

Informe Base de datos

Presentado por:

Breiner Andres Iles Sambony

Presentado A:

Brayan Arcos Burbano

2024-agosto-20

Desarrollo de Base de Datos

TDS

Instituto Tecnológico del Putumayo

Tabla de Contenido

Contenido

Informe Base de datos	1
Tabla de Contenido	2
INFORME BASE DE DATOS	3
Instalación de las herramientas:	3
Practica realizada en clase:	3
Alter table modify:.....	3
Alter table Drop:	3
Alter table add.....	3
Insert into	3
Consultas simples:	4
Concurrencia:	4
Independencia de datos:.....	4
Redundancia de datos:	5
Consistencia de datos:.....	5
Seguridad de base de datos:	5
Tiempo de respuesta:.....	5
Presentación de la base de datos a trabajar:	5
Resumen ejecutivo:	5
Objetivos claves:.....	5
Estructura de la Base de datos:.....	5
Beneficios:	6
Desarrollo de la base de datos:	6
Desarrollo:	6
SUBCONSULTAS:.....	8
Cardinalidad:	10
Normalización de BD:	11
Referencias.	17

INFORME BASE DE DATOS

Instalación de las herramientas:

Se manejará MySQL Workbench como compilador de consultas, con el servidor local SQL server, los cual se logra al descargar las herramientas desde sus páginas oficiales.

Al terminar la descarga se da click en el ejecutable o se lo busca desde el explorador de archivos, esto para completar la instalación de ambos componentes que se usaran durante el desarrollo de las actividades.

Practica realizada en clase:

Se realizo la practica del comando

“alter table”

Se denomina que se puede usar para editar, eliminar, crear una columna de una tabla de la base de datos.

Alter table modify:



```
1 • use hospital;
2
3 • alter table pacientes modify column nombre varchar(100) default
4
5
6
```

Usando el alter table modify, podemos modificar propiedades de la columna o campo de la tabla.

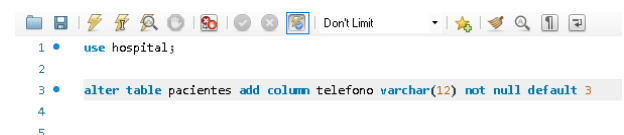
Alter table Drop:



```
1 • use hospital;
2
3 • alter table pacientes drop column telefono
4
5
```

El comando *“alter table drop”* se complementa con el comando de especificación column para eliminar una columna completa con determinado nombre.

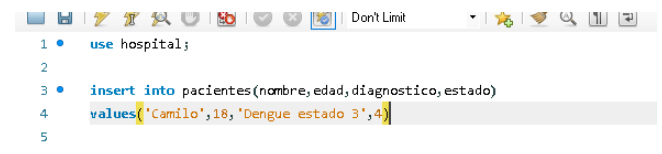
Alter table add



```
1 • use hospital;
2
3 • alter table pacientes add column telefono varchar(12) not null default 3
4
5
```

El comando *“alter table add”* como todos los alter usados durante la actividad se complementa usando el column seguido de las especificaciones de la columna o propiedades.

Insert into

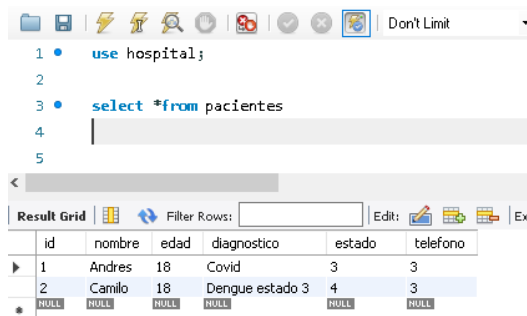


```
1 • use hospital;
2
3 • insert into pacientes(nombre, edad, diagnostico, estado)
4 values('Camilo', 18, 'Dengue estado 3', 4)
5
6
```

El comando *“Insert into”* se usa para agregar información a una columna, siguiendo especificaciones de las relaciones y demás, se llama a la tabla y dentro de los paréntesis se seleccionan que campos deseo llenar.

