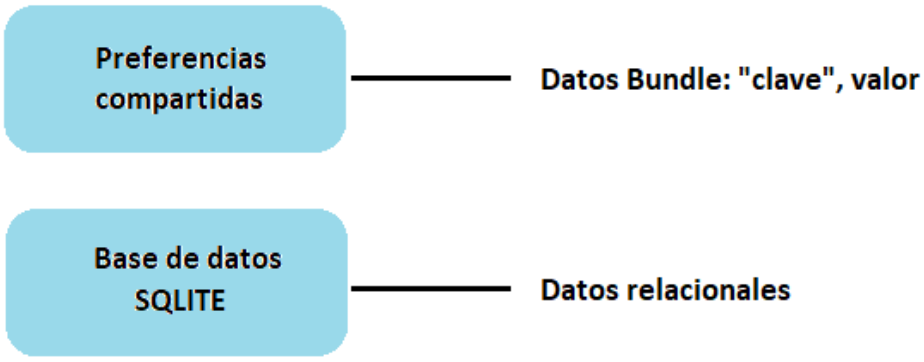
PERSISTENCIA DE DATOS

1. INTRODUCCION

- En Android existen tres formas de almacenar información de forma permanente:
  - **Preferencias compartidas (**clase ***SharedPreferences*****).**
  - **Sistemas tradicionales de archivos**.
  - **Sistema de base de datos relacional (base de datos SQLite**)



2. PREFERENCIAS COMPARTIDAS

- Código para crear una preferencia compartida (datos que queremos almacenar)

```
SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(MainActivity.this);
SharedPreferences.Editor editor = prefs.edit();
editor.putString("nom", "Obdulia");
editor.putString("ape", "Vázquez");
editor.apply();
```

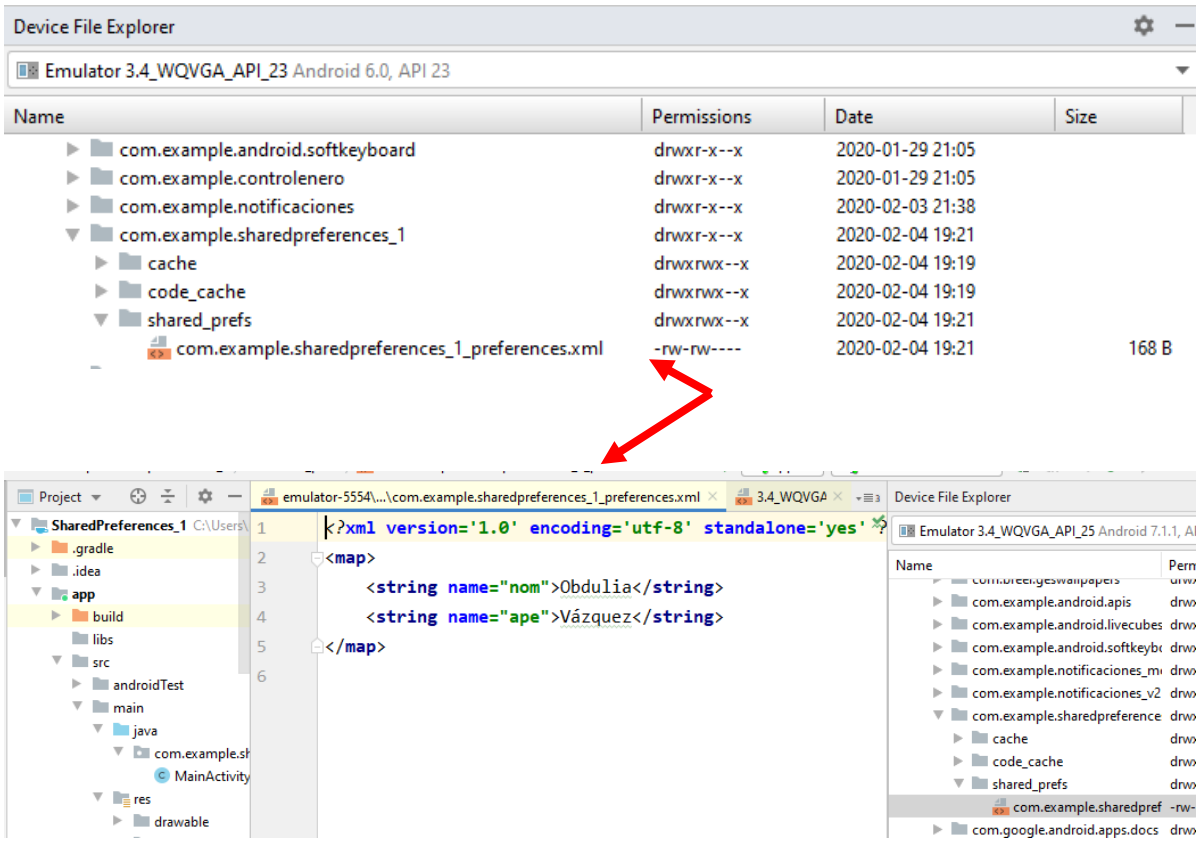
- Código para recuperar los datos almacenados previamente en una preferencia compartida

```
SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(MainActivity.this);
String nombre=prefs.getString("nom", "NO hay nombre guardado");//valor defectivo
String apellido=prefs.getString("ape", "No hay apellido guardado");//valor defectivo
Log.i("preferencias", nombre); //o cq otra forma de comprobar el valor almacenado
Log.i("preferencias", apellido);
```

