

# Tutoriel sur l'utilisation de base de données SQLite sous Android

## Utiliser la base de données Android SQLite

### Table des matières

- **I. SQLite et Android**
  - I-A. Qu'est-ce que SQLite?
  - I-B. SQLite sous Android
- **II. Prérequis pour ce tutoriel**
- **III. Architecture de SQLite**
  - III-A. Paquetages
  - III-B. Création et mise à jour de la base de données avec SQLiteOpenHelper
  - III-C. La classe SQLiteDatabase
  - III-D. Exemple rawQuery()
  - III-E. Exemple query()
  - III-F. Cursor
  - III-G. ListViews, ListActivities et SimpleCursorAdapter
- **IV. Tutoriel : utiliser SQLite**
  - IV-A. Présentation du projet
  - IV-B. Créer le projet
  - IV-C. La base de données et le modèle de données
  - IV-D. L'interface utilisateur
  - IV-E. Exécuter l'application
- **V. Fournisseur de contenu et partage de données**
  - V-A. Qu'est-ce qu'un fournisseur de contenu ?
  - V-B. Accéder à un fournisseur de contenu
  - V-C. Fournisseur de contenu personnalisé
  - V-D. Sécurité et fournisseur de contenu
  - V-E. Sécurité des threads
- **VI. Tutoriel: Utiliser un fournisseur de contenu**
  - VI-A. Vue d'ensemble

- VI-B. Créer des contacts sur votre émulateur
  - VI-C. Utiliser le fournisseur de contenu des contacts
- **VII. Loader**
  - VII-A. Le but de la classe Loader
  - VII-B. Implémentation d'un chargeur
  - VII-C. La base de données SQLite et CursorLoader
- **VIII. Curseurs et Loaders**
- **IX. Tutoriel: SQLite, ContentProvider customisé et Loader**
  - IX-A. Vue d'ensemble
  - IX-B. Projet
  - IX-C. Classes de la base de données
  - IX-D. Créer le fournisseur de contenu
  - IX-E. Ressources
  - IX-F. Mise en page
  - IX-G. Activités
  - IX-H. Démarrez votre application
- **X. Accéder directement à la base de données SQLite**
  - X-A. L'emplacement de stockage de la base de données SQLite
  - X-B. L'accès à la base de données en ligne de commande
- **XI. Plus sur ListView**
- **XII. Performances**
- **XIII. Liens et littérature**
  - XIII-A. Code source
  - XIII-B. Ressources Android SQLite
- **XIV. Remerciements**
- **Developpez**

La base de données Android SQLite et le fournisseur de contenu - Tutoriel  
Basé sur Android 4.3

Ce tutoriel explique comment utiliser la base de données SQLite pour créer des applications Android. Il montre également comment utiliser les fournisseurs de contenu existants et comment en définir des nouveaux. Il montre aussi l'utilisation du framework Loader, qui permet de charger des données de façon asynchrone.

Le tutoriel est basé sur Eclipse 4.2, Java 1.6 et Android 4.2.

Nous remercions Lars Vogel qui nous a aimablement autorisé à traduire et héberger cet article.

N'hésitez pas à commenter cet article ! 6 commentaires ★★★★★

Article lu 84357 fois.

### Les deux auteur et traducteur

Lars Vogel 

Traducteur : **Mishulyna**

### L'article

Publié le 19 août 2013 - Mis à jour le 1<sup>er</sup> juin 2014

***Version PDF Version hors-ligne***

***ePub, Azw et Mobi***

### Liens sociaux

SQLite est une base de données open source, qui supporte les fonctionnalités standards des bases de données relationnelles comme la syntaxe SQL, les transactions et les prepared statement. La base de données nécessite peu de mémoire lors de l'exécution (env. 250 ko), ce qui en fait un bon candidat pour être intégré dans d'autres environnements d'exécution.

SQLite prend en charge les types de données TEXT (similaire à String en Java), INTEGER (similaire à long en Java) et REAL (similaire à double en Java). Tous les autres types doivent être convertis en l'un de ces types avant d'être enregistrés dans la base de données. SQLite ne vérifie pas si les types des données insérées dans les colonnes correspondent au type défini, par exemple, vous pouvez écrire un nombre entier dans une colonne de type chaîne de caractères et vice versa.

Plus d'informations sur SQLite sont disponibles sur [le site web de SQLite](#).

SQLite est intégrée dans chaque appareil Android. L'utilisation d'une base de données SQLite sous Android ne nécessite pas de configuration ou d'administration de la base de données.

Vous devez uniquement définir les instructions SQL pour créer et mettre à jour la base de données. Ensuite, celle-ci est gérée automatiquement pour vous, par la plate-forme Android.

L'accès à une base de données SQLite implique l'accès au système de fichiers. Cela peut être lent. Par conséquent, il est recommandé d'effectuer les opérations de base de données de manière asynchrone.

Si votre application crée une base de données, celle-ci est par défaut enregistrée dans le répertoire DATA  
/data/APP\_NAME/databases/FILENAME.

Ce chemin de fichier est obtenu sur la base des règles suivantes : DATA est le chemin retourné par la méthode `Environment.getDataDirectory()`, APP\_NAME est le nom de votre application. FILENAME est le nom de la base de données que vous renseignez dans le code de votre application.

Ce qui suit implique que vous avez déjà les bases du développement Android. Veuillez consulter [le tutoriel de développement Android](#) pour en apprendre les bases.

Le paquetage `android.database` contient toutes les classes nécessaires pour travailler avec des bases de données. Le paquetage `android.database.sqlite` contient les classes spécifiques à SQLite.

Pour créer et mettre à jour une base de données dans votre application Android, vous créez une classe qui hérite de `SQLiteOpenHelper`. Dans le constructeur de votre sous-classe, vous appelez la méthode `super()` de `SQLiteOpenHelper`, en précisant le nom de la base de données et sa version actuelle.

Dans cette classe, vous devez redéfinir les méthodes suivantes pour créer et mettre à jour votre base de données.

- `onCreate()` - est appelée par le framework pour accéder à une base de données qui n'est pas encore créée.
- `onUpgrade()` - est appelée si la version de la base de données est augmentée dans le code de votre application. Cette méthode vous permet de

mettre à jour un schéma de base de données existant ou de supprimer la base de données existante et la recréer par la méthode `onCreate()`.

Les deux méthodes reçoivent en paramètre un objet `SQLiteDatabase` qui est la représentation Java de la base de données.

La classe `SQLiteOpenHelper` fournit les méthodes `getReadableDatabase()` et `getWritableDatabase()` pour accéder à un objet `SQLiteDatabase` en lecture, respectif en écriture.

Les tables de base de données doivent utiliser l'identifiant `_id` comme clé primaire de la table. Plusieurs fonctions Android s'appuient sur cette norme.

C'est une bonne pratique de créer une classe par table. Cette classe définit des méthodes statiques `onCreate()` et `onUpgrade()`, qui sont appelées dans les méthodes correspondantes de la superclasse `SQLiteOpenHelper`. De cette façon, votre implémentation de `SQLiteOpenHelper` reste lisible, même si vous avez plusieurs tables

`SQLiteDatabase` est la classe de base pour travailler avec une base de données SQLite sous Android et fournit des méthodes pour ouvrir, effectuer des requêtes, mettre à jour et fermer la base de données.

Plus précisément, `SQLiteDatabase` fournit les méthodes `insert()`, `update()` et `delete()`.

En outre, elle fournit la méthode `execSQL()`, qui permet d'exécuter une instruction SQL directement.

L'objet `ContentValues` permet de définir des clés/valeurs. La clé représente l'identifiant de la colonne de la table et la valeur représente le contenu de l'enregistrement dans cette colonne. `ContentValues` peut être utilisé pour insertions et mises à jour des enregistrements de la base de données.

Des requêtes peuvent être créées via les méthodes `rawQuery()` et `query()`, ou par la classe `SQLiteQueryBuilder`.

`rawQuery()` accepte directement une requête SQL `SELECT` en entrée.

`query()` fournit une interface structurée pour spécifier la requête SQL.

`SQLiteQueryBuilder` est une (convenience class) classe de confort permettant de simplifier la construction des requêtes SQL.

