Solución de la guía taller hasta el 3.2

Grupo #6 :

Camilo Andrés Parra Cuenca

Daniela Alexandra Herrera Aponte

Edwin Eleider Amaya

Universidad Antonio Nariño

Gestión de requisitos de software

08/05/2025

Escogimos el proyecto #12 sobre la banda

WORKFLOW DE REQUERIMIENTOS

1.1. Describa y justifique el problema, indicando porque se trata de un problema soluble (referencia pág. 5):

Problema:	Se requiere una aplicación que administre la información de los integrantes de una banda musical y las ganancias obtenidas en los conciertos. La falta de un sistema organizado dificulta el control de seguidores, conciertos y distribución de ingresos, lo que puede generar errores en la gestión financiera de la banda.
Cliente	Banda musical conformada por cuatro integrantes
Usuario	Integrantes de la banda y su administrador.
Requerimiento funcional	Visualizar la información de los integrantes. Agregar seguidores a un integrante. Registrar un concierto y repartir las ganancias. Consultar el total de seguidores de la banda.
Mundo del problema	Entorno musical donde la gestión de ingresos y estadísticas de los integrantes es esencial para el crecimiento de la banda.
Requerimiento no funcional	La aplicación debe ser intuitiva y fácil de usar. Debe garantizar la seguridad de los datos. La interfaz debe ser accesible desde dispositivos móviles. Se requiere almacenamiento persistente de datos.

1.2. Señale mínimo 4 requerimientos funcionales indicando el tipo de dato que existen en el lenguaje de implementación para las variables que serán utilizadas en las entradas y salidas del requerimiento (Pág. 11-12):

REQUERIMIENTO FUNCIONAL 1	Nombre	Visualizar información de los integrantes
	Resumen	Permite consultar los datos de cada integrante.

REQUERIMIENTO FUNCIONAL 2	Entradas	
	Salidas	Nombre (string), Instrumento (string), Imagen (string), Seguidores (int), Conciertos (int), Ganancias (float)
	Nombre	Agregar seguidores a un integrante
	Resumen	Incrementa la cantidad de seguidores de un integrante.
	Entradas	Nombre del integrante (string), Cantidad de seguidores a agregar (int
	Salidas	Nueva cantidad de seguidores (int)
REQUERIMIENTO FUNCIONAL 3	Nombre	Registrar un concierto
	Resumen	Registra un concierto y distribuye equitativamente las ganancias.
	Entradas	Monto total de ganancias (float)
	Salidas	Ganancias actualizadas por integrante (float)
REQUERIMIENTO FUNCIONAL 4	Nombre	Consultar el total de seguidores de la banda
	Resumen	Muestra la cantidad total de seguidores de todos los integrantes.
	Entradas	
	Salidas	Total de seguidores (int)

1. WORKFLOW DE ANÁLISIS Y DISEÑO

ANALISIS DE ATRIBUTOS:

2.1. Identifique las entidades del mundo problema, mínimo 3 (consulta como referencia la pág. 16, J. Villalobos) RUBBY CASALLAS GUTIERREZ, JORGE ALBERTO VILLALOBOS SALCEDO, "Fundamentos de Programación: Aprendizaje Activo Basado en Casos" En: México 2006. Ed:Pearson Education ISBN: 970-26-0846-5

ENTIDADES (Las que necesites)	NOMBRE	DESCRIPCIÓN
1	Integrante	Miembro de la banda con atributos como nombre, instrumento, imagen, seguidores, conciertos y ganancias.
2	Concierto	Evento en el que la banda se presenta y obtiene ganancias.

3	Banda	Agrupación que reúne a los cuatro integrantes y gestiona su información.

2.2. Señale las características de las entidades descritas:

Entidad: Integrante

Valor Posible	Tipo de Dato (C++)	Explicación
Nombre	string	Almacena el nombre del integrante.
Instrumento	string	Indica el instrumento que toca.
Imagen	string	Ruta de la imagen del integrante.
Seguidores	int	Cantidad de seguidores.
Conciertos	int	Número de conciertos realizados.
Ganancias	float	Total de dinero ganado.

Entidad: Concierto

Valor Posible	Tipo de Dato (C++)	Explicación
Fecha	string	Indica la fecha del concierto.
Ganancias	float	Representa el monto total obtenido.
Integrantes	array	Lista de integrantes que participaron.

