago, se **genera una factura** que se almacena en el sistema y que registra el concepto (cadena de caracteres con un máximo de 150), el coste de la operación (número real positivo con 2 decimales), la fecha en la que se realiza el pago (con formato AAAA/MM/DD HH:MM), el NIF del cliente, un campo “reembolsada” (booleano) que se establece a falso y un identificador único para la factura (número entero positivo de 15 caracteres). Para **consultar una factura**, se deberá proporcionar el identificador de la factura y el sistema devolverá toda la información asociada a la factura.

En caso que sea necesario realizar un **reembolso de una factura**, se deberá proporcionar el identificador de la factura y un número de cuenta al que realizar el ingreso (cadena

alfanumérica de 24 caracteres). Posteriormente, quedará reflejado en la factura que ha sido reembolsada.

El subsistema de gestión de reservas se encargará de administrar las reservas y la disponibilidad de las habitaciones del hotel. Por un lado, un cliente (necesariamente registrado) podrá **reservar una única habitación de hotel** según las fechas de entrada y salida deseadas. Se le asignará la habitación dependiendo del tipo que desee (ENUM={"individual","doble","suite"}) y se generará una reserva con un identificador asociado y mostrando el coste total de la reserva. Este coste se calcula mediante el precio de la habitación por noche y multiplicando por la cantidad de días que se reserva. Además, si todas las habitaciones del tipo indicado ya están reservadas para esas fechas, no se crea la reserva y se muestra un error.

Durante la estancia, el cliente podrá **solicitar suplementos** (servicios adicionales como desayuno, cama extra, etc.), los cuales se registrarán y se integrarán en la factura final, siempre y cuando haya cantidad suficiente. Los suplementos se solicitan de uno en uno. Cada suplemento tiene asociado un precio fijo, que no depende del tiempo que se reserve.

El cliente también podrá **cancelar una reserva** eliminando la reserva del sistema y actualizando la disponibilidad de la habitación. Además, podrá **modificar una reserva**, cambiando la fecha de entrada o salida o el tipo de habitación, siempre que sea con 48 horas de antelación a la fecha de inicio de estancia guardada. Al igual que al reservar la habitación, se comprobará que haya habitaciones disponibles del tipo indicado.

Los trabajadores podrán **consultar el listado de reservas**, así como su estado, con detalles como el nombre del cliente, fechas y tipo de habitación.

El subsistema de **gestión de trabajadores** se encargará de controlar a los empleados del hotel, permitiendo **dar de alta/dar de baja** a un trabajador, así como **modificar sus datos** o sólo **consultarlos**. También se puede **mostrar el listado completo de trabajadores**. Todas estas acciones deben ejecutarse por un **administrador** (uno de los posibles puestos de un trabajador).

Para **dar de alta** a un trabajador se deberá proporcionar su DNI, nombre, apellidos, domicilio, número de teléfono, correo electrónico, puesto y nómina. Un mismo DNI sólo puede pertenecer a un único trabajador, por lo que será lo que se utilice como requisito de entrada para el resto de acciones ejecutadas sobre un trabajador (para comprobar si este existe en la base de datos). En el caso de querer **mostrar el listado de trabajadores**, se deberán especificar los campos por los que se quiera filtrar.

Todas las acciones devolverán una confirmación del resultado. Además, **consultar datos de un trabajador** y **mostrar listado de trabajadores** también mostrará los datos de 1 o varios empleados, respectivamente.

El subsistema de **gestión de suministros** se encargará de administrar los productos esenciales para el funcionamiento del hotel, como artículos de limpieza, ropa de cama, y alimentos para el restaurante, entre otros. Este sistema permitirá **consultar la información de los productos**, **filtrar productos por categoría o proveedor**, añad**ir nuevos artículos**, **modificar la cantidad en stock** de productos existentes, y **retirar productos obsoletos** o que ya no se utilicen.

Además, **gestionará alertas automáticas** para avisar cuando el inventario baje de los niveles mínimos, permitiendo garantizar que siempre haya disponibilidad de suministros críticos.

El **subsistema de servicios** se encargará de administrar las distintas actividades que tiene el hotel, teniendo distintas funciones si el usuario es un cliente o un administrador. Los administradores podrán **crear una actividad** indicando su nombre, su precio, el horario y el aforo que tendrá, cada actividad será identificada internamente con un ID que se asigna de manera automática al crearse. También podrán **eliminar una actividad** cuando lo deseen, eliminando, en el caso de que haya, las reservas de los clientes de esa misma actividad. La última acción que pueden hacer los administradores es **consultar filtrando la lista de actividades** para conocer los datos de todos los clientes apuntados a las mismas.

Por otro lado, los clientes podrán **reservar una actividad** en la que haya aforo disponible y su fecha coincida con la estancia de la persona en el hotel. Además podrán **eliminar la reserva** que haya creado anteriormente, que provoca la eliminación de dicha reserva del sistema y el aumento del aforo correspondiente. Por último, podrán ver una **lista de actividades** que los administradores hayan creado con anterioridad para consultar las actividades disponibles, así como sus fechas, el aforo restante y el precio que tiene.

2. Requisitos funcionales

2.1. Gestión de habitaciones - Carmen Azorín Martí

RF 1.1 - RESERVAR UNA HABITACIÓN		
Entrada	Agente externo: cliente Acción:solicitar reserva de hotel Requisito de datos de entrada RDE1.1	
BD	Requisito de datos de escritura RDW1.1 Requisito de datos de lectura RDR1.1	
Salida	Agente externo: cliente Acción: confirmación de resultado Requisito de datos de salida: RDS1.1	
RDE1.1	Datos de entrada de reserva de habitación. - NIF: Cadena de caracteres (9) - Fecha incio de estancia: con formato AAAA/MM/DD HH:MM - Fecha final de estancia: con formato AAAA/MM/DD HH:MM - Tipo de habitación: tipo ENUM={”individual”,”doble”,”suite”}	
RDW1.1	Datos para creación de una reserva: - Número de reserva: Número entero positivo - Fecha incio de estancia: con formato AAAA/MM/DD HH:MM - Fecha final de estancia: con formato AAAA/MM/DD HH:MM - Número de habitación asignado a la reserva: Número entero positivo - NIF: Cadena de caracteres (9)	
RDR1.1	Datos de comprobación para asignar la habitación y calcular su coste: - Número de reserva: Número entero positivo - Número de habitación asignado a la reserva: Número entero positivo - Fecha incio de estancia: con formato AAAA/MM/DD HH:MM - Fecha final de estancia: con formato AAAA/MM/DD HH:MM - Tipo de habitación: tipo ENUM={”individual”,”doble”,”suite”} - Precio de habitación por noche: Número racional positivo con 2 decimales	
RDS1.1	Datos de confirmación de reserva de habitación: - Número de reserva: Número entero positivo - Coste de la reserva: Número racional positivo con 2 decimales	
RS1.1	Descripción	Si todas las habitaciones del tipo indicado ya se corresponden con alguna reserva en las mismas fechas, no se crea la nueva reserva y se muestra un error.
	RD(s)	RDE1.1, RDR1.1

RF 1.2 - CANCELAR RESERVA	
Entrada	Agente externo: cliente Acción: solicitar cancelación de reserva Requisito de datos de entrada RDE1.2
BD	Requisito de datos de escritura RDW1.2
Salida	Agente externo: cliente Acción: confirmación de resultado Requisito de datos de salida: ninguno
RDE1.2	Datos de entrada de cancelación de reserva: - Número de reserva: Número entero positivo
RDW1.2	Datos de reserva para eliminar: - Los mismos que RDW1.1

RF 1.3 - MODIFICAR RESERVA	
Entrada	Agente externo: cliente Acción: solicitar modificación de una reserva Requisito de datos de entrada RDE1.3
BD	Requisito de datos de escritura RDW1.3
Salida	Agente externo: cliente Acción: confirmación de resultado Requisito de datos de salida: RDS1.3
RDE1.3	Datos de entrada de reserva: - Número de reserva: Número entero positivo - Fecha incio de estancia nuevo: con formato AAAA/MM/DD HH:MM - Fecha final de estancia nuevo: con formato AAAA/MM/DD HH:MM - Tipo de habitación nuevo: tipo ENUM={”individual”,”doble”,”suite”}
RDW1.3	Datos de la modificación de la reserva: - Número de reserva: Número entero positivo - Número de habitación guardado: Número entero positivo - Fecha incio de estancia guardado: con formato AAAA/MM/DD HH:MM - Fecha final de estancia guardado: con formato AAAA/MM/DD HH:MM - Tipo de habitación guardado: tipo ENUM={”individual”,”doble”,”suite”} - Número de habitación nuevo: Número entero positivo - Fecha incio de estancia nuevo: con formato AAAA/MM/DD HH:MM - Fecha final de estancia nuevo: con formato AAAA/MM/DD HH:MM - Tipo de habitación nuevo: tipo ENUM={”individual”,”doble”,”suite”}
RDR1.3	Datos de la reserva modificada: - Número de reserva: Número entero positivo - Número de habitación asignado a la reserva: Número entero positivo

	- Precio de habitación por noche: Número racional positivo con 2 decimales	
RDS1.3	Nuevos datos de reserva: - Los mismos que RDS1.1	
RS1.3.1	Descripción	Si la modificación se solicita con menos de 48h antes de la fecha de inicio, no se actualizan los datos y se muestra un error.
	RD(s)	RDE1.3
RS1.3.2	Descripción	Si todas las habitaciones del tipo indicado ya se corresponden con alguna reserva en las mismas fechas, no se modifica la reserva y se muestra un error.
	RD(s)	RDE1.3, RDR1.1

RF 1.4 - MOSTRAR LISTADO DE RESERVAS	
Entrada	Agente externo: administrador Acción: solicitar listado de reservas Requisito de datos de entrada: ninguno
BD	Requisito de datos de lectura RDR1.4
Salida	Agente externo: administrador Acción: confirmación resultado Requisito de datos de salida: RDS1.4
RDR1.4	Datos de reserva almacenado: - Número de habitación asignado: Número entero positivo - Número de reserva: Número entero positivo - NIF: Cadena de caracteres (9) - Nombre: Cadena de caracteres (50) - Apellidos: Cadena de caracteres (50) - Fecha incio de estancia: con formato AAAA/MM/DD HH:MM - Fecha final de estancia: con formato AAAA/MM/DD HH:MM - Tipo de habitación: tipo ENUM={”individual”,”doble”,”suite”} - Precio de habitación por noche: Número racional positivo con 2 decimales
RDS1.4	Listado de registros, cada uno de ellos con los mismos datos de RDR1.4

RF 1.5 - GESTIONAR SUPLEMENTOS	
Entrada	Agente externo: cliente Acción: solicitar añadir suplementos Requisito de datos de entrada RDE1.5
BD	Requisito de datos de escritura RD1.5

Salida	Agente externo: cliente Acción: confirmación de resultado Requisito de datos de salida: ninguno	
RDE1.5	Datos de entrada de la reserva: - Número de reserva: Número entero positivo - Nombre de suplemento: tipo ENUM={”cama extra”,”cuna”,”desayuno”}	
RDW1.5	Datos del suplemento: - Número de reserva: Número entero positivo - Nombre de suplemento: tipo ENUM={”cama extra”,”cuna”,”desayuno”} - Cantidad disponible: Número entero no negativo	
RDR1.5	Comprobar cantidad de suplemento: - Nombre de suplemento: tipo ENUM={”cama extra”,”cuna”,”desayuno”} - Cantidad disponible: Número entero no negativo - Precio de suplemento: Número racional positivo con 2 decimales	
RS1.5	Datos de suplemento: - Precio de suplemento: Número racional positivo con 2 decimales	
RS 1.5	Descripción	Si la cantidad disponible del suplemento indicado es cero, no se inserta el suplemento y se muestra un error.
	RD(s)	RDR1.5

2.2. Facturación y pago - Pablo Luque Salguero

RF 2.1 - AÑADIR MÉTODO DE PAGO		
Entrada	Agente externo: cliente Acción: Solicitar inserción de método de pago Requisito de datos de entrada RDE2.1	
BD	Requisito de datos de escritura RDW2.1. Requisito de datos de lectura RDR2.1	
Salida	Agente externo: cliente Acción: confirmación de resultado Requisito de datos de salida: ninguno	
RDE2.1	Datos de entrada de método de pago. - NIF: Cadena de caracteres (9) - Tarjeta: Cadena numérica (20)	
RDW2.1	Datos del cliente: - Los mismos que RDE2.1	
RDR2.1	Datos del cliente para comprobar que no tenga tarjeta asignada: - Los mismos que RDW2.1	
RS2.1.1	Descripción	Si el cliente ya tiene un método de pago asociado, no se añade el nuevo método y se devuelve un error
	RD(s)	RDE2.1, RDR2.1
RS2.1.2	Descripción	Si no existe un cliente con el NIF dado se devuelve un error
	RD(s)	RDE2.1, RDR2.1

RF 2.2 - ELIMINAR MÉTODO DE PAGO	
Entrada	Agente externo: cliente Acción: Solicitar eliminación de método de pago Requisito de datos de entrada RDE2.2
BD	Requisito de datos de escritura RDW2.2. Requisito de datos de lectura RDR2.2
Salida	Agente externo: cliente Acción: confirmación de resultado Requisito de datos de salida: ninguno
RDE2.2	Datos de entrada de eliminación del método de pago. - NIF: Cadena de caracteres (9)

RDW2.2	Datos del cliente: - NIF: Cadena de caracteres (9) - Tarjeta: Cadena numérica (20)	
RDR2.2	Datos del cliente para comprobar que tenga tarjeta asignada: - NIF: Cadena de caracteres (9) - Tarjeta: Cadena numérica (20)	
RS2.2.1	Descripción	Si no existe tarjeta asociada a ese cliente, no se hace el borrado y se devuelve un error
	RD(s)	RDE2.2, RDR2.2
RS2.2.2	Descripción	Si no existe un cliente con el NIF dado se devuelve un error
	RD(s)	RDE2.2, RDR2.2

RF 2.3 - GENERAR FACTURA		
Entrada	Agente externo: cliente Acción: Solicitar pago Requisito de datos de entrada RDE2.3	
BD	Requisito de datos de escritura RDW2.3. Requisito de datos de lectura RDR2.3 (para comprobar el coste de la reserva)	
Salida	Agente externo: cliente Acción: confirmación de resultado Requisito de datos de salida: RDS2.3	
RDE2.3	Datos de entrada de la factura. - Número de reserva	
RDW2.3	Datos de la factura. - Número de reserva - Concepto: cadena de caracteres (150) - Fecha: con formato AAAA/MM/DD HH:MM - Reembolsada: booleano - Identificador: número entero positivo	
RDR2.3	Datos de la reserva: - Número de reserva - NIF del cliente Datos del cliente. - NIF del cliente - Tarjeta	
RDS2.3	Identificador de la factura: número entero positivo (15)	
RS2.3.1	Descripción	Si no existe la reserva con el identificador dado se devuelve un error

	RD(s)	RDE2.3, RDR2.3
RS2.3.2	Descripción	Si el cliente no tiene una tarjeta asociada, no se genera la factura y se devuelve un error
	RD(s)	RDE2.3, RDR2.3

RF 2.4 - CONSULTAR FACTURA	
Entrada	Agente externo: cliente Acción: Solicitar información de la factura Requisito de datos de entrada RDE2.4
BD	Requisito de datos de lectura RDR2.4
Salida	Agente externo: cliente Acción: información de la factura Requisito de datos de salida: RDS2.4
RDE2.4	Los mismos que RDS2.3
RDR2.4	Los mismos que RDW2.3
RDS2.4	Datos de la factura. Los mismos que RDW2.3

RF 2.5 - REEMBOLSAR FACTURA		
Entrada	Agente externo: administrador Acción: Reembolsar el coste de una factura Requisito de datos de entrada RDE2.5	
BD	Requisito de datos de lectura RDR2.5 Requisito de datos de escritura RDW2.5	
Salida	Agente externo: administrador Acción: confirmación de resultado Requisito de datos de salida: ninguno	
RDE2.5	Identificador de la factura: número entero positivo	
RDW2.5	Datos de la factura: - Identificador de factura - Reembolsada	
RDR2.5	Datos de la factura: - Identificador de factura - Reembolsada	
RS2.5.1	Descripción	Si la factura ya ha sido reembolsada no se realiza el reembolso y se devuelve un error
	RD(s)	RDE2.5, RDR2.5

RS2.5.2	Descripción	Si la factura dada no existe no se realiza el reembolso y se devuelve un error
	RD(s)	RDE2.5, RDR2.5

2.3. Trabajadores - Jaime Martínez Bravo

RF 3.1 - DAR DE ALTA TRABAJADOR		
Entrada	Agente externo: administrador Acción: solicitar añadir un nuevo trabajador a la base de datos Requisito de datos de entrada RDE3.1	
BD	Requisito de datos de escritura RDW3.1 Requisito de datos de lectura RDR3.1	
Salida	Agente externo: administrador Acción: confirmación de resultado Requisito de datos de salida: ninguno	
RDE3.1	Datos de entrada del trabajador: - DNI: cadena de caracteres (9) - Nombre: cadena de caracteres (20) - Apellidos: cadena de caracteres (50) - Domicilio: cadena de caracteres (50) - Teléfono: cadena de caracteres (20) - Email: cadena de caracteres (50) - Puesto: tipo ENUM {“ADMINISTRADOR”, “RECEPCIONISTA”, …} - Nómina: número real positivo con 2 decimales - Fecha último aumento: fecha	
RDW3.1	Datos del trabajador: - Los mismos que RDE3.1	
RDR3.1	Datos del trabajador para comprobar que no existe ya: - DNI: cadena de caracteres (9)	
RS3.1	Descripción	Si ya había un usuario con el mismo DNI/NIF, no se inserta el nuevo trabajador y se devuelve un error
	RD(s)	RDW3.1 (si lo comprueba el SGBD, si no RDE3.1 + RDR3.1)

RF 3.2 - DAR DE BAJA TRABAJADOR	
Entrada	Agente externo: administrador Acción: solicitar eliminar un trabajador de la base de datos Requisito de datos de entrada RDE3.2
BD	Requisito de datos de escritura RDW3.2 Requisito de datos de lectura RDR3.2
Salida	Agente externo: administrador Acción: confirmación de resultado

	Requisito de datos de salida: ninguno	
RDE3.2	Datos de entrada para buscar al trabajador: - Los mismos que RDR3.1	
RDW3.2	Datos de trabajador: - Los mismos que RDE3.1	
RDR3.2	Datos del trabajador para comprobar que existe: - Los mismos que RDR3.1	
RS3.2	Descripción	Si no había un usuario con el mismo DNI/NIF, no se puede eliminar el trabajador y se devuelve un error
	RD(s)	RDW3.2 (si lo comprueba el SGBD, si no RDE3.2 + RDR3.2)

