**BASE DE DATOS I**



**FabiaNatura DB System**

**Modelado y Construcción de una Base de Datos**

**Integrantes del proyecto:**

**Rodrigo Emerson Infanzón Acosta**

**Bekam Eddy Marc Huaracha Cabrera**

**Elias Efrain Manchego Navarro**

**Empresas involucrada:**

**Empresa solicitante: FabiaNatura**

**Arequipa - Perú**

**2024 - II**

1. **Introducción:** 
   1. Propósito del documento
2. **Vista general:**

2.1. Problemática

2.2. Modelo conceptual

2.3 Modelo entidad relación

2.4 Nominalización

2.5 Modelo físico

1. **Desarrollo:**

3.1 Creación de la base de datos - usuarios

3.2 Carga de datos

3.3 Diseño de funciones

3.4 Diseño de procedimientos almacenados

3.5 Diseño de vistas - consultas simples y avanzadas

3.6 Diseño de triggers

**1. Introducción**

**1.1. Propósito del documento**

El propósito de este documento es describir el modelado y la construcción de una base de datos diseñada específicamente para **FabiaNatura**, que permitirá una gestión más eficiente de los productos, ventas, trabajadores y clientes.

Además de consolidar la información, esta base de datos proporcionará una infraestructura **flexible y escalable** que no solo automatizará tareas repetitivas, sino que también podrá ofrecer la capacidad de **generar informes sobre el rendimiento de ventas** proveedores , el **manejo de inventarios y una búsqueda avanzada**. Esta capacidad analítica permitirá a **FabiaNatura** tomar decisiones informadas basadas en datos reales, generando una ventaja competitiva.

El sistema propuesto también está diseñado para integrar de manera sencilla **futuras funcionalidades**, aumentando así poco a poco la presencia de la empresa y facilitando su expansión hacia nuevos mercados.

**2. Vista general**

**2.1. Problemática**

En FabiaNatura, la problemática de gestionar información de manera eficiente abarca varios aspectos clave que afectan tanto a la operación interna de la empresa como a la experiencia del cliente. Para abordar estos desafíos, la empresa necesita contar con un sistema organizado que permita registrar y gestionar de forma detallada los atributos de cada entidad relevante dentro del proceso comercial, tales como los empleados, clientes, productos y ventas. A continuación se detallan los principales atributos necesarios para gestionar correctamente estos aspectos.

**Clientes:**

Uno de los principales desafíos para FabiaNatura es mantener un registro detallado de los clientes y sus características. Los clientes deben ser identificados por su DNI, que actúa como un identificador único para cada persona. Además, se debe almacenar información básica de contacto como el nombre completo, número de teléfono, correo electrónico y dirección. Para poder ofrecer un servicio personalizado, es crucial registrar el tipo de cliente, que puede ser regular o frecuente, dependiendo de la frecuencia con que el cliente realice compras. Esta clasificación se actualiza dinámicamente a medida que el cliente realiza más compras.

**Empleados:**

En cuanto a los empleados, es necesario contar con atributos como el nombre completo, fecha de nacimiento y número de teléfono para cada trabajador. Además, cada empleado debe tener un estado (activo o inactivo), lo que permite conocer si el empleado está actualmente trabajando en la empresa o si ha dejado de hacerlo. Los empleados pueden ser de dos tipos: vendedores o asesores, y para cada tipo de puesto es necesario registrar atributos específicos. Los vendedores tienen un rol o función asignada dentro de la empresa, mientras que los asesores deben contar con información sobre su experiencia laboral y su especialización. Este detalle es fundamental, ya que los asesores son los responsables de ofrecer recomendaciones personalizadas a los clientes en base a sus necesidades.

Además de los atributos personales y laborales, cada empleado tiene un contrato, que debe contener detalles como la fecha de inicio, fecha de vencimiento, salario mensual y observaciones adicionales, permitiendo llevar un control completo de las condiciones laborales de cada trabajador.

**Productos:**

La gestión de los productos es otra de las áreas clave para FabiaNatura. Cada producto debe ser identificado por un código único que permita su fácil localización. Además, se deben registrar atributos como el nombre, descripción, precio de compra y precio de venta, los cuales son cruciales tanto para el proceso de ventas como para el control de inventarios. El stock o cantidad disponible de cada producto es otro atributo importante, ya que permite conocer en tiempo real cuántos productos están disponibles para su venta. También se debe definir el estado del producto, ya sea disponible o agotado, para asegurar que los productos estén correctamente catalogados según su disponibilidad.

Cada producto está asociado a una categoría, la cual se caracteriza por un nombre y una descripción, lo que facilita su clasificación dentro del inventario. Este atributo es esencial para organizar los productos de forma que los clientes puedan encontrar lo que buscan de manera más eficiente.

**Proveedores:**

Los proveedores juegan un rol fundamental en el abastecimiento de productos, y es necesario registrar atributos específicos para gestionarlos correctamente. Cada proveedor debe estar identificado por su RUC, que es un número único, y se debe almacenar su nombre, teléfono de contacto, correo electrónico y dirección. Estos datos permiten mantener una relación fluida con los proveedores, así como facilitar la reposición de productos cuando sea necesario.

**Categorías:**

Las categorías son esenciales para organizar los productos dentro de la tienda y facilitar su búsqueda y clasificación. Cada categoría debe tener un código único que la identifique, y debe ser asociada a un nombre claro que describa el tipo de productos que agrupa. Además, se debe registrar una descripción de cada categoría para proporcionar más contexto sobre los productos que contiene. Esta descripción es importante tanto para el personal de ventas como para los clientes, ya que puede incluir detalles sobre el tipo de productos o servicios que ofrece cada categoría.

La fecha de registro también es un atributo relevante, ya que permite saber cuándo fue creada cada categoría, lo que puede ser útil para el análisis de las tendencias de ventas o para la reorganización del inventario a lo largo del tiempo. Tener estas categorías bien definidas también ayuda a la empresa a segmentar su oferta de productos, lo que mejora tanto la experiencia del cliente como la eficiencia en la gestión de inventarios.

**Ventas y Facturación:**

En el proceso de ventas, se debe registrar cada transacción mediante una factura que contiene varios atributos clave. La fecha de la venta, el cliente que realiza la compra, el vendedor que realizó la venta y el asesor que atendió al cliente son todos atributos esenciales para realizar un seguimiento detallado de cada transacción. Además, es importante registrar los productos vendidos, junto con la cantidad de cada uno, ya que esto impacta directamente en el stock disponible y en el total de la venta.