Consultas simples:

- Select:

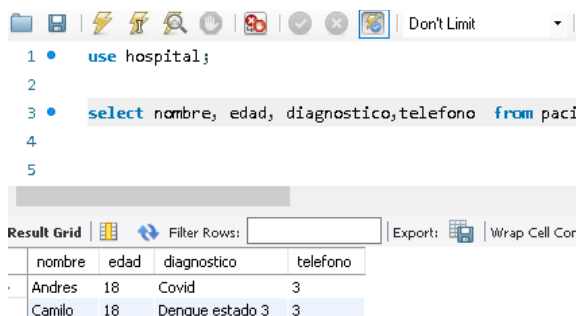


```
1 • use hospital;  
2  
3 • select *from pacientes  
4  
5
```

Result Grid

	id	nombre	edad	diagnostico	estado	telefono
▶	1	Andres	18	Covid	3	3
▶	2	Camilo	18	Dengue estado 3	4	3
*	NULL	NULL	NULL	NULL	NULL	NULL

El comando Select se usa para realizar una consulta a gran escala, nos ayuda a seleccionar los campos que queremos mostrar, con el campo “*” se indica que quiero seleccionar o ver todos los campos de la tabla, con el comando “from” indicamos de donde queremos traer los datos, es decir el nombre de la tabla.

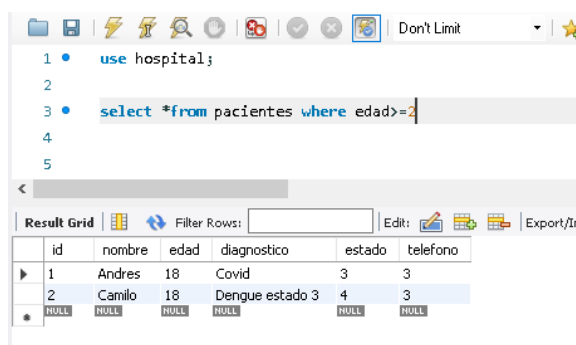


```
1 • use hospital;  
2  
3 • select nombre, edad, diagnostico,telefono from paci  
4  
5
```

Result Grid

	nombre	edad	diagnostico	telefono
▶	Andres	18	Covid	3
▶	Camilo	18	Dengue estado 3	3

En el ejemplo anterior hemos extraído datos específicos de una tabla.



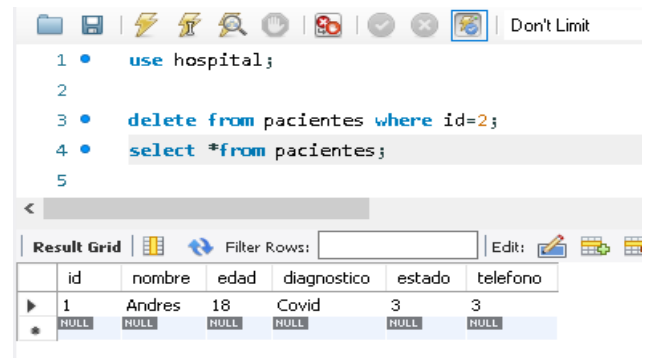
```
1 • use hospital;  
2  
3 • select *from pacientes where edad>=2  
4  
5
```

Result Grid

	id	nombre	edad	diagnostico	estado	telefono
▶	1	Andres	18	Covid	3	3
▶	2	Camilo	18	Dengue estado 3	4	3
*	NULL	NULL	NULL	NULL	NULL	NULL

En el ejemplo anterior hemos extraído todos los datos de una tabla cuando su edad es mayor o igual a 2, de la misma

manera podemos aplicar mayor lógica en cada consulta.



```
1 • use hospital;  
2  
3 • delete from pacientes where id=2;  
4 • select *from pacientes;  
5
```

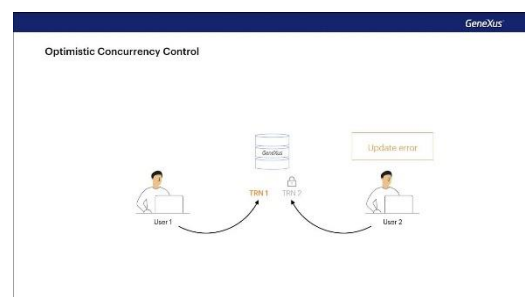
Result Grid

	id	nombre	edad	diagnostico	estado	telefono
▶	1	Andres	18	Covid	3	3
*	NULL	NULL	NULL	NULL	NULL	NULL

Se indica como eliminar un dato de una tabla o registro.

Concurrencia:

Se logra obtener un aprendizaje tal que se logra saber que la concurrencia es las técnicas o formas por las cuales se evitan los datos repetidos en las bases de datos, permitiendo acceder a múltiples dispositivos o usuarios a la información sin que hayan errores, tales como lecturas fantasmas o duplicación de datos.



Independencia de datos:

La independencia de datos se refiere a la capacidad de obtener datos y manipularlos sin conocer sus detalles, ayuda a mantener los datos separados de una manera ordenada y eficiente, los datos se pueden modificar sin afectar a demás datos.

