Informe de Proyecto de Base de Datos

Presentado por:

Michael Alejandro Papamija Pantoja

Docente:

Brayan Arcos

Materia:

Programación Backend

Instituto Tecnológico del Putumayo

Mocoa – Putumayo

2024

1. Resumen Ejecutivo

La base de datos fue diseñada para gestionar eficientemente la información relacionada con clientes, empleados, métodos de pago, y transacciones en un entorno empresarial. El objetivo principal es ofrecer una estructura sólida que permita almacenar, organizar y gestionar los datos esenciales de la organización, garantizando la integridad, consistencia y optimización del acceso a la información.

El diseño de esta base de datos se basa en principios de normalización, asegurando la eliminación de redundancias y la correcta distribución de las dependencias entre las tablas. Para esto, se han implementado diversas entidades como **Cliente**, **Empleado**, **TipoDocumento**, **Cargo**, **Turno**, **MetodoPago**, y las tablas relacionales como **DetallePago** y **TelefonoCliente**, que permiten representar de manera eficiente las relaciones entre las diferentes entidades.

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-b_pbd2-MIchael_Papamija**

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

1. **tipo_documento**
    - Almacena información sobre diferentes tipos de documentos (e.g., ID, tipo de documento, fechas de creación y actualización)

```sql
DROP DATABASE IF EXISTS borrador_pbd2;
CREATE DATABASE borrador_pbd2;
USE borrador_pbd2;

CREATE TABLE tipo_documento (
    id_tipo_documento INT PRIMARY KEY AUTO_INCREMENT,
    tipo_documento VARCHAR(45) NOT NULL,
    createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    updatedAt DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

2. **cargo**

    - Almacena información sobre cargos o roles (e.g., ID, título del cargo, descripción, fechas de creación y actualización)

```sql
CREATE TABLE cargo (
    id_cargo INT PRIMARY KEY AUTO_INCREMENT,
    cargo VARCHAR(45) NOT NULL,
    descripcion TINYTEXT,
    createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    updatedAt DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

3. **turno**

    - Almacena información sobre turnos o horarios de trabajo (e.g., ID, nombre del turno, descripción, fechas de creación y actualización)

```sql
CREATE TABLE turno (
    id_turno INT PRIMARY KEY AUTO_INCREMENT,
    turno VARCHAR(15) NOT NULL,
    descripcion TINYTEXT,
    createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    updatedAt DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMEST
);
```

4. **metodo_pago**

    - Almacena información sobre métodos de pago (e.g., ID, nombre del método de pago, descripción, fechas de creación y actualización)

```sql
CREATE TABLE metodo_pago (
    id_metodo_pago INT PRIMARY KEY AUTO_INCREMENT,
    metodo_pago VARCHAR(45) NOT NULL,
    descripcion TINYTEXT,
    createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    updatedAt DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

5. **cliente**

    - Almacena información sobre clientes (e.g., ID, nombre, apellido, tipo de documento, número de documento, teléfono, email, dirección, fechas de creación y actualización)
    - Tiene una clave foránea que referencia la tabla **tipo_documento**

```sql
CREATE TABLE cliente (
    id_cliente INT PRIMARY KEY AUTO_INCREMENT,
    nombre VARCHAR(30) NOT NULL,
    apellido VARCHAR(30) NOT NULL,
    id_tipo_documento INT,
    FOREIGN KEY (id_tipo_documento) REFERENCES tipo_documento(id_tipo_documento),
    documento VARCHAR(12) NOT NULL UNIQUE,
    telefono VARCHAR(12),
    email VARCHAR(35),
    direccion VARCHAR(45),
    createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    updatedAt DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