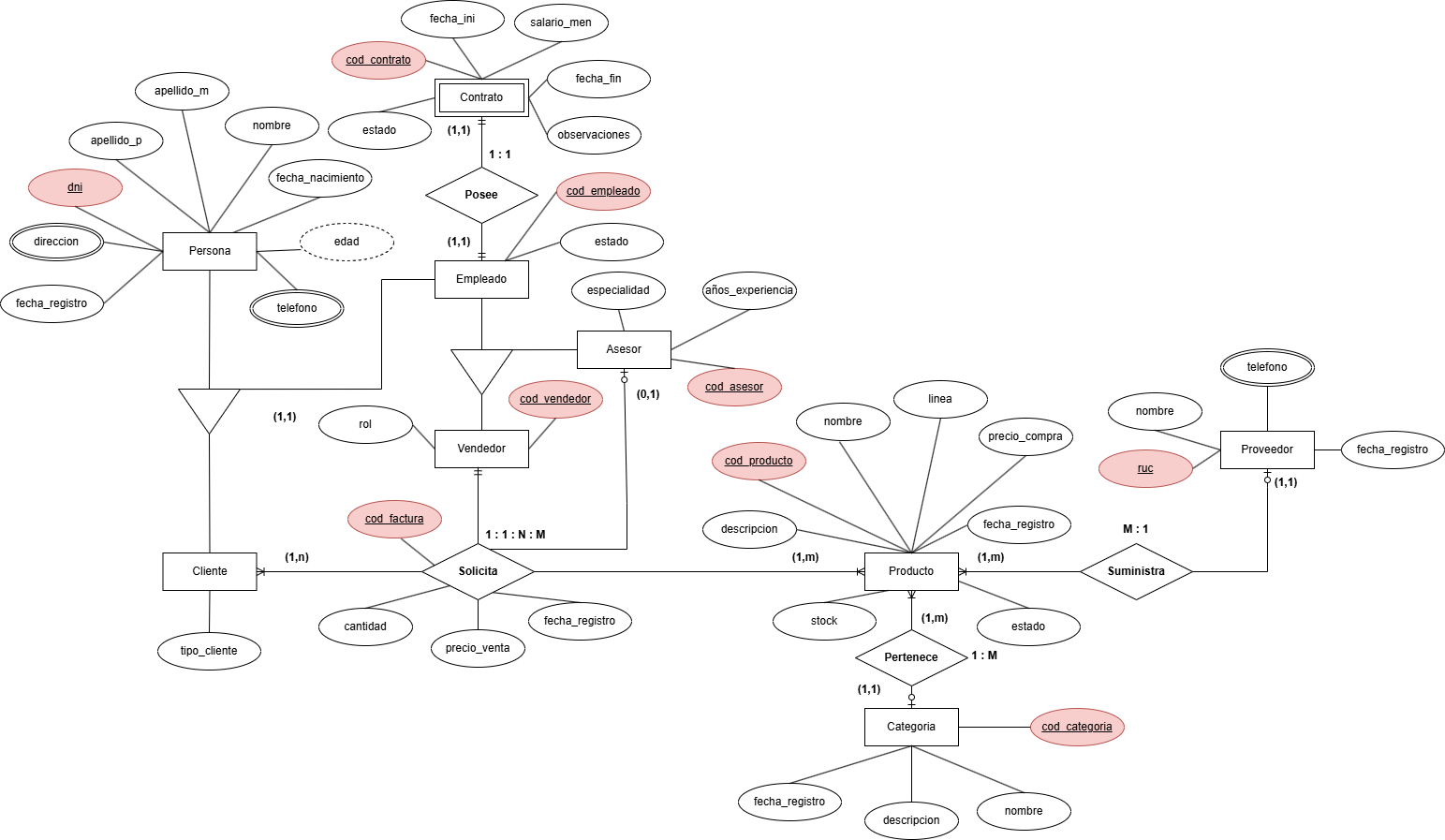
Cada factura también incluye información sobre los productos vendidos, especificando el precio de cada producto y la cantidad adquirida. Estos datos permiten a la empresa realizar cálculos precisos sobre el total de la venta, asegurando una facturación correcta. También es necesario llevar un control de las ventas realizadas, permitiendo a la empresa realizar análisis sobre el rendimiento de los vendedores y asesores, así como sobre las preferencias de los clientes.

**Relación entre Vendedores, Asesores y Clientes:**

Un desafío adicional es el seguimiento de las interacciones entre clientes, vendedores y asesores. Es crucial registrar la participación opcional u obligatoria del asesor en cada venta, ya que son los asesores los encargados de ofrecer un servicio más personalizado y ayudar a los clientes a encontrar los productos que mejor se adapten a sus necesidades. Este detalle también debe ser almacenado, junto con la calificación o satisfacción del cliente, para evaluar la efectividad de la atención brindada por el asesor.

**2.2. Modelo conceptual**

**Figura 1: modelo conceptual**



**2.3. Modelo entidad relación**

**Figura 2: modelo entidad relación**

****

**2.4. Normalización:**

Se aplicará normalización para: Soli\_Cliente

Soli\_Cliente(cod\_factura, cod\_asesor, cod\_producto, cod\_empleado, dni, fecha\_registro, precio\_venta, cantidad)

**1FN :** La relación Soli\_Cliente(cod\_factura, cod\_asesor, cod\_producto, cod\_empleado, dni, fecha\_registro, precio\_venta, cantidad) está en 1NF, ya que cumple con los requisitos de la primera forma normal: los atributos tienen valores atómicos, no hay grupos repetidos de columnas, los atributos tienen dominios adecuados y la clave primaria garantiza que cada tupla es única.

**2FN:**

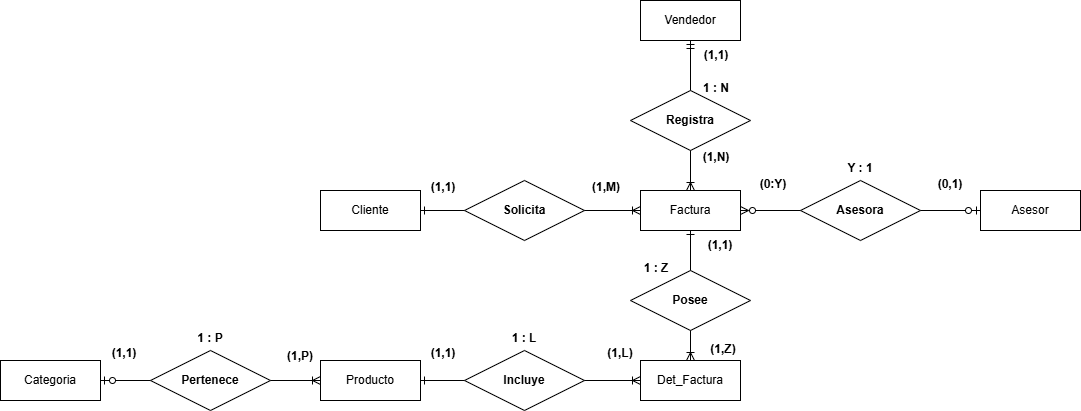
Det\_Factura(cod\_facturad, cod\_factura, cod\_producto, cantidad)

Producto(cod\_producto, precio\_venta, nombre, descripcion, cod\_categoria, precio\_compra, fecha\_registro, ruc)