Redundancia de datos:

La redundancia en base de datos es mayormente conocida como datos repetidos los cuales nos pueden generar algún tipo de error al momento de operar o interactuar con ellos, se centra en dividir los datos de manera que resulte mas eficiente manipularlos en el campo de base de datos.



Consistencia de datos:

Son normas estandarizadas para la creación de base de datos de manera que no afecte la base de datos y de tal manera que sea mas eficiente, logrando así una mayor rapidez y una menor carga sobre el servidor.

Seguridad de base de datos:

Como es bien sabido una base de datos maneja datos privados o un poco delicados sobre el usuario o en dado caso sobre los clientes, se centra en mantener segura nuestra base de datos de cualquier ataque, dando ciertos puntos clave en nuestra base de datos para evitar de alguna manera los ataques.

Tiempo de respuesta:

El tiempo de respuesta se centra en que la base de datos nos de una respuesta rápida y precisa logrando una mayor fluidez para nuestro programa que maneja la gestión de los datos.

Presentación de la base de datos a trabajar:

La base de datos a trabajar teniendo en cuenta los puntos anteriores es una base de datos de una tienda la cual tendría una gestión de ventas y una gestión de inventario, esta base de datos maneja una estructura robusta para cumplir con todos los puntos expuestos en lo anterior, logrando así una mayor velocidad y 0 redundancia de datos, la tienda es local de una ciudad por lo cual se evitan trabajar 2 tablas.

Resumen ejecutivo:

El proyecto tiene como objetivo diseñar, desarrollar, e implementar en lo posible una base de datos de un negocio, tienda la cual maneja múltiples productos y múltiples categorías, con manejo de clientes.

Objetivos claves:

- **Gestión de ventas:**
Lograr realizar una gestión de ventas de tal manera que se guarde información para realizar facturas.
- **Gestión de inventario:**
la gestión de inventario se centra en el manejo de entrada y salida de productos de la tienda, venta y compra(surtir).

Estructura de la Base de datos:

- **Diseño Eficiente:** La base de datos será estructurada para eliminar redundancias y mejorar la velocidad de las consultas, para un manejo eficiente de los datos.

- **Integración de Ventas e Inventario:** Ambas áreas estarán interconectadas, lo que permitirá que las ventas se reflejen automáticamente en los niveles de inventario, optimizando la actualización y el control de existencias, siendo este uno de los puntos mas importantes de la base de datos.

Beneficios:

- Información en tiempo real del stock y ventas
- Acceso a información de forma remota.
- Cero redundancias de datos, sin datos repetidos
- Facilidad de Escalabilidad: Aunque la tienda es local, la base de datos estará diseñada para permitir una posible expansión en el futuro sin comprometer su rendimiento.
- Digitalización del sistema de ventas de la tienda.

Desarrollo de la base de datos:

La base de datos se desarrolla en MySQL, siguiendo los siguientes puntos:

Tablas:

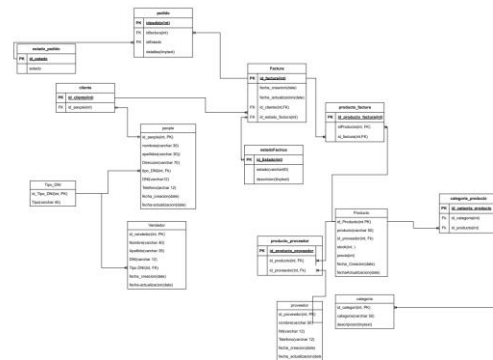
- Cliente
- Producto
- Categoría
- Empleado
- Pedido

Se toma en cuenta las tablas anteriores para comenzar a planificar la base de datos.

Desarrollo:

Se comenzará desarrollando un modelado de base de datos para seguir

una secuencia de las relaciones entre tablas para posteriormente seguir con el desarrollo en código.



Posteriormente se comienza con la codificación de la base de datos con sus entidades y atributos.

Como es común se comienza por tablas las cuales no tengan claves foráneas, es decir que no dependan de otras, por lo cual se comenzó por desarrollar las siguientes tablas.

```
##TENER EN CUENTA EL DROP DATABASE
DROP DATABASE IF EXISTS TIENDA_STORE;

CREATE DATABASE TIENDA_STORE;

use TIENDA_STORE;

CREATE TABLE TIPO_DNI (
  ID_TIPO_DNI INT AUTO_INCREMENT PRIMARY KEY,
  TIPO VARCHAR(45)
);

CREATE TABLE ROL (
  ID_ROL INT AUTO_INCREMENT PRIMARY KEY,
  ROL VARCHAR(50),
  DESCRIPCION TINYTEXT
);
```

Se tiene que tener en cuenta el DROP DATABASE debido a que cuando se comience a trabajar datos en la base de datos, ya no se podrá eliminar la base de datos como lo estamos haciendo en este momento, en ese momento se debe comenzar a trabajar con los alter table y demás.