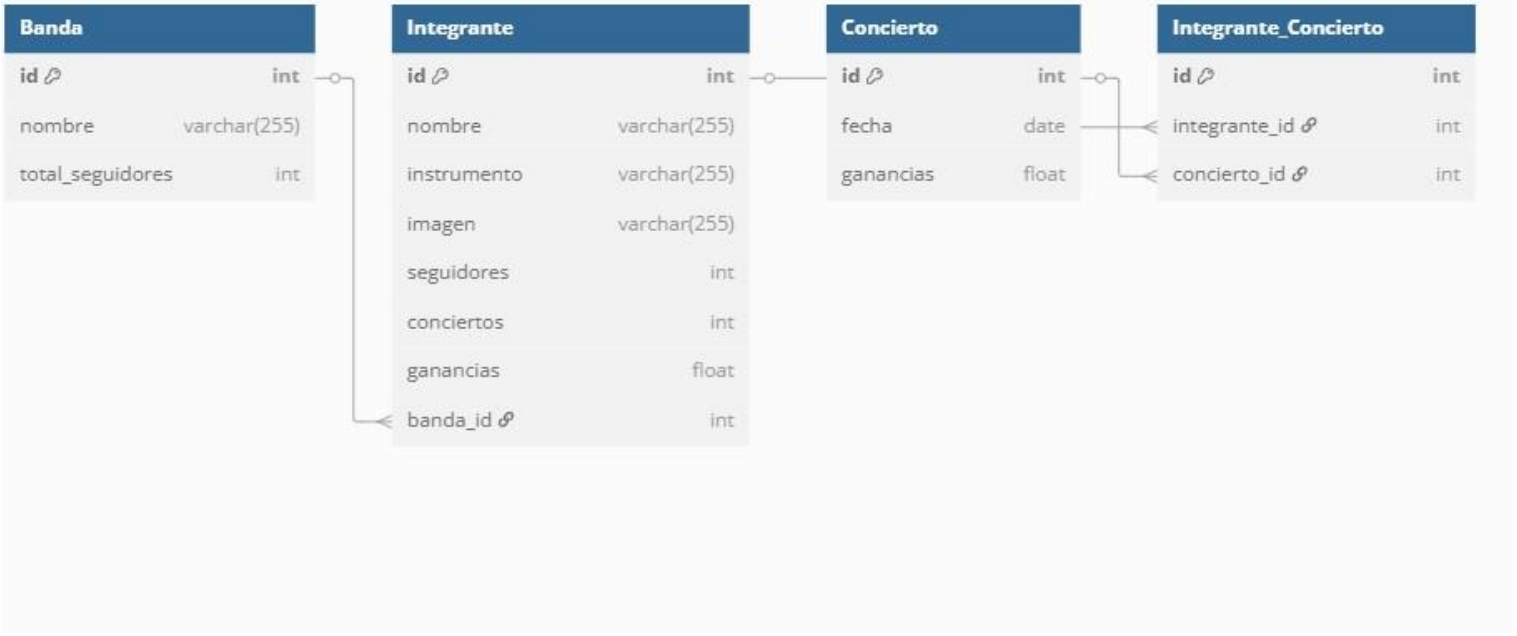
Entidad: Banda

Valor Posible	Tipo de Dato (C++)	Explicación
Nombre	string	Nombre de la banda.
Integrantes	array	Lista de los integrantes de la banda.
Total de seguidores	int	Suma de los seguidores de todos los integrantes.

2.3. Establezca las relaciones entre las entidades de forma lógica, en un esquema gráfico, estableciendo las entidades y las relaciones (investiga: Diagrama E/R) (Pág. 21-22):

Por favor utilizar solo uno de los siguientes recursos en línea

- http://creately.com/Draw-UML-and-Class-Diagrams-Online (Entity Relationship UML)
- https://www.gliffy.com/examples/flowcharts
- https://cacoo.com/signin
- https://online.visual-paradigm.com/es/solutions/free-flowchart-maker/ ❑ https://www.goconqr.com/es/users/sign_up


Banda
id int
nombre varchar(255)
total_seguidores int
Integrante
id int
nombre varchar(255)
instrumento varchar(255)
imagen varchar(255)
seguidores int
conciertos int
ganancias float
banda_id int
Concierto
id int
fecha date
ganancias float
Integrante_Concierto
id int
integrante_id int
concierto_id int

