INFORME BASE DE DATOS INVENTORY

ABEL AUDINO PANTOJA RODRIGUEZ

Presentado a Brayan Arcos

INSTITUTO TECNOLOGICO DEL PUTUMAYO

DESARROLLO DE BASE DE DATOS

MOCOA-PUTUMAYO

2024

INDICE

[RESUMEN EJECUTIVO 4](#_Toc180773121)

[INTRODUCCIÓN 5](#_Toc180773122)

[METODOLOGÍA 6](#_Toc180773123)

[RELACIÓN ENTRE TABLAS PRINCIPALES 7](#_Toc180773124)

[Relación de identificación y direcciones: 7](#_Toc180773125)

[Relaciones entre usuarios y roles: 7](#_Toc180773126)

[Relaciones entre productos y categorías: 7](#_Toc180773127)

[Relaciones en órdenes de compra: 7](#_Toc180773128)

[Relaciones en facturación y pagos: 8](#_Toc180773129)

[1. Gestión de Productos e Inventario: 8](#_Toc180773130)

[2. Gestión de Proveedores: 8](#_Toc180773131)

[3. Gestión de Clientes y Usuarios: 8](#_Toc180773132)

[4. Gestión de Ventas y Facturación: 9](#_Toc180773133)

[5. Cómo sería el flujo de trabajo: 9](#_Toc180773134)

[CREACIÓN DE BASE DE DATOS 9](#_Toc180773135)

[Requerimientos del Sistema 9](#_Toc180773136)

[**1. Requerimientos Funcionales** 9](#_Toc180773137)

[**2. Requerimientos No Funcionales** 10](#_Toc180773138)

[**3. Requerimientos de Integridad** 12](#_Toc180773139)

[**4. Requerimientos de Seguridad en Pagos** 12](#_Toc180773140)

[Modelo Conceptual 12](#_Toc180773141)

[Diseño Relacional 15](#_Toc180773142)

[Modelo entidad relación 16](#_Toc180773143)

[IMPLEMENTACIÓN EN SQL 17](#_Toc180773144)

[Cardinalidad de la base de datos 23](#_Toc180773145)

[Resumen Visual 24](#_Toc180773146)

[Consultas SQL con INNER JOIN 24](#_Toc180773147)

[Consultas SQL con LEFT JOIN 26](#_Toc180773148)

[Consultas SQL con RIGHT JOIN 27](#_Toc180773149)

[Subconsultas SQL con IN 28](#_Toc180773150)

[Subconsultas SQL con EXISTS 30](#_Toc180773151)

[Subconsultas SQL con ANY y ALL 31](#_Toc180773152)

[ANÁLISIS Y DISCUSIÓN 32](#_Toc180773153)

[Discusión 33](#_Toc180773154)

[CONCLUSIONES 35](#_Toc180773155)

[RECOMENDACIONES 36](#_Toc180773156)

[REFERENCIAS 37](#_Toc180773157)

# RESUMEN EJECUTIVO

Este documento presenta el diseño de la base de datos Inventory está destinado a optimizar la gestión de inventarios, órdenes de compra y la relación con proveedores, asegurando un sistema eficiente y libre de redundancias. La estructura se ha desarrollado siguiendo las mejores prácticas de normalización, garantizando integridad referencial y minimizando el almacenamiento innecesario. Se eliminan duplicidades entre personas y proveedores, y se simplifica la relación de roles de usuario. Los resultados esperados incluyen un rendimiento mejorado, menor redundancia de datos, y una administración más eficiente del inventario y proveedores. Se han identificado las entidades y sus relaciones, y se ha implementado una solución SQL que garantiza la integridad y escalabilidad del sistema. La base de datos diseñada incluye seis tablas principales: users, people, products, purchaseOrders, orderDetails, historyProductPrices.

Estas seis tablas son las principales porque capturan la información clave que el sistema necesita para gestionar usuarios, proveedores, productos y las transacciones de órdenes de compra.

# INTRODUCCIÓN

El desarrollo de este proceso lo encamino en el ámbito de aprendizaje teniendo como objetivo principal el desarrollo de la base de datos Inventory para proporcionar una estructura robusta para la gestión eficiente de usuarios, productos, proveedores y órdenes de compra en una organización. El diseño anterior incluía ciertas redundancias y complejidades que podrían dificultar la escalabilidad y el mantenimiento a largo plazo. Este documento detalla las mejoras implementadas para cumplir con las reglas de normalización y optimizar el uso de los recursos, manteniendo la integridad de los datos y evitando duplicidades.

# METODOLOGÍA

El diseño y creación de la base de datos siguió una metodología estructurada en tres fases principales:

1. Análisis de Requerimientos

Se identificaron las necesidades clave del sistema, definiendo las entidades principales: Proveedores, Productos, Órdenes de Compra, y Detalles de Órdenes de Compra. Se analizaron las relaciones entre estas entidades, identificando que cada proveedor puede suministrar múltiples productos, y que cada orden de compra está asociada a un proveedor y contiene varios productos con precios y cantidades específicas.

1. Diseño del Modelo Relacional

Basado en el análisis de requerimientos, se diseñó un modelo relacional que traduce las entidades y sus relaciones en tablas interrelacionadas. Cada tabla se diseñó con claves primarias y foráneas para asegurar la integridad referencial. Se incluyeron campos específicos como el precio y la cantidad de productos en cada orden de compra, con un enfoque en la normalización para evitar redundancias.

1. Implementación en SQL

Una vez definido el diseño, se procedió a la implementación en SQL utilizando MySQL Workbench 8.0 CE, en el cual se utilizó comandos CREATE DATABASE para crear la base de datos ***Inventario*** y CREATE TABLE para generar las tablas con sus relaciones. Las claves primarias y foráneas se implementaron para garantizar la integridad de los datos. Se realizaron pruebas iniciales mediante inserciones de datos y consultas básicas para validar la funcionalidad y consistencia de la base de datos.

1. Herramientas utilizadas

* MySQL Workbench
* Git Hub
* Draw.io

Para desarrollar esta base de datos, se siguieron los siguientes pasos:

* Análisis de las necesidades de la empresa: Identificar las entidades clave como usuarios, roles, proveedores, productos y órdenes de compra, asegurando que cada entidad esté representada en una tabla.
* Normalización: Aplicar las reglas de normalización (1NF, 2NF y 3NF) para evitar redundancias. Esto incluyó la revisión de las dependencias entre tablas y la eliminación de duplicidades. Por ejemplo, se fusionaron las tablas people y suppliers y se eliminó la tabla usersRoles para simplificar la relación entre usuarios y roles.
* Optimización de las relaciones: Se definieron claves foráneas claras entre las tablas para garantizar la integridad referencial. Se usaron columnas como isSupplier en la tabla people para identificar proveedores sin la necesidad de una tabla separada.
* Mejora de la eficiencia de consultas: Se eliminó la columna calculada totalPrice en la tabla orderDetails ya que se puede calcular dinámicamente mediante una consulta, evitando almacenamiento redundante.
* Implementación de timestamps: Se simplificó el uso de las columnas createdAt y updatedAt, aplicándolas solo donde es relevante hacer seguimiento de cambios (como productos y órdenes de compra).