Le code suivant montre un exemple d'appel de la méthode `rawQuery()`.

```
rawQuery()
Sélectionnez
Cursor cursor = getReadableDatabase().
    rawQuery("select * from todo where _id = ?", new String[] { id });
```



Le code suivant montre un exemple d'appel de la méthode `query()`.

```
query()
Sélectionnez
return database.query(DATABASE_TABLE,
    new String[] { KEY_ROWID, KEY_CATEGORY, KEY_SUMMARY,
        KEY_DESCRIPTION }, null, null, null, null, null);
```

La méthode `query()` accepte les paramètres suivants :

Tableau 1 Paramètres de la méthode `query()`

Paramètre	Commentaire
String dbName	Le nom de la table pour laquelle la requête est compilée.
String[] columnNames	Une liste des colonnes à retourner. En passant "null", toutes les colonnes seront retournées.
String whereClause	Clause "where", filtre pour la sélection des données, "null" permet de sélectionner toutes les données.
String[] selectionArgs	Vous pouvez inclure des ? dans la "whereClause". Ces caractères génériques (placeholders) seront remplacés par les valeurs du tableau selectionArgs.
String[] groupBy	Un filtre qui déclare comment regrouper les lignes, avec "null" les lignes ne seront pas groupées.
String[] having	Filtre pour les groupes, null signifie pas de filtrage.

String[] orderBy	Colonnes de la table utilisées pour ordonner les données, avec null les données ne seront pas ordonnées.
------------------	--

Si une condition n'est pas requise, vous pouvez transmettre la valeur null, par exemple, pour une clause group by.

La "whereClause" est spécifiée sans le mot clé "where", par exemple, une clause "where" pourrait ressembler à : " \_id = 19 et résumé =" .

Si vous spécifiez des valeurs génériques dans la clause where par des ?, vous les passez comme paramètres de requête de type selectionArgs.

Une requête retourne un objet Cursor. Un curseur représente le résultat d'une requête et pointe généralement vers une ligne de ce résultat. De cette façon, Android peut gérer les résultats de la requête de manière efficace, car il n'a pas à charger toutes les données en mémoire.

Utilisez la méthode getCount() pour obtenir le nombre d'éléments résultant de la requête.

Pour vous déplacer entre les lignes individuelles de données, vous pouvez utiliser les méthodes MoveToFirst() et MoveToNext(). La méthode isAfterLast() permet de vérifier si la fin du résultat de la requête a été atteinte.

(La classe) Cursor fournit des méthodes get\*() par type de données : getLong(columnIndex), getString(columnIndex), pour accéder aux données d'une colonne de la position courante du résultat. Le "columnIndex" est l'index de la colonne à laquelle vous accédez.

Cursor fournit également la méthode getColumnIndexOrThrow(String) qui permet d'obtenir l'index d'une colonne à partir de son nom passé en paramètre. Un curseur doit être fermé par l'appel à la méthode close().

ListView sont des vues qui permettent d'afficher une liste d'éléments.

ListActivities sont des activités spécialisées qui rendent plus facile l'usage des ListView.

Pour travailler avec des bases de données et ListView vous pouvez utiliser un SimpleCursorAdapter, qui permet de définir une mise en page pour chaque ligne des ListView.

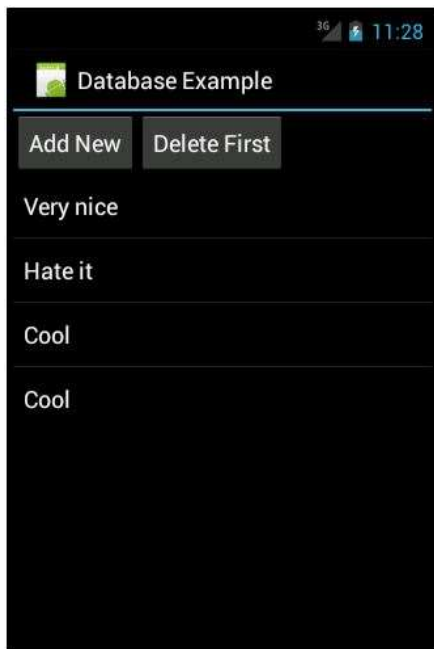
Vous définissez également un tableau qui contient les noms de colonnes et un autre tableau contenant les identifiants des Vues dans lesquelles les données seront affichées.

La classe SimpleCursorAdapter effectuera le mappage des colonnes vers les vues, basé sur le curseur qui lui est passé .

Pour obtenir le curseur, vous devez utiliser la classe Loader.

La partie suivante démontre comment travailler avec une base de données SQLite. Nous allons utiliser un objet d'accès aux données (DAO) qui gérera les données pour nous. Le DAO prend en charge la gestion de la connexion à la base de données, l'accès aux données et leur modification. Il se chargera également de la conversion des objets de la base de données en objets Java, de sorte que le code de notre interface utilisateur n'a pas à s'occuper de la couche de persistance.

L'application résultante se présentera ainsi.



L'utilisation d'un DAO n'est pas toujours une bonne approche. Un DAO crée des modèles objet Java ; l'utilisation d'une base de données directement ou par l'intermédiaire d'un fournisseur de contenu est généralement plus efficace en terme des ressources car vous pouvez éviter la



création des modèles objet.

Je démontre encore l'utilisation du DAO dans cet exemple pour commencer avec un exemple relativement simple. Utilisez la dernière version Android 4.0, API Level 15. Sinon, vous devrez faire usage de la classe Loader, utilisée à partir de Android 3.0 pour la gestion d'un curseur de base de données, et cette classe ajoute une complexité supplémentaire.

Créez le nouveau projet Android, nommez-le de.vogella.android.sqlite.first, puis une activité nommée *TestDatabaseActivity*.

Créez la classe MySQLiteHelper. Cette classe se charge de la création de la base de données. La méthode onUpgrade() va simplement supprimer toutes les données existantes et recréer la table. Elle définit également plusieurs constantes pour le nom et les colonnes de la table.

```

classeMySQLiteHelper
Sélectionnez
package de.vogella.android.sqlite.first;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;

public class MySQLiteHelper extends SQLiteOpenHelper {

    public static final String TABLE_COMMENTS = "comments";
    public static final String COLUMN_ID = "_id";
    public static final String COLUMN_COMMENT = "comment";

    private static final String DATABASE_NAME = "commments.db";
    private static final int DATABASE_VERSION = 1;

    // Commande sql pour la création de la base de données
    private static final String DATABASE_CREATE = "create table "
        + TABLE_COMMENTS + "(" + COLUMN_ID
        + " integer primary key autoincrement, " + COLUMN_COMMENT
        + " text not null);";

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

```

```

}

@Override
public void onCreate(SQLiteDatabase database) {
    database.execSQL(DATABASE_CREATE);
}

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    Log.w(MySQLiteHelper.class.getName(),
        "Upgrading database from version " + oldVersion + " to "
        + newVersion + ", which will destroy all old data");
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_COMMENTS);
    onCreate(db);
}
}

```

Créez la classe Comment. Cette classe est notre modèle et contient les données que nous allons enregistrer dans la base de données, et afficher dans l'interface utilisateur.

```

classeComment
Sélectionnez
package de.vogella.android.sqlite.first;

public class Comment {
    private long id;
    private String comment;

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getComment() {
        return comment;
    }
}

```

```

}

public void setComment(String comment) {
    this.comment = comment;
}

// Sera utilisée par ArrayAdapter dans la ListView
@Override
public String toString() {
    return comment;
}
}

```

Créez la classe `CommentsDataSource`. Cette classe est notre DAO. Elle maintient la connexion avec la base de données et prend en charge l'ajout des nouveaux commentaires et le retour de tous les commentaires.

```

classCommentsDataSource
Sélectionnez
package de.vogella.android.sqlite.first;

import java.util.ArrayList;
import java.util.List;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;

public class CommentsDataSource {

    // Champs de la base de données
    private SQLiteDatabase database;
    private MySQLiteHelper dbHelper;
    private String[] allColumns = { MySQLiteHelper.COLUMN_ID,
        MySQLiteHelper.COLUMN_COMMENT };

    public CommentsDataSource(Context context) {