6. **empleado**

    - Almacena información sobre empleados (e.g., ID, nombre, apellido, tipo de documento, número de documento, teléfono, email, dirección, cargo, turno, fecha de contratación, salario, fechas de creación y actualización)
    - Tiene claves foráneas que referencia las tablas **tipo_documento**, **cargo** y **turno**

```sql
CREATE TABLE empleado (
    id_empleado INT PRIMARY KEY AUTO_INCREMENT,
    nombre VARCHAR(30) NOT NULL,
    apellido VARCHAR(30) NOT NULL,
    id_tipo_documento INT,
    FOREIGN KEY (id_tipo_documento) REFERENCES tipo_documento(id_tipo_documento),
    documento VARCHAR(12) NOT NULL UNIQUE,
    telefono VARCHAR(12),
    email VARCHAR(35),
    direccion VARCHAR(45),
    id_cargo INT,
    FOREIGN KEY (id_cargo) REFERENCES cargo(id_cargo),
    id_turno INT,
    FOREIGN KEY (id_turno) REFERENCES turno(id_turno),
    fecha_contratacion DATETIME NOT NULL,
    salario DOUBLE NOT NULL,
    createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    updatedAt DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

7. **detalle_pago**:

- Almacena información sobre los pagos realizados por los clientes (e.g., ID del pago, fecha del pago, monto, método de pago, y el cliente que realizó el pago).
- Tiene claves foráneas que referencian las tablas cliente y metodo_pago, lo que permite vincular cada pago con un cliente y un método de pago específico.

```sql
-- Tabla débil: detalle_pago
CREATE TABLE detalle_pago (
    id_pago INT PRIMARY KEY AUTO_INCREMENT,
    id_cliente INT,
    FOREIGN KEY (id_cliente) REFERENCES cliente(id_cliente),
    fecha_pago DATETIME NOT NULL,
    monto_pago DOUBLE NOT NULL,
    id_metodo_pago INT,
    FOREIGN KEY (id_metodo_pago) REFERENCES metodo_pago(id_metodo_pago)
);
```

8. **telefono_cliente**:

- Almacena números de teléfono adicionales de los clientes (e.g., ID del teléfono, número de teléfono, cliente asociado).
- Tiene una clave foránea que referencia la tabla cliente, permitiendo almacenar múltiples números de teléfono para un mismo cliente.

```sql
-- Atributo multivalorado: teléfono de cliente
CREATE TABLE telefono_cliente (
    id_telefono INT PRIMARY KEY AUTO_INCREMENT,
    id_cliente INT,
    FOREIGN KEY (id_cliente) REFERENCES cliente(id_cliente),
    telefono VARCHAR(12) NOT NULL
);
```

4.1.2 Relaciones de las tablas de la base de datos

- La tabla cliente está relacionada con las tablas **tipo_documento** y **metodo_pago**.
- La tabla **empleado** está relacionada con las tablas **tipo_documento**, **cargo** y **turno**.
- La tabla **detalle_pago** tiene relaciones con cliente y **metodo_pago**.
- La tabla **telefono_cliente** está relacionada con **cliente**, permitiendo almacenar múltiples teléfonos por cliente.

4.1.3 Claves principales de la base de datos

- **tipo_documento:** Cada registro en esta tabla tiene una clave principal **auto-incremental `id_tipo_documento`.**
- **cargo**: Cada registro en esta tabla tiene una clave principal **auto-incremental `id_cargo`**.
- **turno**: Cada registro en esta tabla tiene una clave principal **auto-incremental `id_turno`.**
- **metodo_pago**: Cada registro en esta tabla tiene una clave principal **auto-incremental `id_metodo_pago`**.
- **cliente**: Cada registro en esta tabla tiene una clave principal **auto-incremental `id_cliente`**.
- **empleado**: Cada registro en esta tabla tiene una clave principal **auto-incremental `id_empleado`.**
- **detalle_pago:** Cada registro en esta tabla tiene una clave principal **auto-incremental `id_pago`.**
- **telefono_cliente:** Cada registro en esta tabla tiene una clave principal **auto-incremental `id_telefono`.**

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.

4.2 Consultas SQL

Se realizaron diversas consultas SQL para obtener datos relevantes. A continuación, se describen algunas de las consultas más importantes ejecutadas.

4.2.1 consulta a tabla ´empleado´

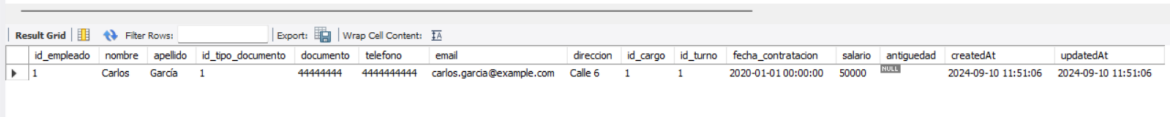
1. Consulta

