UNIVERSIDAD DE GRANADA

Memoria prácticas BDD

Base de Datos Distribuidas

Yunkai Lin Pan
Raúl Rodríguez Rodríguez

Universidad de Granada
España

7 de enero de 2025

Índice

1. Introducción

En este documenta refleja el trabajo recolectado durante las prácticas de la asigantura `Base de Datos Distribuidos`. En éste, se graba los diagramas que hemos empleado, las decisiones que hemos tomado tanto para hacer los diagramas como las fragmentaciones.

Estas prácticas se basa en el enunciado dado sobre una multinacional hotelera andaluza cuya intención es implementar una base de datos distribuidas cuyo propósito consiste en la gestión y almacenamiento de datos de empleados, clientes y proveedores de cada una de sus cadenas. El almacenamiento reside de forma física en cada una de las 4 localidades que se ha planteado

2. Diseño conceptual y lógico global

En esta sección, se recoge los diagramas y tablas que hemos generado durante esta primera fase (diagramas ER y el paso a tablas). Para ello, hemos usado la herramienta `draw.io` [1].

2.1. Diagramas ER

Planteamos dos diseños, un esquema inicial con se han ido definiendo los conceptos (Figura 1) y un esquema final, sobre la que trabajaremos (Figura 2).

2.1.1. Planteamiento inicial

Durante unos esquemas iniciales del diagrama ER, nos planteamos la siguiente propuesta (Figura 1). Al no tener claro cómo podríamos saber que un empleado podía ser un director y relacionarse con un hotel, pensamos en que podría ser una buena idea mantener en los empleados un atributo que nos indique si es director.

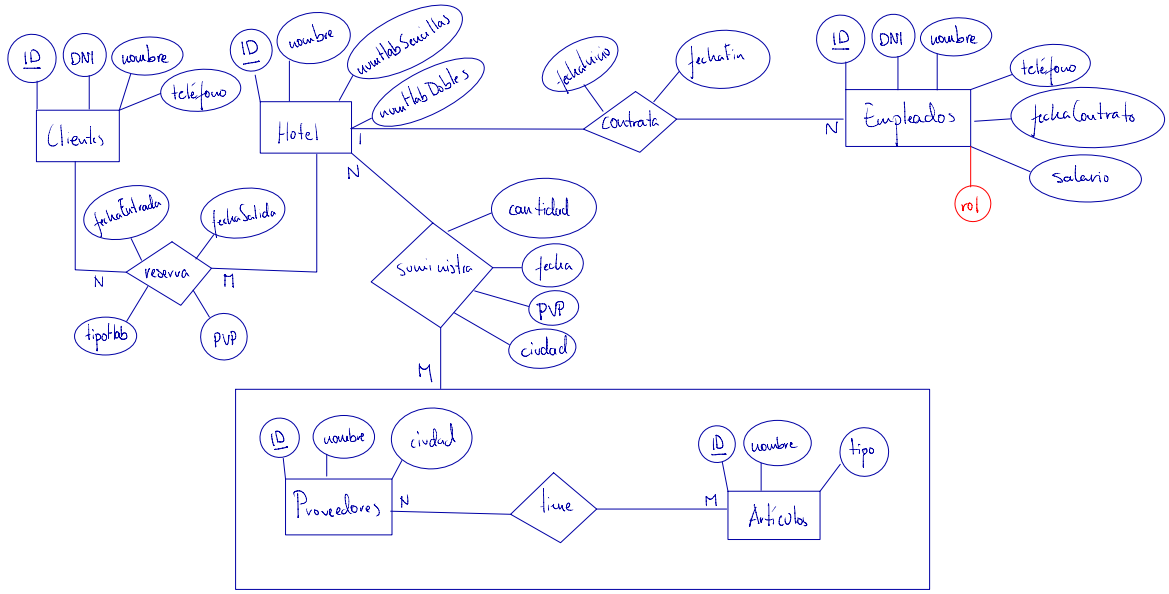

Figura 1: Diagrama ER inicial

2.1.2. Planteamiento final

Este modelo (Figura 2), la cual hemos escogido para trabajar, pensamos que es el mejor que se adecúa con los datos de ejemplo. Además, mediante la relación 1 a 1 (dirige) entre Hotel y Empleado. De esta forma, durante el mapeo de ER a tablas, en la entidad Hotel podemos encontrarnos con un atributo de empleado que nos indica que es el director y que a su vez es una referencia o clave externa a un código de empleado. Por otro lado, nos encontramos con una agregación entre las entidades Proveedor y Artículo. Dicha agregación se debe a que para relacionar un suministro con un hotel, no tiene sentido que hagamos una relación sólo de Hotel con Proveedor u Hotel con Artículo, ya que no se estaría dando la información relevante que necesitamos. Mediante el agregado, conseguimos unificar los proveedores con los artículos que nos devuelven la información que necesitamos para relacionar con los hoteles. La siguiente imagen es el diagrama ER con la que vamos a trabajar.

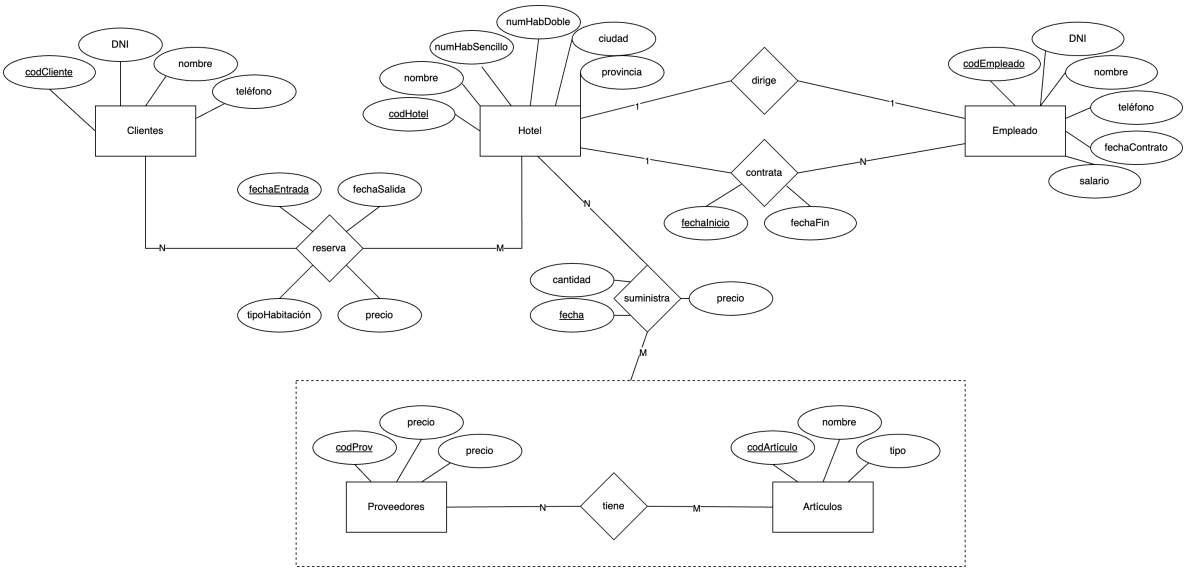

Figura 2: Diagrama ER final

2.2. Conclusiones

En el planteamiento final del diagrama ER (Figura 2), establecemos que dicho ER es el con el que vamos a trabajar, puesto que simplemente pensamos que va a ser el que mejor que se va a adecuar.

Sin embargo, el diagrama inicial que tenemos (Figura 1), nos plantearía un problema en le futuro a la hora de insertar un empleado, puesto teniendo el atributo *rol*, implica que dicho atributo podría tomar x valores, pero seguiríamos sin tener de alguna forma, localizar cuál hotel dirige un empleado si el rol que tiene es de director.

2.3. Tablas

Habiendo escogido el diagrama que tenemos como planteamiento final (ver Figura 2), hemos generado este conjunto de tablas:

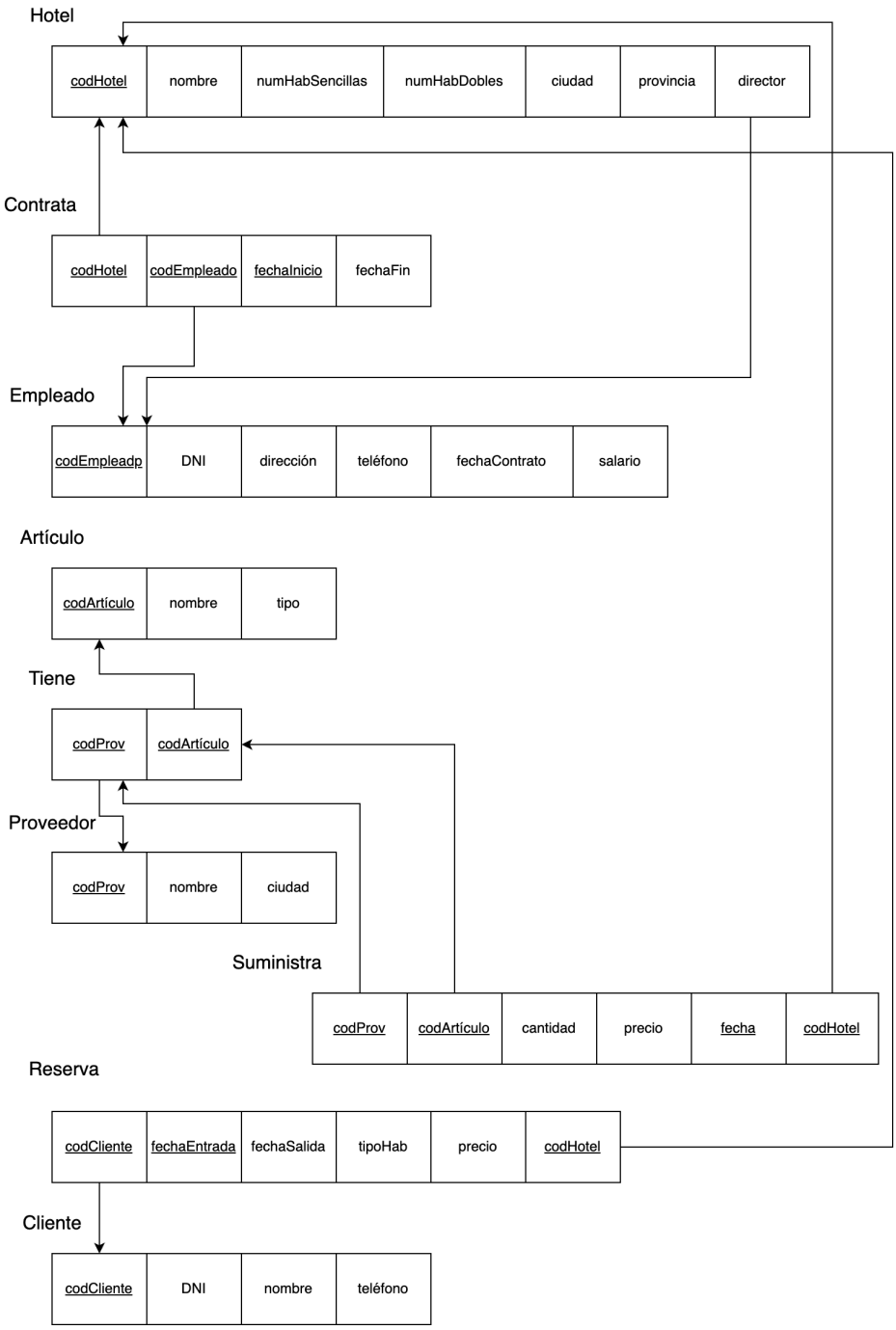

Figura 3: Conjunto de tablas

De esta forma, se ve claramente, las tablas generadas, las claves primarias (subrayadas) así como de las claves de las que referencian, puesto que basta con seguir el sentido de las flechas.

3. Diseño de fragmentación y asignación

Partiendo de que tenemos las localidades, la vamos a denominar de las siguientes formas para simplificar:

- L1: localidad Granada – provincias de Granada y Jaén
- L2: localidad Málaga – provincias de Málaga y Almería
- L3: localidad Sevilla – provincias de Sevilla y Córdoba
- L4: localidad Cádiz – provincias de Cádiz y Huelva.

A simple vista, viendo los atributos que representan las ciudades, concluimos que hay 2 tablas que se pueden fragmentar horizontalmente y que sean primarias. Estas tablas son las referidas a los hoteles y los proveedores.

3.1. Diseño

Los hoteles contienen los atributos ciudad y provincia, por lo que nos delataría con qué atributos con la que procede la fragmentación. Puesto que por cada localidad contiene una pareja de provincias, los hoteles la vamos a fragmentar horizontalmente por provincia.

Para ver si se trata de una fragmentación horizontal primaria, tenemos que ver los predicados simples, que por simplicidad, la vamos a nombrar como a continuación:

- $p_1 \equiv provincia = \ 'Granada'$
- $p_2 \equiv provincia = \ 'Jaen'$
- $p_3 \equiv provincia = \ 'Malaga'$
- $p_4 \equiv provincia = \ 'Almeria'$
- $p_5 \equiv provincia = \ 'Sevilla'$
- $p_6 \equiv provincia = \ 'Cordoba'$
- $p_7 \equiv provincia = \ 'Cadiz'$
- $p_8 \equiv provincia = \ 'Huelva'$

Teniendo estos predicados, obtendremos un total de 256 conjunciones (aplicando la fórmula 2^n donde $n = num.\ predicados\ simples$, equivalente a 8 en este caso).

