Informe de Proyecto de Base de Datos

Presentado por:

Michael Alejandro Papamija Pantoja

Docente:

Brayan Arcos

Materia:

Desarrollo de Base de Datos

Instituto Tecnológico del Putumayo

Mocoa – Putumayo

2024

1. Resumen Ejecutivo

La base de datos ha sido diseñada para gestionar de manera eficiente la información relacionada con clientes, empleados, productos, proveedores, métodos de pago, y transacciones en un entorno empresarial. El objetivo principal es proporcionar una estructura robusta que permita almacenar, organizar y gestionar los datos clave de la organización, asegurando la integridad, consistencia y accesibilidad óptima de la información.

El diseño sigue estrictos principios de normalización, lo que garantiza la eliminación de redundancias y una adecuada distribución de las dependencias entre las tablas. Entre las entidades principales se incluyen Cliente, Empleado, TipoDocumento, Cargo, Turno, MetodoPago, y se han añadido otras tablas importantes como Producto, CategoríaProducto, ProveedorProducto, DetallePago y TelefonoPersona. Estas tablas relacionales permiten representar de manera eficiente las relaciones entre las entidades, facilitando una gestión clara y precisa de los datos. Además, el esquema incluye funcionalidades para controlar inventarios, registrar transacciones de pago y realizar un seguimiento detallado de los proveedores y productos.

2. Introducción

Este informe se realizó con el propósito de documentar y analizar el proceso de trabajo con bases de datos relacionales utilizando SQL, una herramienta clave en la gestión y manipulación de grandes volúmenes de información. La motivación principal detrás de este informe radica en la importancia de dominar el lenguaje SQL para garantizar un manejo eficiente y optimizado de los datos, lo cual es esencial en múltiples áreas tecnológicas.

El informe abarca varios aspectos fundamentales de SQL, tales como la creación de consultas para extraer información relevante, la optimización de dichas consultas para mejorar el rendimiento, y el diseño estructurado de bases de datos relacionales. Estos temas son analizados con el objetivo de presentar las mejores prácticas y enfoques para un manejo eficiente de los datos.

El objetivo principal es mostrar el proceso de creación, consulta, y optimización de bases de datos utilizando SQL, así como el análisis de su diseño y cómo éste influye en el rendimiento y eficiencia del sistema de gestión de bases de datos.

3. Metodología

3.1 Herramientas Utilizadas

Para el desarrollo de este informe se emplearon las siguientes herramientas:

- **MySQL** como sistema de gestión de bases de datos (DBMS).
- **MySQL Workbench** para la creación de la base de datos, definición de tablas y relaciones, y ejecución de consultas SQL.
- **DIA** para la creación del diagrama entidad-relación (ER), lo que facilitó la visualización del diseño de la base de datos.
- **Enlace de GitHub: https://github.com/dev-Alejo24/MySQL-Michael_Papamija-.git**

3.2 Procedimientos

Los procedimientos seguidos para llevar a cabo el análisis fueron los siguientes:

1. Creación del esquema de la base de datos en MySQL Workbench, incluyendo la definición de tablas, claves primarias, foráneas, y las relaciones entre las entidades.
2. Ejecución de consultas SQL en MySQL Workbench para extraer información relevante y validar el correcto funcionamiento de la base de datos.
3. Análisis de los resultados obtenidos a partir de las consultas y optimización de las mismas para mejorar el rendimiento del sistema.
4. Creación del diagrama entidad-relación en DIA para representar visualmente la estructura y relaciones de la base de datos implementada

4. Desarrollo del Informe

4.1 Descripción de la Base de Datos

La base de datos está diseñada para manejar información de clientes, empleados y datos relacionados, como tipos de documentos, cargos, turnos y métodos de pago.

4.1.1 Descripción de las tablas de la base de datos

La base de datos propuesta está diseñada para gestionar de manera eficiente la información relacionada con usuarios, empleados, clientes, métodos de pago, productos, proveedores, entre otros elementos clave del sistema. Su diseño se basa en principios de normalización, lo que garantiza una estructura óptima de las tablas, evitando redundancias, mejorando la

consistencia de los datos y asegurando la integridad referencial entre las distintas entidades. Esto facilita tanto la escalabilidad como la fiabilidad del sistema, permitiendo un manejo más preciso y eficiente de la información.

1. **store**: Representa la tienda o distribuidora. Incluye detalles como el nombre, la dirección, el NIT (Número de Identificación Tributaria), el teléfono y la fecha de apertura.

```sql
-- NEW TABLE
CREATE TABLE store(
    id INT PRIMARY KEY AUTO_INCREMENT,
    `name` VARCHAR(255),
    address VARCHAR(255),
    nit VARCHAR(20), -- NIT como VARCHAR,
    phone VARCHAR(255),
    openingDate date,
    createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    updatedAt DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

2. **documentType**: Define los tipos de documentos de identificación que puede usar una persona, como cédula, pasaporte, etc.

