Android : la persistance des données

Kotlin

* Présentation

Android propose plusieurs possibilités pour persister des données :

* les bases de données,
* les SharedPreferences (préférences partagées),
* le stockage dans des fichiers internes,
* le stockage dans des fichiers externes.

Tous ces points seront traités dans ce chapitre.

* SharedPreferences
* Présentation

L'API SharedPreferences permet d'enregistrer des informations sous forme de clé/valeur dans des fichiers XML de manière très simple. Les objets de type SharedPreferences permettent de stocker des informations dans des fichiers pouvant soit être lus uniquement par l'activité qui a créé le fichier soit être lus par plusieurs activités différentes.

* Fonctionnement

Pour créer un tel fichier de préférences, il existe deux fonctions :

* getSharedPreferences() : permet de créer un fichier de préférences pouvant être lu par plusieurs activités.
* getPreferences() : permet de créer un fichier de préférences pouvant être lu uniquement par l'activité qui l'a créé.

Si le fichier n'existe pas, ces fonctions le créent et retournent un objet permettant de modifier le fichier et si le fichier existe déjà elles retournent un objet permettant de modifier le fichier.

Les étapes pour manipuler un fichier de préférences sont :

* Récupérer un objet de type SharedPreferences pour manipuler le fichier.
* Manipuler le fichier avec les fonctions putXX() et getXX().
* Enregistrer les modifications à l'aide de la fonction commit().
* Récupérer un objet de type SharedPreferences
* Interactivité

Le code ci-dessous montre comment récupérer un objet de type SharedPreferences pour les fichiers manipulables depuis plusieurs activités :

Récupération d'un objet de type SharedPreferences

val objetSharedPreferences =   
activite?.getSharedPreferences("fichier\_de\_preferences",   
Context.MODE\_PRIVATE)

* Intra activité

Le code ci-dessous montre comment récupérer un objet de type SharedPreferences pour les fichiers manipulables depuis une seule activité :

Récupération d'un objet de type SharedPreferences

val objetSharedPreferences =   
activite?.getPreferences(Context.MODE\_PRIVATE)

* Enregistrer dans un fichier de préférences

Voici comment utiliser un objet de type SharedPreferences pour écrire dans un fichier de préférences partagé.

Écrire dans un fichier de préférences partagé

val editeur = objetSharedPreferences.edit()   
editeur.putString("cle01","valeurString")   
editeur.putInt("cle02",888)   
editeur.commit()

cle01 est la clé et "valeurString" est la valeur...

* Lire dans un fichier de préférences

Voici comment utiliser un objet de type SharedPreferences pour lire dans un fichier de préférences partagé.

val recuperation = objetSharedPreference.getString("mot","Pas de   
valeur avec cette clé")

recuperation sera égale à la valeur liée à la clé "mot".

* Démonstration

Cette démonstration montre concrètement comment, depuis une IHM, il est possible de lire et d'écrire dans un fichier de préférences partagé.

La démonstration est réalisée dans un nouveau projet dont l'activité principale se nomme MainActivity.

Activity\_main.xml

<?xml version="1.0" encoding="utf-8"?>   
<android.support.constraint.ConstraintLayout   
xmlns:android="http://schemas.android.com/apk/res/android"   
    xmlns:app="http://schemas.android.com/apk/res-auto"   
    xmlns:tools="http://schemas.android.com/tools"   
    android:layout\_width="match\_parent"   
    android:layout\_height="match\_parent"   
   
    tools:context="fr.acos.sharedpreferenceaveckotlin.MainActivity">   
   
    <Button   
        android:id="@+id/btn\_ecrire"   
        android:layout\_width="wrap\_content"   
        android:layout\_height="wrap\_content"   
        android:text="Ecrire"   
        app:layout\_constraintBottom\_toBottomOf="parent"   
        app:layout\_constraintLeft\_toLeftOf="parent"   
        app:layout\_constraintRight\_toRightOf="parent"   
        app:layout\_constraintTop\_toTopOf="parent"   
        android:onClick="onClickEcrire"   
     />   
   
    <Button   
        android:id="@+id/btn\_lire"   
        android:layout\_width="wrap\_content"   
        android:layout\_height="wrap\_content"   
        android:text="Lire"   
        app:layout\_constraintTop\_toBottomOf="@id/btn\_ecrire"   
        app:layout\_constraintLeft\_toLeftOf="parent"   
        app:layout\_constraintRight\_toRightOf="parent"   
        android:onClick="onClickLire"   
     />   
   
</android.support.constraint.ConstraintLayout>

Dans cet exemple, l'IHM contient :

* Un bouton "Ecrire" qui exécute des instructions permettant d'écrire dans un fichier de préférences partagé.
* Un bouton "Lire" qui exécute des instructions permettant d'écrire dans un fichier de préférences partagé.