Así tendremos este término de predicados:

- $y_1 = p_1 \wedge \ \neg p_2 \wedge \ \neg p_3 \wedge \ \neg p_4 \wedge \ \neg p_5 \wedge \ \neg p_6 \wedge \ \neg p_7 \wedge \ \neg p_8 \Rightarrow$ verdadero
- $y_2 = \neg p_1 \wedge \ p_2 \wedge \ \neg p_3 \wedge \ \neg p_4 \wedge \ \neg p_5 \wedge \ \neg p_6 \wedge \ \neg p_7 \wedge \ \neg p_8 \Rightarrow$ verdadero
- $y_3 = \neg p_1 \wedge \ \neg p_2 \wedge \ p_3 \wedge \ \neg p_4 \wedge \ \neg p_5 \wedge \ \neg p_6 \wedge \ \neg p_7 \wedge \ \neg p_8 \Rightarrow$ verdadero
- $y_4 = \neg p_1 \wedge \ \neg p_2 \wedge \ \neg p_3 \wedge \ p_4 \wedge \ \neg p_5 \wedge \ \neg p_6 \wedge \ \neg p_7 \wedge \ \neg p_8 \Rightarrow$ verdadero
- $y_5 = \neg p_1 \wedge \ \neg p_2 \wedge \ \neg p_3 \wedge \ \neg p_4 \wedge \ p_5 \wedge \ \neg p_6 \wedge \ \neg p_7 \wedge \ \neg p_8 \Rightarrow$ verdadero
- $y_6 = \neg p_1 \wedge \ \neg p_2 \wedge \ \neg p_3 \wedge \ \neg p_4 \wedge \ \neg p_5 \wedge \ p_6 \wedge \ \neg p_7 \wedge \ \neg p_8 \Rightarrow$ verdadero

- $y_7 = \neg p_1 \wedge \neg p_2 \wedge \neg p_3 \wedge \neg p_4 \wedge \neg p_5 \wedge \neg p_6 \wedge p_7 \wedge \neg p_8 \Rightarrow$ verdadero
- $y_8 = \neg p_1 \wedge \neg p_2 \wedge \neg p_3 \wedge \neg p_4 \wedge \neg p_5 \wedge \neg p_6 \wedge \neg p_7 \wedge p_8 \Rightarrow$ verdadero
- $y_9 = \neg p_1 \wedge \neg p_2 \wedge \neg p_3 \wedge \neg p_4 \wedge \neg p_5 \wedge \neg p_6 \wedge \neg p_7 \wedge \neg p_8 \Rightarrow$ falso
- ...
- $y_{256} = p_1 \wedge p_2 \wedge p_3 \wedge p_4 \wedge p_5 \wedge p_6 \wedge p_7 \wedge p_8 \Rightarrow$ falso

Como se puede observar, tenemos 8 predicados que se pueden evaluar a verdadero, por lo que tendremos 8 fragmentos para `Hoteles`, puesto que tenemos que las expresiones $y_1 ... y_8$ se evalúan a verdadero puesto que un hotel sólo se puede ubicar en una provincia y no en dos provincias distintas (misma razón que se aplica a la expresión y_{256}). Ademas, la expresión y_9 se evalúa a falso puesto que asumimos que la multinacional no va a tener hoteles que no estén ubicados en el dominio de provincias que tenemos definido.

Teniendo en cuenta lo anterior, podemos tener el siguiente esquema de fragmentación para *Hoteles*:

- $Hotel_1 = SL_{y_1}(Hotel)$
- $Hotel_2 = SL_{y_2}(Hotel)$
- $Hotel_3 = SL_{y_3}(Hotel)$
- $Hotel_4 = SL_{y_4}(Hotel)$
- $Hotel_5 = SL_{y_5}(Hotel)$
- $Hotel_6 = SL_{y_6}(Hotel)$
- $Hotel_7 = SL_{y_7}(Hotel)$
- $Hotel_8 = SL_{y_8}(Hotel)$

Si seguimos este esquema de fragmentación, contando que tenemos 2 relaciones que requieren del hotel, nos vemos que las relaciones o tablas *Contrata* y *Reserva* se proceden a realizar una fragmentación horizontal derivada, puesto que no tendría sentido que si un empleado trabaja en el hotel de Granada, no tiene sentido que el hotel de Sevilla quiera saber información de ese contrato, sino el propio hotel que contrata al empleado.

Para lograr la fragmentación horizontal derivada, se va a realizar mediante el atributo *codHotel*, ya que ambas relaciones dependen del hotel.

Para la relación `Contrata`:

- $Contrata_1 = Contrata\ SJ_{codHotel=codHotel}(Hotel_1)$
- $Contrata_2 = Contrata\ SJ_{codHotel=codHotel}(Hotel_2)$
- $Contrata_3 = Contrata\ SJ_{codHotel=codHotel}(Hotel_3)$
- $Contrata_4 = Contrata\ SJ_{codHotel=codHotel}(Hotel_4)$
- $Contrata_5 = Contrata\ SJ_{codHotel=codHotel}(Hotel_5)$
- $Contrata_6 = Contrata\ SJ_{codHotel=codHotel}(Hotel_6)$
- $Contrata_7 = Contrata\ SJ_{codHotel=codHotel}(Hotel_7)$

- $Contrata_8 = Contrata\ SJ_{codHotel=codHotel}(Hotel_8)$

Para la relación `Reserva`:

- $Reserva_1 = Reserva\ SJ_{codHotel=codHotel}(Hotel_1)$
- $Reserva_2 = Reserva\ SJ_{codHotel=codHotel}(Hotel_2)$
- $Reserva_3 = Reserva\ SJ_{codHotel=codHotel}(Hotel_3)$
- $Reserva_4 = Reserva\ SJ_{codHotel=codHotel}(Hotel_4)$
- $Reserva_5 = Reserva\ SJ_{codHotel=codHotel}(Hotel_5)$
- $Reserva_6 = Reserva\ SJ_{codHotel=codHotel}(Hotel_6)$
- $Reserva_7 = Reserva\ SJ_{codHotel=codHotel}(Hotel_7)$
- $Reserva_8 = Reserva\ SJ_{codHotel=codHotel}(Hotel_8)$

Para la entidad `Empleado`, la fragmentaremos por una fragmentación horizontal derivada del hotel en la que trabajan mediante el atributo *codEmpleado*:

- $Empleado_1 = Empleado\ SJ_{codEmpleado=codEmpleado}(Hotel_1)$
- $Empleado_2 = Empleado\ SJ_{codEmpleado=codEmpleado}(Hotel_2)$
- $Empleado_3 = Empleado\ SJ_{codEmpleado=codEmpleado}(Hotel_3)$
- $Empleado_4 = Empleado\ SJ_{codEmpleado=codEmpleado}(Hotel_4)$
- $Empleado_5 = Empleado\ SJ_{codEmpleado=codEmpleado}(Hotel_5)$
- $Empleado_6 = Empleado\ SJ_{codEmpleado=codEmpleado}(Hotel_6)$
- $Empleado_7 = Empleado\ SJ_{codEmpleado=codEmpleado}(Hotel_7)$
- $Empleado_8 = Empleado\ SJ_{codEmpleado=codEmpleado}(Hotel_8)$

La entidad `Clientes` no se van a fragmentar, puesto que no tenemos ninguna información sustancial que nos indique la posibilidad de realizar una fragmentación (podríamos hacer la fragmentación en relación a la dirección de alojamiento que tienen asociado pero dicho dato no es proporcionado, o bien, podríamos fragmentar por el número de teléfono que si es un número fijo, se podría deducir la ciudad y mantenerlo como un criterio).

Los proveedores contienen el atributo ciudad, por lo que se usará este para hacer la fragmentación. Dado que hay dos distribuidores posibles, aunque abastezcan cada uno a cuatro provincias, se obtiene el siguiente conjunto de calificaciones, de forma simple:

- $q_1 \equiv provincia = \ 'Granada'$
- $q_2 \equiv provincia = \ 'Sevilla'$

Con solo 2 predicados, obtenemos un total de 4 conjunciones:

- $z_1 = q_1 \wedge \ \neg q_2 \Rightarrow$ verdadero
- $z_2 = \neg q_1 \wedge \ q_2 \Rightarrow$ verdadero
- $z_3 = \neg q_1 \wedge \ \neg q_2 \Rightarrow$ falso

- $z_4 = q_1 \wedge \ q_2 \Rightarrow$ falso

Estas expresiones se evalúan de la forma tal que solamente trabajamos con el dominio de ciudades dentro de Andalucía, que es con la que trabajamos. Como los proveedores sólo parten de Granada o Sevilla, las expresiones $z_1 - z_2$ se evalúan a verdad puesto que vienen bien de un sitio o bien de otro. Sin embargo, las dos restantes, no tiene sentido que un proveedor vengan de dos lugares distintos (como se define en el enunciado de la práctica) o bien que el proveedor no parta de ningún lado dentro del ámbito que hemos definido.

De esta forma, el esquema de fragmentación para los `Proveedor` es:

- $Proveedor_1 = SL_{z_1}(Proveedor)$
- $Proveedor_2 = SL_{z_2}(Proveedor)$

De la misma forma que hemos realizado las fragmentaciones horizontales y derivadas de *Reserva* y `Contrata`, procedemos con una fragmentación horizontal derivada para la relación `Tiene` mediante el atributo *codProv* (sabiendo que las aplicaciones requiere el conocimiento de lo que tiene en posesión el proveedor para realizar una venta):

- $Tiene_1 = Tiene \ SJ_{codProv=codProv}(Proveedor_1)$
- $Tiene_2 = Tiene \ SJ_{codProv=codProv}(Proveedor_2)$

La entidad `Artículos` no tenemos una razón lógica para realizar la fragmentación de una forma relevante como pueden ser las fragmentaciones por las ciudades.

Para la última relación que se nos quedaría, `Suministra`, podemos bien realizar una fragmentación horizontal derivada bien por el proveedor o bien por el hotel. En nuestro caso, nos decantamos por la entidad `Proveedor` mediante el atributo *codProv*:

- $Suministra_1 = Suministra \ SJ_{codProv=codProv}(Proveedor_1)$
- $Suministra_2 = Suministra \ SJ_{codProv=codProv}(Proveedor_2)$

3.2. Reconstrucción

Una vez obtenidos los fragmentos, procedemos con la reconstrucción de las mismas. Al ser todas fragmentaciones horizontales y horizontales derivadas, se reconstruyen mediante el operador `UNIÓN`.

El operador `UNIÓN` la representaremos mediante el símbolo $\cup$.

- Hotel $= Hotel_1 \cup Hotel_2 \cup Hotel_3 \cup Hotel_4 \cup Hotel_5 \cup Hotel_6 \cup Hotel_7 \cup Hotel_8$
- Contrata $= Contrata_1 \cup Contrata_2 \cup Contrata_3 \cup Contrata_4 \cup Contrata_5 \cup Contrata_6 \cup Contrata_7 \cup Contrata_8$
- Empleado $= Empleado_1 \cup Empleado_2 \cup Empleado_3 \cup Empleado_4 \cup Empleado_5 \cup Empleado_6 \cup Empleado_7 \cup Empleado_8$
- Reserva $= Reserva_1 \cup Reserva_2 \cup Reserva_3 \cup Reserva_4 \cup Reserva_5 \cup Reserva_6 \cup Reserva_7 \cup Reserva_8$
- Proveedor $= Proveedor_1 \cup Proveedor_2$
- Tiene $= Tiene_1 \cup Tiene_2$
- Suministra $= Suministra_1 \cup Suministra_2$

3.3. Asignación

Fragmentos asignados a la localidad 1 (Granada y Jaén):

- $Hotel_1, Hotel_2$
- $Empleado_1, Empleado_2$
- $Contrata_1, Contrata_2$
- $Reserva_1, Reserva_2$
- $Proveedor_1$
- $Suministra_1$
- $Tiene_1$

Fragmentos asignados a la localidad 2 (Málaga y Almería):

- $Hotel_3, Hotel_4$
- $Empleado_3, Empleado_4$
- $Contrata_3, Contrata_4$
- $Reserva_3, Reserva_4$

Fragmentos asignados a la localidad 3 (Sevilla y Córdoba):

- $Hotel_5, Hotel_6$
- $Empleado_5, Empleado_6$
- $Contrata_5, Contrata_6$
- $Reserva_5, Reserva_6$
- $Proveedor_2$
- $Suministra_2$
- $Tiene_2$

Fragmentos asignados a la localidad 4 (Cádiz y Huelva):

- $Hotel_7, Hotel_8$
- $Empleado_7, Empleado_8$
- $Contrata_7, Contrata_8$
- $Reserva_7, Reserva_8$

Mientras tanto, como se mencionó previamente, replicamos las tablas `Cliente` y `Articulo`.

4. Implementación del diseño

En esta sección, vamos anotando las sentencias SQL con la que usaremos para las bases de datos de cada localidad. De forma general, añadiremos el código empleado para crear las tablas y vistas así como la concesión de privilegios, disparadores y procedimientos de una localidad solo, puesto que éstos son análogas cada localidad (salvo 2 que no tiene los fragmentos relacionados con los proveedores). Además, hemos decidido también fusionar fragmentos, es decir, si en la localidad 1 contiene dos fragmentos de hoteles, aquellas que se sitúan en las provincias de Jaén y Granada, en vez de crearnos dos tablas diferentes, crearemos una única que contenga todos los datos (y así sucesivamente con todos los fragmentos).

4.1. Creación de tablas

4.1.1. Tablas replicadas

Empezaremos con las tablas que se van a replicar (Clientes y Articulo). Estas sentencias SQL van a ser ejecutadas tal y como están a continuación en todas las localidades.

