PD4 · CEPY · UT3

Excepciones y repositorio

en SQLite

PD4 - CEPY

Índice

1. El gestor de contexto `with conn:`

Commit y rollback automáticos

`with conn:` — transacciones sin esfuerzo

- `with conn:` abre una **transacción**
- Sin errores → `commit()` automático
- Con excepción → `rollback()` automático
- **No cierra la conexión** — usa `finally`

```python
conn = sqlite3.connect("expendedora.db")
try:
    with conn:
        cursor = conn.cursor()
        cursor.execute(
            "INSERT INTO productos "
            "VALUES (?, ?, ?, ?)",
            ("B2", "Zumo", 1.50, 20)
        )
except sqlite3.IntegrityError as e:
    print(f"Error: {e}")
finally:
    conn.close()
```

⚠️ `with conn:` gestiona la **transacción**, no la conexión. Cierra siempre con `conn.close()` en `finally`.

> `sqlite3.IntegrityError` es uno de los tipos de excepción de SQLite — los veremos en la sección siguiente.

2. Jerarquía de excepciones

Qué puede lanzar SQLite y cuándo

Jerarquía de excepciones

```
sqlite3.Error
└── sqlite3.DatabaseError
    ├── sqlite3.IntegrityError    ← PK duplicada, NOT NULL, FK sin referencia
    ├── sqlite3.OperationalError  ← tabla inexistente, SQL incorrecto, BD bloqueada
    ├── sqlite3.ProgrammingError  ← nº de ? incorrecto, conexión cerrada
    ├── sqlite3.DataError         ← datos fuera de rango
    └── sqlite3.NotSupportedError ← operación no soportada
```

💡 Las dos más frecuentes son **`IntegrityError`** (restricciones de datos) y **`OperationalError`** (problemas del motor). Captura siempre la más concreta primero.

3. Excepciones frecuentes

`IntegrityError`, `OperationalError` y `ProgrammingError`

`IntegrityError` — violación de restricciones

Causa	Ejemplo
Clave primaria duplicada	Dos productos con el mismo `codigo`
Campo `NOT NULL` con `None`	`None` en una columna obligatoria
Clave foránea sin referencia	`codigo` en `descuentos` inexistente en `productos`

```
try:
    with conn:
        cursor = conn.cursor()
        cursor.execute("INSERT INTO productos VALUES ('A1', 'Agua', 1.00, 10)")
        cursor.execute("INSERT INTO productos VALUES ('A1', 'Duplicado', 0.50, 5)")
except sqlite3.IntegrityError as e:
    print(f"Error de integridad: {e}")  ## rollback automático
finally:
    conn.close()
```

```
Error de integridad: UNIQUE constraint failed: productos.codigo
```

`OperationalError` — errores del motor

Causa	Ejemplo
Tabla inexistente	`SELECT * FROM ventas`
SQL con sintaxis incorrecta	`SELEC * FROM productos`
Tabla ya existente sin `IF NOT EXISTS`	`CREATE TABLE productos ( ... )`

```
try:
    with conn:
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM ventas")
except sqlite3.OperationalError as e:
    print(f"Error operacional: {e}")
finally:
    conn.close()
```

```
Error operacional: no such table: ventas
```

`ProgrammingError`

Se lanza cuando el número de `?` no coincide con los valores, o se usa la conexión tras cerrarla.

```
try:
    with conn:
        cursor = conn.cursor()
        cursor.execute(
            "INSERT INTO productos VALUES (?, ?, ?)",  ## 3 ? pero 4 valores
            ("A1", "Agua", 1.00, 10)
        )
except sqlite3.ProgrammingError as e:
    print(f"Error de programación: {e}")
finally:
    conn.close()
```

```
Error de programación: binding 4 has no name
```

Resumen de excepciones

Excepción	Cuándo se lanza	Qué hacer
`sqlite3.IntegrityError`	PK duplicada, NOT NULL, FK sin referencia	Informar al usuario — `with` hizo el rollback
`sqlite3.OperationalError`	Tabla inexistente, SQL incorrecto	Revisar la consulta o el esquema
`sqlite3.ProgrammingError`	Nº de `?` incorrecto, conexión cerrada	Corregir el código

💡 Con `with conn:` el `rollback()` es automático — solo necesitas capturar la excepción e informar al usuario.

4. Excepciones propias de dominio

Por qué el menú no debería conocer `sqlite3`

El problema: capas que no deberían mezclarse

🚫 Sin excepciones propias

```
## En el menú — conoce sqlite3
try:
    repo.guardar(item)
except sqlite3.IntegrityError:
    print("Producto duplicado")
```

El menú depende de la infraestructura. Si cambias SQLite, el menú se rompe.