```

```
dbHelper = new MySQLiteHelper(context);
}

public void open() throws SQLException {
    database = dbHelper.getWritableDatabase();
}

public void close() {
    dbHelper.close();
}

public Comment createComment(String comment) {
    ContentValues values = new ContentValues();
    values.put(MySQLiteHelper.COLUMN_COMMENT, comment);
    long insertId = database.insert(MySQLiteHelper.TABLE_COMMENTS, null,
        values);
    Cursor cursor = database.query(MySQLiteHelper.TABLE_COMMENTS,
        allColumns, MySQLiteHelper.COLUMN_ID + " = " + insertId, null,
        null, null, null);
    cursor.moveToFirst();
    Comment newComment = cursorToComment(cursor);
    cursor.close();
    return newComment;
}

public void deleteComment(Comment comment) {
    long id = comment.getId();
    System.out.println("Comment deleted with id: " + id);
    database.delete(MySQLiteHelper.TABLE_COMMENTS, MySQLiteHelper.COLUMN_ID
        + " = " + id, null);
}

public List<Comment> getAllComments() {
    List<Comment> comments = new ArrayList<Comment>();

    Cursor cursor = database.query(MySQLiteHelper.TABLE_COMMENTS,
        allColumns, null, null, null, null, null);

    cursor.moveToFirst();
```

```

while (!cursor.isAfterLast()) {
    Comment comment = cursorToComment(cursor);
    comments.add(comment);
    cursor.moveToNext();
}
// assurez-vous de la fermeture du curseur
cursor.close();
return comments;
}

private Comment cursorToComment(Cursor cursor) {
    Comment comment = new Comment();
    comment.setId(cursor.getLong(0));
    comment.setComment(cursor.getString(1));
    return comment;
}
}

```

Modifiez votre fichier main.xml dans le dossier res/layout comme dans le code suivant. Cette mise en page a deux boutons pour ajouter et supprimer des commentaires et une ListView qui sera utilisée pour afficher les commentaires existants. Le texte du commentaire sera généré plus tard dans l'activité par un petit générateur aléatoire.

mainXML

Sélectionnez

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <LinearLayout
        android:id="@+id/group"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" >

        <Button
            android:id="@+id/add"
            android:layout_width="wrap_content"

```

```

        android:layout_height="wrap_content"
        android:text="Add New"
        android:onClick="onClick"/>

        <Button
            android:id="@+id/delete"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Delete First"
            android:onClick="onClick"/>

    </LinearLayout>

    <ListView
        android:id="@android:id/list"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello" />

</LinearLayout>

```

Modifiez votre classe TestDatabaseActivity comme ci-dessous. Nous utilisons ici une ListActivity pour afficher les données.

```

classe TestDatabaseActivity
Sélectionnez
package de.vogella.android.sqlite.first;

import java.util.List;
import java.util.Random;

import android.app.ListActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.ArrayAdapter;

public class TestDatabaseActivity extends ListActivity {
    private CommentsDataSource datasource;

    @Override

```

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    datasource = new CommentsDataSource(this);
    datasource.open();

    List<Comment> values = datasource.getAllComments();

    // utilisez SimpleCursorAdapter pour afficher les
    // éléments dans une ListView
    ArrayAdapter<Comment> adapter = new ArrayAdapter<Comment>(this,
        android.R.layout.simple_list_item_1, values);
    setListAdapter(adapter);
}

// Sera appelée par l'attribut onClick
// des boutons déclarés dans main.xml
public void onClick(View view) {
    @SuppressWarnings("unchecked")
    ArrayAdapter<Comment> adapter = (ArrayAdapter<Comment>) getListAdapter();
    Comment comment = null;
    switch (view.getId()) {
        case R.id.add:
            String[] comments = new String[] { "Cool", "Very nice", "Hate it" };
            int nextInt = new Random().nextInt(3);
            // enregistrer le nouveau commentaire dans la base de données
            comment = datasource.createComment(comments[nextInt]);
            adapter.add(comment);
            break;
        case R.id.delete:
            if (getListAdapter().getCount() > 0) {
                comment = (Comment) getListAdapter().getItem(0);
                datasource.deleteComment(comment);
                adapter.remove(comment);
            }
            break;
    }
    adapter.notifyDataSetChanged();
}
```

```
}

@Override
protected void onResume() {
    datasource.open();
    super.onResume();
}

@Override
protected void onPause() {
    datasource.close();
    super.onPause();
}
}
```

Installez votre application et utilisez les boutons *Add* et *Delete*. Redémarrez votre application pour vérifier que les données sont toujours là.

Une base de données SQLite est propre à l'application qui la crée. Si vous voulez partager des données avec d'autres applications, vous pouvez utiliser un fournisseur de contenu.

Un fournisseur de contenu permet aux applications d'accéder aux données. Dans la plupart des cas, ces données sont stockées dans une base de données SQLite.

Même si un fournisseur de contenu peut être utilisé dans une application pour accéder aux données, il est généralement utilisé pour partager des données avec d'autres applications. Comme les données de l'application sont privées, par défaut, un fournisseur de contenu est un moyen pratique pour partager vos données avec d'autres applications basées sur une interface structurée.

Un fournisseur de contenu doit être déclaré dans le fichier `AndroidManifest.xml`.

L'accès à un fournisseur de contenu s'effectue par un URI. La base de l'URI est définie dans la déclaration du fournisseur de contenu, dans le fichier `AndroidManifest.xml`, via l'attribut `android:authorities`.

Comme il est nécessaire de connaître les URI d'un fournisseurs de contenu pour y accéder, c'est une bonne pratique que de fournir des constantes publiques pour les URI afin de les documenter pour d'autres développeurs.



Beaucoup de sources de données Android, par exemple les contacts, sont accessibles via les fournisseurs de contenu.

Pour créer votre propre fournisseur de contenu vous devez définir une classe qui hérite de `android.content.ContentProvider`. Vous déclarez également votre fournisseur de contenu dans le fichier `AndroidManifest.xml`. Cette entrée doit spécifier l'attribut `android:authorities` qui permet d'identifier le fournisseur de contenu. Cette autorité est la base de l'URI pour accéder aux données et doit être unique.

```
ownContentProvider
Sélectionnez
<provider
    android:authorities="de.vogella.android.todos.contentprovider"
    android:name=".contentprovider.MyTodoContentProvider" >
</provider>
```

Votre fournisseur de contenu doit implémenter plusieurs méthodes, par exemple `query()`, `insert()`, `update()`, `delete()`, `getType()` et `onCreate()`. Au cas où vous n'implémentez pas certaines méthodes, c'est une bonne pratique de lancer une `UnsupportedOperationException`.

La méthode `query()` doit retourner un objet `Cursor`.

Jusqu'à la version Android 4.2 un fournisseur de contenu est disponible par défaut pour autres applications Android. À partir de la version Android 4.2 un fournisseur de contenu doit être explicitement exporté.

Pour définir la visibilité de votre fournisseur de contenu utilisez le paramètre **`android:exported = false | true`** lors de sa déclaration dans le fichier `AndroidManifest.xml`.

C'est une bonne pratique d'utiliser toujours le paramètre `android:exported` pour assurer un comportement correct à travers les versions Android.

Si vous travaillez directement avec la base de données et avez plusieurs accès en écriture par différents threads, vous pouvez rencontrer des problèmes de concurrence.

Le fournisseur de contenu peut être accessible à partir de plusieurs programmes en même temps, vous devez donc mettre en œuvre l'accès thread-safe. La façon la plus simple, c'est d'utiliser le mot-clé `synchronised` devant toutes les méthodes du fournisseur de contenu, de sorte qu'un seul fil d'exécution puisse accéder à ces méthodes à un moment donné.