Tabla Cliente

```
CREATE TABLE Cliente(
    codCliente NUMBER PRIMARY KEY,
    DNI VARCHAR2(8) UNIQUE NOT NULL,
    nombre VARCHAR2(50),
    telefono VARCHAR2(6)
);
```

Tabla Articulo

```
CREATE TABLE Articulo(
    codArticulo NUMBER PRIMARY KEY,
    nombre VARCHAR2(50) NOT NULL,
    tipo CHAR(1) CHECK (tipo IN ('A', 'B', 'C', 'D'))
);
```

4.1.2. Tablas fragmentadas

Sentencias SQL de las tablas fragmentadas (en las localidades que no se aloja los proveedores no va a contener las tablas que se les asocia como los suministros que realiza o los artículos que tiene):

Tabla Hotel

```
CREATE TABLE Hotel (
    codHotel NUMBER PRIMARY KEY,
    nombre VARCHAR2(100),
    numHabSencillas INTEGER NOT NULL,
    numHabDobles INTEGER NOT NULL,
    ciudad VARCHAR2(50) NOT NULL,
    provincia VARCHAR2(50) NOT NULL,
    director NUMBER UNIQUE,
);
```

Tabla Empleado

```
CREATE TABLE Empleado(
    codEmpleado NUMBER PRIMARY KEY,
    DNI VARCHAR2(8) UNIQUE NOT NULL,
    nombre VARCHAR2(100) NOT NULL,
    direccion VARCHAR2(100),
    telefono VARCHAR2(6),
    fechaContrato DATE NOT NULL,
    salario NUMBER NOT NULL
);
```

Tabla Contrata

```
CREATE TABLE Contrata(
    codHotel NUMBER,
    codEmpleado NUMBER,
    fechaInicio DATE NOT NULL,
    fechaFin DATE,
    PRIMARY KEY (codHotel, codEmpleado, fechaInicio),
    FOREIGN KEY (codEmpleado) REFERENCES Empleado(codEmpleado),
    ADD CONSTRAINT check_fecha CHECK (fechaInicio < fechaFin)
);
```

Tabla Reserva

```
CREATE TABLE Reserva(
    codCliente NUMBER,
    codHotel NUMBER,
    fechaEntrada DATE NOT NULL,
    fechaSalida DATE NOT NULL,
    tipoHab VARCHAR(10) NOT NULL,
    precio NUMBER NOT NULL,
    PRIMARY KEY (codCliente, codHotel, fechaEntrada),
    FOREIGN KEY (codCliente) REFERENCES Cliente(codCliente),
    FOREIGN KEY (codHotel) REFERENCES Hotel(codHotel),
    CHECK (tipoHab IN ('Sencilla', 'Doble')),
    CONSTRAINT chk_fecha CHECK (fechaSalida >= fechaEntrada)
);
```

Tabla Proveedor

```
CREATE TABLE Proveedor(
    codProv NUMBER PRIMARY KEY,
    nombre VARCHAR2(100),
    ciudad VARCHAR2(50) CHECK (ciudad IN ('ciudadX', 'ciudadY'))
);
```

Tabla Tiene

```
CREATE TABLE Tiene(
    codProv NUMBER,
    codArticulo NUMBER,
    PRIMARY KEY (codProv, codArticulo),
```

```
    FOREIGN KEY (codProv) REFERENCES Proveedor(codProv),
    FOREIGN KEY (codArticulo) REFERENCES Articulo(codArticulo)
);
```

Tabla Suministra

```
CREATE TABLE Suministra(
    codProv NUMBER,
    codHotel NUMBER,
    codArticulo NUMBER,
    cantidad NUMBER DEFAULT 0,
    precio NUMBER NOT NULL,
    fecha DATE NOT NULL,
    PRIMARY KEY (codProv, codHotel, codArticulo, fecha),
    FOREIGN KEY (codProv, codArticulo) REFERENCES Tiene(codProv, codArticulo),
    CHECK (cantidad >= 0),
    CHECK (precio > 0)
);
```

4.2. Concesión de privilegios

Empleamos el token `GRANT` para conceder los privilegios:

```
---------------------------------
-- Permisos para tabla Articulo
---------------------------------
GRANT SELECT, INSERT, UPDATE, DELETE ON Articulo TO kartoffeln2, kartoffeln3, kartoffeln4;

---------------------------------
-- Permisos para tabla Cliente
---------------------------------
GRANT SELECT, INSERT, UPDATE, DELETE ON Cliente TO kartoffeln2, kartoffeln3, kartoffeln4;

---------------------------------
-- Permisos para tabla Hotel
---------------------------------
GRANT SELECT, INSERT, UPDATE, DELETE ON Hotel TO kartoffeln2, kartoffeln3, kartoffeln4;

---------------------------------
-- Permisos para tabla Empleado
---------------------------------
GRANT SELECT, INSERT, UPDATE, DELETE ON Empleado TO kartoffeln2, kartoffeln3, kartoffeln4;

---------------------------------
-- Permisos para tabla Contrata
---------------------------------
GRANT SELECT, INSERT, UPDATE, DELETE ON Contrata TO kartoffeln2, kartoffeln3, kartoffeln4;

---------------------------------
-- Permisos para tabla Reserva
---------------------------------
GRANT SELECT, INSERT, UPDATE, DELETE ON Reserva TO kartoffeln2, kartoffeln3, kartoffeln4;

```

```
31 --------------------------------
32 -- Permisos para tabla Proveedor
33 --------------------------------
34 GRANT SELECT, INSERT, UPDATE, DELETE ON Proveedor TO kartoffeln2, kartoffeln3, kartoffeln4;
35
36 --------------------------------
37 -- Permisos para tabla Tiene
38 --------------------------------
39 GRANT SELECT, INSERT, UPDATE, DELETE ON Tiene TO kartoffeln2, kartoffeln3, kartoffeln4;
40
41 --------------------------------
42 -- Permisos para tabla Suministra
43 --------------------------------
44 GRANT SELECT, INSERT, UPDATE, DELETE ON Suministra TO kartoffeln2, kartoffeln3, kartoffeln4;
```

4.3. Creación de vistas

Para las vistas realizaremos la unión de todas las tablas con el mismo nombre de cada localidad salvo las dos que están replicadas totalmente, ya que no tendría sentido hacer la unión de datos que son iguales a todos.

Vista Empleado

```
1 CREATE OR REPLACE VIEW EmpleadoView AS
2 SELECT * FROM Empleado
3 UNION
4 SELECT * FROM kartoffeln2.Empleado
5 UNION
6 SELECT * FROM kartoffeln3.Empleado
7 UNION
8 SELECT * FROM kartoffeln4.Empleado;
```

Vista Hotel

```
1 CREATE OR REPLACE VIEW HotelView AS
2 SELECT * FROM Hotel
3 UNION
4 SELECT * FROM kartoffeln2.Hotel
5 UNION
6 SELECT * FROM kartoffeln3.Hotel
7 UNION
8 SELECT * FROM kartoffeln4.Hotel;
```

Vista Contrata

```
1 CREATE OR REPLACE VIEW ContrataView AS
2 SELECT * FROM Contrata
3 UNION
4 SELECT * FROM kartoffeln2.Contrata
5 UNION
6 SELECT * FROM kartoffeln3.Contrata
7 UNION
8 SELECT * FROM kartoffeln4.Contrata;
```

Vista Cliente

```
CREATE OR REPLACE VIEW ClienteView AS
SELECT * FROM Cliente;
```

Vista Reserva

```
CREATE OR REPLACE VIEW ReservaView AS
SELECT * FROM Reserva
UNION
SELECT * FROM kartoffeln2.Reserva
UNION
SELECT * FROM kartoffeln3.Reserva
UNION
SELECT * FROM kartoffeln4.Reserva;
```

Vista Articulo

```
CREATE OR REPLACE VIEW ArticuloView AS
SELECT * FROM Articulo;
```

Vista Proveedor

```
CREATE OR REPLACE VIEW ProveedorView AS
SELECT * FROM Proveedor
UNION
SELECT * FROM kartoffeln3.Proveedor;
```

Vista Tiene

```
CREATE OR REPLACE VIEW TieneView AS
SELECT * FROM Tiene
UNION
SELECT * FROM kartoffeln3.Tiene;
```

Vista Suministra

```
CREATE OR REPLACE VIEW SuministraView AS
SELECT * FROM Suministra
UNION
SELECT * FROM kartoffeln3.Suministra;
```

4.4. Disparadores

En esta subsección mostraremos los disparadores que hemos empleado para tener en cuenta las demás restricciones de integridad.

Restricciones de Integridad 1, 2, 3

Respecto a las 3 primeras restricciones que se indica en el enunciado, quedan implementadas de por sí con la creación de las tablas, salvo las siguientes:

```
-- Restricción de Integridad 2 - Integridad Referencial
CREATE OR REPLACE TRIGGER check_codHotel_in_HotelView
BEFORE INSERT OR UPDATE ON Suministra
FOR EACH ROW
DECLARE
    v_count NUMBER;
BEGIN
    SELECT COUNT(*)
    INTO v_count
    FROM HotelView
    WHERE codHotel = :NEW.codHotel;

    IF v_count = 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'codHotel does not exist in HotelView');
    END IF;
END;
```

```

CREATE OR REPLACE TRIGGER check_codEmpleado_in_ContrataView
BEFORE INSERT OR UPDATE ON Contrata
FOR EACH ROW
DECLARE
    PRAGMA AUTONOMOUS_TRANSACTION;
    v_count NUMBER;
BEGIN
    -- Verificar si el codEmpleado ya existe en la vista ContrataView solo en caso de UPDATE
    IF UPDATING THEN
        SELECT COUNT(*)
        INTO v_count
        FROM ContrataView
        WHERE codEmpleado = :NEW.codEmpleado;

        IF v_count = 0 THEN
            RAISE_APPLICATION_ERROR(-20101, 'codEmpleado does not exist in ContrataView');
        END IF;
    END IF;
END;
```

```
CREATE OR REPLACE TRIGGER TRIGGER check_director
BEFORE INSERT OR UPDATE ON Hotel
FOR EACH ROW
DECLARE
    v_count NUMBER;
BEGIN
    IF :NEW.director IS NOT NULL THEN
        SELECT COUNT(*)
        INTO v_count
        FROM EmpleadoView
        WHERE codEmpleado = :NEW.director;

        IF v_count = 0 THEN
            RAISE_APPLICATION_ERROR(-20001, 'Este director no está registrado en los
            Empleados');
```

```
        END IF;
    END IF;
END;
```

Restricciones de Integridad 4, 5, 6

```
-- Restricción de Integridad 4 - Capacidad del Hotel
-- Restricción de Integridad 5 - Fecha de entrada de cliente no puede ser posterior
-- a la de salida
-- Restricción de Integridad 6 - Un cliente no puede hacer una reserva en distintos
-- hoteles para la misma fecha
CREATE OR REPLACE TRIGGER comprobar_reserva
BEFORE INSERT OR UPDATE ON Reserva
FOR EACH ROW
DECLARE
    PRAGMA AUTONOMOUS_TRANSACTION;
    sencillasReservadas INTEGER;
    doblesReservadas INTEGER;
    numSencillasDisponibles INTEGER;
    numDoblesDisponibles INTEGER;
    conflictCount INTEGER;
BEGIN
    -- Contar las habitaciones sencillas ya reservadas en el rango de fechas
    SELECT COUNT(*)
    INTO sencillasReservadas
    FROM ReservaView r
    WHERE r.codHotel = :NEW.codHotel
        AND r.tipoHab = 'Sencilla'
        AND r.fechaEntrada < :NEW.fechaSalida
        AND r.fechaSalida > :NEW.fechaEntrada;

    -- Contar las habitaciones dobles ya reservadas en el rango de fechas
    SELECT COUNT(*)
    INTO doblesReservadas
    FROM ReservaView r
    WHERE r.codHotel = :NEW.codHotel
        AND r.tipoHab = 'Doble'
        AND r.fechaEntrada < :NEW.fechaSalida
        AND r.fechaSalida > :NEW.fechaEntrada;

    -- Obtener la capacidad total de habitaciones sencillas y dobles
    SELECT numHabSencillas, numHabDobles
    INTO numSencillasDisponibles, numDoblesDisponibles
    FROM HotelView h
    WHERE h.codHotel = :NEW.codHotel;

    -- Obtener el número de reservas que tenga el cliente en cualquier hotel
    SELECT COUNT(*) INTO conflictCount
    FROM ReservaView r
    WHERE r.codCliente = :NEW.codCliente AND
        ((:NEW.fechaEntrada BETWEEN r.fechaEntrada AND r.fechaSalida) OR
        (:NEW.fechaSalida BETWEEN r.fechaEntrada AND r.fechaSalida)  OR
        (r.fechaEntrada BETWEEN :NEW.fechaEntrada AND :NEW.fechaSalida) OR
        (r.fechaSalida BETWEEN :NEW.fechaEntrada AND :NEW.fechaSalida)) AND
```

```
        r.codHotel <> :NEW.codHotel;

    -- Verificar si excede la capacidad para habitaciones sencillas
    IF :NEW.tipoHab = 'Sencilla' AND sencillasReservadas >= numSencillasDisponibles THEN
        RAISE_APPLICATION_ERROR(-20001, 'No hay habitaciones sencillas
        disponibles en el hotel.');
    END IF;

    -- Verificar si excede la capacidad para habitaciones dobles
    IF :NEW.tipoHab = 'Doble' AND doblesReservadas >= numDoblesDisponibles THEN
        RAISE_APPLICATION_ERROR(-20002, 'No hay habitaciones dobles
        disponibles en el hotel.');
    END IF;

    -- Verificar si excede la capacidad para habitaciones dobles
    IF :NEW.fechaEntrada > :NEW.fechaSalida THEN
        RAISE_APPLICATION_ERROR(-20003, 'La fecha de entrada debe
        ser anterior a la de salida.');
    END IF;

    -- Verificar que no tenga una reserva en esa fecha en otro hotel
    IF conflictCount > 0 THEN
        RAISE_APPLICATION_ERROR(-20004, 'El cliente ya tiene una reserva
         en otro hotel para las mismas fechas.');
    END IF;

    -- Verificar que el tipo de habitación sea válido
    IF :NEW.tipoHab NOT IN ('Sencilla', 'Doble') THEN
        RAISE_APPLICATION_ERROR(-20015, 'El tipo de habitación no es válido.');
    END IF;
END;
```

Restricciones de Integridad 7, 8, 9

Vienen definidas durante la creación de las tablas.

