**Daniel Felipe Cadavid Duque - 000148640**

**SQLite**

**Contenido**

[¿Qué Es SQLite? 1](#_Toc40191752)

[Crear clases de la entidades 2](#_Toc40191753)

[Estudiante\_Actividade 2](#_Toc40191754)

[Estado 4](#_Toc40191755)

[Estudiante 5](#_Toc40191756)

[Actividades 6](#_Toc40191757)

[Recursos 7](#_Toc40191758)

[Crear Base De Datos En SQLite 9](#_Toc40191759)

[Creando helper 9](#_Toc40191760)

[CRUD 11](#_Toc40191761)

[Create 11](#_Toc40191762)

[Read 12](#_Toc40191763)

[Update 13](#_Toc40191764)

[Delete 14](#_Toc40191765)

**¿Qué es SQLite?**

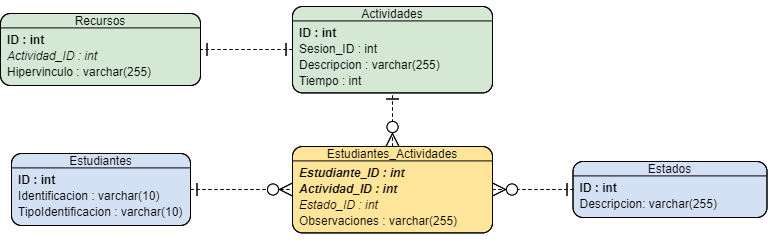
Es un ligero motor de bases de datos de código abierto, que se caracteriza por mantener el almacenamiento de información persistente de forma sencilla. A diferencia de otros Sistemas gestores de bases de datos como MySQL, SQL Server y Oracle DB, SQLite tiene las siguientes ventajas:

* **No requiere el soporte de un servidor:** SQLite no ejecuta un proceso para administrar la información, si no que implementa un conjunto de librerías encargadas de la gestión.
* **No necesita configuración:** Libera al programador de todo tipo de configuraciones de puertos, tamaños, ubicaciones, etc.
* **Usa un archivo para el esquema**: Crea un archivo para el esquema completo de una base de datos, lo que permite ahorrarse preocupaciones de seguridad, ya que los datos de las aplicaciones Android no pueden ser accedidos por contextos externos.
* **Es de Código Abierto:** Esta disponible al dominio público de los desarrolladores al igual que sus archivos de compilación e instrucciones de escalabilidad.

La forma en que una base de datos está estructurada (cantidad de tablas, registros, índices, etc.) y el conjunto de convenciones para nombrar sus objetos se les llama Esquema. Por lo general el esquema inicial se guarda en un Script que nos permita recuperar las condiciones previas en cualquier momento.

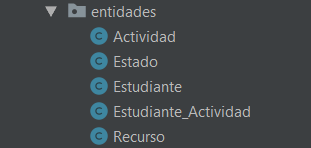
Con SQLite no es diferente, por lo que debes crear un esquema predefinido para implementarlo a la hora de crear tu base de datos.

El esquema definido es el siguiente



**Crear clases de la entidades**

Para representarlas se crea un nuevo paquete Java con el nombre entidades. Dentro de este, añade las clases:



**Estudiante\_Actividade**

La entidad llamada Estudiantes\_actividades, cuyos atributos son:

* estudiante\_ID
* actividad\_ID
* estado\_ID
* observaciones

|  |
| --- |
| package entidades;  import android.content.ContentValues;  import java.io.Serializable;  public class Estudiante\_Actividad implements Serializable {  private Integer estudiante\_ID;  private Integer actividad\_ID;  private Integer estado\_ID;  private String observaciones;   public Estudiante\_Actividad(Integer estudiante\_ID, Integer actividad\_ID, Integer estado\_ID, String observaciones) {  this.estudiante\_ID = estudiante\_ID;  this.actividad\_ID = actividad\_ID;  this.estado\_ID = estado\_ID;  this.observaciones = observaciones;  }   public Integer getEstudiante\_ID() {  return estudiante\_ID;  }   public void setEstudiante\_ID(Integer estudiante\_ID) {  this.estudiante\_ID = estudiante\_ID;  }   public Integer getActividad\_ID() {  return actividad\_ID;  }   public void setActividad\_ID(Integer actividad\_ID) {  this.actividad\_ID = actividad\_ID;  }   public Integer getEstado\_ID() {  return estado\_ID;  }   public void setEstado\_ID(Integer estado\_ID) {  this.estado\_ID = estado\_ID;  }   public String getObservaciones() {  return observaciones;  }   public void setObservaciones(String observaciones) {  this.observaciones = observaciones;  }  */\*\*  \* Este metodo auxiliar es utilizado para realizar  \* una interpretacion del objeto como un ContentValues  \** ***@return*** *representacion del objeto en ContentValue  \*/* public ContentValues toContentValues() {  ContentValues values = new ContentValues();  values.put("actividad\_ID", estudiante\_ID);  values.put("actividad\_ID", actividad\_ID);  values.put("estado\_ID",estado\_ID);  values.put("observaciones", observaciones);  return values;  } } |

**Estado**

La entidad llamada Estado, cuyos atributos son:

* estadoID
* descripción

|  |
| --- |
| package entidades;  import android.content.ContentValues;  import java.io.Serializable;  public class Estado implements Serializable {  private Integer estadoID;  private String descripcion;   public Estado(Integer estadoID, String descripcion) {  this.estadoID = estadoID;  this.descripcion = descripcion;  }   public Integer getEstadoID() {  return estadoID;  }   public void setEstadoID(Integer estadoID) {  this.estadoID = estadoID;  }   public String getDescripcion() {  return descripcion;  }   public void setDescripcion(String descripcion) {  this.descripcion = descripcion;  }   */\*\*  \* Este metodo auxiliar es utilizado para realizar  \* una interpretacion del objeto como un ContentValues  \** ***@return*** *representacion del objeto en ContentValue  \*/* public ContentValues toContentValues() {  ContentValues values = new ContentValues();  values.put("estadoID", estadoID);  values.put("descripcion", descripcion);  return values;  } } |

**Estudiante**

La entidad llamada Estudiante, cuyos atributos son:

* estudianteID
* identificacion
* tipoIdentificacion

|  |
| --- |
| package entidades;  import android.content.ContentValues;  public class Estudiante {  private int estudianteID;  private String identificacion;  private String tipoIdentificacion;   public Estudiante(int estudianteID, String identificacion, String tipoIdentificacion) {  this.estudianteID = estudianteID;  this.identificacion = identificacion;  this.tipoIdentificacion = tipoIdentificacion;  }   public int getEstudianteID() {  return estudianteID;  }   public void setEstudianteID(int estudianteID) {  this.estudianteID = estudianteID;  }   public String getIdentificacion() {  return identificacion;  }   public void setIdentificacion(String identificacion) {  this.identificacion = identificacion;  }   public String getTipoIdentificacion() {  return tipoIdentificacion;  }   public void setTipoIdentificacion(String tipoIdentificacion) {  this.tipoIdentificacion = tipoIdentificacion;  }  */\*\*  \* Este metodo auxiliar es utilizado para realizar  \* una interpretacion del objeto como un ContentValues  \** ***@return*** *representacion del objeto en ContentValue  \*/* public ContentValues toContentValues() {  ContentValues values = new ContentValues();  values.put("estudianteID", estudianteID);  values.put("identificacion", identificacion);  values.put("tipoIdentificacion", tipoIdentificacion);  return values;  } } |

**Actividades**

La entidad llamada Actividades, cuyos atributos son:

* actividadID
* sesión\_ID
* descripción
* tiempo

|  |
| --- |
| package entidades;  import android.content.ContentValues;  public class Actividad {  private int actividadID;  private int sesion\_ID;  private String descripcion;  private int tiempo;   public Actividad(int actividadID, int sesion\_ID, String descripcion, int tiempo) {  this.actividadID = actividadID;  this.sesion\_ID = sesion\_ID;  this.descripcion = descripcion;  this.tiempo = tiempo;  }   public int getActividadID() {  return actividadID;  }   public void setActividadID(int actividadID) {  this.actividadID = actividadID;  }   public int getSesion\_ID() {  return sesion\_ID;  }   public void setSesion\_ID(int sesion\_ID) {  this.sesion\_ID = sesion\_ID;  }   public String getDescripcion() {  return descripcion;  }   public void setDescripcion(String descripcion) {  this.descripcion = descripcion;  }   public int getTiempo() {  return tiempo;  }   public void setTiempo(int tiempo) {  this.tiempo = tiempo;  }  */\*\*  \* Este metodo auxiliar es utilizado para realizar  \* una interpretacion del objeto como un ContentValues  \** ***@return*** *representacion del objeto en ContentValue  \*/* public ContentValues toContentValues() {  ContentValues values = new ContentValues();  values.put("actividadID", actividadID);  values.put("sesion\_ID",sesion\_ID);  values.put("descripcion", descripcion);  values.put("tiempo", tiempo);  return values;  } } |

**Recursos**

La entidad llamada Recursos, cuyos atributos son:

* recursoID
* actividad\_ID
* hipervínculo

|  |
| --- |
| package entidades;  import android.content.ContentValues;  public class Recurso {  private int recursoID;  private String hipervinculo;  private int actividad\_ID;   public Recurso(int recursoID, int actividad\_ID, String hipervinculo) {  this.recursoID = recursoID;  this.actividad\_ID = actividad\_ID;  this.hipervinculo = hipervinculo;  }   public int getRecursoID() {  return recursoID;  }   public void setRecursoID(int recursoID) {  this.recursoID = recursoID;  }   public String getHipervinculo() {  return hipervinculo;  }   public void setHipervinculo(String hipervinculo) {  this.hipervinculo = hipervinculo;  }   public int getActividad\_ID() {  return actividad\_ID;  }   public void setActividad\_ID(int actividad\_ID) {  this.actividad\_ID = actividad\_ID;  }  */\*\*  \* Este metodo auxiliar es utilizado para realizar  \* una interpretacion del objeto como un ContentValues  \** ***@return*** *representacion del objeto en ContentValue  \*/* public ContentValues toContentValues() {  ContentValues values = new ContentValues();  values.put("recursoID", recursoID);  values.put("actividad\_ID", actividad\_ID);  values.put("hipervinculo", hipervinculo);  return values;  } } |

**Crear Base De Datos En SQLite**

El Android SDK nos provee una serie de clases para administrar nuestro archivo de base de datos en SQLite.

Normalmente cuando conectamos otro gestor de bases de datos tenemos que validar los datos del equipo, el usuario y el esquema, pero con SQLite no se requiere nada de eso, ya que podemos trabajar directamente sobre la base de datos.

La clase que nos permitirá comunicar nuestra aplicación con la base de datos se llama SQLiteOpenHelper. Se trata de una clase abstracta que nos provee los mecanismos básicos para la relación entre la aplicación Android y la información.

Para implementar este controlador debes:

* Crear una clase que extienda de SQLiteOpenHelper
* Configurar un constructor apropiado
* Sobrescribir los métodos onCreate() y onUpgrade()

**Creando helper**

1. Crea nueva clase que extienda de SQLiteOpenHelper y ConexionSQLiteHelper.



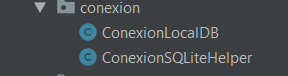
2. Escribe el constructor y usa super para mantener la herencia del helper.