Posteriormente se comienzan a crear tablas como lo son:

```
CREATE TABLE VENDEDOR (
  ID_VENDEDOR INT AUTO_INCREMENT PRIMARY KEY,
  NOMBRES VARCHAR(40),
  APELLIDOS VARCHAR(35),
  TIPO_DNI INT,
  DNI VARCHAR(12),
  ROL INT,
  FECHA_CREACION TIMESTAMP,
  FECHA_ACTUALIZACION DATETIME,
  FOREIGN KEY (TIPO_DNI) REFERENCES TIPO_DNI(ID_TIPO_DNI),
  FOREIGN KEY (ROL) REFERENCES ROL(ID_ROL)
);
```

```
CREATE TABLE PEOPLE(
  ID_PEOPLE INT AUTO_INCREMENT PRIMARY KEY,
  NOMBRE VARCHAR(30),
  APELLIDO VARCHAR(30),
  DIRECCION VARCHAR(70),
  TIPO_DNI INT,
  DNI VARCHAR(12),
  TELEFONO VARCHAR(12),
  FECHA_CREACION TIMESTAMP,
  FECHA_ACTUALIZACION DATETIME,
  FOREIGN KEY (TIPO_DNI) REFERENCES TIPO_DNI(ID_TIPO_DNI)
);
```

En estas tablas ya que son similares se trabajan con los mismos datos, es decir nombres, apellidos y demás pero cambian sus longitudes o propiedades de sus atributos, en el caso de people es la entidad que tiene los atributos que necesitara la otra entidad cliente, por lo cual la entidad cliente solo requiere del atributo id_people para llamarlo y guardarlo.

```
CREATE TABLE CLIENTE(
  ID_CLIENTE INT AUTO_INCREMENT PRIMARY KEY,
  ID_PEOPLE INT,
  FOREIGN KEY (ID_PEOPLE) REFERENCES PEOPLE(ID_PEOPLE)
);
```

De tal manera que la entidad queda menos saturada de datos, en cuanto a la entidad people la cual es de clientes, maneja un atributo llamado dirección, este atributo podría ser dependiente de otra entidad, pero en el caso de Mocoa se decide dejarla como atributo independiente ya que llenar todas las direcciones posibles de Mocoa sería algo demorado.

```
CREATE TABLE PROVEEDOR(
  ID_PROVEEDOR INT AUTO_INCREMENT PRIMARY KEY,
  NOMBRE VARCHAR(50),
  NIT VARCHAR(12),
  TELEFONO VARCHAR(12),
  FECHA_CREACION TIMESTAMP,
  FECHA_ACTUALIZACION DATETIME
);
```

Como se puede apreciar en lo anterior se crea la entidad proveedor la cual maneja atributos sencillos e independientes, la decisión de trabajar algunos atributos que son números como varchar es debido a que estos no se operan, como lo son los números de teléfono, por otro lado tenemos los atributos de FECHA_CREACION los cuales están siendo trabajados con la propiedad timestamp la cual guarda la fecha en la que se agrega un registro, sin necesidad de agregarle por nuestra cuenta alguna fecha

```
CREATE TABLE CATEGORIA(
  ID_CATEGORIA INT AUTO_INCREMENT PRIMARY KEY,
  CATEGORIA VARCHAR(50),
  DESCRIPCION TINYTEXT
);
```

```
CREATE TABLE PRODUCTO(
  ID_PRODUCTO INT AUTO_INCREMENT PRIMARY KEY,
  PRODUCTO VARCHAR(50),
  STOCK INT,
  PRECIO INT,
  FECHA_CREACION TIMESTAMP,
  FECHA_ACTUALIZACION DATETIME
);
```

```
CREATE TABLE CATEGORIA_PRODUCTO(
  ID_PRODUCTO_CATEGORIA INT AUTO_INCREMENT PRIMARY KEY,
  ID_CATEGORIA INT,
  ID_PRODUCTO INT,
  FOREIGN KEY (ID_CATEGORIA) REFERENCES CATEGORIA(ID_CATEGORIA),
  FOREIGN KEY (ID_PRODUCTO) REFERENCES PRODUCTO(ID_PRODUCTO)
);
```

