ALMACENAMIENTO PERMANENTE

Programación multimedia
y dispositivos móviles
2º DAM

ÍNDICE DE CONTENIDOS

1. INTRODUCCIÓN

Android incorpora la librería SQLite, que nos permitirá utilizar bases de datos mediante el lenguaje SQL

SQLite es un motor de base de datos que tiene como principal característica ser uno de los motores más ligeros que podemos encontrar, debido a que no emplea el esquema tradicional cliente-servidor, sino que escribe directamente en ficheros dentro del espacio de almacenamiento.

2. CONTRACT CLASS

Como en cualquier otro gestor de bases de datos, lo primero que debemos hacer es definir el esquema de nuestra base de datos , para ello, Android nos recomienda crear una clase Contract Class en la que almacenaremos, como constantes, las características de la base de datos.
Esta clase implementa la interfaz BaseColumns.
No es obligatorio su uso, pero si recomendable para evitar fallos con el nombre de los campos de la base de datos en los diferentes usos que hagamos.

```
public class AlumnosContract  implements  BaseColumns{
        public static final String TABLE_NAME ="alumnos";


        public static final String DNI = "dni";
        public static final String NOMBRE = "nombre";
        public static final String APELLIDOS = "apellidos";
        public static final String EDAD = "edad";

    }
```

2. MODELO

Una vez hemos creado la clase que almacenará el esquema, es recomendable (Aunque lo podríamos pasar por alto) crear una clase que represente a cada una de las tablas de nuestra Base de Datos, en nuestro caso, la clase Alumno.

```java
public class Alumno {
    private String nombre;
    private String apellido;
    private String dni;
    private int edad;

    public Alumno(String nombre, String apellido, String dni, int edad) {
        this.nombre = nombre;
        this.apellido = apellido;
        this.dni = dni;
        this.edad = edad;
    }

    public Alumno() {
        super();
    }

    public String getNombre() {
        return nombre;
    }
```

3. BASE DE DATOS

El siguiente paso es crear nuestra base de datos, para ello el SDK de Android nos provee de la clase abstracta **SQLiteOpenHelper**.

Esta clase nos proporciona toda la funcionalidad necesaria para trabajar con bases de datos en Android:

- Creación de la base de datos onCreate()
- Modificación de la base de datos onUpdate()
- Abrir la base de datos en modo lectura getReadableDatabase()
- Abrir la base de datos en modo escritura getWritableDatabase()

3. BASE DE DATOS

Por tanto, el siguiente paso es crear una clase que extienda de **SQLiteOpenHelper**

```
public class SQLiteAlumnosHelper extends SQLiteOpenHelper {
    private static final int DATABASE_VERSION = 1;
    private static final String DATABASE_NAME = "Alumnos.db";


    public SQLiteAlumnosHelper(Context context) {

        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }
```

Declaramos las variables para:

- Versión de la base de datos
- Nombre de la base de datos.

Creamos el constructor en el cual llamamos al constructor de la clase padre.

- Context → Contexto de la aplicación
- Nombre de la base de datos,
- Cursor Factory
- Versión de la base de datos

3. BASE DE DATOS

Una vez que hemos creado el constructor, vamos a implementar los métodos onCreate() y onUpgrade(), en los cuales debemos poner la sentencia SQL de creación de la tabla.

```
@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL("CREATE TABLE " + AlumnosContract.TABLE_NAME + " ("
            + AlumnosContract.DNI + " TEXT PRIMARY KEY,"
            + AlumnosContract.NOMBRE + " TEXT NOT NULL,"
            + AlumnosContract.APELLIDOS + " TEXT NOT NULL,"
            + AlumnosContract.EDAD + " INTEGER NOT NULL )");

}

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    db.execSQL("CREATE TABLE " + AlumnosContract.TABLE_NAME + " ("
            + AlumnosContract.DNI + " TEXT PRIMARY KEY,"
            + AlumnosContract.NOMBRE + " TEXT NOT NULL,"
            + AlumnosContract.APELLIDOS + " TEXT NOT NULL,"
            + AlumnosContract.EDAD + " INTEGER NOT NULL )");
}
```

4. INSERTAR

Llegados a este punto, ya tenemos la estructura de la base de datos creada, ahora debemos insertar información en la misma para comenzar a usarla.

