


# TD d'introduction à Android


## Création de la page d'accueil PolyNews

Ce TD a pour but d'introduire le développement Android à travers la création d'une première interface pour PolyNews, une application qui vise à faciliter la communication à Polytech. La première fonction de l'application est la possibilité de visualiser des actualités. Dans ce TD une première vue chargée d'afficher une liste des actualités va être créée.


 Le tutoriel est entrecoupé d'explications sur le fonctionnement d'Android. Si vous estimez ne pas en avoir besoin, vous pouvez passer directement aux paragraphes avec l'icône ci-contre, qui indiquent ce qu'il faut faire.

### 0. Préambule

Il vous a été demandé d'installer la dernière versions des SDK Android, cela afin que le code généré à la création du projet soit le même pour tous. Si vous ne l'avez pas encore, il vous faut le **SDK 7.1.1** installé pour le TD.

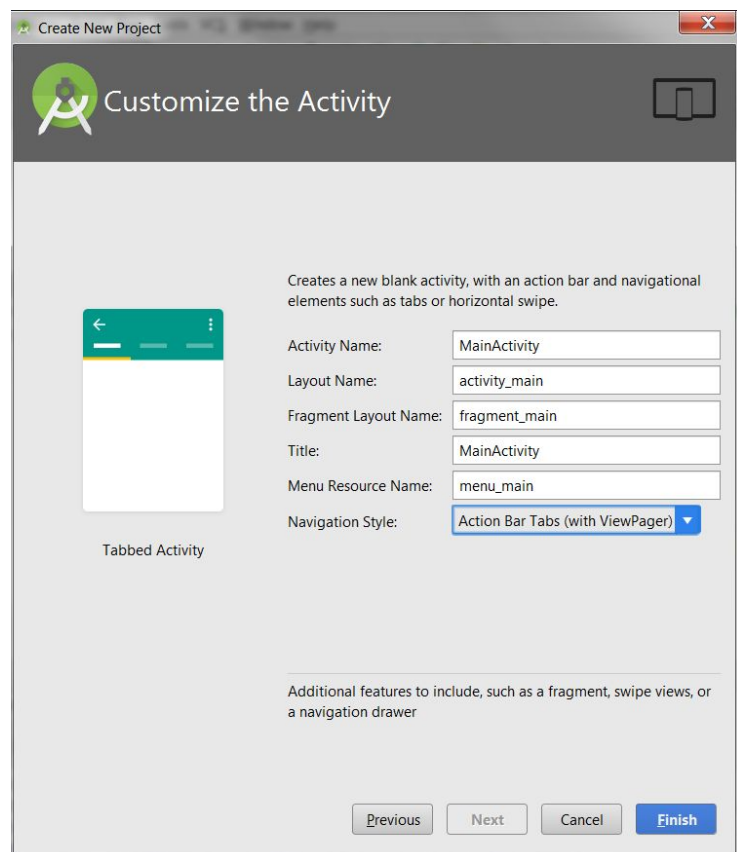
 Il vous sera demandé d'utiliser au cours de ce TD des fichiers à télécharger [ici](#).

### 1. Création du projet

 Sous Android Studio, créez un nouveau projet. Définissez comme SDK Minimum la version **API 21 Android 5.0 (Lollipop)**.

Puisque nous souhaitons par la suite avoir plusieurs types de contenus dans l'application, notre activité principale sera une activité de type *Tabbed Activity*. Sélectionner une navigation de type *Action Bar Tabs with ViewPager*, comme sur la capture d'écran.

Valider. Le nouveau projet et sa première activité vont être créés.




Configuration de la première activité de l'application

## 2. Structure d'un projet Android

Un projet Android est découpé en 3 parties principales: le manifest, les fichiers de ressources et enfin le code source Java.