```sql
CREATE TABLE documentType(
    id INT PRIMARY KEY AUTO_INCREMENT,
    `name` VARCHAR(255) NOT NULL,
    createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    updatedAt DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

3. **users**: Almacena la información de los usuarios del sistema, como el correo electrónico, el nombre, y la contraseña encriptada.

```sql
-- SE ELIMINA jobPosition que es similar a roles
CREATE TABLE users (
    id INT PRIMARY KEY AUTO_INCREMENT,
    email VARCHAR(255) NOT NULL UNIQUE,
    `name` VARCHAR(255) NOT NULL,
    `password` VARCHAR(255) NOT NULL UNIQUE, -- hashed password
    createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    updatedAt DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

4. **roles**: Contiene los diferentes roles que pueden tener los usuarios (por ejemplo, administrador, cliente). Incluye una descripción opcional.

```sql
CREATE TABLE roles(
    id INT PRIMARY KEY AUTO_INCREMENT,
    `name` VARCHAR(255) NOT NULL UNIQUE,
    `description` TINYTEXT,
    createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    updatedAt DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

5. **usersRoles**: Relaciona usuarios con roles, permitiendo asignar múltiples roles a un usuario, y asegurando que cada combinación sea única.

```sql
CREATE TABLE usersRoles(
    id INT PRIMARY KEY AUTO_INCREMENT,
    idRole INT,
    FOREIGN KEY (idRole) REFERENCES roles(id),
    idUser INT,
    FOREIGN KEY (idUser) REFERENCES users(id),
    UNIQUE(idRole, idUser), -- asegurando combinacion unica
    createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    updatedAt DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

6. **shift**: Define los turnos laborales disponibles en una tienda, con una descripción y referencia a la tienda a la que pertenece el turno.

```sql
-- DIRECTA A USERS
-- **CORECCION DE CONEXION A STORE**
CREATE TABLE shift (
    id INT PRIMARY KEY AUTO_INCREMENT,
    `name` VARCHAR(255) NOT NULL UNIQUE,
    `description` TINYTEXT,
    idStore INT,
    FOREIGN KEY (idStore) REFERENCES store(id), -- Corrección de referencia a store
    createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    updatedAt DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

7. **paymentMethod**: Guarda los métodos de pago disponibles, como tarjetas de crédito, efectivo, transferencias bancarias, etc.

```sql
CREATE TABLE paymentMethod (
    id INT PRIMARY KEY AUTO_INCREMENT,
    `name` VARCHAR(255) NOT NULL,
    `description` TINYTEXT,
    createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    updatedAt DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

8. **people**: Almacena información personal de empleados y clientes, como nombres, apellidos, documento de identidad, teléfono, dirección, fecha de contratación, y salario.

```sql
CREATE TABLE people ( -- customer and employee
    id INT PRIMARY KEY AUTO_INCREMENT,
    idUser INT,
    FOREIGN KEY (idUser) REFERENCES users(id),
    firstName VARCHAR(255) NOT NULL,
    middleName VARCHAR(255),
    lastNameMaternal VARCHAR(255) NOT NULL,
    lastNamePaternal VARCHAR(255) NOT NULL,
    idDocumentType INT,
    FOREIGN KEY (idDocumentType) REFERENCES documentType(id),
    document VARCHAR(255) NOT NULL UNIQUE,
    phone VARCHAR(255),
    address VARCHAR(255),
    hiringDate DATETIME NOT NULL,
    salary DOUBLE NOT NULL,
    createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    updatedAt DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

9. **paymentDetail**: Registra los detalles de los pagos realizados por los clientes, incluyendo el monto, la fecha del pago, y el método de pago utilizado.

```sql
CREATE TABLE paymentDetail (
    id INT PRIMARY KEY AUTO_INCREMENT, -- add primary key and convert to strong table
    idCustomer INT,
    FOREIGN KEY (idCustomer) REFERENCES users(id), -- Corregida la referencia a users
    paymentDate DATETIME NOT NULL,
    paymentAmount DOUBLE NOT NULL,
    idPaymentMethod INT,
    FOREIGN KEY (idPaymentMethod) REFERENCES paymentMethod(id),
    createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    updatedAt DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

10. **productsCategories**: Clasifica los productos en diferentes categorías (por ejemplo, bebidas, alimentos, electrónicos).