Para ello usaremos el método SqlDatabase.insert() con los siguientes parámetros:

- Nombre de la tabla→ String
- Nombre de una columna que no acepta nulos → String
- Valores a insertar → ContentValues
    - ContentValues es una clase que nos permite almacenar información en pares de clave-valor.

```
// Contenedor de valores
ContentValues values = new ContentValues();

// Pares clave-valor
values.put(AlumnosContract.DNI, "123145D");
values.put(AlumnosContract.NOMBRE, "Carlos");
values.put(AlumnosContract.APELLIDOS, "Gomez");
values.put(AlumnosContract.EDAD, 12);


// Insertar
db.insert(AlumnosContract.TABLE_NAME, null, values);
```

4. INSERTAR

Llegados a este punto, ya tenemos la estructura de la base de datos creada, ahora debemos insertar información en la misma para comenzar a usarla.

Para ello usaremos el método SqlDatabase.insert() con los siguientes parámetros:

- Nombre de la tabla→ String
- Nombre de una columna que no acepta nulos → String
- Valores a insertar → ContentValues
    - ContentValues es una clase que nos permite almacenar información en pares de clave-valor.

```
// Contenedor de valores
ContentValues values = new ContentValues();

// Pares clave-valor
values.put(AlumnosContract.DNI, "123145D");
values.put(AlumnosContract.NOMBRE, "Carlos");
values.put(AlumnosContract.APELLIDOS, "Gomez");
values.put(AlumnosContract.EDAD, 12);


// Insertar
db.insert(AlumnosContract.TABLE_NAME, null, values);
```

5. INSERTAR

Una vez que sabemos cómo insertar información en la BBDD, lo más oportuno es crear un método específico para realizar inserciones en cada una de las tablas. En nuestro caso, sólo tenemos la tabla alumno, por tanto el método quedaría de la siguiente forma.

```
public long nuevoAlumno(Alumno a){
    ContentValues values = new ContentValues();
    values.put(AlumnosContract.DNI, a.getDni());
    values.put(AlumnosContract.NOMBRE, a.getNombre());
    values.put(AlumnosContract.APELLIDOS, a.getApellido());
    values.put(AlumnosContract.EDAD, a.getEdad());
    SQLiteDatabase sqLiteDatabase = getWritableDatabase();
    return sqLiteDatabase.insert(AlumnosContract.TABLE_NAME, null, values);
}
```

Devuelve un número, que se corresponderá con las tuplas afectadas. Devolverá -1 en caso de error.

6. LEER

Para leer información de una BBDD utilizaremos el método SQLiteDatabase.query()

```
SQLiteDatabase db = getReadableDatabase();
Cursor c = db.query(
        AlumnosContract.TABLE_NAME,  // Nombre de la tabla
        null,  // Lista de Columnas a consultar
        null,  // Columnas para la cláusula WHERE
        null,  // Valores a comparar con las columnas del WHERE
        null,  // Agrupar con GROUP BY
        null,  // Condición HAVING para GROUP BY
        null  // Cláusula ORDER BY
);
```

- Este método nos devuelve un cursor con todos los datos recogidos por la consulta.

```
while(c.moveToNext()){
    String nombre = c.getString(c.getColumnIndex(AlumnosContract.NOMBRE));
    // Acciones a realizar...
}
```

7. MODIFICAR

Para actualizar registros de la BBDD utilizaremos el método SQLiteDatabase.update(Tabla, contentValues, camposWhere, valoresWhere)

```
return sql.update(AlumnosColumnas.TABLE_NAME,
        values,
        whereClause: AlumnosColumnas.DNI + " = ?",
        new String[]{alumno.getDni()});
```

Este método nos devuelve un int que contiene el número de tuplas afectadas, en caso de error devuelve -1.

8. ELIMINAR

Para eliminar registros de la BBDD utilizaremos el método SQLiteDatabase.delete(tabla, camposWhere, valoresWhere)

```
return sql.delete(AlumnosColumnas.TABLE_NAME,
        whereClause: AlumnosColumnas.DNI + " = ?",
        new String[]{alumno.getDni()});
```

Este método nos devuelve un int que contiene el número de tuplas afectadas, en caso de error devuelve -1.

9. DUDAS Y PREGUNTAS