### 2.0 Gradle

Le build d'un projet Android est géré grâce à Gradle, un équivalent de maven. Vous pouvez voir dans votre projet, deux fichiers **build.gradle**. L'un correspond à votre projet, l'autre à votre module principal: de la même manière que pour Maven, un projet gradle peut être découpé en hiérarchie modules. Ces deux fichiers sont équivalents au *pom.xml* racine d'un projet maven et au *pom* d'un de ses modules.

 Observer le contenu du build.gradle de votre module, appelé app. Il contient notamment une dépendances vers les librairies 'com.android.support:appcompat-v7:25.2.0'. La version devrait bien être au moins de **25.2.0** si vous avez le bon SDK d'installé avant de créer le projet. Cette librairie permet d'utiliser des fonctionnalités des dernières version d'Android dans les versions Android, en fournissant des implémentations spécifiques aux anciennes versions si nécessaire. Dans la suite, faites donc bien attention à utiliser les classes provenant des librairies appcompat, plutôt que les classes "par défaut".


### 2.1 Manifest

Le manifest sert à déclarer vos activités, c'est à dire les différents écrans de votre application, les informations de compatibilité de l'application avec les différents systèmes, ainsi que les droits d'accès de l'application aux fonctionnalités des appareils. Si vous définissez une nouvelle activité ou si vous souhaitez que votre application ait accès à Internet ou à un capteur, il faudra modifier ce manifest.

Ici, l'activité créée automatiquement a déjà été ajoutée dans le manifest. Les lignes

```
<intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

servent à définir le rôle de l'activité. Ici notre activité est l'activité principale de notre application, c'est elle qui sera créée et affichée lorsqu'un utilisateur lancera l'application.

 Sous Android, l'affichage est actualisé dès que l'utilisateur fait pivoter son appareil pour passer au format paysage par exemple. Pour simplifier ce tutoriel, nous allons bloquer cette activité dans une orientation portrait à l'aide des attributs suivants pour l'activité :

```
<activity
    android:name=".MainActivity"
    ...
    android:screenOrientation="portrait"
    android:configChanges="orientation">
```

**android:screenOrientation** indique que l'activité doit rester bloquée en mode portrait.

**android:configChanges** empêche les fonctions *onResume()* et *onPause()* de l'activité d'être appelées lorsque l'orientation de l'appareil change. Sans cette ligne, la vue ne changera pas si l'utilisateur tourne son téléphone, mais ces méthodes seraient malgré tout appelées.

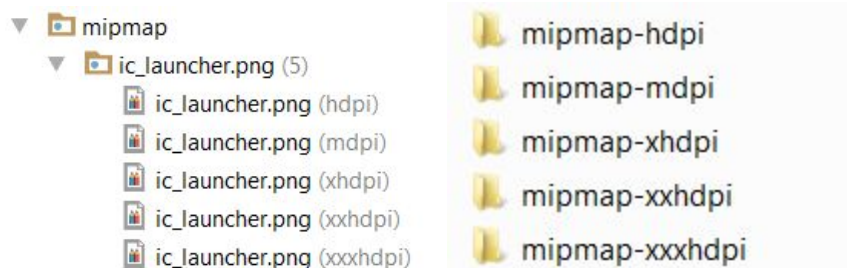
## 2.2 Fichiers de ressources

▼ res	Le dossier res contient tout ce dont l'application a besoin pour fonctionner.
drawable	<b>Drawable</b> sert à intégrer les images et graphismes utilisées par l'application.
▶ layout	<b>Layout</b> contient la description des vues au format XML.
▶ menu	<b>Menu</b> est similaire à layout mais contient les fichiers XML de description des menus de l'application.
▶ mipmap	<b>Mipmap</b> est utilisé pour l'icône de l'application.
▶ values	<b>Values</b> contient différents fichiers définissant des constantes de l'application.

Tous les dossiers de ressources peuvent être précisés selon différents critères en ajoutant “-filtres” derrière le nom des dossiers en question. Par exemple, le fichier *values/strings.xml* doit être utilisé pour définir les textes qui seront affichés à l'utilisateur dans l'application. Si l'on souhaite localiser l'application en Français par exemple, il faut créer un fichier *values-fr/strings.xml*.

Si l'appareil faisant tourner l'application est configuré avec la locale “fr”, c'est ce fichier qui sera utilisé. S'il est dans une autre locale ou que la string cherchée n'est pas spécifiée pour le Français, le fichier par défaut *values/strings.xml* sera utilisé.

De la même manière, on peut définir des images ou layout différents selon l'orientation, la résolution d'écran ou encore la version système de l'appareil. C'est Android qui se charge d'afficher l'image correcte selon l'appareil. C'est par exemple le cas dans le dossier mipmap, qui contient des images de l'icône de l'application à différentes résolutions.



Affichage du dossier mipmap sous Android Studio par rapport à la structure de dossier réelle à droite. Chacun des dossiers mipmap-xxx contient un fichier *ic\_launcher.png* qui sera utilisé selon l'appareil.

**Attention!** Les images devraient toujours être placées dans le dossier **drawable**, qui permet d'enlever les images des résolutions qui ne correspondent pas à l'appareil utilisé. Mipmap est un cas particulier à réserver uniquement pour l'icône de l'application, qui conserve toutes les images placées à l'intérieur afin que l'icône soit toujours affichée de manière optimale selon le contexte.

Enfin, on peut combiner plusieurs filtres pour avoir véritablement une application personnalisée selon son utilisateur et son contexte d'utilisation. Par exemple, un dossier *res/layout-land-night-v23* définit des layout utilisés uniquement lorsque l'application est en mode nuit et paysage, sur un système utilisant une version Android supérieure ou égale à 23.

## Appeler les fichiers de ressources

### Depuis d'autres ressources :



Ouvrez le fichier `menu/menu_main.xml`. Cliquez sur la valeur de `android:title` dans l'item. Regardez également dans les autres fichiers de `layout/`. Comment sont appelées les différents types de ressources?

### Depuis le code :

A la compilation, un fichier `R.java` contenant des références sur les différentes ressources de l'application est créé. C'est à l'aide de ce fichier qu'on peut accéder aux ressources depuis du code.

