Base de datos en Android

SQLite

Es una clase

Contrato

Para tener otra tabla hay que poner otro public static abstract class. Tantos como tablas tenga la db.

```
import android.provider.BaseColumns;

public class Contrato{
    private Contrato (){
    }

    public static abstract class TablaAgenda implements BaseColumns{
        public static final String TABLA = "agenda";
        public static final String NOMBRE = "nombre";
        public static final String TELEFONO = "telefono";
        public static final String CORREO = "correo";
    }
}
```

Crea un campo adicional, el id, automaticamente

Nombre de la tabla

Campos de la tabla

Ayudante

En onUpgrade
1º Crear Tablas Temporales
2º Llevo los Datos a las Tablas
3º Borro las Tablas
4º Creo las Tablas Nuevas
5º Importo los Datos de las Tablas Temporales
6º Borro las Tablas Temporales

```
public class Ayudante extends SQLiteOpenHelper {
        public static final String DATABASE_NAME = "agenda.db";
        public static final int DATABASE_VERSION = 1;
        public Ayudante(Context context) {
                super(context, DATABASE_NAME, null, DATABASE_VERSION);
        }
        @Override
        public void onCreate(SQLiteDatabase db) {
                String sql;
                sql="create table "+Contrato.TablaAgenda.TABLA+
                                " ("+_ID+" integer primary key autoincrement, “+
                                Contrato.TablaAgenda.NOMBRE+" text, "+
                                Contrato.TablaAgenda.TELEFONO+" text, "+
                                Contrato.TablaAgenda.CORREO+" text)";
                db.execSQL(sql);
        }
        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
                String sql="drop table if exists "+Contrato.TablaAgenda.TABLA;
                db.execSQL(sql);
                onCreate(db);
        }
}
```

Controla la Version de la Base de Datos. Si no cambia, no se ejecuta el onUpgrade si existe la DB.

El metodo onCreate del SQLiteOpenHelper crea la Base de Datos

El metodo onUpgrade actualiza la Bae de Datos (Cuando la estructura inicial ya existe)

Plain Old Java Object

```
public class Agenda {
	private long id; private String nombre; private String telefono; private String correo;
	public Agenda(){
		this(0,"","","");
	}
	public Agenda(long id, String nombre, String telefono, String correo) {
		this.id = id;
		this.nombre = nombre;
		this.telefono = telefono;
		this.correo = correo;
	}
	public long getId() {
		return id;
	}
	public void setId(long id) {
		this.id = id;
	}
	public String getNombre() {
		return nombre;
	}
	public void setNombre(String nombre) {
		this.nombre = nombre;
	}
	public String getTelefono() {
		return telefono;
	}
	public void setTelefono(String telefono) {
		this.telefono = telefono;
	}
	public String getCorreo() {
		return correo;
	}
	public void setCorreo(String correo) {
		this.correo = correo;
	}
	@Override
	public String toString(){
		return id+" "+nombre+" "+telefono+" "+correo;
	}
}
```

Gestión de la tabla

- Implementaremos las operaciones CRUD: create, read, update y delete.
- Crearemos además todos los métodos necesarios para implementar las operaciones que requiera la aplicación.
- public class GestionAgenda{
  private Ayudante abd;
  private **SQLiteDatabase** bd;
  }

Constructor y conexiones

```
public GestionAgenda(Context c) {
        abd = new Ayudante(c);
}

public void open() {
        bd = abd.getWritableDatabase();
}

public void openRead() {
        bd = abd.getReadableDatabase();
}

public void close() {
        abd.close();
}
```

Insertar

```
public long insert(Agenda ag) {
    ContentValues valores = new ContentValues();
    valores.put(Contrato.TablaAgenda.CORREO,
                ag.getCorreo());
    valores.put(Contrato.TablaAgenda.NOMBRE,
                ag.getNombre());
    valores.put(Contrato.TablaAgenda.TELEFONO,
                ag.getTelefono());
    long id = bd.insert(Contrato.TablaAgenda.TABLA,
                null, valores);
    return id;
}
```

