

Apunts

Descarregar aquest apunts [pdf](#) y [html](#)

Tema 11.1. Persistència de Dades I SQLite

Índex

1. [Accés a bases de dades locals. SQLite](#)
 1. [Introducció](#)
 2. [Creació i obertura de la BD](#)
 3. [Accés i modificació de les dades en la BD](#)
 1. [Accés a la BD mitjançant sentències sql](#)
 2. [Accés a la BD mitjançant SQLiteDatabase](#)
 3. [Recuperació de dades amb rawQuery](#)
 4. [Recuperació de dades amb query](#)
 4. [Ús de SimpleCursorAdapter](#)
 1. [Cursors amb RecyclerView](#)

Accés a bases de dades locals. SQLite

Introducció

La plataforma Android proporciona dues eines principals per a l'emmagatzematge i consulta de dades estructurades: la base de dades **SQLite** i **Content Providers** (ho treballarem en temes posteriors).

Ens centrarem en **SQLite**, encara que no entrarem ni en el disseny de BBDD relacionals ni en l'ús avançat de la BBDD. Per a conèixer tota la funcionalitat de SQLite es recomana usar la documentació oficial.

Per a l'accés a les bases de dades tindrem tres classes que hem de conèixer. La classe **SQLiteOpenHelper**, que encapsula totes les funcions de creació de la base de dades i versions d'aquesta. La classe **SQLiteDatabase**, que incorpora la gestió de taules i dades. I finalment la classe **Cursor**, que usarem per a moure'ns pel *recordset que ens retorna una consulta *SELECT*.

Creació i obertura de la BD

El mètode recomanat per a la creació de la base de dades és estendre la classe **SQLiteOpenHelper** i sobreescriure els mètodes **onCreate** i **onUpgrade**. El primer mètode s'utilitza per a crear la base de dades per primera vegada i el segon per a actualitzacions d'aquesta. Si la base de dades ja existeix i la seua versió actual coincideix amb la sol·licitada, es realitzarà la connexió a ella.

Per a executar la sentència SQL utilitzarem el mètode **execSQL** proporcionat per la API per a SQLite d'Android.

```

internal inner class BDClientes(
    context: Context?,
    name: String?,
    factory: CursorFactory?,
    version: Int) : SQLiteOpenHelper(context, name, factory, version)
{
    var sentencia =
        "create table if not exists clientes" +
        "(dni VARCHAR PRIMARY KEY NOT NULL, nombre TEXT, apellidos TEXT);"
    override fun onCreate(sqliteDatabase: SQLiteDatabase) {
        SQLiteDatabase.execSQL(sentencia)
    }
    override fun onUpgrade(sqliteDatabase: SQLiteDatabase, i: Int, i2: Int) {}
}

```

🚩 **Nota:** Si la base de dades existeix però la seua versió actual és anterior a la sol·licitada, es dirà automàticament al mètode `onUgrde()` . Si la base de dades no existeix es diu automàticament a `onCreate()` .

Accés i modificació de les dades en la BD

Una vegada creada la classe, instanciem un objecte del tipus creat i usarem un dels dos següents mètodes per a tindre accés a la gestió de dades:

`getWritableDatabase()` per a accés d'escriptura i `getReadableDatabase()` per a accés de només lectura. En tots dos casos es retornarà un objecte de la classe `SQLiteDatabase` per a l'accés a les dades.

Per a realitzar una codificació neta i reutilitzable, el millor és crear una classe amb tots els mètodes que necessitem per a treballar amb la Base de dades, a més és útil tindre la classe derivada de `SQLiteOpenHelper` interna a aquesta.

```


class BDAdapter(context: Context?)
{
    private var clientes: BDClientes

    init {
        clientes = BDClientes(context, "BDClientes", null, 1)
    }

    ...

    internal inner class BDClientes(
        context: Context?,
        name: String?,
        factory: CursorFactory?,
        version: Int) : SQLiteOpenHelper(context, name, factory, version)
    {
        var sentencia =
            "create table if not exists clientes" +
            "(dni VARCHAR PRIMARY KEY NOT NULL, nombre TEXT, apellidos TEXT);"
        override fun onCreate(sqliteDatabase: SQLiteDatabase) {
            SQLiteDatabase.execSQL(sentencia)
        }
        override fun onUpgrade(sqliteDatabase: SQLiteDatabase, i: Int, i2: Int)
    }
}

```

 **Nota:** Com podem veure en el codi, creguem un objecte del tipus de BDClientes amb el qual treballarem cridant al constructor BDClientes(), a aquest mètode li passem el context, el nom de la BBDD, un objecte cursor (amb valor null) i la versió de la BD que necessitem.