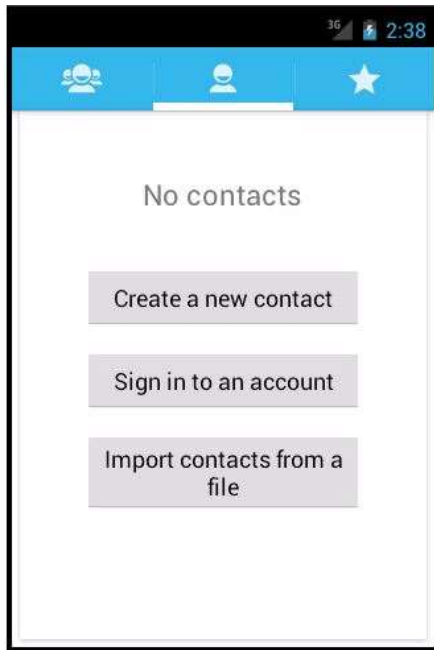
Si vous ne souhaitez pas qu'Android synchronise l'accès aux données du fournisseur de contenu, utilisez l'attribut **android:multitprocess = true** dans la définition de votre <provider> dans le fichier AndroidManifest.xml. Ceci permet qu'une instance du fournisseur soit créée dans chaque processus client, ce qui élimine la nécessité d'effectuer une communication interprocessus (IPC).

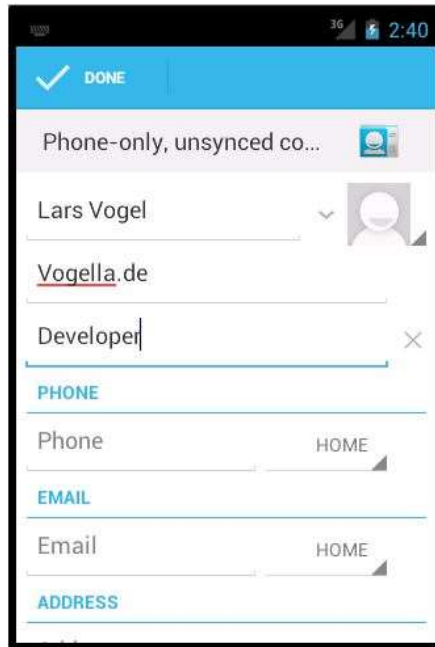
L'exemple suivant utilisera un fournisseur de contenu existant de l'application People.

Pour cet exemple, nous avons besoin de quelques contacts. Sélectionnez le menu d'accueil, puis l'élément du menu People pour créer des contacts.



L'application vous demandera si vous voulez vous connecter. Identifiez-vous ou sélectionnez "Pas maintenant". Appuyez sur "Créer un nouveau contact". Vous pouvez créer des contacts locaux.





Finalisez l'ajout de votre premier contact. Par la suite, l'application vous permet d'ajouter plus de contacts via le bouton +. En conséquence, vous devriez avoir quelques nouveaux contacts dans votre application.

Créez un nouveau projet Android appelé `de.vogella.android.contentprovider` avec l'activité nommée `ContactsActivity`.

Modifiez le fichier de mise en page correspondant dans le dossier `res/layout`. Renommez l'ID de la `TextView` existante `contactview`. Supprimez le texte par défaut.

Le fichier de mise en page résultant devrait ressembler à ce qui suit.

peopleLayout

Sélectionnez

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```

        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical" >

        <TextView
            android:id="@+id/contactview"
            android:layout_width="match_parent"
            android:layout_height="match_parent" />

    </LinearLayout>

```

L'accès au fournisseur de contenu des contacts nécessite une certaine autorisation, toutes les applications ne devraient pas avoir accès à l'information des contacts. Ouvrez le fichier AndroidManifest.xml et sélectionnez l'onglet "Autorisations". Dans cet onglet cliquez sur le bouton "Ajouter" et sélectionnez Uses Permission. Dans la liste déroulante, sélectionnez l'entrée android.permission.READ\_CONTACTS.

Modifiez le code de l'activité.

```

classeContactsActivity
Sélectionnez
package de.vogella.android.contentprovider;

import android.app.Activity;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.provider.ContactsContract;
import android.widget.TextView;

public class ContactsActivity extends Activity {

    /** Appelée quand l'activité est créée pour la première fois */

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_contacts);
        TextView contactView = (TextView) findViewById(R.id.contactview);
    }
}

```

```

Cursor cursor = getContacts();

while (cursor.moveToNext()) {

    String displayName = cursor.getString(cursor
        .getColumnIndex(ContactsContract.Data.DISPLAY_NAME));
    contactView.append("Name: ");
    contactView.append(displayName);
    contactView.append("\n");
}

}

private Cursor getContacts() {
    // Exécuter la requête
    Uri uri = ContactsContract.Contacts.CONTENT_URI;
    String[] projection = new String[] { ContactsContract.Contacts._ID,
        ContactsContract.Contacts.DISPLAY_NAME };
    String selection = ContactsContract.Contacts.IN_VISIBLE_GROUP + " = '1'"
        + ("1") + "'";
    String[] selectionArgs = null;
    String sortOrder = ContactsContract.Contacts.DISPLAY_NAME
        + " COLLATE LOCALIZED ASC";

    return managedQuery(uri, projection, selection, selectionArgs,
        sortOrder);
}
}

```

Si vous exécutez cette application, les données sont lues à partir du fournisseur de contenu de l'application People et affichées dans une TextView. Typiquement vous afficheriez ces données dans une ListView.

La classe Loader permet de charger des données de manière asynchrone dans une activité ou un fragment. Les chargeurs peuvent contrôler la source de données et fournir des nouveaux résultats quand son contenu change. Ils persistent également des données entre les changements de configuration.

Si le résultat est récupéré par le chargeur après que l'objet ait été déconnecté de son parent (activité ou fragment), il peut mettre les données en cache.

Les chargeurs ont été introduits dans Android 3.0 et font partie de la couche de compatibilité pour des versions Android, comme 1,6.

Vous pouvez utiliser la classe abstraite `AsyncTaskLoader` comme superclasse de vos propres implémentations de chargeurs.

Le `LoaderManager` d'une activité ou d'un fragment gère une ou plusieurs instances de chargeur. La création d'un chargeur se fait via l'appel de méthode suivant.

