

Sqlite 関連コマンド紹介

■adb shell コマンド

現在、どのディレクトリに居るかは「pwd」コマンドでわかります。

pwd

次に今いるディレクトリ上にどんなファイル・ディレクトリ一覧があるかを知るために「ls」コマンドを実行します。

ls

ディレクトリへの移動は「cd」コマンドを実行します。

cd フォルダ（ディレクトリ）名

tmp ディレクトリを作ってみます。ディレクトリ作成は **mkdir** コマンドを使います。

mkdir tmp

■Sqlite コマンド

データベースの作成

データベースの作成はコマンドラインツールにデータベースファイル名を指定することで行う。データベースへの接続も同時に行う。

sqlite3 test.db

データベースへの接続

データベースの作成と同じです。ファイルが無ければ新規作成れる。

sqlite3 test.db

データベースを削除する

データベースは単なるファイルなので、adb Shell コマンドで削除することができます。

#rm test.db

テーブルの確認

```
sqlite> .tables
```

.tables コマンドでデータベース内のテーブル一覧を表示する。

テーブルスキーマの表示

```
sqlite> .schema mytable
```

.schema コマンドで指定のテーブルのスキーマを表示できる。

テーブル出力の整形

.explain コマンドを使うと、テーブル表示の際の整形機能の ON/OFF を設定できる。

- ・ OFF(デフォルト)の場合

```
sqlite> .explain OFF
```

```
sqlite> select * from mytable;
```

```
10 | komine
```

```
3 | suzuki
```

- ・ ON の場合

```
sqlite> .explain ON
```

```
sqlite> select * from mytable;
```

```
id    name
```

```
----  -
```

```
10    komine
```

```
3
```

ツールの終了

.quit または.exit コマンドを使う。

```
sqlite> .exit
```

Android で、データベースの作成(パターン1)

データベースの作成から行います。「ApplicationContext」クラスで用意されている「openOrCreateDatabase」メソッドを使います。

サンプルコード)

```
package jp.javadrive.android;

import android.app.Activity;
import android.os.Bundle;
import android.widget.LinearLayout;
import android.widget.Toast;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.view.View.OnClickListener;
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import java.io.FileNotFoundException;

public class JavaDriveAndroid extends Activity implements OnClickListener {
    private final int FP = ViewGroup.LayoutParams.FILL_PARENT;
    private final int WC = ViewGroup.LayoutParams.WRAP_CONTENT;

    private Button button;

    private final String DB_NAME = "db01_01.db";//ファイル名
    private final int DB_MODE = Context.MODE_PRIVATE;//非公開

    @Override
    protected void onCreate(Bundle icle) {
        super.onCreate(icle);
```

```

        LinearLayout linearLayout = new LinearLayout(this);
        linearLayout.setOrientation(LinearLayout.HORIZONTAL);
        setContentView(linearLayout);

        button = new Button(this);
        button.setText("Create Database");
        button.setOnClickListener(this);
        linearLayout.addView(button, createParam(WC, WC));
    }

    private LinearLayout.LayoutParams createParam(int w, int h) {
        return new LinearLayout.LayoutParams(w, h);
    }

    public void onClick(View v) {
        // データベースファイルがなければ生成、既にあれば開く
        SQLiteDatabase db =
            this.openOrCreateDatabase(DB_NAME, DB_MODE, null);

        Toast.makeText(this, "db01_01.db が作成されました",
            Toast.LENGTH_SHORT).show();
        // this.deleteDatabase(DB_NAME);//削除の場合

        db.close();//データベースファイルを閉じる
    }
}

```

テーブルの作成

作成したデータベースに対してテーブルを作成します。テーブルの作成やデータの追加など SQL 文を実行するには「SQLiteDatabase」クラスで用意されている「execSQL」メソッドを使います。

※イベント処理の部分のみコードを変更してください。

```
public void onClick(View v) {

    // データベースファイルがなければ生成、既にあれば開く
    SQLiteDatabase db =
        this.openOrCreateDatabase(DB_NAME, DB_MODE, null);

    // テーブル生成の SQL 文
    String sql = "create table shouhin ("
        + "id integer primary key autoincrement, "
        + "name text not null);";

    try {
        db.execSQL(sql); // SQL 文を実行
        // 確認表示
        Toast.makeText(this, "shouhin テーブルが作成されました",
            Toast.LENGTH_SHORT).show();

    } catch (SQLException e) {
        Log.e("ERROR", e.toString());
    }

    db.close(); // データベースファイルを閉じる

} // onClick()
```