Per a poder usar els mètodes creats en la classe, tan sols instanciem un objecte d'aquesta passant-li el context (de tipus activity). Amb aquesta instància podrem cridar a qualsevol dels mètodes que crearem en la classe d'utilitat.

```

var bdAdapter = BDAdapter(this)
bdAdapter.insertarDatos()

```

Accés a la BD mitjançant sentències sql

L'accés a la BBDD es fa en dues fases, primer s'aconsegueix un objecte del tipus **SQLiteDatabase** i posteriorment usem les seues funcions per a gestionar les dades. Si hem obert correctament la base de dades llavors podrem començar amb les insercions, actualitzacions, esborrat, etc. No haurem d'oblidar tancar l'accés a la BD, sempre que no s'estiga usant un cursor.

Per exemple, inserirem 10 registres en la nostra taula clients. Per a això podrem crear un mètode en el **BDAdapter** semblant al següent:

```
fun insertarDatos() {
    var dbClientes = clientes.writableDatabase
    if (dbClientes != null) {
        for (i in 0..9) {
            val sentencia = "INSERT INTO Clientes (dni, nombre, apellidos) V
                            " ('" + i + "', 'nombre" + i + "', 'apellido" + i + "');"
            dbClientes.execSQL(sentencia)
        }
        dbClientes.close()
    }
}
```

👉 Val, i ara què? On està la base de dades que acabem de crear? Com podem comprovar que tot ha anat bé i que els registres s'han inserit correctament?, en primer lloc vegem on s'ha creat la nostra base de dades. Totes les bases de dades SQLite creades per aplicacions Android s'emmagatzemen en la memòria del telèfon en un fitxer amb el mateix nom de la base de dades situat en una ruta que segueix el següent patró:

/data/data/paquete.java.de.la.aplicacion/databases/nom_base_dades

En el cas del nostre exemple, la base de dades s'emmagatzemaria per tant en la ruta següent: **/data/data/exemple.basedatos/databases/DBClientes**

Per a comprovar que les dades s'han gravat podem accedir de manera remota a l'emulador a través de la seua consola de comandos (shell). Però és més senzill obrir l'arxiu creat amb un bloc de notes i donar una ullada al seu contingut.

Accés a la BD mitjançant SQLiteDatabase

La API d'Android ens ofereix dos mètodes per a accedir a les dades. El primer d'ells ja ho hem vist, es tracta d'executar sentències SQL a través de **execSql**. Aquest mètode té com a paràmetre una cadena amb qualsevol instrucció SQL vàlida. L'altra forma és utilitzar els mètodes específics **insert()**, **update()** i **delete()** de la classe **SQLiteDatabase**.

Vegem a continuació cadascun d'aquests mètodes.

- **insert()** : rep tres paràmetres `insert(table, nullColumnHack, values)`, el primer és el nom de la taula, el segon s'utilitza en cas que necessitem inserir valors nuls en la taula "nullColumnHack" en aquest cas el deixarem passar ja

que no l'usarem i per tant el posem a *null i el tercer són els valors del registre a inserir. Els valors a inserir els passarem al seu torn com a elements d'una col·lecció de tipus `ContentValues`. Aquests elements s'emmagatzemen com a parelles **clau-valor**, on la clau serà el nom de cada camp i el valor serà la dada que s'inserirà.

- `update()` : Pràcticament és igual que l'anterior mètode però amb l'excepció que ací estem usant el mètode `update(table, values, whereClause, whereArgs)` per a actualitzar/modificar registres de la nostra taula. Aquest mètode ens demana el nom de la taula "table", els valors a modificar/actualitzar "values" (`ContentValues`), una condició WHERE "whereClause" que ens serveix per a indicar-li que valor volem que actualitzi i com a últim paràmetre "whereArgs" podem passar-li els valors nous a inserir, en aquest cas no el necessitem per tant el posem a null.
- `delete()` : el mètode `delete(table, whereClause, whereArgs)` ens demana el nom de la taula "table", el registre a esborrar "whereClause" que prendrem com a referència la seua aneu i com a últim paràmetre "whereArgs" els valors a esborrar.