RF 3.3 - MODIFICAR DATOS TRABAJADOR		
Entrada	Agente externo: administrador Acción: solicitar modificación de datos de un trabajador Requisito de datos de entrada RDE3.3	
BD	Requisito de datos de escritura RDW3.3 Requisito de datos de lectura RDR3.3	
Salida	Agente externo: administrador Acción: confirmación de resultado Requisito de datos de salida: ninguno	
RDE3.3	Datos de entrada del trabajador: - Los mismos que RDE3.1	
RDW3.3	Datos del trabajador a modificar: - Los mismos que RDE3.3	
RDR3.3	Datos del trabajador para comprobar que existe: - Los mismos que RDR3.1	
RS3.3	Descripción	Si no había un usuario con el mismo DNI/NIF, no se puede modificar los datos y se devuelve un error
	RD(s)	RDW3.3 (si lo comprueba el SGBD, si no RDE3.3 + RDR3.3)

RF 3.4 - CONSULTAR DATOS TRABAJADOR	
Entrada	Agente externo: administrador Acción: solicitar datos de un trabajador Requisito de datos de entrada RDE3.4
BD	Requisito de datos de lectura RDR3.4

Salida	Agente externo: administrador Acción: confirmación de resultado Requisito de datos de salida RDS3.4	
RDE3.4	Datos de entrada para buscar al trabajador: - Los mismos que RDR3.1	
RDR3.4	Datos del trabajador: - Los mismos que RDE3.1	
RDS3.4	Datos del trabajador: - Los mismos que RDE3.1	
RS3.4	Descripción	Si no había un usuario con el mismo DNI/NIF, no se puede consultar sus datos y se devuelve un error
	RD(s)	RDW3.4 (si lo comprueba el SGBD, si no RDE3.4 + RDR3.4)

RF 3.5 - MOSTRAR LISTADO TRABAJADORES	
Entrada	Agente externo: administrador Acción: solicitar listado de trabajadores Requisito de datos de entrada RDE3.5
BD	Requisito de datos de lectura RDR3.5
Salida	Agente externo: administrador Acción: confirmación de resultado Requisito de datos de salida: RDS3.5
RDE3.5	Datos de entrada para filtrar resultados: - Los mismos que RDE3.1
RDR3.5	Datos de trabajadores almacenados: - Los mismos datos RDE3.1
RDS3.5	Listado de registros, cada uno de ellos con los mismos datos de RDR3.5

2.4. Suministros - Adrián Jaén Fuentes

RF 4.1 - CONSULTAR INFORMACIÓN DE PRODUCTO		
Entrada	Agente externo: Administrador. Acción: Solicitar consulta de información de producto por ID o nombre. Requisito de datos de entrada RDE4.1.	
BD	Requisito de datos de lectura RDR4.1.	
Salida	Agente externo: cliente. Acción: confirmación de resultado. Requisito de datos de salida: RDS4.1.	
RDE4.1	- ID del producto: Número entero.	
RDR4.1	Información almacenada del producto: - Nombre del producto: Cadena de caracteres (100). - Cantidad de stock: Número entero. - Proveedor: Cadena de caracteres (50). - Fecha de última reposición: Fecha (AAAA/MM/DD HH).	
RDS4.1	Información almacenada del producto : - Los mismos datos que RDR4.1.	
RS4.1	Descripción	Si no hay un producto con el mismo ID, no se pueden consultar sus datos y se devuelve un error
	RD(s)	RDE4.1 + RDR4.1

RF 4.2 - FILTRAR PRODUCTOS POR PROVEEDOR	
Entrada	Agente externo: Administrador. Acción: Solicitar filtrado de productos. Requisito de datos de entrada RDE4.2
BD	Requisito de datos de lectura RDR 4.2
Salida	Agente externo: Administrador. Acción: Mostrar lista de productos filtrados. Requisito de datos de salida RDS 4.2:
RDS4.2	Lista de productos filtrados: - ID del producto: Número entero. - Nombre: Cadena de caracteres (100). - Cantidad de stock: Número entero. - Fecha de última reposición: Fecha (AAAA/MM/DD HH).
RDE4.2	- ID del producto: Número entero. - Proveedor del producto: Cadena de caracteres (50).

RDR4.2	- ID del producto: Número entero. - Nombre del producto: Cadena de caracteres (100). - Proveedor del producto: Cadena de caracteres (50). - Cantidad de stock: Número entero. - Fecha de última reposición: Fecha (AAAA/MM/DD HH).

RF 4.3 - AÑADIR NUEVO PRODUCTO		
Entrada	Agente externo: Administrador. Acción: Solicitar inserción de nuevo producto. Requisito de datos de entrada RDE4.3	
BD	Requisito de datos de lectura RDR4.3 Requisito de datos de escritura RDW4.3	
Salida	Agente externo: Administrador. Acción: Confirmación de añadido. Requisito de datos de salida ninguno:	
RDE4.3	Datos del nuevo producto: - Nombre del producto: Cadena de caracteres (100). - Cantidad de stock: Número entero. - Proveedor: Cadena de caracteres (50). - Fecha de última reposición: Fecha (AAAA/MM/DD HH).	
RDR4.3	Datos para comprobar duplicidad: - Nombre del producto: Cadena de caracteres (100). - Proveedor: Cadena de caracteres (50).	
RDW4.3	- ID del producto: Número entero. - Nombre del producto: Cadena de caracteres (100). - Cantidad de stock: Número entero. - Proveedor: Cadena de caracteres (50). - Fecha de última reposición: Fecha (AAAA/MM/DD HH).	
RDS4.3	- ID del producto: Número entero.	
RS4.3	Descripción	Si ya hay un producto con el mismo nombre y proveedor, no se puede añadir y se devuelve un error.
	RD(s)	RDW4.3 (si lo comprueba el SGBD, si no RDE3.4 + RDR3.4)

RF 4.4 - MODIFICAR LA CANTIDAD DE UN PRODUCTO	
Entrada	Agente externo: Administrador. Acción: Solicitar modificación de stock. Requisito de datos de entrada RDE4.4.
BD	Requisito de datos de escritura RDW4.4.

Salida	Agente externo: Administrador. Acción: Confirmación de la modificación de la cantidad. Requisito de datos de salida: Ninguno.
RDW4.4	Datos del producto a modificar: - Cantidad de stock: Número entero. - Fecha de última reposición: Fecha (AAAA/MM/DD HH).
RDE4.4	- ID del producto: Número entero. - Nueva cantidad de stock: Número entero.

RF 4.5 - RETIRAR PRODUCTO		
Entrada	Agente externo: Administrador. Acción: Solicitar eliminación de un producto. Requisito de datos de entrada RDE4.5	
BD	Requisito de datos de escritura RDW 4.5: Eliminar registro del producto	
Salida	Agente externo: Administrador. Acción: Confirmación de la eliminación del producto. Requisito de datos de salida: Ninguno.	
RDW4.5	Datos del producto a modificar: - Eliminación del producto de la base de datos	
RDE4.5	- ID del producto: Número entero.	
RDR4.5	Datos para comprobar existencia: - ID del producto: Número entero.	
RS4.5	Descripción	Si no hay un producto con el mismo ID, no se pueden consultar sus datos y se devuelve un error
	RD(s)	RDW4.5 (si lo comprueba el SGBD, si no RDE3.5 + RDR3.5)

2.5. Gestión de clientes - Rafael Luque Framit

RF 5.1 - DAR DE ALTA CLIENTE		
Entrada	Agente externo: cliente Acción: solicitar inserción Requisito de datos de entrada RDE5.1	
BD	Requisito de datos de escritura RDW5.1	
Salida	Agente externo: cliente Acción: confirmación de resultado Requisito de datos de salida: ninguno	
RDE5.1	Datos de entrada de alta de cliente: - DNI/NIF: Cadena de caracteres (9) - Nombre: Cadena de caracteres (50) - Apellidos: Cadena de caracteres (50) - Domicilio: cadena de caracteres (50) - Teléfono: Cadena de caracteres (20) - Email: Cadena de caracteres con un símbolo @ - Puntos de cliente: número entero mayor o igual que 0 - Rango: tipo ENUM {“INICIAL”, “AVANZADO”, “VIP”, “PLATINO”} - Tarjeta: Cadena de caracteres (16)	
RDW5.1	- Datos almacenados del cliente: - Los mismos datos RDE5.1	
RDR5.1	Datos del cliente para comprobar que no existe ya: - DNI/NIF: cadena de caracteres (9)	
RS5.1	Descripción	Si ya había un usuario con el mismo DNI/NIF, no se inserta el nuevo cliente y se devuelve un error.
	RD(s)	RDW5.1 (RDE5.1 + RDR5.1)

RF 5.2 - DAR DE BAJA CLIENTE	
Entrada	Agente externo: cliente Acción: solicitar borrado Requisito de datos de entrada RDE5.2
BD	Requisito de datos de escritura RDW5.2
Salida	Agente externo: cliente Acción: confirmación de resultado Requisito de datos de salida: ninguno
RDE5.2	Datos de entrada de baja de cliente:

	- DNI/NIF: Cadena de caracteres (9)	
RDW5.2	Datos almacenados del cliente: - Los mismos datos RDW5.1	
RDR5.2	Datos almacenados de reservas y facturas: - Mismos datos que RDW1.2 y RDW2.3	
RS5.2	Descripción	Si existe un reserva o una factura asociada a ese DNI/NIF del cliente, no se puede eliminar el cliente y se devuelve un error
	RD(s)	RDW5.2 (RDE5.2 + RDR.5.2)

RF 5.3 - CONSULTAR UN CLIENTE	
Entrada	Agente externo: administrador Acción: solicitar listado Requisito de datos de entrada RDE5.3
BD	Requisito de datos de lectura RDR5.3
Salida	Agente externo: administrador Acción: confirmación de resultado Requisito de datos de salida: RDS5.3
RDE5.3	Datos de entrada de consulta de información de cliente: - DNI/NIF: Cadena de caracteres (9) (Los mismos datos que RDE5.2)
RDR5.3	Datos almacenados del cliente: - Los mismos datos RDW5.1
RDS5.3	Tipo Cliente: mismos datos que se introducen en RF5.1

RF 5.4 - MODIFICAR INFORMACIÓN DE CLIENTE	
Entrada	Agente externo: administrador Acción: solicitar modificación de datos de un cliente Requisito de datos de entrada RDE5.4
BD	Requisito de datos de escritura RDW5.4
Salida	Agente externo: administrador Acción: confirmación de resultado Requisito de datos de salida: ninguno
RDE5.4	Datos de entrada del cliente: - Los mismos datos que RDE5.1
RDW5.4	Datos del cliente a modificar: - Los mismos datos RDE5.4
RDR5.4	Datos del cliente para comprobar que existe:

	- DNI/NIF: Cadena de caracteres (9)	
RS5.4	Descripción	Si no existe un usuario con ese DNI/NIF, se devuelve un error.
	RD(s)	RDW5.4 (RDE5.4 + RDR5.4)

RF 5.5 - CONSULTAR RANGO DE CLIENTE	
Entrada	Agente externo: administrador Acción: solicitar listado Requisito de datos de entrada RDE5.5
BD	Requisito de datos de lectura RDR5.5
Salida	Agente externo: administrador Acción: confirmación de resultado Requisito de datos de salida: Requisitos de datos de salida RDS5.5
RDE5.5	Datos de entrada de consulta de rango: - DNI/NIF: Cadena de caracteres (9) (Los mismos datos que RDE5.2)
RDR5.5	Datos del cliente almacenados: - DNI/NIF: Cadena de caracteres (9) - Rango: tipo ENUM {“INICIAL”, “AVANZADO”, “VIP”, “PLATINO”}
RDS5.5	Tipo Rango, el rango del cliente consultado.

2.6. Servicios - Juan Pedro Moreno Ruiz

RF 6.1 - REGISTRAR ACTIVIDAD	
Entrada	Agente externo: Administrador Acción: solicitar añadir una actividad a la base de datos Requisito de datos de entrada RDE6.1
BD	Requisito de datos de escritura RDW6.1
Salida	Agente externo: Administrador Acción: confirmación de resultado Requisito de datos de salida: ninguno
RDE6.1	Datos de la actividad: - ID: número entero positivo - Nombre: Cadena de caracteres (50) - Precio: número real positivo con 2 decimales - Horario: con formato DD/MM/AAAA HH:MM - Aforo: número entero positivo
RDW6.1	Datos almacenados de la actividad: - Los mismos datos que RDE6.1
RDS6.1	Mensaje de confirmación indicando el ID de la actividad

RF 6.2 - ELIMINAR ACTIVIDAD	
Entrada	Agente externo: Administrador Acción: solicitar eliminar una actividad a la base de datos Requisito de datos de entrada RDE6.2
BD	Requisito de datos de escritura RDW6.2 Requisito de datos de lectura RDR6.2
Salida	Agente externo: administrador Acción: confirmación de resultado Requisito de datos de salida: ninguno
RDE6.2	Datos de entrada de eliminación de la actividad: - ID: número entero positivo
RDW6.2	Datos de la actividad: - Los mismos que en RDW6.1
RDR6.2	Datos de la actividad para comprobar que existe: - Los mismos que RDE6.1

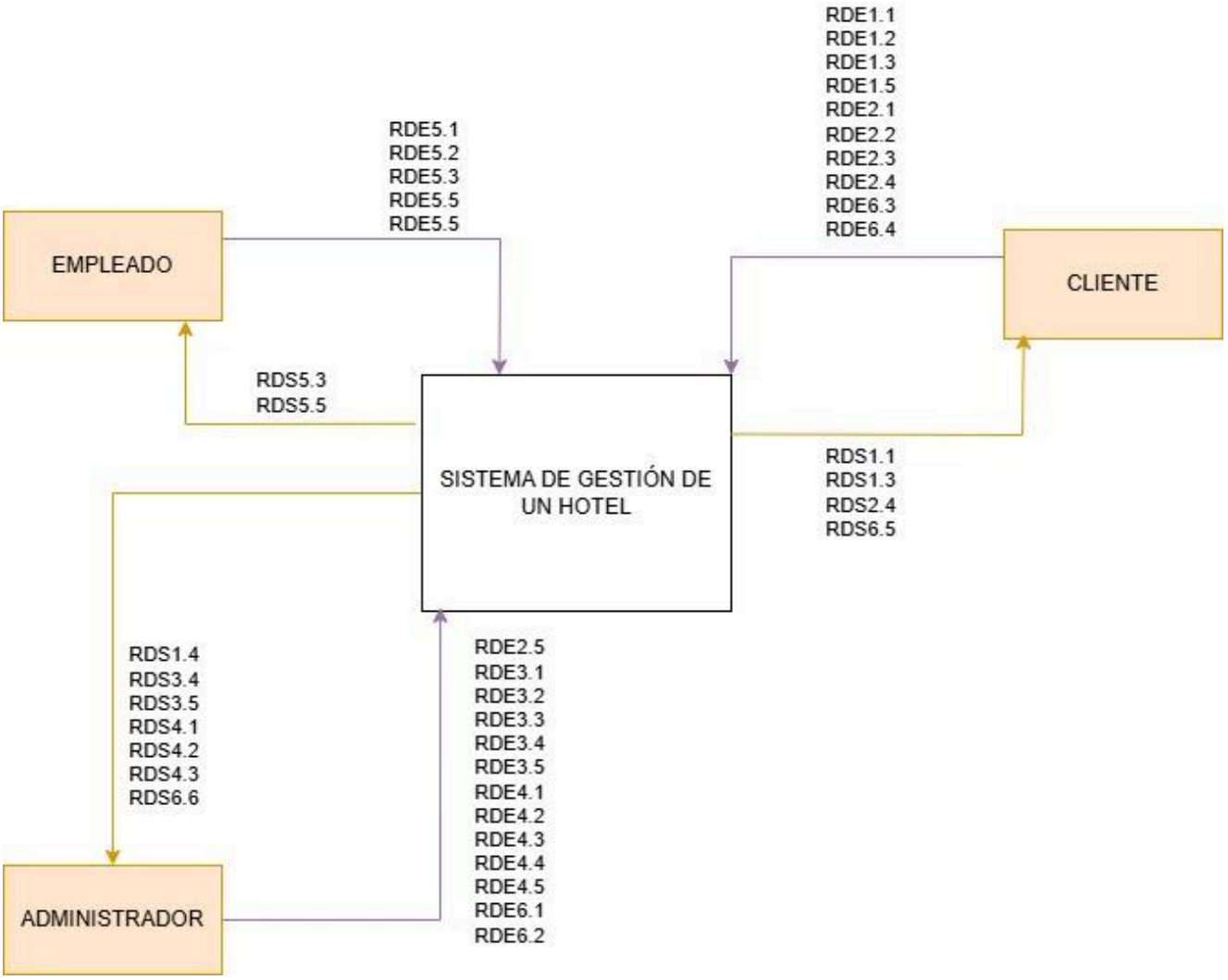
RF 6.3 - CONTRATAR ACTIVIDAD		
Entrada	Agente externo: Cliente Acción: solicitar apuntarse a una actividad Requisito de datos de entrada RDE6.3	
BD	Requisito de datos de escritura RDW6.3	
Salida	Agente externo: Cliente Acción: confirmación de resultado Requisito de datos de salida: ninguno	
RDE6.3	Datos de entrada para reservar actividad: - NIF: Cadena de caracteres (9) - Tipo de actividad	
RDW6.3	Datos almacenados de la reserva de actividad: - Los mismos datos que RDE6.3	
RS6.3.1	Descripción	Si un cliente ya tenía una actividad e intenta reservar otra a la vez, no se inserta la nueva reserva y se devuelve un error.
	RD(s)	RDW6.3

RF 6.4 - CANCELAR ACTIVIDAD	
Entrada	Agente externo: Cliente Acción: solicitar cancelar una actividad reservada de la base de datos Requisito de datos de entrada RDE6.4
BD	Requisito de datos de escritura RDW6.4 Requisito de datos de lectura RDR6.4
Salida	Agente externo: Cliente Acción: confirmación de resultado Requisito de datos de salida: ninguno
RDE6.4	Datos de entrada de cancelación de la actividad: - DNI: Cadena de caracteres (9)
RDW6.4	Datos de la reserva de las actividad: - Los mismos que en RDW6.3
RDR6.4	Datos de la reserva de la actividad para comprobar que existe: - Los mismos que RDE6.3