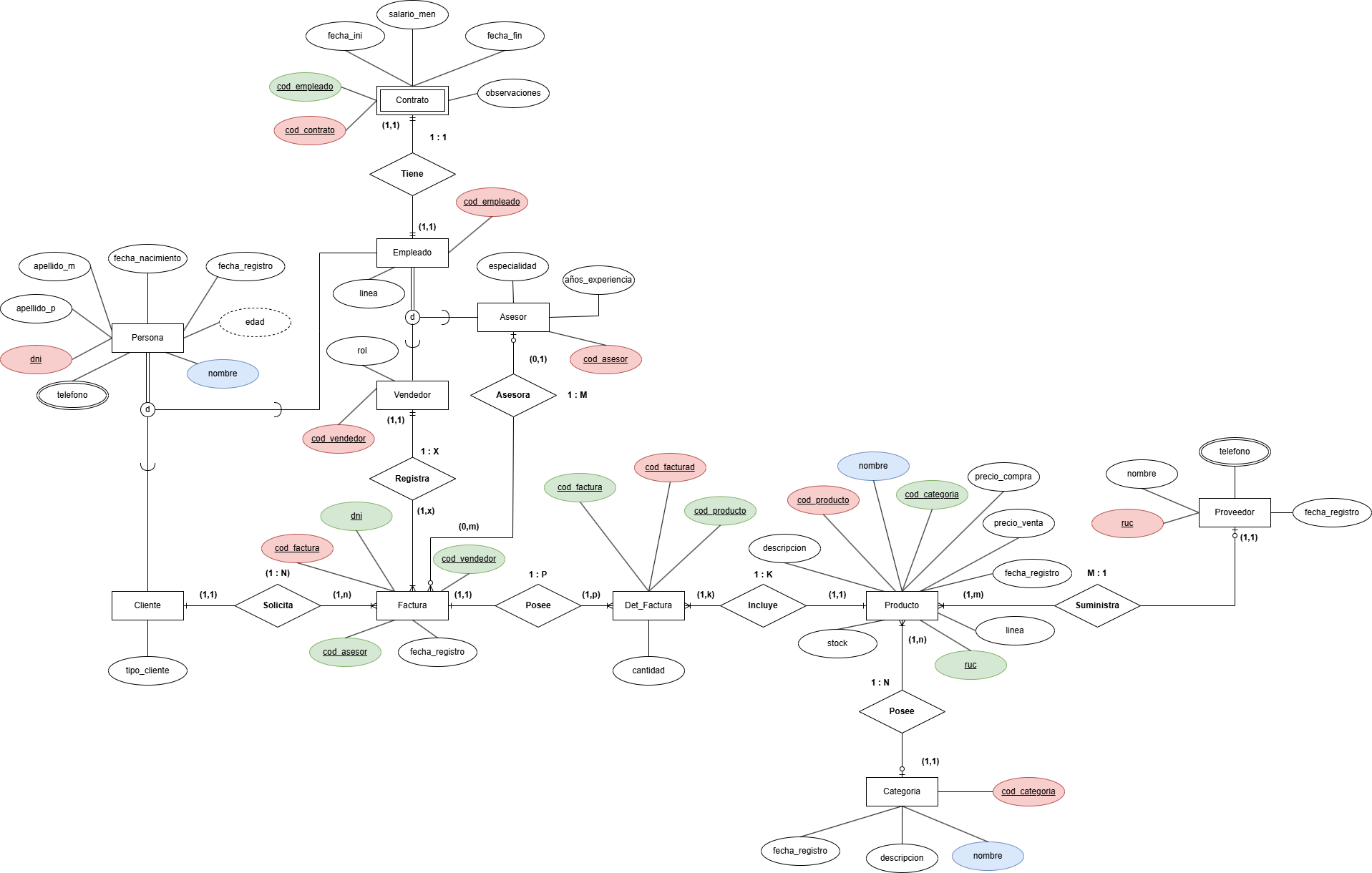
**3FN:**

Factura(cod\_factura, fecha\_registro, dni, cod\_asesor, cod\_empleado)

**Figura 3: concepto normalización**

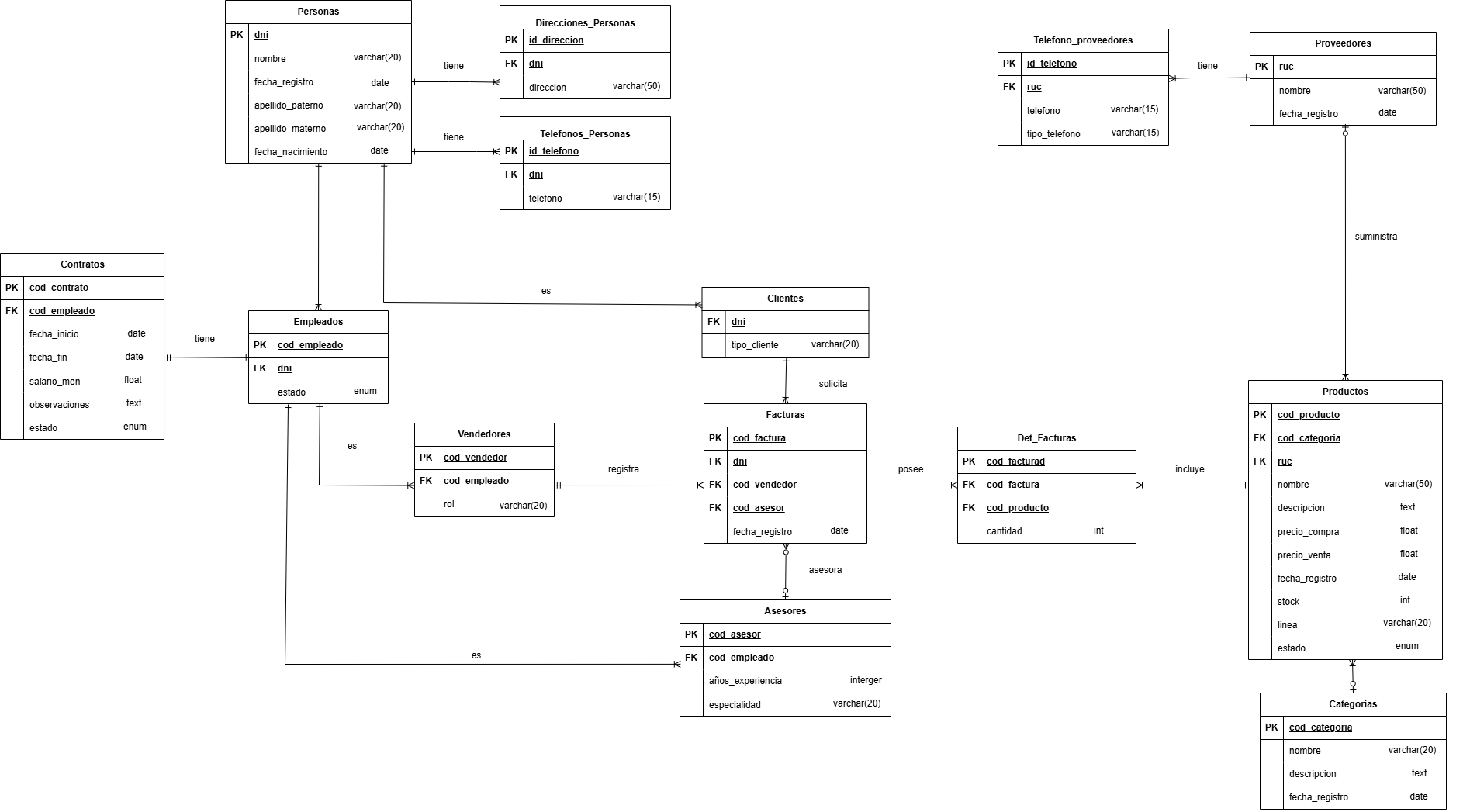
****

**Figura 4: modelo entidad relación normalizado**

****

**2.5 Modelo físico:**

**Figura 5: modelo físico**

****

**3. Desarrollo**

**3.1. Creación de la base de datos - usuarios**

En el proceso de desarrollo y gestión de una base de datos para FabiaNatura, es necesario crear tanto la base de datos como los usuarios que interactuaron con ella. Este documento detalla cómo llevar a cabo estas tareas utilizando MySQL 9.1, asegurando un sistema seguro y eficiente para la gestión de la información relacionada con clientes, empleados, productos, ventas, proveedores y categorías:

**3.1.1. Creación de la Base de Datos**

El primer paso en el proceso de creación de la base de datos es crear la base de datos en sí. Para ello, se utiliza el siguiente comando SQL:

**CREATE DATABASE IF NOT EXISTS FabiaNatura;**

Este comando crea una nueva base de datos llamada FabiaNatura si no existe ya. De lo contrario, si la base de datos ya está presente, no se realiza ninguna acción. Después de crear la base de datos, se debe indicar que se va a trabajar en ella mediante el comando:

**USE FabiaNatura;**

Este comando selecciona la base de datos FabiaNatura para que todas las operaciones posteriores se realicen dentro de esta.