データの追加/更新/削除

テーブルに対してデータを追加したり削除するには SQL 文を用意して実行します。SQL 文が異なるだけでテーブルの作成の場合と同じく「SQLiteDatabase」クラスで用意されている「execSQL」メソッドを使います。

※イベント処理の部分のみコードを変更してください。

```
public void onClick(View v) {

    // データベースファイルがなければ生成、既にあれば開く
    SQLiteDatabase db =
        this.openOrCreateDatabase(DB_NAME, DB_MODE, null);

    // レコード追加の SQL 文
    String sql = "insert into shouhin (name) values('PC');";

    //String sqlDelete = "DELETE FROM shouhin WHERE name = 'PC'";//削除の場合

    try {
        db.execSQL(sql);//sqlDelete
        //確認表示
        Toast.makeText(this, "レコードが 1 件、追加されました",
            Toast.LENGTH_SHORT).show();

    } catch (SQLException e) {
        Log.e("ERROR", e.toString());
    }

    db.close();// データベースファイルを閉じる
} // onClick()
```

SQL 文を指定してデータを取得

テーブルからデータを取得するには SQL 文を用意してクエリを発行します。SQL 文に適合するデータがあった場合に、データベースからデータを受け取ります。SQL 文を指定してクエリを発行するに「SQLiteDatabase」クラスで用意されている query メソッドや rawQuery メソッドを使用します。

query() 簡単な検索ならこれが使いやすい。ただし、使いにくい複雑な検索も可能

rawQuery() SQL 文を渡して、データもバインドできるので、複雑な検索や一部条件だけ異なる繰り返しの検索などに便利

例えば、以下のような SQL なら rawQuery の方が断然分かりやすいです。

```
// rawQuery で SELECT を実行
```

```
String sql = "select foo.a, bar.b from foo, bar where foo.a = bar.a;";
```

```
Cursor c = db.rawQuery(sql, null);
```

```
// query で SELECT を実行
```

```
String table = "foo, bar";
```

```
String[] columns = {"foo.a", "bar.b"};
```

```
String selection = "foo.a = bar.a";
```

```
Cursor c = db.query(table, columns, selection, null, null, null, null);
```