Inserint amb ContentValues

```
val dbClientes = clientes.writableDatabase
val valores = ContentValues()
valores.put("nombre", "Xavi")
valores.put("dni", "22222111")
valores.put("apellidos", "Perez Rico")
dbClientes.update("clientes", valores, "dni=4", null)
dbClientes.close()
```

Borrant amb ContentValues

```
val dbClientes = clientes.writableDatabase
if (dbClientes != null) {
    val valores = ContentValues()
    valores.put("nombre", cliente.nombre)
    valores.put("dni", cliente.dni)
    valores.put("apellidos", cliente.apellidos)
    dbClientes.insert("clientes", null, valores)
    dbClientes.close()
}
```

Modificant amb ContentValues

```
val dbClientes = clientes.writableDatabase
val arg = arrayOf("1")
dbClientes.delete("clientes", "dni=?", arg)
dbClientes.close()
```

Els valors que hem deixat anteriorment com null són realment arguments que podem utilitzar en la sentència SQL. Vegem-ho amb un exemple:

```
val dbClientes = clientes.writableDatabase
val valores = ContentValues()
valores.put("nombre", "Carla")
val arg= arrayOf("6", "7")
dbClientes.update("clientes", valores, "dni=? OR dni=?", arg)
dbClientes.close()
```

On les `?` indiquen els emplaçaments dels arguments.



Crea un exercici EjercicioResueltoBD en el qual proves el codi vist anteriorment: Creació de la BD, inserir elements amb SQL i modificar, eliminar i inserir mitjançant ContentValues.

mde_sa

Recuperació de dades amb rawQuery

Existeixen dues maneres de recuperar informació (SELECT de la base de dades), encara que ambdues es recolzen en el concepte de **cursor**, que és en definitiva l'objecte que recull els resultats de la consulta. Completarem l'exercici anterior, perquè ens permeti anar inserint registres en la BD i visualitzant-los en un listView.

La primera manera d'obtenir dades és utilitzar el mètode `rawQuery()` de la classe `SQLiteDatabase`. Aquest mètode rep com a paràmetre la sentència SQL, on s'indiquen els camps a recuperar i els criteris de selecció.

EjemploBD

dni

nombre

apellidos

AÑADIR

MOSTRAR

nombre5
apellido5
5

nombre0
apellido0
0

nombre3
apellido3
3

nombre1
apellido1
1

```
fun seleccionarDatosSelect(sentencia: String?): Boolean {  
    val listaCliente: ArrayList<Clientes>?  
    val dbClientes = clientes.readableDatabase  
    if (dbClientes != null) {  
        val cursor: Cursor = dbClientes.rawQuery(sentencia, null)  
        listaCliente = getClientes(cursor)  
        dbClientes.close()  
        return if (listaCliente == null) false else true  
    }  
    return false  
}
```

Evidentment, ara haurem de recórrer el cursor i visualitzar les dades retornades. Per a això es disposa dels mètodes `moveToFirst()`, `moveToNext()`, `moveToLast()`, `moveToPrevious()`, `isFirst()` i `isLast()`.

Existeixen mètodes específics per a la recuperació de dades `getXXX(indexe)`, on XXX indica el tipus de dada (String, Blob, Float,...) i el paràmetre índex permet recuperar la columna indicada en aquest, tenint en compte que comença en 0.

El mètode `getClientes` (implementat per nosaltres) és el que ens permet llegir les dades existents en la BD i portar-los al `arrayList`:


```

fun getClientes(cursor: Cursor): ArrayList<Clientes>? {
    val clientes: ArrayList<Clientes>
    var cliente: Clientes
    cursor.moveToFirst()
    if (!cursor.isAfterLast()) {
        clientes = ArrayList()
        while (!cursor.isAfterLast()) {
            cliente =
                Clientes(cursor.getString(0), cursor.getString(1), cursor.getStrin
            clientes.add(cliente)
            cursor.moveToNext()
        }
        return clientes
    }
    return null
}

```

Recuperació de dades amb query

La segona manera de recuperar informació de la BD és utilitzar el mètode `query()`. Rep com a paràmetres el nom de la taula, un string amb els camps a recuperar, un string on especificar les condicions del WHERE, un altre per als arguments si n'hi haguera, un altre per al GROUP BY, un altre per a HAVING i finalment un altre per a ORDER BY.


```

fun seleccionarDatosCodigo(
    columnas: Array<String?>?,
    where: String?,
    valores: Array<String?>?,
    orderBy: String?): ArrayList<Clientes>?
{
    var listaCliente: ArrayList<Clientes>? = ArrayList()
    val dbClientes = clientes.readableDatabase
    if (dbClientes != null) {
        val cursor: Cursor =
            dbClientes.query("clientes", columnas, where, valores, null, null,
            listaCliente = getClientes(cursor)
            dbClientes.close()
            return listaCliente
        }
    }
    return null
}