Restricción de Integridad 10

```
CREATE OR REPLACE TRIGGER prevenir_reduccion_salario
BEFORE UPDATE ON Empleado
FOR EACH ROW
DECLARE
    PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    IF :NEW.salario < :OLD.salario THEN
            RAISE_APPLICATION_ERROR(-20005, 'El salario de un empleado no
            puede disminuir.');
    END IF;
END;
```

Restricción de Integridad 11, 12

```
-- Restricción de Integridad 11: Fecha de inicio de un empleado debe ser igual o
-- posterior que la de inicio de contrato
-- Restricción de Integridad 12: La fecha de inicio de un empleado será igual o
-- posterior a la de fin en el hotel en el que estaba asignado anteriormente
CREATE OR REPLACE TRIGGER validar_baja_contrato
BEFORE INSERT OR UPDATE ON Contrata
FOR EACH ROW
BEGIN
    IF :NEW.fechaFin < :NEW.fechaInicio THEN
        RAISE_APPLICATION_ERROR(-20109, 'La fecha de fin de trabajo no puede ser anterior a la
        ↪  fecha de inicio.');
    END IF;
END;

CREATE OR REPLACE TRIGGER validar_fechas_empleado
BEFORE INSERT OR UPDATE ON Contrata
FOR EACH ROW
DECLARE
    comienzoContrato DATE;
BEGIN
    SELECT fechaContrato INTO comienzoContrato
    FROM EmpleadoView e
    WHERE e.codEmpleado = :NEW.codEmpleado;

    IF :NEW.fechaInicio < comienzoContrato THEN
        RAISE_APPLICATION_ERROR(-20006, 'La fecha de inicio en el hotel debe ser igual o
        ↪  posterior a la del contrato.');
    END IF;
END;
```

Restricción de Integridad 13

Viene dada en la creación de la tabla.

Restricción de Integridad 14

```
-- Restricción de Integridad 14: El precio por unidad de un artículo para un suministro
-- determinado a un hotel determinado, nunca podrá ser menor que el de ese mismo artículo
-- en suministros anteriores a ese mismo hotel.
CREATE OR REPLACE TRIGGER validar_precio_suministro
BEFORE INSERT OR UPDATE ON Suministra FOR EACH ROW
DECLARE
PRAGMA AUTONOMOUS_TRANSACTION;
precioAnterior NUMBER;
BEGIN
    SELECT MAX(precio) INTO precioAnterior
    FROM Suministra
    WHERE codArticulo = :NEW.codArticulo AND codHotel = :NEW.codHotel;

    IF precioAnterior IS NOT NULL AND :NEW.precio < precioAnterior THEN
        RAISE_APPLICATION_ERROR(-20007, 'El precio no puede ser menor que
        suministros anteriores.');
    END IF;
END;
```

Restricción de Integridad 15

```
-- Restricción de Integridad 15: Un artículo solo puede ser suministrado como mucho por
-- dos proveedores
CREATE OR REPLACE TRIGGER validar_supledores_articulos
FOR INSERT OR UPDATE ON Tiene
COMPOUND TRIGGER

    TYPE t_articles IS TABLE OF NUMBER INDEX BY PLS_INTEGER;
    affected_articles t_articles;

BEFORE EACH ROW IS
BEGIN
    IF NOT affected_articles.EXISTS(:NEW.codArticulo) THEN
        affected_articles(affected_articles.COUNT + 1) := :NEW.codArticulo;
    END IF;
END BEFORE EACH ROW;

AFTER STATEMENT IS
    v_count NUMBER;
BEGIN
    FOR i IN 1..affected_articles.COUNT LOOP
        SELECT COUNT(*) INTO v_count
        FROM Tiene
        WHERE codArticulo = affected_articles(i);

        IF v_count > 2 THEN
            RAISE_APPLICATION_ERROR(-20001,
            'Error: Un artículo no puede tener más de 2 proveedores.');
        END IF;
    END LOOP;
END AFTER STATEMENT;

END validar_supledores_articulos;
```

Restricción de Integridad 16

Viene dada durante la creación de la tabla.

Restricciones de Integridad 17, 18

```
-- Restricción de Integridad 17: Ningún hotel de las provincias de Granada, Jaén, Málaga o
-- Almería podrán solicitar artículos a proveedores de Sevilla.
-- Restricción de Integridad 18: Ningún hotel de las provincias de Córdoba, Sevilla, Cádiz o
-- Huelva podrán solicitar artículos a proveedores de Granada.
CREATE OR REPLACE TRIGGER validar_provincias_suministros
BEFORE INSERT OR UPDATE ON Suministra FOR EACH ROW
DECLARE
provinciaHotel VARCHAR2(50);
provinciaProveedor VARCHAR2(50);
BEGIN
    SELECT provincia INTO provinciaHotel FROM HotelView WHERE codHotel = :NEW.codHotel;
```

```
    SELECT ciudad INTO provinciaProveedor FROM ProveedorView WHERE codProv = :NEW.codProv;

    IF (provinciaHotel IN ('Granada', 'Jaén', 'Málaga', 'Almería') AND
        provinciaProveedor = 'Sevilla') OR
       (provinciaHotel IN ('Córdoba', 'Sevilla', 'Cádiz', 'Huelva') AND
        provinciaProveedor = 'Granada') THEN
            RAISE_APPLICATION_ERROR(-20009, 'No se permite el suministro entre hotel
            con provincia en ' || provinciaHotel || ' y proveedor con ciudad en '
            || provinciaProveedor);
    END IF;
END;
```

Restricción de Integridad 19

```
-- Restricción de Integridad 19: Los datos referentes a un proveedor solamente
-- podrán eliminarse de la base de datos si, para cada artículo que suministre,
-- la cantidad total suministrada es 0, o no existe ningún suministro.
CREATE OR REPLACE TRIGGER prevencion_borrado_proveedor
BEFORE DELETE ON Proveedor FOR EACH ROW
DECLARE
    suministrosTotal NUMBER;
BEGIN
    SELECT SUM(cantidad) INTO suministrosTotal
    FROM Suministra
    WHERE codProv = :OLD.codProv;

    IF suministrosTotal > 0 THEN
        RAISE_APPLICATION_ERROR(-20010, 'No se puede eliminar el
        proveedor mientras tenga suministros activos.');
    END IF;
END;
```

Restricción de Integridad 20

```
-- Restricción de Integridad 19: Los datos referentes a un artículo, sólo podrán
-- eliminarse de la base de datos, si la cantidad total suministrada de ese artículo
-- es 0, o no existe ningún suministro.
CREATE OR REPLACE TRIGGER prevencion_borrado_articulo
BEFORE DELETE ON Articulo FOR EACH ROW
DECLARE
    totalSuministrados NUMBER;
BEGIN
    SELECT SUM(s.cantidad) INTO totalSuministrados
    FROM SuministraView s
    WHERE s.codArticulo = :OLD.codArticulo;

    IF totalSuministrados > 0 THEN
        RAISE_APPLICATION_ERROR(-20011, 'No se puede eliminar el artículo
         mientras tenga suministros activos.');
    END IF;
END;
```

5. Implementación de actualizaciones

En esta sección quedan alojadas las actualizaciones a realizar mediante el uso de los procedimientos. A continuación mostraremos las actualizaciones llevadas a cabo para la primera localidad.

Actualización 1

```
-- Actualización 1: Dar de alta a un empleado
CREATE OR REPLACE PROCEDURE alta_empleado(
    p_codEmpleado NUMBER,
    p_DNI VARCHAR2,
    p_nombre VARCHAR2,
    p_direccion VARCHAR2,
    p_telefono VARCHAR2,
    p_fechaContrato DATE,
    p_salario NUMBER,
    p_codHotel NUMBER,
    p_fechaInicio DATE)
IS
    contador NUMBER;
    prov Hotel.provincia%TYPE;
BEGIN
    SELECT COUNT (*) INTO contador
    FROM EmpleadoView e WHERE e.codEmpleado = p_codEmpleado;

    IF(contador <> 0) THEN
        RAISE_APPLICATION_ERROR(-20030, 'El empleado no puede ser
        insertado porque ya existe');

    ELSE
        SELECT provincia INTO prov
        FROM HotelView h WHERE h.codHotel = p_codHotel;

        IF (prov = 'Granada' OR prov = 'Jaén') THEN
            INSERT INTO  kartoffeln1.Empleado (codEmpleado, DNI, nombre, direccion,
            telefono, fechaContrato, salario)
            VALUES (p_codEmpleado, p_DNI, p_nombre, p_direccion, p_telefono,
            p_fechaContrato, p_salario);

            INSERT INTO  kartoffeln1.Contrata (codHotel, codEmpleado, fechaInicio)
            VALUES (p_codHotel, p_codEmpleado, p_fechaInicio);

        ELSIF (prov = 'Málaga' OR prov = 'Almería') THEN
            INSERT INTO  kartoffeln2.Empleado (codEmpleado, DNI, nombre, direccion,
             telefono, fechaContrato, salario)
            VALUES (p_codEmpleado, p_DNI, p_nombre, p_direccion, p_telefono,
            p_fechaContrato, p_salario);

            INSERT INTO  kartoffeln2.Contrata (codHotel, codEmpleado, fechaInicio)
            VALUES (p_codHotel, p_codEmpleado, p_fechaInicio);

        ELSIF (prov = 'Sevilla' OR prov = 'Córdoba') THEN
            INSERT INTO  kartoffeln3.Empleado (codEmpleado, DNI, nombre, direccion,
```

```
            telefono, fechaContrato, salario)
            VALUES (p_codEmpleado, p_DNI, p_nombre, p_direccion, p_telefono,
            p_fechaContrato, p_salario);

            INSERT INTO  kartoffeln3.Contrata (codHotel, codEmpleado, fechaInicio)
            VALUES (p_codHotel, p_codEmpleado, p_fechaInicio);

        ELSIF (prov = 'Cádiz' OR prov = 'Huelva') THEN
            INSERT INTO  kartoffeln4.Empleado (codEmpleado, DNI, nombre, direccion,
            telefono, fechaContrato, salario)
            VALUES (p_codEmpleado, p_DNI, p_nombre, p_direccion, p_telefono,
            p_fechaContrato, p_salario);

            INSERT INTO  kartoffeln4.Contrata (codHotel, codEmpleado, fechaInicio)
            VALUES (p_codHotel, p_codEmpleado, p_fechaInicio);


        ELSE
            RAISE_APPLICATION_ERROR(-20031, 'La provincia del empleado no corresponde
            a ningún fragmento válido.');
        END IF;

        COMMIT;
        DBMS_OUTPUT.PUT_LINE('El empleado con código ' || p_codEmpleado ||
        ' y nombre ' || p_nombre || ' fue dado de alta correctamente.');
    END IF;
END;
```

