Índice

Ejercicios Unidad 24 - Bases de Datos Relacionales

> **Antes de empezar**
>
> Para realizar estos ejercicios, deberás descargar los recursos del enlace de proyecto_poo. Como puedes ver, la solución está compuesta de varios proyectos. Cada uno de ellos corresponde con un ejercicio, deberás implementar todo el código, tanto de la Main como de los métodos que se piden en cada ejercicio. Cada proyecto contiene el test correspondiente, que deberás pasar para comprobar que has hecho el ejercicio correctamente.

Ejercicio 1. Gestión de Plantas (DDL y DML)

Vamos a crear una pequeña gestión de base de datos para un vivero. En este primer ejercicio nos centraremos en crear la estructura de la base de datos y poblarla con datos iniciales utilizando **ADO.NET** y **SQLite**.

```
Ejercicio 1: Gestión de Plantas

Insertando datos de ejemplo...

Insertando -> Rosa, Rosa spp., 10.50, 50
Insertando -> Lirio, Lilium, 7.80, 80
Insertando -> Tulipán, Tulipa, 5.25, 100
Insertando -> Orquídea, Orchidaceae, 25.00, 20
Datos insertados correctamente.
```

Requisitos

1. **Crea** un nuevo proyecto de consola y añade el paquete NuGet `Microsoft.Data.Sqlite`.
2. **Crea** una conexión a una base de datos llamada `vivero.db` en el `Main`.
3. **Implementa un método** `CreaTabla(SqliteConnection conexion)` que reciba la conexión abierta y cree la tabla `planta` (si no existe) con las columnas:
   - `id`: Entero, Clave Primaria y Autoincremental.
   - `nombre_comun`: Texto, no nulo.
   - `nombre_cientifico`: Texto, puede ser nulo o vacío.
   - `precio`: Real.
   - `stock`: Entero.
4. **Implementa un método** para insertar datos (ej. `InsertaPlanta(...)`) que reciba la conexión y los datos necesarios.
5. **Inserta** al menos **3 plantas** diferentes llamando a este método.
   - **Importante**: Debes utilizar **consultas parametrizadas** (usando `@parametro`) para evitar la inyección SQL.

Pistas

- Usa `using SqliteConnection ...` para gestionar la conexión correctamente.
- Usa sentencias SQL `CREATE TABLE IF NOT EXISTS...` y `INSERT INTO...`.
- Recuerda usar `comando.ExecuteNonQuery()` para las operaciones que no devuelven datos.

> **"Explora tu Base de Datos"**
>
> Una vez ejecutado el programa, se creará el archivo `vivero.db`. ¡No lo abras con un editor de texto!
> Te recomendamos utilizar una extensión de VS Code como **SQLite Viewer** (tal y como se indica en los apuntes) para abrir el archivo `.db` y comprobar visualmente que la tabla y los registros se han creado correctamente.

Ejercicio 2. Clase Vivero, Consultas y Transacciones

Vamos a profesionalizar nuestro código separando la lógica de acceso a datos en una clase propia y añadiendo la capacidad de consultar información.

```
Ejercicio 2: Clase Vivero, Consultas y Transacciones

Planta { Id = 1, NombreComun = Margarita, NombreCientifico = Argyranthemum, Precio = 3,
Planta { Id = 3, NombreComun = Lavanda, NombreCientifico = Lavandula, Precio = 6,75, St
Planta { Id = 4, NombreComun = Cactus, NombreCientifico = Cactaceae, Precio = 8, Stock
Planta { Id = 5, NombreComun = Bonsai, NombreCientifico = Ficus retusa, Precio = 45, St

Operaciones completadas.
```

Requisitos

1. **Crea un Modelo**: Define una clase o record `Planta` con las propiedades que se corresponden con la tabla (`Id`, `NombreComun`, `NombreCientifico`, `Precio`, `Stock`).
2. **Crea la clase `Vivero`**: Esta clase encapsulará la lógica de base de datos y la información del comercio.
   - Añade las propiedades `Nombre` (string), `Direccion` (string) y una lista de plantas (`List<Planta> Plantas`).
   - Debe tener un método `CreaTabla()`.
   - Debe tener un método `InsertaPlanta(Planta p)`.
   - Implementa un método `ObtenPlantas()` que realice un `SELECT * FROM planta`, lea los datos con `SqliteDataReader`, convierta cada fila en un objeto `Planta` y devuelva una

`List<Planta>`.
    - Implementa el método `ModificaStock(int id, int nuevoStock)` que actualice el campo stock (`UPDATE`).
    - Implementa el método `EliminaPlanta(int id)` que borre el registro correspondiente (`DELETE`).
3. **Programa Principal**:
    - Crea una `List<Planta>` en memoria y rellénala con 5 plantas nuevas.
    - Recorre la estructura `Vivero` para insertarlas en la base de datos (¡Usa transacciones!).
    - Llama a `ModificaStock` para cambiar el stock de una de las plantas.
    - Llama a `EliminaPlanta` para borrar otra.
    - Finalmente, llama a `ObtenPlantas()` y muestra el listado completo para verificar los cambios.