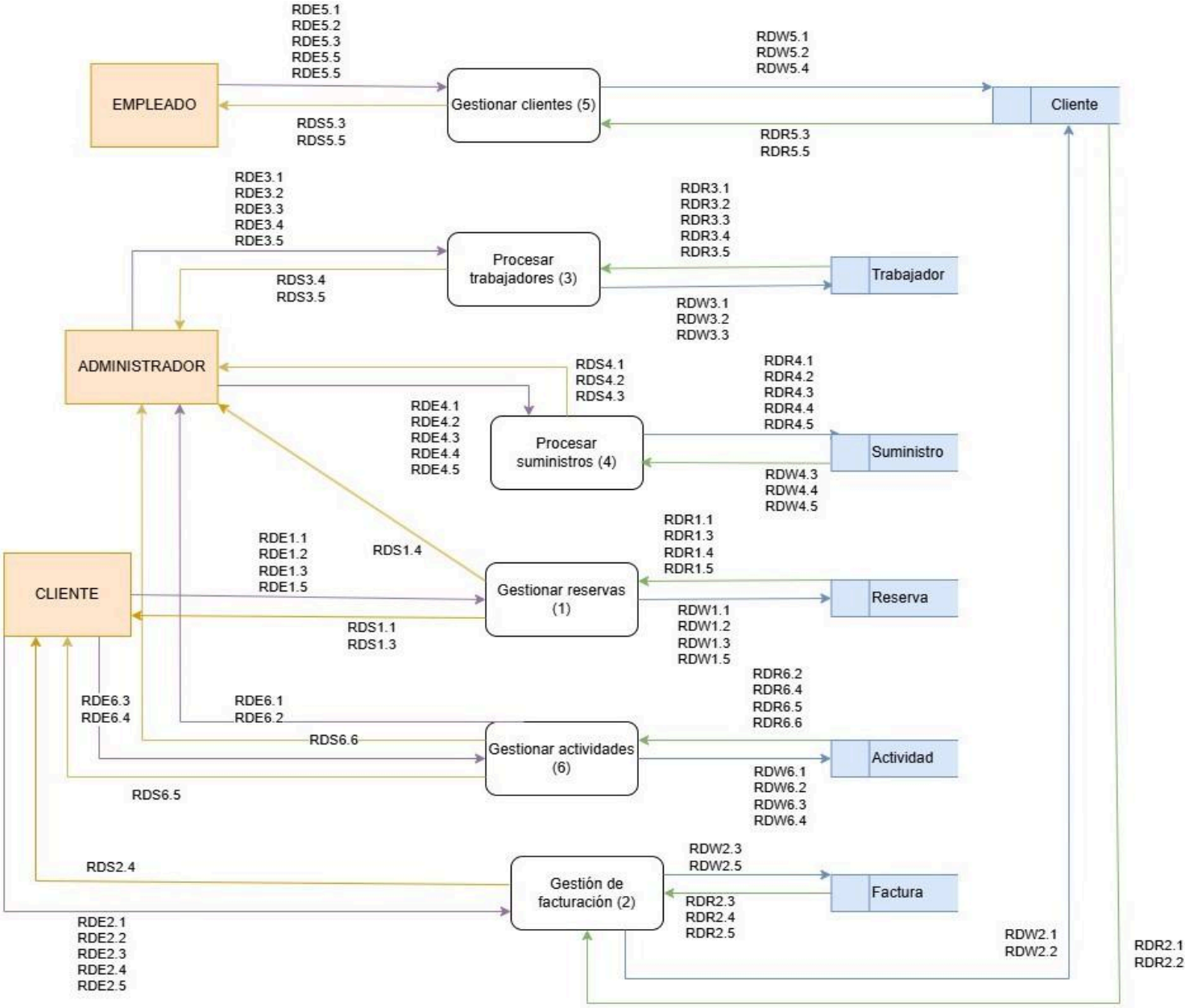
RF 6.5 - MOSTRAR LISTADO DE ACTIVIDADES	
Entrada	Agente externo: Usuario Acción: solicitar listado de actividades Requisito de datos de entrada: ninguno
BD	Requisito de datos de lectura RDR6.5
Salida	Agente externo: Usuario Acción: confirmación de resultado Requisito de datos de salida: RDS6.5
RDR6.5	Datos de actividades almacenadas: - Los mismos que RDE6.1
RDS6.5	Listado de actividades: - Cada una de ellas con los mismos datos de RDR6.5

RF 6.6 - CONSULTAR ACTIVIDADES CON FILTRADO	
Entrada	Agente externo: administrador Acción: solicitar listado filtrado de actividades Requisito de datos de entrada RDE6.6
BD	Requisito de datos de lectura RDR6.6
Salida	Agente externo: administrador Acción: confirmación de resultado Requisito de datos de salida: RDS6.6
RDE6.6	Datos de entrada para filtrar resultados: - Los mismos que en RDE6.1
RDR6.6	Datos de actividades almacenadas: - Los mismos que RDE6.1
RDS6.6	Listado de actividades: - Cada una de ellas con los mismos datos de RDR6.6