- Las preferencias compartidas NO se almacenan en archivos binarios (como por ejemplo las bases de datos SQLite), sino en archivos XML. Estos ficheros XML se guardan en una ruta que sigue el siguiente patrón:

  /data/data/nombre_del_paquete/shared_prefs/nombre_preferencias.xml

  Podemos comprobarlo accediendo al explorador de ficheros con la herramienta ***Device File Explorer*** (antes, ***Android Device Monitor)***, como se muestra en la captura siguiente:



3. BASE DE DATOS SQLITE

3.1 CREADA DESDE CODIGO

- Una base de datos SQLite que se crea por medio de programación siempre se almacena en la memoria del teléfono, en la carpeta ***/data/data/nombre_del_paquete/databases/nombre_bbdd***

- Definimos una clase auxiliar que derive de ***SQLiteOpenHelper***, y la personalizamos para adaptarnos a las características de nuestra app.

- Esta clase:
  - **Crea** la base de datos si no existe.
  - **Abre** la base de datos si existe.

- **Actualiza** la versión si decidimos crear una nueva estructura de la base de datos.

Public constructors
`SQLiteOpenHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version)` Create a helper object to create, open, and/or manage a database.

Public methods	
`void`	`close()` Close any open database object.
`SQLiteDatabase`	`getReadableDatabase()` Create and/or open a database.
`SQLiteDatabase`	`getWritableDatabase()` Create and/or open a database that will be used for reading and writing.
`abstract void`	`onCreate(SQLiteDatabase db)` Called when the database is created for the first time.
`abstract void`	`onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)` Called when the database needs to be upgraded.

- Contiene métodos: ***getReadableDatabase()*** y ***getWritableDatabase()*** que abren la base de datos en modo sólo lectura o lectura/escritura.
- Debemos sobreescribir sus métodos abstractos ***onCreate()*** y ***onUpgrade()***

- Podemos ayudarnos del generador de código.
- Ejemplo de código:

```
public class BDSimple_SQLiteOpenHelper extends SQLiteOpenHelper {

    private String sentenciaCreate = "CREATE TABLE Usuarios (codigo INTEGER,nombre TEXT)";
    //constructor
    public BDSimple_SQLiteOpenHelper(@Nullable Context context,
                                     @Nullable String name,
                                     @Nullable SQLiteDatabase.CursorFactory factory,
                                     int version) {
        super(context, name, factory, version);
    }

    @Override
    public void onCreate(SQLiteDatabase sqLiteDatabase) {
        //operaciones de creacion de la bd
        sqLiteDatabase.execSQL(sentenciaCreate);
    }

    @Override
    public void onUpgrade(SQLiteDatabase sqLiteDatabase, int i, int i1) {
        //todo: operaciones de actualización de la bd
    }
}
```

