Nombre de la práctica	Problemario DDL y Constraints en MySQL			No.	1
Asignatura:	Taller de Base de datos	Carrera:	Ingeniería en Sistemas Computacionales	Duración de la práctica (Hrs)	

NOMBRE DEL ALUMNO: Vanesa Hernández Martínez
GRUPO: 3501

II. Lugar de realización de la práctica (laboratorio, taller, aula u otro):
Actividades en aula de clases y en equipo personal

III. Material empleado:
- Laptop
- Navicat

Problema 1: Registro de empleados con restricciones salariales

Una empresa quiere guardar los Empleados, no cuentan con un registro como numero de empleados, anteriormente se tenia registro en un Excel con el nombre, email y salario, las reglas de la empresa no permiten un salario mensual menor a $3000 ni mayor a $50000. Crea la base de datos llamada `Ejercicio_Constraints_1` y la tabla `Empleados` que cumpla con estos requisitos.

```
CREATE TABLE Empleados(
  id_empleado INT PRIMARY KEY AUTO_INCREMENT,
  nombre VARCHAR(30) NOT NULL,
  apellido1 VARCHAR(30) NOT NULL,
  apellido2 VARCHAR(30) NOT NULL,
  email VARCHAR(30) NOT NULL CHECK(email LIKE '%_@__%.__%'),
  sueldo INT NOT NULL CHECK (sueldo >= 3000 & sueldo<=50000)
);
```

Tabla

✓ **email VARCHAR(30) NOT NULL CHECK(email LIKE '%_@%.%')**:

- **VARCHAR(30)**: Define una cadena de texto de hasta 30 caracteres.
- **NOT NULL**: No permite valores vacíos.
- **CHECK(email LIKE '%_@__%.__%')**: Verifica que el formato del correo tenga una estructura básica, como "nombre@dominio.com", asegurándose de que haya al menos un carácter antes del @, dos caracteres después, y un punto seguido de al menos dos caracteres para el dominio.

✓ **sueldo INT NOT NULL CHECK (sueldo >= 3000 AND sueldo <= 50000)**:

- **INT**: Es un número entero.
- **NOT NULL**: No permite valores vacíos.
- **CHECK (sueldo >= 3000 AND sueldo <= 50000)**: Asegura que el sueldo esté entre 3000 y 50000.

Problema 2: Relación entre empleados y departamentos

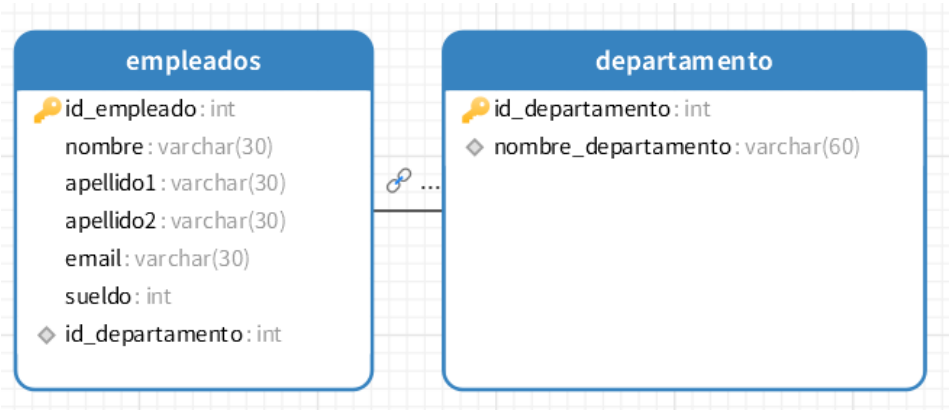
Una empresa necesita organizar a sus empleados según los departamentos a los que pertenecen. Cada departamento tiene un nombre único. La empresa quiere almacenar la información de los empleados junto con el departamento al que están asignados. Actualmente, cada empleado tiene un número de identificación `Numero de empleado` y el nombre del departamento. Se requiere crear una relación entre ambas tablas para que cada empleado esté asignado a un departamento.