3. DFD Esquema Caja Negra

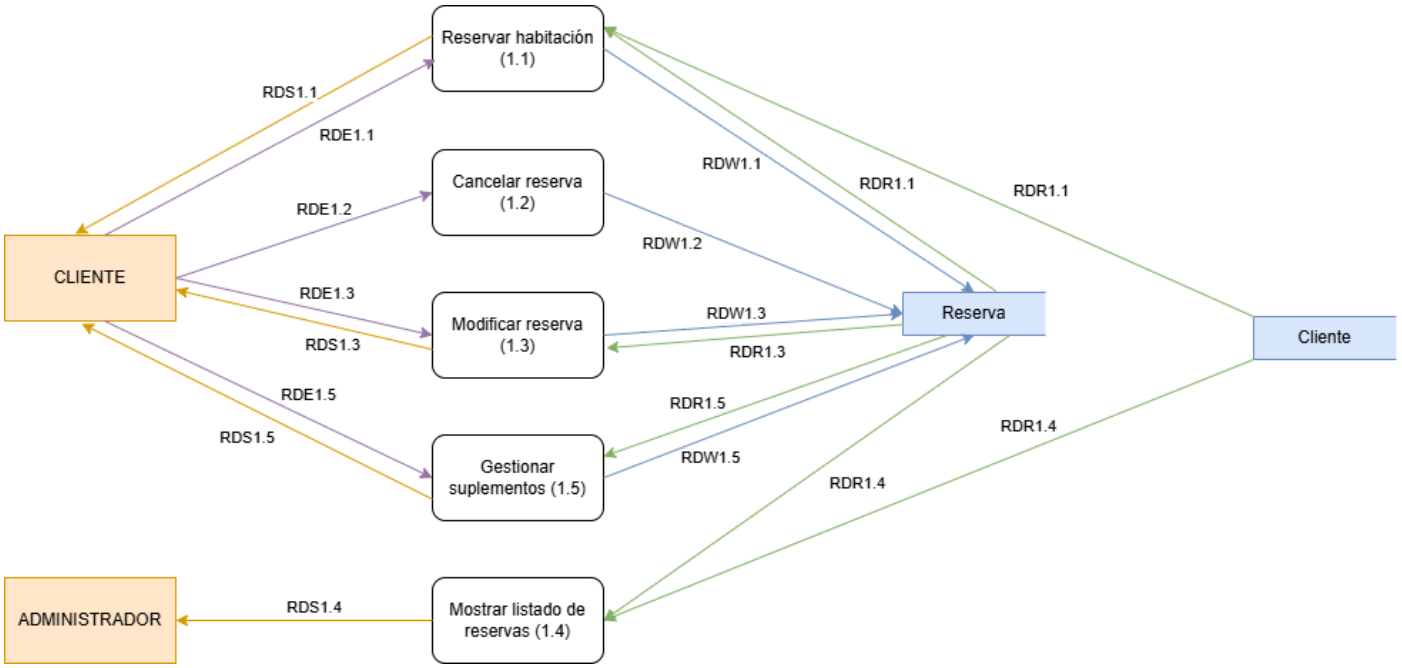


4. DFD Armazón

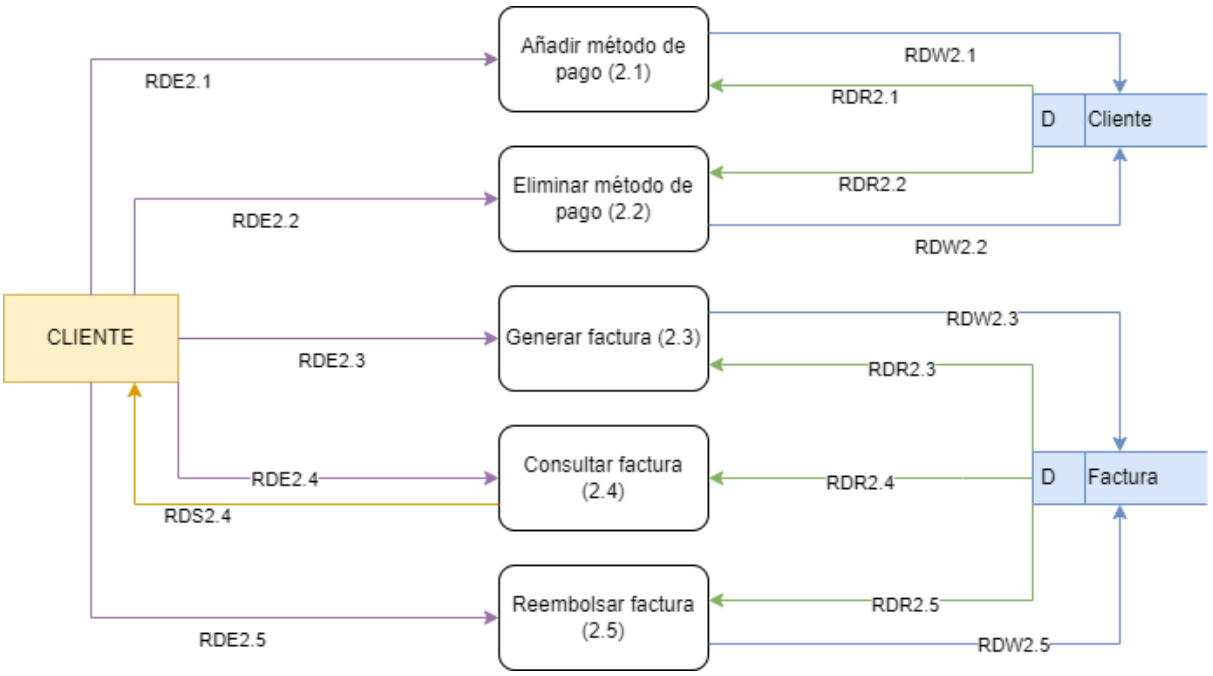


5. DFD de los Subsistemas

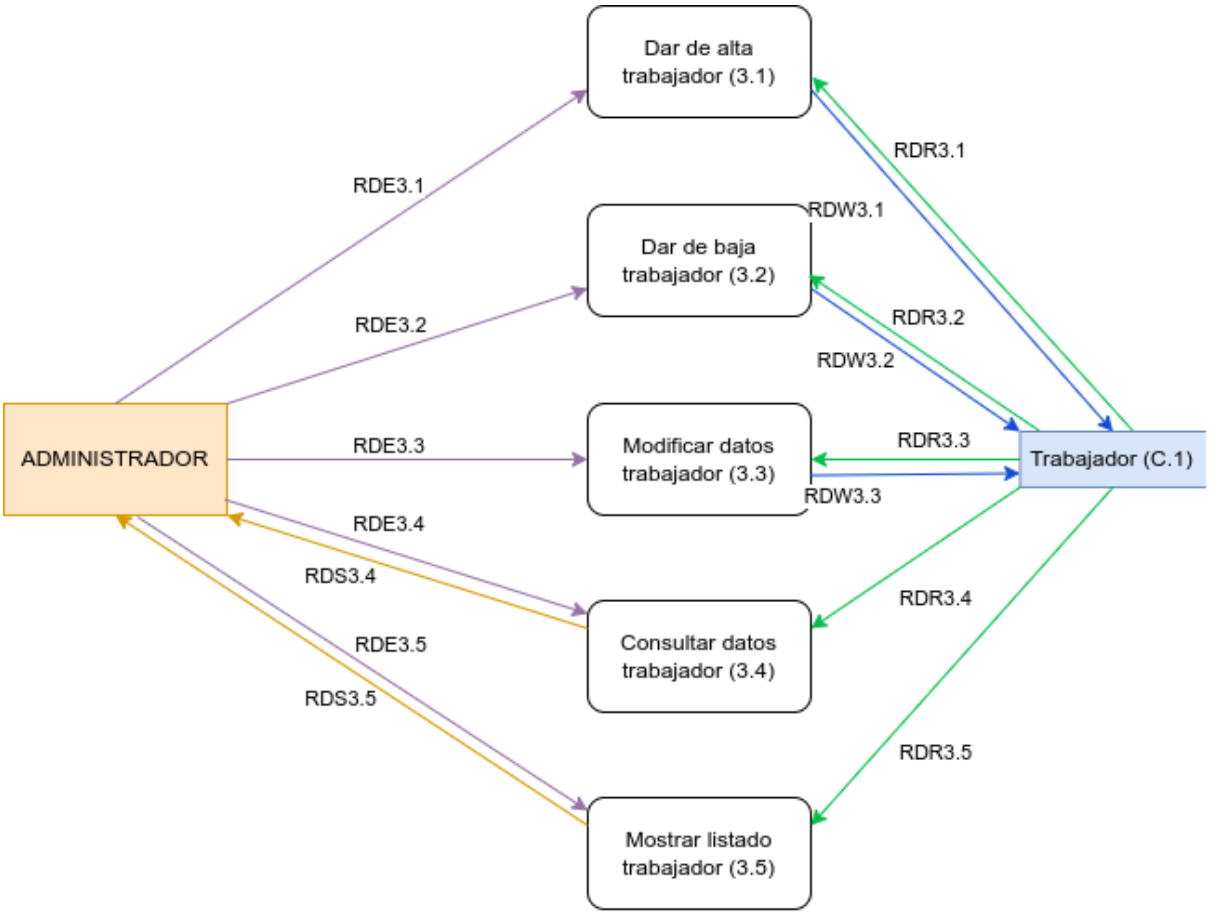
5.1. Gestión de habitaciones



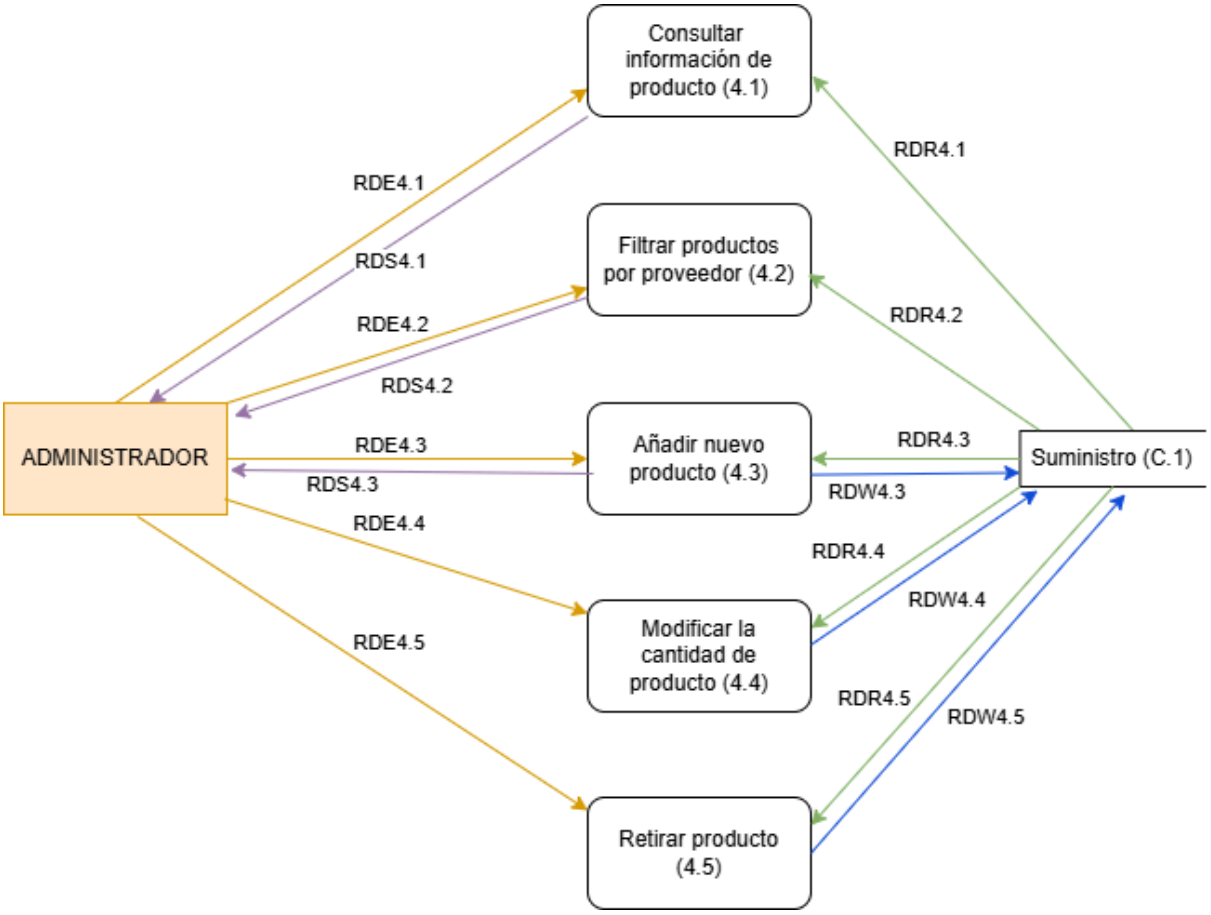
5.2. Facturación y pago



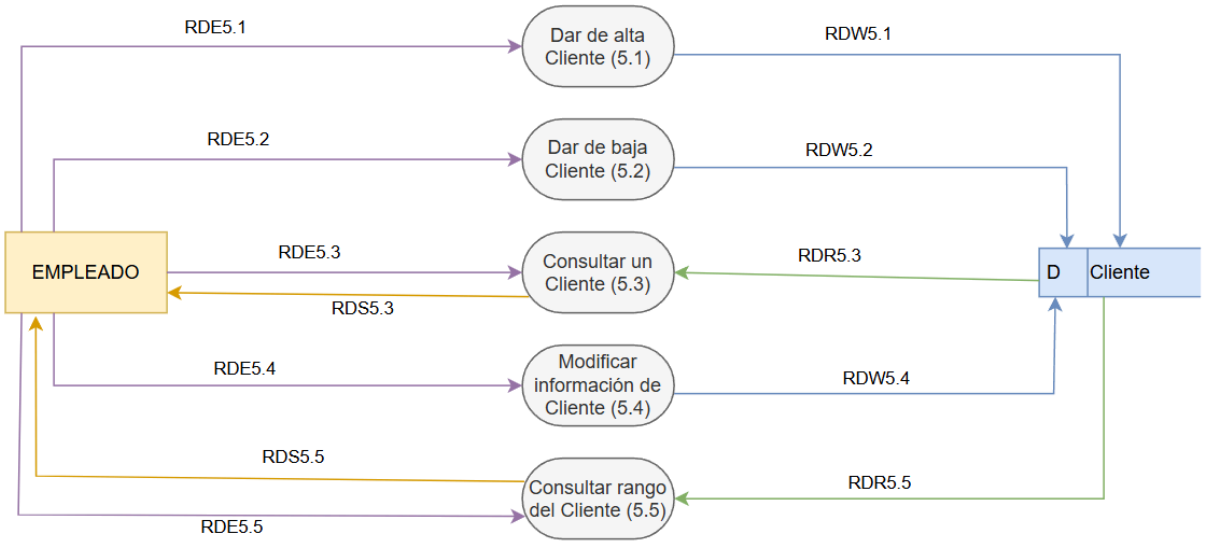
5.3. Trabajadores



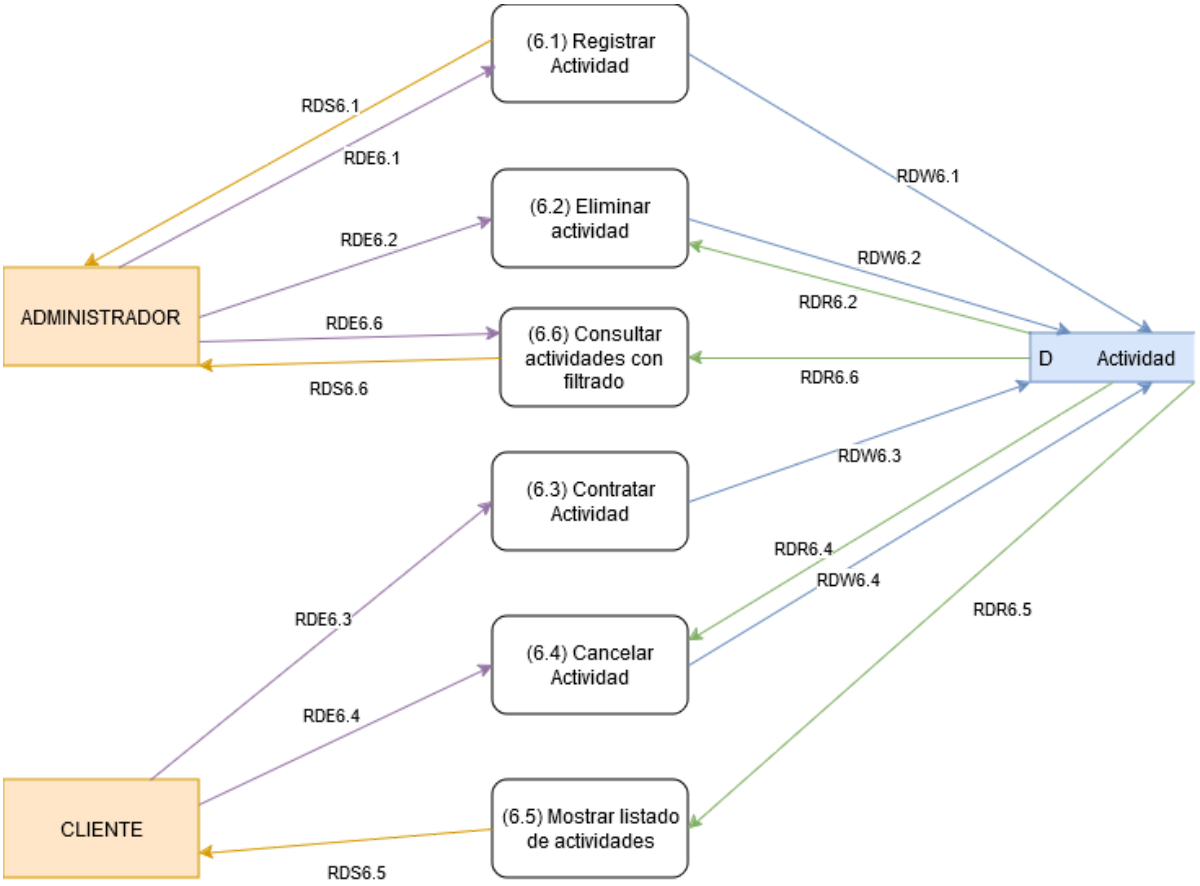
5.4. Suministros



5.5. Gestión de Clientes

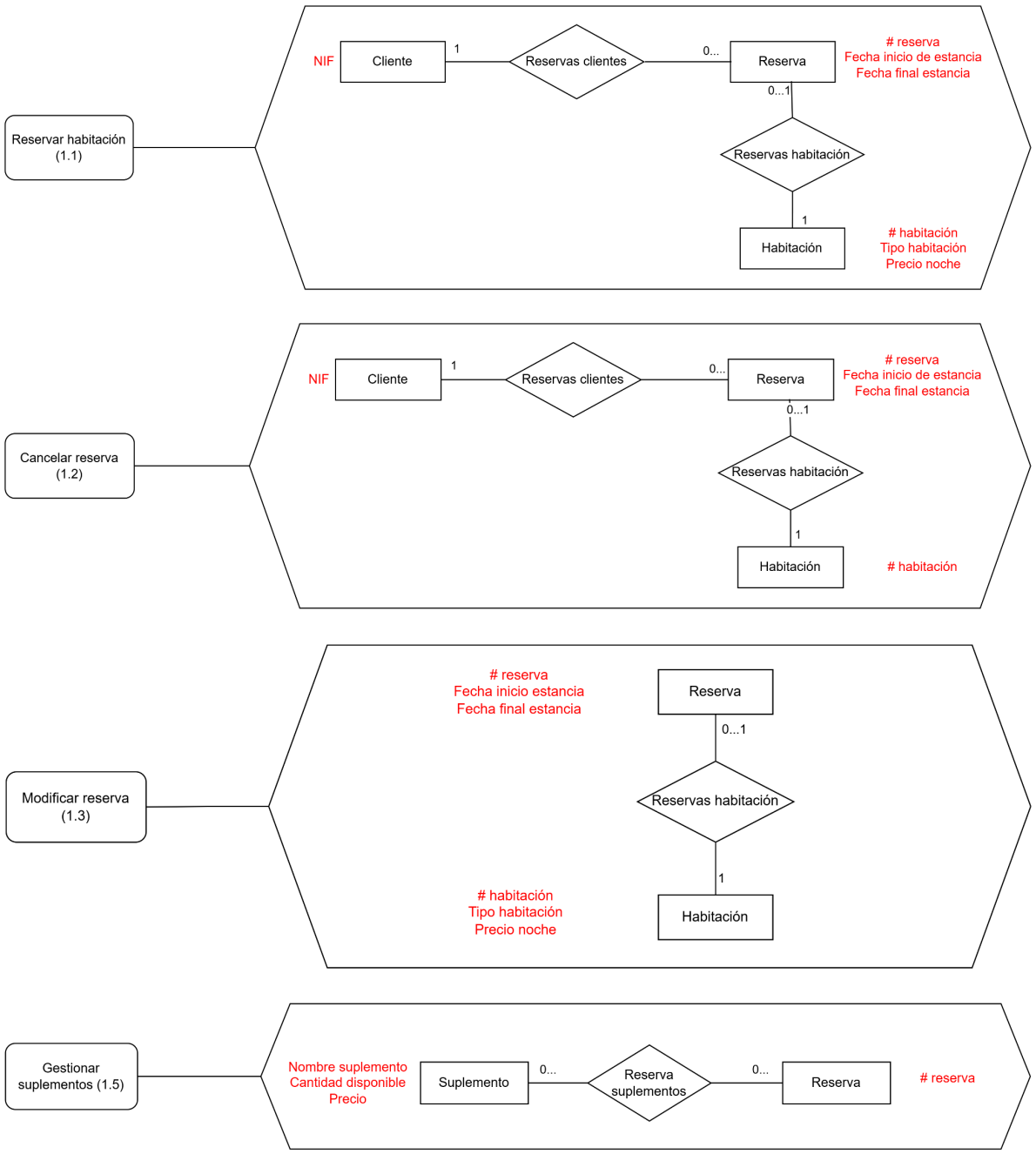


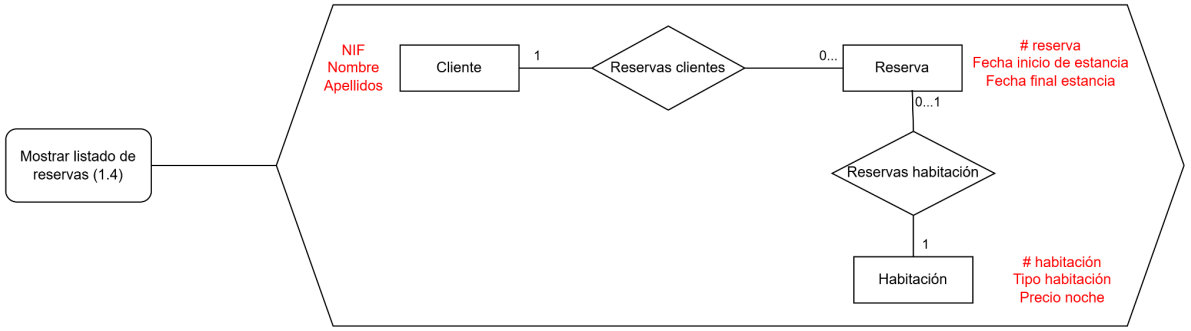
5.6. Servicios

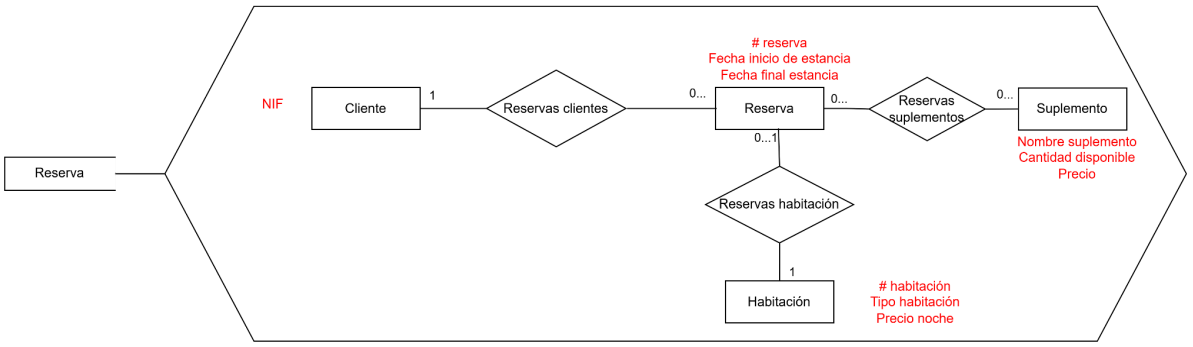


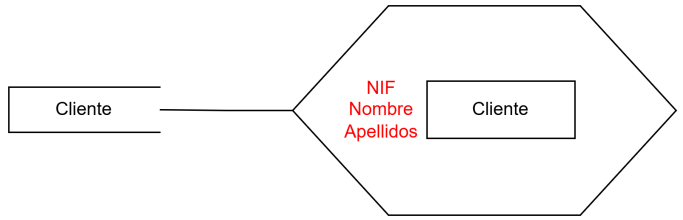
6. Esquemas Externos

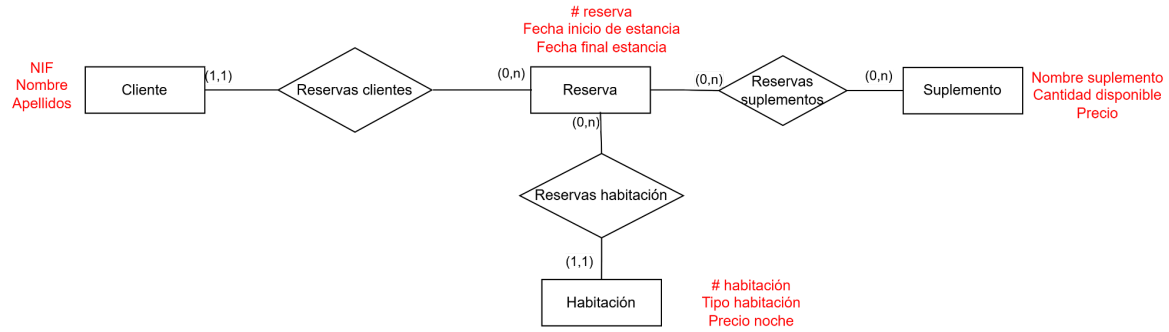
6.1. Gestión de habitaciones



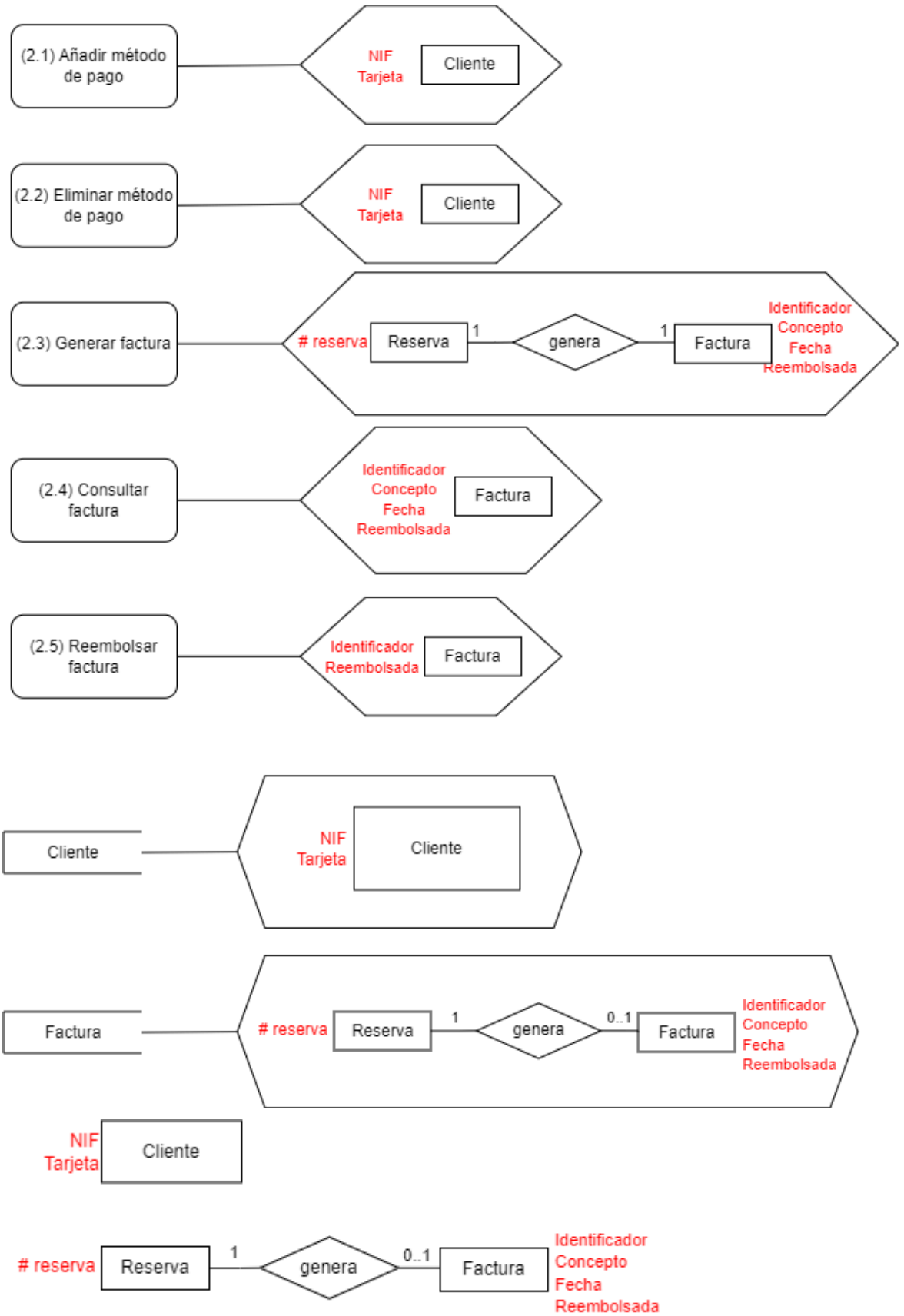

Mostrar listado de reservas (1.4)
NIF
Nombre
Apellidos
Cliente
1
Reservas clientes
0...
Reserva
reserva
Fecha inicio de estancia
Fecha final estancia
0...1
Reservas habitación
1
Habitación
habitación
Tipo habitación
Precio noche


Reserva
NIF
Cliente
1
Reservas clientes
0...
Reserva
reserva
Fecha inicio de estancia
Fecha final estancia
0...
Reservas suplementos
0...
Suplemento
Nombre suplemento
Cantidad disponible
Precio
0...1
Reservas habitación
1
Habitación
habitación
Tipo habitación
Precio noche

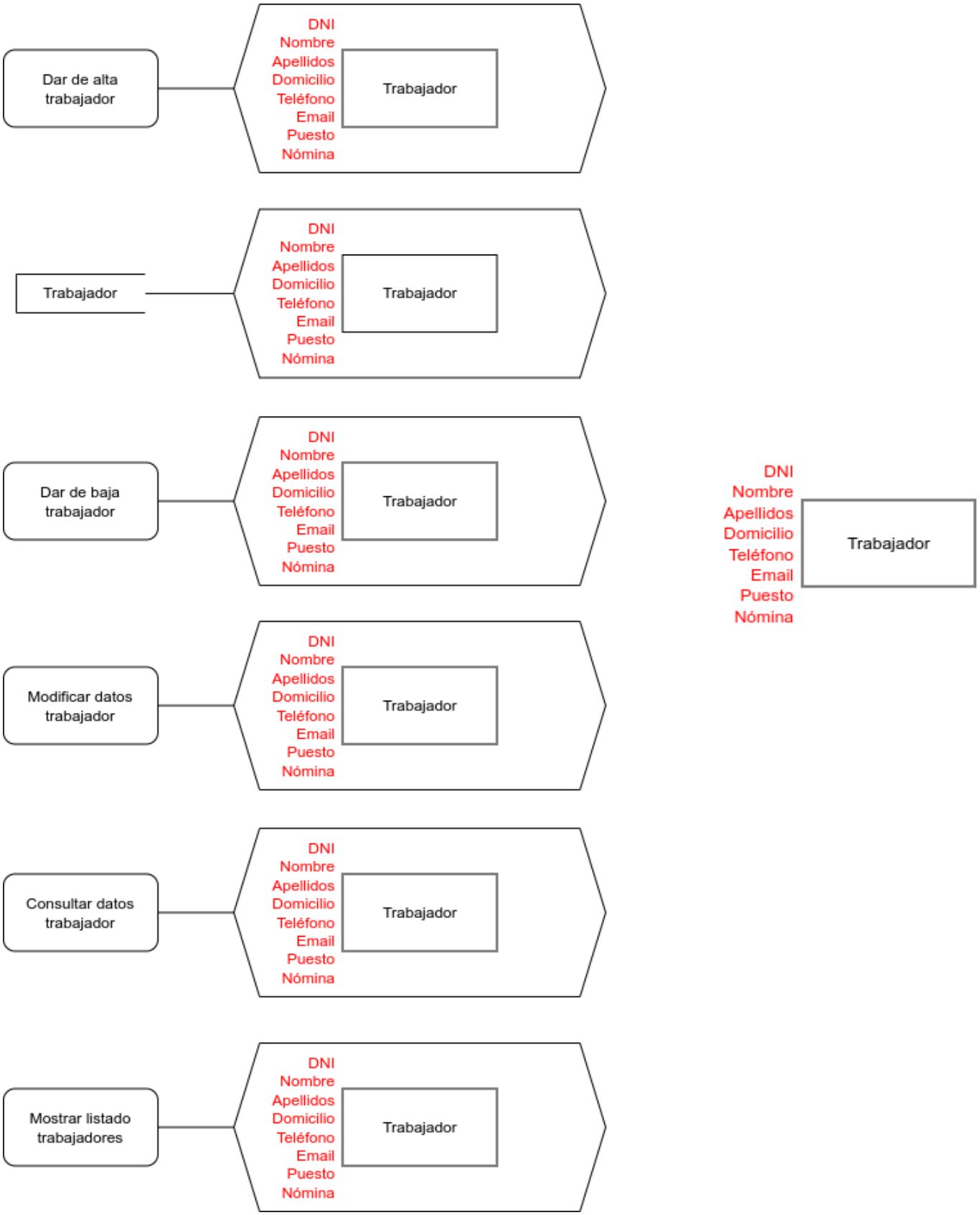

Cliente
NIF
Nombre
Apellidos
Cliente


NIF
Nombre
Apellidos
Cliente
(1,1)
Reservas clientes
(0,n)
Reserva
reserva
Fecha inicio de estancia
Fecha final estancia
(0,n)
Reservas suplementos
(0,n)
Suplemento
Nombre suplemento
Cantidad disponible
Precio
(0,n)
Reservas habitación
(1,1)
Habitación
habitación
Tipo habitación
Precio noche