✅ Con excepciones propias

```
## En el menú — solo conoce el dominio
try:
    repo.guardar(item)
except ProductoYaExisteError as e:
    print(f"Error: {e}")
```

El menú no sabe nada de SQLite. Funciona igual con cualquier repositorio.

Definir excepciones propias

Las excepciones propias son clases que heredan de `Exception`:

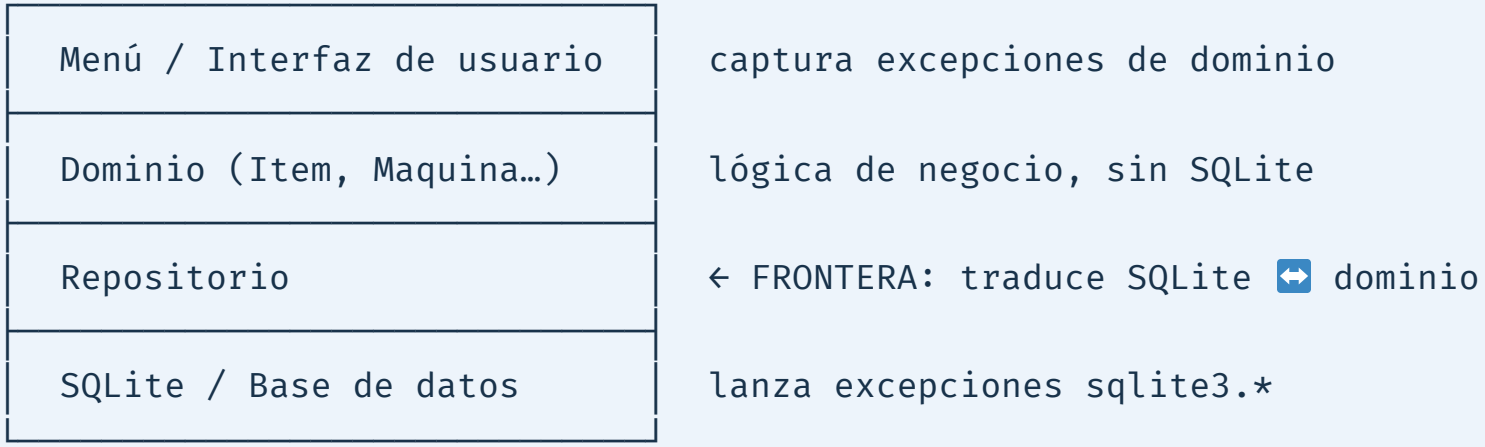
```
class ErrorRepositorio(Exception):
    """Excepción base para errores de persistencia."""
    pass

class ProductoYaExisteError(ErrorRepositorio):
    """Código ya existente en la base de datos."""
    pass

class ProductoNoEncontradoError(ErrorRepositorio):
    """Producto no encontrado en la base de datos."""
    pass

class ErrorPersistencia(ErrorRepositorio):
    """Cualquier otro error del motor de base de datos."""
    pass
```

💡 La misma lógica de jerarquía que `sqlite3`: captura `ErrorRepositorio` para atrapar cualquier error

5. El repositorio como frontera

Traducir entre SQLite y el dominio

Las capas de la aplicación



💡 El repositorio es la única capa que importa `sqlite3`. Todo lo que está por encima solo ve objetos de dominio y excepciones de dominio.

El repositorio transforma las excepciones

```
def guardar(self, item: Item) → None:
    conn = sqlite3.connect(self._ruta_bd)
    try:
        with conn:
            cursor = conn.cursor()
            cursor.execute("PRAGMA foreign_keys = ON")
            cursor.execute(
                "INSERT INTO productos VALUES (?, ?, ?, ?)",
                (item.codigo, item.nombre, item.precio, item.cantidad)
            )
    except sqlite3.IntegrityError:
        raise ProductoYaExisteError(
            f"Ya existe un producto con código '{item.codigo}'"
        )
    except sqlite3.OperationalError as e:
        raise ErrorPersistencia(f"Error al guardar: {e}")
    finally:
        conn.close()
```

Cómo queda el menú

El menú nunca importa `sqlite3`. Solo captura excepciones de dominio:

```
def opcion_guardar(repo, item):
    try:
        repo.guardar(item)
        print(f"Producto '{item.codigo}' guardado correctamente.")
    except ProductoYaExisteError as e:
        print(f"Error: {e}")
    except ErrorPersistencia as e:
        print(f"Error inesperado de base de datos: {e}")
```

✅ Si en el futuro cambias SQLite por PostgreSQL o por un JSON, solo reescribes el repositorio. El menú, el dominio y los tests de dominio no cambian.

¿Preguntas?

PD4 · CEPY · UT3 — Excepciones y repositorio en SQLite