```
USE borrador_pbd;

SELECT * FROM empleado
WHERE id_turno = 1 AND salario > 6000;
```

2. Resultados de consulta

La consulta selecciona todos los registros de la tabla empleado donde el id_turno es igual a 1 y el salario es mayor a 6000. Los resultados mostrarán los detalles de los empleados que cumplen con estos criterios, incluyendo columnas como id_empleado, nombre, apellido, id_tipo_documento, documento, telefono, email, direccion, id_cargo, id_turno, fecha_contratacion, antiguedad, y salario.



id_empleado	nombre	apellido	id_tipo_documento	documento	telefono	email	direccion	id_cargo	id_turno	fecha_contratacion	salario	antiguedad	createdAt	updatedAt
1	Carlos	García	1	44444444	4444444444	carlos.garcia@example.com	Calle 6	1	1	2020-01-01 00:00:00	50000	NULL	2024-09-10 11:51:06	2024-09-10 11:51:06

3. Explicacion de consulta

Explicación:

- **SELECT * FROM empleado**: Esta parte de la consulta indica que se deben seleccionar todos los campos (columnas) de la tabla empleado.
- **WHERE id_turno = 1 AND salario > 6000**: Esta es la cláusula de filtro que restringe los resultados:
    - **id_turno = 1**: Filtra los empleados que están asignados al turno con id_turno igual a 1 (por ejemplo, "Mañana").
    - **salario > 6000**: Filtra los empleados cuyo salario es mayor a 6000.

Lógica:

- **WHERE**: Se utiliza para aplicar condiciones a los registros que se seleccionan. En este caso, ambas condiciones deben ser verdaderas para que un registro sea incluido en los resultados:
    - La primera condición asegura que solo se seleccionen empleados que trabajen en el turno con id_turno igual a 1.
    - La segunda condición asegura que solo se incluyan empleados cuyo salario sea superior a 6000.

No se utilizan joins ni subconsultas en esta consulta específica. Solo se aplican filtros directos sobre la tabla empleado para obtener los registros que cumplen con ambas condiciones.

4.2.2 Consulta a tabla ´cliente´ LIKE, AND

1. Consulta

```
SELECT * FROM cliente
WHERE documento LIKE '2%' AND id_metodo_pago != 1;
```

2. Resultados de Consultas

La consulta selecciona todos los registros de la tabla cliente donde el campo documento comienza con el dígito '2' y el id_metodo_pago es diferente de 1. Los resultados mostrarán los detalles de los clientes que cumplen con estos criterios, incluyendo columnas como id_cliente, nombre, apellido, id_tipo_documento, documento, telefono, email, direccion, id_metodo_pago, antiguedad, createdAt, y updatedAt.

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id_cliente	nombre	apellido	id_tipo_documento	documento	telefono	email	id_metodo_pago	direccion	createdAt	updatedAt
4	Ana	Sánchez	4	22222222	2222222222	ana.sanchez@example.com	2	Calle 4	2024-09-10 11:50:34	2024-09-10 11:50:34

3. Explicación de Consultas

Explicación:

- **SELECT * FROM cliente**: Esta parte de la consulta indica que se deben seleccionar todos los campos (columnas) de la tabla cliente.
- **WHERE documento LIKE '2%' AND id_metodo_pago != 1**: Esta es la cláusula de filtro que restringe los resultados:
  - **documento LIKE '2%'**: Filtra los clientes cuyos documentos comienzan con el dígito '2'. El operador LIKE se utiliza para realizar búsquedas basadas en patrones, y el % actúa como un comodín que permite cualquier secuencia de caracteres después del '2'.
  - **id_metodo_pago != 1**: Filtra los clientes cuyo método de pago (id_metodo_pago) no es igual a 1. Aquí, != significa "diferente de".

Lógica:

- **WHERE**: Se utiliza para aplicar condiciones a los registros que se seleccionan. En este caso, ambas condiciones deben ser verdaderas para que un registro sea incluido en los resultados:
  - La primera condición (documento LIKE '2%') asegura que solo se seleccionen clientes cuyos documentos comienzan con '2'.
  - La segunda condición (id_metodo_pago != 1) asegura que solo se incluyan clientes cuyo método de pago no es el que tiene id_metodo_pago igual a 1.

No se utilizan joins ni subconsultas en esta consulta específica. Solo se aplican filtros directos sobre la tabla cliente para obtener los registros que cumplen con ambas condiciones.

4.2.3 Consulta tabla ´cliente´ IN and LIKE

1. Consulta