```
appelLoader
Sélectionnez
# start a new loader or re-connect to existing one
getLoaderManager().initLoader(0, null, this);
```

Le premier paramètre est un identifiant unique, qui peut être utilisé par la classe de rappel (callback), pour identifier ce chargeur plus tard. Le deuxième paramètre est un bundle qui peut être donné à la classe de rappel pour plus d'informations.

Le troisième paramètre de `initLoader()` est la classe qui est appelée une fois l'initialisation lancée (classe de rappel). Cette classe doit implémenter l'interface `LoaderManager.LoaderCallbacks`. C'est une bonne pratique qu'une activité ou un fragment qui utilise `Loader`, implémente l'interface `LoaderManager.LoaderCallbacks`.

Le chargeur n'est pas directement créé par l'appel à la méthode `getLoaderManager().initLoader()`, mais doit être créé par la classe de rappel dans la méthode `onCreateLoader()`.

Une fois que le chargeur a terminé la lecture de données de manière asynchrone, la méthode `onLoadFinished()` de la classe de rappel est appelée. Ici vous pouvez actualiser votre interface utilisateur.

Android fournit une implémentation de chargeur par défaut pour gérer les connexions à la base de données SQLite, la classe `CursorLoader`.

Pour un fournisseur de contenu à partir d'une base de données SQLite vous pouvez généralement utiliser un chargeur de curseur. Celui-ci exécute la requête dans la base de données dans un fil d'exécution et en arrière-plan, de sorte que l'application n'est pas bloquée.

La classe `CursorLoader` est le remplacement pour les curseurs d'activité gérés (Activity-managed cursors) qui sont maintenant obsolètes.

Si le curseur n'est plus valide, la méthode `onLoaderReset()` est appelée sur la classe de rappel.

L'un des défis des accès aux bases de données est que cet accès est lent. L'autre défi est que l'application doit tenir compte correctement du cycle de vie des composants, par exemple, l'ouverture et la fermeture du curseur si un changement de configuration se produit.

Pour gérer le cycle de vie, vous pouviez utiliser la méthode `ManagedQuery()` dans des activités avant la version Android 3.0.

Comme depuis Android 3.0 cette méthode est obsolète vous devriez utiliser le framework `Loader` pour accéder au fournisseur de contenu.

La classe `SimpleCursorAdapter`, qui peut être utilisée avec les `ListViews`, possède la méthode `swapCursor()`. Votre chargeur peut utiliser cette méthode pour mettre à jour le curseur dans sa méthode `onLoadFinished()`.

La classe `CursorLoader` reconnecte le curseur après un changement de configuration.

La démo suivante est également disponible dans Android Market. Pour permettre à plusieurs utilisateurs de jouer avec l'application, il a été rendu compatible pour Android 2.3. Si vous avez un scanner de codes à barres installé sur votre téléphone Android, vous pouvez scanner le code QR ci-dessous, pour aller à l'exemple de l'application sur le marché Android. Veuillez prendre note que l'application a un aspect et un comportement différent par rapport aux différentes versions Android, par exemple, vous avez un `OptionsMenu` à la place de l'`ActionBar` et le thème est différent.





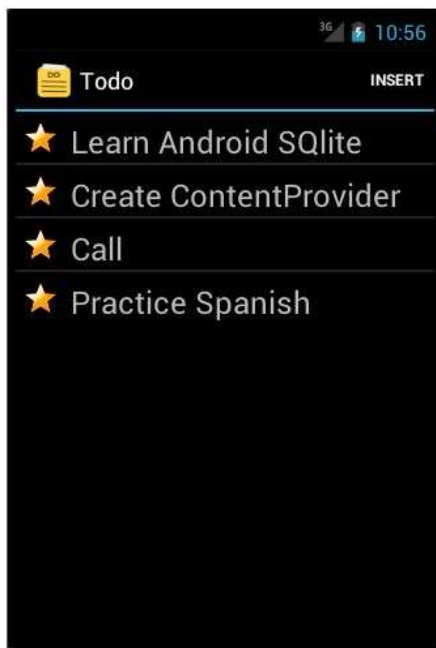
Nous allons créer une application "To-do" qui permet à l'utilisateur d'entrer des tâches pour lui-même. Ces éléments seront stockés dans la base de données SQLite et accessibles via un fournisseur de contenu.

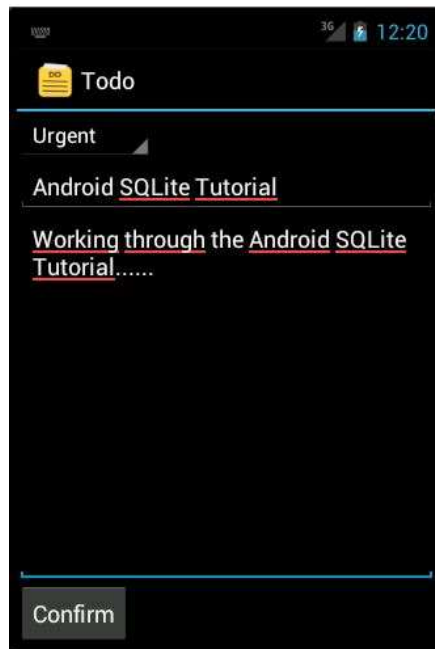
Les tâches sont appelés "éléments todo" ou "todos" dans ce tutoriel.

L'application se compose de deux activités, une pour voir une liste de tous les éléments todo et l'autre pour la création et la modification d'un élément todo spécifique. Ces deux activités communiqueront par des intentions.

Pour charger et gérer de manière asynchrone le curseur, l'activité principale utilisera un chargeur.

L'application résultante ressemblera à ceci.





Créez le projet de.vogella.android.todos avec l'activité appelée TodosOverviewActivity, puis créez une autre activité appelée TodoDetailActivity.

Créez le paquetage de.vogella.android.todos.database. Ce paquetage contiendra les classes pour la manipulation de la base de données.

Comme dit précédemment, je considère comme la meilleure pratique, d'avoir une classe séparée par table. Même si nous avons une seule table dans cet exemple, nous allons suivre cette pratique. De cette façon, nous sommes prêts, dans le cas où notre schéma de base de données augmente.

Créez la classe suivante, qui contient également des constantes pour le nom de la table et les colonnes.

```
classeToDoTable  
Sélectionnez  
package de.vogella.android.todos.database;
```

```
import android.database.sqlite.SQLiteDatabase;
import android.util.Log;

public class TodoTable {

    // Table de la base de données
    public static final String TABLE_TODO = "todo";
    public static final String COLUMN_ID = "_id";
    public static final String COLUMN_CATEGORY = "category";
    public static final String COLUMN_SUMMARY = "summary";
    public static final String COLUMN_DESCRIPTION = "description";

    // Instruction SQL de création de la base de données
    private static final String DATABASE_CREATE = "create table "
        + TABLE_TODO
        + "("
        + COLUMN_ID + " integer primary key autoincrement, "
        + COLUMN_CATEGORY + " text not null, "
        + COLUMN_SUMMARY + " text not null, "
        + COLUMN_DESCRIPTION
        + " text not null"
        + ");";

    public static void onCreate(SQLiteDatabase database) {
        database.execSQL(DATABASE_CREATE);
    }

    public static void onUpgrade(SQLiteDatabase database, int oldVersion,
        int newVersion) {
        Log.w(TodoTable.class.getName(), "Upgrading database from version "
            + oldVersion + " to " + newVersion
            + ", which will destroy all old data");
        database.execSQL("DROP TABLE IF EXISTS " + TABLE_TODO);
        onCreate(database);
    }
}
```

Créez la classe `TodoDatabaseHelper` suivante, qui étend `SQLiteOpenHelper` et appelle les méthodes statiques de la classe utilitaire `TodoTable`.

```

classeTodoDatabaseHelper
Sélectionnez
package de.vogella.android.todos.database;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class TodoDatabaseHelper extends SQLiteOpenHelper {

    private static final String DATABASE_NAME = "todotable.db";
    private static final int DATABASE_VERSION = 1;

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

    // Méthode appelée pendant la création de la base de données
    @Override
    public void onCreate(SQLiteDatabase database) {
        TodoTable.onCreate(database);
    }

    // Méthode appelée pendant une mise à jour de la base de
    // données, par exemple vous augmentez sa version
    @Override
    public void onUpgrade(SQLiteDatabase database, int oldVersion,
        int newVersion) {
        TodoTable.onUpgrade(database, oldVersion, newVersion);
    }
}

```

Nous allons utiliser un fournisseur de contenu pour accéder à la base de données ; nous n'allons pas écrire un objet d'accès aux données (DAO) comme nous l'avons fait dans l'exemple SQLite précédent.

Créez le package `de.vogella.android.todos.contentprovider`.

Créez la classe `MyTodoContentProvider` suivante qui étend `ContentProvider`.

classeMyTodoContentProvider

Sélectionnez

```
package de.vogella.android.todos.contentprovider;
```

```
import java.util.Arrays;
```

```
import java.util.HashSet;
```

```
import android.content.ContentProvider;
```

```
import android.content.ContentResolver;
```

```
import android.content.ContentValues;
```

```
import android.content.UriMatcher;
```

```
import android.database.Cursor;
```

```
import android.database.sqlite.SQLiteDatabase;
```

```
import android.database.sqlite.SQLiteQueryBuilder;
```

```
import android.net.Uri;
```

```
import android.text.TextUtils;
```

```
import de.vogella.android.todos.database.TODODatabaseHelper;
```

```
import de.vogella.android.todos.database.TODOTable;
```

```
public class MyTodoContentProvider extends ContentProvider {
```

```
    // base de données
```

```
    private TODODatabaseHelper database;
```

```
    // utilisées pour UriMatcher
```

```
    private static final int TODOS = 10;
```

```
    private static final int TODO_ID = 20;
```

```
    private static final String AUTHORITY = "de.vogella.android.todos.contentprovider";
```

```
    private static final String BASE_PATH = "todos";
```

```
    public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY  
        + "/" + BASE_PATH);
```

```
    public static final String CONTENT_TYPE = ContentResolver.CURSOR_DIR_BASE_TYPE  
        + "/todos";
```

```
    public static final String CONTENT_ITEM_TYPE = ContentResolver.CURSOR_ITEM_BASE_TYPE  
        + "/todo";
```