3. Sobrescribe el método onCreate().

4. Sobrescribe el método onUpgrade().

|  |
| --- |
| package conexion;  import android.content.Context; import android.database.sqlite.SQLiteDatabase; import android.database.sqlite.SQLiteOpenHelper;   public class ConexionSQLiteHelper extends SQLiteOpenHelper {  */\*\*  \* Queries para la creacion de la Tablas, donde se definen  \* las columnas de cada tabla con el tipo correspondoiente de datos,  \* la llave primaria y las llaves foraneas  \* para estas ultimas en caso de eliminacion o actualizacion  \* la respuesta sera hacer esto mismo en cascada.  \*/* final String Crear\_Tabla\_estados="CREATE TABLE estados(estadoID INTEGER PRIMARY KEY, descripcion TEXT)";   final String Crear\_Tabla\_recursos="CREATE TABLE recursos(recursoID INTEGER PRIMARY KEY," +  "actividad\_ID INTEGER,hipervinculo TEXT,FOREIGN KEY (actividad\_ID) REFERENCES actividades(actividadID) ON DELETE CASCADE ON UPDATE CASCADE)";   final String Crear\_Tabla\_estudiantes="CREATE TABLE estudiantes(estudianteID INTEGER PRIMARY KEY," +  "identificacion TEXT,tipoIdentificacion TEXT)";   final String Crear\_Tabla\_actividades="CREATE TABLE actividades(actividadID INTEGER PRIMARY KEY, " +  "sesion\_ID INTEGER, descripcion TEXT, tiempo INTEGER)";   final String Crear\_Tabla\_estudiante\_actividades="CREATE TABLE estudiantes\_actividades(estudiante\_ID INTEGER NOT NULL, " +  "actividad\_ID INTEGER NOT NULL, estado\_ID INTEGER, observaciones TEXT,FOREIGN KEY (estudiante\_ID) REFERENCES estudiantes(estudianteID) ON DELETE CASCADE ON UPDATE CASCADE," +  "FOREIGN KEY (actividad\_ID) REFERENCES actividades(actividadID) ON DELETE CASCADE ON UPDATE CASCADE," +  "FOREIGN KEY (estado\_ID) REFERENCES estados(estadoID) ON DELETE CASCADE ON UPDATE CASCADE," +  "PRIMARY KEY ( estudiante\_ID, actividad\_ID) )";     public ConexionSQLiteHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version) {  super(context, name, factory, version);  }  @Override  public void onCreate(SQLiteDatabase db) {  db.execSQL(Crear\_Tabla\_estados);  db.execSQL(Crear\_Tabla\_estudiantes);  db.execSQL(Crear\_Tabla\_actividades);  db.execSQL(Crear\_Tabla\_recursos);  db.execSQL(Crear\_Tabla\_estudiante\_actividades);  //Activar las foreign Keys  db.execSQL("PRAGMA foreign\_keys=ON");  //o  db.setForeignKeyConstraintsEnabled(true);  }   @Override  public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {  db.execSQL("DROP TABLE IF EXISTS estudiantes\_actividades");  db.execSQL("DROP TABLE IF EXISTS estados");  db.execSQL("DROP TABLE IF EXISTS recursos");  db.execSQL("DROP TABLE IF EXISTS estudiantes");  db.execSQL("DROP TABLE IF EXISTS actividades");   db.execSQL(Crear\_Tabla\_estados);  db.execSQL(Crear\_Tabla\_estudiantes);  db.execSQL(Crear\_Tabla\_actividades);  db.execSQL(Crear\_Tabla\_recursos);  db.execSQL(Crear\_Tabla\_estudiante\_actividades);  //Activar las foreign Keys  db.execSQL("PRAGMA foreign\_keys=ON");  //o  db.setForeignKeyConstraintsEnabled(true);   }   } |

**CRUD**  
Para simplificar el Uso de la Base de Datos, el CRUD se realizada desde una nueva clase llamada ConexionLocalDB



|  |
| --- |
| public class ConexionLocalDB {  private ConexionSQLiteHelper conn; //conexion a BD.   public ConexionLocalDB(Context context, String name, SQLiteDatabase.CursorFactory factory, int version) {  conn=new ConexionSQLiteHelper(context,name,factory,version);  }  //CRUD  } |