DISEÑO:

2.4. Por cada requerimiento funcional desarrollado en el numeral 1.2, crea un diagrama de flujo: ❑ https://www.gliffy.com/examples/flowcharts

Diagrama 1: Visualizar información de los integrantes

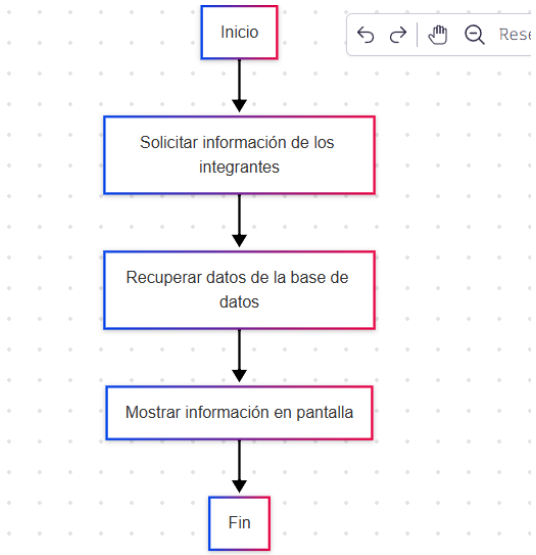


Diagrama 2: Agregar seguidores a un integrante

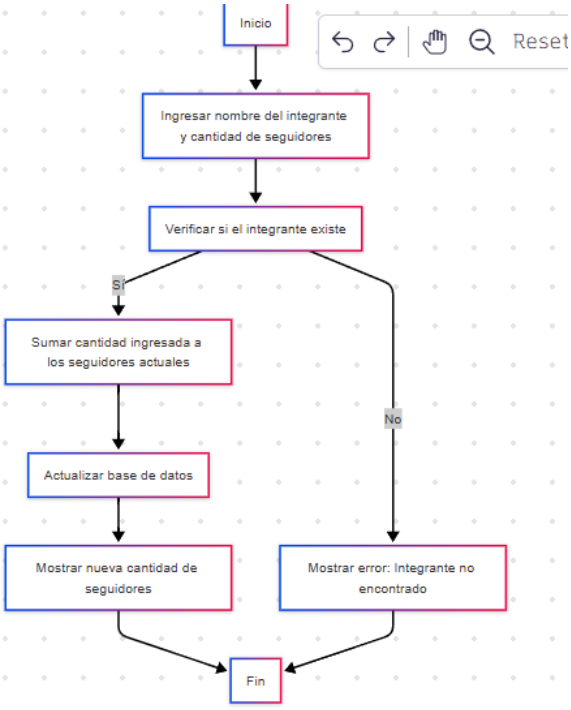


Diagrama 3: Registrar un concierto

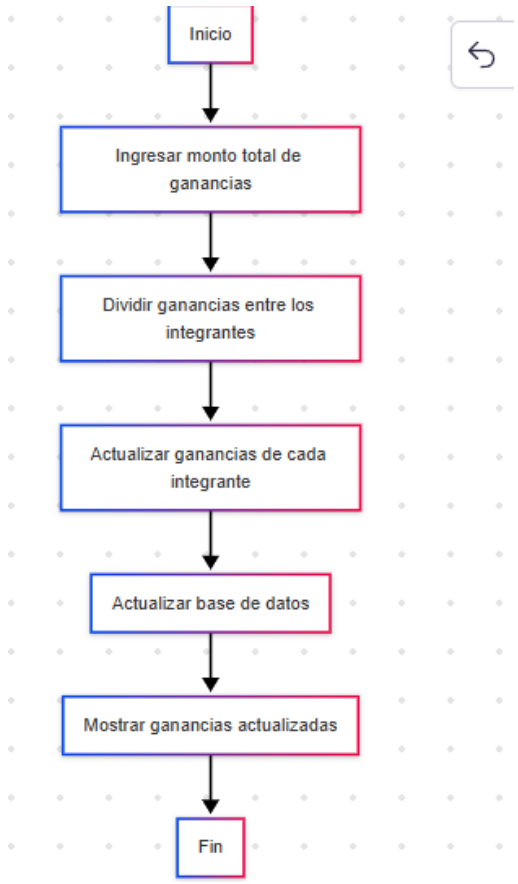
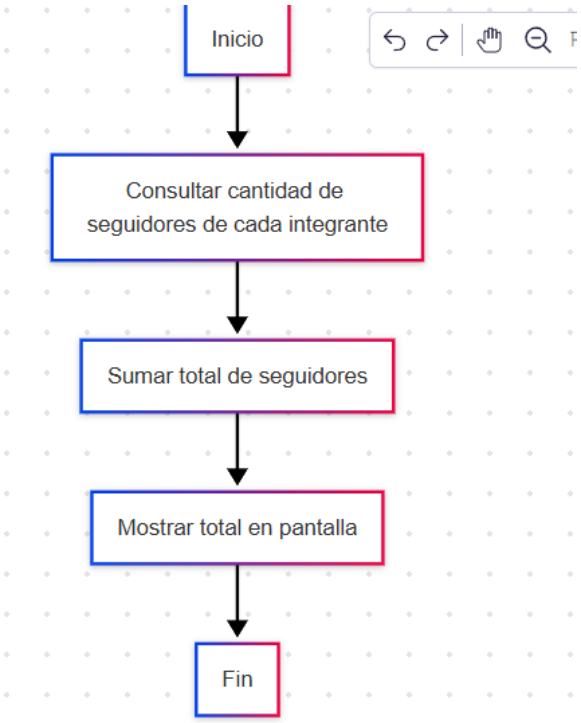
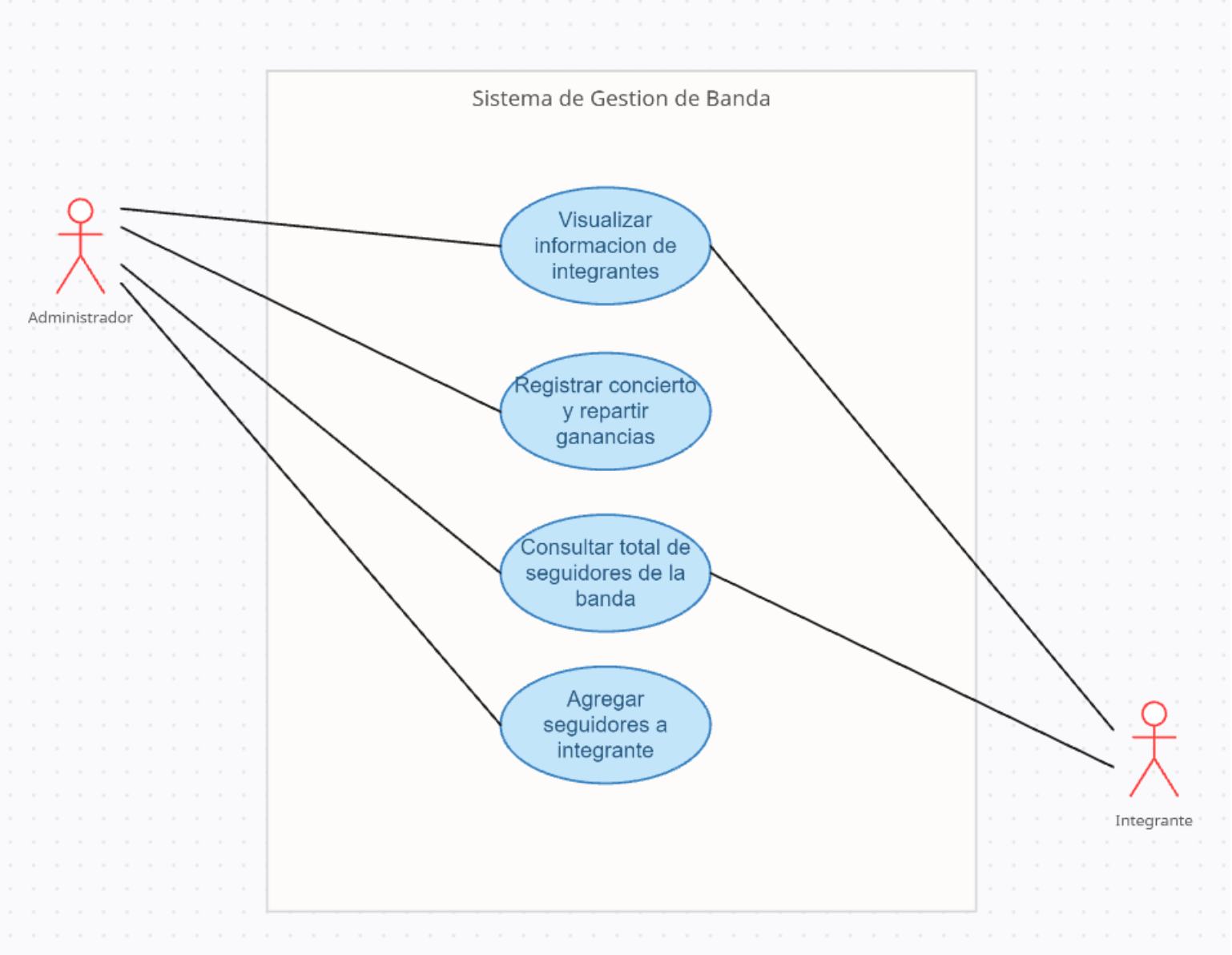