```
private static final UriMatcher sURIMatcher = new UriMatcher(UriMatcher.NO_MATCH);
static {
    sURIMatcher.addURI(AUTHORITY, BASE_PATH, TODOS);
    sURIMatcher.addURI(AUTHORITY, BASE_PATH + "/#", TODO_ID);
}

@Override
public boolean onCreate() {
    database = new TodoDatabaseHelper(getContext());
    return false;
}

@Override
public Cursor query(Uri uri, String[] projection, String selection,
    String[] selectionArgs, String sortOrder) {

    // Utiliser SQLiteQueryBuilder à la place de la méthode query()
    SQLiteQueryBuilder queryBuilder = new SQLiteQueryBuilder();

    // vérifier si l'appelant a demandé une colonne qui n'existe pas
    checkColumns(projection);

    // Préciser la table
    queryBuilder.setTables(TodoTable.TABLE_TODO);

    int uriType = sURIMatcher.match(uri);
    switch (uriType) {
        case TODOS:
            break;
        case TODO_ID:
            // ajouter l'ID à la requête d'origine
            queryBuilder.appendWhere(TodoTable.COLUMN_ID + "="
                + uri.getLastPathSegment());
            break;
        default:
            throw new IllegalArgumentException("Unknown URI: " + uri);
    }

    SQLiteDatabase db = database.getWritableDatabase();
```

```
Cursor cursor = queryBuilder.query(db, projection, selection,
    selectionArgs, null, null, sortOrder);
// assurez-vous que les écouteurs potentiels seront notifiés
cursor.setNotificationUri(getContext().getContentResolver(), uri);

return cursor;
}

@Override
public String getType(Uri uri) {
    return null;
}

@Override
public Uri insert(Uri uri, ContentValues values) {
    int uriType = sURIMatcher.match(uri);
    SQLiteDatabase sqlDB = database.getWritableDatabase();
    int rowsDeleted = 0;
    long id = 0;
    switch (uriType) {
        case TODOS:
            id = sqlDB.insert(TodoTable.TABLE_TODO, null, values);
            break;
        default:
            throw new IllegalArgumentException("Unknown URI: " + uri);
    }
    getContext().getContentResolver().notifyChange(uri, null);
    return Uri.parse(BASE_PATH + "/" + id);
}

@Override
public int delete(Uri uri, String selection, String[] selectionArgs) {
    int uriType = sURIMatcher.match(uri);
    SQLiteDatabase sqlDB = database.getWritableDatabase();
    int rowsDeleted = 0;
    switch (uriType) {
        case TODOS:
            rowsDeleted = sqlDB.delete(TodoTable.TABLE_TODO, selection,
                selectionArgs);
    }
}
```

```

        break;
    case TODO_ID:
        String id = uri.getLastPathSegment();
        if (TextUtils.isEmpty(selection)) {
            rowsDeleted = sqlDB.delete(TodoTable.TABLE_TODO,
                TodoTable.COLUMN_ID + "=" + id,
                null);
        } else {
            rowsDeleted = sqlDB.delete(TodoTable.TABLE_TODO,
                TodoTable.COLUMN_ID + "=" + id
                + " and " + selection,
                selectionArgs);
        }
        break;
    default:
        throw new IllegalArgumentException("Unknown URI: " + uri);
    }
    getContext().getContentResolver().notifyChange(uri, null);
    return rowsDeleted;
}

@Override
public int update(Uri uri, ContentValues values, String selection,
    String[] selectionArgs) {

    int uriType = sURIMatcher.match(uri);
    SQLiteDatabase sqlDB = database.getWritableDatabase();
    int rowsUpdated = 0;
    switch (uriType) {
        case TODOS:
            rowsUpdated = sqlDB.update(TodoTable.TABLE_TODO,
                values,
                selection,
                selectionArgs);
            break;
        case TODO_ID:
            String id = uri.getLastPathSegment();
            if (TextUtils.isEmpty(selection)) {
                rowsUpdated = sqlDB.update(TodoTable.TABLE_TODO,

```



```

        values,
        TodoTable.COLUMN_ID + "=" + id,
        null);
    } else {
        rowsUpdated = sqlDB.update(TodoTable.TABLE_TODO,
            values,
            TodoTable.COLUMN_ID + "=" + id
            + " and "
            + selection,
            selectionArgs);
    }
    break;
default:
    throw new IllegalArgumentException("Unknown URI: " + uri);
}
getContext().getContentResolver().notifyChange(uri, null);
return rowsUpdated;
}

private void checkColumns(String[] projection) {
    String[] available = { TodoTable.COLUMN_CATEGORY,
        TodoTable.COLUMN_SUMMARY, TodoTable.COLUMN_DESCRIPTION,
        TodoTable.COLUMN_ID };
    if (projection != null) {
        HashSet<String> requestedColumns = new HashSet<String>(Arrays.asList(projection));
        HashSet<String> availableColumns = new HashSet<String>(Arrays.asList(available));
        // vérifier si toutes les colonnes demandées sont disponibles
        if (!availableColumns.containsAll(requestedColumns)) {
            throw new IllegalArgumentException("Unknown columns in projection");
        }
    }
}
}
}
}

```

MyTodoContentProvider met en œuvre les méthodes `update()`, `insert()`, `delete()` et `query()`. Ces méthodes effectuent plus ou moins directement le mappage vers l'interface `SQLiteDatabase`.

Il dispose également de la méthode `checkColumns()` pour valider qu'une requête ne demande que des colonnes valides.

Enregistrez votre fournisseur de contenu dans votre fichier AndroidManifest.xml.

```
androidManifestEntry
Sélectionnez
<application
    <!-- Place the following after the Activity
        Definition
    -->
    <provider
        android:name=".contentprovider.MyTodoContentProvider"
        android:authorities="de.vogella.android.todos.contentprovider" >
    </provider>
</application>
```

Notre application nécessite plusieurs ressources. D'abord définissez un menu listmenu.xml dans le dossier res/menu. Si vous utilisez l'assistant de ressources Android pour créer le fichier "listmenu.xml", le dossier sera créé pour vous ; si vous créez le fichier manuellement, vous devez également créer le dossier.

Ce fichier XML sera utilisé pour définir le menu d'options dans notre application. L'attribut android:showAsAction = "always" veillera à ce que cette entrée du menu soit affichée dans l'ActionBar de notre application.

```
listmenuXML
Sélectionnez
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">

    <item
        android:id="@+id/insert"
        android:showAsAction="always"
        android:title="Insert">
    </item>

</menu>
```

L'utilisateur sera en mesure de choisir la priorité pour les éléments de todo. Pour les priorités nous créons un tableau de chaînes de caractères. Créez le fichier `priority.xml` suivant dans le dossier `res/values`.

```
priorityXML
Sélectionnez
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string-array name="priorities">
        <item>Urgent</item>
        <item>Reminder</item>
    </string-array>

</resources>
```

Définissez également des chaînes de caractères supplémentaires pour l'application. Modifiez le fichier `strings.xml` sous `res/valeurs`.

```
stringsXML
Sélectionnez
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, Todo!</string>
    <string name="app_name">Todo</string>
    <string name="no_todos">Currently there are no Todo items maintained</string>
    <string name="menu_insert">Add Item</string>
    <string name="menu_delete">Delete Todo</string>
    <string name="todo_summary">Summary</string>
    <string name="todo_description">Delete Todo</string>
    <string name="todo_edit_summary">Summary</string>
    <string name="todo_edit_description">Description</string>
    <string name="todo_edit_confirm">Confirm</string>
</resources>
```

Nous allons définir trois mises en page. L'une servira pour l'affichage d'une ligne de la liste, les autres seront utilisées par nos activités.

La mise en page de la ligne fait référence à une icône appelée `reminder`. Collez une icône de type "png" appelée `reminder.png` dans vos fichiers `res/drawable` (`drawable-hdpi`, `drawable-MDPI`, `drawable-LDPI`).

Si vous n'avez pas une icône disponible, vous pouvez copier l'icône créée par l'assistant Android (ic\_launcher.png dans les dossiers res/drawable\*) ou renommer la référence dans le fichier de mise en page. Veuillez prendre note que les Outils de Développement Android changent parfois le nom de cette icône générée, alors votre fichier peut ne pas s'appeler "ic\_launcher.png".

Sinon, vous pouvez supprimer la définition de l'icône du fichier de mise en page "todo\_row.xml" que vous allez créer à l'étape suivante.

Créez le fichier de mise en page "todo\_row.xml" dans le dossier res/layout.

todo\_rowXML

Sélectionnez

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" >

    <ImageView
        android:id="@+id/icon"
        android:layout_width="30dp"
        android:layout_height="24dp"
        android:layout_marginLeft="4dp"
        android:layout_marginRight="8dp"
        android:layout_marginTop="8dp"
        android:src="@drawable/reminder" >
    </ImageView>

    <TextView
        android:id="@+id/label"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="6dp"
        android:lines="1"
        android:text="@+id/TextView01"
        android:textSize="24dp" >
    </TextView>

</LinearLayout>
```

Créez le fichier de mise en page `todo_list.xml`. Celui-ci définit la façon dont la liste est affichée.