```
SELECT * FROM cliente
WHERE id_metodo_pago IN (2, 5) AND nombre LIKE '%Mar%';
```

2. Resultados de Consultas

La consulta selecciona todos los registros de la tabla cliente donde el id_metodo_pago es 2 o 5 y el nombre contiene la secuencia de caracteres "Mar". Los resultados mostrarán los detalles de los clientes que cumplen con estos criterios, incluyendo columnas como

id_cliente, nombre, apellido, id_tipo_documento, documento, telefono, email, direccion, id_metodo_pago, antiguedad, createdAt, y updatedAt.

id_cliente	nombre	apellido	id_tipo_documento	documento	telefono	email	id_metodo_pago	direccion	createdAt	updatedAt
2	María	González	2	98765432	9876543210	maria.gonzalez@example.com	2	Calle 2	2024-09-10 11:50:34	2024-09-10 11:52:09

3. Explicación de Consultas

Explicación:

- **SELECT * FROM cliente**: Esta parte de la consulta indica que se deben seleccionar todos los campos (columnas) de la tabla cliente.
- **WHERE id_metodo_pago IN (2, 5) AND nombre LIKE '%Mar%'**: Esta es la cláusula de filtro que restringe los resultados:
    - **id_metodo_pago IN (2, 5)**: Filtra los clientes cuyo método de pago (id_metodo_pago) es 2 o 5. El operador IN se utiliza para especificar múltiples valores en una condición WHERE. Aquí, solo se seleccionan los clientes que tienen un método de pago de los especificados.
    - **nombre LIKE '%Mar%'**: Filtra los clientes cuyo nombre contiene la secuencia de caracteres "Mar" en cualquier parte del nombre. El operador LIKE con % como comodín permite que el patrón coincida con cualquier secuencia de caracteres antes y después de "Mar".

Lógica:

- **WHERE**: Se utiliza para aplicar condiciones a los registros que se seleccionan. En este caso, ambas condiciones deben ser verdaderas para que un registro sea incluido en los resultados:
    - La primera condición (id_metodo_pago IN (2, 5)) asegura que solo se seleccionen clientes que tienen uno de los métodos de pago especificados.
    - La segunda condición (nombre LIKE '%Mar%') asegura que solo se incluyan clientes cuyo nombre contiene "Mar" en cualquier parte del texto.