Eliminar

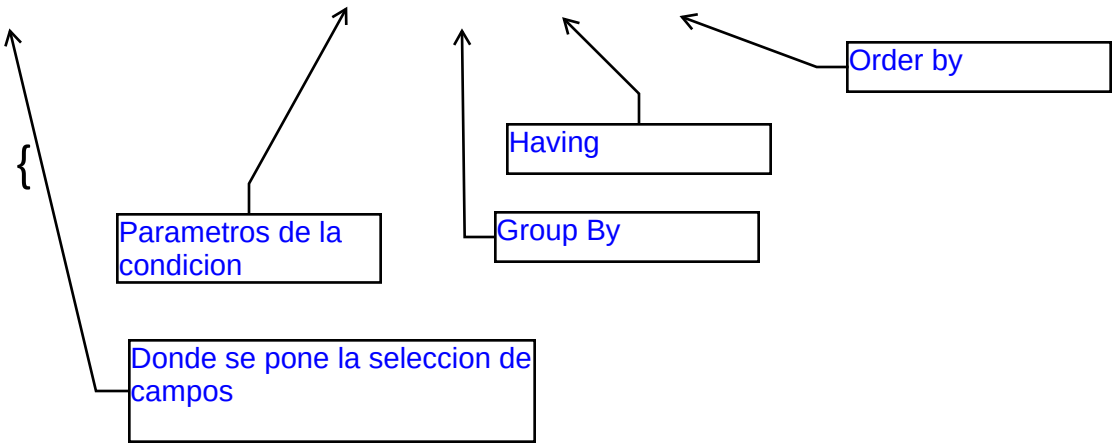
```
public int delete(Agenda ag) {
   String condicion = _ID + " = ?";
   String[] argumentos = { ag.getId() + "" };
   int cuenta = bd.delete(
         Contrato.TablaAgenda.TABLA, condicion,
         argumentos);
   return cuenta;
}
```

Editar

```
public int update(Agenda ag) {
    ContentValues valores = new ContentValues();
    valores.put(Contrato.TablaAgenda.CORREO, ag.getCorreo());
    valores.put(Contrato.TablaAgenda.NOMBRE, ag.getNombre());
    valores.put(Contrato.TablaAgenda.TELEFONO, ag.getTelefono());
    String condicion = _ID + " = ?";
    String[] argumentos = { ag.getId() + "" };
    int cuenta = bd.update(Contrato.TablaAgenda.TABLA, valores,
                                condicion, argumentos);
    return cuenta;
}
```

Consultar

```
public List<Agenda> select(String condicion) {
    List<Agenda> la = new ArrayList<Agenda>();
    Cursor cursor = bd.query(Contrato.TablaAgenda.TABLA,
                                    null, condicion, null, null, null, null);
    cursor.moveToFirst();
    Agenda ag;
    while (!cursor.isAfterLast()) {
            ag = getRow(cursor);
            la.add(ag);
            cursor.moveToNext();
    }
    cursor.close();
    return la;
}
```



Obtener fila a partir del cursor

```
public Agenda getRow(Cursor c) {
    Agenda ag = new Agenda();
    ag.setId(c.getLong(0));
    ag.setNombre(c.getString(1));
    ag.setTelefono(c.getString(2));
    ag.setCorreo(c.getString(3));
    return ag;
}
```