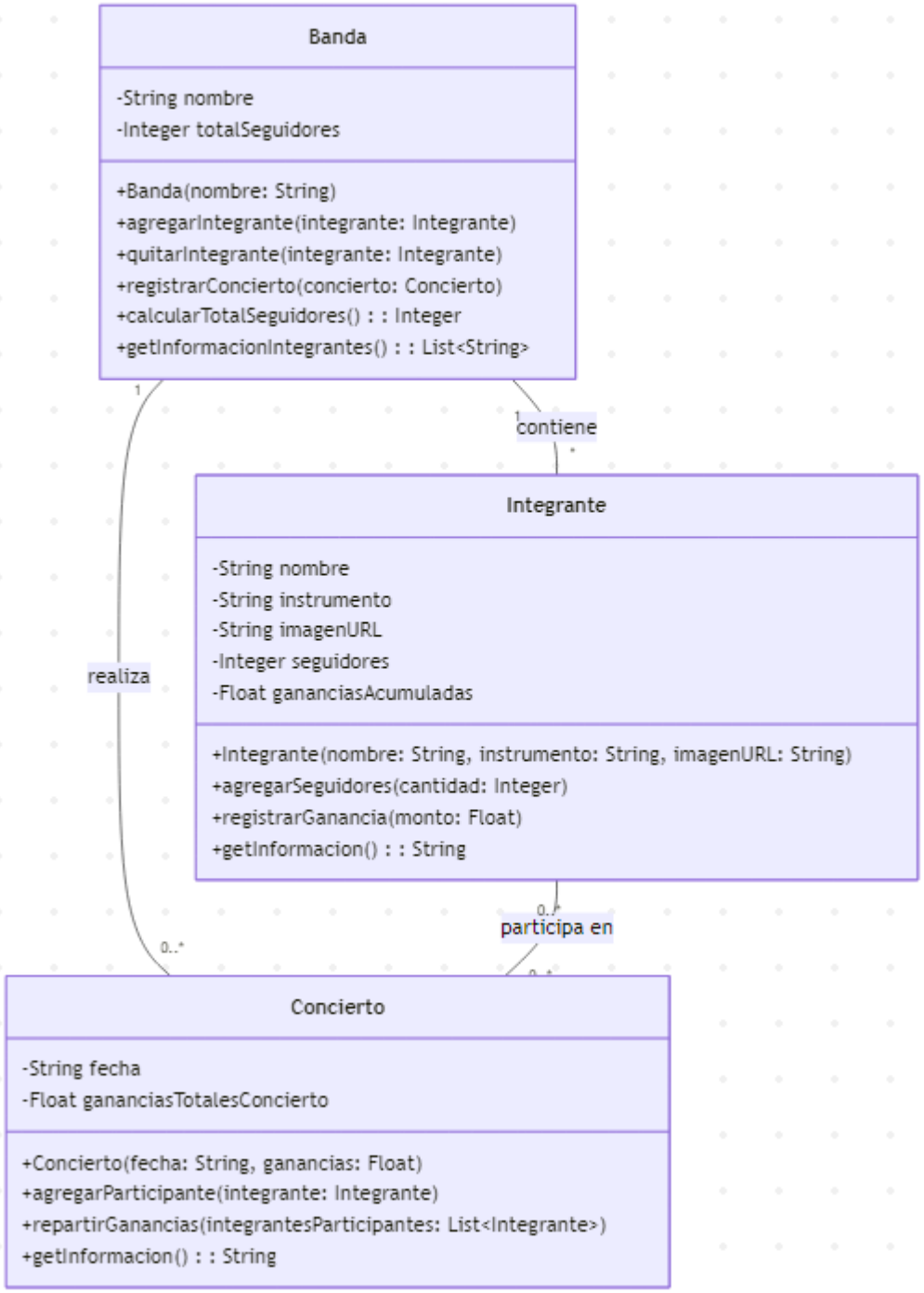
Diagrama 4: Consultar el total de seguidores de la banda