**Create**

|  |
| --- |
| */\*\*  \* Metodos sobrecargados utilizados para la insercion de un nuevo objeto a la BD.  \* Reciben el objeto/registro que se desea Insertar.  \** ***@return*** *resultado de la insercion  \*/* //Create public long Create(Estado estado ){  SQLiteDatabase sqLiteDatabase = conn.getWritableDatabase();  return sqLiteDatabase.insert("estados", null, estado.toContentValues()); } public long Create(Actividad actividad){  SQLiteDatabase sqLiteDatabase = conn.getWritableDatabase();  return sqLiteDatabase.insert("actividades", null, actividad.toContentValues()); } public long Create(Estudiante estudiante){  SQLiteDatabase sqLiteDatabase = conn.getWritableDatabase();  return sqLiteDatabase.insert("estudiantes", null, estudiante.toContentValues()); } public long Create(Recurso recurso){  SQLiteDatabase sqLiteDatabase = conn.getWritableDatabase();  return sqLiteDatabase.insert("recursos", null, recurso.toContentValues()); } public long Create(Estudiante\_Actividad est\_act){  SQLiteDatabase sqLiteDatabase = conn.getWritableDatabase();  return sqLiteDatabase.insert("estudiantes\_actividades", null, est\_act.toContentValues()); } |

**Read**

|  |
| --- |
| */\*\*  \* Metodos para leer infromacion de la BD.  \* Reciben la llave primaria del registro que se desea buscar.  \** ***@return*** *el objeto/registro encontrado o NULL en caso negativo de la consulta  \*/* //Read public Estado ReadEstado(int estadoID){  SQLiteDatabase sqLiteDatabase = conn.getReadableDatabase();  String query = "select \* from estados WHERE estadoID=?";  Cursor c= sqLiteDatabase.rawQuery(query, new String[]{String.*valueOf*(estadoID)});  if (c.moveToFirst()) return new Estado(estadoID,c.getString(c.getColumnIndex("descripcion")));  return null;   } public Actividad ReadActividad(int actividadID){  SQLiteDatabase sqLiteDatabase = conn.getReadableDatabase();  String query = "select \* from actividades WHERE actividadID=?";  Cursor c= sqLiteDatabase.rawQuery(query, new String[]{String.*valueOf*(actividadID)});  if (c.moveToFirst()) return new Actividad(actividadID,c.getInt(c.getColumnIndex("sesion\_ID")),c.getString(c.getColumnIndex("descripcion")),c.getInt(c.getColumnIndex("tiempo")));  return null; } public Estudiante ReadEstudiante(int estudianteID){  SQLiteDatabase sqLiteDatabase = conn.getReadableDatabase();  String query = "select \* from estudiantes WHERE estudianteID=?";  Cursor c= sqLiteDatabase.rawQuery(query, new String[]{String.*valueOf*(estudianteID)});  if (c.moveToFirst()) return new Estudiante(estudianteID,c.getString(c.getColumnIndex("identificacion")),c.getString(c.getColumnIndex("tipoIdentificacion")));  return null; }  public Recurso ReadRecurso(int recursoID){  SQLiteDatabase sqLiteDatabase = conn.getReadableDatabase();  String query = "select \* from recursos WHERE estudiante\_ID=?";  Cursor c= sqLiteDatabase.rawQuery(query, new String[]{String.*valueOf*(recursoID)});  if (c.moveToFirst()) return new Recurso(recursoID,c.getInt(c.getColumnIndex("actividad\_ID")),c.getString(c.getColumnIndex("hipervinculo")));  return null; } public Estudiante\_Actividad ReadEstudiante\_Actividad(int estudiante\_ID,int actividad\_ID){  SQLiteDatabase sqLiteDatabase = conn.getReadableDatabase();  String query = "select \* from estudiantes\_actividades WHERE estdiante\_ID=? AND actividad\_ID=?";  Cursor c= sqLiteDatabase.rawQuery(query, new String[]{String.*valueOf*(estudiante\_ID),String.*valueOf*(actividad\_ID)});  if (c.moveToFirst()) return new Estudiante\_Actividad(estudiante\_ID,actividad\_ID,c.getInt(c.getColumnIndex("estado\_ID")),c.getString(c.getColumnIndex("observaciones")));  return null; } |