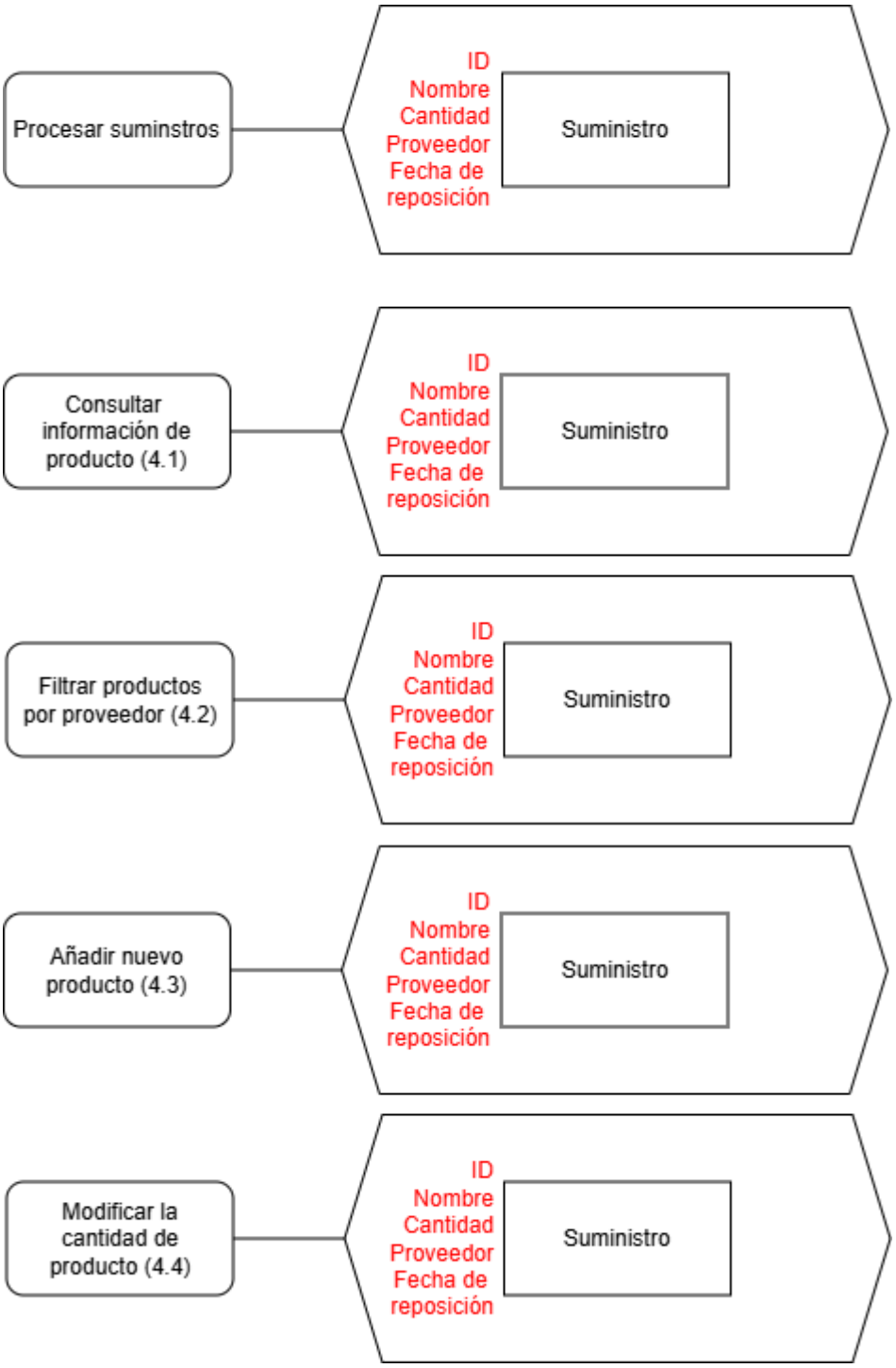
6.2. Facturación y pago

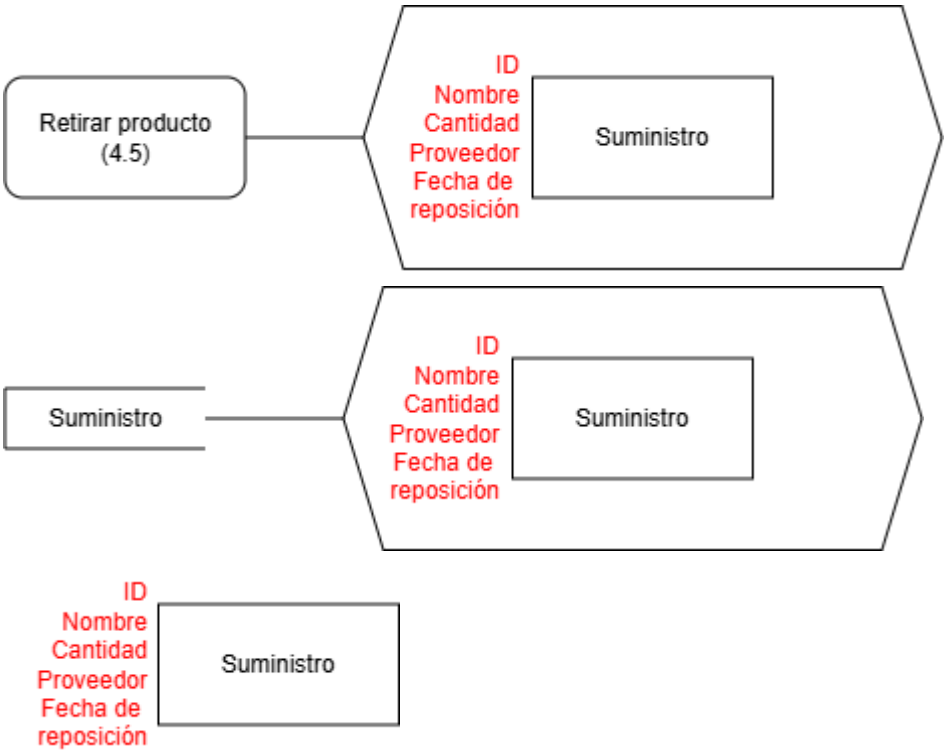


6.3. Trabajadores

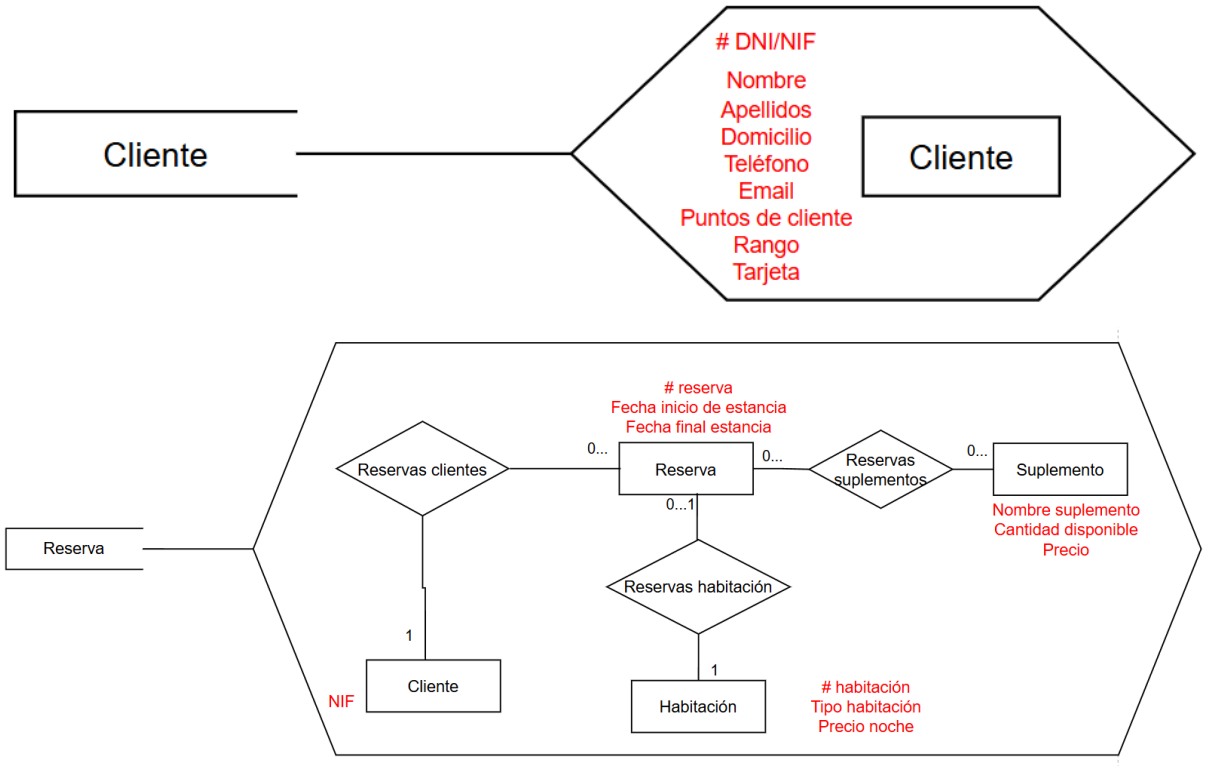


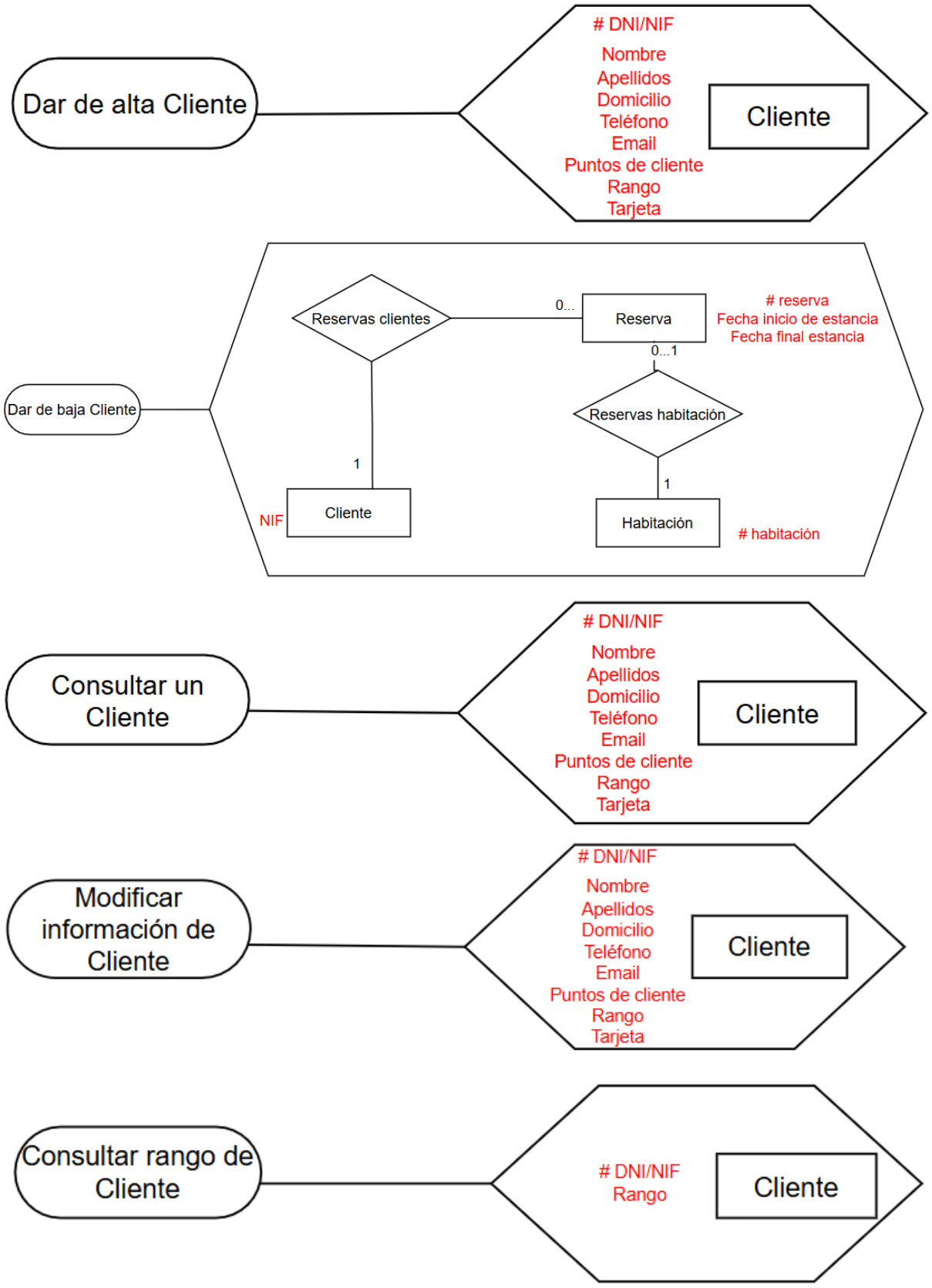
6.4. Suministros

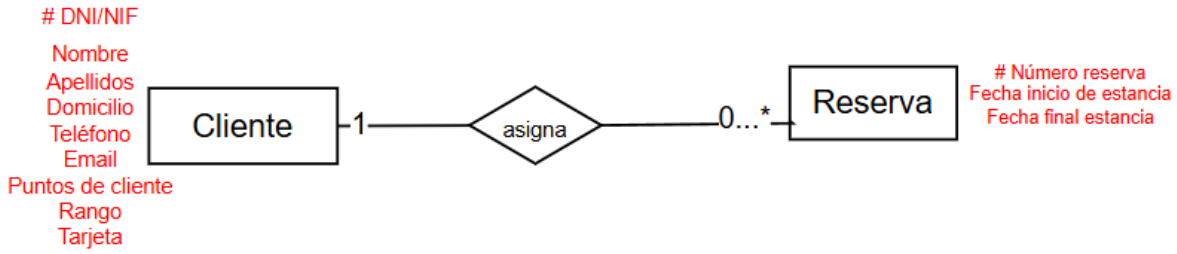




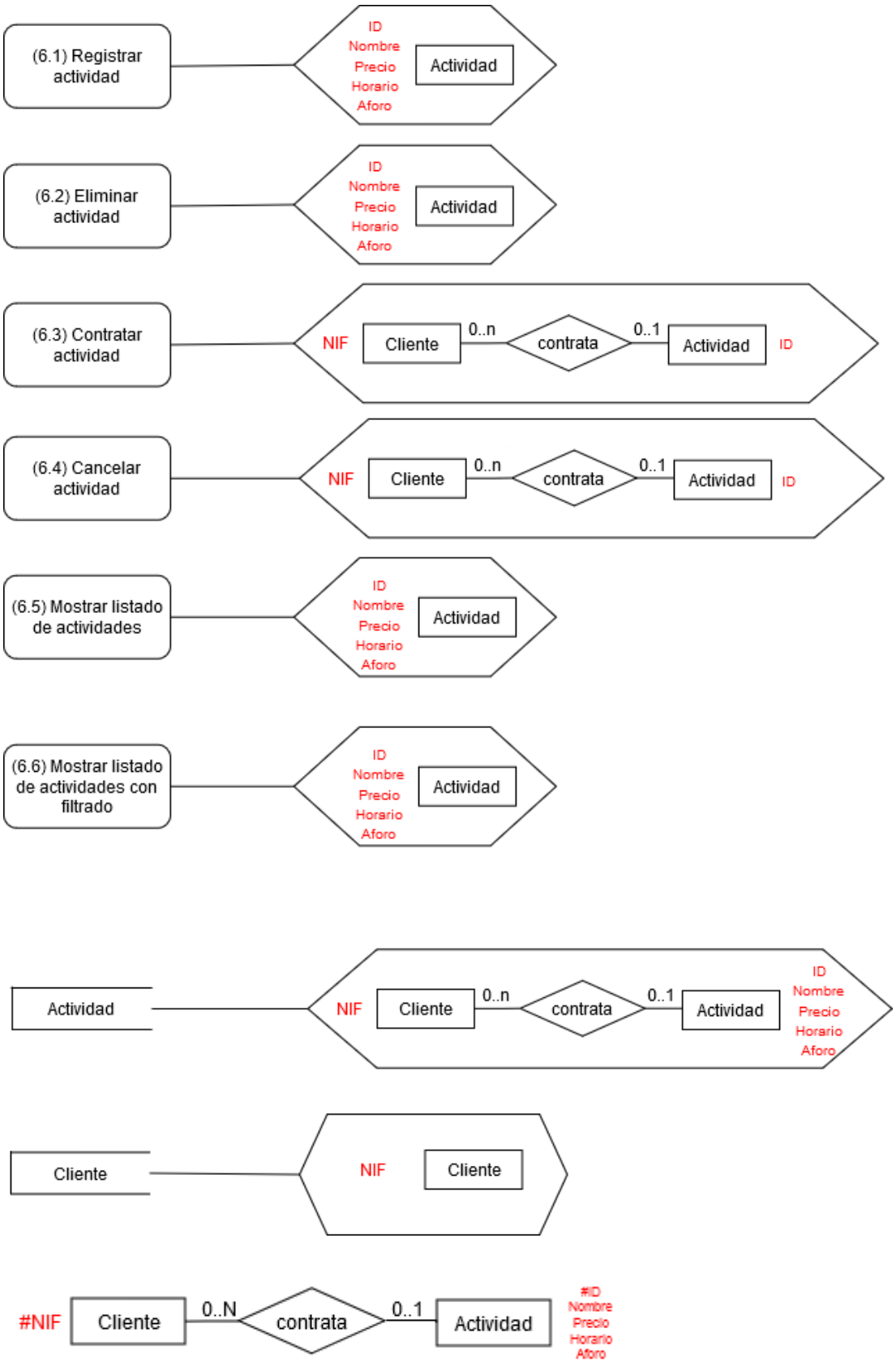
6.5. Gestión de Clientes



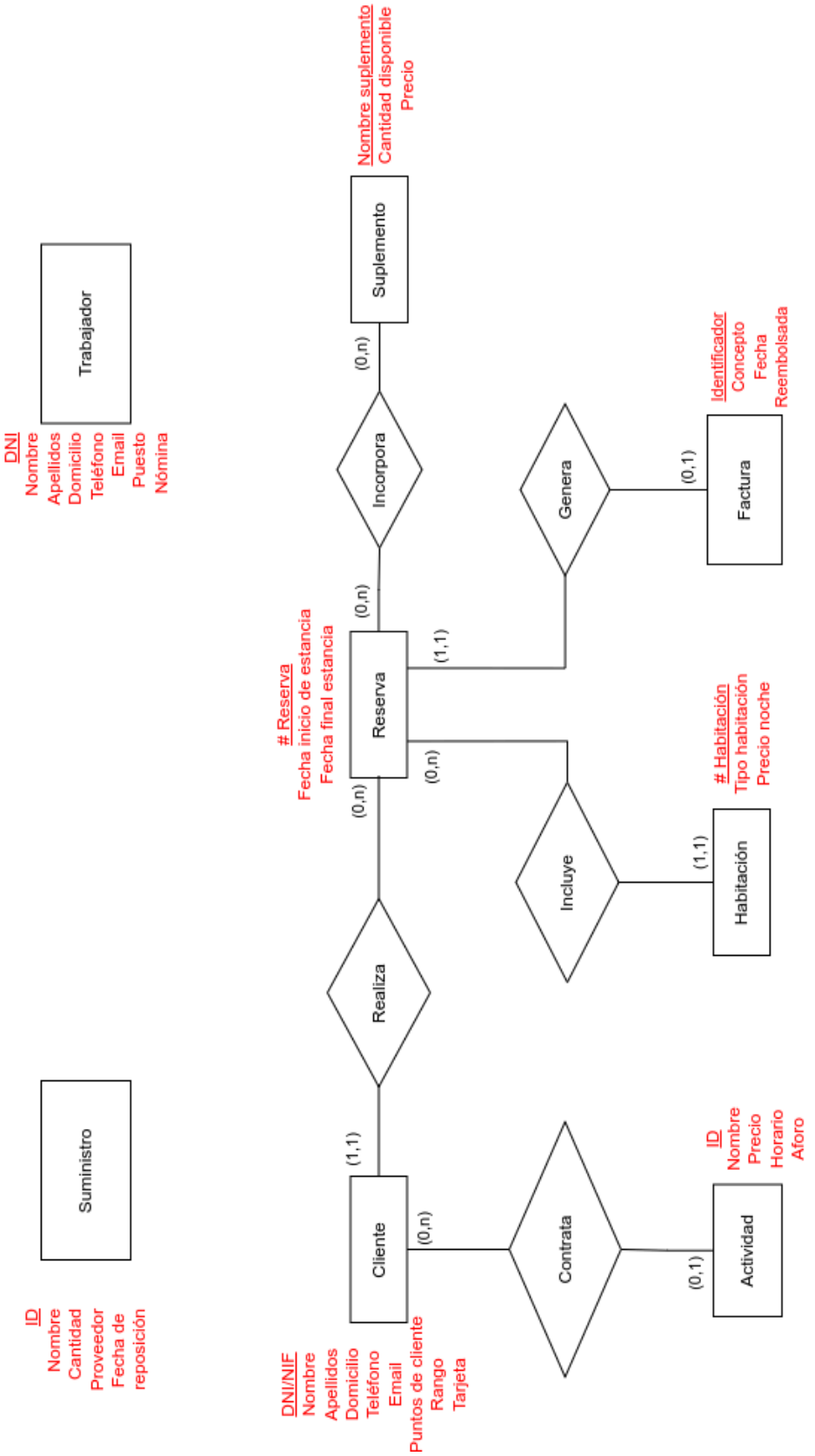

Dar de alta Cliente
DNI/NIF
Nombre
Apellidos
Domicilio
Teléfono
Email
Puntos de cliente
Rango
Tarjeta
Cliente
Dar de baja Cliente
Reservas clientes
0...
Reserva
reserva
Fecha inicio de estancia
Fecha final estancia
0...1
Reservas habitación
1
1
NIF
Cliente
Habitación
habitación
Consultar un Cliente
DNI/NIF
Nombre
Apellidos
Domicilio
Teléfono
Email
Puntos de cliente
Rango
Tarjeta
Cliente
Modificar información de Cliente
DNI/NIF
Nombre
Apellidos
Domicilio
Teléfono
Email
Puntos de cliente
Rango
Tarjeta
Cliente
Consultar rango de Cliente
DNI/NIF
Rango
Cliente


DNI/NIF
Nombre
Apellidos
Domicilio
Teléfono
Email
Puntos de cliente
Rango
Tarjeta
Cliente
1
asigna
0...*
Reserva
Número reserva
Fecha inicio de estancia
Fecha final estancia

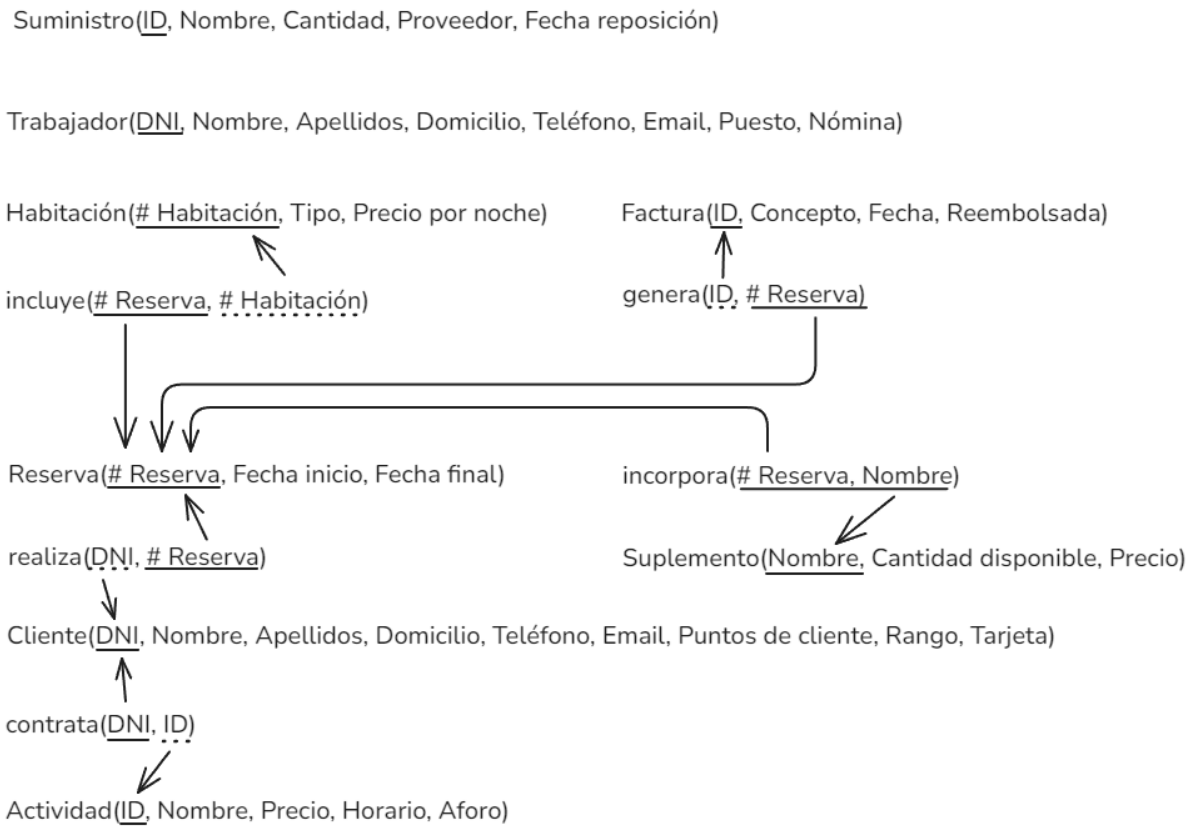
6.6. Servicios

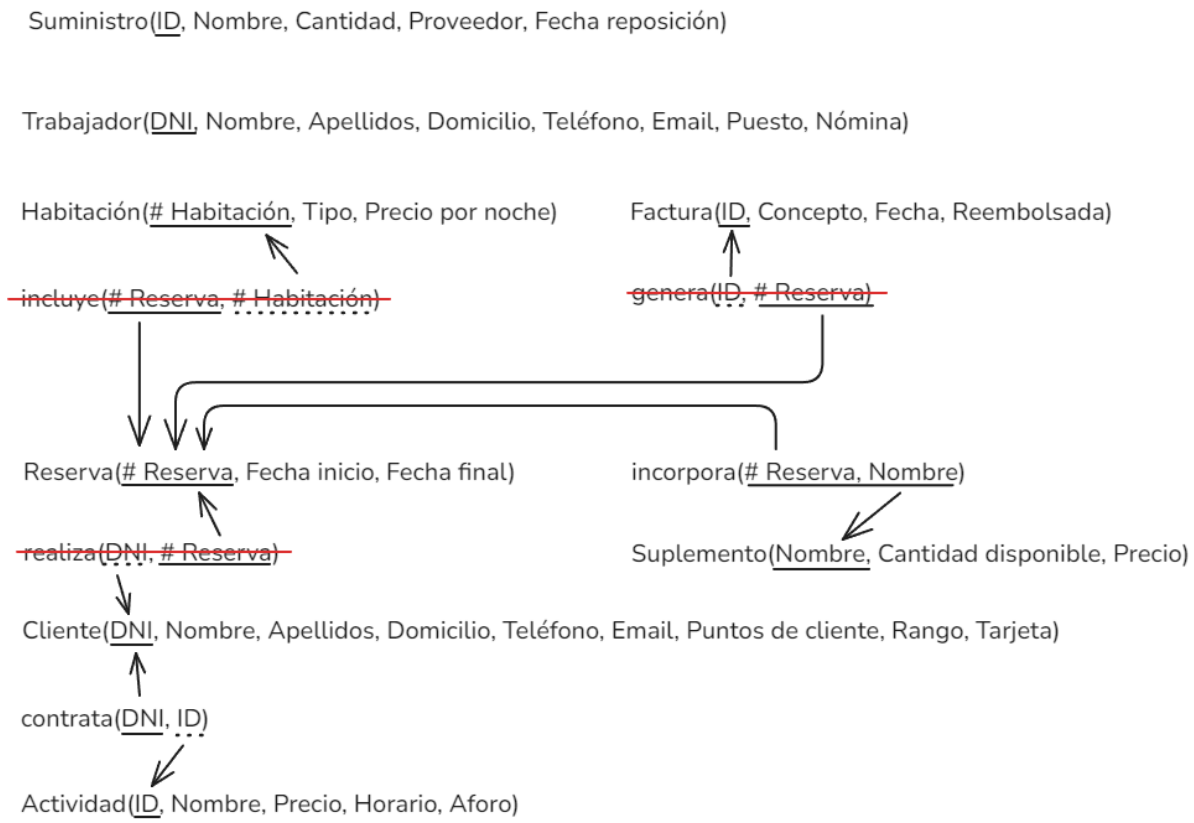


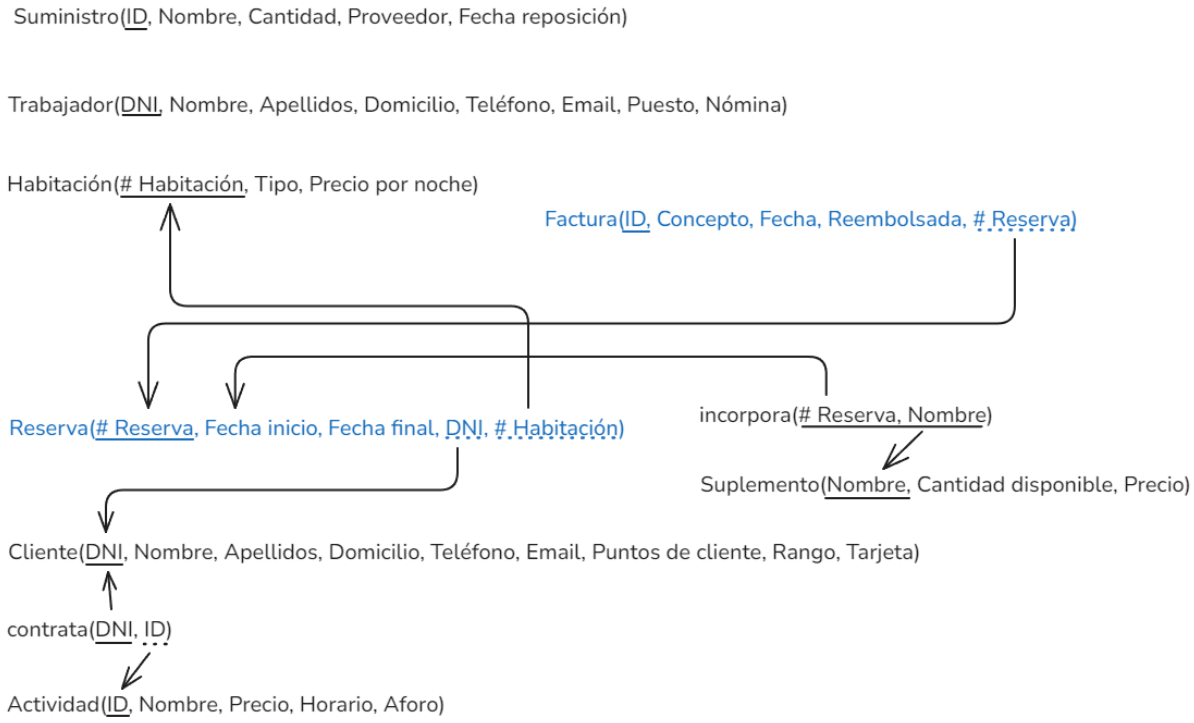
7. Esquema E/R del Sistema



8. Conjunto de Tablas (paso a tablas)




Suministro(ID, Nombre, Cantidad, Proveedor, Fecha reposición)
Trabajador(DNI, Nombre, Apellidos, Domicilio, Teléfono, Email, Puesto, Nómina)
Habitación(# Habitación, Tipo, Precio por noche)
Factura(ID, Concepto, Fecha, Reembolsada)
incluye(# Reserva, # Habitación)
genera(ID, # Reserva)
Reserva(# Reserva, Fecha inicio, Fecha final)
incorpora(# Reserva, Nombre)
realiza(DNI, # Reserva)
Suplemento(Nombre, Cantidad disponible, Precio)
Cliente(DNI, Nombre, Apellidos, Domicilio, Teléfono, Email, Puntos de cliente, Rango, Tarjeta)
contrata(DNI, ID)
Actividad(ID, Nombre, Precio, Horario, Aforo)


Suministro(ID, Nombre, Cantidad, Proveedor, Fecha reposición)
Trabajador(DNI, Nombre, Apellidos, Domicilio, Teléfono, Email, Puesto, Nómina)
Habitación(# Habitación, Tipo, Precio por noche)
Factura(ID, Concepto, Fecha, Reembolsada, # Reserva)
Reserva(# Reserva, Fecha inicio, Fecha final, DNI, # Habitación)
incorpora(# Reserva, Nombre)
Suplemento(Nombre, Cantidad disponible, Precio)
Cliente(DNI, Nombre, Apellidos, Domicilio, Teléfono, Email, Puntos de cliente, Rango, Tarjeta)
contrata(DNI, ID)
Actividad(ID, Nombre, Precio, Horario, Aforo)

9. Dependencias funcionales y normalización

Todas las tablas están en forma normal de Boyce y Codd, por lo que no hay que hacer el proceso de normalización. Las dependencias funcionales son las siguientes:

Cliente:

```
Unset
DNI → Nombre, Apellidos, Domicilio, Teléfono, Email, Puntos de cliente, Rango, Tarjeta
```

Actividad:

```
Unset
ID → Nombre, Precio, Horario, Aforo
```

contrata:

```
Unset
(DNI, ID) → (DNI, ID)
```

Habitación:

```
Unset
#Habitación → Tipo, Precio por noche
```

Suplemento:

```
Unset
Nombre → Cantidad disponible, Precio
```

incorpora:

```
Unset
(#Reserva, Nombre) → (#Reserva, Nombre)
```

Suministro:

```
Unset
ID → Nombre, Cantidad, Proveedor, Fecha reposición
```

Trabajador:

```
Unset
DNI → Nombre, Apellidos, Domicilio, Teléfono, Email, Puesto, Nómina
```

Reserva:

```
Unset
#Reserva → Fecha inicio, Fecha final, DNI, #Habitación
```

Factura:

```
Unset
ID → Concepto, Fecha, Reembolsada, Reserva
```

10. Implementación

Selección e instalación del software

El lenguaje utilizado en esta práctica ha sido Java. Con este lenguaje hemos implementado la conexión a la base de datos así como el propio trabajo que se pide en el seminario. Como API hemos usado JDBC para la implementación del código.

Como hemos elegido Java para este proyecto, hemos utilizado Maven como gestor de dependencias. Para instalar Maven es necesario instalar Java previamente. Para ello hemos descargado la versión de Java 21 JDK de https://adoptium.net/es/temurin/releases/.

Una vez instalado Java hemos instalado Maven. Para ello hemos seguido las instrucciones oficiales: https://maven.apache.org/install.html

Podemos comprobar que ambos están instalados con los siguientes comandos:

```
Unset
java -version
mvn -version
```

Una vez instalados, hemos querido usar VisualStudio Code para gestionar el proyecto, por lo que hemos instalado las extensiones recomendadas por Microsoft para trabajar con proyectos de Java:

- https://marketplace.visualstudio.com/items?itemName=vscjava.vscode-java-pack

Con esto, ya podíamos crear un proyecto de Maven desde Visual Studio Code y agregar las dependencias necesarias. La primera dependencia es la de Oracle JDBC, que se encuentra en los repositorios de Maven y se puede instalar agregando al fichero pom.xml las siguientes líneas:

```
Unset
<dependency>
            <groupId>com.oracle.database.jdbc</groupId>
            <artifactId>ojdbc11</artifactId>
            <version>23.5.0.24.07</version>
</dependency>
```

Otra dependencia que hemos utilizado es dotenv-java. Esta permite el uso de ficheros .env para establecer variables que no queremos incluir en el código fuente de la aplicación, como

puede ser el usuario y la contraseña de la base de datos. Para incluir la dependencia, de igual forma que para la anterior podemos añadir lo siguiente al fichero pom.xml:

```
Unset
<dependency>
            <groupId>io.github.cdimascio</groupId>
            <artifactId>dotenv-java</artifactId>
            <version>3.0.2</version>
</dependency>
```

Para ejecutar la aplicación basta con escribir el siguiente comando dentro de la carpeta del proyecto, que descargará e incluirá las dependencias necesarias:

```
Unset
mvn exec:java
```

Para saber cómo usar la API de Java para usar SQL, hemos consultado el siguiente enlace:

- https://docs.oracle.com/en/java/javase/11/docs/api/java.sql/java/sql/package-summary.html

Por otro lado, nos hemos ayudado de Chat GPT para generar parte del código en Java:

- https://chatgpt.com

Creación de tablas

El sistema está dividido en varios subsistemas, cada uno asignado a un integrante del equipo. Cada subsistema está desarrollado como una clase independiente que implementa sus funcionalidades específicas aparte de las generales como: crearTablas, mostrarTablas y bucleInteractivo. Además, existe una clase principal `Main` que orquesta la ejecución del programa y maneja el menú principal.