En las anteriores tablas podemos ver el manejo de distintos datos sobre el producto, debido a que un producto puede tener muchas categorías y una categoría muchos productos se crea una entidad para relacionar ambas entidades.

```
CREATE TABLE PRODUCTO_PROVEEDOR(
  ID_PRODUCTO_PROVEEDOR INT AUTO_INCREMENT PRIMARY KEY,
  ID_PRODUCTO INT,
  ID_PROVEEDOR INT,
  FOREIGN KEY(ID_PRODUCTO) REFERENCES PRODUCTO(ID_PRODUCTO),
  FOREIGN KEY(ID_PROVEEDOR) REFERENCES PROVEEDOR(ID_PROVEEDOR))
);
```

Lo mismo sucede con los productos y sus proveedores por lo cual se decide tomar en cuenta que un proveedor maneja muchos productos y un producto muchos proveedores.

```
CREATE TABLE ESTADO_FACTURA(
  ID_ESTADO_FACTURA INT AUTO_INCREMENT PRIMARY KEY,
  ESTADO VARCHAR(50),
  DESCRIPCION TINYTEXT
);

CREATE TABLE FACTURA(
  ID_FACTURA INT AUTO_INCREMENT PRIMARY KEY,
  ID_CLIENTE INT,
  ID_ESTADO INT,
  FECHA_CREACION TIMESTAMP,
  FECHA_ACTUALIZACION DATETIME,
  FOREIGN KEY(ID_CLIENTE) REFERENCES CLIENTE(ID_CLIENTE),
  FOREIGN KEY(ID_ESTADO) REFERENCES ESTADO_FACTURA(ID_ESTADO_FACTURA)
);

CREATE TABLE PRODUCTO_FACTURA(
  ID_PRODUCTO_FACTURA INT AUTO_INCREMENT PRIMARY KEY,
  ID_PRODUCTO INT,
  ID_FACTURA INT,
  FOREIGN KEY(ID_PRODUCTO) REFERENCES PRODUCTO(ID_PRODUCTO),
  FOREIGN KEY(ID_FACTURA) REFERENCES FACTURA(ID_FACTURA)
);
```

En lo anterior podemos apreciar lo mismo, ya que una factura es simplemente datos generales, mientras que producto_factura guarda los productos y la factura a la que pertenece.

```
CREATE TABLE ESTADO_PEDIDO(
  ID_ESTADO_PEDIDO INT AUTO_INCREMENT PRIMARY KEY,
  ESTADO VARCHAR(45)
);

CREATE TABLE PEDIDO(
  ID_PEDIDO INT AUTO_INCREMENT PRIMARY KEY,
  ID_FACTURA INT,
  ID_ESTADO INT,
  DETALLES TINYTEXT,
  FOREIGN KEY(ID_FACTURA) REFERENCES FACTURA(ID_FACTURA),
  FOREIGN KEY(ID_ESTADO) REFERENCES ESTADO_PEDIDO(ID_ESTADO_PEDIDO)
);
```

Por ultimo creamos las ultimas 2 tablas de relación para complementar los datos de envío.

```
SELECT PRODUCT.id AS product_id,
PRODUCT.name AS PRODUCT_NAME, PRODUCT.STOCK, PRODUCT.PRICE, CATEGORY.name AS CATEGORY_NAME,
PRODUCT.IS_ACTIVE FROM category INNER JOIN product_category ON category.id = product_category.category_id
INNER JOIN product ON product_category.product_id = product.id;
```

Modificación de la columna precio en la tabla producto

```
ALTER TABLE PRODUCT MODIFY COLUMN PRICE FLOAT;
```

SUBCONSULTAS:

Las subconsultas dentro de sql se usan para agilizar procesos de consulta.

Métodos usados:

- **ANY:** se usa para obtener un valor si cumple con la condición, esto depende de donde se ubique la subconsulta en este caso se la usa en el where:

```
SELECT product.name
FROM product
]WHERE product.price > ANY (SELECT price FROM product
INNER JOIN productcategory ON productcategory.productId=product.id
INNER JOIN category ON productcategory.categoryId=category.id
-WHERE category.id = 10);
```

En este caso obtenemos los nombres de los productos donde su precio sea mayor al precio del producto con categoría=10; esto solo cuando con la categoría solo se encuentra relacionado un solo producto, de lo contrario se debe hacer un promedio del precio de todos los precios de los productos donde su categoría sea =10,

```
SELECT product.name
FROM product
]WHERE product.price > ANY (SELECT AVG(price) FROM product
INNER JOIN productcategory ON productcategory.productId=product.id
INNER JOIN category ON productcategory.categoryId=category.id
-WHERE category.id = 10);
```

Quedaría de esta manera, pero esta consulta se puede resumir sin necesidad del ANY, ya que al usar AVG devolverá un solo valor con el cual compara, de forma diferente el ANY se lo usa cuando una subconsulta devuelve mas de un valor

por lo cual se la podría dejar de esta manera:

```
SELECT product.name
FROM product
WHERE product.price > (SELECT AVG(price) FROM product
INNER JOIN productcategory ON productcategory.productId=product.id
INNER JOIN category ON productcategory.categoryId=category.id
WHERE category.id = 11);
```

