SQLite – Actualización de tablas y campos autoincrementales

En el tutorial anterior habíamos aprendido a crear bases de datos ***SQlite***, y sus tablas. Vimos que existe el método ***onCreate***, que se ejecuta en forma automática por la clase ***SQLiteOpenHelper*** cuando esta considera que debe crear la base, es decir, cuando la base no existe.

Pero también vimos que hay otro método, llamado ***onUpgrade***, al que habíamos decidido matar con nuestra indiferencia. Pues bien, debemos arrepentirnos, porque ahora lo necesitamos. Este método es usado, también en forma automática, por la clase ***SQliteOpenHelper***, cuando considere que la versión de la aplicación actual es distinta a la versión de la aplicación que creó la base de datos, y por lo tanto, podría requerir actualizaciones en la base.

onUpgrade

*Cómo sabe el **onUpgrade** cuál es la versión actual de la aplicación? Eh, eh, cómo lo sabe?!?*

Lo sabe por medio del cuarto parámetro de instanciación de la clase manejadora de la base de datos:

```
protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);
    setContentView(R.layout.actividad_principal);

    Log.d("BaseDeDatos", "Instancio el manejador de bases de datos");
    MiManejador=new ManejadorDeBaseDeDatos(this, "BasePersonas", null, 1);

}


private boolean BaseDeDatosAbierta() {

    boolean Respuesta;

    Log.d("BaseDeDatos", "Intento abrir la base en modo lectura/escritura");
    MiBase=MiManejador.getWritableDatabase();

    Log.d("BaseDeDatos", "Verifico que se haya podido abrir correctamente");
    Respuesta = (MiBase != null);

    return Respuesta;

}
```

Ese parámetro le indica al manejador de la base de datos qué número de versión tiene nuestra aplicación actualmente. El ***SQLiteOpenHelper*** usa ese valor para determinar qué debe hacer con la base de datos, respecto a su actualización. Es decir, el ***SQLiteOpenHelper*** determina, en base a este número, si la base de datos que está en el dispositivo corresponde con la versión de la aplicación que se está ejecutando ahora, o fue creada con una versión más vieja y requiere algunos cambios. Pasemos al párrafo siguiente para entenderlo mejor.

La cuestión de las versiones

Vamos a detallar mejor esta cuestión del ***onUpgrade*** y las versiones. Supongamos que yo creo una aplicación desde cero, es decir, NO estoy mejorando una versión previa, sino que estoy creando una aplicación en su primera versión. Esta aplicación requiere una base de datos ***SQLite*** llamada ***BasePersonas*** con una sola tabla llamada ***Personas***, con campos ***Nombre, Edad*** y ***Domicilio***. Dado que esta es la versión ***1*** de mi aplicación, ya que es la primera, en el cuarto parámetro de la instanciación mando un ***1***, para indicar que la base de datos corresponde a la versión ***1*** de la aplicación que la creó.

```
MiManejador=new ManejadorDeBaseDeDatos(this, "BasePersonas", null, 1);
```

A partir de este instante, cada vez que mi aplicación versión ***1*** se ejecute en un dispositivo (ya sea físico o virtual), hay dos escenarios posibles:

A) Que ese dispositivo esté ejecutando mi aplicación por primera vez.
B) Que ese dispositivo ya haya ejecutado esa aplicación anteriormente, es decir, que esta NO sea la primera vez.

Si es **A)**, resulta obvio que la ***BasePersonas*** no existe. En este caso, automáticamente es ejecutado el ***onCreate***, y NO se ejecuta el ***onUpgrade***. El ***onCreate*** entonces crea la base de datos, registra que la base fue creada con la versión ***1*** de nuestra aplicación, y luego ejecuta todas las instrucciones que le programemos, para crear la tabla ***Personas*** con sus tres campos ***Nombre***, ***Edad*** y ***Domicilio***.

En cambio, si es ***B)***, el ***onCreate*** NO se ejecuta, ya que la base existe, y el ***onCreate*** se ejecuta solamente cuando la base de datos NO existe.

