

## 1. Interfaces: El Poder de los Contratos

Imagina que una interfaz es un **contrato**. No dice *cómo* hacer el trabajo, solo dice *qué* trabajo se debe hacer.

### ¿Qué es una Interfaz?

- Una interfaz es un tipo de referencia que define un conjunto de métodos, propiedades o eventos que una clase

*debe* implementar si decide "firmar" ese contrato.

- No contiene implementación (código), solo las firmas de los miembros. Por convención, sus nombres empiezan con una "I" mayúscula (ej.

IImprimible, IRepositoryCliente).

- Una clase puede heredar de

**una sola clase base**, pero puede implementar **múltiples interfaces**.

### ¿Para qué sirven? El Problema que Resuelven

En la clase se presentó un problema: tenías una clase Empleado (que hereda de Persona) y una clase Moto (que hereda de Automotor). Ambas tenían un método

Imprimir(), pero no había forma de tratarlas de manera polimórfica porque no compartían una clase base común (aparte de object). Esto te obligaba a usar código "feo" y poco mantenible:

// Código que queremos evitar

```
if (o is Empleado e) {  
    e.Imprimir();  
} else if (o is Moto m) {  
    m.Imprimir();  
}
```

### La solución es la interfaz:

1. **Creas un contrato:** Se define la interfaz IImprimible con un único método void Imprimir();.
2. **Las clases "firman" el contrato:** Haces que tanto Empleado como Moto implementen la interfaz:
  - class Empleado : Persona, IImprimible { ... }

- `class Moto : Automotor, Imprimible { ... }`

3. **Obtienes polimorfismo:** Ahora, puedes tratar a cualquier objeto de esas clases como un `Imprimible` y llamar al método `Imprimir()` sin saber de qué clase concreta se trata. Esto permite un código limpio y extensible:

```
// Código limpio gracias a la interfaz
```

```
foreach (Imprimible imp in vector) {
```

```
    imp.Imprimir(); // El sistema sabe qué método Imprimir() llamar
```

```
}
```

La clave es que la interfaz te permite establecer un **comportamiento común** entre clases que no están relacionadas por herencia.

JournalMode

**JournalMode determina cómo SQLite maneja las transacciones para asegurar que los datos no se corrompan si algo falla mientras estás escribiendo en la base.**

#### ⚙️ ¿Qué es una transacción?

Cuando haces cambios en la base (INSERT, UPDATE, DELETE), SQLite **no los aplica directamente**. En su lugar:

1. Guarda un “registro” temporal (el **journal**).
2. Intenta aplicar los cambios.
3. Si todo sale bien, los guarda definitivamente.
4. Si algo falla (por ejemplo, se corta la luz), **puede revertir** usando ese journal.

**JournalMode determina cómo SQLite maneja las transacciones para asegurar que los datos no se corrompan si algo falla mientras estás escribiendo en la base.**

---

#### ⚙️ ¿Qué es una transacción?





Cuando haces cambios en la base (INSERT, UPDATE, DELETE), SQLite **no los aplica directamente**. En su lugar:

1. Guarda un “registro” temporal (el **journal**).
2. Intenta aplicar los cambios.
3. Si todo sale bien, los guarda definitivamente.
4. Si algo falla (por ejemplo, se corta la luz), **puede revertir** usando ese journal.

---



## ¿Por qué es importante el JournalMode?

Porque define **dónde y cómo se guarda ese journal**, y eso impacta en:

Tema	Ejemplo de impacto
 Seguridad	Evita que se corrompa la base si crashea
 Rendimiento	Puede hacer que las escrituras sean más rápidas o lentas
 Concurrencia	Permite (o no) leer mientras se escribe
 Comportamiento	Cómo maneja el rollback ante errores

---

## Modos más comunes (resumen rápido):

Modo	Qué hace	Ideal para...
DELETE	Crea un archivo .journal y lo borra al terminar	Seguro, tradicional, pero lento
WAL	Usa un archivo .wal separado y permite más concurrencia	 Mejor rendimiento y lectura/escritura simultáneas
MEMORY	Guarda el journal en RAM (rápido, pero sin protección real)	Pruebas o datos no críticos
OFF	Sin journal. Si algo falla, perdés datos	Solo lectura o entornos controlados 

---

## ¿Qué deberías usar?

✓ Para una **app real con EF Core y SQLite**, lo más recomendado hoy es:

sql

CopiarEditar

```
PRAGMA journal_mode=WAL;
```

Porque te da:

- Mejor rendimiento

- Lectura sin bloqueos durante escrituras
- Seguridad ante fallos