- **EXISTS:** básicamente me dice que traiga los datos donde exista al menos un registro que cumpla con la condición:

```
SELECT c.`name`
FROM category AS c
WHERE EXISTS (SELECT 1 FROM product AS p
INNER JOIN productcategory ON productcategory.productId=p.id
INNER JOIN category ON productcategory.categoryId=category.id
WHERE productcategory.categoryId = c.id);
```

La consulta nos indica que me traiga las categorías donde exista al menos un producto.

- **IN:** indica de que sección o lugar quiere traer los datos, indicando así que traiga valores dependiendo el id, nombre, propiedades varias.

```
SELECT product.`name`
FROM product
INNER JOIN productcategory ON productcategory.productId=product.id
INNER JOIN category ON productcategory.categoryId=category.id
WHERE category.id IN (SELECT id FROM category WHERE category.`name` = 'abarrotes');
```

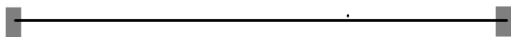
Esta consulta traerá los nombres de cada producto donde su categoría sea abarrotes.

Cardinalidad:

Cardinalidad La cardinalidad define el número de instancias de una entidad que pueden o deben estar asociadas con una instancia de otra entidad en una relación.

Simbolización

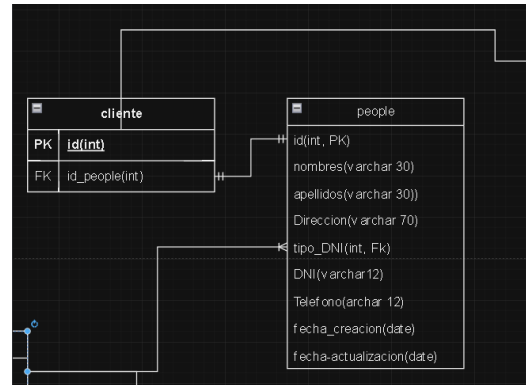
Uno a Uno (1:1): Una línea continua con un solo marcador en cada extremo.



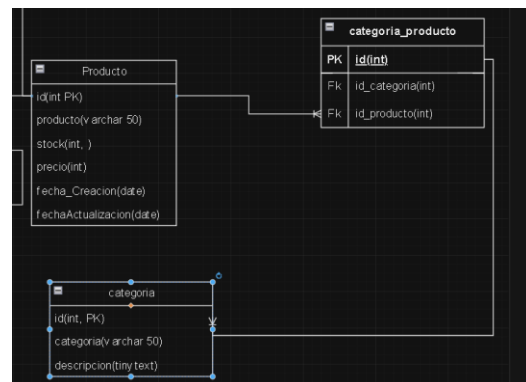
Uno a Muchos (1:N): Una línea continua con un marcador en el extremo de la entidad "uno" y un marcador en forma de flecha en el extremo de la entidad "muchos".



Muchos a Muchos (N:N): Dos líneas continuas con flechas en ambos extremos.



en la anterior tabla se puede evidenciar una relación 1 a 1 ya que un cliente es una persona, pero no pueden existir muchos clientes de la misma persona, o no con los mismos datos, de la misma manera tampoco puede haber muchas personas de un cliente.



En la anterior imagen se logra evidenciar el uso de 1 a muchos ya que un estado_pedido tiene muchos pedidos.

Normalización de BD:

1- Primera Forma Normal (1FN):

Una tabla está en primera forma si:

- Todos los atributos son atómicos. Un atributo es atómico si los elementos del dominio son simples e indivisibles.

Estudiante	Cursos
Juan	Matemáticas, física
Ana	Química



Estudiante	Cursos
Juan	Matemáticas
Juan	Física
Ana	Química

- No debe existir variación en el número de columnas.

Empleado	Habilidad 1	Habilidad 2	Habilidad 3
Juan	Ingles	Programación	
Ana	Diseño		
Pedro	Ventas	Negocios	Liderazgo



Empleado	Habilidad
Juan	Programación
juan	Inglés
Ana	Diseño
Pedro	Ventas
Pedro	Negocios
Pedro	Liderazgo

- Los campos no clave deben identificarse por la clave (dependencia funcional).

Cliente(ID)	Nombre	Teléfono	Dirección
1	Juan	555-1234	Calle A,#4
2	Ana	555-5678	Calle B,5
3	Pedro	555-9876	Calle C,D5

- Debe existir una independencia del orden tanto de las filas como de las columnas; es decir, si los datos cambian de orden no deben cambiar sus significados.