```sql
CREATE TABLE productsCategories (
    id INT PRIMARY KEY AUTO_INCREMENT,
    `name` VARCHAR(255) NOT NULL,
    `description` TEXT,
    createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    updatedAt DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

11. **products**: Guarda la información de los productos que vende la tienda, como el nombre, el código, el precio, descuentos, el stock, y la categoría a la que pertenecen.

```sql
CREATE TABLE products (
    id INT PRIMARY KEY AUTO_INCREMENT,
    `name` VARCHAR(255) NOT NULL,
    `code` VARCHAR(255) UNIQUE NOT NULL, -- codigo del producto
    price DECIMAL(10, 2) NOT NULL,
    priceDiscount DECIMAL(10,2),
    stock INT NOT NULL DEFAULT 0,
    idCategory INT,
    FOREIGN KEY (idCategory) REFERENCES productsCategories(id),
    `description` TEXT,
    createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    updatedAt DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

12. **storeProducts**: Relaciona los productos con la tienda, indicando el stock disponible y la fecha de reabastecimiento.

```sql
-- NEW TABLE
CREATE TABLE storeProducts(
    id INT PRIMARY KEY AUTO_INCREMENT,
    idStore INT,
    FOREIGN KEY (idStore) REFERENCES store(id),
    idProduct INT,
    FOREIGN KEY (idProduct) REFERENCES products(id),
    stock BIGINT,
    restockDate DATE,
    createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    updatedAt DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

13. **supplierProduct**: Registra las compras de productos a proveedores, con información sobre cantidad, precio de compra y venta, y fecha de caducidad.

```sql
CREATE TABLE supplierProduct(
    id INT PRIMARY KEY AUTO_INCREMENT,
    idSupplier INT,
    FOREIGN KEY (idSupplier) REFERENCES users(id), -- Relacion directa con users
    idProduct INT,
    FOREIGN KEY (idProduct) REFERENCES products(id),
    quantity INT NOT NULL,
    purchasePrice DECIMAL(10, 2) NOT NULL,
    salePrice DECIMAL(10, 2) NOT NULL,
    expirationDate DATE,
    createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    updatedAt DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

14. **productDetails**: Almacena detalles específicos de los productos (por ejemplo, tamaño, color, características adicionales).

```
REATE TABLE productDetails (
   id INT PRIMARY KEY AUTO_INCREMENT,
   idProduct INT,
   FOREIGN KEY (idProduct) REFERENCES products(id),
   detailName VARCHAR(255) NOT NULL,
   detailValue VARCHAR(255),
   createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
   updatedAt DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
;
```

15. **peoplePhone**: Permite almacenar varios números de teléfono para una persona (cliente o empleado).

```
CREATE TABLE peoplePhone (
    id INT PRIMARY KEY AUTO_INCREMENT,
    idPeople INT,
    FOREIGN KEY (idPeople) REFERENCES people(id),
    phone VARCHAR(255) NOT NULL,
    createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    updatedAt DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

16. **invoices**: Registra las facturas emitidas a los clientes, incluyendo la fecha y el monto total de la factura.

```
-- TABLE INVOICES
CREATE TABLE invoices (
    id INT PRIMARY KEY AUTO_INCREMENT,
    invoiceDate DATETIME NOT NULL,
    totalAmount DECIMAL(10, 2) NOT NULL,
    idCustomer INT,
    FOREIGN KEY (idCustomer) REFERENCES users(id), -- conexion a *users*
    createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    updatedAt DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

17. **invoiceDetails**: Almacena los detalles de cada factura, como los productos vendidos, su cantidad, precio y el total calculado.

```
-- TABLE INVOICE DETAILS
CREATE TABLE invoiceDetails (
    id INT PRIMARY KEY AUTO_INCREMENT,
    idInvoice INT,
    FOREIGN KEY (idInvoice) REFERENCES invoices(id),
    idProduct INT,
    FOREIGN KEY (idProduct) REFERENCES products(id),
    quantity INT NOT NULL,
    price DECIMAL(10, 2) NOT NULL,
    total DECIMAL(10, 2) GENERATED ALWAYS AS (quantity * price) STORED, -- auto-calculated
    createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    updatedAt DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

18. **payment**: Registra los pagos asociados a facturas, incluyendo la fecha del pago, el método de pago y el monto.

```
-- TABLE PAYMENT
CREATE TABLE payment (
    id INT PRIMARY KEY AUTO_INCREMENT,
    idInvoice INT,
    FOREIGN KEY (idInvoice) REFERENCES invoices(id),
    paymentDate DATETIME NOT NULL,
    idPaymentMethod INT,
    FOREIGN KEY (idPaymentMethod) REFERENCES paymentMethod(id),
    paymentAmount DECIMAL(10, 2) NOT NULL,
    createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    updatedAt DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

19. **paymentGatewayRecords**: Guarda registros de las transacciones realizadas a través de una pasarela de pago, incluyendo referencias o códigos de transacción y datos adicionales relacionados con el pago.