2.4 Diagrama de casos de uso:



2.5 Diagrama de clases:



1. WORKFLOW DE IMPLEMENTACIÓN:

3.1 Declaración de las funciones

Nombre:	__init__	{ self.server = 'DESKTOP-BF307G4\\SQLEXPRESS01' self.database = 'BandaDB' self.driver = '{ODBC Driver 17 for SQL Server}' self.username = '' self.password = '' try: self.conexion = pyodbc.connect(f"DRIVER={self.driver};SERVER={self.server};DATABASE={self.database};Trusted_Connection=yes;" |
Parametros:	Ninguno explícito (el parámetro `self` es implícito para los métodos de instancia).	

Retorno:	Nada (es un constructor).	) self.cursor = self.conexion.cursor() print(" Conexión exitosa a la base de datos.") except Exception as e: print(" Error de conexión:") self.conexion = None }
Descripción:	Constructor de la clase `BandaDB`. Inicializa los atributos de configuración para la conexión a SQL Server (servidor, base de datos, driver, etc.) e intenta establecer la conexión. Imprime un mensaje indicando si la conexión fue exitosa o si ocurrió un error.	

Nombre:	agregar_banda	{
		if self.conexion:
		try:
		query = "INSERT INTO Banda (nombre, total_seguidores) VALUES (?,)"
		self.cursor.execute(query, (nombre, total_seguidores))
		self.conexion.commit()
		print(f"Banda '{nombre}' agregada con éxito.")
		except Exception as e:
		print("Error al agregar banda:")
		}

Parametros:	- nombre (string): El nombre de la banda a agregar. - total_seguidores (int, opcional, default=0): El número total de seguidores de la banda.	
Retorno:	Nada (imprime un mensaje en consola indicando el éxito o error de la operación).	
Desripción:	Agrega una nueva banda a la tabla `Banda` en la base de datos. Si la conexión está activa, ejecuta una consulta INSERT con el nombre y total de seguidores proporcionados	

Nombre:	agregar_integrante	{
		if self.conexion:
		try:
		query = """
		INSERT INTO Integrante (nombre, instrumento, imagen, seguidores, conciertos, ganancias, banda_id)
		VALUES (?, ?, ?, ?, ?, ?, ?)
		"""
		self.cursor.execute(query, (nombre, instrumento, imagen, seguidores, conciertos, ganancias, banda_id))
		self.conexion.commit()
		print(f" Integrante '{nombre}' agregado con éxito.")
		except Exception as e:
		print(" Error al agregar integrante:")
		}

Parametros:	- nombre (string): Nombre del integrante. - instrumento (string): Instrumento que toca el integrante. - imagen (string): Ruta o nombre del archivo de imagen del integrante. - seguidores (int): Número de seguidores del integrante. - conciertos (int): Número de conciertos en los que ha participado. - ganancias (float): Ganancias generadas por el integrante. - banda_id (int): ID de la banda a la que pertenece el integrante.	
Retorno:	Nada (imprime un mensaje en consola indicando el éxito o error de la operación).	
Descripción:	Agrega un nuevo integrante a la tabla `Integrante` en la base de datos, asociándolo a una banda existente mediante `banda_id`.	

Nombre:	agregar_concierto	{
		if self.conexion:
		try:
		query = "INSERT INTO Concierto (fecha, ganancias) VALUES (?, ?)"
		self.cursor.execute(query, (fecha, ganancias))
		self.conexion.commit()
		print(f"Concierto del {fecha} agregado con éxito.")
		except Exception as e:
		print(" Error al agregar concierto:")
		}

Parametros:	- fecha (string): Fecha del concierto (ej: "YYYY-MM-DD"). - ganancias (float, opcional, default=0): Ganancias obtenidas en el concierto.	

Retorno:	Nada (imprime un mensaje en consola indicando el éxito o error de la operación).	
Descripción:	Registra un nuevo concierto en la tabla `Concierto` de la base de datos con la fecha y ganancias especificadas.	