Cliente(ID)	Nombre	Teléfono	Dirección
1	Juan	555-1234	Calle A,#4
2	Ana	555-5678	Calle B,5
3	Pedro	555-9876	Calle C,D5

Id	Producto	Id_categoria	Stock	categoría
1	Pan	2	15	Comestibles
2	Agua	5	18	Bebidas

Al tener una entidad bien diseñada, y realizar una consulta, esta no debe afectar los datos de cada atributo, el orden en que se muestran estos datos, no deben afectarlas.

2- Segunda forma normal(2FN):

Dependencia funcional. Una relación está en 2FN si está en 1FN y si los atributos que no forman parte de ninguna clave dependen de forma completa de la clave principal. Es decir, que no existen dependencias parciales. Todos los atributos que no son clave principal deben depender únicamente de la clave principal.

Id_Producto	ID_Categoría	Categoría	Producto
1	2	Comestibles	Arroz
2	3	Aseo	Jabón Rey

Para cumplir con esta forma de normalización lo que se haría con esta tabla es dividirla en 2 tablas ya que por el momento existen dependencias parciales.

3- Tercera forma de normalización(3FN):

No hay dependencias transitivas: Este es el nuevo requisito de la 3FN. Un atributo no clave no debe depender de otro atributo no clave. Es decir, los atributos no clave deben depender **directamente** de la clave primaria, y no de otros atributos no clave.

En esta entidad podemos evidenciar dependencias transitivas ya que categoría depende del id_categoría y cambiaría cada que el id cambie, igualmente sucede con el producto

4- Cuarta Forma de Normalización(4FN):

No tiene dependencias multivaluadas:

Este es el nuevo criterio que introduce la 4FN. Las dependencias multivaluadas ocurren cuando un atributo en una tabla depende de más de un valor de otro atributo, lo que genera redundancia y problemas de consistencia.

Estudiante	Curso	Club
Juan	Matemáticas	Fútbol
Juan	Física	Fútbol
Juan	Matemáticas	Fútbol
Ana	Química	Ajedrez
Ana	Química	Teatro

Esta regla nos indica separar las tablas en diferentes tablas ya que esta estructura de tabla nos genera mucha redundancia.

Estudiantes:

Estudiante	Curso
Juan	Matemáticas
Juan	Física
Ana	Química

Clubes

Estudiante	Club
Juan	Fútbol
Juan	Teatro
Ana	Ajedrez
Ana	Teatro

MANUAL

la base de datos ha sido diseñado para gestionar de manera eficiente las operaciones de ventas e inventario de una tienda, con la capacidad de escalar a múltiples sucursales, tanto dentro como fuera de la ciudad, su estructura permite un crecimiento flexible y una administración centralizada de los datos.

¿Qué es una tabla?

Una tabla es una entidad la cual guarda registros tales como una factura, las categorías disponibles o los productos

- **category:** informacion de las categorías de los productos.
- **documenttype:** informacion de los tipos de identificacion usada por los diferentes usuarios.
- **invoice:** informacion de facturas generadas.
- **invoiceproduct:** productos correspondientes a las facturas generadas.
- **invoicestatus:** informacion de los estados de las facturas, "cancelado", "pagado"....
- **order:** informacion de envios de productos correspondientes a una factura.
- **orderstatus:** estado del envio correspondiente al envio.
- **paymentgatewayrecords:** respuesta del servicio de pagos.
- **paymentmethod:** información de los métodos de pago.
- **people:** información de todas las personas que hacen parte de la tienda, sea como cliente, como vendedor o como proveedor.
- **product:** información de productos de la tienda.
- **productcategory:** categorías correspondientes a su producto.
- **productdesc:** productos los cuales tienen descuento.
- **productsstore:** productos los cuales pertenecen a determinada sucursal, por lo general a la principal ya que es de una sola tienda, antes de escalar.
- **productsupplier:** proveedor de determinado productos.
- **roles:** roles de los usuarios(e.g., customer, admin).
- **store:** sucursales de la tienda(como minimo 1, la principal)
- **user:** usuarios de la tienda, es el encargado de relacionar con todas las entidades, pero solo en el caso del vendedor almacenara valores como email(user), password, para acceder al sistema.
- **userroles:** los roles de los usuarios(un usuario puede tener mas de un rol).

RELACIONES DE TABLAS(DEV):

• UNO A MUCHOS:

- category Y productcategory (una categoría puede tener muchos productos).
- documenttype y people (un tipo de documento puede tener muchas personas).
- invoice y invoiceproduct (una factura puede tener muchos productos).