- Para manejar la base de datos desde la actividad principal hay que:
  - Instanciar la clase que hemos creado (heredada de SQLiteOpenHelper)
  - Invocar el método de apertura correspondiente: ***getReadableDatabase()*** o ***getWritableDatabase()***

- Código necesario

```
BDSimple_SQLiteOpenHelper alumnosBD=new BDSimple_SQLiteOpenHelper(this, "DBAlumnos",null,1);
SQLiteDatabase sqLiteDatabase=alumnosBD.getWritableDatabase();
```

- Operaciones para gestionar el contenido de una base de datos
  - **Inyectando código SQL** directamente como parámetro del método ***execSQL()*** para operaciones que NO devuelven datos, o bien ***rawQuery()***, para consultas
  - **Mediante métodos específicos** (formato parametrizado, más seguro): ***insert()***, ***delete()***, **update()**, **query()**

- Insertar un registro

```
//insercion directa
sqLiteDatabase.execSQL("INSERT INTO Alumnos (codigo, nombre) VALUES (1, 'Juan')");

//insercion mediante objeto ContentValues
ContentValues registroNuevo=new ContentValues();
registroNuevo.put("codigo", 4);
registroNuevo.put("nombre", "Ana");
sqLiteDatabase.insert("Alumnos", null, registroNuevo);
```

- Eliminar un registro

```
//eliminación directa
sqLiteDatabase.execSQL("DELETE FROM Alumnos WHERE codigo=1");

//eliminación con método específico
sqLiteDatabase.delete("Alumnos", "codigo=2", null);
```

- Modificar un registro

```
//modificacion directa
sqLiteDatabase.execSQL("UPDATE Alumnos SET nombre='Eva' WHERE codigo=3");

//modificacion mediante objeto ContentValues
ContentValues registroParaModificar=new ContentValues();
registroParaModificar.put("nombre","Margarita" );
sqLiteDatabase.update("Alumnos", registroParaModificar, "codigo=4",null);
```

- Consultar uno/varios registros

  - Las consultas devuelven un objeto de tipo ***Cursor***, que no es más que un **objeto que contiene los resultados de la consulta**.
  - El cursor se recorre mediante un bucle para extraer cada uno de los datos encontrados.
  - Al final de su uso **hay que cerrar el cursor**.
  - Si buscamos por clave, dado que solo se recuperará un registro, no es necesario el bucle.
  - También tenemos la posibilidad de escribir la consulta directamente (método ***rawQuery()***) mediante el método parametrizado ***query()***

```
//busqueda directa mediante rawQuery
//UNICO REGISTRO
Cursor cursor1=sqLiteDatabase.rawQuery("SELECT nombre FROM Alumnos WHERE codigo=1",null);
if (cursor1.moveToFirst()){
    String nombre=cursor1.getString(0);//0 pq en SELECT solo recupero un dato=nombre
    Toast.makeText(this, "Nombre: "+nombre, Toast.LENGTH_SHORT).show();
}else
    Toast.makeText(this, "DATO INEXISTENTE", Toast.LENGTH_SHORT).show();
cursor1.close();
```

```
//busqueda mediante método específico
//UNICO REGISTRO
String[] datos={"nombre"};
Cursor cursor2=sqLiteDatabase.query("Alumnos",datos,"codigo=4",null, null,null,null,null);
if (cursor2.moveToFirst()){
    String nombre=cursor2.getString(0);//0 pq en SELECT solo recupero un dato=nombre
    Toast.makeText(this, "Nombre: "+nombre, Toast.LENGTH_SHORT).show();
}else
    Toast.makeText(this, "DATO INEXISTENTE", Toast.LENGTH_SHORT).show();
cursor2.close();
```