`todo_listXML`

Sélectionnez

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <ListView
        android:id="@android:id/list"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >
    </ListView>

    <TextView
        android:id="@android:id/empty"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/no_todos" />

</LinearLayout>
```

Créez le fichier de mise en page `todo_edit.xml`. Cette disposition sera utilisée pour afficher et modifier un élément de todo dans l'activité `TodoDetailActivity`.

`todo_editXML`

Sélectionnez

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <Spinner
        android:id="@+id/category"
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:entries="@array/priorities" >
    </Spinner>

    <LinearLayout
        android:id="@+id/LinearLayout01"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >

        <EditText
            android:id="@+id/todo_edit_summary"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:hint="@string/todo_edit_summary"
            android:imeOptions="actionNext" >
        </EditText>
    </LinearLayout>

    <EditText
        android:id="@+id/todo_edit_description"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:gravity="top"
        android:hint="@string/todo_edit_description"
        android:imeOptions="actionNext" >
    </EditText>

    <Button
        android:id="@+id/todo_edit_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/todo_edit_confirm" >
    </Button>

</LinearLayout>
```

Modifiez le code de vos activités comme suit. D'abord TodosOverviewActivity.java.

```

classeTodosOverviewActivity
Sélectionnez
package de.vogella.android.todos;

import android.app.ListActivity;
import android.app.LoaderManager;
import android.content.CursorLoader;
import android.content.Intent;
import android.content.Loader;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.view.ContextMenu;
import android.view.ContextMenu.ContextMenuInfo;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView.AdapterContextMenuInfo;
import android.widget.ListView;
import android.widget.SimpleCursorAdapter;
import de.vogella.android.todos.contentprovider.MyTodoContentProvider;
import de.vogella.android.todos.database.TODOTable;

/*
 * TodosOverviewActivity affiche les éléments todo existants
 * dans une liste
 *
 * Vous pouvez en créer des nouveaux via l'option "Insert" de la
 * ActionBar
 * Vous pouvez en effacer un via un appui prolongé sur l'élément
 */

public class TodosOverviewActivity extends ListActivity implements
    LoaderManager.LoaderCallbacks<Cursor> {
    private static final int ACTIVITY_CREATE = 0;
    private static final int ACTIVITY_EDIT = 1;
    private static final int DELETE_ID = Menu.FIRST + 1;
    // private Cursor cursor;

```

```
private SimpleCursorAdapter adapter;

/** Appelée quand l'activité est créée pour la première fois. */

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.todo_list);
    this.getListView().setDividerHeight(2);
    fillData();
    registerForContextMenu(getListView());
}

// crée le menu à base de la définition XML
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.listmenu, menu);
    return true;
}

// Réaction à la sélection dans le menu
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.insert:
            createTodo();
            return true;
    }
    return super.onOptionsItemSelected(item);
}

@Override
public boolean onContextItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case DELETE_ID:
            AdapterContextMenuInfo info = (AdapterContextMenuInfo) item
                .getMenuInfo();
```



```
        Uri uri = Uri.parse(MyTodoContentProvider.CONTENT_URI + "/"
            + info.id);
        getResolver().delete(uri, null, null);
        fillData();
        return true;
    }
    return super.onContextItemSelected(item);
}

private void createTodo() {
    Intent i = new Intent(this, TodoDetailActivity.class);
    startActivity(i);
}

// Ouvre la deuxième activité si un élément est cliqué
@Override
protected void onItemClick(ListView l, View v, int position, long id) {
    super.onItemClick(l, v, position, id);
    Intent i = new Intent(this, TodoDetailActivity.class);
    Uri todoUri = Uri.parse(MyTodoContentProvider.CONTENT_URI + "/" + id);
    i.putExtra(MyTodoContentProvider.CONTENT_ITEM_TYPE, todoUri);

    startActivity(i);
}

private void fillData() {
    // Champs de la (projection) de la base de données
    // Doit inclure le _id colonne pour que l'adapter fonctionne
    String[] from = new String[] { TodoTable.COLUMN_SUMMARY };
    // Champs dans l'UI vers laquelle le mappage est effectué
    int[] to = new int[] { R.id.label };

    getLoaderManager().initLoader(0, null, this);
    adapter = new SimpleCursorAdapter(this, R.layout.todo_row, null,
        from, to, 0);

    setListAdapter(adapter);
}
```

```

}

@Override
public void onCreateContextMenu(ContextMenu menu, View v,
    ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    menu.add(0, DELETE_ID, 0, R.string.menu_delete);
}

// crée un nouveau chargeur après l'appel à initLoader()
@Override
public Loader<Cursor> onCreateLoader(int id, Bundle args) {
    String[] projection = { TodoTable.COLUMN_ID, TodoTable.COLUMN_SUMMARY };
    CursorLoader cursorLoader = new CursorLoader(this,
        MyTodoContentProvider.CONTENT_URI, projection, null, null, null);
    return cursorLoader;
}

@Override
public void onLoadFinished(Loader<Cursor> loader, Cursor data) {
    adapter.swapCursor(data);
}

@Override
public void onLoaderReset(Loader<Cursor> loader) {
    // les données ne sont plus disponibles, supprimer la référence
    adapter.swapCursor(null);
}
}

```

Et TodoDetailActivity.java.

```

classeTodoDetailActivity
Sélectionnez
package de.vogella.android.todos;

import android.app.Activity;
import android.content.ContentValues;