■ 検索結果の取り扱い

検索した結果は、**Cursor** という形で取得できます。カーソルの概念は下の図のような感じです。

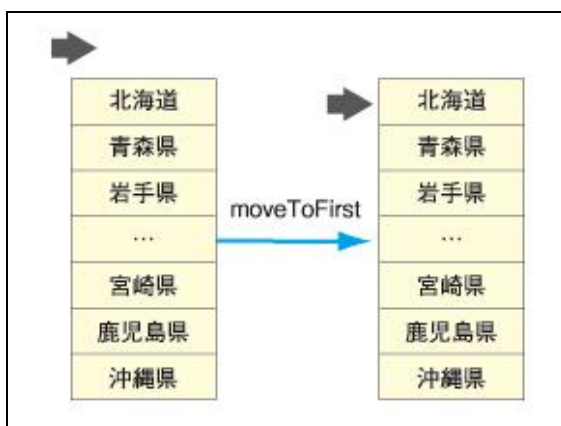


図: カーソルの概念図

結果のリストとそのリストを指すカーソルがあって、カーソルを動かしながら結果を取り出す
というような処理をします。

サンプルコードを修正)

```
package jp.javadrive.android;

import android.app.Activity;
import android.os.Bundle;
import android.widget.LinearLayout;
import android.widget.TextView;
import android.widget.Toast;
import android.util.Log;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.view.View.OnClickListener;
import android.content.Context;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;

public class JavaDriveAndroid extends Activity implements OnClickListener {
    private final int FP = ViewGroup.LayoutParams.FILL_PARENT;
    private final int WC = ViewGroup.LayoutParams.WRAP_CONTENT;

    private Button button;
    // 以下追加
    private TextView textResult;

    private final String DB_NAME = "db01_01.db";
    private final int DB_MODE = Context.MODE_PRIVATE;

    @Override
    protected void onCreate(Bundle icle) {
        super.onCreate(icle);
```



```

        LinearLayout linearLayout = new LinearLayout(this);
        // 以下変更
        linearLayout.setOrientation(LinearLayout.VERTICAL);
        setContentView(linearLayout);

        button = new Button(this);
        // 以下変更
        button.setText("レコード表示");
        button.setOnClickListener(this);
        linearLayout.addView(button, createParam(WC, WC));
        // 以下追加
        textResult = new TextView(this);
        textResult.setText("");
        linearLayout.addView(textResult, createParam(WC, WC));
    }

    private LinearLayout.LayoutParams createParam(int w, int h) {
        return new LinearLayout.LayoutParams(w, h);
    }

    public void onClick(View v) {

        // データベースファイルがなければ生成、既にあれば開く
        SQLiteDatabase db =
            this.openOrCreateDatabase(DB_NAME, DB_MODE, null);
        // 検索 SQL 文
        String sql = "select * from shouhin;";

        try {
            // 検索結果を文字列で詰め込む
            StringBuilder sb = new StringBuilder();
            // 検索結果 Cursor オブジェクトの取得
            Cursor curs = db.rawQuery(sql, null);
            // 1行1行、下に移動しながら、行の各列データを取り出す
            while (curs.moveToNext()) {

```

```

        // 列データを取り出す
        sb.append(curs.getInt(0) + ":" +
            curs.getString(1) + "¥n");
    }

    // 検索結果のレコードをテキストビューに表示
    textResult.setText(new String(sb));
} catch (SQLException e) {
    Log.e("ERROR", e.toString());
}

db.close();// データベースファイルを閉じる
}
}

```

```

public void onClick(View v) {

    // データベースファイルがなければ生成、既にあれば開く
    SQLiteDatabase db =
        this.openOrCreateDatabase(DB_NAME, DB_MODE, null);

    try {
        ContentValues insertValue = new ContentValues();
        insertValue.put("name", "orange");

        // 追加
        db.insert("shouhin", null, insertValue);
        // 確認表示
        textResult.setText("1 件、レコードを追加しました。");
    } catch (SQLException e) {
        Log.e("ERROR", e.toString());
    }

    db.close();// データベースファイルを閉じる
}
}

```

update メソッドによるデータの更新

データ追加の場合と同じくデータ更新のためのメソッドも用意されています。
「**SQLiteDatabase**」クラスで用意されている「**update**」メソッドを使います。

```
update
public int update(String table,
                  ContentValues values,
                  String whereClause,
                  String[] whereArgs)
```

1 番目の引数には対象のテーブル名を指定します。

2 番目の引数には更新するデータの各カラムに対する値を保持している「**ContentValues**」クラスのオブジェクトを指定します。

3 番目の引数には更新の対象となるレコードを特定するための条件を記述します。**SQL** 文で言えば **WHERE** 句の箇所に該等します。例えば「**id=10**」などです。

4 番目の引数もレコード特定のための条件だと思われそうですが、説明が無い為詳細が分かりませんでした。

```
public void onClick(View v) {

    // データベースファイルがなければ生成、既にあれば開く
    SQLiteDatabase db =
        this.openOrCreateDatabase(DB_NAME, DB_MODE, null);

    try {

        //id が 2 の商品名 (name) を banana に変更
        ContentValues cv = new ContentValues();
        cv.put("name", "banana");
        db.update("shouhin", cv, "id = 2", null);
        //db.delete("shouhin ", "id = 3", null);//削除
```

```
        // 確認表示
        textResult.setText("1 件、レコードを更新しました。");
    } catch (SQLException e) {
        Log.e("ERROR", e.toString());
    }

    db.close();// データベースファイルを閉じる
}
```

テーブル結合用サンプル)

#商品テーブル

```
create table goods (code primary key, cateid, name);
```

```
insert into goods values('B001', 1, '吾輩は猫である');
```

```
insert into goods values('D001', 2, 'ゴジラ');
```

#カテゴリーテーブル

```
create table cate (id primary key, name);
```

```
insert into cate values(1, '和書');
```

```
insert into cate values(2, 'DVD');
```