Nombre:	registrar_asistencia	{
		if self.conexion:
		try:
		query = "INSERT INTO Integrante_Concierto (integrante_id, concierto_id) VALUES (?, ?)"
		self.cursor.execute(query, (integrante_id, concierto_id))
		self.conexion.commit()
		print(f" Integrante {integrante_id} registrado en el concierto {concierto_id}.")
		except Exception as e:
		print(" Error al registrar asistencia:")
		}

Parametros:	- integrante_id (int): ID del integrante que asistió. - concierto_id (int): ID del concierto al que asistió el integrante.	
Retorno:	Nada (imprime un mensaje en consola indicando el éxito o error de la operación).	
Descripción:	Registra la asistencia de un integrante a un concierto específico en la tabla de relación 'Integrante_Concierto'.	

Nombre:	obtener_integrantes	{
		if self.conexion:
		try:
		query = "SELECT * FROM Integrante"
		self.cursor.execute(query)
		integrantes = self.cursor.fetchall()
		for integrante in integrantes:
		print(f" ID: {integrante[0]}, Nombre: {integrante[1]}, Instrumento: {integrante[2]}, Seguidores: {integrante[4]}")
		except Exception as e:
		print(" Error al obtener integrantes:")
		}

Parametros:	Ninguno explicito	
Retorno:	Nada (imprime la informacion de los integrantes en consola)	
Descripción:	Obtiene todos los registros de la tabla `Integrante` y los imprime en la consola, mostrando ID, nombre, instrumento y número de seguidores de cada uno.	

Nombre:	obtener_conciertos	{
		if self.conexion:
		try:
		query = "SELECT * FROM Concierto"
		self.cursor.execute(query)

Parametros:	Ninguno explicito	

Retorno:	Nada (imprime la informacion de los conciertos en consola)	conciertos = self.cursor.fetchall()
		for concierto in conciertos:
		print(f"ID: {concierto[0]}, Fecha: {concierto[1]}, Ganancias: {concierto[2]}")
		except Exception as e:
		print("Error al obtener conciertos:")
		}

Descripción:	Obtiene todos los registros de la tabla `Concierto` y los imprime en la consola, mostrando ID, fecha y ganancias de cada uno.	

Nombre:	cerrar_conexion	{
		if self.conexion:
		self.conexion.close()
		print(" Conexión cerrada.")
		}

Parametros:	Ninguno explicito	
Retorno:	Nada (imprime un mensaje en consola).	
Descripción:	Cierra la conexión activa con la base de datos SQL Server, si existe.	

