**BASE DE DATOS CON SQLITE.NET**

Las bases de datos son ampliamente utilizadas por las aplicaciones móviles, ya sea para almacenar alguna información generada y/o requerida por la aplicación, o porque la aplicación en sí, se basa en una base de datos.

Una de las librerías más utilizadas para bases de datos locales en Xamarin es la de SQLite.Net, ya que permite almacenar y recuperar objetos de la base de datos sin escribir sentencias SQL, esto gracias a la API ORM (Object Relational Mapping)

1. En este ejemplo vamos a crear una base de datos sencilla y realizaremos las operaciones básicas sobre ella (crear, leer, actualizar y borrar ).
2. Crear un nuevo proyecto del tipo Aplicación de Android (Xamarin)
3. Llamarlo UsoSQLite
4. Aplicación vacía
5. Minimun Android Version (4.4)
6. Por medio del administrador de paquetes NuGet, agregar las librerías sqlite-net-pcl y SQLiteNetExtensions (Para crear relaciones de unos a muchos, uno a uno y demas).
7. Crear nuestro modelo de la base de datos, que en este caso contendrá tres tablas. Gracias a ORM, podemos crear clases que posteriormente serán mapeadas a tablas. Para ello, lo primero es agregar una nueva clase llamada Libro, en seguida creamos un nuevo nombre de espacio dentro del ya existente (UsoSQLite), para ello enseguida de namespace UsoSQLite agregamos .Modelo, esto en el archivo recién creado, después agregamos los nombres de espacio SQLite y SQLiteNetExtensions.Attributes al archivo recién creado. Ahora bien, dentro de la clase agregamos el siguiente código que es el esquema de la tabla Libro:

...

[AutoIncrement, PrimaryKey, Column("id")]

public int id { get; set; }

[MaxLength(100), NotNull]

public string titulo { get; set; }

[ForeignKey(typeof(Editorial))]

public int EditorialId { get; set; }

[OneToOne("EditorialId")]

public Editorial editorial { get; set; }

[MaxLength(70)]

public string ciudad { get; set; }

[MaxLength(70)]

public string pais { get; set; }

[Ignore]

public double valor\_extra { get; set; }

[OneToMany]

public List<Autor> autores { get; set; }

...

1. Agregar la clase Autor al proyecto
2. Agregar los nombres de espacio SQLite y SQLiteNetExtensions.Attributes al archivo recién creado
3. Agregar el nombre de espacio Modelo al nombre de espacio UsoSQLite
4. Agregar el siguiente código dentro de la clase Autor

...

[PrimaryKey, AutoIncrement]

public int id { get; set; }

[NotNull]

public string nombre1 { get; set; }

public string nombre2 { get; set; }

public string nombre3 { get; set; }

[NotNull]

public string apellidoPaterno { get; set; }

public string apellidoMaterno { get; set; }

[ForeignKey(typeof(Libro))]

public int LibroId { get; set; }

...

1. Agregar la clase Editorial
2. Agregar el nombre de espacio SQLite al archivo recién creado
3. Agregar el nombre de espacio Modelo al nombre de espacio UsoSQLite
4. Agregar el siguiente código dentro de la clase Editorial:

...

[PrimaryKey,AutoIncrement]

public int id { get; set; }

[NotNull]

public string nombreEditorial { get; set; }

[NotNull,MaxLength(50)]

public string pais { get; set; }

...

1. Abrir el archivo MainActivity.cs y agregarle los nombres de espacio SQLite, System.IO, UsoSQLite.Modelo, System.Collections.Generic y SQLiteNetExtensions.Extensions.
2. En el método OnCreate() de la clase MainActivity crear la base de datos, crear las tablas basadas en el modelo diseñado y agregar registros a las tablas si es que no contienen. Todo esto agregando el siguiente código después del método SetContentView():

using (var connection = new SQLiteConnection(pathBaseDeDatos))

{

connection.CreateTable<Libro>();

connection.CreateTable<Autor>();

connection.CreateTable<Editorial>();

if (connection.Table<Autor>().Count() == 0)

{

List<Autor> autores = new List<Autor> {

new Autor {

nombre1 = "Luis",

nombre2 = "Moises",

nombre3 = null,

apellidoPaterno = "Burgara",

apellidoMaterno = "Lopez" },

new Autor { nombre1 = "Luis",

nombre2 = null,

nombre3 = null,

apellidoPaterno = "Leithold",

apellidoMaterno = null }

};

connection.InsertAll(autores);

}

if (connection.Table<Editorial>().Count()==0)

{

List<Editorial> editoriales = new List<Editorial> {

new Editorial {

nombreEditorial = "Springer",

pais = "USA" },

new Editorial {

nombreEditorial = "Limusa",

pais = "Mexico" },

};

connection.InsertAll(editoriales);

}

var editorialess = connection.Table<Editorial>().Where(r => r.nombreEditorial == "Springer").ToList()[0];

var autoress = connection.Table<Autor>().Where(r => r.apellidoPaterno == "Burgara").ToList()[0];

if (connection.Table<Libro>().Count() == 0)

{

List<Libro> libros = new List<Libro> {

new Libro {

titulo = "Calculo",

EditorialId = connection.Table<Editorial>().Where(r=>r.nombreEditorial=="Springer").ToList()[0].id,

editorial = connection.Table<Editorial>().Where(r=>r.nombreEditorial=="Springer").ToList()[0],

ciudad = "New York",

pais = "USA",

autores = new List<Autor>{connection.Table<Autor>().Where(r=>r.apellidoPaterno=="Leithold").ToList()[0]}

},

new Libro {

titulo = "Algebra",

EditorialId = connection.Table<Editorial>().Where(r=>r.nombreEditorial=="Limusa").ToList()[0].id,

editorial = connection.Table<Editorial>().Where(r=>r.nombreEditorial=="Limusa").ToList()[0],

ciudad = "Ciudad de Mexico",

pais = "Mexico",

autores = new List<Autor>{connection.Table<Autor>().Where(r=>r.apellidoPaterno=="Burgara").ToList()[0]}

}

};

connection.InsertAll(libros);

libros[0].autores = new List<Autor> { connection.Table<Autor>().Where(r => r.apellidoPaterno == "Leithold").ToList()[0] };

connection.UpdateWithChildren(libros[0]);

libros[1].autores = new List<Autor> { connection.Table<Autor>().Where(r => r.apellidoPaterno == "Burgara").ToList()[0] };

connection.UpdateWithChildren(libros[1]);

}

List<Libro> librosTodos = connection.GetAllWithChildren<Libro>();

var editoriale = connection.Table<Editorial>().Where(r => r.nombreEditorial == "Springer").ToList()[0];

librosTodos[0].editorial = editoriale;

connection.Update(librosTodos[0]);

librosTodos = connection.Table<Libro>().ToList();

connection.Close();

connection.Dispose();

}

...

1. Guardar todo, compilar e implementar en emulador o dispositivo.

**Actividad complementaria**

Nota: Esta actividad es la doceava que impacta en la calificación para el criterio de evaluación “Prácticas varias”.

1. Realizar una aplicación que busque imágenes jpeg en todo los almacenamientos interno y externos del dispositivo y que muestre las ubicaciones en un EditText.