Y el ***onUpgrade*** se ejecuta? La propia clase manejadora de base de datos, internamente, se hace a sí misma las siguientes preguntas: ***qué versión de la aplicación creó la base de datos?*** La 1. ***Y qué versión de la aplicación se está ejecutando ahora?*** También la 1. Entonces, NO ejecuta el ***onUpgrade***, porque asume que si la base fue creada con la misma versión que está en ejecución ahora, entonces la base no requiere ningún cambio. O sea, si mi aplicación es la 1, el ***onUpgrade*** no se ejecuta nunca.

Resulta que, tiempo después, mejoramos nuestra aplicación de forma que permita registrar la nacionalidad de las personas. Es decir, tenemos que agregarle un cuarto campo a la tabla ***Personas***, llamado ***Nacionalidad***. Dado que nuestra nueva versión tiene una estructura de la base de datos distinta a la anterior (aunque que solo sea un campo nuevo), nuestra aplicación ya es versión 2, y debemos mandar este 2 en la instanciación de nuestra clase manejadora de la base. Nuestro ***onCreate***, en la instrucción que crea la tabla, ya tendrá este cuarto campo, de forma que será algo así:

```java
public class ManejadorDeBaseDeDatos extends SQLiteOpenHelper {

    public ManejadorDeBaseDeDatos(Context Contexto, String NombreDeLaBase, SQLiteDatabase.CursorFactory Fabrica, int NumeroDeVersion) {
        super(Contexto, NombreDeLaBase, Fabrica, NumeroDeVersion);
    }

    @Override
    public void onCreate(SQLiteDatabase BaseDeDatos) {
        Log.d("Base", "Se ejecuta el onCreate");

        String InstruccionAEjecutar;
        InstruccionAEjecutar="create table personas (nombre text, domicilio text, edad integer, nacionalidad text)";

        Log.d("Base", "Voy a ejecutar: "+InstruccionAEjecutar);
        BaseDeDatos.execSQL(InstruccionAEjecutar);

        Log.d("Base", "Ejecutada exitosamente");
    }
```

Pero aparecen ahora, para el dispositivo en el que estamos ejecutando la aplicación, **tres escenarios posibles**, ya no solo dos:

A) Que se esté ejecutando mi aplicación por primera vez.
B) Que se haya ejecutado mi aplicación anteriormente, también con la versión 2, que es la actual.
C) Que se haya ejecutado mi aplicación anteriormente, pero con la versión 1.

Si es **A)**, se ejecutará el ***onCreate***, se creará la base registrando que se creó con la versión 2, se creará la tabla ***Personas*** con los cuatro campos, y todos felices.

Pero si es B) o C), cuando la clase manejadora de base de datos se haga la pregunta de "*Qué versión de la aplicación se está ejecutando ahora, y qué versión se estaba ejecutando cuando se creó la base?*", hay dos respuestas posibles: Que la versión registrada en la base sea la misma que se está ejecutando ahora, o que sea distinta. **Si la versión es la misma**, estamos en presencia del caso B), y por lo tanto **NO se ejecuta el *onUpgrade***.

Pero si la versión que creó la base NO es la misma que la que está en ejecución ahora, entonces estamos en el caso C), SI se ejecuta el ***onUpgrade***, que recibe dos parámetros automáticamente: ***VersionAnterior*** y ***VersionActual***. ***VersionAnterior*** indica el número de versión registrado en la base de datos. ***VersionActual*** indica el número de versión de la aplicación actualmente en uso, es decir, la 2.

Para qué queremos saber con qué versión fue creada la base, y qué versión tenemos ahora, si con saber que la base fue creada con una versión más vieja que la actual alcanza?

Porque, a medida que nuestra aplicación crezca más y más, ya no solo tendremos bases creadas con versión 1 o con versión 2, sino que podremos tener muchas versiones. Y cuando actualicemos la base de datos a la versión actual, tendremos que hacer una serie de condicionales que, dependiendo de qué versión venimos, sean los cambios a aplicar: si venimos de la 1, los campos y tablas nuevos son estos. Pero si venimos de la 2, son estos otros. Si venimos de la 3 son estos de más allá. Y tal.