Actualización 2

```
-- Actualización 2: Dar de baja a un empleado
CREATE OR REPLACE PROCEDURE baja_empleado(
    p_codEmpleado NUMBER,
    p_fechaBaja DATE)
IS
    contador NUMBER;
    esDirector NUMBER;
    prov Hotel.provincia%TYPE;
    provHotel Hotel.provincia%TYPE;
BEGIN
    SELECT COUNT (*) INTO contador
    FROM EmpleadoView e WHERE e.codEmpleado = p_codEmpleado;

    IF(contador = 0) THEN
        RAISE_APPLICATION_ERROR(-20032, 'No existe ningún empleado con el código ' ||
        p_codEmpleado || ' .');

    ELSE
        SELECT provincia INTO prov
        FROM HotelView h
        JOIN ContrataView c ON h.codHotel = c.codHotel
        WHERE c.codEmpleado = p_codEmpleado AND c.fechaFin IS NULL;
        DBMS_OUTPUT.PUT_LINE('El empleado con código ' || p_codEmpleado || ' es de la
        provincia: ' || prov);
```

```
23
24          -- Verificar si el empleado es director de algún hotel
25          SELECT COUNT(*) INTO esDirector
26          FROM HotelView h
27          WHERE h.director = p_codEmpleado;
28
29          -- Si es director, actualizar el atributo director a NULL
30          IF esDirector > 0 THEN
31              SELECT provincia INTO provHotel
32              FROM HotelView h
33              WHERE h.director = p_codEmpleado;
34
35              CASE
36                  WHEN provHotel IN ('Granada', 'Jaén') THEN
37                      UPDATE kartoffeln1.Hotel
38                      SET director = NULL
39                      WHERE director = p_codEmpleado;
40                  WHEN provHotel IN ('Málaga', 'Almería') THEN
41                      UPDATE kartoffeln2.Hotel
42                      SET director = NULL
43                      WHERE director = p_codEmpleado;
44                  WHEN provHotel IN ('Sevilla', 'Córdoba') THEN
45                      UPDATE kartoffeln3.Hotel
46                      SET director = NULL
47                      WHERE director = p_codEmpleado;
48                  WHEN provHotel IN ('Cádiz', 'Huelva') THEN
49                      UPDATE kartoffeln4.Hotel
50                      SET director = NULL
51                      WHERE director = p_codEmpleado;
52              END CASE;
53          END IF;
54
55          -- Caso: Fragmento 1 (Granada, Jaén)
56          IF prov IN ('Granada', 'Jaén') THEN
57              UPDATE kartoffeln1.Contrata
58              SET fechaFin = p_fechaBaja
59              WHERE codEmpleado = p_codEmpleado AND fechaFin IS NULL;
60
61              DELETE FROM kartoffeln1.Empleado WHERE codEmpleado = p_codEmpleado;
62
63          -- Caso: Fragmento 2 (Málaga, Almería)
64          ELSIF prov IN ('Málaga', 'Almería') THEN
65              UPDATE kartoffeln2.Contrata
66              SET fechaFin = p_fechaBaja
67              WHERE codEmpleado = p_codEmpleado AND fechaFin IS NULL;
68
69              DELETE FROM kartoffeln2.Empleado WHERE codEmpleado = p_codEmpleado;
70
71          -- Caso: Fragmento 3 (Sevilla, Córdoba)
72          ELSIF prov IN ('Sevilla', 'Córdoba') THEN
73              UPDATE kartoffeln3.Contrata
74              SET fechaFin = p_fechaBaja
75              WHERE codEmpleado = p_codEmpleado AND fechaFin IS NULL;
76
```

```
77              DELETE FROM kartoffeln3.Empleado WHERE codEmpleado = p_codEmpleado;
78
79          -- Caso: Fragmento 4 (Cádiz, Huelva)
80          ELSIF prov IN ('Cádiz', 'Huelva') THEN
81              UPDATE kartoffeln4.Contrata
82              SET fechaFin = p_fechaBaja
83              WHERE codEmpleado = p_codEmpleado AND fechaFin IS NULL;
84
85              DELETE FROM kartoffeln4.Empleado WHERE codEmpleado = p_codEmpleado;
86
87              ELSE
88              RAISE_APPLICATION_ERROR(-20033, 'La provincia del empleado no corresponde a ningún
                fragmento válido.');
89          END IF;
90
91          COMMIT;
92          -- Mensaje de confirmación
93          DBMS_OUTPUT.PUT_LINE('El empleado con código ' || p_codEmpleado || ' fue dado de baja
            correctamente.');
94      END IF;
95  END;
```

Actualización 3

```
1   -- Actualización 3: Modificar salario de un empleado
2   CREATE OR REPLACE PROCEDURE modificar_salario(
3       p_codEmpleado NUMBER,
4       p_nuevoSalario NUMBER)
5   IS
6       contador NUMBER;
7       prov Hotel.provincia%TYPE;
8   BEGIN
9       SELECT COUNT (*) INTO contador
10      FROM EmpleadoView e WHERE e.codEmpleado = p_codEmpleado;
11
12      IF(contador = 0) THEN
13          RAISE_APPLICATION_ERROR(-20034, 'No existe ningún empleado con el código ' ||
            p_codEmpleado || ' .');
14
15      ELSE
16          SELECT h.provincia INTO prov
17          FROM HotelView h
18          JOIN ContrataView c ON h.codHotel = c.codHotel
19          WHERE c.codEmpleado = p_codEmpleado AND c.fechaFin IS NULL;
20
21          CASE
22              WHEN prov IN ('Granada', 'Jaén') THEN
23                  UPDATE kartoffeln1.Empleado
24                  SET salario = p_nuevoSalario
25                  WHERE codEmpleado = p_codEmpleado;
26
27              WHEN prov IN ('Málaga', 'Almería') THEN
28                  UPDATE kartoffeln2.Empleado
29                  SET salario = p_nuevoSalario
```

```
                WHERE codEmpleado = p_codEmpleado;

            WHEN prov IN ('Sevilla', 'Córdoba') THEN
                UPDATE kartoffeln3.Empleado
                SET salario = p_nuevoSalario
                WHERE codEmpleado = p_codEmpleado;

            WHEN prov IN ('Cádiz', 'Huelva') THEN
                UPDATE kartoffeln4.Empleado
                SET salario = p_nuevoSalario
                WHERE codEmpleado = p_codEmpleado;

            ELSE
                RAISE_APPLICATION_ERROR(-20035, 'La provincia del empleado no corresponde a
                ningún fragmento válido.');
        END CASE;

        COMMIT;
        DBMS_OUTPUT.PUT_LINE('El salario del empleado con código ' || p_codEmpleado || ' ha
        sido modificado a ' || p_nuevoSalario || ' .');
    END IF;
END;
```

Actualización 4

```
-- Actualización 4: Trasladar un empleado
CREATE OR REPLACE PROCEDURE trasladar_empleado (
    p_codEmpleado NUMBER,
    p_fechaFinActual DATE,
    p_codHotel NUMBER,
    p_fechaInicioNuevo DATE,
    p_nuevaDireccion VARCHAR2 DEFAULT NULL,
    p_nuevoTelefono VARCHAR2 DEFAULT NULL)
IS
    contador NUMBER;
    contadorHotel NUMBER;
    prov Hotel.provincia%TYPE;
    provNuevoHotel Hotel.provincia%TYPE;
BEGIN
    IF p_fechaInicioNuevo < p_fechaFinActual THEN
        RAISE_APPLICATION_ERROR(-20035, 'La fecha de fin actual no puede ser menor o igual a la
        nueva fecha de inicio.');
    END IF;

    SELECT COUNT (*) INTO contador
    FROM EmpleadoView e WHERE e.codEmpleado = p_codEmpleado;

    IF(contador = 0) THEN
        RAISE_APPLICATION_ERROR(-20036, 'No existe ningún empleado con el código ' ||
        p_codEmpleado || ' .');
    END IF;

    SELECT COUNT (*) INTO contadorHotel
    FROM HotelView h WHERE h.codhotel = p_codHotel;
```

```
28
29      IF(contadorHotel = 0) THEN
30          RAISE_APPLICATION_ERROR(-20100, 'No existe ningún hotel con el código ' || p_codHotel
            || ' .');
31
32      ELSE
33          -- Determinar la provincia del hotel actual del empleado
34          SELECT h.provincia INTO prov
35          FROM HotelView h
36          JOIN ContrataView c ON h.codHotel = c.codHotel
37          WHERE c.codEmpleado = p_codEmpleado AND c.fechaFin IS NULL;
38
39          -- Determinar la provincia del nuevo hotel
40          SELECT h.provincia INTO provNuevoHotel
41          FROM HotelView h
42          WHERE h.codHotel = p_codHotel;
43
44          -- Registrar la finalización en el hotel actual según el fragmento
45          CASE
46              WHEN prov IN ('Granada', 'Jaén') THEN
47                  UPDATE kartoffeln1.Contrata
48                  SET fechaFin = p_fechaFinActual
49                  WHERE codEmpleado = p_codEmpleado AND fechaFin IS NULL;
50
51              WHEN prov IN ('Málaga', 'Almería') THEN
52                  UPDATE kartoffeln2.Contrata
53                  SET fechaFin = p_fechaFinActual
54                  WHERE codEmpleado = p_codEmpleado AND fechaFin IS NULL;
55
56              WHEN prov IN ('Sevilla', 'Córdoba') THEN
57                  UPDATE kartoffeln3.Contrata
58                  SET fechaFin = p_fechaFinActual
59                  WHERE codEmpleado = p_codEmpleado AND fechaFin IS NULL;
60
61              WHEN prov IN ('Cádiz', 'Huelva') THEN
62                  UPDATE kartoffeln4.Contrata
63                  SET fechaFin = p_fechaFinActual
64                  WHERE codEmpleado = p_codEmpleado AND fechaFin IS NULL;
65
66              ELSE
67                  RAISE_APPLICATION_ERROR(-20037, 'La provincia del hotel actual no corresponde a
                    ningún fragmento válido.');
68          END CASE;
69
70          -- Crear el nuevo contrato en el hotel al que es trasladado según el fragmento
71          CASE
72              WHEN provNuevoHotel IN ('Granada', 'Jaén') THEN
73                  INSERT INTO kartoffeln1.Contrata (codHotel, codEmpleado, fechaInicio)
74                  VALUES (p_codHotel, p_codEmpleado, p_fechaInicioNuevo);
75
76                  IF (prov != provNuevoHotel) THEN
77                      INSERT INTO kartoffeln1.Empleado (codEmpleado, DNI, nombre, direccion,
                        telefono, fechaContrato, salario)
78                      SELECT codEmpleado, DNI, nombre, direccion, telefono, fechaContrato,
                        salario
```

```
79                      FROM EmpleadoView e
80                      WHERE e.codEmpleado = p_codEmpleado;
81
82                      CASE
83                          WHEN prov IN ('Málaga', 'Almería') THEN
84                              DELETE FROM kartoffeln2.Empleado e WHERE e.codEmpleado =
                                p_codEmpleado;
85                          WHEN prov IN ('Sevilla', 'Córdoba') THEN
86                              DELETE FROM kartoffeln3.Empleado e WHERE e.codEmpleado =
                                p_codEmpleado;
87                          WHEN prov IN ('Cádiz', 'Huelva') THEN
88                              DELETE FROM kartoffeln4.Empleado e WHERE e.codEmpleado =
                                p_codEmpleado;
89                      END CASE;
90                  END IF;
91
92              WHEN provNuevoHotel IN ('Málaga', 'Almería') THEN
93                  INSERT INTO kartoffeln2.Contrata (codHotel, codEmpleado, fechaInicio)
94                  VALUES (p_codHotel, p_codEmpleado, p_fechaInicioNuevo);
95
96                  IF (prov != provNuevoHotel) THEN
97                      INSERT INTO kartoffeln2.Empleado (codEmpleado, DNI, nombre, direccion,
                        telefono, fechaContrato, salario)
98                      SELECT codEmpleado, DNI, nombre, direccion, telefono, fechaContrato,
                        salario
99                      FROM EmpleadoView e
100                     WHERE e.codEmpleado = p_codEmpleado;
101
102                     CASE
103                         WHEN prov IN ('Granada', 'Jaén') THEN
104                             DELETE FROM kartoffeln1.Empleado e WHERE e.codEmpleado =
                                p_codEmpleado;
105                         WHEN prov IN ('Sevilla', 'Córdoba') THEN
106                             DELETE FROM kartoffeln3.Empleado e WHERE e.codEmpleado =
                                p_codEmpleado;
107                         WHEN prov IN ('Cádiz', 'Huelva') THEN
108                             DELETE FROM kartoffeln4.Empleado e WHERE e.codEmpleado =
                                p_codEmpleado;
109                     END CASE;
110                 END IF;
111
112             WHEN provNuevoHotel IN ('Sevilla', 'Córdoba') THEN
113                 INSERT INTO kartoffeln3.Contrata (codHotel, codEmpleado, fechaInicio)
114                 VALUES (p_codHotel, p_codEmpleado, p_fechaInicioNuevo);
115
116                 IF (prov != provNuevoHotel) THEN
117                     INSERT INTO kartoffeln3.Empleado (codEmpleado, DNI, nombre, direccion,
                        telefono, fechaContrato, salario)
118                     SELECT codEmpleado, DNI, nombre, direccion, telefono, fechaContrato,
                        salario
119                     FROM EmpleadoView e
120                     WHERE e.codEmpleado = p_codEmpleado;
121
122                     CASE
```

```
123                         WHEN prov IN ('Granada', 'Jaén') THEN
124                             DELETE FROM kartoffeln1.Empleado e WHERE e.codEmpleado =
                                p_codEmpleado;
125                         WHEN prov IN ('Málaga', 'Almería') THEN
126                             DELETE FROM kartoffeln2.Empleado e WHERE e.codEmpleado =
                                p_codEmpleado;
127                         WHEN prov IN ('Cádiz', 'Huelva') THEN
128                             DELETE FROM kartoffeln4.Empleado e WHERE e.codEmpleado =
                                p_codEmpleado;
129                     END CASE;
130                 END IF;
131
132             WHEN provNuevoHotel IN ('Cádiz', 'Huelva') THEN
133                 INSERT INTO kartoffeln4.Contrata (codHotel, codEmpleado, fechaInicio)
134                 VALUES (p_codHotel, p_codEmpleado, p_fechaInicioNuevo);
135
136                 IF (prov != provNuevoHotel) THEN
137                     INSERT INTO kartoffeln4.Empleado (codEmpleado, DNI, nombre, direccion,
                        telefono, fechaContrato, salario)
138                     SELECT codEmpleado, DNI, nombre, direccion, telefono, fechaContrato,
                        salario
139                     FROM EmpleadoView e
140                     WHERE e.codEmpleado = p_codEmpleado;
141
142                     CASE
143                         WHEN prov IN ('Málaga', 'Almería') THEN
144                             DELETE FROM kartoffeln2.Empleado e WHERE e.codEmpleado =
                                p_codEmpleado;
145                         WHEN prov IN ('Sevilla', 'Córdoba') THEN
146                             DELETE FROM kartoffeln3.Empleado e WHERE e.codEmpleado =
                                p_codEmpleado;
147                         WHEN prov IN ('Granada', 'Jaén') THEN
148                             DELETE FROM kartoffeln1.Empleado e WHERE e.codEmpleado =
                                p_codEmpleado;
149                     END CASE;
150                 END IF;
151
152             ELSE
153                 RAISE_APPLICATION_ERROR(-20038, 'La provincia del nuevo hotel no corresponde a
                    ningún fragmento válido.');
154             END CASE;
155
156             -- Actualizar los datos de dirección y teléfono del empleado según el fragmento
157             IF p_nuevaDireccion IS NOT NULL OR p_nuevoTelefono IS NOT NULL THEN
158                 CASE
159                     WHEN provNuevoHotel IN ('Granada', 'Jaén') THEN
160                         UPDATE kartoffeln1.Empleado
161                         SET direccion = NVL(p_nuevaDireccion, direccion),
162                         telefono = NVL(p_nuevoTelefono, telefono)
163                         WHERE codEmpleado = p_codEmpleado;
164
165                     WHEN provNuevoHotel IN ('Málaga', 'Almería') THEN
166                         UPDATE kartoffeln2.Empleado
167                         SET direccion = NVL(p_nuevaDireccion, direccion),
```

```
                        telefono = NVL(p_nuevoTelefono, telefono)
                        WHERE codEmpleado = p_codEmpleado;

                    WHEN provNuevoHotel IN ('Sevilla', 'Córdoba') THEN
                        UPDATE kartoffeln3.Empleado
                        SET direccion = NVL(p_nuevaDireccion, direccion),
                        telefono = NVL(p_nuevoTelefono, telefono)
                        WHERE codEmpleado = p_codEmpleado;

                    WHEN provNuevoHotel IN ('Cádiz', 'Huelva') THEN
                        UPDATE kartoffeln4.Empleado
                        SET direccion = NVL(p_nuevaDireccion, direccion),
                        telefono = NVL(p_nuevoTelefono, telefono)
                        WHERE codEmpleado = p_codEmpleado;

                    ELSE
                        RAISE_APPLICATION_ERROR(-20039, 'No se pudo determinar el fragmento del
                        empleado para actualizar sus datos personales.');
                END CASE;
            END IF;

        COMMIT;
        DBMS_OUTPUT.PUT_LINE('El empleado con código ' || p_codEmpleado || ' ha sido trasladado
        al hotel ' || p_codHotel || '.');
    END IF;
END;
```