# RELACIÓN ENTRE TABLAS PRINCIPALES

### Relación de identificación y direcciones:

1. **people ↔ identificationType** (Uno a Muchos):
   * La tabla people tiene una clave foránea (idType) que referencia a la tabla identificationType. Esto indica que cada persona tiene un tipo de identificación (por ejemplo, CC, CE, etc), pero un tipo de identificación puede estar asociado con múltiples personas.
2. **people ↔ addresses** (Uno a Muchos):
   * La tabla people también tiene una clave foránea (idAddress) que referencia a la tabla addresses. Cada persona puede tener una dirección registrada, y una dirección puede estar vinculada con varias personas.

### Relaciones entre usuarios y roles:

1. **usersRoles ↔ users & roles** (Muchos a Muchos):
   * La tabla usersRoles actúa como tabla intermedia para representar una relación muchos a muchos entre users y roles. Un usuario puede tener múltiples roles, y un rol puede aplicarse a múltiples usuarios.

### Relaciones entre productos y categorías:

1. **products ↔ productCategories** (Uno a Muchos):
   * La tabla products tiene una clave foránea (idCategory) que hace referencia a la tabla productCategories. Esto indica que un producto pertenece a una categoría específica, pero una categoría puede tener múltiples productos.
2. **historyProductPrices ↔ products** (Uno a Muchos):
   * La tabla historyProductPrices está relacionada con la tabla products mediante la clave foránea idProduct. Cada producto puede tener un historial de cambios de precios, pero cada registro de precios pertenece a un solo producto.

### Relaciones en órdenes de compra:

1. **purchaseOrders ↔ users** (Uno a Muchos):
   * La tabla purchaseOrders tiene una clave foránea (idSupplier) que referencia a la tabla users, representando al proveedor que realiza la orden de compra. Un proveedor (usuario) puede tener múltiples órdenes de compra asociadas.
2. **purchaseOrders ↔ orderStatus** (Uno a Muchos):
   * Cada orden de compra tiene un estado (por ejemplo, pendiente, completada), que está almacenado en la tabla orderStatus. Un estado puede aplicarse a varias órdenes de compra.
3. **orderDetails ↔ purchaseOrders & products** (Uno a Muchos):
   * La tabla orderDetails vincula las órdenes de compra (idPurchaseOrder) con los productos (idProduct). Esto permite detallar qué productos están incluidos en cada orden, en qué cantidad y a qué precio.

### Relaciones en facturación y pagos:

1. **userPaymentMethods ↔ users & paymentMethods** (Muchos a Muchos):
   * La tabla userPaymentMethods vincula a los usuarios con los métodos de pago disponibles. Un usuario puede tener varios métodos de pago, y un método de pago puede ser utilizado por varios usuarios.
2. **invoices ↔ users** (Uno a Muchos):
   * La tabla invoices tiene una clave foránea (idUser) que hace referencia a users, lo que indica que cada factura es generada por un usuario. Un usuario puede generar múltiples facturas.
3. **paymentGatewayRecords ↔ invoices** (Uno a Uno):
   * Cada registro de la tabla paymentGatewayRecords está vinculado a una factura específica mediante la clave foránea idInvoice. Esto guarda información sobre la transacción realizada a través de un gateway de pagos.

### 1. **Gestión de Productos e Inventario:**

* **Tablas involucradas**: products, productCategories, historyProductPrices
* Cada producto pertenece a una categoría (productCategories) y se guarda información detallada como su descripción y precios históricos en historyProductPrices.
* Al agregar nuevos productos, se insertan en la tabla products y se actualizan los precios en la tabla historyProductPrices.

### 2. **Gestión de Proveedores:**

* **Tablas involucradas**: users, people, addresses, purchaseOrders, orderDetails
* Los proveedores se gestionan en la tabla people, que almacena información personal y de contacto.
* Los pedidos a proveedores se registran en la tabla purchaseOrders, y los detalles de los productos pedidos se guardan en orderDetails.
* Cuando se realiza una compra, el pedido se genera en purchaseOrders, especificando los productos, cantidades y precios.

### 3. **Gestión de Clientes y Usuarios:**

* **Tablas involucradas**: users, people, roles, usersRoles, addresses
* Los usuarios (tanto empleados como clientes) se registran en la tabla users, y la tabla people guarda información personal detallada.
* Los roles de usuario (cliente, empleado, administrador) se gestionan en las tablas roles y usersRoles, permitiendo asignar permisos y accesos.
* Los clientes podrían registrarse como usuarios y asociarse con una dirección.

### 4. **Gestión de Ventas y Facturación:**

* **Tablas involucradas**: invoices, orderDetails, paymentMethods, paymentGatewayRecords
* Al realizar una venta, se crea una factura en invoices y se asocian los productos vendidos en la tabla orderDetails (similar al manejo de órdenes de compra).
* El total de la factura se calcula sumando los precios de los productos vendidos, almacenados en orderDetails con su cantidad y precio unitario.
* La tabla paymentMethods permite registrar diferentes métodos de pago (efectivo, transferencia, tarjeta de crédito/débito) asociados a usuarios.
* Para registrar pagos electrónicos, la tabla paymentGatewayRecords almacena la respuesta del gateway de pago y la referencia de la transacción.

### 5. **Cómo sería el flujo de trabajo:**

* **Ingreso de productos**: Un empleado administra los productos a través de la interfaz de inventario, insertando nuevos productos y actualizando precios en products y historyProductPrices.
* **Pedidos a proveedores**: Cuando faltan productos, se realiza un pedido a un proveedor registrado en purchaseOrders, y se registran los detalles del pedido en orderDetails.
* **Ventas a clientes**: En el momento de una venta, se crea una nueva factura en invoices, se asocian los productos vendidos en orderDetails y se genera el total de la venta.
* **Métodos de pago**: Dependiendo del método de pago, se actualiza paymentMethods y si es electrónico, los detalles se registran en paymentGatewayRecords con la respuesta del gateway de pago.
* **Facturación**: Cada venta genera una factura con el total a pagar, la cual se almacena en invoices y se puede relacionar con un método de pago, y en el caso de pagos electrónicos, se tiene el registro de la transacción en paymentGatewayRecords.

# CREACIÓN DE BASE DE DATOS

## Requerimientos del Sistema

**1. Requerimientos Funcionales**

Los **requerimientos funcionales** son aquellos que definen qué debe hacer el sistema. Estos están directamente relacionados con las acciones y procesos que el sistema debe soportar.

**1.1 Gestión de Usuarios**

* El sistema debe permitir registrar, modificar y eliminar usuarios.
* Los usuarios deben poder tener múltiples roles (e.g., administrador, cliente, proveedor).
* Los roles definen qué acciones puede realizar cada usuario (gestionar productos, visualizar facturas, etc.).

**1.2 Gestión de Personas**

* El sistema debe permitir almacenar la información personal de los usuarios, incluyendo el tipo de identificación y dirección.
* Debe soportar distintos tipos de identificación como cédulas, pasaportes o carnets.