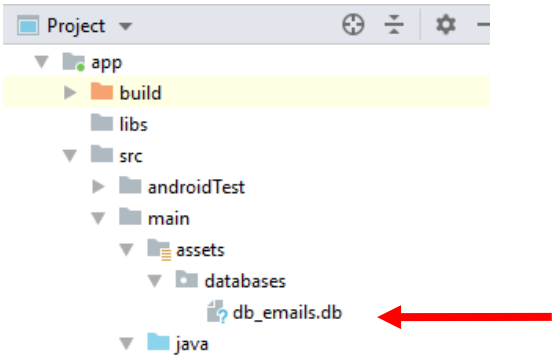
```
//busqueda directa
//VARIOS REGISTROS
Cursor cursor3=sqLiteDatabase.rawQuery("SELECT * FROM Alumnos",null);
if (cursor3.moveToFirst()){
    do{
        int codigo=cursor3.getInt(0);//campo1=codigo
        String nombre=cursor3.getString(1);//campo2=nombre
        Toast.makeText(this,"Código: "+codigo +"\nNombre: "+nombre,Toast.LENGTH_SHORT).show();
    }while(cursor3.moveToNext());
}else
    Toast.makeText(this, "DATOs INEXISTENTEs", Toast.LENGTH_SHORT).show();
cursor3.close();
```

```
//busqueda mediante método específico
//VARIOS REGISTROS
String[] datos2={"codigo","nombre"};
Cursor cursor4=sqLiteDatabase.query("Alumnos", datos2, null,null, null,null,null,null);
if (cursor4.moveToFirst()){
    do {
        int codigo = cursor4.getInt(0);//campo1=codigo
        String nombre = cursor4.getString(1);//campo2=nombre
        Toast.makeText(this, "Código: " + codigo + "\nNombre: " + nombre,
Toast.LENGTH_SHORT).show();
    }while(cursor4.moveToNext());
}else
    Toast.makeText(this, "DATO INEXISTENTE", Toast.LENGTH_SHORT).show();
cursor4.close();
```

3.2 CREADA DE FORMA EXTERNA.

- **Crear la base de datos durante la fase de diseño** (herramienta ***SQLite Studio***, por ejemplo)

- **Incluir la base de datos en nuestro proyecto**, **en carpeta "*assets*",** que es "hermana" de "res". El fichero de base de datos se copia en la ruta ***assets/databases***



- **Acceso a la base de datos**

  - Para acceder a la base de datos podemos trasladar su contenido a la carpeta /data/data/nombre_paquete/databases/ del dispositivo Android.

  - Esto puede hacerse leyendo el archivo fuente mediante un objeto ***InputStream*** y escribiendo los bytes en el archivo destino mediante un objeto ***OutputStream***

- Proceso más simple: utilizar una clase de ayuda (de forma similar a como hicimos anteriormente con SQLiteOpenHelper). Se trata de la clase ***SQLiteAssetHelper***.

  https://github.com/jgilfelt/android-sqlite-asset-helper

- Para poder emplear esta clase debemos **incorporar a nuestra aplicación la correpondiente librería**, añadiendo en el archivo ***build.gradle*** la siguiente dependencia:

Gradle

If you are using the Gradle build system, simply add the following dependency in your `build.gradle` file:

```
dependencies {
    compile 'com.readystatesoftware.sqliteasset:sqliteassethelper:+'
}
```

- El proceso es similar al anterior:

  - Creamos una clase que derive de ***SQLiteAssetHelper***, pero en este caso no necesitamos sobreescribir los métodos ***onCreate()*** y **onUpgrade()**, que recordamos que son los empleados para la creación y actualización de la base de datos en caso de que ésta sea generada desde el mismo entorno de Android.

  - Creamos un objeto de dicha clase y, sobre él, aplicamos el método ***getWriteableDatabase()*** o ***getReadableDatabase()*** según nos interese.

  - Cada uno de estos métodos devuelve una instancia de la clase ***SQLiteDatabase*** con la que podemos trabajar igual que en el caso de una base de datos generada desde la aplicación.