```
public static final class mipmap {  
    public static final int ic_launcher=0x7f030000;  
}  
  
public static final class string {  
    ...  
    public static final int action_settings=0x7f060014;  
    public static final int app_name=0x7f060015;  
    ...  
}
```

Aperçu du contenu du fichier `R.java` :

Depuis le code, on peut accéder aux différentes ressources grâce à leur identifiant contenu dans `R`. Par exemple, pour accéder au texte défini dans `strings.xml`, on appelle :

```
getString(R.string.section_format);
```

## 3. Étude de l'activité `MainActivity`



- Lancer l'application sur le terminal de votre choix.
- Observer rapidement le contenu et fonctionnement de l'application.

### 3.1 Structure du layout et lien avec le code

On s'intéresse dans un premier temps aux fichiers `MainActivity.java` contenant le code de l'activité et au fichier `layout/activity_main.xml` qui contient la structure de composants graphiques de l'activité.

#### Structure du fichier `layout/activity_main.xml`

- La balise `AppBarLayout` décrit la barre principale des applications Android, il n'est pas nécessaire de s'en préoccuper pour l'instant.
- **ViewPager** est un composant qui gère automatiquement le défilement horizontal entre différents éléments, c'est ce que nous allons utiliser pour passer d'un onglet à l'autre de l'application. C'est ce `ViewPager` qui va contenir par la suite le contenu des onglets.
- Le composant `FloatingActionButton` a été ajouté à la génération, mais n'est pas nécessaire à notre application.

La balise racine `CoordinatorLayout` générée dans le fichier de layout pose des problèmes de compatibilité avec les listes et grille que nous allons utiliser par la suite.



- Remplacez `android.support.design.widget.CoordinatorLayout` par **RelativeLayout**.
- Indiquez au `ViewPager` à l'aide de l'attribut "below" qu'il doit se placer sous la barre.

## Liens layout-code



L'activité et ses composants sont initialisés dans la méthode *onCreate()* de MainActivity.

- Comment l'activité sait-elle quel layout elle doit afficher?

Dans le fichier du layout, supprimez la balise contenant le bouton flottant et compilez le projet. Une erreur de compilation apparaît dans l'activité. Annulez la suppression de la balise et observez ses attributs, en particulier `android:id`. Cet attribut permet de définir un identifiant unique pour les éléments graphiques d'un layout. Le `+` signifie que c'est ici que la définition de l'id se fait.

Par la suite cet élément pourra être référencé depuis le layout ou depuis le code.



- Comment la vue du bouton est-elle appelée depuis MainActivity.java?
- Supprimer le code qui concerne le bouton et retirez-le du layout.

## 3.2 Affichage des onglets

Les onglets sont également initialisés dans le *onCreate()* :