Actualización 5

```
-- Actualización 5: Dar de alta a un hotel
CREATE OR REPLACE PROCEDURE alta_hotel (
    p_codHotel NUMBER,
    p_nombre VARCHAR2,
    p_ciudad VARCHAR2,
    p_provincia VARCHAR2,
    p_numHabSencillas INTEGER,
    p_numHabDobles INTEGER)
IS
BEGIN
    -- Insertar el nuevo hotel en el fragmento correspondiente según la provincia
    CASE
        WHEN p_provincia IN ('Granada', 'Jaén') THEN
            INSERT INTO kartoffeln1.Hotel (codHotel, nombre, ciudad, provincia,
            numHabSencillas, numHabDobles)
            VALUES (p_codHotel, p_nombre, p_ciudad, p_provincia, p_numHabSencillas,
            p_numHabDobles);

        WHEN p_provincia IN ('Málaga', 'Almería') THEN
            INSERT INTO kartoffeln2.Hotel (codHotel, nombre, ciudad, provincia,
            numHabSencillas, numHabDobles)
            VALUES (p_codHotel, p_nombre, p_ciudad, p_provincia, p_numHabSencillas,
            p_numHabDobles);

        WHEN p_provincia IN ('Sevilla', 'Córdoba') THEN
```

```
            INSERT INTO kartoffeln3.Hotel (codHotel, nombre, ciudad, provincia,
            numHabSencillas, numHabDobles)
            VALUES (p_codHotel, p_nombre, p_ciudad, p_provincia, p_numHabSencillas,
            p_numHabDobles);

        WHEN p_provincia IN ('Cádiz', 'Huelva') THEN
            INSERT INTO kartoffeln4.Hotel (codHotel, nombre, ciudad, provincia,
            numHabSencillas, numHabDobles)
            VALUES (p_codHotel, p_nombre, p_ciudad, p_provincia, p_numHabSencillas,
            p_numHabDobles);

        ELSE
            RAISE_APPLICATION_ERROR(-20040, 'La provincia especificada no corresponde a ningún
            fragmento válido.');
    END CASE;

    COMMIT;
    DBMS_OUTPUT.PUT_LINE('El hotel "' || p_nombre || '" ha sido dado de alta correctamente en
    la provincia ' || p_provincia || '.');
END;
```

Actualización 6

```
-- Actualización 6: Cambiar el director de un hotel
CREATE OR REPLACE PROCEDURE cambiar_director (
    p_codHotel NUMBER,
    p_ID_Director NUMBER)
IS
    prov Hotel.provincia%TYPE;
    esEmpleadoActivo NUMBER;
    esDirector NUMBER;
    provinciaDirector Hotel.provincia%TYPE;
BEGIN
    -- Determinar la provincia del hotel
    SELECT h.provincia INTO prov
    FROM HotelView h
    WHERE h.codHotel = p_codHotel;

    -- Verificar si el nuevo director es un empleado activo en algún fragmento
    SELECT COUNT(*) INTO esEmpleadoActivo
    FROM EmpleadoView e
    WHERE e.codEmpleado = p_ID_Director;

    IF esEmpleadoActivo = 0 THEN
        RAISE_APPLICATION_ERROR(-20041, 'El nuevo director no es un empleado válido.');
    END IF;

    -- Verificar si el empleado es director de algún hotel
    SELECT COUNT(*) INTO esDirector
    FROM HotelView h
    WHERE h.director = p_ID_Director;

    -- Si es director, actualizar el atributo director a NULL
    IF esDirector > 0 THEN
```

```
        SELECT h.provincia INTO provinciaDirector
        FROM HotelView h
        WHERE h.director = p_ID_Director;

        CASE
            WHEN provinciaDirector IN ('Granada', 'Jaén') THEN
                UPDATE kartoffeln1.Hotel
                SET director = NULL
                WHERE director = p_ID_Director;
            WHEN provinciaDirector IN ('Málaga', 'Almería') THEN
                UPDATE kartoffeln2.Hotel
                SET director = NULL
                WHERE director = p_ID_Director;
            WHEN provinciaDirector IN ('Sevilla', 'Córdoba') THEN
                UPDATE kartoffeln3.Hotel
                SET director = NULL
                WHERE director = p_ID_Director;
            WHEN provinciaDirector IN ('Cádiz', 'Huelva') THEN
                UPDATE kartoffeln4.Hotel
                SET director = NULL
                WHERE director = p_ID_Director;
        END CASE;
    END IF;

    -- Actualizar el director del hotel en el fragmento correspondiente
    CASE
        WHEN prov IN ('Granada', 'Jaén') THEN
            UPDATE kartoffeln1.Hotel
            SET director = p_ID_Director
            WHERE codHotel = p_codHotel;

        WHEN prov IN ('Málaga', 'Almería') THEN
            UPDATE kartoffeln2.Hotel
            SET director = p_ID_Director
            WHERE codHotel = p_codHotel;

        WHEN prov IN ('Sevilla', 'Córdoba') THEN
            UPDATE kartoffeln3.Hotel
            SET director = p_ID_Director
            WHERE codHotel = p_codHotel;

        WHEN prov IN ('Cádiz', 'Huelva') THEN
            UPDATE kartoffeln4.Hotel
            SET director = p_ID_Director
            WHERE codHotel = p_codHotel;

        ELSE
            RAISE_APPLICATION_ERROR(-20042, 'La provincia especificada no corresponde a ningún
            fragmento válido.');
    END CASE;

    COMMIT;
    DBMS_OUTPUT.PUT_LINE('El director del hotel con código ' || p_codHotel || ' ha sido
    cambiado correctamente al empleado ' || p_ID_Director || '.');
```

```
END;
```

Actualización 7

```
-- Actualización 7: Dar de alta a un nuevo cliente
CREATE OR REPLACE PROCEDURE alta_cliente (
    p_codCliente NUMBER,
    p_DNI VARCHAR2,
    p_nombre VARCHAR2,
    p_telefono VARCHAR2)
IS
BEGIN
    -- Insertar el cliente en cada fragmento
    BEGIN
        INSERT INTO kartoffeln1.Cliente (codCliente, DNI, nombre, telefono)
        VALUES (p_codCliente, p_DNI, p_nombre, p_telefono);
    EXCEPTION
        WHEN DUP_VAL_ON_INDEX THEN
            RAISE_APPLICATION_ERROR(-20043, 'El cliente ya existe en el fragmento
            kartoffeln1.');
    END;

    BEGIN
        INSERT INTO kartoffeln2.Cliente (codCliente, DNI, nombre, telefono)
        VALUES (p_codCliente, p_DNI, p_nombre, p_telefono);
    EXCEPTION
        WHEN DUP_VAL_ON_INDEX THEN
            RAISE_APPLICATION_ERROR(-20044, 'El cliente ya existe en el fragmento
            kartoffeln2.');
    END;

    BEGIN
        INSERT INTO kartoffeln3.Cliente (codCliente, DNI, nombre, telefono)
        VALUES (p_codCliente, p_DNI, p_nombre, p_telefono);
    EXCEPTION
        WHEN DUP_VAL_ON_INDEX THEN
            RAISE_APPLICATION_ERROR(-20045, 'El cliente ya existe en el fragmento
            kartoffeln3.');
    END;

    BEGIN
        INSERT INTO kartoffeln4.Cliente (codCliente, DNI, nombre, telefono)
        VALUES (p_codCliente, p_DNI, p_nombre, p_telefono);
    EXCEPTION
        WHEN DUP_VAL_ON_INDEX THEN
            RAISE_APPLICATION_ERROR(-20046, 'El cliente ya existe en el fragmento
            kartoffeln4.');
    END;

    COMMIT;
    DBMS_OUTPUT.PUT_LINE('El cliente "' || p_nombre || '" ha sido dado de alta correctamente en
    todas las localidades.');
END;
```

Actualización 8

```
-- Actualización 8: Dar de alta o actualizar una reserva
CREATE OR REPLACE PROCEDURE gestionar_reserva (
    p_codCliente NUMBER,
    p_codHotel NUMBER,
    p_tipoHab VARCHAR2,
    p_fechaEntrada DATE,
    p_fechaSalida DATE,
    p_precio NUMBER
) IS
    prov Hotel.provincia%TYPE;
    clienteExiste NUMBER;
    reservaExiste NUMBER;
BEGIN
    -- Verificar si el cliente existe en al menos un fragmento
    SELECT COUNT(*) INTO clienteExiste
    FROM (
        SELECT codCliente FROM kartoffeln1.Cliente
        UNION ALL
        SELECT codCliente FROM kartoffeln2.Cliente
        UNION ALL
        SELECT codCliente FROM kartoffeln3.Cliente
        UNION ALL
        SELECT codCliente FROM kartoffeln4.Cliente
    ) WHERE codCliente = p_codCliente;

    IF clienteExiste = 0 THEN
        RAISE_APPLICATION_ERROR(-20047, 'Error: El cliente no existe.');

    ELSE
        -- Determinar la provincia del hotel
        SELECT h.provincia INTO prov
        FROM HotelView h
        WHERE h.codHotel = p_codHotel;

        -- Proceder según el fragmento del hotel
        CASE
            WHEN prov IN ('Granada', 'Jaén') THEN
                -- Verificar si ya existe una reserva
                SELECT COUNT(*)
                INTO reservaExiste
                FROM kartoffeln1.Reserva
                WHERE codCliente = p_codCliente AND codHotel = p_codHotel
                    AND fechaEntrada = p_fechaEntrada AND fechaSalida = p_fechaSalida;

                IF reservaExiste > 0 THEN
                    -- Actualizar la reserva existente
                    UPDATE kartoffeln1.Reserva
                    SET tipoHab = p_tipoHab, precio = p_precio
                    WHERE codCliente = p_codCliente AND codHotel = p_codHotel
                        AND fechaEntrada = p_fechaEntrada AND fechaSalida = p_fechaSalida;
                ELSE
                    -- Crear una nueva reserva
```

```
53                      INSERT INTO kartoffeln1.Reserva (codCliente, codHotel, tipoHab,
                        fechaEntrada, fechaSalida, precio)
54                      VALUES (p_codCliente, p_codHotel, p_tipoHab, p_fechaEntrada, p_fechaSalida,
                        p_precio);
55                  END IF;
56
57              WHEN prov IN ('Málaga', 'Almería') THEN
58                  SELECT COUNT(*)
59                  INTO reservaExiste
60                  FROM kartoffeln2.Reserva
61                  WHERE codCliente = p_codCliente AND codHotel = p_codHotel
62                      AND fechaEntrada = p_fechaEntrada AND fechaSalida = p_fechaSalida;
63
64                  IF reservaExiste > 0 THEN
65                      UPDATE kartoffeln2.Reserva
66                      SET tipoHab = p_tipoHab, precio = p_precio
67                      WHERE codCliente = p_codCliente AND codHotel = p_codHotel
68                          AND fechaEntrada = p_fechaEntrada AND fechaSalida = p_fechaSalida;
69                  ELSE
70                      INSERT INTO kartoffeln2.Reserva (codCliente, codHotel, tipoHab,
                        fechaEntrada, fechaSalida, precio)
71                      VALUES (p_codCliente, p_codHotel, p_tipoHab, p_fechaEntrada, p_fechaSalida,
                        p_precio);
72                  END IF;
73
74              WHEN prov IN ('Sevilla', 'Córdoba') THEN
75                  SELECT COUNT(*)
76                  INTO reservaExiste
77                  FROM kartoffeln3.Reserva
78                  WHERE codCliente = p_codCliente AND codHotel = p_codHotel
79                      AND fechaEntrada = p_fechaEntrada AND fechaSalida = p_fechaSalida;
80
81                  IF reservaExiste > 0 THEN
82                      UPDATE kartoffeln3.Reserva
83                      SET tipoHab = p_tipoHab, precio = p_precio
84                      WHERE codCliente = p_codCliente AND codHotel = p_codHotel
85                          AND fechaEntrada = p_fechaEntrada AND fechaSalida = p_fechaSalida;
86                  ELSE
87                      INSERT INTO kartoffeln3.Reserva (codCliente, codHotel, tipoHab,
                        fechaEntrada, fechaSalida, precio)
88                      VALUES (p_codCliente, p_codHotel, p_tipoHab, p_fechaEntrada, p_fechaSalida,
                        p_precio);
89                  END IF;
90
91              WHEN prov IN ('Cádiz', 'Huelva') THEN
92                  SELECT COUNT(*)
93                  INTO reservaExiste
94                  FROM kartoffeln4.Reserva
95                  WHERE codCliente = p_codCliente AND codHotel = p_codHotel
96                      AND fechaEntrada = p_fechaEntrada AND fechaSalida = p_fechaSalida;
97
98                  IF reservaExiste > 0 THEN
99                      UPDATE kartoffeln4.Reserva
100                     SET tipoHab = p_tipoHab, precio = p_precio
```

```
                    WHERE codCliente = p_codCliente AND codHotel = p_codHotel
                        AND fechaEntrada = p_fechaEntrada AND fechaSalida = p_fechaSalida;
                ELSE
                    INSERT INTO kartoffeln4.Reserva (codCliente, codHotel, tipoHab,
                    fechaEntrada, fechaSalida, precio)
                    VALUES (p_codCliente, p_codHotel, p_tipoHab, p_fechaEntrada, p_fechaSalida,
                    p_precio);
                END IF;

            ELSE
                RAISE_APPLICATION_ERROR(-20048, 'Error: La provincia del hotel no corresponde a
                ningún fragmento válido.');
        END CASE;

        COMMIT;
        DBMS_OUTPUT.PUT_LINE('La reserva para el cliente ' || p_codCliente || ' en el hotel '
        || p_codHotel || ' ha sido gestionada correctamente.');
    END IF;
END;
```