**3.1.2. Creación del Usuario**

Una vez creada la base de datos, el siguiente paso es crear un usuario que tendrá acceso a la misma. En este caso, se creará un usuario llamado 'rodrigo', con acceso local al servidor MySQL. Este usuario podrá ejecutar consultas, agregar, modificar y eliminar registros según los privilegios otorgados.

Para crear el usuario, se usa el siguiente comando SQL:

**CREATE USER 'rodrigo'@'localhost' IDENTIFIED BY 'ubnt';**

Este comando crea un usuario llamado 'rodrigo' y le asigna la contraseña 'ubnt'. El acceso está restringido a localhost, lo que significa que solo podrá acceder desde el mismo servidor donde está alojada la base de datos, mejorando la seguridad.

**3.1.3. Otorgamiento de Privilegios**

Una vez creado el usuario, es necesario asignarle los privilegios que definan qué operaciones podrá realizar dentro de la base de datos. En este caso, se le otorgarán todos los privilegios sobre todas las bases de datos. Esto permite que el usuario 'rodrigo' realice cualquier acción, como insertar, actualizar, eliminar o consultar datos, en cualquier base de datos dentro del servidor.

El comando para otorgar estos privilegios es:

**GRANT ALL PRIVILEGES ON \*.\* TO 'rodrigo'@'localhost' WITH GRANT OPTION;**

Este comando otorga todos los privilegios sobre todas las bases de datos (.) al usuario 'rodrigo' desde localhost. Además, el WITH GRANT OPTION permite que el usuario 'rodrigo' también pueda otorgar privilegios a otros usuarios en el futuro.

**3.1.4. Aplicación de Cambios de Privilegios**

Para que los privilegios otorgados al usuario tengan efecto, es necesario ejecutar el siguiente comando:

**FLUSH PRIVILEGES;**

Este comando recarga la tabla de privilegios y asegura que los cambios realizados en los permisos se apliquen de inmediato.

**3.1.5. Creación de las Tablas**

Con el usuario y los privilegios establecidos, el siguiente paso es la creación de las tablas que almacenarán los datos en la base de datos FabiaNatura. A continuación, se presentan las tablas necesarias para organizar la información de personas, empleados, productos, ventas, categorías, proveedores y otros elementos esenciales para la empresa.

**Tabla de Personas**

La tabla Personas almacena la información básica de los clientes, empleados y otros individuos relacionados con la empresa. Los atributos de esta tabla incluyen:

* dni: Documento de Identidad único de la persona.
* nombre: Nombre de la persona.
* apellido\_paterno: Apellido paterno.
* apellido\_materno: Apellido materno.
* fecha\_nacimiento: Fecha de nacimiento.
* fecha\_registro: Fecha de registro de la persona.

**Tabla de Teléfonos de Personas**

La tabla Telefonos\_Personas almacena los números de teléfono asociados a cada persona. Los atributos incluyen:

* telefono: Número de teléfono.
* dni: Documento de Identidad de la persona asociada al teléfono.
* Tabla de Direcciones de Personas

**En la tabla Direcciones\_Personas** se almacenan las direcciones de cada persona:

* id\_direccion: Identificador único de la dirección.
* dni: Documento de Identidad de la persona a la que pertenece la dirección.
* direccion: Dirección completa de la persona.

**Tabla de Empleados**

La tabla Empleados almacena los registros de los empleados de la empresa. Los atributos son:

* cod\_empleado: Código único del empleado.
* dni: Documento de Identidad del empleado.
* estado: Estado del empleado (activo o inactivo).

**Tabla de Vendedores**

La tabla Vendedores almacena los registros específicos de los vendedores. Los atributos son:

* cod\_vendedor: Código único del vendedor.
* cod\_empleado: Código del empleado asociado al vendedor.
* rol: El rol del vendedor dentro de la empresa.

**Tabla de Asesores**

La tabla Asesores almacena los registros de los asesores de ventas. Los atributos son:

* cod\_asesor: Código único del asesor.
* cod\_empleado: Código del empleado asociado al asesor.
* experiencia: Años de experiencia del asesor.
* especialidad: Especialidad del asesor.

**Tabla de Clientes**

La tabla Clientes almacena la información de los clientes. Los atributos son:

* dni: Documento de Identidad del cliente.
* tipo\_cliente: Tipo de cliente (normal o frecuente).

**Tabla de Contratos**

La tabla Contratos almacena los contratos de los empleados. Los atributos son:

* cod\_contrato: Código único del contrato.
* cod\_empleado: Código del empleado asociado al contrato.
* fecha\_inicio: Fecha de inicio del contrato.
* fecha\_fin: Fecha de finalización del contrato.
* salario\_men: Salario mensual del empleado.
* observaciones: Notas adicionales sobre el contrato.
* estado: Estado del contrato (activo o inactivo).

**Tabla de Proveedores**

La tabla Proveedores almacena información sobre los proveedores. Los atributos son:

* ruc: Número de Registro Único de Contribuyente (RUC) del proveedor.
* nombre: Nombre del proveedor.
* fecha\_registro: Fecha de registro del proveedor.

**Tabla de Teléfonos de Proveedores**

La tabla Telefonos\_Proveedores almacena los números de teléfono asociados a cada proveedor. Los atributos incluyen:

* ruc: RUC del proveedor.
* telefono: Número de teléfono del proveedor.

**Tabla de Categorías**

La tabla Categorias almacena las categorías de productos. Los atributos son:

* cod\_categoria: Código único de la categoría.
* nombre: Nombre de la categoría.
* descripcion: Descripción de la categoría.
* fecha\_registro: Fecha de registro de la categoría.

**Tabla de Productos**

La tabla Productos almacena información sobre los productos que se venden. Los atributos son:

* cod\_producto: Código único del producto.
* cod\_categoria: Código de la categoría a la que pertenece el producto.
* ruc: RUC del proveedor.
* nombre: Nombre del producto.
* linea: Línea del producto.
* descripcion: Descripción del producto.
* precio\_compra: Precio de compra del producto.
* precio\_venta: Precio de venta del producto.
* stock: Cantidad disponible en stock.
* estado: Estado del producto (disponible o agotado).
* fecha\_registro: Fecha de registro del producto.

**3.2. Carga de datos**

Una vez creada la base de datos FabiaNatura y establecidas sus tablas, es importante cargar los datos necesarios para que el sistema funcione correctamente. Este proceso de carga de datos incluye la inserción de información en varias tablas clave: Personas, Empleados, Vendedores, Asesores, Clientes, Contratos, Proveedores, Telefonos\_Personas, Direcciones\_Personas, entre otras. A continuación, se explica cómo se realiza la carga de datos en cada una de estas tablas.