```

On l'anomenada al mètode podria ser la següent:

```
val listaCliente = bdAdapter.seleccionarDatosCodigo(  
    arrayOf("dni", "nombre", "apellidos"),  
    null,  
    null,  
    "apellidos")
```

 **Crea un exercici ejemploBD en el qual proves el codi vist anteriorment: Recuperació de les dades amb rawQuery i amb query. Per a això crea l'aplicació com en la imatge, de manera que amb un botó es guarde la informació en la BD i amb l'altre s'extraiga i es mostre en una llista o en un recycler. Si utilitzareu una llista com es mostra en la imatge, deureu crear una classe adaptador com la de la següent imatge i associar-la a la llista amb el setAdapter:**

```
binding.listView.setAdapter(  
    AdaptadorClientes(  
        this@MainActivity,  
        R.layout.list_layout,  
        listaCliente!! ))
```

Classe Adaptador per a un ListView:

```
class AdaptadorClientes(var activitycontext: Activity,  
    resource: Int,  
    objects: ArrayList<Clientes>  
    ) :  
    ArrayAdapter<Any>(activitycontext, resource, objects as List<Any>) {  
    var objects: ArrayList<Clientes>  
    override fun getView(position: Int,  
        convertView: View?,  
        parent: ViewGroup): View {  
        var vista: View? = convertView  
        if (vista == null) {  
            val inflater = activitycontext.layoutInflater  
            vista = inflater.inflate(R.layout.list_layout, null)  
            (vista?.findViewById(R.id.apellidolist) as TextView)  
                .setText(objects[position].apellidos)  
            (vista?.findViewById(R.id.nombrelist) as TextView)  
                .setText(objects[position].nombre)  
            (vista?.findViewById(R.id.dnिलist) as TextView)  
                .setText(objects[position].dni)  
        }  
        return vista  
    }  
  
    init {  
        this.objects = objects  
    }  
}
```

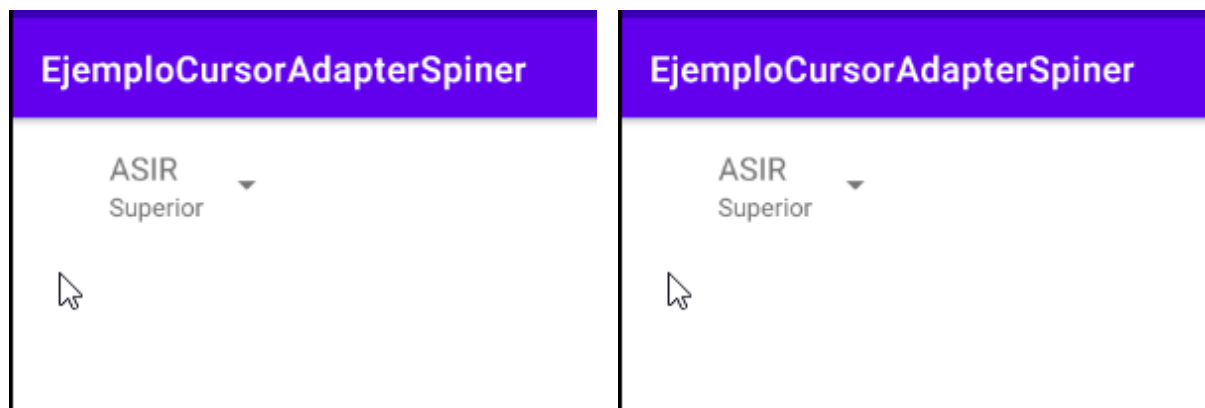
 **Exercici Propost Recórrer BD**

Ús de SimpleCursorAdapter

S'utilitza per a mapear les columnes d'un cursor obert en una base de dades als diferents elements visuals continguts en el control de selecció. Així s'evita el bolcat de tots els elements descarregats de la BD a una col·lecció, amb l'estalvi de memòria que comporta.

Ens trobem amb dos casos: quan usem `Spinner` o `ListView` senzills l'adaptador que gestiona aquest bolcat ja està creat, si usem `RecyclerView` haurem de crear i gestionar l'adaptador.