```
- TABLE PAYMENT GATEWAY RECORDS
REATE TABLE paymentGatewayRecords (
   id INT PRIMARY KEY AUTO_INCREMENT,
   reference VARCHAR(255) NOT NULL, -- store reference or transaction code
   idPayment INT,
   FOREIGN KEY (idPayment) REFERENCES payment(id), -- Link to payment table
   dataRespons TEXT, -- Additional data or information related to the invoice ****
   createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
   updatedAt DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
;
```

4.1.2 Relaciones de las tablas de la base de datos

- **Tabla people (cliente y empleado)**: Está relacionada con las tablas documentType y users, utilizando las claves foráneas idDocumentType y idUser, respectivamente. Esta tabla también almacena la información personal y laboral (para los empleados) y permite manejar tanto a clientes como empleados.
- **Tabla usersRoles**: Relaciona usuarios con roles, identificando los diferentes permisos y funciones asignados a cada usuario mediante las claves foráneas idUser y idRole.
- **Tabla paymentDetail**: Está relacionada con las tablas users (para identificar al cliente) y paymentMethod, a través de las claves foráneas idCustomer e idPaymentMethod.
- **Tabla peoplePhone**: Está relacionada con la tabla people, permitiendo almacenar múltiples teléfonos por persona, a través de la clave foránea idPeople.
- **Tabla supplierProduct**: Está relacionada con las tablas users (para identificar al proveedor) y products, a través de las claves foráneas idSupplier e idProduct.
- **Tabla invoices**: Se relaciona con la tabla users (para identificar al cliente) a través de la clave foránea idCustomer, y almacena la información básica de cada factura emitida a un cliente.
- **Tabla invoiceDetails**: Se relaciona con las tablas invoices y products, permitiendo almacenar los productos y sus detalles específicos en cada factura mediante las claves foráneas idInvoice e idProduct.
- **Tabla payment**: Relaciona la tabla invoices con paymentMethod, guardando el registro de pagos realizados por cada factura mediante las claves foráneas idInvoice e idPaymentMethod.

4.1.3 Claves principales de la base de datos

- **documentType**: Cada registro tiene una clave principal auto-incremental id.
- **roles**: Cada registro tiene una clave principal auto-incremental id.
- **shift**: Cada registro tiene una clave principal auto-incremental id.
- **paymentMethod**: Cada registro tiene una clave principal auto-incremental id.
- **people**: Cada registro tiene una clave principal auto-incremental id, que identifica tanto a clientes como empleados.
- **users**: Cada registro tiene una clave principal auto-incremental id.
- **paymentDetail**: Cada registro tiene una clave principal auto-incremental id.
- **peoplePhone**: Cada registro tiene una clave principal auto-incremental id.
- **productsCategories**: Cada registro tiene una clave principal auto-incremental id.
- **products**: Cada registro tiene una clave principal auto-incremental id.
- **storeProducts**: Cada registro tiene una clave principal auto-incremental id.
- **supplierProduct**: Cada registro tiene una clave principal auto-incremental id.
- **productDetails**: Cada registro tiene una clave principal auto-incremental id.
- **invoices**: Cada registro tiene una clave principal auto-incremental id.
- **invoiceDetails**: Cada registro tiene una clave principal auto-incremental id.
- **payment**: Cada registro tiene una clave principal auto-incremental id.
- **paymentGatewayRecords**: Cada registro tiene una clave principal auto-incremental id.

4.1.4 Características clave de la base de datos

- La base de datos utiliza claves primarias **auto-incrementales** para cada tabla.
- Las columnas **createdAt** y **updatedAt** se utilizan para rastrear las fechas de creación y actualización de cada registro.
- La cláusula **ON UPDATE CURRENT_TIMESTAMP** se utiliza para actualizar automáticamente la fecha de actualización cuando se modifica un registro.

4.3 Diseño de Base de Datos