```
mViewPager = (ViewPager) findViewById(R.id.container);  
mViewPager.setAdapter(mSectionsPagerAdapter);
```

Ici, on récupère l'objet ViewPager qui a été défini dans le layout xml et on lui associe un *Adapter*. Un *Adapter* Android est un objet chargé de faire le lien entre ce que l'on souhaite afficher, et l'objet graphique qui va contenir cette affichage. Ici, il nous faut donc définir un *SectionsPagerAdapter* qui fournira au ViewPager la vue des différents onglets qu'il faudra afficher.

```
TabLayout tabLayout = (TabLayout) findViewById(R.id.tabs);  
tabLayout.setupWithViewPager(mViewPager);
```

Ces deux lignes servent à lier l'affichage des onglets définis dans le layout avec le ViewPager.

### SectionsPagerAdapter

L'adapter utilisé est défini en temps que classe interne de notre activity et étend de *FragmentPagerAdapter*. Cette classe va associer à chaque onglet un **Fragment**.

En Android, les fragments sont des portions d'interface graphiques. On peut combiner plusieurs fragments dans une activité et réutiliser les fragments dans différentes activités. Ils permettent donc un meilleur découpage ainsi qu'une meilleure évolutivité du code. La combinaison de fragments permet par exemple de créer des vues différentes selon le type d'appareil (tablette ou téléphone), sans avoir à dupliquer de code.



Regardez la définition de la classe SectionsPagerAdapter.

- A quoi servent les 3 opérations définies?

### PlaceholderFragment

L'adapter crée des fragments de type PlaceholderFragment qui affichent tous une chaîne de caractère contenant le numéro de l'onglet associé.



Regardez la définition de la classe PlaceholderFragment.

- Comment le fragment sait-il quels éléments graphiques il doit afficher?



A partir de maintenant, il n'y a principalement que des consignes à suivre, qui ne seront donc plus précédées du symbole utilisé jusque là.

## 4. Définition de notre fragment personnalisé

Dans cette section nous allons définir le fragment d'affichage d'une liste d'actualités.

- Commencez par sortir la classe de la PlaceholderFragment de l'activité vers son propre fichier java.
- Créez un nouveau fragment NewsGridFragment, ainsi qu'un nouveau fichier de layout fragment\_news\_grid.xml contenant une simple TextView avec le texte de votre choix.
- Dans NewsGridFragment, chargez ce layout.
- Depuis l'Adapter de l'activité principale, faites en sorte que le premier onglet affiche votre nouveau fragment, laissez les autres onglets afficher les PlaceholderFragment. N'oubliez pas de changer l'intitulé de l'onglet.
- Lancer l'application et vérifier que le fragment est correctement chargé et affiché.

### 4.1 Définition du layout du fragment

- Dans le fichier de layout que vous avez défini, remplacez votre TextView par une **GridView**.

Ce composant permet de gérer un affichage en grille d'une liste de contenus.

- Donnez un identifiant à ce GridView pour qu'on puisse y accéder depuis le code du fragment.
- Vous pouvez définir d'autres attributs comme `columnWidth` ou `numColumns` pour définir le format de votre grille. Vous pouvez vous servir de l'auto-complétion ou de la vue "Design" pour découvrir les différentes possibilités de personnalisation des composants.

