**Ejercicio #1 – Determinar la ubicación para una base de datos**

1. \*La BD se guardará en carpeta files en Android, y en carpeta Library en iOS\*.
2. Crear carpeta nueva **Employees**; dentro de carpeta crear pantalla **EmployeesListPage** (con título) e interfaz **IFileAccessHelper**.
3. En **FileAccessHelper** definir un método para obtener una ruta, a partir del nombre de un archivo.
4. En proyectos de Android y iOS crear clases **LocalFileAccess**, para implementar la interfaz.
   1. En Android, usar carpeta **files**; en iOS, carpeta **Library**.
   2. En ambas clases, incluir llamada a clase **Dependency**, con **assembly**.
5. En **EmployeesListPage**:
   1. En XAML: Crear un label con nombre.
   2. En codebehind: Mostrar en el texto del label la implementación del método de la interfaz **IFileAccessHelper**, enviando el nombre de una BD como parámetro.
6. Agregar pantalla a master.
7. Correr la app (se muestra la ruta donde se guardará la BD).
8. Continuar con la presentación.

**Ejercicio #2 – Trabajar con una base de datos SQLite**

1. Instalar el paquete **Nuget** **sqlite-net-pcl** en los tres proyectos de la solución.
2. Crear clase **Employee** para mapear a una tabla en la base de datos; con las propiedades **Id**, **Name** y **Position**.
3. Agregar anotaciones de **SQLite** en la clase **Employee**:
   1. Agregar using **SQLite**.
   2. Asignar el atributo **Table** a la clase, con el nombre **Employee**.
   3. Asignar **PrimaryKey** y **AutoIncrement** a **Id**.
   4. Asignar **MaxLength** a **Name** y **Position**.
4. \*Primero se realizará la operación **SELECT** para traer todos los registros\*
5. Crear clase **EmployeeRepository**:
   1. Agregar **using** **SQLite** y **Xamarin.Forms.**
   2. Crear variables globales **connection** y **dbPath**.
   3. En el constructor, inicializar **dbPath** y **connection**; crear tabla **Employee** en la bd.
   4. Crear método **GetAll**:
      1. Con LINQ, traer los registros ordenados de **Employee** (con método **Table<>**).
      2. Convertir el query a una lista.
6. En **EmployeesListPage**:
   1. En XAML:
      1. Comentar el label con la ubicación de la bd.
      2. Crear control **ListView**, con nombre y atributo **ItemsSource** igual a **Binding**.
         1. Mostrar los atributos **Name** y **Position** de cada objeto en un **TextCell**.
   2. En codebehind:
      1. Crear variable global **repository** (de tipo **EmployeeRepository**).
      2. En constructor:
         1. Instanciar **repository**.
         2. Hacer que el **BindingContext** de la pantalla sea el método **GetAll** de **repository**.
7. \*Ahora se realizará la operación **INSERT** para agregar un registro\*
8. En **EmployeeRepository**, crear método **AddNewEmployee**:
   * 1. Devuelve una cadena y acepta dos: **name** y **position**.
     2. Llamar al método **Insert** de la conexión y enviar un nuevo objeto **Employee**, con los valores de los parámetos; devolver el resultado de la operación en una cadena.
9. Crear pantalla **AddEmployeePage**:
   1. En XAML:
      1. \*Copiar el contenido de **AddBookPage**\*, personalizando la cantidad e información de los controles.
      2. Incluir un label para mostrar el resultado del **Insert**.
   2. En codebehind:
      1. Crear variable global **repository** (de tipo **EmployeeRepository**).
      2. Agregar un parámetro tipo **EmployeeRepository** al constructor; dentro de constructor, inicializar **repository** y crear **handler** de evento **Clicked** del botón.
      3. \*En el handler, copiar el contenido de handler de **AddBookPage**\*.
         1. Personalizar código para usar método **AddNewEmployee** de **repository**; mostrar resultado en el label.
10. Modificar pantalla **EmployeesListPage**:que muestre un **ToolbarItem** para abrir pantalla **AddNewEmployee** (como modal); incluir imágenes de **Recursos**.
11. Correr la app (de entrada la bd está vacía, se agregan registros, pero no se muestran en automático).
12. En codebehind de **EmployeesListPage**:
    1. Sobre escribir método **OnAppearing**, donde ahora se asigna el valor de **BindingContext**.
13. Correr la app (cuando se muestra la pantalla, se actualiza la lista).
14. \*Incluir **SearchBar** para hacer búsquedas en la lista\*
15. En **EmployeesListPage**:
    1. En XAML: Incluir control **SearchBar**, con nombre.
    2. En codebehind:
       1. En constructor, generar handler para evento **TextChanged** de **SearchBar**.
       2. En el handler:
          1. Si la cadena del control está vacía, asignar **BindingContext** a **GetAll** de **repository**.
          2. Si no, asignar **BindingContext** al mismo método, pero trayendo los registros que empiecen con la cadena del control (método **StartsWith**).
16. \*Ahora se realizará la operación **UPDATE** para actualizar un registro (**ellos**)\*
17. En **EmployeeRepository**:
    1. Crear método **GetEmployee**: devuelve un objeto **Employee** que coincida con el **id** que recibe como parámetro.
    2. Crear método **UpdateEmployee**: recibe un objeto **Employee** como parámetro, obtiene un objeto con el método **GetEmployee**, enviando el id del parámetro; modifica el objeto y lo envía como parámetro de **Update** de la conexión.
18. Crear pantalla **EditEmployeePage**:
    1. En XAML:
       1. \*Copiar el contenido de **AddEmployeePage**\*.
          1. Quitar el label y personalizar botón.
          2. Agregar los atributos de **Binding** correspondientes a los controles **EntryCell**.
    2. En codebehind:
       1. Crear variable global **repository** (de tipo **EmployeeRepository**).
       2. Agregar dos parámetros al constructor: un **EmployeeRepository** y un **Employee**.
       3. Dentro de constructor, inicializar **repository**, asignar parámetro **Employee** como **BindingContext** y crear **handler** de evento **Clicked** del botón.
       4. \*En el handler, copiar el contenido de handler de **AddEmployeePage**\*.
          1. Llamar a método **UpdateEmployee** de **repository**, enviando a **BindingContext** como parámetro; cerrar pantalla.
19. En **EmployeesListPage**:
    1. Utilizar **ContextActions** para mostrar botón en **Toolbar**, que llevará a pantalla **EditEmployeePage** (con parámetros correspondientes).
20. Correr la app (se actualiza un registro).
21. \*Ahora se realizará la operación **DELETE** para eliminar un registro (**ellos**)\*
22. En **EmployeeRepository**: Crear método **DeleteEmployee** que acepta un objeto **Employee** como parámetro; obtiene un objeto con el método **GetEmployee** y lo envía como parámetro de **Delete** de la conexión.
23. En **EmployeesListPage**:
    1. Utilizar **ContextActions** para mostrar botón en **Toolbar**.
    2. Al dar clic en botón, se corrobora que el usuario quiera eliminar el registro; llamar a **DeleteEmployee** de **repository** y reasignar **BindingContext.**
24. Correr la app (se elimina un registro).

**Ejercicio #3 – Trabajar con una base de datos SQLite de manera asíncrona**

1. Modificar las operaciones en **EmployeeRepository,** para que ahora sean asíncronas.
2. Hacer las modificaciones necesarias (await, async) en todas las pantallas.
3. Correr la app (mismo funcionamiento, de manera asíncrona).