El diseño de la base de datos se basa en un modelo entidad-relación con una correcta normalización para evitar redundancia y asegurar la integridad de los datos.

4.3.1 Modelo Entidad Relación

1. **Store (Strong Entity)**
    - **ID_Store** (Primary Key)
    - Name (Store name)
    - Address (Store address)
    - NIT (Tax identification number)
    - Phone (Store phone number)
    - Opening Date (Date when the store opened)
    - **CreatedAt** (Record creation timestamp)
    - **UpdatedAt** (Record update timestamp)

2. **DocumentType (Strong Entity)**
    - **ID_Document_Type** (Primary Key)
    - Name_dt (Name of the document type)
    - **CreatedAt** (Record creation timestamp)
    - **UpdatedAt** (Record update timestamp)

3. **Users (Strong Entity)**
    - **ID_User** (Primary Key)
    - Email (User's email address, unique)
    - Name (User's name)
    - Password (Hashed password, unique)
    - **CreatedAt** (Record creation timestamp)
    - **UpdatedAt** (Record update timestamp)

4. **Roles (Strong Entity)**
    - **ID_Role** (Primary Key)
    - Name_role (Role name, unique)
    - Description (Role description)
    - **CreatedAt** (Record creation timestamp)
    - **UpdatedAt** (Record update timestamp)

5. **UsersRoles (Weak Entity, dependent on Users and Roles)**

Tambien es entidad fuerte ya que tiene primary key

- **ID_UsersRoles** (Primary Key)
- **ID_Role** (Foreign Key, references roles)
- **ID_User** (Foreign Key, references users)
- **CreatedAt** (Record creation timestamp)
- **UpdatedAt** (Record update timestamp)

6. **Shift (Strong Entity)**
    - **ID_Shift** (Primary Key)
    - Name_sh (Name of the shift, unique)
    - Description (Shift description)
    - **ID_Store** (Foreign Key, references store)
    - **CreatedAt** (Record creation timestamp)
    - **UpdatedAt** (Record update timestamp)

7. **PaymentMethod (Strong Entity)**
    - **ID_Payment_Method** (Primary Key)
    - Name_payment_method (Name of the payment method)
    - Description (Description of the payment method)
    - **CreatedAt** (Record creation timestamp)
    - **UpdatedAt** (Record update timestamp)

8. **People (Strong Entity)**
    - **ID_People** (Primary Key)
    - **ID_User** (Foreign Key, references users)
    - First Name
    - Middle Name
    - Maternal Last Name
    - Paternal Last Name

- **ID_Document_Type** (Foreign Key, references documentType)
- Document (Unique identifier)
- Phone
- Address
- Hiring Date
- Salary
- **CreatedAt** (Record creation timestamp)
- **UpdatedAt** (Record update timestamp)

9. **PaymentDetail (Weak Entity, dependent on Users)**

Tambien es entidad fuerte ya que tiene primary key

- **ID** (Primary Key)
- **ID_Customer** (Foreign Key, references users)
- Payment Date
- Payment Amount
- **ID_Payment_Method** (Foreign Key, references paymentMethod)
- CreatedAt
- UpdatedAt

10. **ProductsCategories (Strong Entity)**

- **ID_Category** (Primary Key)
- Name_category
- Description
- CreatedAt
- UpdatedAt

11. **Products (Strong Entity)**

- **ID_Product** (Primary Key)
- Name_product
- Code_product
  – Price
  – PriceDiscount
  – Stock

– Description
– ID_Category
– CreatedAt
– UpdatedAt

12. **StoreProducts (Weak Entity, dependent on Store and Products)**

Tambien es entidad fuerte ya que tiene primary key

– ID_StoreProduct
– ID_Store
– ID_Product
– Stock
– RestockDate
– CreatedAt
– UpdatedAt

13. **SupplierProduct(Strong Entity)**
    – ID_SupplierProduct
    – ID_Supplier
    – ID_Product
    – Quantity
    – PurchasePrice
    – SalePrice
    – ExpirationDate
    – CreatedAt
    – UpdatedAt

14. **ProductDetails(Strong Entity)**
    – ID_ProductDetail
    – ID_Product
    – DetailName
    – DetailValue
    – CreatedAt
    – UpdatedAt

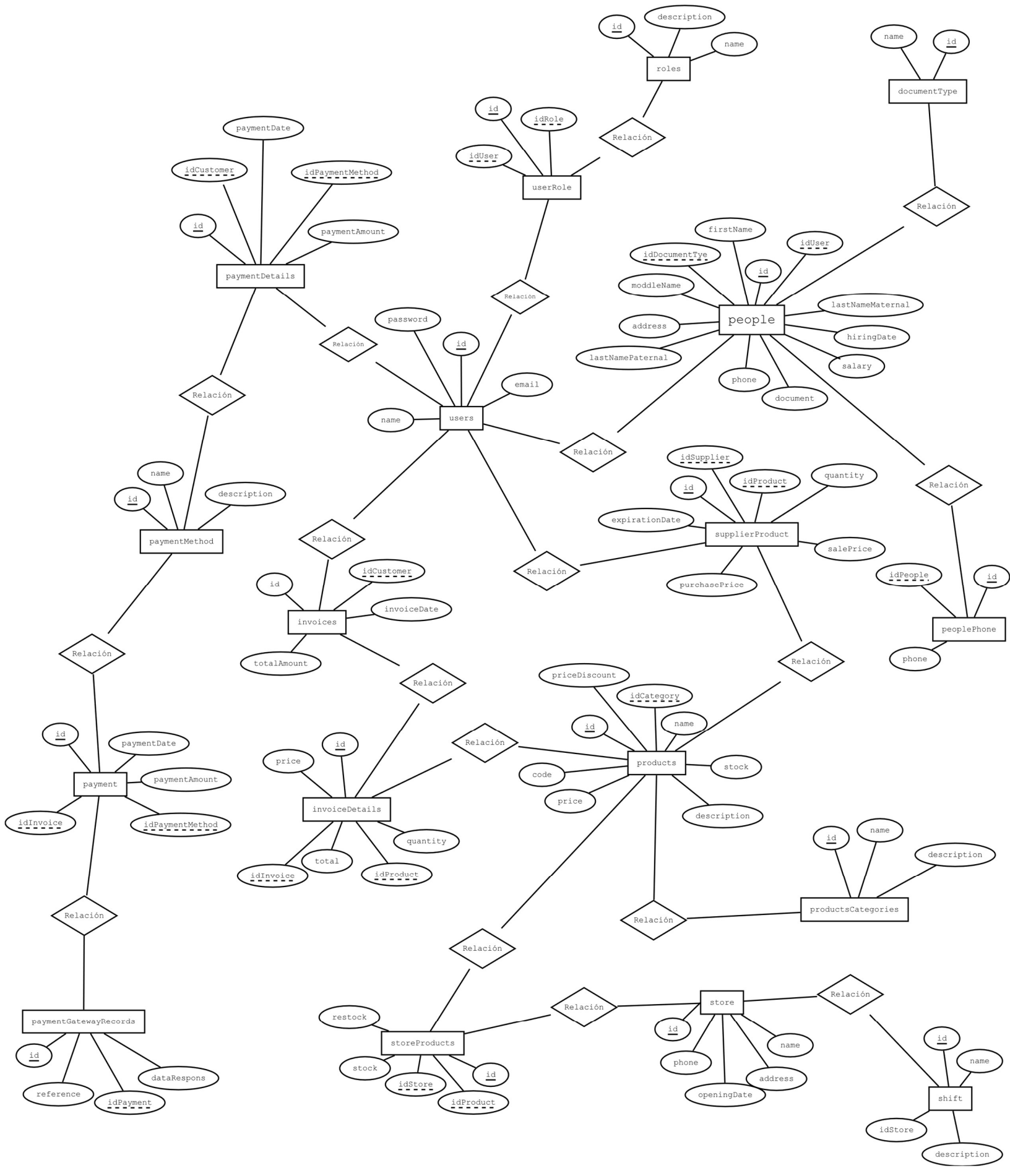
15. **PeoplePhone(Multivalued Attribute, represented as a separate table)**
    – ID_Phone
    – ID_People
    – PhoneNumber

16. **Invoices(Strong Entity)**
    – ID_Invoice
    – InvoiceDate
    – TotalAmount
    – ID_Customer
    – CreatedAt
    – UpdatedAt

17. **InvoiceDetails(Strong Entity)**
    – ID_InvoiceDetail
    – ID_Invoice