**1.3 Gestión de Productos**

* El sistema debe permitir la creación, modificación y eliminación de productos.
* Los productos deben estar organizados en categorías para facilitar la búsqueda y clasificación.
* Se debe almacenar un historial de precios para los productos, permitiendo visualizar cambios históricos.

**1.4 Gestión de Órdenes de Compra**

* El sistema debe permitir la creación de órdenes de compra por parte de los proveedores.
* Cada orden debe tener un estado asociado (pendiente, completada, cancelada, etc.).
* El detalle de cada orden debe incluir productos, cantidades y precios unitarios.

**1.5 Facturación**

* El sistema debe generar facturas asociadas a las órdenes de compra completadas.
* Cada factura debe estar asociada a un usuario y registrar la fecha de emisión.
* Las facturas deben permitir registros de pagos, con detalles como el método de pago utilizado (efectivo, transferencia, tarjeta de crédito).

**1.6 Métodos de Pago y Gateway de Pagos**

* El sistema debe gestionar los métodos de pago disponibles (efectivo, tarjeta, transferencia, etc.).
* Cada usuario puede tener asociados varios métodos de pago.
* Se deben registrar todas las transacciones realizadas mediante el gateway de pagos, incluyendo la referencia y los datos de respuesta.

**2. Requerimientos No Funcionales**

Los **requerimientos no funcionales** son aquellos que definen cómo debe operar el sistema en términos de rendimiento, seguridad y mantenibilidad.

**2.1 Seguridad**

* El sistema debe cifrar las contraseñas de los usuarios.
* Solo los administradores deben poder acceder a la funcionalidad de gestión de usuarios, roles y productos.
* Las claves foráneas deben garantizar la integridad referencial, evitando que se borren o modifiquen registros que están en uso por otras tablas.

**2.2 Rendimiento**

* El sistema debe ser capaz de manejar cientos de productos y transacciones sin afectar el rendimiento.
* Las consultas a la base de datos deben ser optimizadas mediante índices en campos frecuentemente consultados como userName en la tabla de users y productName en la tabla products.

**2.3 Escalabilidad**

* El sistema debe ser escalable para permitir la adición de nuevas funcionalidades en el futuro, como el seguimiento de inventario en tiempo real o la integración con otros sistemas de gestión.

**2.4 Disponibilidad**

* La base de datos debe estar disponible en todo momento para garantizar la continuidad del negocio, especialmente en áreas críticas como facturación y procesamiento de órdenes de compra.
* Se recomienda implementar un sistema de respaldo periódico de la base de datos para evitar la pérdida de información.

**2.5 Usabilidad**

* Las consultas a la base de datos deben ser sencillas y eficientes para permitir que los usuarios accedan a la información que necesitan de manera rápida (e.g., consultar productos en órdenes, generar facturas).
* El sistema debe permitir una fácil incorporación de nuevos usuarios, proveedores y productos.

**2.6 Mantenibilidad**

* El diseño de la base de datos debe ser modular, facilitando la actualización y mantenimiento del sistema sin interrumpir las operaciones del negocio.
* Se debe permitir la adición de nuevos tipos de métodos de pago y roles sin necesidad de reestructurar la base de datos.

**3. Requerimientos de Integridad**

* Todas las relaciones entre tablas deben ser respetadas mediante claves foráneas para mantener la integridad referencial.
* Deben existir reglas claras para la eliminación y actualización de registros (uso de ON DELETE CASCADE o ON UPDATE CASCADE según corresponda).

**4. Requerimientos de Seguridad en Pagos**

* Los datos sensibles, como la respuesta del gateway de pagos, deben almacenarse de forma segura y encriptada.
* Se debe garantizar que las transacciones sean únicas mediante la generación de referencias únicas para cada transacción registrada en la tabla paymentGatewayRecords.