Acá voy a mostrar una versión simplificada del ***onUpgrade***, que NO verifica distintas versiones, sino que directamente actualiza desde nuestra versión 1 a la 2. Nuestro ***onUpgrade***, podría ser más o menos así:

```
@Override
public void onUpgrade(SQLiteDatabase BaseDeDatos, int VersionDeLaBase, int VersionAplicacionActual) {
    Log.d("Base", "Se ejecuta el onUpgrade");
    Log.d("Base", "La versión de la base era: "+VersionDeLaBase);
    Log.d("Base", "La versión de la aplicación actual es "+VersionAplicacionActual);

    String SQLAEjecutar;
    SQLAEjecutar = "alter table personas add column Nacionalidad text";
    Log.d("Base", "Voy a ejecutar: "+SQLAEjecutar);
    BaseDeDatos.execSQL(SQLAEjecutar);
    Log.d("Base", "Ejecutado");

}
```

Aparece en escena una nueva instrucción del estándar ***SQL***, llamada ***alter table***, cuya funcionalidad no requiere demasiada explicación: le estoy diciendo que voy a modificar la estructura de la tabla ***personas***, agregándole un nuevo campo llamado ***Nacionalidad*** de tipo ***text***.

Autoincrementales

Los ***autoincrementales*** son campos cuyo valor no podemos determinar nosotros, sino que se completan automáticamente con una numeración secuencial respecto al último registro anteriormente agregado. Es decir, su contenido se rellena en forma automática: cada vez que agregemos un registro, el campo que nosotros definamos como autoincremental se completará automáticamente con un número más que el del registro anterior.

Indicarle a SQLite cuál campos queremos que sea autoincremental es tán simple como esto:

```
InstruccionAEjecutar="create table cliente (IdCliente integer autoincremental, razon_social text, domicilio text, edad integer)";
```

Con solo agregarle la cláusula ***autoincremental*** a cualquiera campo ***integer***, este campo asume automáticamente este comportamiento.

Para qué sirve los campos autoincrementales? En la mayoría de los casos (por no decir todos), las tablas tienen que tener algún campo que permita identificar unívocamente a cada registro, es decir, un campo que, con absoluta certeza, sepamos que tiene un valor distinto en cada registro. No es importante si este valor es numérico o alfanumérico. Tampoco es importante si es secuencia, o va de 2 en 2, o de 114 en 114, o random. Pero es absolutamente fundamental que sea diferente en todos los registros, porque esté será el único campo que podremos usar, con total certeza, para identificar a un registro de otro.

Los campos incrementales sirven justamente para esto: dado que es la propia base de datos la que se ocupa de asignarle valor al campo autoincremental de cada registro, nosotros podemos tener la certeza que jamás habrá dos registros con el mismo valor.

Cualquier alumno inteligente (como ustedes, claramente), debería estar preguntándose cómo obtener el valor autoincremental de un registro que se acaba de agregar. La respuesta es simple:

```
conjuntoDeRegistros = baseDatos.rawQuery("select last_insert_rowid()", null);
if (conjuntoDeRegistros.moveToFirst() == true) {
    ultimoIDGenerado = conjuntoDeRegistros.getInt(0);
}
```

Y para qué querríamos saber el valor de último campo autoincremental que se asignó a un ***insert***? Simplemente para cuando querramos asignarle ese valor a otra tabla relacionada. Por ejemplo, si damos de alta un equipo de fútbol, y luego queremos dar de alta un jugador cuyo campo ***CodigoEquipo*** contenga el ***Id*** del equipo que dimos de alta, necesitaremos, ni bien ejecutemos el ***insert*** en ***Equipos***, obtener qué código le tocó, para asignárselo al ***CodigoEquipo***.

