**1. Párrafo de solución**

El proyecto CRM Evolve fue desarrollado como una aplicación de consola en Python utilizando una base de datos SQLite para la gestión de usuarios y facturas. La estructura del código se organizó en módulos separados dentro de la carpeta **src/**, manteniendo una arquitectura clara y modular. Se creó un script principal con un menú interactivo para navegar por las operaciones de usuarios y facturas, y scripts auxiliares para insertar datos de ejemplo en la base de datos. Los archivos de esquema SQL y la base de datos SQLite se guardan en la carpeta **database/**, separando los datos del código y siguiendo buenas prácticas de organización. Además, se ha añadido una carpeta docs/ donde se alojan diagramas de flujo, figuras y documentos de diseño (por ejemplo, el archivo Word de análisis).

La base de datos SQLite consta de dos tablas principales:

* **users**: almacena los datos de cada usuario, incluyendo un identificador único (user\_id), nombre, apellidos, correo, teléfono, dirección y fecha de registro automática.
* **invoices**: guarda las facturas emitidas, con un identificador (invoice\_id), referencia al usuario (user\_id), fecha de emisión automática, descripción, importe y estado (Pending, Paid, Cancelled).

La relación entre **users** e **invoices** es uno a muchos: cada usuario puede tener varias facturas, pero cada factura pertenece a un solo usuario. Gracias a esto, se garantiza que no existan facturas sin usuario asociado.

Para garantizar la calidad del código y la estabilidad del sistema, se implementaron pruebas unitarias en la carpeta **test/** usando **pytest**. Estas pruebas cubren la creación de usuarios, la generación de facturas y la validación de errores comunes (por ejemplo, claves duplicadas o datos faltantes). Además, se incluyó la generación de un diagrama ER automatizado a partir del esquema SQL para facilitar la visualización de las relaciones entre tablas.

El repositorio en GitHub (<https://github.com/Samuel-Cantarero/crm-evolve>)

cuenta con tres ramas:

* **main**: versiones estables del proyecto.
* **develop**: integración de nuevas funcionalidades y pruebas.
* **feature**: desarrollo de características específicas.

La organización de carpetas sigue buenas prácticas:

* **src/**: código fuente de la aplicación.
* **database/**: archivos de esquema SQL y base de datos SQLite.
* **test/**: pruebas automatizadas con pytest.
* **docs/**: documentación adicional (diagramas de flujo, figuras y diseño del sistema en .docx).
* **README.md**: instrucciones de instalación y uso.
* **requirements.txt**: dependencias del proyecto.

**2.Justificación de tipos de datos**

Se han tomado dos niveles de decisión: la estructuración de la base de datos en SQLite y el modelado de la lógica en Python.

**a) Tipos de datos en base de datos (SQLite)**

* **INTEGER**:
  + **user\_id** e **invoice\_id** como INTEGER PRIMARY KEY AUTOINCREMENT para asegurar identificadores únicos, generados automáticamente sin riesgo de colisión manual.
  + **phone** como entero para representar números de teléfono sin decimales y optimizar búsquedas e índices.
* **TEXT**:
  + Campos de texto (first\_name, last\_name, email, address, description, status) para almacenar cadenas de longitud variable.
  + **NOT NULL** en campos críticos (first\_name, last\_name, email, registration\_date, issue\_date, description, amount, status) garantiza datos completos.
  + **UNIQUE** en email para evitar duplicados de usuario.
  + Fechas (registration\_date, issue\_date) en formato ISO 8601 con DEFAULT(datetime('now')) aseguran consistencia y permiten ordenar y filtrar cronológicamente.
* **REAL**:
  + **amount** para manejar importes con decimales.
* **FOREIGN KEY**:
  + Relaciones entre tablas (invoices.user\_id → users.user\_id) para mantener integridad referencial y facilitar operaciones JOIN.

**b) Estructuras de datos en Python**

* **Listas de tuplas**: agrupamos los datos de usuarios y facturas en pequeños bloques ordenados. Así resulta muy sencillo recorrerlos y escribirlos en la base de datos de una sola vez.
* **Lista de correos**: guardamos los correos en una lista para saber a quién pertenece cada factura. Es una forma clara de emparejar facturas con usuarios.
* **Diccionario de estados**: con status relacionamos la opción del menú (por ejemplo, "1") con la palabra que describe el estado ("Pending", "Paid" o "Cancelled"). Evita tener que escribir muchos if y hace el código más limpio.
* **Tuplas para filas**: cada vez que obtenemos o enviamos datos a SQLite, usamos tuplas. Al ser inmutables, no hay riesgo de cambiar los datos sin querer.