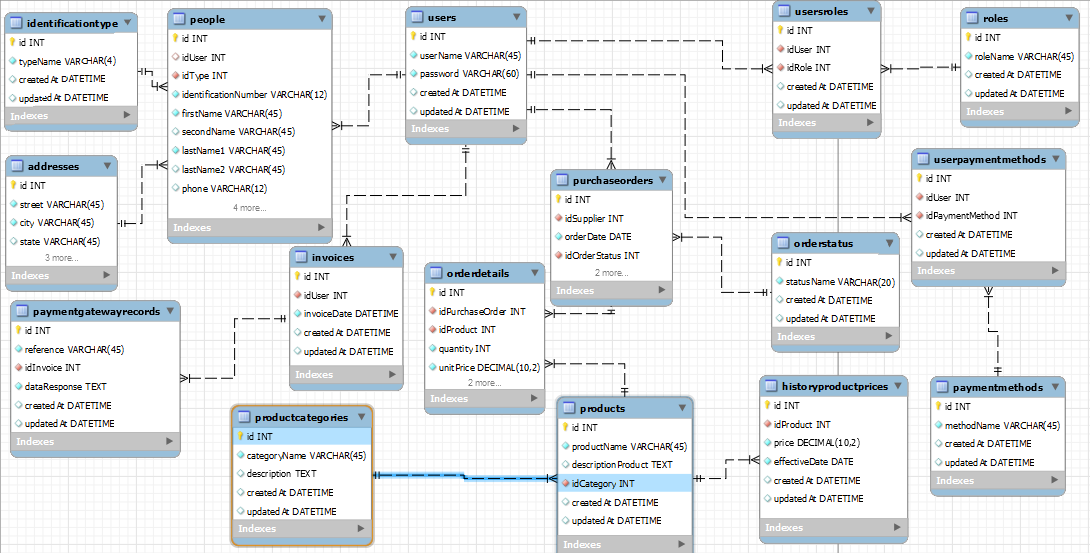
## Modelo Conceptual

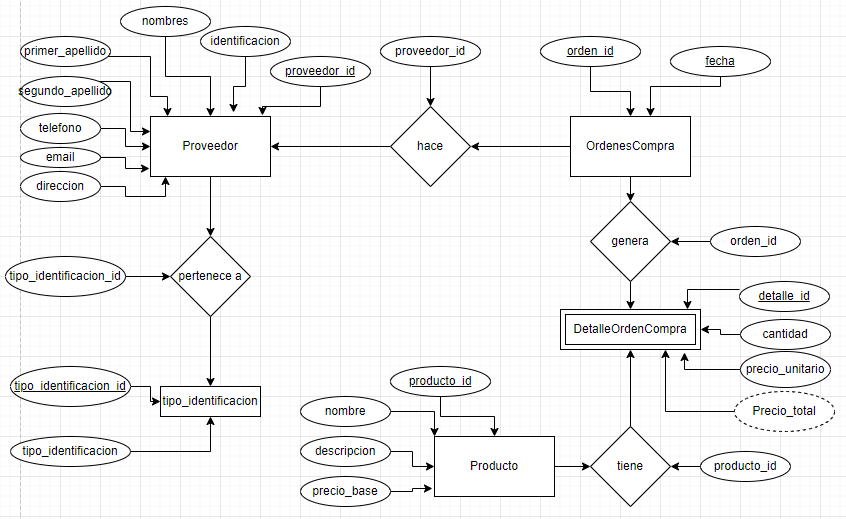
1. **Usuario (User)**
   * **Atributos:**
     + id (PK)
     + userName
     + password
     + createdAt
     + updatedAt
2. **Rol (Role)**
   * **Atributos:**
     + id (PK)
     + roleName
     + createdAt
     + updatedAt
3. **Persona (Person)**
   * **Atributos:**
     + id (PK)
     + idUser (FK)
     + idType (FK)
     + identificationNumber
     + firstName
     + secondName
     + lastName1
     + lastName2
     + phone
     + email
     + idAddress (FK)
     + createdAt
     + updatedAt
4. **Tipo de Identificación (IdentificationType)**
   * **Atributos:**
     + id (PK)
     + typeName
     + createdAt
     + updatedAt
5. **Dirección (Address)**
   * **Atributos:**
     + id (PK)
     + street
     + city
     + state
     + zipCode
     + createdAt
     + updatedAt
6. **Categoría de Producto (ProductCategory)**
   * **Atributos:**
     + id (PK)
     + categoryName
     + description
     + createdAt
     + updatedAt
7. **Producto (Product)**
   * **Atributos:**
     + id (PK)
     + productName
     + descriptionProduct
     + idCategory (FK)
     + createdAt
     + updatedAt
8. **Historial de Precios de Producto (HistoryProductPrice)**
   * **Atributos:**
     + id (PK)
     + idProduct (FK)
     + price
     + effectiveDate
     + createdAt
     + updatedAt
9. **Orden de Compra (PurchaseOrder)**
   * **Atributos:**
     + id (PK)
     + idSupplier (FK)
     + orderDate
     + idOrderStatus (FK)
     + createdAt
     + updatedAt
10. **Detalles de Orden (OrderDetail)**
    * **Atributos:**
      + id (PK)
      + idPurchaseOrder (FK)
      + idProduct (FK)
      + quantity
      + unitPrice
      + createdAt
      + updatedAt
11. **Estado de Orden (OrderStatus)**
    * **Atributos:**
      + id (PK)
      + statusName
      + createdAt
      + updatedAt
12. **Factura (Invoice)**
    * **Atributos:**
      + id (PK)
      + idUser (FK)
      + invoiceDate
      + total
      + createdAt
      + updatedAt
13. **Detalles de Factura (InvoiceDetail)**
    * **Atributos:**
      + id (PK)
      + idInvoice (FK)
      + idProduct (FK)
      + quantity
      + unitPrice
      + createdAt
      + updatedAt
14. **Método de Pago (PaymentMethod)**
    * **Atributos:**
      + id (PK)
      + methodName
      + createdAt
      + updatedAt
15. **Pago (Payment)**
    * **Atributos:**
      + id (PK)
      + idInvoice (FK)
      + idPaymentMethod (FK)
      + paymentDate
      + amount
      + status
      + createdAt
      + updatedAt
16. **Registros del Gateway de Pago (PaymentGatewayRecord)**
    * **Atributos:**
      + id (PK)
      + reference
      + idPayment (FK)
      + dataResponse
      + createdAt
      + updatedAt

## Diseño Relacional

* usersRoles.idUser → users.id
* usersRoles.idRole → roles.id
* people.idUser → users.id
* people.idType → identificationTypes.id
* people.idAddress → addresses.id
* products.idCategory → productCategories.id
* historyProductPrices.idProduct → products.id
* purchaseOrders.idSupplier → people.idUser
* purchaseOrders.idOrderStatus → orderStatus.id
* orderDetails.idPurchaseOrder → purchaseOrders.id
* orderDetails.idProduct → products.id
* invoices.idUser → users.id
* invoiceDetails.idInvoice → invoices.id
* invoiceDetails.idProduct → products.id
* payments.idInvoice → invoices.id
* payments.idPaymentMethod → paymentMethods.id
* paymentGatewayRecords.idPayment → payments.id

## Modelo entidad relación





# IMPLEMENTACIÓN EN SQL

**Tipos de Identificación (identificationType):**

* Esta tabla almacena los diferentes tipos de identificación que pueden tener las personas. Incluye campos para el ID, el nombre del tipo de identificación, y las fechas de creación y actualización.

CREATE TABLE identificationType (

id INT(20) PRIMARY KEY AUTO\_INCREMENT,

typeName VARCHAR(4) NOT NULL,

createdAt DATETIME,

updatedAt DATETIME

);

**Direcciones (addresses):**

* Contiene información sobre las direcciones, incluyendo la calle, ciudad, estado y código postal, junto con las fechas de creación y actualización.

CREATE TABLE addresses (

id INT(20) PRIMARY KEY AUTO\_INCREMENT,

street VARCHAR(45) NOT NULL,

city VARCHAR(45) NOT NULL,

state VARCHAR(45),

zipCode VARCHAR(10),

createdAt DATETIME,

updatedAt DATETIME

);

**Estado de Órdenes (orderStatus):**

* Almacena los diferentes estados que puede tener una orden de compra, como "Pendiente", "Completada", etc.

CREATE TABLE orderStatus (

id INT(20) PRIMARY KEY AUTO\_INCREMENT,

statusName VARCHAR(20) NOT NULL,

createdAt DATETIME,

updatedAt DATETIME

);

**Roles de Usuarios (roles):**

* Define los roles que pueden ser asignados a los usuarios, como "Admin", "Vendedor", etc.

CREATE TABLE roles (

id INT(20) PRIMARY KEY AUTO\_INCREMENT,

roleName VARCHAR(45) NOT NULL,

createdAt DATETIME,

updatedAt DATETIME

);

**Usuarios (users):**

* Registra la información de los usuarios del sistema, incluyendo su nombre de usuario y contraseña.

CREATE TABLE users (

id INT(20) PRIMARY KEY AUTO\_INCREMENT,

userName VARCHAR(45) NOT NULL UNIQUE,

`password` VARCHAR(60) NOT NULL,

createdAt DATETIME,

updatedAt DATETIME

);

**Relación entre Usuarios y Roles (usersRoles):**

* Tabla intermedia que permite asignar múltiples roles a un usuario.

CREATE TABLE usersRoles (

id INT(20) PRIMARY KEY AUTO\_INCREMENT,

idUser INT(20) NOT NULL,

idRole INT(20) NOT NULL,

createdAt DATETIME,

updatedAt DATETIME,

FOREIGN KEY (idUser) REFERENCES users(id),

FOREIGN KEY (idRole) REFERENCES roles(id)

);

**Personas (people):**

* Almacena la información personal de los usuarios, incluyendo su tipo de identificación y dirección.

CREATE TABLE people (

id INT(20) PRIMARY KEY AUTO\_INCREMENT,

idUser INT(20),

idType INT(20) NOT NULL,

identificationNumber VARCHAR(12) NOT NULL,

firstName VARCHAR(45) NOT NULL,

secondName VARCHAR(45),

lastName1 VARCHAR(45) NOT NULL,

lastName2 VARCHAR(45),

phone VARCHAR(12),

email VARCHAR(45),

idAddress INT(20),

createdAt DATETIME,

updatedAt DATETIME,

FOREIGN KEY (idUser) REFERENCES users(id),

FOREIGN KEY (idType) REFERENCES identificationType(id),

FOREIGN KEY (idAddress) REFERENCES addresses(id)

);

**Categorías de Productos (productCategories):**

* Define las categorías a las que pertenecen los productos, junto con una descripción opcional.