En este mismo texto, un poco más arriba, vimos una forma de agregarle un campo a una tabla, usando la instrucción ***alter table***, y su comando ***add column***

Si bien eso sirve, aprendamos otra forma de hacer lo mismo, algo más compleja, pero más abarcativa. Conocer ambas modalidades nos permite elegir, en cada caso, cuál nos conviene. De esta forma, en nuestro ***onUpgrade***, podemos usar cualquiera de las dos formas, de acuerdo a nuestra decisión.

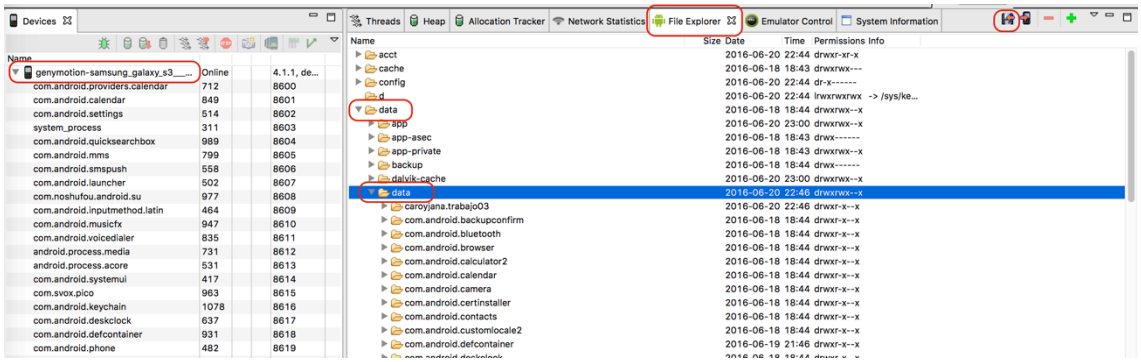
```
Log.d("Base", "Voy a renombrar la base");
BaseDeDatos.execSQL("alter table personas rename to PersonasTemporal");

Log.d("Base", "Voy a crear la tabla nuevamente, ya con el campo nuevo");
BaseDeDatos.execSQL("create table personas (nombre text, domicilio text, edad integer, nacionalidad text)");

Log.d("Base", "Voy a ponerle los registros que tenía la tabla que renombré antes");
BaseDeDatos.execSQL("insert into personas (nombre, domicilio, edad) select nombre, domicilio, edad from PersonasTemporal");

Log.d("Base", "Borro la tabla temporal que ya no me sirve para nada");
BaseDeDatos.execSQL("drop table PersonasTemporal");

Log.d("Base", "Terminé la actualización");
```

El código autocomentado y los Log ayudan muchísimo. Primero, renombro la tabla. Luego, la creo nuevamente, pero ahora con el cuarto campo. A continuación, a esta nueva tabla, le inserto el conjunto de registros que me trae el ***select*** de la tabla anterior. Finalmente, borro la tabla anterior, para que solo me quede la nueva, con el mismo nombre que antes, con los mismos datos que antes, y el campo nuevo que antes no existía.

Recordemos cómo podemos ver la base de datos en vivo y en directo. Primero, abrimos el ***Android Device Monitor***, una aplicación integrada al ***Android Studio*** que nos permite monitorear todo lo que está pasando en el emulador, entre otras cosas, su sistema de archivos.

Una vez abierto, nos aseguramos que nuestro emulador en ejecución esté seleccionado en el panel de dispositivos de la izquierda, seleccionamos "***File Explorer***" en la parte principal, vamos a la carpeta ***data***, a la subcarpeta ***data***, buscamos la subcarpeta correspondiente al ***package*** de nuestra aplicación, a la subcarpeta ***databases***, y ahí tendremos nuestro archivo de base de datos ***SQlite***.

Podemos copiarlo afuera del dispositivo por medio del botón "***Pull a file from the device***", copiándolo a cualquier carpeta de nuestra PC.



Ya con la base en nuestra PC, abrimos el ***SQLite Manager*** (cuya versión gratuita podemos descargar de acá: http://www.sqlabs.com/sqlitemanager.php), clickeamos en "***Open Other...***", y abrimos la base de la carpeta en la que la hayamos guardado.

Por último, seleccionamos "***Data***", la tabla que queramos, y ahí están nuestros datos.