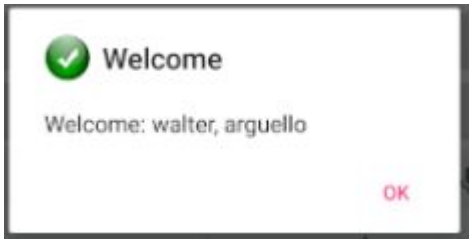
Desarrollo de aplicaciones móviles con Android

Babel Learning

SharedPreferences

Controles personalizados

Mensajes en pantalla donde se requiere solicitar una acción específica.




```
AlertDialog.Builder builder = new
AlertDialog.Builder(context);

builder.setTitle("Welcome");

builder.setMessage(mensaje);

builder.setIcon(R.drawable.okicon);

builder.show();
```

Controles personalizados

Adaptador personalizado.

Mostrar datos y organizarlos según sea la necesidad.

Ejercicios

#1. Modificar la aplicación donde se guardan los usuarios para que permita almacenar un listado de usuarios en las preferencias.

#2. Crear un control personalizado, para listar los usuarios guardados en las preferencias.

#3. Hacer un broadcast receiver para detectar:

a. Activo o desactivo el modo avión.
b. Conecto o desconecto el dispositivo de la electricidad.

¿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.

Ventajas

No requiere el soporte de un servidor: SQLite no ejecuta un proceso para administrar la información, sino que implementa un conjunto de librerías encargadas de la gestión.

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.

No necesita configuración: Libera al programador de todo tipo de configuraciones de puertos, tamaños, ubicaciones, etc.

Es de Código Abierto: Está disponible al dominio público de los desarrolladores al igual que sus archivos de compilación e instrucciones de escalabilidad.

Es por eso que SQLite es una tecnología cómoda para los dispositivos móviles. Su simplicidad, rapidez y usabilidad permiten un desarrollo muy amigable.

Definir Contrato De La Base De Datos

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.

La documentación de Android nos recomienda crear una clase llamada Contract Class, la cual guarda como constantes todas las características de la base de datos.

```
public class User {
    private String id;
    private String name;
    private String phoneNumber;

    public String getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public String getPhoneNumber() {
        return phoneNumber;
    }
}
```

Ejemplo clase

```
public class UserContract {
    public static abstract class UserEntry implements BaseColumns{
        public static final String TABLE_NAME ="user";

        public static final String ID = "id";
        public static final String NAME = "name";
        public static final String PHONE_NUMBER = "phoneNumber";
    }
}
```

Ejemplo de contrato

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.

Implementación

Para implementar este controlador debes:

- Crear una clase que extienda de SQLiteOpenHelper
- Configurar un constructor apropiado
- Sobrescribir los métodos onCreate() y onUpgrade()
- Creando helper de abogados

Crea nueva clase que extienda de SQLiteOpenHelper y llamarla UserDbHelper.

```
public class LawyersDbHelper extends SQLiteOpenHelper {
}


public static final int DATABASE_VERSION = 1;
public static final String DATABASE_NAME = "User.db";

public UserDbHelper(Context context) {
    super(context, DATABASE_NAME, null, DATABASE_VERSION);
}
```

Implementación

Parámetros del constructor

Los parámetros del constructor tienen la siguiente finalidad:

Context context: Contexto de acción para el helper.

String name: Nombre del archivo con extensión .db, donde se almacenará la base de datos, que a su vez corresponde al nombre de la base de datos.

CursorFactory factory: Asignamos null, por ahora no es necesario comprender el funcionamiento de este parámetro.

int version: Entero que representa la versión de la base de datos. Su valor inicial por defecto es 1. Si en algún momento la versión es mayor se llama al método onUpgrade() para actualizar la base de datos a la nueva versión. Si es menor, se llama a downUpgrade() para volver a una versión previa.

Este método es llamado automáticamente cuando creamos una instancia de la clase SQLiteOpenHelper. En su interior establecemos la creación de las tablas y registros.

```
@Override
public void onCreate(SQLiteDatabase sqLiteDatabase) {
    // Comandos SQL
}
```

Recibe como parámetro una referencia de la clase SQLiteDataBase, la cual actúa como manejadora de la base de datos.

Por defecto el archivo de la base de datos será almacenado en:

/data/data/<paquete>/databases/<nombre-de-la-bd>.db

```java
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    // No hay operaciones
}
```

SQLiteDatabase db: Manejador de la base de datos.

int oldVersion: Se trata de un entero que indica la versión antigua de la base de datos.

int newVersion: Entero que se refiere a la versión nueva de la base de datos.

```
query (String table,
       String[] columns,
       String selection,
       String[] selectionArgs,
       String groupBy,
       String having,
       String orderBy)
```

```
Cursor c = db.query(
    UserEntry.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
    );
```

Leer información de la base de datos

Propósito de los parámetros :

String table: Nombre de la tabla a consultar

String[] columns: Lista de nombres de las columnas que se van a consultar. Si deseas obtener todas las columnas usas null.

String selection: Es el cuerpo de la sentencia WHERE con las columnas a condicionar. Es posible usar el placeholder '?' para generalizar la condición.

String[] selectionArgs: Es una lista de los valores que se usaran para reemplazar las incógnitas de selection en el WHERE.

String groupBy: Aquí puedes establecer cómo se vería la cláusula GROUP BY, si es que la necesitas.

String having: Establece la sentencia HAVING para condicionar a groupBy.

String orderBy: Reordena las filas de la consulta a través de ORDER BY.

Otra forma de realizar consultas

Ahora, existe otro método alternativo para realizar consultas llamado rawQuery(). Con él pasas como parámetro un String del código SQL de la consulta.

```
db.rawQuery(

"select * from " +
UserEntry.TABLE_NAME, null);
```

Cursores en SQLite

Tanto query() como rawQuery() retornan un objeto de tipo Cursor.Este objeto es un apuntador al conjunto de valores obtenidos de la consulta.

Al inicio el cursor apunta a una dirección previa a la primera fila. Por lo que debes leer cada tupla moviendo el cursor a la fila siguiente.

Emplea el método booleano moveToNext() para avanzar al siguiente registro. Este retorna true si fue posible o false si ya no existen más elementos.

Este concepto es similar a cuando vimos cursores en SQL Server o en MySQL.

Donde recorremos cada elemento con un bucle while hasta que ya no existieran más elementos que referenciar.

```
while(c.moveToNext()){
 String name = c.getString(c.getColumnIndex(UserEntry.NAME));
 // Acciones...
}
```

Ejmplo

Cursores en SQLite

Usa métodos `get*()` para obtener el valor de cada columna a través del índice según su tipo de dato. Es decir, obtienes enteros con `getInt()`, flotantes con `getFloat()`, etc.

El índice de la columna se obtiene con `getColumnsIndex()`.

Cursores en SQLite

Se puede aprovechar este nuevo concepto e implementar un método de lectura para todos los abogados (getAllUsers()) y otro por ID (getUserById()).

La diferencia estaría en la lectura por id requiere ese elemento como parámetro.

```
public Cursor getAllUsers() {
    return getReadableDatabase()
            .query(
                    LawyerEntry.TABLE_NAME,
                    null,
                    null,
                    null,
                    null,
                    null,
                    null);
}
```

```
public Cursor getUserById(String userId) {
    Cursor c = getReadableDatabase().query(
            USerEntry.TABLE_NAME,
            null,
            UserEntry.ID + " LIKE ?",
            new String[]{userId},
            null,
            null,
            null);
    return c;
}
```

Cursores en SQLite