- invoicestatus y invoice (una factura puede tener un solo estado, pero un estado tiene muchas facturas).
- orderstatus y order (una orden puede tener un estado, pero un estado puede tener muchas ordenes).
- paymentmethod y invoice (un metodo de pago puede tener muchas facturas, pero una factura un solo método de pago).

- **Uno a uno:**

- invoice y order (una factura puede tener un envio o una orden).
- invoice y paymentgatewayrecords (un pago por servicio externo de transferencia puede tener una sola respuesta de pago).
- people and user (una persona puede tener un usuario).
- product y productdesc (un producto puede tener un descuento).

- **Muchos a muchos:**

- Roles, user y userroles (un usuario puede tener muchos roles y un rol puede tener muchos usuarios)
- Producto, supplier y productsupplier (un producto puede tener muchos proveedores y un proveedor muchos productos).
- Producto, category y productcategory (un producto puede tener muchas categorias y

una categoría muchos productos).

- Product, store y productsstore (un producto puede estar en múltiples tiendas y una tienda tiene múltiples productos).

CONSULTAS SIMPLES:

1. Buscar productos cuando su categoría sea 'granos'

```
"SELECT p.name
FROM product AS p
INNER JOIN productcategory
AS pc ON p.id =
pc.productId
INNER JOIN category AS c
ON pc.categoryId = c.id
WHERE c.name = 'granos'"
```

2. Busca todas las facturas correspondientes a un usuario:

```
"SELECT i.id, i.createdAt
FROM invoice AS i
WHERE i.userId = 1"
```

3. Buscar los 5 productos mas vendidos:

```
"SELECT p.name,
SUM(ip.cant) AS total_sold
FROM product p
INNER JOIN invoiceproduct
AS ip ON p.id = ip.productId
GROUP BY p.name
ORDER BY total_sold DESC
LIMIT 5"
```

4. Buscar productos que tienen descuento:

```
"SELECT p.name,
pd.discount
FROM product p
INNER JOIN productdesc AS
pd ON p.id = pd.productId
WHERE pd.discount > (
SELECT AVG(discount)
FROM productdesc
)"
```

5. Buscar producto que no se han vendido en el ultimo mes:

```
“SELECT p.name
FROM product AS p
WHERE p.id NOT IN (
  SELECT ip.productId
  FROM invoiceproduct AS
ip
  INNER JOIN invoice AS i
ON ip.invoiceId = i.id
  WHERE i.createdAt >
DATE_SUB(CURDATE(),
INTERVAL 1 MONTH)
)”
```

6. Buscar productos cuando su precio es mayor al promedio de los productos en categoría “granos”:

```
“SELECT p.name, p.price
FROM product AS p
WHERE p.price > (
  SELECT AVG(p2.price)
  FROM product AS p2
  INNER JOIN
productcategory pc ON
p2.id = pc.productId
  INNER JOIN category c ON
pc.categoryId = c.id
  WHERE c.name = 'granos'
)”
```

Flujo de Información en la Base de Datos

La base de datos está diseñada para gestionar de manera eficiente las operaciones de venta e inventario de una tienda. A continuación se muestra cómo fluye la información:

1. Gestión de Productos:

- Se agregan productos a la base de datos, clasificándolos en categorías.

- Se detalla información sobre los productos, como descuentos y descripciones.
- Se asigna la disponibilidad de productos a diferentes tiendas.
- Se registran los proveedores de cada producto.

2. Gestión de Clientes:

- Se almacenan los datos personales de los clientes.
- Se crea un usuario, solo si su rol es vendedor se registrara un usuario y una contraseña de lo contrario quedara como null.

3. Ventas y Facturación:

- Los clientes realizan pedidos.
- Se generan facturas con los productos seleccionados y sus cantidades.
- Se registra el método de pago utilizado.
- Se actualiza el estado de las facturas (pagadas, canceladas, etc.).
- Se almacenan los detalles de los pagos realizados.

Flujo de una Venta:

1. Un cliente realiza un pedido:

- Se crea un nuevo registro del pedido.

2. Se procesa el pedido:

- Se seleccionan los productos del inventario.
- Se registran los detalles del pedido, como cantidades y precios.

3. Se genera una factura:

- Se crea una nueva factura, vinculándola a un pedido si es necesario y especificando el total, método de pago y cliente.

4. Se procesa el pago:

- Se registra el método de pago utilizado.
- Se almacena la información del proceso de pago.

5. Se actualiza el estado de la factura:

- Se marca la factura como pagada o cancelada.

Referencias.

- **Github:**
https://github.com/llesandres/Andres_Iles_BD
- **SQL Workbench:**
<https://www.mysql.com/products/workbench/>