Actualización 9

```
-- Actualización 9: Anular una reserva
create or replace PROCEDURE anular_reserva (
    p_codCliente NUMBER,
    p_codHotel NUMBER,
    p_fechaEntrada DATE,
    p_fechaSalida DATE
) IS
    prov Hotel.provincia%TYPE;
    clienteExiste NUMBER;
    hotelExiste NUMBER;
    hayReserva NUMBER;
BEGIN
    -- Verificar si el cliente existe en al menos un fragmento
    SELECT COUNT(*) INTO clienteExiste
    FROM (
    SELECT codCliente FROM kartoffeln1.Cliente
    UNION ALL
    SELECT codCliente FROM kartoffeln2.Cliente
    UNION ALL
    SELECT codCliente FROM kartoffeln3.Cliente
    UNION ALL
    SELECT codCliente FROM kartoffeln4.Cliente
    ) WHERE codCliente = p_codCliente;

    IF clienteExiste = 0 THEN
        RAISE_APPLICATION_ERROR(-20049, 'Error: El cliente no existe.');

    ELSE
        -- Verificar si el hotel existe
        SELECT COUNT(*) INTO hotelExiste
        FROM HotelView h WHERE h.codHotel = p_codHotel;

```

```
33          IF hotelExiste = 0 THEN
34              RAISE_APPLICATION_ERROR(-20050, 'No se ha encontrado ningún hotel con el código '
                || p_codHotel ||' .');
35
36          ELSE
37              -- Determinar la provincia del hotel
38              SELECT h.provincia INTO prov
39              FROM HotelView h
40              WHERE h.codHotel = p_codHotel;
41
42              SELECT COUNT(*) INTO hayReserva
43              FROM ReservaView r
44              WHERE r.codCliente = p_codCliente AND r.codHotel = p_codHotel AND
45              r.fechaEntrada = p_fechaEntrada AND r.fechaSalida = p_fechaSalida;
46
47              IF hayReserva = 0 THEN
48                  RAISE_APPLICATION_ERROR(-20168, 'NO SE HA ENCONTRADO RESERVAS CON LOS DATOS
                    ESPECIFICADOS');
49
50              ELSE
51                  -- Proceder según el fragmento del hotel
52                  CASE
53                      WHEN prov IN ('Granada', 'Jaén') THEN
54                          DELETE FROM kartoffeln1.Reserva
55                          WHERE codCliente = p_codCliente AND codHotel = p_codHotel
56                          AND fechaEntrada = p_fechaEntrada AND fechaSalida = p_fechaSalida;
57
58                      WHEN prov IN ('Málaga', 'Almería') THEN
59                          DELETE FROM kartoffeln2.Reserva
60                          WHERE codCliente = p_codCliente AND codHotel = p_codHotel
61                          AND fechaEntrada = p_fechaEntrada AND fechaSalida = p_fechaSalida;
62
63                      WHEN prov IN ('Sevilla', 'Córdoba') THEN
64                          DELETE FROM kartoffeln3.Reserva
65                          WHERE codCliente = p_codCliente AND codHotel = p_codHotel
66                          AND fechaEntrada = p_fechaEntrada AND fechaSalida = p_fechaSalida;
67
68                      WHEN prov IN ('Cádiz', 'Huelva') THEN
69                          DELETE FROM kartoffeln4.Reserva
70                          WHERE codCliente = p_codCliente AND codHotel = p_codHotel
71                          AND fechaEntrada = p_fechaEntrada AND fechaSalida = p_fechaSalida;
72
73                      ELSE
74                          RAISE_APPLICATION_ERROR(-20051, 'Error: La provincia del hotel no
                            corresponde a ningún fragmento válido.');
75                  END CASE;
76              END IF;
77          END IF;
78      END IF;
79
80      COMMIT;
81      DBMS_OUTPUT.PUT_LINE('La reserva para el cliente ' || p_codCliente || ' en el hotel ' ||
        p_codHotel || ' ha sido anulada correctamente.');
82  END;
```

Actualización 10

```
-- Actualización 10: Dar de alta a un proveedor
CREATE OR REPLACE PROCEDURE alta_proveedor (
    p_codProv NUMBER,
    p_nombre VARCHAR2,
    p_ciudad VARCHAR2)
IS
BEGIN
    -- Verificar que la ciudad sea válida (Granada o Sevilla)
    IF p_ciudad NOT IN ('Granada', 'Sevilla') THEN
        RAISE_APPLICATION_ERROR(-20052, 'La ciudad debe ser Granada o Sevilla.');

    ELSE
        -- Insertar el proveedor en el fragmento correspondiente según la ciudad
        CASE
            WHEN p_ciudad = 'Granada' THEN
                BEGIN
                    INSERT INTO kartoffeln1.Proveedor (codProv, nombre, ciudad)
                    VALUES (p_codProv, p_nombre, p_ciudad);
                EXCEPTION
                    WHEN DUP_VAL_ON_INDEX THEN
                        RAISE_APPLICATION_ERROR(-20053, 'El proveedor ya existe en Granada.');
                END;

            WHEN p_ciudad = 'Sevilla' THEN
                BEGIN
                    INSERT INTO kartoffeln3.Proveedor (codProv, nombre, ciudad)
                    VALUES (p_codProv, p_nombre, p_ciudad);
                EXCEPTION
                    WHEN DUP_VAL_ON_INDEX THEN
                        RAISE_APPLICATION_ERROR(-20054, 'El proveedor ya existe en Sevilla.');
                END;

            ELSE
                RAISE_APPLICATION_ERROR(-20055, 'Error inesperado: la ciudad no corresponde a
                ningún fragmento válido.');
        END CASE;

        COMMIT;
        DBMS_OUTPUT.PUT_LINE('El proveedor "' || p_nombre || '" ha sido dado de alta
        correctamente en ' || p_ciudad || '.');
    END IF;
END;
```

Actualización 11

```
-- Actualización 11: Dar de baja a un proveedor
CREATE OR REPLACE PROCEDURE baja_proveedor (
    p_codProv NUMBER
) IS
    suministrosActivos NUMBER := 0;
    tieneRelacion NUMBER := 0;
```

```
7       ciudadProveedor Proveedor.ciudad%TYPE;
8   BEGIN
9       -- Determinar la ciudad del proveedor
10      SELECT ciudad INTO ciudadProveedor
11      FROM ProveedorView p
12      WHERE p.codProv = p_codProv;
13
14      -- Verificar si el proveedor tiene suministros activos
15      CASE
16          WHEN ciudadProveedor = 'Granada' THEN
17              SELECT COUNT(*) INTO suministrosActivos
18              FROM kartoffeln1.Suministra
19              WHERE codProv = p_codProv AND cantidad > 0;
20
21          WHEN ciudadProveedor = 'Sevilla' THEN
22              SELECT COUNT(*) INTO suministrosActivos
23              FROM kartoffeln3.Suministra
24              WHERE codProv = p_codProv AND cantidad > 0;
25
26          ELSE
27          RAISE_APPLICATION_ERROR(-20056, 'La ciudad del proveedor debe ser Granada o Sevilla.');
28      END CASE;
29
30      IF suministrosActivos > 0 THEN
31          RAISE_APPLICATION_ERROR(-20057, 'No se puede eliminar el proveedor porque tiene
            suministros activos.');
32      END IF;
33
34      -- Verificar si el proveedor tiene relaciones activas en la tabla Tiene
35      CASE
36          WHEN ciudadProveedor = 'Granada' THEN
37              SELECT COUNT(*) INTO tieneRelacion
38              FROM kartoffeln1.Tiene
39              WHERE codProv = p_codProv;
40
41          WHEN ciudadProveedor = 'Sevilla' THEN
42              SELECT COUNT(*) INTO tieneRelacion
43              FROM kartoffeln3.Tiene
44              WHERE codProv = p_codProv;
45
46          ELSE
47              RAISE_APPLICATION_ERROR(-20056, 'La ciudad del proveedor debe ser Granada o
                Sevilla.');
48      END CASE;
49
50
51      -- Eliminar el proveedor del fragmento correspondiente
52      CASE
53          WHEN ciudadProveedor = 'Granada' THEN
54              IF tieneRelacion > 0 THEN
55                  -- Eliminar la relación con los proveedores en los fragmentos correspondientes
56                  DELETE FROM kartoffeln1.Tiene WHERE codProv = p_codProv;
57              END IF;
58              DELETE FROM kartoffeln1.Proveedor WHERE codProv = p_codProv;
```

```
59
60          WHEN ciudadProveedor = 'Sevilla' THEN
61              IF tieneRelacion > 0 THEN
62                  -- Eliminar la relación con los proveedores en los fragmentos correspondientes
63                  DELETE FROM kartoffeln3.Tiene WHERE codProv = p_codProv;
64              END IF;
65              DELETE FROM kartoffeln3.Proveedor WHERE codProv = p_codProv;
66
67          ELSE
68              RAISE_APPLICATION_ERROR(-20056, 'La ciudad del proveedor debe ser Granada o
                Sevilla.');
69      END CASE;
70
71      COMMIT;
72      DBMS_OUTPUT.PUT_LINE('El proveedor con código ' || p_codProv || ' ha sido eliminado
        correctamente.');
73  END;
```