CREATE TABLE productCategories (

id INT(20) PRIMARY KEY AUTO\_INCREMENT,

categoryName VARCHAR(45) NOT NULL,

`description` TEXT,

createdAt DATETIME,

updatedAt DATETIME

);

**Productos (products):**

* Almacena información sobre los productos, incluyendo su nombre, descripción y categoría.

CREATE TABLE products (

id INT(20) PRIMARY KEY AUTO\_INCREMENT,

productName VARCHAR(45) NOT NULL,

descriptionProduct TEXT,

idCategory INT(20) NOT NULL,

createdAt DATETIME,

updatedAt DATETIME,

FOREIGN KEY (idCategory) REFERENCES productCategories(id)

);

**Historial de Precios de Productos (historyProductPrices):**

* Registra los precios históricos de los productos.

CREATE TABLE historyProductPrices (

id INT(20) PRIMARY KEY AUTO\_INCREMENT,

idProduct INT(20) NOT NULL,

price DECIMAL(10, 2) NOT NULL,

effectiveDate DATE NOT NULL,

createdAt DATETIME,

updatedAt DATETIME,

FOREIGN KEY (idProduct) REFERENCES products(id)

);

**Órdenes de Compra (purchaseOrders):**

* Contiene información sobre las órdenes de compra, incluyendo el proveedor y el estado de la orden.

CREATE TABLE purchaseOrders (

id INT(20) PRIMARY KEY AUTO\_INCREMENT,

idSupplier INT(20) NOT NULL,

orderDate DATE NOT NULL,

idOrderStatus INT(20) NOT NULL,

createdAt DATETIME,

updatedAt DATETIME,

FOREIGN KEY (idSupplier) REFERENCES users(id),

FOREIGN KEY (idOrderStatus) REFERENCES orderStatus(id)

);

**Detalles de Órdenes (orderDetails):**

* Almacena los detalles de cada orden de compra, incluyendo la cantidad y el precio unitario de los productos.

CREATE TABLE orderDetails (

id INT(20) PRIMARY KEY AUTO\_INCREMENT,

idPurchaseOrder INT(20) NOT NULL,

idProduct INT(20) NOT NULL,

quantity INT(20) NOT NULL,

unitPrice DECIMAL(10, 2) NOT NULL,

createdAt DATETIME,

updatedAt DATETIME,

FOREIGN KEY (idPurchaseOrder) REFERENCES purchaseOrders(id),

FOREIGN KEY (idProduct) REFERENCES products(id)

);

**Facturas (invoices):**

* Registra las facturas generadas por los usuarios, incluyendo la fecha y el total.

CREATE TABLE invoices (

id INT(20) PRIMARY KEY AUTO\_INCREMENT,

idUser INT(20) NOT NULL, -- Usuario que generó la factura

invoiceDate DATETIME NOT NULL, -- Fecha de emisión de la factura

total DECIMAL(10, 2) NOT NULL, -- Total de la factura

createdAt DATETIME,

updatedAt DATETIME,

FOREIGN KEY (idUser) REFERENCES users(id)

);

**Detalles de Facturas (invoiceDetails):**

* Contiene información sobre los productos incluidos en cada factura.

CREATE TABLE invoiceDetails (

id INT(20) PRIMARY KEY AUTO\_INCREMENT,

idInvoice INT(20) NOT NULL, -- Relación con la tabla de facturas

idProduct INT(20) NOT NULL, -- Producto facturado

quantity INT(20) NOT NULL, -- Cantidad del producto

unitPrice DECIMAL(10, 2) NOT NULL, -- Precio unitario del producto

createdAt DATETIME,

updatedAt DATETIME,

FOREIGN KEY (idInvoice) REFERENCES invoices(id),

FOREIGN KEY (idProduct) REFERENCES products(id)

);

**Métodos de Pago (paymentMethods):**

* Almacena los diferentes métodos de pago que pueden ser utilizados en las transacciones.

CREATE TABLE paymentMethods (

id INT(20) PRIMARY KEY AUTO\_INCREMENT,

methodName VARCHAR(45) NOT NULL, -- Efectivo, Transferencia, Tarjeta de crédito, etc.

createdAt DATETIME,

updatedAt DATETIME

);

**Pagos (payments):**

* Registra los pagos realizados, incluyendo el método de pago y el estado del mismo.

CREATE TABLE payments (

id INT(20) PRIMARY KEY AUTO\_INCREMENT,

idInvoice INT(20) NOT NULL, -- Relación con la tabla de facturas

idPaymentMethod INT(20) NOT NULL, -- Método de pago utilizado

paymentDate DATETIME NOT NULL, -- Fecha del pago

amount DECIMAL(10, 2) NOT NULL, -- Monto pagado

`status` VARCHAR(20) NOT NULL, -- Estado del pago (Ej. completado, pendiente)

createdAt DATETIME,

updatedAt DATETIME,

FOREIGN KEY (idInvoice) REFERENCES invoices(id),

FOREIGN KEY (idPaymentMethod) REFERENCES paymentMethods(id)

);

**Registros del Gateway de Pagos (paymentGatewayRecords):**

* Almacena la respuesta de las transacciones realizadas a través de un gateway de pago.

CREATE TABLE paymentGatewayRecords (

id INT(20) PRIMARY KEY AUTO\_INCREMENT,

`reference` VARCHAR(45) NOT NULL, -- Referencia de la transacción

idPayment INT(20) NOT NULL, -- Relación con la tabla de pagos

dataResponse TEXT NOT NULL, -- Respuesta completa del gateway de pago en formato JSON

createdAt DATETIME,

updatedAt DATETIME,

FOREIGN KEY (idPayment) REFERENCES payments(id)

);

Esta implementación de la base de datos establece una estructura organizada y relacional para gestionar la información de un sistema de inventario de tienda. Cada tabla está diseñada para cumplir una función específica y mantener la integridad de los datos mediante el uso de claves foráneas y relaciones entre tablas. Esto permite una gestión eficiente de usuarios, productos, órdenes, y pagos dentro del sistema.