#### Note: les dimensions

Il est très fortement déconseillé de définir des dimensions en pixels, en raison de l'immense variété de terminaux sur le marché. A la place, il faut utiliser des **dp**, ou density-independent pixels. Cette unité tient compte de la résolution et de la taille d'écran des appareils pour afficher les éléments, et permet d'avoir des tailles réelles plus consistantes d'un appareil à un autre.

- Essayez de mettre la largeur des colonnes de la GridView à 300px et le nombre de colonnes à "auto-fit". Que se passe-t-il sur la visualisation si vous changez d'appareil (par exemple entre un Nexus 4 et un Nexus 5) ?
- Changez la largeur à 150dp. Que se passe-t-il quand vous changez d'appareil?

Evidemment, cela n'est pas toujours suffisant pour obtenir un affichage correct sur tous les appareils, notamment entre tablettes et téléphones. Ainsi, comme les chaînes de caractères, les dimensions utilisées par vos layout devraient être définies dans les ressources de l'application, dans les fichiers *values/dimens.xml*.



## 4.2 Affichage de la liste des articles

### 4.2.1 Définition du modèle et affichage simple d'une liste

- Créez une classe Java représentant le modèle d'un article contenant : un identifiant, un titre, un contenu texte, un auteur, une date, une catégorie (politique ou société), un type de média (image ou vidéo) et une url pour accéder au média.
- Définissez un constructeur initialisant tous ces attributs.
- Définissez la méthode *toString()* de votre article, et renvoyez par exemple son titre.

On va maintenant peupler notre GridView avec une liste d'articles.

- Dans votre fragment, redéfinissez l'opération *onActivityCreated(..)*. Laissez l'appel à super.
  - Dans cette opération, créez une liste contenant au moins 2 articles. Le premier doit avoir un média de type image (url [http://static.eyrolles.com/img/2/7/5/4/0/3/1/7/9782754031790\\_h430.jpg](http://static.eyrolles.com/img/2/7/5/4/0/3/1/7/9782754031790_h430.jpg)) et le deuxième une vidéo (url <https://www.youtube.com/watch?v=dQw4w9WgXcQ>).
- Nous chargerons les articles depuis une base de données dans la suite du TD.

Il nous faut maintenant associer le contenu de notre liste avec la *GridView* à l'aide d'un *Adapter*.

- Créez un nouvel objet *ListAdapter* affichant le *toString()* des Articles dans la grille.

```
new ArrayAdapter<News>(this.getContext(),  
    android.R.layout.simple_list_item_1, newsList);
```

- Récupérez l'objet *GridView* correspondant à votre liste d'article et appelez *setAdapter* dessus en lui donnant votre *Adapter*. Android se charge du reste.
- Lancer l'application et assurez-vous que la grille et son contenu s'affiche bien.

### 4.2.2 Création d'un Adapter personnalisé

Plutôt que d'utiliser cet Adapter par défaut, nous allons définir un Adapter avec un layout personnalisé, nous offrant ainsi plus de liberté sur la façon de présenter nos articles.

- Créez un layout xml personnalisé représentant un élément de la liste. Organisez-le à comme vous le souhaitez avec les différents layouts à disposition. La vue doit permettre d'afficher une image d'aperçu, le titre, la date ainsi que la catégorie de l'article. Créez un objet graphique pour chacun de ces éléments et donnez-leur un identifiant.

Si vous le souhaitez, vous pouvez utiliser une *CardView* et y placer vos layouts et éléments. La *CardView* est un élément respectant le style Material Design. Pour pouvoir utiliser des *CardView*, il vous faut modifier le fichier *build.gradle* et y ajouter la ligne suivante dans la section *dependencies* :