    – ID_Product
    – Quantity
    – Price
    – Total

18. **Payment(Strong Entity)**
    – ID_Payment
    – ID_Invoice
    – PaymentDate
    – ID_Payment_Method
    – PaymentAmount

19. **PaymentGatewayRecords(Strong Entity)**
    – ID_PaymentGatewayRecord
    – Reference
    — ID_Payment
    — DataResponse


id
description
name
roles
Relación
name
id
documentType
Relación
id
idRole
idUser
userRole
Relación
paymentDate
idCustomer
idPaymentMethod
id
paymentAmount
paymentDetails
Relación
firstName
idUser
idDocumentTye
id
moddleName
lastNameMaternal
address
people
hiringDate
lastNamePaternal
salary
phone
document
password
id
email
name
users
Relación
Relación
name
id
description
paymentMethod
Relación
idSupplier
idProduct
quantity
id
expirationDate
supplierProduct
salePrice
purchasePrice
Relación
Relación
idPeople
id
peoplePhone
phone
Relación
id
idCustomer
invoices
invoiceDate
totalAmount
Relación
Relación
priceDiscount
idCategory
id
name
products
stock
code
price
description
Relación
id
paymentDate
payment
paymentAmount
idInvoice
idPaymentMethod
Relación
price
id
invoiceDetails
quantity
total
idInvoice
idProduct
id
name
description
productsCategories
Relación
Relación
Relación
paymentGatewayRecords
id
dataRespons
reference
idPayment
restock
storeProducts
stock
idStore
id
idProduct
Relación
store
Relación
id
name
phone
address
openingDate
id
name
shift
idStore
description

4.3.2 Relaciones Normalizadas

1. **Realiza (Entre Users y PaymentDetail)**
    - **Atributos de la relación:**
        - **fecha_pago** (paymentDate)
        - **monto_pago** (paymentAmount)
2. **Está en (Entre People y Shift)**
    - Relación entre una persona (empleado o cliente) y el turno en el que trabaja. Ya está implementada en la tabla people como idShift.
3. **Tiene (Entre People y DocumentType)**
    - Relación entre una persona y su tipo de documento. Ya está implementada en la tabla people como idDocumentType.
4. **Utiliza (Entre PaymentDetail y PaymentMethod)**
    - Relación entre un detalle de pago y el método de pago utilizado. Este vínculo está presente a través de la tabla paymentDetail con idPaymentMethod.
5. **Tiene (Entre Users y UsersRoles)**
    - Relación entre un usuario y sus roles asignados. Esta relación se implementa en la tabla usersRoles, vinculando idUser con idRole.
6. **Asigna (Entre Roles y UsersRoles)**
    - Relación entre un rol y su asignación a los usuarios. Cada combinación de rol y usuario se gestiona en la tabla usersRoles.
7. **Contiene (Entre Store y Shift)**
    - Relación entre una tienda y los turnos disponibles en ella. Esta relación está implementada en la tabla shift como idStore.
8. **Pertenece a (Entre Products y ProductsCategories)**
    - Relación entre un producto y su categoría correspondiente. Esta relación está implementada en la tabla products como idCategory.
9. **Incluye (Entre Store y StoreProducts)**
    - Relación entre una tienda y los productos disponibles en ella. Esta relación se gestiona a través de la tabla storeProducts, que vincula idStore con idProduct.

10. **Provee (Entre Users y SupplierProduct)**
    - Relación entre un usuario que actúa como proveedor y los productos que suministra. Este vínculo está presente en la tabla supplierProduct, vinculando idSupplier con idProduct.
11. **Detalla (Entre Products y ProductDetails)**
    - Relación entre un producto y sus detalles específicos. Esta relación se gestiona en la tabla productDetails, donde cada detalle está vinculado a un único producto.
12. **Contiene (Entre Invoices y InvoiceDetails)**
    - Relación entre una factura y los detalles que la componen. Esta relación está implementada en la tabla invoiceDetails, vinculando cada detalle con una factura específica.
13. **Registra (Entre Payment y PaymentGatewayRecords)**
    - Relación entre un pago realizado y los registros generados en la pasarela de pago correspondiente. Este vínculo se gestiona a través de la tabla paymentGatewayRecords, donde cada registro está vinculado a un único pago.
14. **Asocia (Entre People y PeoplePhone)**