Crea la base de datos llamada `Ejercicio_Constraints_2` y las tablas `Empleados` y `Departamentos` con las restricciones necesarias para cumplir con esta relación.

```sql
CREATE TABLE Departamento(
  id_departamento INT PRIMARY KEY AUTO_INCREMENT,
  nombre_departamento VARCHAR(60) NOT NULL UNIQUE
);

CREATE TABLE Empleados(
  id_empleado INT PRIMARY KEY AUTO_INCREMENT,
  nombre VARCHAR(30) NOT NULL,
  apellido1 VARCHAR(30) NOT NULL,
  apellido2 VARCHAR(30) NOT NULL,
  email VARCHAR(30) NOT NULL CHECK(email LIKE '%_@__%.__%'),
  sueldo INT NOT NULL CHECK (sueldo >= 3000 & sueldo<=50000),
  id_departamento INT,
  FOREIGN KEY (id_departamento) REFERENCES departamento(id_departamento) ON UPDATE CASCADE ON DELETE RESTRICT
);
```

Tablas



- ✓ **nombre_departamento VARCHAR(60) NOT NULL UNIQUE**:
  - **VARCHAR(60)**: Es una cadena de texto de hasta 60 caracteres.
  - **NOT NULL**: El campo no puede estar vacío, debe tener siempre un valor.
  - **UNIQUE**: Garantiza que no se repitan valores en esta columna, es decir, cada departamento debe tener un nombre único.

Problema 3: Control de inventario de productos

Una tienda quiere llevar el control de los productos que vende. La información almacenada en la tabla de `Productos` debe incluir el nombre del producto, código de barras y su precio. A su vez, necesitan añadir una columna para el stock de cada producto, la cual no puede ser nula y debe tener un valor por defecto de 100. Además, el precio de los productos debe ser siempre mayor a 0, y el nombre de cada producto debe ser único para evitar duplicados.

Crea la base de datos llamada `Ejercicio_Constraints_3` y realiza las modificaciones necesarias en la tabla `Productos` para cumplir con estos requisitos.

```sql
CREATE TABLE Producto(
    id_producto INT PRIMARY KEY AUTO_INCREMENT,
    nombre VARCHAR(100) NOT NULL UNIQUE,
    codigo_barras VARCHAR(100) NOT NULL UNIQUE,
    precio INT NOT NULL CHECK(precio > 0),
    stock INT NOT NULL DEFAULT(100)
);
```

Tabla

- ✓ **UNIQUE**: Esta restricción asegura que los valores en una columna sean únicos en toda la tabla, es decir, no se permiten duplicados. Se utiliza para garantizar que no haya registros con el mismo valor en campos específicos, como nombres de usuarios o códigos de productos.
- ✓ **CHECK**: Se utiliza para imponer una condición sobre los valores de una columna. Solo se permiten valores que cumplan la condición especificada. Por ejemplo, usamos CHECK(precio > 0) para asegurarnos de que el precio de un producto siempre sea mayor a cero.
- ✓ **DEFAULT**: Define un valor por defecto para una columna cuando no se proporciona ningún valor al insertar un nuevo registro. Si no se especifica un valor, se asignará el valor por defecto. Por ejemplo, DEFAULT(100) puede establecer que el stock inicial de un producto sea 100 si no se indica otro valor.

Problema 4: Control de pedidos con validación de montos

Una empresa quiere registrar los pedidos que recibe, pero necesita asegurarse de que el total de cada pedido sea proporcional a la cantidad de productos solicitados. Específicamente, el `total` de cada pedido debe ser al menos igual a la cantidad de productos multiplicada por 10. Además, cada pedido debe contener al menos 1 producto.

Crea la base de datos llamada `Ejercicio_Constraints_4` y la tabla `Pedidos` que cumpla con estas validaciones usando restricciones `CHECK` .

```sql
CREATE TABLE Pedido(
   id_pedido INT PRIMARY KEY AUTO_INCREMENT,
     cantidad_productos INT NOT NULL CHECK (cantidad_productos >= 1),
     total INT NOT NULL,
     CHECK (total >= cantidad_productos * 10)
 );
```

Tabla

- ✓ **cantidad_productos INT NOT NULL CHECK (cantidad_productos >= 1)**:
    - **INT**: Define que el campo almacena números enteros.
    - **NOT NULL**: El campo no puede estar vacío, siempre debe tener un valor.
    - **CHECK (cantidad_productos >= 1)**: Impone una restricción para asegurarse de que la cantidad de productos sea al menos 1, evitando valores negativos o cero.
- ✓ **total INT NOT NULL**:
    - **INT**: El campo almacena un número entero.
    - **NOT NULL**: El valor del total no puede ser nulo, debe tener siempre un valor.
- ✓ **CHECK (total >= cantidad_productos * 10)**:
    - Esta restricción asegura que el total debe ser al menos 10 unidades monetarias por cada producto. Es decir, el valor total debe ser igual o mayor que la cantidad de productos multiplicada por 10, lo que garantiza un precio mínimo por producto.