```
compile 'com.android.support:cardview-v7:25.2.0'
```

(note: Le numéro de version doit correspondre aux autres numéros définis dans votre *build.gradle*).

- Créez une classe *NewsCustomAdapter* héritant de *ArrayAdapter<News>*.

Définissez un constructeur prenant en entrée un *Context*, et une *List* de *News*.

Appelez le constructeur super en lui donnant en paramètre le contexte, 0 et la liste.



Exemple d'organisation possible des éléments dans une CardView

- Dans NewsCustomAdapter, redéfinir la méthode `getView(..)`. Cette méthode sera appelée dès qu'il y aura besoin d'afficher un élément de la liste.

```
LayoutInflater inflater = (LayoutInflater) getContext()
    .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
if (convertView == null) {
    convertView = inflater.inflate(R.layout.news_grid_item, null);
}
```

Le `LayoutInflater` va lire le fichier xml, et instancier les objets graphiques dans `convertView`. Si `convertView` n'est pas nul, c'est que l'objet graphique est déjà en mémoire, il n'y donc pas besoin de le recharger entièrement. Par exemple quand on scrolle dans une liste, l'élément graphique qui disparaît va être réutilisé pour celui qui apparaît, sans avoir à recharger le XML.

- Pour récupérer l'article qui doit être affiché, vous devez appeler la méthode `getItem(position)`
- Une fois ceci fait, récupérez les différents éléments de la vue à l'aide de `findViewById(..)`, et mettez à jour leur contenu pour qu'ils affichent les informations relatives à l'article concerné. Cela doit être fait même si `convertView` n'est pas nul. Les images seront chargées dans la prochaine section.

Il n'y a plus qu'à retourner dans notre fragment, et lui faire utiliser ce nouvel Adapter plutôt que celui par défaut.

- Lancer l'application et confirmez que votre affichage fonctionne.



## 5. Chargement des images d'aperçu

Les articles définis ont un lien vers leur images d'illustration. Il va donc falloir les télécharger.

- Pour que l'application ait l'autorisation de se connecter à Internet, il vous faut demander la permission dans le Manifest :

```
<uses-permission android:name="android.permission.INTERNET"/>
```

A présent, il va falloir charger les images, tout en évitant de bloquer l'application. Pour cela, Android a créé les *AsyncTask*, qui permettent de lancer des tâches de fond. Tous les appels internet doivent être effectués dans des *AsyncTask*.

*AsyncTask* est une classe définissant trois paramètres génériques :

- Le premier, Params, correspond aux paramètres dont la tâche a besoin pour s'exécuter. Dans notre cas il s'agira d'une chaîne de caractère contenant l'adresse de l'image à charger.
- Le deuxième, Progress, est utilisé pour représenter la progression de la tâche. Cela peut par exemple être un entier représentant le pourcentage de la tâche déjà effectué. Ce ne sera pas utile dans notre cas, vous pouvez mettre Void.
- Le dernier, Result correspond à la classe de l'objet qui sera renvoyé par l'*AsyncTask*. Dans notre cas ce sera un objet de type Bitmap.

Grâce à ces paramètres, nous pouvons redéfinir 3 opérations:

- *doInBackground* est la méthode qui va effectuer la tâche et renvoyer le résultat.
- *onPostExecute* est appelée une fois que *doInBackground* a fini, elle reçoit en paramètre le résultat renvoyé par l'opération, et peut donc actualiser la vue par exemple.
  - Créez une *AsyncTask<String, Void, Bitmap>* qui téléchargera les images.
  - Définissez un constructeur prenant en paramètre une *ImageView*, et sauvegardez-la.
  - Redéfinissez *doInBackground*. Utiliser un *InputStream* et *BitmapFactory.decodeStream()* pour charger l'image.
  - Redéfinissez *onPostExecute*. Associez le Bitmap téléchargé à l'*ImageView*.
- Dans la méthode *getView(..)* de *NewsCustomAdapter*, créez une instance de votre *AsyncTask* et appelez *execute()* avec l'URL de l'image à charger en paramètres.
- Enfin, pour afficher les aperçus de vidéos, vous pouvez faire la même chose. Les url dans la base sont des liens YouTube, mais vous pouvez à partir de ceux-ci obtenir l'identifiant YouTube de la vidéo, et télécharger l'image d'aperçu à l'adresse suivante (où XXXX est l'id de la vidéo) :