## Cardinalidad de la base de datos

1. **identificationType** (1**──< N**)
   * **Descripción:** Un tipo de identificación puede estar asociado a muchas personas.
   * **Relación:** identificationType (id) a people (idType)
2. **addresses** (1**──< N**)
   * **Descripción:** Una dirección puede estar asociada a muchas personas.
   * **Relación:** addresses (id) a people (idAddress)
3. **orderStatus** (1**──< N**)
   * **Descripción:** Un estado de orden puede estar asociado a muchas órdenes de compra.
   * **Relación:** orderStatus (id) a purchaseOrders (idOrderStatus)
4. **roles** (1**──< N**)
   * **Descripción:** Un rol puede ser asignado a muchos usuarios.
   * **Relación:** roles (id) a usersRoles (idRole)
5. **users** (1**──< N**)
   * **Descripción:** Un usuario puede estar asociado a muchas personas.
   * **Relación:** users (id) a people (idUser)
6. **users** (1**──< N**)
   * **Descripción:** Un usuario puede crear muchas órdenes de compra.
   * **Relación:** users (id) a purchaseOrders (idSupplier)
7. **users** (1**──< N**)
   * **Descripción:** Un usuario puede generar muchas facturas.
   * **Relación:** users (id) a invoices (idUser)
8. **productCategories** (1**──< N**)
   * **Descripción:** Una categoría de producto puede incluir muchos productos.
   * **Relación:** productCategories (id) a products (idCategory)
9. **products** (1**──< N**)
   * **Descripción:** Un producto puede tener muchos precios históricos.
   * **Relación:** products (id) a historyProductPrices (idProduct)
10. **purchaseOrders** (1**──< N**)
    * **Descripción:** Una orden de compra puede tener muchos detalles de orden.
    * **Relación:** purchaseOrders (id) a orderDetails (idPurchaseOrder)
11. **products** (1**──< N**)
    * **Descripción:** Un producto puede aparecer en muchas facturas.
    * **Relación:** products (id) a invoiceDetails (idProduct)
12. **invoices** (1**──< N**)
    * **Descripción:** Una factura puede tener muchos detalles.
    * **Relación:** invoices (id) a invoiceDetails (idInvoice)
13. **invoices** (1**──< N**)
    * **Descripción:** Una factura puede tener muchos pagos asociados.
    * **Relación:** invoices (id) a payments (idInvoice)
14. **paymentMethods** (1**──< N**)
    * **Descripción:** Un método de pago puede estar asociado a muchos pagos.
    * **Relación:** paymentMethods (id) a payments (idPaymentMethod)
15. **payments** (1**──< N**)
    * **Descripción:** Un pago puede tener un solo registro de gateway, pero un registro de gateway puede corresponder a muchos pagos.
    * **Relación:** payments (id) a paymentGatewayRecords (idPayment)
16. **users** (M**──< N**)
    * **Descripción:** Un usuario puede tener muchos roles, y un rol puede ser asignado a muchos usuarios.
    * **Relación:** users (id) a usersRoles (idUser) y roles (id) a usersRoles (idRole)

### Resumen Visual

Para visualizar esta cardinalidad, normalmente se utilizarían diagramas ER (Entidades-Relaciones), que representan las entidades y sus relaciones. Aquí está un resumen textual:

* **identificationType (1) ──< (N) people**
* **addresses (1) ──< (N) people**
* **orderStatus (1) ──< (N) purchaseOrders**
* **roles (1) ──< (N) usersRoles**
* **users (1) ──< (N) people**
* **users (1) ──< (N) purchaseOrders**
* **users (1) ──< (N) invoices**
* **productCategories (1) ──< (N) products**
* **products (1) ──< (N) historyProductPrices**
* **purchaseOrders (1) ──< (N) orderDetails**
* **products (1) ──< (N) invoiceDetails**
* **invoices (1) ──< (N) invoiceDetails**
* **invoices (1) ──< (N) payments**
* **paymentMethods (1) ──< (N) payments**
* **payments (1) ──< (N) paymentGatewayRecords**
* **users (M) ──< (N) usersRoles >── (M) roles**

## Consultas SQL con INNER JOIN

-- \*\*\*\*\* Consultas con INNER JOIN \*\*\*\*\*

-- 1. Obtener todos los usuarios con sus roles correspondientes

SELECT u.userName, r.roleName

FROM users u

INNER JOIN usersRoles ur ON u.id = ur.idUser

INNER JOIN roles r ON ur.idRole = r.id;

-- 2. Listar los productos con sus categorías

SELECT p.productName, pc.categoryName

FROM products p

INNER JOIN productCategories pc ON p.idCategory = pc.id;

-- 3. Mostrar los detalles de las órdenes de compra junto con los nombres de los productos

SELECT od.idPurchaseOrder, p.productName, od.quantity, od.unitPrice

FROM orderDetails od

INNER JOIN products p ON od.idProduct = p.id;

-- 4. Obtener las órdenes de compra con su estado

SELECT po.id, po.orderDate, os.statusName

FROM purchaseOrders po

INNER JOIN orderStatus os ON po.idOrderStatus = os.id;

-- 5. Listar los productos con sus precios históricos

SELECT p.productName, hpp.price, hpp.effectiveDate

FROM products p

INNER JOIN historyProductPrices hpp ON p.id = hpp.idProduct;

-- 6. Obtener los proveedores que han realizado órdenes de compra

SELECT p.firstName, p.lastName1, po.orderDate

FROM people p

INNER JOIN purchaseOrders po ON p.idUser = po.idSupplier;

-- 7. Listar los usuarios junto con los detalles de su dirección

SELECT u.userName, a.street, a.city, a.zipCode

FROM users u

INNER JOIN people p ON u.id = p.idUser

INNER JOIN addresses a ON p.idAddress = a.id;

-- 8. Mostrar los precios actuales de los productos

SELECT p.productName, hpp.price

FROM products p

INNER JOIN historyProductPrices hpp ON p.id = hpp.idProduct

WHERE hpp.effectiveDate = (

SELECT MAX(effectiveDate)

FROM historyProductPrices

WHERE idProduct = p.id

);

-- 9. Obtener todas las órdenes de compra junto con el nombre del proveedor

SELECT po.id, p.firstName, p.lastName1, po.orderDate

FROM purchaseOrders po

INNER JOIN people p ON po.idSupplier = p.idUser;

-- 10. Listar las categorías que tienen productos asociados

SELECT pc.categoryName, COUNT(p.id) AS productCount

FROM productCategories pc

INNER JOIN products p ON pc.id = p.idCategory

GROUP BY pc.categoryName;

## Consultas SQL con LEFT JOIN

-- \*\*\*\*\* Consultas con LEFT JOIN \*\*\*\*\*

-- 1. Listar todos los usuarios y sus roles (incluso si algún usuario no tiene rol asignado)

SELECT u.userName, r.roleName

FROM users u

LEFT JOIN usersRoles ur ON u.id = ur.idUser

LEFT JOIN roles r ON ur.idRole = r.id;

-- 2. Mostrar todas las categorías de productos, incluso aquellas que no tienen productos asociados

SELECT pc.categoryName, p.productName

FROM productCategories pc

LEFT JOIN products p ON pc.id = p.idCategory;

-- 3. Obtener todos los productos, incluso si no tienen precios registrados

SELECT p.productName, hpp.price

FROM products p

LEFT JOIN historyProductPrices hpp ON p.id = hpp.idProduct;

-- 4. Listar todas las órdenes de compra con el nombre del proveedor (incluso si el proveedor no está registrado como persona)

SELECT po.id, po.orderDate, p.firstName, p.lastName1

FROM purchaseOrders po

LEFT JOIN people p ON po.idSupplier = p.idUser;

-- 5. Mostrar todas las direcciones, incluso aquellas que no están asociadas a personas