**3.2.1. Carga de Datos en la Tabla Personas**

La tabla Personas contiene la información básica de todas las personas involucradas en el sistema, tales como empleados, clientes y proveedores. Para cada persona, se debe registrar:

DNI: Documento Nacional de Identidad.

Nombre completo: Incluye nombre, apellido paterno y apellido materno.

Fecha de nacimiento: Fecha en la que la persona nació.

Ejemplo de carga de datos en la tabla Personas:

* Valeria Infanzón Cantoral, con DNI 89281109, nacida el 22 de febrero de 1980.
* Mario Perez Videla, con DNI 78291222, nacido el 12 de junio de 1990.
* Maria Gomez Torrez, con DNI 71827188, nacida el 11 de octubre de 2002.

**3.2.2. Carga de Datos en la Tabla Telefonos\_Personas**

La tabla Telefonos\_Personas registra los números telefónicos asociados a cada persona. Cada entrada en esta tabla tiene:

Número de teléfono: Un número único de teléfono que se asocia a la persona registrada en la tabla Personas.

Por ejemplo, se pueden asociar los siguientes números telefónicos a las personas:

* El teléfono 982819222 corresponde a Valeria Infanzón.
* El teléfono 982712212 corresponde a Mario Perez.
* El teléfono 982918223 corresponde a Maria Gomez.

**3.2.3. Carga de Datos en la Tabla Direcciones\_Personas**

La tabla Direcciones\_Personas almacena las direcciones de las personas. Cada entrada incluye:

DNI de la persona a la que se le asigna la dirección.

Dirección: Dirección física de la persona (puede ser una calle, número, etc.).

Ejemplo de carga de direcciones:

* Valeria Infanzón tiene la dirección Pj pepe 209.
* Mario Perez tiene la dirección Calle Callao 409.
* Maria Gomez tiene la dirección Calle Callao 902.

**3.2.4. Carga de Datos en la Tabla Empleados**

La tabla Empleados registra la información sobre los empleados, que incluye tanto a los vendedores como a los asesores. Se cargan los siguientes datos:

DNI: El mismo que se encuentra en la tabla Personas.

Estado: Si el empleado está activo o inactivo.

Ejemplo de carga de empleados:

* Valeria Infanzón (vendedora) está activa.
* Mario Perez (asesor) está activo.

**3.2.5. Carga de Datos en la Tabla Vendedores**

La tabla Vendedores almacena información específica de los vendedores, como su rol dentro de la empresa. En este caso:

Rol: El rol que desempeña el vendedor (por ejemplo, supervisor).

Por ejemplo, Valeria Infanzón es registrada como vendedora con el rol de supervisor.

**3.2.6. Carga de Datos en la Tabla Asesores**

La tabla Asesores almacena la información relacionada con los asesores. En esta tabla, se cargan:

Experiencia: Los años de experiencia del asesor.

Especialidad: El área en la que el asesor se especializa.

Por ejemplo, Mario Perez es registrado como asesor con 5 años de experiencia y con la especialidad en Dermatología.

**3.2.7. Carga de Datos en la Tabla Clientes**

La tabla Clientes contiene información sobre los clientes que realizan compras o utilizan los servicios de la empresa. Se cargan los siguientes datos:

DNI: El mismo que aparece en la tabla Personas.

Tipo de cliente: Puede ser regular o frecuente.

Por ejemplo, Maria Gomez es registrada como cliente regular.

**3.2.8. Carga de Datos en la Tabla Contratos**

La tabla Contratos almacena información sobre los contratos de los empleados (vendedores y asesores). Los datos que se cargan son:

Código de empleado: El código que se asigna al empleado.

Fecha de inicio y fin: Las fechas de inicio y fin del contrato.

Salario mensual: El salario que recibe el empleado.

Observaciones: Detalles adicionales sobre el contrato.

Estado: Si el contrato está activo o inactivo.

Por ejemplo, se registra un contrato para Valeria Infanzón con un salario mensual de 2000.00 y un contrato activo. Lo mismo se realiza para Mario Perez, con un salario mensual de 2500.00.

**3.2.9. Carga de Datos en la Tabla Proveedores**

La tabla Proveedores almacena información sobre los proveedores de productos. Los datos que se cargan incluyen:

RUC: El número de RUC del proveedor.

Nombre: El nombre de la empresa proveedora.

Por ejemplo, se registran los proveedores:

* NATURA COSMETICOS S.A., con RUC 20101796532.
* PRODUCTOS AVON S.A., con RUC 20100078792.
* ESIKA COSMETICS PERU S.A.C., con RUC 20517667502.

**3.2.10. Carga de Datos en la Tabla Telefonos\_Proveedores**

La tabla Telefonos\_Proveedores almacena los números de teléfono asociados a cada proveedor. Cada entrada tiene:

RUC: El número de RUC del proveedor.

Teléfono: El número de contacto del proveedor.

Por ejemplo:

* NATURA COSMETICOS S.A. tiene el teléfono 440-1362.
* PRODUCTOS AVON S.A. tiene el teléfono 317-2866.
* ESIKA COSMETICS PERU S.A.C. tiene el teléfono 0801-1-3030.

**3.2.11. Carga de Datos en la Tabla Categorias**

La tabla Categorias almacena las diferentes categorías de productos. Los datos incluyen:

Nombre de la categoría: El nombre de la categoría, como Maquillaje, Fragancias, Cuidado de la piel, etc.

Descripción: Una breve descripción de la categoría de productos.

Por ejemplo, las categorías cargadas pueden ser:

* Maquillaje: Productos diseñados para embellecer el rostro.
* Fragancias: Perfumes y colonias para hombres y mujeres.
* Cuidado de la piel: Productos para mantener la piel saludable.
* Cuidado del cabello: Productos para limpiar e hidratar el cabello.

**3.3. Diseño de funciones**

A continuación, se describirán las funciones desarrolladas para el sistema FabiaNatura:

**3.3.1 CalcularSubtotal**

Propósito: Esta función calcula el subtotal de una factura, que es el total de los productos vendidos multiplicado por su cantidad en el detalle de la factura.

Entrada: Recibe un cod\_factura (ID de la factura).

Proceso:

* Realiza una suma de los productos (precio de venta \* cantidad) para cada detalle de la factura.
* Se asegura de que si no hay resultados, el subtotal sea 0.
* Salida: Retorna el subtotal de la factura.

CREATE FUNCTION CalcularSubtotal(cod\_factura INT)

RETURNS FLOAT

DETERMINISTIC

BEGIN

DECLARE subtotal FLOAT;

SELECT SUM(p.precio\_venta \* df.cantidad) INTO subtotal

FROM Detalle\_Facturas df

JOIN Productos p ON df.cod\_producto = p.cod\_producto

WHERE df.cod\_factura = cod\_factura;

RETURN IFNULL(subtotal, 0);

END;

**3.3.2. VerificarTelefonoExisteProveedores**

Propósito: Verifica si un teléfono ya está registrado en la tabla de telefonos de proveedores.

Entrada: Recibe un número de teléfono (p\_telefono).