En la clase Main tenemos la función borrarYCrearTablas encargada de llamar a los métodos implementados en cada clase:

```
public static void borrarYCrearTablas(Connection conn) {
    GestionHotel.borrarTablas(conn);
    GestionClientes.crearTablas(conn);
    GestionReservas.crearTablas(conn);
    Trabajadores.crearTablas(conn);
    Facturacion.crearTablas(conn);
    Servicios.crearTablas(conn);
    GestionSuministros.crearTablas(conn);
```

```
    System.out.println("Tablas borradas y creadas con éxito");
}
```

A continuación, se expone la implementación del método *crearTablas* de cada subsistema.

Subsistema de reservas

El método `crearTablas` define las tablas necesarias para el subsistema de gestión de reservas en la base de datos del hotel. Este subsistema se encarga de almacenar información sobre habitaciones, reservas, suplementos asociados a las reservas y sus relaciones.

```
public static void crearTablas(Connection conn) {
        try {
            Statement stmt = conn.createStatement();
            // Crear la tabla Habitacion
            stmt.executeUpdate("CREATE TABLE Habitacion ("
                    + "id NUMBER GENERATED BY DEFAULT AS IDENTITY,"
                    + "tipo VARCHAR2(10) CHECK (tipo IN ('individual', 'doble',
'suite')) NOT NULL,"
                    + "precioPorNoche NUMBER(10,2) NOT NULL,"
                    + "PRIMARY KEY (id)"
                    + ")");

            // Crear la tabla Suplemento
            stmt.executeUpdate("CREATE TABLE Suplemento ("
                    + "nombre VARCHAR2(100) NOT NULL,"
                    + "cantidadDisponible NUMBER NOT NULL,"
                    + "precio NUMBER(10,2) NOT NULL,"
                    + "PRIMARY KEY (nombre)"
                    + ")");

            // Crear la tabla Reserva

            stmt.executeUpdate("CREATE TABLE Reserva ("
                    + "id NUMBER GENERATED BY DEFAULT AS IDENTITY,"
                    + "fechaInicio DATE NOT NULL,"
                    + "fechaFinal DATE NOT NULL,"
                    + "dni VARCHAR2(9),"
                    + "habitacion NUMBER NOT NULL,"
                    + "PRIMARY KEY (id),"
                    + "FOREIGN KEY (dni) REFERENCES Cliente(dni),"
                    + "FOREIGN KEY (habitacion) REFERENCES Habitacion(id))");

            // Crear la tabla Incorpora
            stmt.executeUpdate("CREATE TABLE Incorpora ("
                    + "idReserva NUMBER NOT NULL,"
                    + "nombreSuplemento VARCHAR(100),"
                    + "PRIMARY KEY (idReserva, nombreSuplemento),"
                    + "FOREIGN KEY (idReserva) REFERENCES Reserva(id),"
                    + "FOREIGN KEY (nombreSuplemento) REFERENCES Suplemento(nombre)"
                    + ")");
```

```
            stmt.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
       // Aquí hay código de insertar filas en las tablas y crear el disparador que se
explicará más adelante
}
```

Las tablas creadas son:

- Habitación: almacena información sobre las habitaciones disponibles en el hotel. Las columnas son:
    - id: identificador único de la habitación.
        - Tipo: NUMBER
        - Restricción: generado automáticamente con GENERATED BY DEFAULT AS IDENTITY
        - Es clave primaria
    - tipo: tipo de habitación (individual, doble, suite).
        - Tipo: VARCHAR2(10)
        - Restricción: CHECK para asegurar que solo tome valores específicos (‘individual’, ‘doble’, ‘suite’).
        - No nulo.
    - precioPorNoche: precio por noche de la habitación.
        - Tipo: NUMBER(10,2)
        - No nulo
- Suplemento: contiene información sobre los suplementos disponibles para las reservas.
    - nombre: nombre del suplemento
        - Tipo: VARCHAR2(100)
        - No nulo
        - Clave primaria
    - cantidadDisponible: cantidad disponible del suplemento.
        - Tipo: NUMBER
        - No nulo
    - precio: precio del suplemento.
        - Tipo: NUMBER(10,2)
        - No nulo
- Reserva: registra las reservas realizadas por los clientes.
    - id: identificador único de la reserva.
        - Tipo: NUMBER
        - Restricción: generado automáticamente con GENERATED BY DEFAULT AS IDENTITY
        - Clave primaria
    - fechaInicio y fechaFinal: fechas de inicio y final de la estancia.
        - Tipo: DATE
        - No nulo
    - dni: DNI del cliente que realiza la reserva.
        - Tipo: VARCHAR2(9)

        - Restricción: clave foránea que referencia a Cliente(dni)
    - habitación: identificador de la habitación reservada.
        - Tipo:NUMBER
        - Restricción: clave foránea que referencia Habitación(id)
- Incorpora: define los suplementos asociados a una reserva específica.
    - idReserva: identificador de la reserva.
        - Tipo: NUMBER
        - Restricción: clave foránea que referencia Reserva(id)
        - Parte de la clave primaria.
    - nombreSuplemento: nombre del suplemento asociado.
        - Tipo: VARCHAR(100)
        - Restricción: clave foránea que referencia Suplemento(nombre)
        - Parte de la clave primaria.

El orden de creación de tablas asegura que las tablas de habitaciones y suplementos deben existir antes de crearse las de reservas e incorpora, que dependen de ellas mediante las claves foráneas.

Subsistema de facturación

El método `crearTablas` define las tablas necesarias para el subsistema de facturación en la base de datos del hotel. Este método crea la tabla Factura:

```
public static void crearTablas(Connection conn) {
        // Creación de la tabla Factura
        try {
            Statement stmt = conn.createStatement();
            GestionHotel.borrarTabla(conn, "Factura");
            stmt.executeUpdate("CREATE TABLE Factura ("
                    + "id NUMBER GENERATED BY DEFAULT AS IDENTITY,"
                    + "concepto VARCHAR2(255) NOT NULL,"
                    + "fecha DATE NOT NULL,"
                    + "reembolsada NUMBER(1) DEFAULT 0,"
                    + "codReserva NUMBER NOT NULL,"
                    + "PRIMARY KEY (id),"
                    // FIX: Asegurar que primero se crea la tabla de Reserva
                    + "FOREIGN KEY (codReserva) REFERENCES Reserva(id)"
                    + ")");
            stmt.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
}
```

La tabla Factura almacena la información relativa a las facturas del hotel. La tabla almacena los siguientes datos:

- id: identificador único de la factura
    - Tipo: NUMBER
    - Restricción: generado automáticamente
    - Es clave primaria
- concepto: el concepto asociado a la factura
    - Tipo: VARCHAR2 con un máximo de 255 caracteres

    - Restricción: no puede estar vacío
- fecha: la fecha en la que se ha generado la factura
    - Tipo: DATE
    - Restricción: no puede estar vacío
- reembolsada: indica si la factura ha sido reembolsada
    - Tipo: NUMBER
    - Actúa como un booleano, por defecto es 0 (False)
- codReserva: identificador de la reserva
    - Tipo: NUMBER
    - Restricción: clave externa que referencia a Reserva(id)

Subsistema de trabajadores

El método `crearTablas` define las tablas necesarias para el subsistema de gestión de trabajadores en la base de datos del hotel. En este método creo una única tabla, Trabajador, con 2 trabajadores de ejemplo, así como el disparador del subsistema.
Los datos de una tabla Trabajador son:

- dni: DNI del trabajador.
    - Tipo: CHAR(9)
    - No nulo
    - **Es Clave Primaria**
- nombre: nombre del trabajador.
    - Tipo: VARCHAR(20)
    - No nulo
- apellidos: apellidos del trabajador.
    - Tipo: VARCHAR(50)
    - No nulo
- domicilio: domicilio del trabajador.
    - Tipo: VARCHAR(50)
    - No nulo
- teléfono: teléfono de contacto del trabajador.
    - Tipo: VARCHAR(20)
    - No nulo
- email: correo electrónico del trabajador.
    - Tipo: VARCHAR(50)
    - No nulo
- puesto: puesto de trabajo,
    - Tipo: VARCHAR(20)
    - No nulo
    - Debe ser uno de los siguientes: ADMINISTRADOR, RECEPCIONISTA, LIMPIADOR.
- nomina: sueldo del trabajador.
    - Tipo: DECIMAL(10,2)
    - No nulo
- fecha_contratación: fecha en la que se contrató al trabajador.
    - Tipo: DATE
    - Valor por defecto: SYSDATE (fecha del sistema)

- fecha_ultimo_aumento: fecha en la que se modificó por última vez la nómina del trabajador.
    - Tipo: DATE
    - Valor por defecto: SYSDATE (fecha del sistema)

A continuación, se incluye el código de `crearTablas:`

```
public static void crearTablas(Connection conn) {

        // Creamos la tabla Trabajador
        try {
            Statement stmt = conn.createStatement();
            GestionHotel.borrarTabla(conn, "Trabajador");
            stmt.executeUpdate("CREATE TABLE Trabajador ("
                + "dni CHAR(9) NOT NULL,"
                + "nombre VARCHAR(20) NOT NULL,"
                + "apellidos VARCHAR(50) NOT NULL,"
                + "domicilio VARCHAR(50) NOT NULL,"
                + "telefono VARCHAR(20) NOT NULL,"
                + "email VARCHAR(50) NOT NULL,"
                + "puesto VARCHAR(20) NOT NULL CHECK (puesto IN ('ADMINISTRADOR',
'RECEPCIONISTA', 'LIMPIADOR')),"
                + "nomina DECIMAL(10, 2) NOT NULL,"
                + "fecha_contratacion DATE DEFAULT SYSDATE,"
                + "fecha_ultimo_aumento DATE DEFAULT SYSDATE,"
                + "PRIMARY KEY (dni)"
                + ")");
            conn.commit(); // Hacer commit después de crear la tabla
        } catch (SQLException e) {
            System.out.println("Error al crear la tabla Trabajador: " + e.getMessage());
            try {
                conn.rollback(); // Hacer rollback en caso de error
            } catch (SQLException ex) {
                System.out.println("Error al tratar de hacer rollback: " +
ex.getMessage());
            }
        }
        // Aquí hay código insertar los trabajadores de ejemplo y crear el disparador, Se
explicará después.
    }
```

Subsistema de clientes

El método `crearTablas` define las tablas necesarias para el subsistema de gestión de clientes en la base de datos del hotel. Este subsistema se encarga de almacenar información sobre los clientes del hotel, almacenando entre otros datos su nombre, DNI, puntos de cliente, etc. En este método creo una única tabla Cliente la cual almacena información sobre los clientes del hotel. Los datos a almacenar son:

- nombre: nombre del cliente.
    - Tipo: VARCHAR(20)
- apellidos: apellidos del cliente.

    - Tipo: VARCHAR(40)
- teléfono: precio por noche de la habitación.
    - Tipo: VARCHAR(40)
- dni: identificador único del cliente (Clave primaria)
    - Tipo: VARCHAR(9)
    - Es clave primaria.
- domicilio:
    - Tipo: VARCHAR(60)
- email:
    - Tipo: VARCHAR(30)
    - CONSTRAINT email_clave_candidata UNIQUE
    - No nulo.
- puntos:
    - Tipo: INTEGER
- rango: rango del cliente
    - Tipo: VARCHAR(20)
    - Restringido por trigger: debe ser ('Inicial', 'Avanzado', 'VIP', 'Platino')
- tarjeta: tarjeta asociada al cliente con la que pagará las facturas.
    - Tipo: VARCHAR(20)

El código del método sería el siguiente:

```
public static void crearTablas(Connection conn) {
        try {
            Statement stmt = conn.createStatement();
            GestionHotel.borrarTabla(conn, "Trabajador");
            stmt.executeUpdate("CREATE TABLE Cliente ("
                    + "nombre VARCHAR(20),"
                    + "apellidos VARCHAR(40),"
                    + "telefono VARCHAR(20),"
                    + "dni VARCHAR(9),"
                    + "domicilio VARCHAR(60),"
                    + "email VARCHAR(30) CONSTRAINT email_clave_candidata UNIQUE
NOT NULL,"
                    + "puntos INTEGER,"
                    + "rango VARCHAR(20),"
                    + "tarjeta VARCHAR(20),"
                    + "PRIMARY KEY (dni)"
                    + ")");
            stmt.executeUpdate(
                    "INSERT INTO Cliente (nombre, apellidos, telefono, dni,
domicilio, email, puntos, rango, tarjeta) VALUES ('Rafael','Córdoba
Lopez','684848493','28394823G','Mesones 54','rafacorlopg@gmail.com', 0,
'Inicial', '1234567890')");
            stmt.executeUpdate(
                    "INSERT INTO Cliente (nombre, apellidos, telefono, dni,
domicilio, email, puntos, rango, tarjeta) VALUES ('Néstor','Martinez
Saez','764665788','78943659L','Puentezuelas 12','nestormm@hotmail.es', 0,
'Inicial', '0987654321')");
```

```
            stmt.close();
        } catch (SQLException e) {
            System.out.println("Error al crear la tabla Cliente: " +
e.getMessage());

        }

        // Create trigger to validate rango
        try {
            Statement stmt = conn.createStatement();
            String triggerSQL = "CREATE OR REPLACE TRIGGER trg_verificar_rango "
                    + "BEFORE INSERT OR UPDATE ON Cliente "
                    + "FOR EACH ROW "
                    + "BEGIN "
                    + "    IF :NEW.rango NOT IN ('Inicial', 'Avanzado', 'VIP',
'Platino') THEN "
                    + "        raise_application_error(-20681, 'El rango debe
ser uno de los siguientes valores: Inicial, Avanzado, VIP, Platino'); "
                    + "    END IF; "
                    + "END;";

            stmt.execute(triggerSQL);
            System.out.println("Disparador 'trg_verificar_rango' creado o
reemplazado correctamente.");
            stmt.close();
        } catch (SQLException e) {
            System.out.println("Error al crear el disparador: " +
e.getMessage());

        }
    }
```

Subsistema de suministros

El método `crearTablas` define las estructuras necesarias en la base de datos para gestionar los suministros del hotel, garantizando que la información relacionada con los productos, proveedores y su gestión esté debidamente organizada y validada. Este subsistema permite almacenar información detallada sobre los suministros, como su nombre, cantidad disponible, proveedor asociado y la última fecha de reposición. El diseño de la base de datos incluye una tabla principal, `Suministro`, y un disparador que refuerza la integridad de los datos al evitar duplicados. Los datos almacenados en la tabla son los siguientes:

Tabla suministro:

- **id**: Identificador único del suministro.
    - **Tipo**: NUMBER

    - **Restricción**: Generado automáticamente mediante `GENERATED BY DEFAULT AS IDENTITY`.
    - Es la clave primaria.
- **nombre**: Nombre del producto suministrado.
    - **Tipo**: VARCHAR2(100)
    - No puede ser nulo.
- **cantidad**: Cantidad disponible del suministro.
    - **Tipo**: NUMBER
    - No puede ser nulo.
- **proveedor**: Nombre del proveedor que entrega el suministro.
    - **Tipo**: VARCHAR2(100)
    - No puede ser nulo.
- **ultima_fecha_reposicion**: Fecha en que el suministro fue repuesto por última vez.
    - **Tipo**: DATE
    - No puede ser nulo.

Además, se incluye un **disparador** denominado `trg_suministro_duplicado`. Este disparador se activa antes de cada inserción en la tabla `Suministro` y verifica si ya existe un registro con el mismo nombre y proveedor. Si se detecta un duplicado, se genera un error que impide la inserción, garantizando así la unicidad de los datos críticos.

Código del método:

```
public static void crearTablas(Connection conn) {
    try (Statement stmt = conn.createStatement()) {
        String sqlTabla = "CREATE TABLE suministro (" +
                "id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY," +
                "nombre VARCHAR2(100)," +
                "cantidad NUMBER," +
                "proveedor VARCHAR2(100)," +
                "ultima_fecha_reposicion DATE" +
                ")";
        stmt.executeUpdate(sqlTabla);

        String sqlTrigger = "CREATE OR REPLACE TRIGGER trg_suministro_duplicado " +
                "BEFORE INSERT OR UPDATE ON suministro " +
                "FOR EACH ROW " +
                "DECLARE " +
                "v_count NUMBER; " +
                "BEGIN " +
                "SELECT COUNT(*) INTO v_count " +
                "FROM suministro " +
                "WHERE nombre = :NEW.nombre AND proveedor = :NEW.proveedor; " +
                "IF v_count > 0 THEN " +
                "RAISE_APPLICATION_ERROR(-20001, 'Error: Producto duplicado con el
mismo nombre y proveedor.'); " +
                "END IF; " +
                "END;";
```

```
        stmt.executeUpdate(sqlTrigger);
        System.out.println("Disparador 'trg_suministro_duplicado' creado o
reemplazado correctamente.");
    } catch (SQLException e) {
        System.out.println("Error al verificar o crear la tabla 'suministro': " +
e.getMessage());
    }
}
```

Subsistema de actividades

El método `crearTablas` define las tablas necesarias para el subsistema de gestión de actividades en la base de datos del hotel. Este subsistema se encarga de almacenar información sobre las actividades reservadas por los clientes del hotel, almacenando entre otros datos su nombre, fecha, aforo, etc. En este método se crean dos tablas; Actividad y contrata, las cuales almacenan información sobre las actividades del hotel y las actividades contratadas por clientes. Los datos a almacenar son:

- **Tabla Actividades**:
    - id: Identificador de la actividad
        - Tipo: NUMBER
        - Restricción: generado automáticamente con GENERATED BY DEFAULT AS IDENTITY
        - Es la clave primaria
    - nombre: nombre con el que se identifica
        - Tipo: VARCHAR2(20)
        - No puede ser nulo.
    - precio: coste de la actividad para los clientes
        - Tipo: Number(10, 2)
        - No puede ser nulo.
    - horario: fecha en la que se va a realizar
        - Tipo: Date
        - No puede ser nulo.
    - aforo: cantidad de clientes que pueden reservar la actividad
        - Tipo: Number
        - No puede ser nulo.
- **Tabla contrata**:
    - dni: Identificador del cliente que hace la reserva
        - Tipo: VARCHAR2(20)
        - No puede ser nulo.
        - Es clave foránea de la tabla Clientes.
        - Parte de la clave primaria.
    - id: Identificador de la actividad que se ha reservado
        - Tipo: Number
        - No puede ser nulo.

- Es clave foránea de la tabla Actividades.
- Parte de la clave primaria.