SELECT a.street, a.city, p.firstName

FROM addresses a

LEFT JOIN people p ON a.id = p.idAddress;

-- 6. Obtener todos los productos con sus categorías, incluso si algún producto no tiene una categoría asociada

SELECT p.productName, pc.categoryName

FROM products p

LEFT JOIN productCategories pc ON p.idCategory = pc.id;

-- 7. Listar todas las personas y sus direcciones (aunque alguna persona no tenga dirección)

SELECT p.firstName, p.lastName1, a.street, a.city

FROM people p

LEFT JOIN addresses a ON p.idAddress = a.id;

-- 8. Mostrar todos los roles y los usuarios asignados a ellos (incluyendo roles sin usuarios)

SELECT r.roleName, u.userName

FROM roles r

LEFT JOIN usersRoles ur ON r.id = ur.idRole

LEFT JOIN users u ON ur.idUser = u.id;

-- 9. Obtener todos los productos junto con los detalles de órdenes de compra (incluyendo productos que no han sido comprados)

SELECT p.productName, od.quantity

FROM products p

LEFT JOIN orderDetails od ON p.id = od.idProduct;

-- 10. Mostrar todas las órdenes de compra, con los detalles de los productos (incluyendo órdenes sin productos)

SELECT po.id, od.idProduct, od.quantity

FROM purchaseOrders po

LEFT JOIN orderDetails od ON po.id = od.idPurchaseOrder;

## Consultas SQL con RIGHT JOIN

-- \*\*\*\*\* Consultas con RIGHT JOIN \*\*\*\*\*

-- 1. Listar todos los productos y sus categorías (incluyendo categorías sin productos)

SELECT p.productName, pc.categoryName

FROM products p

RIGHT JOIN productCategories pc ON p.idCategory = pc.id;

-- 2. Mostrar todos los precios de productos, incluso si algunos productos no están registrados

SELECT p.productName, hpp.price

FROM products p

RIGHT JOIN historyProductPrices hpp ON p.id = hpp.idProduct;

-- 3. Obtener todas las personas y sus direcciones (incluyendo direcciones no asociadas a personas)

SELECT p.firstName, p.lastName1, a.street

FROM people p

RIGHT JOIN addresses a ON p.idAddress = a.id;

-- 4. Mostrar todos los roles y los usuarios asignados (incluyendo roles que no tienen usuarios)

SELECT r.roleName, u.userName

FROM usersRoles ur

RIGHT JOIN roles r ON ur.idRole = r.id

RIGHT JOIN users u ON ur.idUser = u.id;

-- 5. Listar todas las órdenes de compra junto con sus detalles (incluso si no tienen productos asociados)

SELECT po.id, od.idProduct, od.quantity

FROM orderDetails od

RIGHT JOIN purchaseOrders po ON od.idPurchaseOrder = po.id;

-- 6. Obtener todos los proveedores y sus órdenes de compra (incluso si algún proveedor no tiene órdenes)

SELECT p.firstName, p.lastName1, po.id AS orderId

FROM purchaseOrders po

RIGHT JOIN people p ON po.idSupplier = p.idUser;

-- 7. Mostrar todas las órdenes de compra con sus estados (incluso si alguna orden no tiene un estado asociado)

SELECT po.id, os.statusName

FROM purchaseOrders po

RIGHT JOIN orderStatus os ON po.idOrderStatus = os.id;

-- 8. Listar todos los productos y sus órdenes de compra (incluso si algún producto no ha sido comprado)

SELECT p.productName, od.quantity

FROM orderDetails od

RIGHT JOIN products p ON od.idProduct = p.id;

-- 9. Mostrar todos los usuarios y sus roles (incluso si algún rol no tiene usuarios asociados)

SELECT u.userName, r.roleName

FROM usersRoles ur

RIGHT JOIN roles r ON ur.idRole = r.id

RIGHT JOIN users u ON ur.idUser = u.id;

-- 10. Obtener todos los productos y sus categorías (incluso si alguna categoría no tiene productos)

SELECT p.productName, pc.categoryName

FROM products p

RIGHT JOIN productCategories pc ON p.idCategory = pc.id;

## Subconsultas SQL con IN

-- Consultar los productos que están en órdenes realizadas por el usuario 'carlos123'.

SELECT productName

FROM products

WHERE id IN (

SELECT idProduct

FROM orderDetails

WHERE idPurchaseOrder IN (

SELECT id

FROM purchaseOrders

WHERE idSupplier IN (

SELECT id

FROM users

WHERE userName = 'carlos123'

)

)

);

-- Consultar los nombres de los usuarios que tienen el rol 'Admin'.

SELECT userName

FROM users

WHERE id IN (

SELECT idUser

FROM usersRoles

WHERE idRole IN (

SELECT id

FROM roles

WHERE roleName = 'Admin'

)

);

-- Consultar las órdenes que tienen el estado 'Pending'.

SELECT id, orderDate

FROM purchaseOrders

WHERE idOrderStatus IN (

SELECT id

FROM orderStatus

WHERE statusName = 'Pending'

);

-- Consultar las facturas generadas por el usuario 'maria123'.

SELECT id, invoiceDate

FROM invoices

WHERE idUser IN (

SELECT id

FROM users

WHERE userName = 'maria123'

);

-- Consultar los productos que pertenecen a la categoría 'Electronics'.

SELECT productName

FROM products

WHERE idCategory IN (

SELECT id

FROM productCategories

WHERE categoryName = 'Electronics'

);

## Subconsultas SQL con EXISTS

-- Buscar usuarios que han generado al menos una factura

SELECT \*

FROM users u

WHERE EXISTS (

SELECT 1

FROM invoices i

WHERE i.idUser = u.id

);

-- Encontrar productos que tienen un historial de precios registrado

SELECT \*

FROM products p

WHERE EXISTS (

SELECT 1

FROM historyProductPrices hpp

WHERE hpp.idProduct = p.id

);

-- Listar órdenes de compra que tienen al menos un detalle de orden

SELECT \*

FROM purchaseOrders po

WHERE EXISTS (

SELECT 1

FROM orderDetails od

WHERE od.idPurchaseOrder = po.id

);

-- Buscar categorías de productos que tienen productos asociados

SELECT \*

FROM productCategories pc

WHERE EXISTS (

SELECT 1

FROM products p

WHERE p.idCategory = pc.id

);

-- Encontrar direcciones que han sido asociadas a al menos una persona

SELECT \*

FROM addresses a

WHERE EXISTS (

SELECT 1

FROM people p

WHERE p.idAddress = a.id

);

## Subconsultas SQL con ANY y ALL

-- Buscar usuarios cuyo nombre de usuario es mayor que cualquier otro en la tabla

SELECT \*

FROM users u

WHERE u.userName > ANY (

SELECT userName

FROM users

);

-- Listar productos cuyo precio es menor que el precio más alto de cualquier producto

SELECT \*

FROM products p

WHERE p.id < ALL (

SELECT price

FROM historyProductPrices

);

-- Encontrar categorías de productos que tienen al menos un producto cuyo nombre es más corto que cualquier otro