Proceso:

* Realiza una consulta COUNT para contar cuántas veces aparece el teléfono en la tabla.
* Si el teléfono está presente, devuelve 1; de lo contrario, devuelve 0.
* Salida: Retorna 1 si el teléfono existe, o 0 si no.

CREATE FUNCTION VerificarTelefonoExisteProveedores(p\_telefono VARCHAR(9))

RETURNS TINYINT

DETERMINISTIC

BEGIN

DECLARE existe INT;

SELECT COUNT(\*) INTO existe

FROM Telefonos\_Proveedores

WHERE telefono = p\_telefono;

RETURN IF(existe > 0, 1, 0);

END;

**3.3.3. VerificarPersonaExiste**

Propósito: Verifica si una persona (identificada por su DNI) existe en la tabla Personas.

Entrada: Recibe un DNI (p\_dni).

Proceso:

* Realiza una consulta COUNT para verificar si existe algún registro con ese DNI.
* Salida: Retorna TRUE si la persona existe, o FALSE si no.

CREATE FUNCTION VerificarPersonaExiste(p\_dni VARCHAR(8))

RETURNS BOOLEAN

DETERMINISTIC

BEGIN

DECLARE existe INT;

SELECT COUNT(\*) INTO existe

FROM Personas

WHERE dni = p\_dni;

RETURN existe > 0;

END;

**3.3.4. VerificarTelefonoExiste**

Propósito: Similar a la función anterior, pero esta busca el teléfono en la tabla Telefonos\_Personas, que parece estar relacionada con los números de teléfono de personas.

Entrada: Recibe un número de teléfono (p\_telefono).

Proceso:

* Realiza una consulta COUNT para contar las veces que aparece el teléfono en la tabla.
* Salida: Retorna 1 si el teléfono existe, o 0 si no.

CREATE FUNCTION VerificarTelefonoExiste(p\_telefono VARCHAR(9))

RETURNS TINYINT

DETERMINISTIC

BEGIN

DECLARE existe INT;

SELECT COUNT(\*) INTO existe

FROM Telefonos\_Personas

WHERE telefono = p\_telefono;

RETURN IF(existe > 0, 1, 0);

END;

**3.3.5. ObtenerCodEmpleado**

Propósito: Obtiene el código de un empleado a partir de su DNI.

Entrada: Recibe un DNI (p\_dni).

Proceso:

* Realiza un JOIN entre las tablas Personas y Empleados para obtener el cod\_empleado asociado al DNI proporcionado.
* Salida: Retorna el código de empleado correspondiente.

CREATE FUNCTION ObtenerCodEmpleado(p\_dni VARCHAR(8))

RETURNS INT

DETERMINISTIC

BEGIN

DECLARE cod\_empleado INT;

SELECT e.cod\_empleado

INTO cod\_empleado

FROM Personas p

JOIN Empleados e ON p.dni = e.dni

WHERE p.dni = p\_dni

LIMIT 1;

RETURN cod\_empleado;

END;

**3.3.6. VerificarVendedor**

Propósito: Verifica si un vendedor existe en la tabla Vendedores utilizando su código.

Entrada: Recibe un código de vendedor (p\_cod\_vendedor).

Proceso:

* Realiza una consulta COUNT para verificar si existe un vendedor con ese código en la tabla.
* Salida: Retorna 1 si el vendedor existe, o 0 si no.

CREATE FUNCTION VerificarVendedor(p\_cod\_vendedor INT)

RETURNS INT

DETERMINISTIC

BEGIN

DECLARE vendedor\_existente INT;

SELECT COUNT(\*) INTO vendedor\_existente

FROM Vendedores

WHERE cod\_vendedor = p\_cod\_vendedor;

IF vendedor\_existente > 0 THEN

RETURN 1;

ELSE

RETURN 0;

END IF;

END;

**3.3.7. VerificarAsesor**

Propósito: Verifica si un asesor existe en la tabla Asesores usando su código.

Entrada: Recibe un código de asesor (p\_cod\_asesor).

Proceso:

* Realiza una consulta EXISTS para verificar si el código de asesor existe en la tabla.
* Salida: Retorna 1 si el asesor existe, o 0 si no.

CREATE FUNCTION VerificarAsesor(p\_cod\_asesor INT)

RETURNS INT

DETERMINISTIC

BEGIN

DECLARE resultado INT;

IF EXISTS (SELECT 1 FROM Asesores WHERE cod\_asesor = p\_cod\_asesor) THEN

SET resultado = 1;

ELSE

SET resultado = 0;

END IF;

RETURN resultado;

END;

**3.3.8. ObtenerUltimaFactura**

Propósito: Obtiene el código de la última factura creada en la tabla Facturas.

Entrada: No recibe parámetros.

Proceso:

* Realiza una consulta MAX sobre el campo cod\_factura para obtener el mayor valor (la última factura).
* Salida: Retorna el código de la última factura.

CREATE FUNCTION ObtenerUltimaFactura()

RETURNS INT

DETERMINISTIC

BEGIN

DECLARE ultimo\_id INT;

SELECT MAX(cod\_factura) INTO ultimo\_id FROM Facturas;

RETURN ultimo\_id;

END;

**3.4. Diseño de procedimientos almacenados**

A continuación, se detallarán los procedimientos almacenados más relevanntes en nuestro sistema FabiaNatura:

**3.4.1. InsertarDetalleFactura**

Propósito: Insertar un nuevo detalle de factura en la tabla Detalle\_Facturas.

Entrada: p\_cod\_factura (código de la factura), p\_cod\_producto (código del producto), p\_cantidad (cantidad del producto).

Proceso: Verifica que la factura y el producto existan, y que la cantidad sea mayor que cero. Si las condiciones se cumplen, inserta el detalle en la tabla.

DELIMITER $$

CREATE PROCEDURE InsertarDetalleFactura(

IN p\_cod\_factura INT,

IN p\_cod\_producto INT,

IN p\_cantidad INT

)

BEGIN

IF NOT EXISTS (SELECT 1 FROM Facturas WHERE cod\_factura = p\_cod\_factura) THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'La factura con el código proporcionado no existe.';

END IF;

IF NOT EXISTS (SELECT 1 FROM Productos WHERE cod\_producto = p\_cod\_producto) THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'El producto con el código proporcionado no existe.';

END IF;

IF p\_cantidad <= 0 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'La cantidad debe ser un número mayor que 0.';

END IF;

INSERT INTO Detalle\_Facturas (cod\_factura, cod\_producto, cantidad) VALUES

(p\_cod\_factura, p\_cod\_producto, p\_cantidad);

END$$

**3.4.2. EditarStockProducto**

Propósito: Actualizar el stock de un producto en la tabla Productos.

Entrada: p\_cod\_producto (código del producto), p\_stock (nuevo stock).

Proceso: Verifica que el stock no sea negativo. Si es válido, actualiza el stock y el estado del producto.

DELIMITER $$

CREATE PROCEDURE EditarStockProducto(

IN p\_cod\_producto INT,

IN p\_stock INT

)

BEGIN

IF p\_stock < 0 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'El stock no puede ser negativo.';

END IF;

UPDATE Productos

SET stock = p\_stock,

estado = IF(p\_stock > 0, 'disponible', 'agotado')

WHERE cod\_producto = p\_cod\_producto;

END$$

**3.4.3. EditarProveedorProducto**

Propósito: Cambiar el proveedor de un producto.