No se utilizan joins ni subconsultas en esta consulta específica. Solo se aplican filtros directos sobre la tabla cliente para obtener los registros que cumplen con ambas condiciones.

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 Relacion

1. **Cliente (Entidad Fuerte)**

- **ID_Cliente** (Llave Primaria)
- **Nombre**
- **Apellido**
- **ID_Tipo_Documento** (Llave foránea, referencia a tipo_documento)
- **Documento** (Debe ser único)
- **Teléfono**
- **Email**
- **ID_Metodo_Pago** (Llave foránea, referencia a metodo_pago)
- **Dirección**
- **CreatedAt**
- **UpdatedAt**

2. **Empleado (Entidad Fuerte)**

- **ID_Empleado** (Llave Primaria)
- **Nombre**
- **Apellido**
- **ID_Tipo_Documento** (Llave foránea, referencia a tipo_documento)
- **Documento** (Debe ser único)
- **Teléfono**
- **Email**
- **Dirección**
- **ID_Cargo** (Llave foránea, referencia a cargo)
- **ID_Turno** (Llave foránea, referencia a turno)
- **Fecha_Contratacion**
- **Salario**

- **CreatedAt**
- **UpdatedAt**

3. **TipoDocumento (Entidad Fuerte)**

- **ID_Tipo_Documento** (Llave Primaria)
- **Tipo_Documento**
- **CreatedAt**
- **UpdatedAt**

4. **Cargo (Entidad Fuerte)**

- **ID_Cargo** (Llave Primaria)
- **Cargo**
- **Descripción**
- **CreatedAt**
- **UpdatedAt**

5. **Turno (Entidad Fuerte)**

- **ID_Turno** (Llave Primaria)
- **Turno**
- **Descripción**
- **CreatedAt**
- **UpdatedAt**

6. **MetodoPago (Entidad Fuerte)**

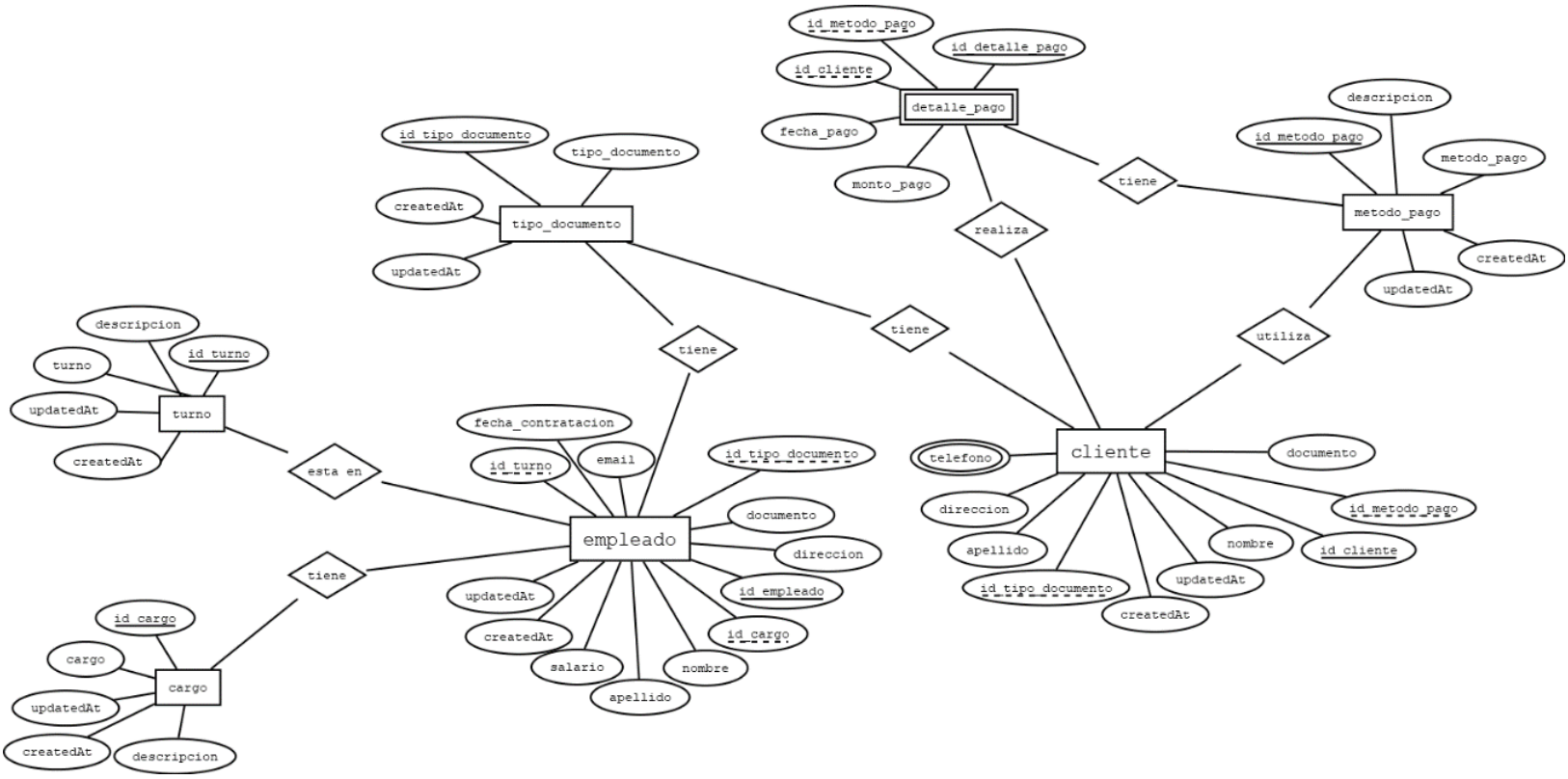
- **ID_Metodo_Pago** (Llave Primaria)
- **Metodo_Pago**
- **Descripción**
- **CreatedAt**
- **UpdatedAt**

7. **DetallePago (Entidad Débil, porque depende de Cliente)**

- **ID_Pago** (Llave Primaria)
- **ID_Cliente** (Llave foránea, referencia a cliente)
- **Fecha_Pago**
- **Monto_Pago**
- **ID_Metodo_Pago** (Llave foránea, referencia a metodo_pago)

8. **TelefonoCliente (Atributo Multivalorado, representado en una tabla separada)**

- **ID_Telefono** (Llave Primaria)
- **ID_Cliente** (Llave foránea, referencia a cliente)
- **Telefono**



4.3.2 Relaciones Normalizadas

1. Realiza (Entre Cliente y DetallePago)

   - Fecha_pago (Atributo de la relación: fecha de realización del pago)
   - Monto_pago (Atributo de la relación: cantidad pagada)

2. Tiene (Entre Empleado y Cargo)