**Update**

|  |
| --- |
| */\*\*  \* Metodos utilizados para actualizar la BD  \* Reciben la llave primaria de la tabla en cuestion  \* y un objeto con los parametros que se desean cambiar.  \** ***@return*** *el resultado de la operacion en la BD  \*/* //Update public int UpdateEstado(Estado estado,int estadoID){  SQLiteDatabase sqLiteDatabase = conn.getWritableDatabase();  return sqLiteDatabase.update("estados",estado.toContentValues(),"estadoID LIKE ?",new String[]{String.*valueOf*(estadoID)}); } public int UpdateActividad(Actividad actividad,int actividadID){  SQLiteDatabase sqLiteDatabase = conn.getWritableDatabase();  return sqLiteDatabase.update("actividades",actividad.toContentValues(),"actividadID LIKE ?",new String[]{String.*valueOf*(actividadID)}); } public int UpdateEstudiante(Estudiante estudiante,int estudianteID){  SQLiteDatabase sqLiteDatabase = conn.getWritableDatabase();  return sqLiteDatabase.update("estudiantes",estudiante.toContentValues(),"actividadID LIKE ?",new String[]{String.*valueOf*(estudianteID)}); } public int UpdateRecurso(Recurso recurso,int recursoID){  SQLiteDatabase sqLiteDatabase = conn.getWritableDatabase();  return sqLiteDatabase.update("recursos",recurso.toContentValues(),"recursoID LIKE ?",new String[]{String.*valueOf*(recursoID)}); } public int UpdateEstudiante\_Actividad(Estudiante\_Actividad estudiante\_Actividad,int estudiante\_ID,int actividad\_ID){  SQLiteDatabase sqLiteDatabase = conn.getWritableDatabase();  return sqLiteDatabase.update("recursos",estudiante\_Actividad.toContentValues(),"estdiante\_ID LIKE ? AND actividad\_ID LIKE ?",new String[]{String.*valueOf*(estudiante\_ID),String.*valueOf*(actividad\_ID)}); } |

**Delete**

|  |
| --- |
| */\*\*  \* Metodos Utilizados para eliminar informacion de la BD  \* Reciben la llave primaria del objeto que se desea eliminar  \** ***@return*** *el resultado de la operacion en la BD  \*/* //Delete public int DeleteEstado(int estadoID){  SQLiteDatabase sqLiteDatabase = conn.getWritableDatabase();  return sqLiteDatabase.delete("estados","estodoID LIKE ?",new String[]{String.*valueOf*(estadoID)}); } public int DeleteActividad(int actividadID){  SQLiteDatabase sqLiteDatabase = conn.getWritableDatabase();  return sqLiteDatabase.delete("actividades","actividadID LIKE ?",new String[]{String.*valueOf*(actividadID)}); } public int DeleteEstudiante(int estudianteID){  SQLiteDatabase sqLiteDatabase = conn.getWritableDatabase();  return sqLiteDatabase.delete("estudiantes","estudianteID LIKE ?",new String[]{String.*valueOf*(estudianteID)}); } public int DeleteRecurso(int recursoID){  SQLiteDatabase sqLiteDatabase = conn.getWritableDatabase();  return sqLiteDatabase.delete("recursos","recursoID LIKE ?",new String[]{String.*valueOf*(recursoID)}); } public int DeleteEstudiante\_Actividad(int estudiante\_ID,int actividad\_ID){  SQLiteDatabase sqLiteDatabase = conn.getWritableDatabase();  return sqLiteDatabase.delete("estudiantes\_actividades","estdiante\_ID LIKE ? AND actividad\_ID LIKE ?",new String[]{String.*valueOf*(estudiante\_ID),String.*valueOf*(actividad\_ID)}); } |

**Referencias**

<http://www.hermosaprogramacion.com/2014/10/android-sqlite-bases-de-datos/>

<https://stackoverflow.com/questions/734689/sqlite-primary-key-on-multiple-columns>

<https://sqlite.org/foreignkeys.html>

<https://sqlite.org/foreignkeys.html>

<http://www.tutorialesprogramacionya.com/sqliteya/detalleconcepto.php?punto=44&codigo=44&inicio=40>

<https://entityframework.net/es/knowledge-base/47554766/como-usar-on-delete-cascade-con-sqlite-y-entity-framework-6>

<https://www.techonthenet.com/sqlite/foreign_keys/foreign_delete.php>