Entrada: p\_cod\_producto (código del producto), p\_ruc (RUC del nuevo proveedor).

Proceso: Verifica que el producto y el proveedor existan. Si ambos son válidos, actualiza el proveedor del producto.

DELIMITER $$

CREATE PROCEDURE EditarProveedorProducto(

IN p\_cod\_producto INT,

IN p\_ruc VARCHAR(11)

)

BEGIN

IF NOT EXISTS (SELECT 1 FROM Productos WHERE cod\_producto = p\_cod\_producto) THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'El producto con el código proporcionado no existe.';

END IF;

IF NOT EXISTS (SELECT 1 FROM Proveedores WHERE ruc = p\_ruc) THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'El proveedor con el RUC proporcionado no existe.';

END IF;

UPDATE Productos SET ruc\_proveedor = p\_ruc WHERE cod\_producto = p\_cod\_producto;

END$$

**3.4.4. AgregarFactura**

Propósito: Agregar una nueva factura a la tabla Facturas.

Entrada: p\_dni (DNI del cliente), p\_cod\_vendedor (código del vendedor), p\_cod\_asesor (código del asesor).

Proceso: Verifica que el cliente, vendedor y asesor (si se proporciona) existan. Si son válidos, inserta la nueva factura.

DELIMITER $$

CREATE PROCEDURE AgregarFactura(

IN p\_dni VARCHAR(8),

IN p\_cod\_vendedor INT,

IN p\_cod\_asesor INT

)

BEGIN

IF NOT EXISTS (SELECT 1 FROM Clientes WHERE dni = p\_dni) THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'El cliente con el DNI proporcionado no existe.';

END IF;

IF NOT EXISTS (SELECT 1 FROM Vendedores WHERE cod\_vendedor = p\_cod\_vendedor) THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'El vendedor con el código proporcionado no existe.';

END IF;

IF p\_cod\_asesor IS NOT NULL AND NOT EXISTS (SELECT 1 FROM Asesores WHERE cod\_asesor = p\_cod\_asesor) THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'El asesor con el código proporcionado no existe.';

END IF;

INSERT INTO Facturas (dni, cod\_vendedor, cod\_asesor) VALUES (p\_dni, p\_cod\_vendedor, p\_cod\_asesor);

END$$

**3.4.5. EliminarFactura**

Propósito: Eliminar una factura de la tabla Facturas.

Entrada: p\_cod\_factura (código de la factura a eliminar).

Proceso: Verifica que la factura exista y, si es así, la elimina de la tabla.

DELIMITER $$

CREATE PROCEDURE EliminarFactura(

IN p\_cod\_factura INT

)

BEGIN

IF NOT EXISTS (SELECT 1 FROM Facturas WHERE cod\_factura = p\_cod\_factura) THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'La factura con el código proporcionado no existe.';

END IF;

DELETE FROM Facturas WHERE cod\_factura = p\_cod\_factura;

END$$

**3.4.6. CancelarFactura**

Propósito: Cancelar una factura y restaurar el stock de los productos asociados.

Entrada: p\_cod\_factura (código de la factura a cancelar).

Proceso: Verifica que la factura exista, recupera los detalles de la factura, actualiza el stock de los productos y elimina la factura y sus detalles.

DELIMITER $$

CREATE PROCEDURE CancelarFactura(

IN p\_cod\_factura INT

)

BEGIN

DECLARE done INT DEFAULT 0;

DECLARE cod\_producto INT;

DECLARE cantidad INT;

DECLARE cur CURSOR FOR

SELECT cod\_producto, cantidad

FROM Detalle\_Facturas

WHERE cod\_factura = p\_cod\_factura;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

IF NOT EXISTS (SELECT 1 FROM Facturas WHERE cod\_factura = p\_cod\_factura) THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'La factura con el código proporcionado no existe.';

END IF;

OPEN cur;

read\_loop: LOOP

FETCH cur INTO cod\_producto, cantidad;

IF done THEN

LEAVE read\_loop;

END IF;

UPDATE Productos

SET stock = stock + cantidad

WHERE cod\_producto = cod\_producto;

END LOOP;

CLOSE cur;

DELETE FROM Detalle\_Facturas WHERE cod\_factura = p\_cod\_factura;

DELETE FROM Facturas WHERE cod\_factura = p\_cod\_factura;

END$$

**3.4.7. MostrarBoletaVenta**

Propósito: Mostrar los detalles de una boleta de venta.

Entrada: p\_cod\_factura (código de la factura).

Proceso: Selecciona y muestra el nombre del producto, cantidad, precio unitario y total del producto para la factura especificada.

DELIMITER $$

CREATE PROCEDURE MostrarBoletaVenta(

IN p\_cod\_factura INT

)

BEGIN

SELECT

p.nombre AS Producto,

df.cantidad AS Cantidad,

ROUND(p.precio\_venta, 2) AS PrecioUnitario,

ROUND(p.precio\_venta \* df.cantidad, 2) AS TotalProducto

FROM

Detalle\_Facturas df

JOIN

Productos p ON df.cod\_producto = p.cod\_producto

WHERE

df.cod\_factura = p\_cod\_factura;

END$$

**3.5. Diseño de vistas - consultas simples y avanzadas**

A continuación, se explica las consultas simples y avanzadas más importantes del sistema FabiaNatura.

**3.5.1. MostrarCategorias**

Propósito: Mostrar todas las categorías disponibles en el sistema.

Entrada: Tabla Categorias.

Proceso: Selecciona el código y nombre de cada categoría y los ordena por código de categoría.

CREATE VIEW MostrarCategorias AS

SELECT

cod\_categoria AS Codigo,

nombre AS Nombre

FROM Categorias

ORDER BY cod\_categoria ASC;

**3.5.2. MostrarCategoriasConProductosDisponibles**

Propósito: Mostrar las categorías que tienen productos disponibles.

Entrada: Tablas Categorias y Productos.

Proceso: Realiza un conteo de productos disponibles por categoría y los ordena de mayor a menor cantidad.

CREATE VIEW MostrarCategoriasConProductosDisponibles AS

SELECT

c.cod\_categoria AS CodigoCategoria,

c.nombre AS NombreCategoria,

COUNT(p.cod\_producto) AS ProductosDisponibles

FROM

Categorias c

JOIN

Productos p ON c.cod\_categoria = p.cod\_categoria

WHERE

p.estado = 'disponible'

GROUP BY

c.cod\_categoria, c.nombre

ORDER BY

ProductosDisponibles DESC;

**3.5.3 . MostrarProductosMasVendidos**

Propósito: Mostrar los productos que más se han vendido.

Entrada: Tablas Productos y Detalle\_Facturas.

Proceso: Suma la cantidad vendida de cada producto y calcula los ingresos totales, ordenando por cantidad vendida.

CREATE VIEW MostrarProductosMasVendidos AS

SELECT

p.cod\_producto AS CodigoProducto,

p.nombre AS NombreProducto,

SUM(df.cantidad) AS TotalVendidos,

FORMAT(SUM(df.cantidad \* p.precio\_venta), 2) AS TotalIngresos

FROM

Productos p

INNER JOIN

Detalle\_Facturas df ON p.cod\_producto = df.cod\_producto

GROUP BY

p.cod\_producto, p.nombre

ORDER BY

TotalVendidos DESC;

**3.5.4. MostrarProductosMenosVendidos**

Propósito: Mostrar los productos que menos se han vendido.