Obtener fila a partir del id

```
public Agenda getRow(long id) {
    String[] proyeccion = { Contrato.TablaAgenda._ID,
                            Contrato.TablaAgenda.NOMBRE,
                            Contrato.TablaAgenda.TELEFONO,
                            Contrato.TablaAgenda.CORREO };
    String where = Contrato.TablaAgenda._ID + " = ?";
    String[] parametros = new String[] { id+"" };
    String groupby = null; String having = null;
    String orderby = Contrato.TablaAgenda.NOMBRE + " ASC";
    Cursor c = bd.query(Contrato.TablaAgenda.TABLA, proyeccion,
                        where, parametros, groupby, having, orderby);
    c.moveToFirst();
    Agenda  ag = getRow(c);
    c.close();
    return ag;
}
```

Obtener fila a partir de otro campo

```
public Agenda getRow(String nombre) {
    String[] parametros = new String[] { nombre };
    Cursor c = bd.rawQuery("select * from "
                                + Contrato.TablaAgenda.TABLA
                                + " where "
                                + Contrato.TablaAgenda.NOMBRE
                                + " = ?", parametros);
    c.moveToFirst();
    Agenda  ag = getRow(c);
    c.close();
    return ag;
}
```

Los métodos query() y rawQuery() representan dos posibles formas de lanzar una consulta, aunque se prefiere normalmente el método query(), el método rawQuery() pueda dar más juego en consultas más complejas.

Tratamiento de las excepciones

```
try {
    … // operación sobre la base de datos
} catch (android.database.sqlite.SQLiteConstraintException e){
…
} catch(android.database.sqlite.SQLiteException e){
…
}
```

Apertura y cierre de conexión

En principio no es necesario cerrar las conexiones puesto que éstas se cierran solas, sin embargo, esto puede ser problemático al pasar de una actividad a otra. En este caso, lo mejor es cerrar la conexión con el cierre temporal de la actividad y abrirla al activarla.

```
@Override
public void onResume() {
   gestortabla.open();
   super.onResume();
}
```

```
@Override
public void onPause() {
   super.onPause();
   gestortabla.close();
}
```

Mostrar los datos en un ListView

- Se extiende la clase adaptadora CursorAdapter.

- Constructor:

```
public Adaptador (Context co, Cursor cu) {
    super(co, cu, true);
}
```

Método bindView()

```
@Override
public void bindView(View v, Context co,
                                        Cursor cu) {
   TextView tv1;
   tv1=(TextView) v.findViewById(R.id.tv);
   tv1.setText(cu.getString(1));
   …
}
```

Método newView()

```
@Override
public View newView(Context co, Cursor cu,
                                        ViewGroup vg) {
   LayoutInflater i = LayoutInflater.from(
                                          vg.getContext());
   View v = i.inflate(R.layout.detalle, vg, false);
   return v;
}
```

Uso del cursor y el adaptador

Asignación del adaptador:

```
GestionAgenda ga = new GestionAgenda(this);
ga.open();
Cursor c = ga.getCursor();
Adaptador ad = new Adaptador(getApplicationContext(), c);
ListView lv = (ListView) findViewById(R.id.lista);
listView.setAdapter(ad);
```

Reasignación del adaptador:

```
c = ga.getCursor();
ad.changeCursor(c);
```

Uso de las clases

```
GestionAgenda gta = new GestionAgenda(this);
gta.open();
Agenda ag = new Agenda(1,"Pepe","958123456","pepe@pepe.es");
long r=ag.insert(ag);
ag = new Agenda(2,"Juan","958123457",“juan@pepe.es");
r=gta.insert(ag);
Cursor c=gta.getCursor();
c.moveToFirst();
while (!c.isAfterLast()) {
        ag = gta.getRow(c);
        tv.append("\ncursor"+ag+"\n");
        c.moveToNext();
}
ArrayList a=(ArrayList<Agenda>)gta.select(null);
tv.append("\n"+a.toString()+"\n");
a=(ArrayList<Agenda>)gta.select("nombre='Juan'");
tv.append("\n2"+a.toString()+"\n");
ag=gta.getRow(1);
tv.append("\nrow"+ag+"\n");
ag=gta.getRow("Juan");
tv.append("\nrow"+ag+"\n");
gta.close();
```