SELECT \*

FROM productCategories pc

WHERE EXISTS (

SELECT 1

FROM products p

WHERE p.idCategory = pc.id AND LENGTH(p.productName) < ANY (

SELECT LENGTH(productName)

FROM products

)

);

-- Listar órdenes de compra con un estado de orden que es el mismo que todos los estados de orden registrados

SELECT \*

FROM purchaseOrders po

WHERE po.idOrderStatus = ALL (

SELECT id

FROM orderStatus

);

-- Encontrar personas que tienen un número de identificación mayor que todos los números de identificación en la base de datos

SELECT \*

FROM people p

WHERE p.identificationNumber > ALL (

SELECT identificationNumber

FROM people

);

# ANÁLISIS Y DISCUSIÓN

#### 1. **Estructura General**

La base de datos está compuesta por varias tablas que cubren diferentes aspectos del sistema de inventario, incluyendo la gestión de usuarios, productos, órdenes de compra, facturas y métodos de pago. Cada tabla está diseñada para almacenar información relevante y está conectada a través de claves foráneas, lo que permite una gestión eficiente de los datos.

#### 2. **Relaciones entre Entidades**

Las relaciones definidas en el modelo son principalmente de tipo uno a muchos (1

), lo cual es adecuado para este tipo de sistema. Por ejemplo:

* **Usuarios y Personas:** Un usuario puede tener múltiples registros de personas, lo que permite gestionar diferentes identidades bajo un mismo perfil de usuario.
* **Órdenes de Compra y Detalles de Órdenes:** Las órdenes pueden tener múltiples detalles, permitiendo que cada orden de compra registre varios productos y cantidades.

Este tipo de diseño facilita la consulta de datos, ya que se puede obtener información detallada de una orden de compra sin redundancia.

#### 3. **Normalización**

La estructura parece estar bien normalizada, minimizando la redundancia de datos. Las tablas están organizadas de tal manera que:

* **Tipos de Identificación:** Separar los tipos de identificación en su propia tabla evita la duplicación de datos y permite la fácil gestión y modificación de los tipos de identificación en caso de ser necesario.
* **Roles de Usuario:** La tabla de roles permite la asignación flexible de múltiples roles a diferentes usuarios, lo que es fundamental para gestionar permisos y accesos.

La normalización también ayuda a mantener la integridad referencial, dado que cada tabla depende de claves foráneas que refuerzan la relación entre entidades.

#### 4. **Eficiencia de Consultas**

El diseño de la base de datos favorece la eficiencia en las consultas. Al tener relaciones bien definidas y una estructura clara, las consultas que utilizan JOIN, EXISTS y subconsultas pueden ejecutarse de manera más rápida y con un menor uso de recursos.

Por ejemplo, las consultas que buscan productos asociados a órdenes de compra o facturas pueden realizarse rápidamente gracias a las relaciones establecidas entre las tablas, lo que mejora la rapidez de las operaciones del sistema.

#### 5. **Consideraciones de Seguridad**

La implementación de una tabla de usuarios con contraseñas encriptadas y la asignación de roles son prácticas recomendadas para mejorar la seguridad del sistema. La separación de roles permite una gestión más controlada del acceso a la base de datos, asegurando que solo los usuarios autorizados puedan realizar acciones específicas.

#### 6. **Flexibilidad y Escalabilidad**

El diseño permite la adición de nuevas características en el futuro. Por ejemplo, se pueden agregar nuevos métodos de pago, tipos de productos o estados de órdenes sin necesidad de rediseñar completamente la base de datos. Esto ofrece flexibilidad y facilita el mantenimiento a largo plazo.

### Discusión

#### Fortalezas

* **Organización Clara:** El modelo presenta una organización lógica de las entidades y sus relaciones, lo que facilita la comprensión del sistema.
* **Integridad de Datos:** La normalización y las restricciones de clave foránea garantizan la integridad de los datos, lo que es crucial en sistemas donde la precisión es fundamental.
* **Consultas Eficientes:** Las relaciones bien definidas permiten que las consultas sean más eficientes, lo que es esencial para aplicaciones que requieren un acceso rápido a los datos.

#### Debilidades y Oportunidades de Mejora

* **Manejo de Errores:** Se debe considerar la implementación de mecanismos de manejo de errores para situaciones como inserciones de datos incorrectos o relaciones que no cumplen con las restricciones de integridad.
* **Optimización de Consultas:** Aunque el diseño es eficiente, las consultas complejas que involucran múltiples tablas podrían beneficiarse de optimizaciones adicionales, como la creación de índices.
* **Seguridad de Datos:** La implementación de medidas de seguridad adicionales, como la auditoría de cambios en las tablas y el cifrado de datos sensibles, podría mejorar la seguridad general del sistema.

# CONCLUSIONES

1. **Diseño Estructurado y Coherente:**
   * La base de datos presenta una estructura bien organizada que facilita la gestión de la información relacionada con usuarios, productos, órdenes de compra, facturas y métodos de pago. Cada tabla cumple una función específica y está relacionada de manera lógica con otras, lo que mejora la usabilidad del sistema.
2. **Integridad de los Datos:**
   * Gracias a la implementación de claves foráneas y la normalización, se asegura la integridad de los datos, evitando la duplicación y garantizando que las relaciones entre entidades se mantengan. Esto es crucial para un sistema que requiere precisión en la información, como el manejo de inventarios y órdenes de compra.
3. **Flexibilidad y Escalabilidad:**
   * El diseño permite agregar nuevas funcionalidades y tablas sin necesidad de una reestructuración completa. Esta flexibilidad es vital para adaptarse a futuras necesidades del negocio, como la inclusión de nuevos productos, métodos de pago o tipos de usuarios.
4. **Eficiencia en Consultas:**
   * La relación clara entre las tablas y el uso de índices permiten que las consultas sean rápidas y eficientes. Esto es esencial para mantener un buen rendimiento en un entorno donde se espera un alto volumen de transacciones y consultas de datos.

# RECOMENDACIONES

* **Mantenimiento Regular:**
  + Es importante realizar un mantenimiento regular de la base de datos para asegurar su rendimiento y adecuación a las necesidades del negocio.
* **Actualización de Seguridad:**
  + La base de datos debe actualizarse constantemente para implementar nuevas medidas de seguridad y adaptarse a las amenazas emergentes.
* **Capacitación de Usuarios:**
  + Ofrecer capacitación a los usuarios del sistema para maximizar su eficacia en el uso de la base de datos y garantizar el cumplimiento de las políticas de seguridad.
* **Oportunidad de mejora**
* Aunque el diseño es sólido, hay oportunidades para optimizar aún más el rendimiento de las consultas y mejorar el manejo de errores. La implementación de auditorías de cambios y el cifrado de datos sensibles también son áreas que se pueden considerar para elevar la seguridad.

# REFERENCIAS

<https://lucid.app/documents#/home?folder_id=recent>

<https://www.youtube.com/watch?v=TKuxYHb-Hvc>

<https://miro.com/es/diagrama/como-hacer-diagrama-entidad-relacion/>

draw.io= <https://app.diagrams.net/?src=about>