```
"http://img.youtube.com/vi/XXXX/default.jpg";
```

- Que se passe-t-il avec les images si vos éléments sortent de l'écran et que vous scrollez?
- Faites en sorte que pendant le chargement des images, c'est l'image par défaut qui vous a été fournie qui est affichée.
- Rajoutez une *ImageView* avec un icône "Play" au dessus de l'aperçu des vidéos uniquement.

Vous pouvez utiliser l'icône "Play" intégré à Android : `@android:drawable/ic_media_play`

Il vous faudra la rendre visible uniquement si l'Article est illustré par une vidéo.

## 6. Bases de données

Afin d'afficher de véritables articles, nous allons les charger depuis une base de données qui vous a été fournie avec le TD. Pour cela, Android fournit une classe **SQLiteOpenHelper** que l'on peut étendre pour lire dans une base de données. Chaque application Android dispose d'une base de données qu'elle peut utiliser comme elle le souhaite.

- Placez la base de données dans le répertoire *src/main/assets* de votre projet.
- Récupérez le squelette de la classe `NewsDataBaseHelper` fourni. Cette classe hérite de `SQLiteOpenHelper` et contient les méthodes permettant de charger la base de données depuis le fichier, et de la copier à l'emplacement par défaut de chaque application Android.
- Complétez la méthode `getAllArticles()`, qui doit lire la base et retourner la liste des articles qu'elle contient. On accède à l'ensemble de la base avec l'appel suivant :

```
myDataBase.rawQuery("SELECT * FROM news ORDER BY date DESC", null);
```

Cette requête renvoie un objet de type `Cursor`. On l'utilise pour parcourir la base :