**3. Tipos de datos utilizados por campo**

| **Campo** | **Tipo de dato (SQLite)** | **Descripción** |
| --- | --- | --- |
| first\_name | TEXT | Nombre del usuario |
| last\_name | TEXT | Apellidos del usuario |
| email | TEXT UNIQUE | Correo electrónico, único por usuario |
| phone | INTEGER | Teléfono de contacto (opcional) |
| address | TEXT | Dirección del usuario (opcional) |
| registration\_date | TEXT (ISO 8601) | Fecha de registro con valor por defecto automático |
| user\_id | INTEGER | Clave foránea que enlaza la factura con el usuario |
| issue\_date | TEXT (ISO 8601) | Fecha de emisión de la factura |
| description | TEXT | Descripción del servicio o producto facturado |
| amount | REAL | Importe monetario de la factura |
| status | TEXT | Estado de la factura: 'Pending', 'Paid' o 'Cancelled' |

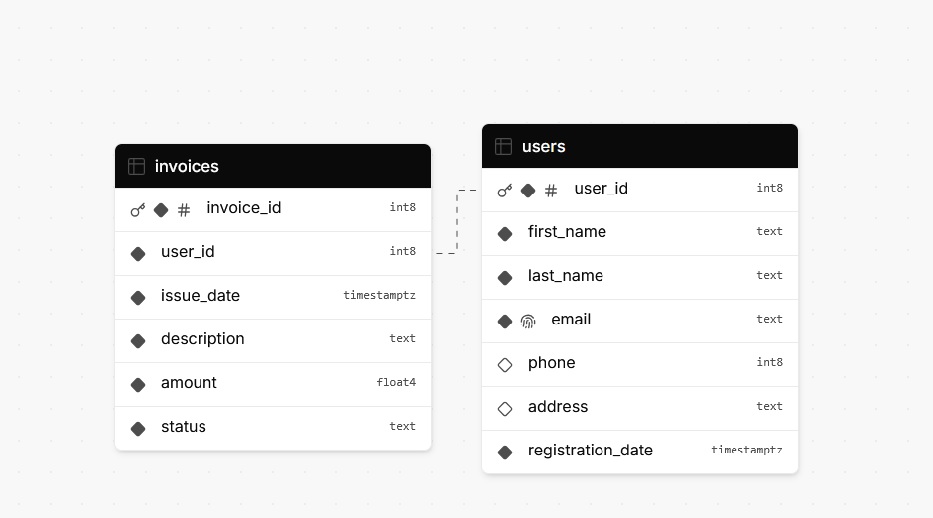


Diagrama generado por database.build

**4. Diagrama de flujo**

A continuación, se muestra un diagrama simplificado de los procesos principales de CRM Evolve, compatible con la extensión de Visual Studio Code de Markdown Preview Mermaid Support.

flowchart TD

Start["Start"] --> Menu["Show menu"]

Menu --> Choice{"Select option"}

Choice -->|1| RegUser["Register new user"]

Choice -->|2| SearchUser["Search user"]

Choice -->|3| CreateInvoice["Create invoice for user"]

Choice -->|4| ShowUsers["Show all users"]

Choice -->|5| ShowInvoices["Show invoices for a user"]

Choice -->|6| Summary["Financial summary per user"]

Choice -->|7| Exit["Exit"]

RegUser --> Menu

SearchUser --> Menu

CreateInvoice --> Menu

ShowUsers --> Menu

ShowInvoices --> Menu

Summary --> Menu

Exit --> End["End"]

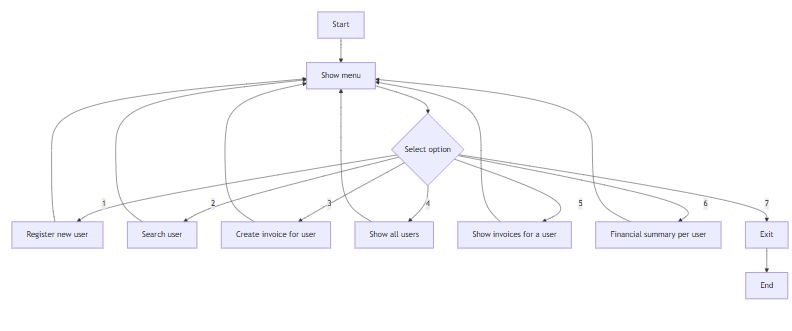


Diagrama de flujo generado Visual Studio Code (extensión Markdown Mermaid)