Programarem un exemple per a practicar aquest concepte, serà una senzilla activitat amb un `Spinner` que mostrarà els cicles informàtics amb la categoria a la qual pertanyen:



Per a això ens creem la base de dades i inserim elements com ja hem vist fins al moment:

```

class DBAdapter(context: Context) {
    private var ohCategoria: OHCategoria
    init {
        ohCategoria = OHCategoria(context, "BBDCategorias", null, 1)
    }
    fun insertarDatosCodigo() {
        val sqLiteDatabase = ohCategoria.writableDatabase
        if (sqLiteDatabase != null) {
            val valores = ContentValues()
            valores.put("nombre", "ASIR")
            valores.put("cate", "Superior")
            valores.put("idcategoria", 1)
            sqLiteDatabase.insert("categoria", null, valores)
            valores.put("nombre", "DAM")
            valores.put("cate", "Superior")
            valores.put("idcategoria", 2)
            sqLiteDatabase.insert("categoria", null, valores)
            valores.put("nombre", "SMR")
            valores.put("cate", "Medio")
            valores.put("idcategoria", 3)
            sqLiteDatabase.insert("categoria", null, valores)
            sqLiteDatabase.close()
        }
    }
    fun leerDatos():Cursor?
    {
        val sqLiteDatabase = ohCategoria.readableDatabase
        if (sqLiteDatabase != null) {
            return sqLiteDatabase.rawQuery(
                "select idcategoria as _id, nombre, cate from categoria",
                null )
        }
        return null
    }

    inner class OHCategoria(
        context: Context?, name: String?,
        factory: SQLiteDatabase.CursorFactory?,
        version: Int):QLiteOpenHelper(context, name, factory, version)
    {
        var cadena =
            "create table if not exists categoria(idcategoria
            INTEGER PRIMARY KEY NOT NULL, nombre TEXT, cate TEXT);"
        override fun onCreate(db: SQLiteDatabase) {
            db.execSQL(cadena)
        }
        override fun onUpgrade(db: SQLiteDatabase,
                                oldVersion: Int, newVersion: Int) {}
    }
}

```

Posteriorment passem a crear el cursorAdapter, però per a això usarem un Spinner per a mostrar la informació, ho incloem en el layout principal.

```
1  override fun onCreate(savedInstanceState: Bundle?) {  
2      super.onCreate(savedInstanceState)  
3      val from = arrayOf("nombre", "cate")  
4      val to = intArrayOf(R.id.ciclo, R.id.cate)  
5      setContentView(R.layout.activity_my)  
6      val dbAdapter=DBAdapter(this)  
7      dbAdapter.insertarDatosCodigo()  
8      val desplegable = findViewById<Spinner>(R.id.spinner)  
9      val cursor=dbAdapter.leerDatos()  
10     val mAdapter = SimpleCursorAdapter(this, R.layout.spinner_layout,  
11                                         cursor, from, to, 0x0)  
12     desplegable.setAdapter(mAdapter)  
13 }
```

✦ **Nota:** En la **Línia 9** es retorna el cursor amb la selecció de les dades de la BD (compte!! no s'haurà de tancar el cursor). **Línia 19** per al funcionament del **SimpleCursorAdapter**, és necessari indicar en la seua definició els següents elements: el context, el layout sobre el qual es definirà la vista, el cursor que serà origen de les dades, llista amb el nom de les columnes que es vol enllaçar, una llista amb els aneu de les vistes on es mostraran els camps (l'ordre és important) i un flag (0 per defecte)

✍ **Prova l'aplicació de l'exemple anterior**

Cursors amb RecyclerView

Com era d'esperar, no es pot aplicar directament un adaptador de tipus

`SimpleCursorAdapter` a un tipus `RecyclerView`. Si volem aprofitar els beneficis dels cursors en els recylcers, haurem de fabricar-los a mesura. Els passos són els següents:

1. Crear una classe **Abstracta** base que derive de **RecyclerView.Adapter** i en la qual implementarem els mètodes sobreescrits derivats de `RecyclerView.Adapter`:

```
1  abstract class CursorRecyclerViewAdapter(cursor: Cursor) :
2  RecyclerView.Adapter<RecyclerView.ViewHolder>() {
3  var mCursor: Cursor
4  override fun onBindViewHolder(holder: RecyclerView.ViewHolder,
5                                position: Int)
6  {
7      checkNotNull(mCursor) { "ERROR, cursos vacio" }
8      check(mCursor.moveToPosition(position)) { "ERROR, no se puede" +
9          " encontrar la posicion: $position" }
10     onBindViewHolder(holder, mCursor)
11 }
12 abstract fun onBindViewHolder(holder: RecyclerView.ViewHolder,
13                                cursor: Cursor)
14 override fun getItemCount(): Int
15 {
16     return if (mCursor != null)    mCursor.getCount()
17         else 0
18 }
19 init {
20     mCursor = cursor
21 }
22 }
```

🚩 **Nota:** En heretar de `RecyclerView.Adapter` caldrà implementar els mètodes `onBindViewHolder()` al qual li arriba la posició de la línia del recycler que està activa en aqueix moment i `getItemCount()` que haurà de retornar els elements que té el cursor **Línia 16**, li arriba en el constructor. En el mètode `onBindViewHolder` es comprovarà que el cursor no està vacio i que es pot accedir a la posició que arriba com a paràmetre **Línia 8**, i en aqueix cas es dirà al mètode

`onBindViewHolder(holder: RecyclerView.ViewHolder, cursor: Cursor)` abstracte que hem de crear en aquesta classe **Línia 12**, però aquesta vegada amb un `Cursor` com a paràmetre.

2. Ara haurem de crear la nostra classe `RecyclerView` derivada de l'anterior.

```

1  class RecyclerViewAdapter(c: Cursor) : CursorRecyclerViewAdapter(c) {
2      override fun onCreateViewHolder(parent: ViewGroup,
3                                     viewType: Int): RecyclerView.ViewHolder
4      {
5          val v: View = LayoutInflater.from(parent.context)
6              .inflate(R.layout.recycler_layout, parent, false)
7          return SimpleViewHolder(v)
8      }
9      override fun onBindViewHolder(holder: RecyclerView.ViewHolder,
10                                  cursor: Cursor)
11      {
12          (holder as SimpleViewHolder).bind(cursor)
13      }
14      internal inner class SimpleViewHolder(itemView: View) :
15          RecyclerView.ViewHolder(itemView)
16      {
17          var nombre: TextView
18          var cate: TextView
19          var imagen: ImageView
20          fun bind(dato: Cursor) {
21              nombre.setText(dato.getString(0))
22              cate.setText(dato.getString(1))
23              val theImage: Bitmap = MainActivity.
24                  convertirStringBitmap(dato.getString(3))
25              imagen.setImageBitmap(theImage)
26          }
27          init {
28              nombre = itemView.findViewById(R.id.ciclo)
29              cate = itemView.findViewById(R.id.cate)
30              imagen = itemView.findViewById(R.id.imagen) as ImageView
31          }
32      }
33  }

```

📌 **Nota:** A aquesta classe li arribarà el cursor pel constructor que al seu torn serà passat a la classe pare. En heretar de la classe abstracta, haurem d'implementar el seu mètode abstracte i serà ací on aprofitarem que el cursos s'ha posionat en l'element correcte de la BD, per a cridar al mètode **bind(cursor)** **Línia 12** del **Holder** , amb aqueix estat del cursor. En el **Holder** tractarem els elements que traiem del cursos de manera correcta.

3. Com aquesta App és una mica diferent de l'anterior, s'han inclòs imatges per a donar més funcionalitat, hi ha dos mètodes que ens permeten passar imatges de Mapa de bits a String i viceversa per a poder-les guardar en la BD a través del cursor (En la BD es guardaran com Blob). Estan localitzats en MyActivity i són estàtics per a poder accedir a ells sense necessitat d'objecte.

EjemploCursorAdapter

ASIR
Superior

DAM
Superior

SMR
Medio

```
companion object
{
    fun convertirStringBitmap(imagen: String?): Bitmap {
        val decodedString: ByteArray = Base64.decode(imagen, Base64.DEFAULT)
        return BitmapFactory.decodeByteArray(decodedString, 0, decodedString.size)
    }

    fun ConvertirImagenString(bitmap: Bitmap): String {
        val stream = ByteArrayOutputStream()
        bitmap.compress(Bitmap.CompressFormat.PNG, 90, stream)
        val byte_arr: ByteArray = stream.toByteArray()
        return Base64.encodeToString(byte_arr, Base64.DEFAULT)
    }
}
```

 **Proba la aplicació del exemple anterior**

 **Exercici proposat cursor amb recycler**