Entrada: Tablas Productos y Detalle\_Facturas.

Proceso: Realiza un conteo de la cantidad vendida y los ingresos totales, mostrando productos con cero ventas también.

CREATE VIEW MostrarProductosMenosVendidos AS

SELECT

p.cod\_producto AS CodigoProducto,

p.nombre AS NombreProducto,

IFNULL(SUM(df.cantidad), 0) AS TotalVendidos,

FORMAT(IFNULL(SUM(df.cantidad \* p.precio\_venta), 0), 2) AS TotalIngresos

FROM

Productos p

LEFT JOIN

Detalle\_Facturas df ON p.cod\_producto = df.cod\_producto

GROUP BY

p.cod\_producto, p.nombre

ORDER BY

TotalVendidos ASC;

**3.5.5. MostrarCategoriasConMasProductos**

Propósito: Mostrar las categorías que tienen la mayor cantidad de productos.

Entrada: Tablas Categorias y Productos.

Proceso: Cuenta el número de productos por categoría y los ordena de mayor a menor.

CREATE VIEW MostrarCategoriasConMasProductos AS

SELECT

c.cod\_categoria AS CodigoCategoria,

c.nombre AS NombreCategoria,

COUNT(p.cod\_producto) AS TotalProductos

FROM

Categorias c

LEFT JOIN

Productos p ON c.cod\_categoria = p.cod\_categoria

GROUP BY

c.cod\_categoria, c.nombre

ORDER BY

TotalProductos DESC;

**3.5.6. MostrarProductosAgotados**

Propósito: Mostrar los productos que están agotados.

Entrada: Tabla Productos.

Proceso: Filtra los productos cuyo estado es 'agotado' y los ordena por código de producto.

CREATE VIEW MostrarProductosAgotados AS

SELECT

cod\_producto AS Codigo,

nombre AS Nombre,

linea AS Linea,

precio\_venta AS Precio,

stock AS Disponible

FROM Productos

WHERE estado = 'agotado'

ORDER BY cod\_producto DESC;

**3.5.7. MostrarTopVendedores**

Propósito: Mostrar los vendedores con más ventas.

Entrada: Tablas Vendedores, Empleados, Facturas, Detalle\_Facturas, y Productos.

Proceso: Cuenta las ventas y calcula los ingresos generados por cada vendedor.

CREATE VIEW MostrarTopVendedores AS

SELECT

v.cod\_vendedor AS Codigo,

CONCAT(p.nombre, ' ', p.apellido\_paterno, ' ', p.apellido\_materno) AS Vendedor,

COUNT(f.cod\_factura) AS TotalVentas,

FORMAT(SUM(df.cantidad \* pr.precio\_venta), 2) AS TotalIngresosGenerados

FROM

Vendedores v

INNER JOIN

Empleados e ON v.cod\_empleado = e.cod\_empleado

INNER JOIN

Personas p ON e.dni = p.dni

INNER JOIN

Facturas f ON v.cod\_vendedor = f.cod\_vendedor

INNER JOIN

Detalle\_Facturas df ON f.cod\_factura = df.cod\_factura

INNER JOIN

Productos pr ON df.cod\_producto = pr.cod\_producto

GROUP BY

v.cod\_vendedor, Vendedor

ORDER BY

TotalVentas DESC;

**3.5.8. MostrarClientesAtendidosPorVendedor**

Propósito: Mostrar la cantidad de clientes únicos atendidos por cada vendedor.

Entrada: Tablas Vendedores, Empleados, Personas, y Facturas.

Proceso: Cuenta los clientes únicos por vendedor.

CREATE VIEW MostrarClientesAtendidosPorVendedor AS

SELECT

v.cod\_vendedor AS CodigoVendedor,

CONCAT(p.nombre, ' ', p.apellido\_paterno, ' ', p.apellido\_materno) AS NombreVendedor,

COUNT(DISTINCT f.dni) AS ClientesUnicos

FROM

Vendedores v

INNER JOIN

Empleados e ON v.cod\_empleado = e.cod\_empleado

INNER JOIN

Personas p ON e.dni = p.dni

INNER JOIN

Facturas f ON v.cod\_vendedor = f.cod\_vendedor

GROUP BY

v.cod\_vendedor, NombreVendedor;

**3.5.9. MostrarCategoriasConProductosMasVendidos**

Propósito: Mostrar las categorías que tienen los productos más vendidos.

Entrada: Tablas Categorias, Productos, Detalle\_Facturas.

Proceso: Realiza un conteo de productos vendidos por categoría.

CREATE VIEW MostrarCategoriasConProductosMasVendidos AS

SELECT

c.cod\_categoria AS CodigoCategoria,

c.nombre AS NombreCategoria,

SUM(df.cantidad) AS TotalVendidos

FROM

Categorias c

JOIN Productos p ON c.cod\_categoria = p.cod\_categoria

JOIN Detalle\_Facturas df ON p.cod\_producto = df.cod\_producto

GROUP BY

c.cod\_categoria, c.nombre

ORDER BY

TotalVendidos DESC;

**3.5.10. MostrarProductosVendidos**

Propósito: Mostrar los productos vendidos con su cantidad y monto recaudado.

Entrada: Tablas Detalle\_Facturas, Productos, y Categorias.

Proceso: Suma la cantidad vendida y calcula el total recaudado por producto.

CREATE VIEW MostrarProductosVendidos AS

SELECT

p.cod\_producto AS CodigoProducto,

p.nombre AS NombreProducto,

c.nombre AS Categoria,

p.linea AS Linea,

SUM(df.cantidad) AS CantidadVendida,

ROUND(SUM(df.cantidad \* p.precio\_venta), 2) AS TotalVenta

FROM

Detalle\_Facturas df

JOIN Productos p ON df.cod\_producto = p.cod\_producto

LEFT JOIN Categorias c ON p.cod\_categoria = c.cod\_categoria

GROUP BY

p.cod\_producto, p.nombre, c.nombre, p.linea

ORDER BY

TotalVenta DESC;

**3.4. Diseño de triggers**

A continuación, se explica el trigger implementado en el sistema FabiaNatura

Propósito

El propósito del trigger ActualizarTipoClienteFrecuente es clasificar a los clientes como "frecuentes" en la tabla Clientes si han realizado 30 o más compras (facturas). Esto ayuda a identificar y gestionar mejor a los clientes que compran con regularidad.

Entrada

* Evento: Inserción de una nueva fila en la tabla Facturas.
* Datos de entrada: El dni del cliente asociado a la nueva factura.

Proceso

* Se activa el trigger después de que se inserta una nueva factura.
* Se cuenta el número total de facturas asociadas al dni del cliente de la nueva factura.
* Si el total de facturas es 30 o más, se actualiza el campo tipo\_cliente en la tabla Clientes para ese cliente, estableciéndolo como "frecuente".

DELIMITER $

CREATE TRIGGER ActualizarTipoClienteFrecuente

AFTER INSERT ON Facturas

FOR EACH ROW

BEGIN

DECLARE total\_facturas INT;

SELECT COUNT(\*) INTO total\_facturas

FROM Facturas

WHERE dni = NEW.dni;

IF total\_facturas >= 30 THEN

UPDATE Clientes

SET tipo\_cliente = 'frecuente'

WHERE dni = NEW.dni;

END IF;

END $$

DELIMITER ;