```
cursor.moveToFirst();  
while (!cursor.isAfterLast()) {  
    cursor.moveToNext();  
}
```

On accède ensuite au contenu de la base en appelant les opérations `getString(index)` et `getInt(index)` sur le `Cursor`. Ces méthodes prennent pour paramètre la position de l'élément souhaité dans la base.

news	
_id	int PK
title	text
content	text
author	text
date	text
category	int
media_type	int
media_path	text

Schéma de la base.

!! L'attribut `_id` est à l'index 7. Les autres sont de l'index 0 à 6 dans l'ordre de la figure. !!

Dans la base, la catégorie politique correspond à l'entier 1, et société à 2. Le type de média photo à 0 et vidéo à 1.

Les dates sont sous forme de texte, au format :  
"yyyy-MM-dd HH:mm:ss.SSS"

- N'oubliez pas de fermer le curseur une fois que vous avez fini de l'utiliser.
- Pour accéder à la base dans votre fragment, créer une instance de `NewsDatabaseHelper` en lui donnant en paramètre le contexte d'exécution, dans ce cas c'est l'activité qui a lancé le fragment, que l'on récupère à l'aide de `getActivity()` ou `getContext()`.
- Appelez `createDatabase()` puis `openDatabase()` afin de la charger la base, récupérer la liste des articles, puis fermez la base avec `close()`. Donnez cette nouvelle liste à votre `Adapter` à la place de l'ancienne.

**!! ATTENTION !!** `createDatabase`, dans le cas où la base n'existe pas encore, crée une nouvelle base, puis y copie le contenu du fichier que vous avez placé dans `assets`. Si cette copie échoue, la base aura malgré tout été définie, et la copie ne sera donc pas tentée à nouveau au prochain lancement. Pour forcer à nouveau la création de la base, il vous faudra désinstaller complètement l'application de votre appareil (physique ou virtuel) avant de relancer l'application. !!

## 7. Pour aller plus loin

Les deux éléments ci dessous sont indépendants et proposent des bonnes pratiques pour améliorer le code que nous avons créé dans ce TD.

### 7.1 Utilisation de RecyclerView

Le mécanisme mis en place via les `convertView` dans les `ListView/GridView` permet de ne pas recharger complètement les vues depuis le XML quand on scrolle dans une liste.

Android recommande à présent de ne plus utiliser de `ListView` ou `GridView`, mais des `RecyclerView` avec `layout` linéaires ou `layout` en grille qui forcent l'utilisation de cette optimisation.

- Dans votre fichier `build.gradle`, ajouter la dépendance sur les `RecyclerView`.

compile "com.android.support:recyclerview-v7:25.2.0"

- Définissez un nouvel `Adapter`, héritant de `RecyclerView.Adapter<NewsViewHolder>`
  - Cet `adapter` doit garder en mémoire la liste d'objet qu'il affiche, et redéfinir la méthode `getItemCount()` pour renvoyer la taille de sa liste.
  - Un `ViewHolder` est une classe qui contient les objets graphiques représentant un élément (`TextView`, `ImageView`, etc.). C'est un simple `POJO` qui permet de garder proprement tous les éléments graphiques liés à un objet de modèle. Définissez votre classe `NewsViewHolder`, qui contiendra des références sur les mêmes éléments qui sont défini dans votre fichier `xml` représentant une `news`. Créez un constructeur initialisant ces objets.
  - Dans votre `adapter`, définissez la méthode `onCreateViewHolder`, qui charge le fichier `xml` et stocke les objets graphique dans une instance de `NewsViewHolder`.
  - Redéfinissez la méthode `onBindViewHolder` dans laquelle vous donnez la valeur correcte à vos objets graphiques selon l'objet à afficher.
- 
- Dans votre fichier `xml`, remplacez la `GridView` par un `RecyclerView`.
  - Dans le fragment, récupérer le `RecyclerView`, lui donner la liste et assigner votre nouvel `Adapter`.
  - Enfin, indiquez au `RecyclerView` le `manager` qu'il doit utiliser pour ses éléments:

```
mLayoutManager = new GridLayoutManager(this);  
mRecyclerView.setLayoutManager(mLayoutManager);
```

Plus d'information dans la [doc Android](#), qui est très complète et contient beaucoup d'exemples.

## 7.2 Adapter pour base de données

Dans la version proposée, tout le contenu de la base de données est chargé d'un coup et donné à l'Adapter de la GridView. Dans des conditions réelles ce n'est pas quelque chose d'acceptable. Il faudrait plutôt utiliser un **CursorAdapter**, qui va se charger d'aller charger les informations dans la base uniquement quand nécessaire.

- Créez un nouvel Adapter héritant de CursorAdapter. Il vous faut redéfinir deux méthodes.
- `newView(..)` sert à initialiser le composant qui va afficher l'élément de la liste. Utiliser l'Inflater comme vu précédemment.
- `bindView(...)` sert à mettre à jour la vue créée en fonction de l'élément à afficher. C'est la même chose que dans le `getView(..)` de l'ArrayAdapter, mais il n'y a pas besoin d'inflater la vue ici, puisque cela est fait dans `newView(..)`.
- Si ce n'est pas déjà fait, ajoutez dans le NewsDatabaseHelper une méthode prenant un Cursor et renvoyant un objet de type Article. Utilisez-là dans `bindView()` pour récupérer la news à afficher.

Une fois ceci fait, il nous faut initialiser notre nouvel adapteur. Pour ça, il a besoin de connaître le cursor. La première solution est d'appeler le NewsDatabaseHelper pour qu'il renvoie le résultat de la requête utilisée pour récupérer les articles. Cependant il faudrait pouvoir faire cela en fond, afin de ne pas bloquer le Thread UI avec la requête à la base.

- Créer une AsyncTask chargée d'effectuer la requête à la base de données.