```

```
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.text.TextUtils;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Spinner;
import android.widget.Toast;
import de.vogella.android.todos.contentprovider.MyTodoContentProvider;
import de.vogella.android.todos.database.TODOTable;

/*
 * TodoDetailActivity permet de saisir un nouveau a élément
 * todo ou d'en modifier un
 */
public class TodoDetailActivity extends Activity {
    private Spinner mCategory;
    private EditText mTitleText;
    private EditText mBodyText;

    private Uri todoUri;

    @Override
    protected void onCreate(Bundle bundle) {
        super.onCreate(bundle);
        setContentView(R.layout.todo_edit);

        mCategory = (Spinner) findViewById(R.id.category);
        mTitleText = (EditText) findViewById(R.id.todo_edit_summary);
        mBodyText = (EditText) findViewById(R.id.todo_edit_description);
        Button confirmButton = (Button) findViewById(R.id.todo_edit_button);

        Bundle extras = getIntent().getExtras();

        // vérification de l'Instance sauvegardée
        todoUri = (bundle == null) ? null : (Uri) bundle
            .getParcelable(MyTodoContentProvider.CONTENT_ITEM_TYPE);
    }
}
```

```
// Ou passée par l'autre activité
if (extras != null) {
    todoUri = extras
        .getParcelable(MyTodoContentProvider.CONTENT_ITEM_TYPE);

    fillData(todoUri);
}

confirmButton.setOnClickListener(new View.OnClickListener() {
    public void onClick(View view) {
        if (TextUtils.isEmpty(mTitleText.getText().toString())) {
            makeToast();
        } else {
            setResult(RESULT_OK);
            finish();
        }
    }
});
}

private void fillData(Uri uri) {
    String[] projection = { TodoTable.COLUMN_SUMMARY,
        TodoTable.COLUMN_DESCRIPTION, TodoTable.COLUMN_CATEGORY };
    Cursor cursor = getContentResolver().query(uri, projection, null, null,
        null);
    if (cursor != null) {
        cursor.moveToFirst();
        String category = cursor.getString(cursor
            .getColumnIndexOrThrow(TodoTable.COLUMN_CATEGORY));

        for (int i = 0; i < mCategory.getCount(); i++) {

            String s = (String) mCategory.getItemAtPosition(i);
            if (s.equalsIgnoreCase(category)) {
                mCategory.setSelection(i);
            }
        }
    }
}
```

```
mTitleText.setText(cursor.getString(cursor
    .getColumnIndexOrThrow(TodoTable.COLUMN_SUMMARY)));
mBodyText.setText(cursor.getString(cursor
    .getColumnIndexOrThrow(TodoTable.COLUMN_DESCRIPTION)));

// toujours fermer le curseur
cursor.close();
}
}

protected void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    saveState();
    outState.putParcelable(MyTodoContentProvider.CONTENT_ITEM_TYPE, todoUri);
}

@Override
protected void onPause() {
    super.onPause();
    saveState();
}

private void saveState() {
    String category = (String) mCategory.getSelectedItem();
    String summary = mTitleText.getText().toString();
    String description = mBodyText.getText().toString();

    // sauvegarder uniquement si sommaire ou description disponible

    if (description.length() == 0 && summary.length() == 0) {
        return;
    }

    ContentValues values = new ContentValues();
    values.put(TodoTable.COLUMN_CATEGORY, category);
    values.put(TodoTable.COLUMN_SUMMARY, summary);
    values.put(TodoTable.COLUMN_DESCRIPTION, description);

    if (todoUri == null) {
```

```

    // New todo
    todoUri = getResolver().insert(MyTodoContentProvider.CONTENT_URI, values);
} else {
    // Update todo
    getResolver().update(todoUri, values, null, null);
}
}

private void makeToast() {
    Toast.makeText(TodoDetailActivity.this, "Please maintain a summary",
        Toast.LENGTH_LONG).show();
}
}

```

Le fichier AndroidManifest.xml résultant ressemble à ceci.

AndroidManifestXML

Sélectionnez

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="de.vogella.android.todos"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="15" />

    <application
        android:icon="@drawable/icon"
        android:label="@string/app_name" >
        <activity
            android:name=".TodosOverviewActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

```

```
        android:name=".TodoDetailActivity"
        android:windowSoftInputMode="stateVisible|adjustResize" >
    </activity>

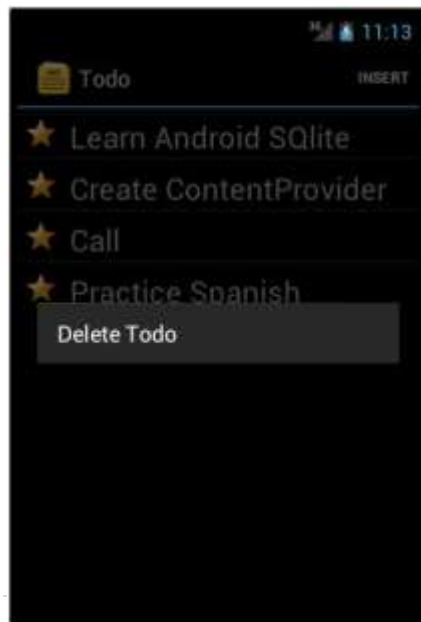
    <provider
        android:name=".contentprovider.MyTodoContentProvider"
        android:authorities="de.vogella.android.todos.contentprovider" >
    </provider>
</application>

</manifest>
```

Veuillez prendre note que `android:windowSoftInputMode = "stateVisible | adjustResize"` est défini pour `TodoDetailActivity`. Cela rend le clavier mieux harmonisé avec les widgets, mais il n'est pas nécessaire pour ce tutoriel.

Lancez votre application. Vous devriez être en mesure d'ajouter un nouvel élément de todo via le bouton "Insérer" dans l'ActionBar.

Un élément de todo existant peut être supprimé de la liste via un appui prolongé.



Pour modifier un élément todo existant, appuyez sur la ligne correspondante. Cela déclenche la deuxième activité.

SQLite stocke l'ensemble de la base dans un fichier. Si vous avez accès à ce fichier, vous pouvez travailler directement avec la base de données. L'accès au fichier de base de données SQLite ne fonctionne que dans l'émulateur ou sur un dispositif sur lequel vous avez les droits de "root".

Un dispositif standard Android n'accordera pas d'accès en lecture sur le fichier de base de données.

SQLite stocke l'ensemble de la base dans un fichier. Si vous avez accès à ce fichier, vous pouvez travailler directement avec la base de données. L'accès au fichier de base de données SQLite ne fonctionne que dans l'émulateur ou sur un dispositif sur lequel vous avez les droits de "root".

Un dispositif standard Android n'accordera pas d'accès en lecture sur le fichier de base de données.

Il est possible d'accéder à une base de données SQLite sur l'émulateur ou un dispositif sur lequel vous avez les droits de "root", via l'invite de commandes. Utilisez la commande suivante pour vous connecter à l'appareil.

```
adbshell
Sélectionnez
adb shell
```

La commande adb se trouve dans votre dossier d'installation du SDK Android dans le sous-dossier "outils de la plate-forme" ("platform-tools").

Ensuite, vous utilisez la commande "cd" pour naviguer vers le répertoire des bases de données et utilisez la commande "sqlite3" pour vous connecter à une base de données. Par exemple, dans mon cas:

```
cdshell
Sélectionnez
# Switch to the data directory
cd /data/data
# Our application
cd de.vogella.android.todos
# Switch to the database dir
cd databases
```



```
# Check the content
ls
# Assuming that there is a todotable file
# connect to this table
sqlite3 todotable.db
```

Les commandes les plus importantes sont:

### Tableau2 Commandes SQLite

Commande	Description
.help	Affiche toutes les commandes et les options.
.exit	Engendre la sortie de la commande sqlite3.
.schema	Affiche toutes les instructions CREATE qui ont été utilisées pour créer les tables de la base de données courante.

Vous allez trouver la documentation complète de SQLite [ici](http://www.sqlite.org/sqlite.html) (sur <http://www.sqlite.org/sqlite.html>).

Veuillez consulter le [Tutoriel sur l'utilisation des ListView](#) pour une introduction sur les ListViews and ListActivities.

Les changements dans SQLite sont ACID (atomique, cohérente, isolée, durable). Cela signifie que chaque opération de mise à jour, insertion et suppression est ACID. Malheureusement cela nécessite une certaine surcharge dans les processus de la base de données et vous devez donc englober les mises à jour de la base de données SQLite dans une transaction, et valider cette transaction après plusieurs opérations. Cela peut améliorer considérablement les performances.

Le code suivant démontre cette optimisation de la performance.

```
codeTransaction
Sélectionnez
db.beginTransaction();
try {
    for (int i= 0; i< values.lenght; i++){
```

```
// TODO préparer les valeurs des objets ContentValues
db.insert(your_table, null, values);
// Pour le cas où vous faites des MAJ plus amples
yieldIfContededSafely()
}
db.setTransactionSuccessful();
} finally {
    db.endTransaction();
}
```





Pour les mises à jour plus amples, vous devriez utiliser la méthode `yieldIfContededSafely()`. SQLite verrouille la base de données lors d'une transaction. Lors de cet appel, Android vérifie si un autre processus interroge la base de données et finit automatiquement la transaction et en ouvre une nouvelle. De cette façon, l'autre processus peut accéder aux données de façon asynchrone.

- [Codes source d'exemple](#)
- [Site web de SQLite](#)
- [Plugin SQLiteManager pour Eclipse](#)

Vous pouvez retrouver l'article original ici: [Android SQLite database and content provider - Tutorial](#). Nous remercions Lars Vogel qui nous a aimablement autorisé à traduire et héberger ses articles.

Nous remercions aussi [Mishulyna](#) pour sa traduction, ainsi que [Jacques THÉRY](#) pour sa relecture orthographique.

N'hésitez pas à commenter cet article ! 6 commentaires ★★★★★

Vous avez aimé ce tutoriel ? Alors partagez-le en cliquant sur les boutons suivants :     [inPartager](#)