El código del método es:

```
public static void crearTablas(Connection conn) {
   try {
       Statement stmt = conn.createStatement();

       // Tabla Actividad
       String crearTablaActividad = """
       CREATE TABLE Actividad (
          id NUMBER GENERATED BY DEFAULT AS IDENTITY,
          nombre VARCHAR2(50) NOT NULL,
          precio NUMBER(10, 2) NOT NULL,
          horario DATE NOT NULL,
          aforo NUMBER NOT NULL,
          PRIMARY KEY (id)
       )
       """;
       stmt.executeUpdate(crearTablaActividad);

       // Tabla Contrata
       String crearTablaContrata = """
       CREATE TABLE contrata (
          dni VARCHAR2(20) NOT NULL,
          id NUMBER NOT NULL,
          PRIMARY KEY (dni, id),
          FOREIGN KEY (dni) REFERENCES Cliente(dni),
          FOREIGN KEY (id) REFERENCES Actividad(id)
       )
       """;
       stmt.executeUpdate(crearTablaContrata);

       stmt.close();

   } catch (SQLException e) {
       System.out.println("Error al crear las tablas: " + e.getMessage());
   }
}
```

Transacciones

Subsistema de reservas

Para identificar las transacciones en el subsistema de gestión de reservas, hemos considerado las operaciones que implican modificaciones en la base de datos y aquellas que requieren consistencia y control de errores para garantizar que las modificaciones se realicen correctamente. Las transacciones implementadas son las siguientes:

1. **Reservar habitación** con el método *reservarHabitacion()* donde se verifica que el cliente esté registrado, se comprueba la disponibilidad de la habitación, se calcula el precio total y se inserta la reserva. Al comprobar la disponibilidad de la habitación, se

deja un caso incorrecto para poder comprobar el funcionamiento del disparador en ese caso.
Si el DNI no está registrado o si no hay habitaciones disponibles para las fechas seleccionadas, se detiene el proceso.

2. **Cancelar reserva** con el método *cancelarReserva()* donde se elimina una reserva existente en la tabla Reserva identificada por su id. Esta operación puede implicar también la eliminación de registros en la tabla Incorpora para asegurar integridad funcional.
3. **Modificar reserva** con el método *modificarReserva()* donde se actualizan los detalles de una reserva existente, como las fechas de inicio o final y el tipo de habitación. Realiza comprobaciones similares a *reservarHabitacion* para garantizar disponibilidad de la habitación y validación de fechas.
4. **Gestionar suplementos** con el método *gestionarSuplementos()* que despliega un submenú para insertar y eliminar los suplementos de una reserva o para mostrar los suplementos disponibles.

Subsistema de facturación

Las operaciones que realizan transacciones en el subsistema de facturación son las siguientes:

1. **Añadir el método de pago al cliente** mediante el método *añadirMétodoPago()*. En primer lugar se solicita al usuario el DNI del cliente. Tras esto, se busca al cliente con el DNI dado en la base de datos para comprobar que este existe. Si el cliente existe, se solicita el nuevo número de la tarjeta y se actualiza para el cliente dado.
2. **Eliminar el método de pago** al cliente mediante el método *eliminarMétodoPago()*. En primer lugar se solicita al usuario el DNI del cliente. Tras esto, se busca al cliente con el DNI dado en la base de datos para comprobar que este existe. Si el cliente existe, se actualiza el campo tarjeta con el valor NULL.
3. **Generar la factura de una reserva** con el método *generarFactura()*. En primer lugar se solicita al usuario el identificador de la reserva. Si la reserva con el identificador dado existe, se comprueba que el cliente asociado a esa reserva exista y tenga un método de pago asociado, devolviendo al usuario un error en caso negativo. Si el cliente existe y tiene método de pago asociado, se le pide al usuario que indique el concepto de la factura. Por último se inserta una nueva factura con el concepto, hora actual del sistema y el identificador de la reserva. Por último se muestra al usuario el identificador de la factura generada para posteriores consultas.
4. **Reembolsar una factura** con el método *reembolsarFactura()*. En primer lugar, se pide al usuario que introduzca el identificador de la factura. Se comprueba que la factura existe, y en caso afirmativo, se modifica el campo *reembolsada* de la correspondiente factura a 1 (True). Existe un disparador que asegura que la factura a reembolsar no haya sido reembolsada previamente es decir, que su campo reembolsada no sea True antes de modificarlo.

Subsistema de trabajadores

Las transacciones implementadas en el subsistema de gestión de trabajadores son las siguientes:

1. **Insertar un trabajador** en la BD con el método *insertarTrabajador(conn, scanner).* Se solicitan todos los datos de un trabajador y se inserta en la tabla, siempre que no hubiese ya otro trabajador con el mismo DNI. Si hay se produce algún error debido al disparador, se pide al usuario que introduzca un nuevo valor para la nómina hasta que se proporcione uno válido. Si se produce un error por otro motivo se informa al usuario y se vuelve al submenú.

2. **Eliminar un trabajador** de la BD con el método *eliminarTrabajador(conn, scanner)*. Se solicita el DNI del trabajador a eliminar. Si existe un trabajador en la BD con ese DNI se elimina, sino se informa al usuario de que no se ha encontrado y se vuelve al submenú.
3. **Modificar datos de un trabajador** con el método *modificarTrabajador(conn, scanner)*. Se solicita el DNI del trabajador que se quiere modificar. Si no se encuentra en la BD, se informa al usuario y se vuelve al submenú. Si el DNI corresponde a algún trabajador, se solicitan los datos modificables (todos excepto el DNI) y se actualizan en la tabla. Si algún campo no se quiere modificar se puede dejar en blanco, pero no modificar ninguno produce un error. Si hay se produce algún error debido al disparador, se pide al usuario que introduzca un nuevo valor para la nómina hasta que se proporcione uno válido. Si se produce un error por otro motivo se informa al usuario y se vuelve al submenú.
4. **Mostrar los datos de un trabajador** con el método *consultarTrabajador(conn, scanner)*. Se solicita el DNI del trabajador a consultar. Si existe un trabajador en la BD con ese DNI se muestran sus datos, sino se informa al usuario de que no se ha encontrado y se vuelve al submenú.
5. **Mostrar los datos de todos los trabajadores** en la BD con el método *mostrarTablas(conn)*. Se llama a la función *mostrarTablas(conn, tableName)* de la clase *GestionHotel*. Esta función recibe el nombre de la tabla (“Trabajador”) y sirve para que todos los subsistemas muestren las tablas de forma uniforme.

Subsistema de suministros

El subsistema de gestión de suministros implementa transacciones clave que garantizan la consistencia y el control de errores en la base de datos. Las operaciones principales son las siguientes:

1. **Añadir suministro**: Mediante el método `anadirSuministro()`, se posibilita el registro de nuevos suministros en la base de datos. Este proceso incluye una validación exhaustiva para garantizar que no existan duplicados que compartan el mismo nombre y proveedor. Para reforzar esta validación, se utiliza un disparador que impide la inserción de datos duplicados, lo que asegura la integridad de la información desde el momento de su creación.
2. **Modificar suministro**: A través del método `modificarSuministro()`, se pueden actualizar los detalles de un suministro existente. Esto incluye cambios en el nombre, la cantidad o el proveedor asociado. Antes de realizar la actualización, el sistema verifica la validez del ID proporcionado, garantizando que los cambios sólo se apliquen a registros válidos y existentes. Además esta función también se ve

afectada por el trigger de comprobación de suministros duplicados del mismo proveedor.

3. **Eliminar suministro**: El método `eliminarSuministro()` permite eliminar registros específicos de la tabla de suministros. Esta operación está diseñada para procesar únicamente aquellos IDs que sean válidos, lo que evita la eliminación accidental o incorrecta de información. Además, asegura que la base de datos mantenga su consistencia después de cada operación de eliminación.
4. **Mostrar suministros**: Con el método `mostrarSuministros()`, los usuarios pueden obtener una lista completa de todos los suministros registrados en la base de datos. Esta funcionalidad es crucial para proporcionar una visión general del inventario y apoyar la toma de decisiones basadas en la información disponible.
5. **Filtrar productos por proveedor**: Esta operación permite a los usuarios realizar consultas específicas para listar únicamente aquellos suministros asociados a un proveedor determinado. Esta funcionalidad es especialmente útil para realizar análisis enfocados o para localizar información de manera más rápida y precisa.

Subsistema de clientes

Las transacciones implementadas en el subsistema de gestión de clientes son las siguientes:

1. **Dar de alta a un cliente** con el método *darAltaCliente()* donde se pide al usuario que introduzca sus datos personales entre ellos el DNI, si el DNI ya existe en la Base de Datos se muestra un mensaje de error y se anula el proceso. Además, si el rango introducido no es alguno de los permitidos, salta el disparador y se detiene el proceso. Si los datos introducidos son válidos se inserta el cliente en la base de datos.
2. **Dar de baja un cliente** con el método *darBajaCliente()* donde se elimina un cliente existente en la tabla Cliente identificado por su DNI. Esta operación puede realizarse si el DNI introducido existe en la Base de datos, sino se detiene el proceso.
3. **Consultar la información de un Cliente** con el método *consultarCliente*() donde se muestra la información de un cliente existente en la tabla Cliente identificado por su DNI. Esta operación puede realizarse si el DNI introducido existe en la Base de datos, sino se detiene el proceso.
4. **Modificar información de un cliente** con el método *modificarCliente()* donde se actualizan los datos de un cliente existente. Realiza comprobaciones similares a *darAltaCliente()* para garantizar el rango correcto del cliente, el email, etc. Esta operación puede realizarse si el DNI introducido existe en la Base de datos, sino se detiene el proceso.
5. **Consultar rango del cliente** con el método *consultarRangoCliente()* que muestra el rango de un cliente existente en la tabla Cliente identificado por su DNI. Esta operación puede realizarse si el DNI introducido existe en la Base de datos, sino se detiene el proceso.

Subsistema de servicios

Las transacciones implementadas en el subsistema de gestión de servicios son las siguientes:

1. **Registrar actividad** con el método `registrarActividad()`, se inserta una nueva actividad en la tabla `Actividad`. Durante esta operación, se valida que los datos introducidos sean correctos, como que el precio sea mayor o igual a cero, que el aforo sea un número positivo y que la fecha y hora de la actividad sean posteriores a la fecha actual. También se deja un caso incorrecto que desencadena el funcionamiento del disparador en la base de datos para validar el aforo y las fechas.

2. **Eliminar actividad** con el método `eliminarActividad()`, se elimina una actividad existente de la tabla `Actividad`, identificada por su ID. Además, se eliminan los registros asociados en la tabla `contrata` para mantener la integridad referencial. Si la actividad no existe o no se encuentra, el proceso no tiene efecto. Cualquier error durante esta operación implica un rollback para deshacer los cambios realizados.

3. **Contratar actividad** con el método `contratarActividad()`, un cliente puede contratar una actividad específica, siempre que cumpla con las siguientes condiciones:
    - El cliente debe estar registrado en la base de datos (`Cliente`).
    - La actividad debe tener aforo disponible.
    - El cliente no puede contratar más de una actividad al mismo tiempo.

    Si las condiciones se cumplen, se inserta un registro en la tabla `contrata`. Si alguna de las validaciones falla, el proceso se detiene y se deshacen los cambios en la transacción.

4. **Cancelar actividad** con el método `cancelarActividad()`, se elimina una actividad previamente contratada por un cliente, identificada por su DNI. La operación elimina el registro correspondiente en la tabla `contrata`. Si no hay actividades contratadas con el DNI proporcionado, no se realiza ninguna modificación. Ante cualquier error, se realiza un rollback.

5. **Crear tablas y disparadores** con el método `crearTablas()`. Asegura la correcta creación de las tablas `Actividad` y `contrata` en la base de datos. También se genera el disparador `trg_validar_actividad` para garantizar que los datos insertados en la tabla `Actividad` cumplan con las restricciones requeridas, como fechas válidas y un aforo mayor a cero. Este método incluye inserciones iniciales de datos de prueba y realiza un commit al finalizar, o un rollback en caso de error.

6. **Mostrar actividades y filtrado**, aunque estas operaciones no modifican los datos de la base de datos, los métodos como `mostrarActividades()` y `mostrarActividadesConFiltrado()` permiten al usuario consultar las actividades existentes y aplicar filtros según diferentes criterios. Estas operaciones son de solo lectura y no requieren transacciones.

Sentencias SQL de creación de disparadores

Subsistema de reservas: Carmen Azorín Martí

Verificar la disponibilidad al crear una reserva. El objetivo es garantizar que solo se cree una reserva si hay habitaciones disponibles del tipo solicitado para las fechas específicas.

```
CREATE OR REPLACE TRIGGER trg_check_habitacion_disponible
BEFORE INSERT ON Reserva
FOR EACH ROW
DECLARE
  habitaciones_ocupadas NUMBER;
BEGIN
  SELECT COUNT(*)
  INTO habitaciones_ocupadas
  FROM reserva
  WHERE habitacion = :NEW.habitacion
      AND (:NEW.fechaInicio BETWEEN fechaInicio AND fechaFinal
          OR :NEW.fechaFinal BETWEEN fechaInicio AND fechaFinal
          OR fechaInicio BETWEEN :NEW.fechaInicio AND :NEW.fechaFinal
          OR fechaFinal BETWEEN :NEW.fechaInicio AND :NEW.fechaFinal);

  IF habitaciones_ocupadas > 0 THEN
     raise_application_error(-20001, 'La habitación ya está reservada en las fechas
especificadas.');
  END IF;
END;
```

Subsistema de facturación: Pablo Luque Salguero

Validar el reembolso. El objetivo es garantizar que la factura no haya sido previamente reembolsada.

```
CREATE OR REPLACE TRIGGER validar_reembolso
BEFORE UPDATE ON Factura
FOR EACH ROW
BEGIN
       IF :OLD.reembolsada = 1 THEN
RAISE_APPLICATION_ERROR(-20301, 'La factura ya ha sido reembolsada.');
END IF;
END;
```

Subsistema de trabajadores: Jaime Martínez Bravo

Validar el salario de un trabajador. El objetivo es garantizar que el salario sea superior al salario mínimo, y que no hayan pasado más de dos años desde la última modificación del sueldo del trabajador.

```
CREATE OR REPLACE TRIGGER trg_verificar_sueldo
```

```
    BEFORE INSERT OR UPDATE ON Trabajador
    FOR EACH ROW
    DECLARE
        salario_minimo CONSTANT NUMBER := 1134;
        fecha_actual DATE := SYSDATE;

BEGIN
    IF :NEW.nomina < salario_minimo THEN
        raise_application_error(-20601, 'El salario no puede ser inferior al salario
mínimo interprofesional: ' || salario_minimo || ' euros');
    END IF;
    IF ABS(MONTHS_BETWEEN(fecha_actual, :NEW.fecha_ultimo_aumento)) > 24 THEN
       raise_application_error(-20602, 'Han pasado más de 2 años sin que se modifique el
sueldo del trabajador con DNI: ' || :NEW.dni);
    END IF;
    IF UPDATING AND :NEW.nomina = :OLD.nomina THEN
       IF ABS(MONTHS_BETWEEN(:OLD.fecha_ultimo_aumento, :NEW.fecha_ultimo_aumento)) > 24
THEN
          raise_application_error(-20602, 'Han pasado más de 2 años sin que se modifique
el sueldo del trabajador con DNI: ' || :NEW.dni);
       END IF;
    END IF;
END;
```

Subsistema de gestión de clientes: Rafael Luque Framit

Validar el valor del rango del cliente. El objetivo es asegurar que el valor esté dentro del ENUM permitido.

```
CREATE OR REPLACE TRIGGER trg_verificar_rango
   BEFORE INSERT OR UPDATE ON Cliente
   FOR EACH ROW
   BEGIN
       IF :NEW.rango NOT IN ('Inicial', 'Avanzado', 'VIP', 'Platino') THEN
              raise_application_error(-20601, 'El rango debe ser uno de los siguientes
valores: Inicial, Avanzado, VIP, Platino');
   END IF;
END;
```

Subsistema de suministros: Adrián Jaén Fuentes

Comprobar la existencia de un producto duplicado al insertar un nuevo registro. El objetivo es garantizar que no haya productos con el mismo nombre y proveedor en la base de datos y, en caso de que no exista duplicado, asignar automáticamente un nuevo ID único al producto.

```
CREATE OR REPLACE TRIGGER check_duplicate_suministro
```

```
    BEFORE INSERT OR UPDATE ON suministro
    FOR EACH ROW
    DECLARE
        v_count NUMBER;
    BEGIN
        SELECT COUNT(*) INTO v_count
        FROM suministro
        WHERE nombre = :NEW.nombre AND proveedor = :NEW.proveedor;
        IF v_count > 0 THEN
                RAISE_APPLICATION_ERROR(-20001, 'Error: Producto duplicado con el mismo
nombre y proveedor.');
        END IF;
    END;
```

Subsistema de actividades: Juan Pedro Moreno Ruiz

Validar las inserciones de actividades. El objetivo es que al insertar una nueva actividad se verifica que el aforo sea mayor que 0 y que la fecha sea válida, si no da error y no es posible insertarla.

```
CREATE OR REPLACE TRIGGER trg_validar_actividad
BEFORE INSERT ON Actividad
FOR EACH ROW
BEGIN
   IF :NEW.aforo <= 0 THEN
       RAISE_APPLICATION_ERROR(-20001, 'El aforo debe ser mayor que 0.');
   END IF;

   IF :NEW.horario < SYSDATE THEN
           RAISE_APPLICATION_ERROR(-20002, 'La fecha debe ser igual o posterior a la
actual.');
   END IF;
END;
```

Trámites Legales y Responsabilidades

- **Registro de la Base de Datos:** en España, las bases de datos que contienen datos personales deben ser registradas en la Agencia Española de Protección de Datos (AEPD).
- **Responsabilidad del Tratamiento de Datos:** el responsable del tratamiento de los datos debe garantizar que se cumplan todas las normativas de protección de datos, como el Reglamento General de Protección de Datos (RGPD) de la Unión Europea. Debe asegurarse de que los datos se recopilan, almacenan y procesan de manera segura y solo para los fines específicos para los que fueron recopilados.

- **Consentimiento:** es necesario obtener el consentimiento explícito de los trabajadores para recopilar y procesar sus datos personales. Los trabajadores deben ser informados sobre cómo se utilizarán sus datos y sus derechos en relación con sus datos personales.
- **Derechos de los Trabajadores:** los trabajadores tienen derecho a acceder, rectificar, cancelar y oponerse al tratamiento de sus datos personales. Deben ser informados sobre cómo pueden ejercer estos derechos.
- **Medidas de Seguridad:** implementar medidas técnicas y organizativas adecuadas para proteger los datos personales contra el acceso no autorizado, la pérdida o la destrucción. Asegurar que solo el personal autorizado tenga acceso a los datos sensibles.
- **Notificación de Brechas de Seguridad:** en caso de una brecha de seguridad que afecte a los datos personales, se debe notificar a la AEPD y a los afectados dentro de las 72 horas siguientes a la detección de la brecha.
- **Organismos Competentes:**
    - *Agencia Española de Protección de Datos (AEPD):* es el organismo encargado de velar por el cumplimiento de la normativa de protección de datos en España.
    - *Autoridades Laborales:* dependiendo de la legislación local, puede ser necesario informar a las autoridades laborales sobre la gestión de los datos de los trabajadores.