   - Fecha_contratacion (Atributo de la relación: fecha en la que el empleado fue contratado)
   - (Atributo de la relación: salario del empleado)

3. Está en (Entre Empleado y Turno)

   - Relación entre un empleado y el turno en el que trabaja.

4. Tiene (Entre Cliente y TipoDocumento)

   - Relación entre un cliente y su tipo de documento.

5. Tiene (Entre Empleado y TipoDocumento)

   - Relación entre un empleado y su tipo de documento.

6. Utiliza (Entre Cliente y MetodoPago)

   - Relación entre un cliente y el método de pago que usa para sus compras.

7. Tiene (Entre Cliente y TelefonoCliente)

   - Relación entre un cliente y sus números de teléfono.

Estas relaciones muestran cómo se conectan las diferentes tablas en la base de datos y reflejan los vínculos entre entidades clave, como clientes, empleados, documentos y pagos.

4.3.2 Cardinalidad

1. *Cliente* (1) ---- (N) *realiza* (N) ---- (1) *DetallePago*

   - Un cliente puede realizar varios pagos (detalle_pago), pero cada pago está asociado a un único cliente.

2. *Empleado* (1) ---- (N) tiene (N) ---- (1) *Cargo*

   - Un empleado tiene un único cargo, pero varios empleados pueden tener el mismo cargo.

3. *Empleado* (1) ---- (N) *está_en* (N) ---- (1) *Turno*

   - Un empleado trabaja en un turno específico, pero varios empleados pueden compartir el mismo turno.

4. *Cliente* (1) ---- (N) *tiene* (N) ---- (1) *TipoDocumento*

- Un cliente tiene un tipo de documento, pero un tipo de documento puede estar asociado con varios clientes.

5. *Empleado* (1) ---- (N) *tiene* (N) ---- (1) *TipoDocumento*

- Un empleado tiene un tipo de documento, pero un tipo de documento puede estar asociado con varios empleados.

6. *Cliente* (1) ---- (N) *utiliza* (N) ---- (1) *MetodoPago*

- Un cliente utiliza un método de pago, pero varios clientes pueden utilizar el mismo método de pago.

8. *Cliente* (1) ---- (N) *tiene* (N) ---- (1) *TelefonoCliente*

- Un cliente puede tener varios números de teléfono, pero cada número de teléfono pertenece a un único cliente.

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, **TelefonoCliente**, 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*.