Actualización 12

```
1   -- Actualización 10: Dar de alta o actualizar un suministro
2   CREATE OR REPLACE PROCEDURE gestionar_suministro (
3       p_codArticulo NUMBER,
4       p_codProv NUMBER,
5       p_codHotel NUMBER,
6       p_fecha DATE,
7       p_cantidad NUMBER,
8       p_precio NUMBER
9   ) IS
10      ciudadProveedor Proveedor.ciudad%TYPE;
11      suministroExiste NUMBER := 0;
12      cantidadActual NUMBER := 0;
13  BEGIN
14      -- Determinar la ciudad del proveedor
15      SELECT ciudad INTO ciudadProveedor
16      FROM ProveedorView p
17      WHERE p.codProv = p_codProv;
18
19      IF p_precio <= 0 THEN
20          RAISE_APPLICATION_ERROR(-20109, 'El precio debe ser mayor que 0');
21      END IF;
22
23      -- Proceder según la ciudad del proveedor
24      IF ciudadProveedor = 'Granada' THEN
25          -- Verificar si ya existe un suministro en kartoffeln1.Suministra
26          SELECT COUNT(*) INTO suministroExiste
27          FROM kartoffeln1.Suministra
28          WHERE codArticulo = p_codArticulo AND codProv = p_codProv AND codHotel = p_codHotel AND
            fecha = p_fecha;
29
30          IF suministroExiste > 0 THEN
31              -- Verificar que la cantidad total no sea negativa
32              SELECT cantidad INTO cantidadActual
33              FROM kartoffeln1.Suministra
```

```
34              WHERE codArticulo = p_codArticulo AND codProv = p_codProv AND codHotel = p_codHotel
                AND fecha = p_fecha;
35
36              IF cantidadActual + p_cantidad < 0 THEN
37                  RAISE_APPLICATION_ERROR(-20057, 'La cantidad total no puede ser negativa.');
38              END IF;
39
40              -- Actualizar el suministro existente
41              UPDATE kartoffeln1.Suministra
42              SET cantidad = cantidad + p_cantidad, precio = p_precio
43              WHERE codArticulo = p_codArticulo AND codProv = p_codProv AND codHotel = p_codHotel
                AND fecha = p_fecha;
44
45          ELSE
46              -- Insertar un nuevo suministro
47              INSERT INTO kartoffeln1.Suministra (codProv, codHotel, codArticulo, cantidad,
                precio, fecha)
48              VALUES (p_codProv, p_codHotel, p_codArticulo, p_cantidad, p_precio, p_fecha);
49          END IF;
50
51      ELSIF ciudadProveedor = 'Sevilla' THEN
52          -- Verificar si ya existe un suministro en kartoffeln3.Suministra
53          SELECT COUNT(*) INTO suministroExiste
54          FROM kartoffeln3.Suministra
55          WHERE codArticulo = p_codArticulo AND codProv = p_codProv AND codHotel = p_codHotel AND
            fecha = p_fecha;
56
57          IF suministroExiste > 0 THEN
58
59              -- Verificar que la cantidad total no sea negativa
60              SELECT cantidad INTO cantidadActual
61              FROM kartoffeln3.Suministra
62              WHERE codArticulo = p_codArticulo AND codProv = p_codProv AND codHotel = p_codHotel
                AND fecha = p_fecha;
63
64              IF cantidadActual + p_cantidad < 0 THEN
65                  RAISE_APPLICATION_ERROR(-20057, 'La cantidad total no puede ser negativa.');
66              END IF;
67
68              -- Actualizar el suministro existente
69              UPDATE kartoffeln3.Suministra
70              SET cantidad = cantidad + p_cantidad, precio = p_precio
71              WHERE codArticulo = p_codArticulo AND codProv = p_codProv AND codHotel = p_codHotel
                AND fecha = p_fecha;
72
73          ELSE
74              -- Insertar un nuevo suministro
75              INSERT INTO kartoffeln3.Suministra (codProv, codHotel, codArticulo, cantidad,
                precio, fecha)
76              VALUES (p_codProv, p_codHotel, p_codArticulo, p_cantidad, p_precio, p_fecha);
77          END IF;
78
79      ELSE
80          RAISE_APPLICATION_ERROR(-20058, 'La ciudad del proveedor debe ser Granada o Sevilla.');
```

```
    END IF;

    COMMIT;
    DBMS_OUTPUT.PUT_LINE('El suministro del artículo ' || p_codArticulo || ' con proveedor ' ||
    p_codProv || ' ha sido gestionado correctamente.');
END;
```

Actualización 13

```
-- Actualización 10: Dar de baja suministros
CREATE OR REPLACE PROCEDURE baja_suministros (
    p_codHotel NUMBER,
    p_codArticulo NUMBER,
    p_fecha DATE DEFAULT NULL
) IS
    v_count1 NUMBER;
    v_count2 NUMBER;
BEGIN
    IF p_fecha IS NOT NULL THEN
        SELECT COUNT(*) INTO v_count1
        FROM kartoffeln1.Suministra
        WHERE codHotel = p_codHotel AND codArticulo = p_codArticulo AND
        fecha = p_fecha;

        SELECT COUNT(*) INTO v_count2
        FROM kartoffeln3.Suministra
        WHERE codHotel = p_codHotel AND codArticulo = p_codArticulo AND
        fecha = p_fecha;

        IF v_count1 > 0 THEN
            DELETE FROM kartoffeln1.Suministra
            WHERE codHotel = p_codHotel AND codArticulo = p_codArticulo AND fecha = p_fecha;
            ELSIF v_count2 > 0 THEN
            DELETE FROM kartoffeln3.Suministra
            WHERE codHotel = p_codHotel AND codArticulo = p_codArticulo AND fecha = p_fecha;
        ELSE
            RAISE_APPLICATION_ERROR(-20059, 'No se ha encontrado resultados en la fecha
            especificada: ' || p_fecha);
        END IF;

    ELSE
        SELECT COUNT(*) INTO v_count1
        FROM kartoffeln1.Suministra
        WHERE codHotel = p_codHotel AND codArticulo = p_codArticulo;

        SELECT COUNT(*) INTO v_count2
        FROM kartoffeln3.Suministra
        WHERE codHotel = p_codHotel AND codArticulo = p_codArticulo;

        IF v_count1 > 0 THEN
            DELETE FROM kartoffeln1.Suministra
            WHERE codHotel = p_codHotel AND codArticulo = p_codArticulo;
            ELSIF v_count2 > 0 THEN
            DELETE FROM kartoffeln3.Suministra
```

```
            WHERE codHotel = p_codHotel AND codArticulo = p_codArticulo;
        ELSE
            RAISE_APPLICATION_ERROR(-20059, 'No se ha encontrado resultados');
        END IF;
    END IF;

    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Los suministros del artículo ' || p_codArticulo || ' en el hotel ' ||
    p_codHotel || ' han sido eliminados correctamente.');
END;
```

Actualización 14

```
-- Actualización 14: Dar de alta un nuevo artículo
CREATE OR REPLACE PROCEDURE alta_articulo (
    p_codArticulo NUMBER,
    p_nombre VARCHAR2,
    p_tipo CHAR,
    p_codProv NUMBER)
IS
    ciudadProveedor Proveedor.ciudad%TYPE;
BEGIN
    -- Validar el tipo de artículo
    IF p_tipo NOT IN ('A', 'B', 'C', 'D') THEN
        RAISE_APPLICATION_ERROR(-20060, 'El tipo de artículo debe ser A, B, C o D.');
    END IF;

    -- Determinar la ciudad del proveedor
    SELECT ciudad INTO ciudadProveedor
    FROM ProveedorView p
    WHERE p.codProv = p_codProv;

    -- Insertar el artículo en todas las réplicas
    BEGIN
        INSERT INTO kartoffeln1.Articulo (codArticulo, nombre, tipo)
        VALUES (p_codArticulo, p_nombre, p_tipo);
    EXCEPTION
        WHEN DUP_VAL_ON_INDEX THEN
            NULL; -- Si ya existe en esta réplica, continuar
    END;

    BEGIN
        INSERT INTO kartoffeln2.Articulo (codArticulo, nombre, tipo)
        VALUES (p_codArticulo, p_nombre, p_tipo);
    EXCEPTION
        WHEN DUP_VAL_ON_INDEX THEN
            NULL; -- Si ya existe en esta réplica, continuar
    END;

    BEGIN
        INSERT INTO kartoffeln3.Articulo (codArticulo, nombre, tipo)
        VALUES (p_codArticulo, p_nombre, p_tipo);
    EXCEPTION
        WHEN DUP_VAL_ON_INDEX THEN
```

```
            NULL; -- Si ya existe en esta réplica, continuar
    END;

    BEGIN
        INSERT INTO kartoffeln4.Articulo (codArticulo, nombre, tipo)
        VALUES (p_codArticulo, p_nombre, p_tipo);
    EXCEPTION
        WHEN DUP_VAL_ON_INDEX THEN
            NULL; -- Si ya existe en esta réplica, continuar
    END;

    -- Asociar el artículo con el proveedor en el fragmento correspondiente
    CASE
        WHEN ciudadProveedor = 'Granada' THEN
            INSERT INTO kartoffeln1.Tiene (codProv, codArticulo)
            VALUES (p_codProv, p_codArticulo);

        WHEN ciudadProveedor = 'Sevilla' THEN
            INSERT INTO kartoffeln3.Tiene (codProv, codArticulo)
            VALUES (p_codProv, p_codArticulo);

        ELSE
            (-20061, 'La ciudad del proveedor debe ser Granada o Sevilla.');
    END CASE;

    COMMIT;
    DBMS_OUTPUT.PUT_LINE('El artículo "' || p_nombre || '" ha sido dado de alta correctamente y
    asociado al proveedor ' || p_codProv || '.');
END;
```

Actualización 15

```
-- Actualización 15: Dar de baja a un artículo
CREATE OR REPLACE PROCEDURE baja_articulo (
    p_codArticulo NUMBER
) IS
    suministrosActivos NUMBER := 0;
BEGIN
    -- Verificar si el artículo tiene suministros activos en cualquier fragmento
    SELECT COUNT(*) INTO suministrosActivos
    FROM SuministraView s
    WHERE s.codArticulo = p_codArticulo AND s.cantidad > 0;

    IF suministrosActivos > 0 THEN
        RAISE_APPLICATION_ERROR(-20062, 'No se puede eliminar el artículo porque tiene
        suministros activos.');
    END IF;

    -- Eliminar los suministros relacionados con el artículo en todos los fragmentos
    DELETE FROM kartoffeln1.Suministra WHERE codArticulo = p_codArticulo;
    DELETE FROM kartoffeln3.Suministra WHERE codArticulo = p_codArticulo;

    -- Eliminar la relación con los proveedores en los fragmentos correspondientes
    DELETE FROM kartoffeln1.Tiene WHERE codArticulo = p_codArticulo;
```

```
    DELETE FROM kartoffeln3.Tiene WHERE codArticulo = p_codArticulo;

    -- Eliminar el artículo de todas las réplicas
    BEGIN
        DELETE FROM kartoffeln1.Articulo WHERE codArticulo = p_codArticulo;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            NULL; -- Si no existe, continuar
    END;

    BEGIN
        DELETE FROM kartoffeln2.Articulo WHERE codArticulo = p_codArticulo;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            NULL; -- Si no existe, continuar
    END;

    BEGIN
        DELETE FROM kartoffeln3.Articulo WHERE codArticulo = p_codArticulo;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            NULL; -- Si no existe, continuar
    END;

    BEGIN
        DELETE FROM kartoffeln4.Articulo WHERE codArticulo = p_codArticulo;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            NULL; -- Si no existe, continuar
    END;

    COMMIT;
    DBMS_OUTPUT.PUT_LINE('El artículo con código ' || p_codArticulo || ' ha sido eliminado
    correctamente.');
END;
```

6. Implementación de consultas

Consulta 1

```
-- Consulta 1: Listar los hoteles (nombre y ciudad) de las provincias de Granada, Huelva o
-- Almería, y los proveedores (nombre y ciudad), a los que se le ha suministrado
-- ``Queso" o ``Mantequilla" entre el 12 de mayo de 2024 y el 28 de mayo de 2024.
SELECT DISTINCT H.nombre AS hotel_nombre,
                H.ciudad AS hotel_ciudad,
                P.nombre AS proveedor_nombre,
                P.ciudad AS proveedor_ciudad
FROM HotelView H
```

```
JOIN SuministraView S ON H.codHotel = S.codHotel
JOIN ArticuloView A ON S.codArticulo = A.codArticulo
JOIN ProveedorView P ON S.codProv = P.codProv
WHERE H.provincia IN ('Granada', 'Huelva', 'Almería') AND
      A.nombre IN ('Queso', 'Mantequilla') AND
      S.fecha BETWEEN TO_DATE('2024-05-12', 'YYYY-MM-DD') AND
      TO_DATE('2024-05-28', 'YYYY-MM-DD');
```

Consulta 2

```
-- Consulta 2: Dado por teclado el código de un productor, \Listar los productos (nombre), los
-- hoteles (nombre y ciudad) y la cantidad total de cada producto, suministrados por
-- dicho productor a hoteles de las provincias de Jaén o Almería".
ACCEPT codigo NUMBER PROMPT 'Introduce el código de proveedor deseado: '
SELECT A.nombre AS producto_nombre,
       H.nombre AS hotel_nombre,
       H.ciudad AS hotel_ciudad,
       SUM(S.cantidad) AS cantidad_total
FROM SuministraView S
JOIN ArticuloView A ON S.codArticulo = A.codArticulo
JOIN HotelView H ON S.codHotel = H.codHotel
WHERE S.codProv = &codigo AND H.provincia IN ('Jaén', 'Almería')
GROUP BY A.nombre, H.nombre, H.ciudad
ORDER BY A.nombre, H.nombre;

```

Consulta 3

```
-- Consulta 3: Dado por teclado el código de un hotel, \Listar los clientes (nombre y teléfono),
-- que tengan registrada más de una reserva en dicho hotel.
ACCEPT codigo NUMBER PROMPT 'Introduce el código de hotel deseado: '
SELECT C.nombre AS cliente_nombre, C.telefono AS cliente_telefono
FROM ClienteView C
JOIN ReservaView R ON R.codCliente = C.codCliente
WHERE R.codHotel = &codigo
GROUP BY C.codCliente, C.nombre, C.telefono
HAVING COUNT(*) > 1 ORDER BY C.nombre;
```

7. Experiencia

Durante el desarrollo del curso, hemos logrado aprender un aspecto fundamental que no lo teníamos dado por sabido, puesto a niveles reales, el paso de datos que existe a día de hoy son de cantidades descomunales como para ser almacenados en un mismo punto físico. Por ello, este acercamiento de las bases de datos distribuidas contribuye a un manejo de datos a un nivel más inferior.

Para el desarrollo de la práctica, la implementación de los esquemas conceptual, lógico y de las fragmentaciones ha resultado bastante sencilla, asimismo como la implementación de las bases de datos como tal. Sin embargo, en el camino de la implementación nos hemos encontrado con problemas

desprevenidos como es el caso de una mala referencia a una tabla cuando queríamos asociar datos de un fragmento de un lado a otro. No obstante, ha sido satisfactorio el poder resolver ciertos problemas puesto que se deben por su "rareza" en este ámbito universitario.

Referencias

[1] Plataforma empleada para generar los diagramas: `https://www.drawio.com/`

[2] Guión de la práctica y seminarios

[3] Transparencias de los temarios de la asignatura