3.2 Codigo fuente

```
import pyodbc

class BandaDB:
    def __init__(self):
        """ Configuración de la conexión a SQL Server """
        self.server = 'DESKTOP-BF307G4\\SQLEXPRESS01'  # Nombre del servidor poner el nombre de tu servidor si es local el nombre del pc mas sqlexpress01
        self.database = 'BandaDB'
        self.driver = '{ODBC Driver 17 for SQL Server}'  # Driver para SQL Server
        self.username = ''  # Usuario de SQL Server (si usas autenticación SQL)
        self.password = ''  # Contraseña (si usas autenticación SQL)

        try:
            self.conexion = pyodbc.connect(
                f"DRIVER={self.driver};SERVER={self.server};DATABASE={self.database};Trusted_Connection=yes;"
            )
            self.cursor = self.conexion.cursor()
            print("✅ Conexión exitosa a la base de datos.")
        except Exception as e:
            print("❌ Error de conexión:", e)
            self.conexion = None

    def agregar_banda(self, nombre, total_seguidores=0):
        """ Agrega una nueva banda a la base de datos """
        if self.conexion:
            try:
                query = "INSERT INTO Banda (nombre, total_seguidores) VALUES (?, ?)"
                self.cursor.execute(query, (nombre, total_seguidores))
                self.conexion.commit()
                print(f"🎸 Banda '{nombre}' agregada con éxito.")
            except Exception as e:
                print("❌ Error al agregar banda:", e)

    def agregar_integrante(self, nombre, instrumento, imagen, seguidores, conciertos, ganancias, banda_id):
        """ Agrega un integrante a la banda """
```

```
    if self.conexion:
        try:
            query = """
            INSERT INTO Integrante (nombre, instrumento, imagen, seguidores, conciertos, ganancias, banda_id)
            VALUES (?, ?, ?, ?, ?, ?, ?)
            """
            self.cursor.execute(query, (nombre, instrumento, imagen, seguidores, conciertos, ganancias, banda_id))
            self.conexion.commit()
            print(f"🎤 Integrante '{nombre}' agregado con éxito.")
        except Exception as e:
            print("❌ Error al agregar integrante:", e)

def agregar_concierto(self, fecha, ganancias=0):
    """ Registra un nuevo concierto """
    if self.conexion:
        try:
            query = "INSERT INTO Concierto (fecha, ganancias) VALUES (?, ?)"
            self.cursor.execute(query, (fecha, ganancias))
            self.conexion.commit()
            print(f"🎶 Concierto del {fecha} agregado con éxito.")
        except Exception as e:
            print("❌ Error al agregar concierto:", e)

def registrar_asistencia(self, integrante_id, concierto_id):
    """ Registra la asistencia de un integrante a un concierto """
    if self.conexion:
        try:
            query = "INSERT INTO Integrante_Concierto (integrante_id, concierto_id) VALUES (?, ?)"
            self.cursor.execute(query, (integrante_id, concierto_id))
            self.conexion.commit()
            print(f"✓ Integrante {integrante_id} registrado en el concierto {concierto_id}.")
        except Exception as e:
            print("❌ Error al registrar asistencia:", e)

def obtener_integrantes(self):
    """ Obtiene todos los integrantes de la banda """
    if self.conexion:
        try:
            query = "SELECT * FROM Integrante"
            self.cursor.execute(query)
            integrantes = self.cursor.fetchall()
            for integrante in integrantes:
                print(f"🎸 ID: {integrante[0]}, Nombre: {integrante[1]}, Instrumento: {integrante[2]}, Seguidores: {integrante[4]}")
        except Exception as e:
            print("❌ Error al obtener integrantes:", e)

def obtener_conciertos(self):
    """ Obtiene todos los conciertos registrados """
    if self.conexion:
        try:
            query = "SELECT * FROM Concierto"
            self.cursor.execute(query)
            conciertos = self.cursor.fetchall()
            for concierto in conciertos:
                print(f"🎤 ID: {concierto[0]}, Fecha: {concierto[1]}, Ganancias: {concierto[2]}")
        except Exception as e:
```

```
            print("❌ Error al obtener conciertos:", e)

    def cerrar_conexion(self):
        """ Cierra la conexión con la base de datos """
        if self.conexion:
            self.conexion.close()
            print("🔌 Conexión cerrada.")


if __name__ == "__main__":
    db = BandaDB()

    if db.conexion:
        while True:
            print("\n📌 Menú de la aplicación:")
            print("1. Agregar banda")
            print("2. Agregar integrante")
            print("3. Agregar concierto")
            print("4. Registrar asistencia")
            print("5. Ver integrantes")
            print("6. Ver conciertos")
            print("7. Salir")

            opcion = input("Selecciona una opción: ").strip()

            if opcion == "1":
                nombre = input("Nombre de la banda: ")
                seguidores = int(input("Total de seguidores: "))
                db.agregar_banda(nombre, seguidores)
            elif opcion == "2":
                nombre = input("Nombre del integrante: ")
                instrumento = input("Instrumento: ")
                imagen = input("Nombre del archivo de imagen: ")
                seguidores = int(input("Número de seguidores: "))
                conciertos = int(input("Número de conciertos: "))
                ganancias = float(input("Ganancias: "))
                banda_id = int(input("ID de la banda: "))
                db.agregar_integrante(nombre, instrumento, imagen, seguidores, conciertos, ganancias, banda_id)
            elif opcion == "3":
                fecha = input("Fecha del concierto (YYYY-MM-DD): ")
                ganancias = float(input("Ganancias del concierto: "))
                db.agregar_concierto(fecha, ganancias)
            elif opcion == "4":
                integrante_id = int(input("ID del integrante: "))
                concierto_id = int(input("ID del concierto: "))
                db.registrar_asistencia(integrante_id, concierto_id)
            elif opcion == "5":
                db.obtener_integrantes()
            elif opcion == "6":
                db.obtener_conciertos()
            elif opcion == "7":
                db.cerrar_conexion()
                break
            else:
                print("❌ Ingresa una opción válida, por favor.")
```