Problema 5: Control de ventas de productos por empleados

Una empresa de ventas necesita registrar las ventas que realiza. Cada venta está asociada a un empleado y a un producto específico. Para garantizar la integridad de los datos, se requiere que cada venta tenga la referencia tanto del empleado como del producto, y que las ventas sean realizadas en una fecha válida (no futura). Además, la cantidad de productos vendidos debe ser mayor a 0.

Crea la base de datos llamada `Ejercicio_Constraints_5` y las tablas necesarias para almacenar esta información, incluyendo las claves foráneas para relacionar las tablas de empleados y productos con las ventas.

```sql
CREATE TABLE Empleado (
    id_empleado INT PRIMARY KEY AUTO_INCREMENT,
    nombre VARCHAR(100) NOT NULL,
    apellido1 VARCHAR(100) NOT NULL,
    apellido2 VARCHAR(100) NOT NULL
);

CREATE TABLE Producto (
    id_producto INT PRIMARY KEY AUTO_INCREMENT,
    nombre VARCHAR(100) NOT NULL,
    precio DECIMAL(10,2) NOT NULL
);


CREATE TABLE Venta (
    id_venta INT PRIMARY KEY AUTO_INCREMENT,
    id_empleado INT,
    id_producto INT,
    cantidad INT CHECK (cantidad > 0),
    fecha_venta DATE DEFAULT (CURRENT_DATE),
    FOREIGN KEY (id_empleado) REFERENCES empleado(id_empleado) ON UPDATE CASCADE ON DELETE RESTRICT,
    FOREIGN KEY (id_producto) REFERENCES producto(id_producto) ON UPDATE CASCADE ON DELETE RESTRICT
);
```

Tablas

- ✓ **precio DECIMAL(10,2) NOT NULL**:
  - **DECIMAL(10,2)**: Define un número decimal con hasta 10 dígitos en total, de los cuales 2 están después del punto decimal.
  - **NOT NULL**: No permite valores vacíos, por lo que siempre debe tener un precio asignado.
- ✓ **cantidad INT CHECK (cantidad > 0)**:
  - **INT**: Define un campo de números enteros.
  - **CHECK (cantidad > 0)**: Asegura que la cantidad de productos sea mayor que 0, impidiendo que se registre una cantidad negativa o nula.
- ✓ **fecha_venta DATE DEFAULT (CURRENT_DATE)**:
  - **DATE**: Es un tipo de dato que almacena una fecha (año, mes, día).
  - **DEFAULT (CURRENT_DATE)**: Establece que, si no se proporciona una fecha de venta al registrar un producto, se asignará automáticamente la fecha actual (el día en que se realiza la inserción).

Conclusión

Los **constraints** (restricciones) en bases de datos son reglas que se aplican a las columnas de una tabla para garantizar la **integridad** y **consistencia** de los datos.:

1. **UNIQUE**: Evita la duplicación de valores en una columna, asegurando que los datos sean únicos, como nombres de usuario o identificadores.
2. **CHECK**: Impone condiciones específicas sobre los valores de una columna, validando que cumplan ciertos criterios, como que un número sea positivo o que una fecha esté en un rango válido.
3. **DEFAULT**: Asigna un valor predeterminado cuando no se proporciona uno, lo que facilita la inserción de datos consistentes sin necesidad de introducir valores manualmente en cada inserción.
4. **NOT NULL**: Asegura que los campos siempre tengan un valor, evitando datos vacíos en columnas donde la información es crítica.
5. **PRIMARY KEY**: Define una clave única para cada fila, garantizando que cada registro sea identificable de manera única.

En conjunto, estas restricciones ayudan a proteger la base de datos contra **inconsistencias**, **errores** y **datos duplicados o inválidos**, mejorando la fiabilidad y precisión de la información almacenada.