    - Relación entre una persona y sus números de teléfono asociados. Esta relación se gestiona a través de la tabla peoplePhone, donde cada número está vinculado a una única persona.

4.3.2 Cardinalidad

1. **Store (1) ---- (N) tiene (N) ---- (1) Shift**
    - Una tienda puede tener múltiples turnos, pero cada turno está asociado a una única tienda.
2. **DocumentType (1) ---- (N) tiene (N) ---- (1) People**
    - Un tipo de documento puede estar asociado a múltiples personas, pero cada persona tiene un único tipo de documento.
3. **Users (1) ---- (N) tiene (N) ---- (1) People**
    - Un usuario puede estar asociado a una o más personas, pero cada persona está asociada a un único usuario.
4. **Roles (1) ---- (N) tiene (N) ---- (1) UsersRoles**

- Un rol puede ser asignado a múltiples usuarios, pero cada asignación en UsersRoles está asociada a un único rol.

5. **Users (1) ---- (N) tiene (N) ---- (1) UsersRoles**
    - Un usuario puede tener múltiples roles a través de la tabla UsersRoles, pero cada asignación en UsersRoles está asociada a un único usuario.
6. **PaymentMethod (1) ---- (N) utiliza (N) ---- (1) PaymentDetail**
    - Un método de pago puede ser utilizado en múltiples detalles de pago, pero cada detalle de pago está asociado a un único método de pago.
7. **Users (1) ---- (N) realiza (N) ---- (1) PaymentDetail**
    - Un usuario puede realizar múltiples pagos, pero cada detalle de pago está asociado a un único usuario.
8. **ProductsCategories (1) ---- (N) tiene (N) ---- (1) Products**
    - Una categoría de productos puede incluir múltiples productos, pero cada producto pertenece a una única categoría.
9. **Products (1) ---- (N) tiene (N) ---- (1) StoreProducts**
    - Un producto puede estar disponible en múltiples tiendas, y cada entrada en StoreProducts está asociada a un único producto y una única tienda.
10. **Store (1) ---- (N) tiene (N) ---- (1) StoreProducts**
    - Una tienda puede tener múltiples productos asociados a ella, pero cada entrada en StoreProducts está asociada a una única tienda y un único producto.
11. **Users (1) ---- (N) tiene (N) ---- (1) SupplierProduct**
    - Un usuario puede ser proveedor de múltiples productos, pero cada entrada en SupplierProduct está asociada a un único usuario y un único producto.
12. **Products (1) ---- (N) tiene (N) ---- (1) ProductDetails**
    - Un producto puede tener múltiples detalles asociados, pero cada detalle está relacionado con un único producto.
13. **People(1) ----(N)tiene(N)--(1)PeoplePhone**
    - Una persona puede tener múltiples números de teléfono, pero cada número pertenece a una única persona.
14. **Invoices(1)----(N)tiene(N)--(1)Payment**
    - Una factura puede tener múltiples pagos asociados, pero cada pago corresponde a una única factura.
15. **Invoices(1)----(N)tiene(N)--(1)InvoiceDetails**

- Una factura puede tener múltiples detalles de factura asociados, pero cada detalle pertenece a una única factura.

16. **Payment(1)----(N)tiene(N)--(1)PaymentGatewayRecords**

- Un pago puede generar múltiples registros en la pasarela de pago, pero cada registro corresponde a un único pago.

4.3.3 Normalización

1. Primera Forma Normal (1NF):

- Se asegura de que todos los atributos sean atómicos, es decir, que cada campo contiene valores indivisibles. En este caso, las tablas de la base de datos ya cumplen con esta forma normal, ya que cada columna tiene valores únicos y no repetidos.

2. Segunda Forma Normal (2NF):

- Cada tabla tiene una clave primaria completa, y no hay dependencias parciales. Esto significa que los atributos no clave dependen completamente de la clave primaria. Las tablas actuales ya cumplen con esta forma normal, ya que los atributos como el nombre, apellido, etc., dependen completamente de las claves primarias en las tablas.

3. Tercera Forma Normal (3NF):

- No hay dependencias transitivas, lo que significa que los atributos no clave no dependen de otros atributos no clave. Todas las tablas están organizadas de manera que cada atributo depende directamente de la clave primaria. Esto evita redundancias y asegura una estructura eficiente de las tablas.