> **Transacciones (Bonus Recomendado)**
>
> Cuando realizamos múltiples operaciones de escritura seguidas (como insertar una lista de plantas), es una muy buena práctica utilizar **Transacciones**.
>
> - Una transacción asegura que **o se guardan todas las plantas o no se guarda ninguna** (ideal si hay errores a mitad).
> - Además, mejora drásticamente el rendimiento.
> - **Intenta implementarlo**: Abre la transacción antes de cualquier modificación (`conexion.BeginTransaction()`) y haz el `Commit()` al acabar.

Ejercicio 3. Patrón DAO (Data Access Object)

Vamos a refactorizar el ejercicio anterior para seguir estrictamente el patrón **DAO** (Data Access Object) tal como se explica en los apuntes de la unidad. Esto nos permitirá separar completamente la lógica de acceso a datos de la lógica de negocio. ***Recuerda crear cada entidad en ficheros distintos, así como los DAO.***

```
Ejercicio 3: Patrón DAO (Data Access Object)

Vivero: Vivero El Jardín
Dirección: Calle de las Flores 123
Inventario Valorado:
Petunia: 150 unidades - Valor Total: 375,00 ?
Geranio: 80 unidades - Valor Total: 240,00 ?

Operaciones completadas.
```

Requisitos

1. **Clases Entidad**: Separa los datos de la lógica.
   - `Planta`: Propiedades (`Id`, `NombreComun`, `NombreCientifico`, `Precio`, `Stock`).
   - `Vivero`: Propiedades (`Nombre`, `Direccion`) y una lista de plantas (`List<Planta>`).
2. **Clase DAO**: Crea una clase `PlantaDAO` que implemente `IDisposable` para gestionar la conexión automáticamente.
   - Debe recibir la cadena de conexión en el constructor y abrir la conexión.
   - Debe implementar `Dispose()` para cerrar la conexión.
3. **Operaciones CRUD**: Implementa los siguientes métodos en `PlantaDAO`:
   - `Create(Planta p)`: Inserta una nueva planta.
   - `Read()`: Devuelve `IEnumerable<Planta>` con todas las plantas (usa `yield return`).
   - `Read(int id)`: Devuelve `Planta?` buscando por ID.
   - `Update(Planta p)`: Actualiza una planta existente. Comprueba antes si existe con `Read(id)`.
   - `Delete(int id)`: Borra una planta por ID. Comprueba antes si existe.
   - `Count()`: Devuelve el número total de plantas (usa `ExecuteScalar`).
4. **Consulta con DTO (Valor del Stock)**:
   - Crea un record `PlantaStock` con las propiedades: `Nombre_Comun`, `Stock` y `ValorTotal` (este último será el cálculo de precio * stock).
   - Implementa en el DAO un método `ObteneInventario()` que devuelva `IEnumerable<PlantaStock>`.
   - La consulta SQL debe proyectar los campos necesarios y calcular el valor total (`SELECT ..., precio * stock as valor ...`).
5. **Programa Principal**:
   - **Carga JSON**: Lee el contenido del fichero `vivero.json` y deserialízalo en un objeto de la clase `Vivero` (que contendrá el nombre, dirección y las plantas iniciales).
   - **Instancia el DAO**: Utiliza `using PlantaDAO dao = new(...)`.
   - **Carga BD**: Si la base de datos está vacía (`dao.Count() == 0`), recorre las plantas del objeto `Vivero` y guárdalas una a una usando `dao.Create()`.

- **Manipulación**: Realiza pruebas de modificación (`Update` / `Delete`) utilizando el DAO.
- **Informe Final**: Genera un informe por consola que muestre cabecera con el **Nombre y Dirección del Vivero** (del objeto Vivero) y luego el listado de valoración del inventario (obtenido del método `ObteneInventario` del DAO).

> ⚠️ **Mantén el método de crear la tabla fuera del DAO o como un método utilitario separado, ya que el DDL no suele formar parte del DAO estándar.**