5. Análisis y Discusión

Los resultados obtenidos al implementar la base de datos demuestran que el diseño relacional cumple eficazmente con los objetivos principales planteados: estructurar y organizar la información de clientes, empleados, transacciones y demás elementos clave de la operación de la empresa. Las consultas realizadas a la base de datos permiten obtener información detallada y específica sobre pagos, clientes, empleados y otros elementos, garantizando la integridad de los datos.

- **Integridad de los datos**: Las relaciones definidas entre las tablas, utilizando claves foráneas, aseguran que la información esté bien conectada y que no haya inconsistencias en los datos. Los pagos, por ejemplo, están vinculados correctamente con los clientes y los métodos de pago, lo que permite un seguimiento preciso de las transacciones.

- **Consultas eficientes**: Las consultas ejecutadas sobre la base de datos, como la búsqueda de empleados por turno o la búsqueda de clientes por tipo de documento, fueron eficientes y produjeron resultados precisos. La estructura normalizada de las tablas contribuyó a optimizar los tiempos de respuesta de las consultas, especialmente en consultas complejas que involucraban múltiples tablas.

- **Atributos multivalorados y derivados**: La separación de los números de teléfono de los clientes en una tabla independiente, **people_phone**, demostró ser efectiva al evitar redundancias y facilitar la administración de varios números de teléfono por cliente. El cálculo de la antigüedad de los empleados, basado en la fecha de contratación, proporcionó una información útil para el análisis del rendimiento y la experiencia del personal.

En relación con los objetivos, la base de datos ha logrado cumplir con las expectativas planteadas al permitir:

- Un manejo eficiente de la información, evitando la redundancia y optimizando el acceso a los datos.
- Facilitar la administración de las relaciones entre entidades como empleados, clientes, y pagos.
- Asegurar que la información pueda escalar sin necesidad de reestructuración importante a medida que crezca la cantidad de datos.

6. Conclusiones

La base de datos es un sistema sólido que ha logrado alcanzar los objetivos de integridad, consistencia, y eficiencia. El diseño relacional de la base de datos, con las relaciones y dependencias bien definidas, garantiza un almacenamiento de datos libre de redundancias y una consulta eficiente de la información. La normalización hasta la **Tercera Forma Normal (3NF)** asegura que la base de datos sea fácil de mantener y escalar, mientras que las relaciones entre las entidades clave (como clientes, empleados, métodos de pago, y cargos) permiten un análisis más profundo y confiable.

La implementación de atributos derivados y multivalorados mediante tablas separadas ha mejorado la flexibilidad y el control sobre los datos, asegurando que la base de datos pueda adaptarse fácilmente a los cambios sin comprometer su rendimiento o integridad.

Principales conclusiones:

- La estructura de la base de datos asegura la integridad de los datos a través de la correcta implementación de claves primarias y foráneas.
- La normalización de las tablas hasta la 3NF ha permitido eliminar dependencias no deseadas y redundancias, mejorando la eficiencia de las consultas.

- La separación de atributos multivalorados y el uso de atributos derivados ha proporcionado flexibilidad y mejor control de la información.

.

7. Recomendaciones

Se recomienda realizar una revisión periódica del diseño de la base de datos para asegurar que las consultas SQL se ejecuten de manera óptima. Además, es importante considerar la implementación de índices en las tablas con mayor volumen de datos.

8. Referencias

1. Elmasri, R., & Navathe, S. (2010). Fundamentos de Sistemas de Bases de Datos.
2. Silberschatz, A., Korth, H. F., & Sudarshan, S. (2019). Database System Concepts.

3. MySQL Documentation. (2023). MySQL 8.0 Reference Manual.

4. DIA *Herramienta de Modelado ERD*.