MainActivity.kt

class MainActivity : AppCompatActivity() {   
   
    override fun onCreate(savedInstanceState: Bundle?) {   
        super.onCreate(savedInstanceState)   
        setContentView(R.layout.activity\_main)   
    }   
   
    fun onClickEcrire(view: View)   
    {   
        val objetSharedPreferences =   
this?.getPreferences(Context.MODE\_PRIVATE)   
   
        val editeur = objetSharedPreferences.edit()   
   
        editeur.putString("mot","Hello")   
   
        editeur.commit()   
    }   
   
    fun onClickLire(view: View)   
    {   
        val objetSharedPreferences =   
this?.getPreferences(Context.MODE\_PRIVATE)   
   
        val recuperation =   
objetSharedPreferences.getString("mot","Pas de valeur avec cette clé")   
   
        Toast.makeText(this,recuperation,Toast.LENGTH\_LONG).show()   
    }   
}

Dans cet exemple, le contrôleur contient :

* Une fonction onClickEcrire() qui écrit dans un fichier de préférences partagé disponible uniquement depuis l'activité MainActivity. La fonction écrit la valeur "Hello" liée à la clé "mot".
* Une fonction onClickCoucou() qui lit dans le même fichier de préférences partagé disponible uniquement depuis l'activité MainActivity. La fonction lit la valeur liée à la clé "mot". La valeur est ensuite affichée dans un toast.
* Fichier interne
* Présentation

Les appareils Android peuvent enregistrer des données sur l'espace interne du téléphone. Par défaut, les fichiers enregistrés sont privés, les autres applications ne peuvent y accéder. Android fournit un répertoire privé sur le système de fichiers pour chaque application où il est possible d'organiser les fichiers dont l'application a besoin.

* Écrire dans un fichier interne

Voici les instructions permettant d'écrire dans un fichier interne.

Exemple d'écriture dans un fichier interne

File(this.filesDir, "unFichier.txt").outputStream().use {   
     it.write("Bonjour !!!".toByteArray())   
 }

* Lire dans un fichier interne

Voici les instructions permettant de lire dans un fichier interne.

Exemple de lecture d'un fichier interne

val recuperation = File(this.filesDir,   
"unFichier.txt").bufferedReader().readText();

* Mettre en cache un fichier interne

Il est possible de stocker des données de manière temporaire, plutôt que de les stocker de manière persistante. Il existe un répertoire de cache spécifique à cet usage pour chaque application. Lorsque l'espace de stockage interne de l'appareil est insuffisant, Android peut supprimer ces fichiers de cache pour récupérer de l'espace.

Exemple d'enregistrement de données de manière temporaire

File.createTempFile("test", ".txt",   
this.cacheDir).outputStream().use {   
    it.write("Bonjour !!!".toByteArray())   
}   
//fr.acos.fichierinternewithkotlin/cache/test9009195298206874274.txt

* Quelques fonctions

Voici quelques fonctions utiles :

|  |  |
| --- | --- |
| Fonction | Description |
| fileList() | Retourne la liste des fichiers internes de l'application. |
| getFilesDir() | Retourne un objet de type File qui représente le dossier contenant les fichiers internes de l'application. |
| getDir(name, mode) | Permet de créer un nouveau dossier |
| getCacheDir() | Retourne un objet de type File qui représente le dossier contenant les fichiers temporaires de l'application. |

* Démonstration

Cette démonstration montre concrètement comment, depuis une IHM, il est possible de lire et d’écrire dans un fichier interne.

Activity\_main.xml

<?xml version="1.0" encoding="utf-8"?>   
<android.support.constraint.ConstraintLayout   
xmlns:android="http://schemas.android.com/apk/res/android"   
    xmlns:app="http://schemas.android.com/apk/res-auto"   
    xmlns:tools="http://schemas.android.com/tools"   
    android:layout\_width="match\_parent"   
    android:layout\_height="match\_parent"   
    tools:context="fr.acos.fichierinternewithkotlin.MainActivity">   
   
   
    <Button   
        android:id="@+id/btn\_ecrire"   
        android:layout\_width="wrap\_content"   
        android:layout\_height="wrap\_content"   
        android:text="Ecrire"   
        app:layout\_constraintBottom\_toBottomOf="parent"   
        app:layout\_constraintLeft\_toLeftOf="parent"   
        app:layout\_constraintRight\_toRightOf="parent"   
        app:layout\_constraintTop\_toTopOf="parent"   
        android:onClick="onClickEcrire"   
        />   
   
    <Button   
        android:id="@+id/btn\_lire"   
        android:layout\_width="wrap\_content"   
        android:layout\_height="wrap\_content"   
        android:text="Lire"   
        app:layout\_constraintTop\_toBottomOf="@id/btn\_ecrire"   
        app:layout\_constraintLeft\_toLeftOf="parent"   
        app:layout\_constraintRight\_toRightOf="parent"   
        android:onClick="onClickLire"   
        />   
   
</android.support.constraint.ConstraintLayout>

Dans cet exemple, l'IHM contient :

* Un bouton "Ecrire" qui exécute des instructions permettant d'écrire dans un fichier interne.
* Un bouton "Lire" qui exécute des instructions permettant de lire dans un fichier interne.

MainActivity.kt

class MainActivity : AppCompatActivity() {   
   
    override fun onCreate(savedInstanceState: Bundle?) {   
        super.onCreate(savedInstanceState)   
        setContentView(R.layout.activity\_main)   
    }   
   
    fun onClickEcrire(view: View)   
    {   
       File(this.filesDir, "unFichier.txt").outputStream().use {   
            it.write("Bonjour !!!".toByteArray())   
        }   
    }   
   
    fun onClickLire(view: View)   
    {   
        val recuperation = File(this.filesDir,   
"unFichier.txt").bufferedReader().readText();   
   
        Toast.makeText(this,recuperation, Toast.LENGTH\_LONG).show()   
    }   
}

Dans cet exemple, le contrôleur contient :

* Une fonction onClickEcrire() qui écrit dans un fichier interne disponible uniquement depuis l'application. La fonction écrit la valeur "Bonjour !!!" dans le fichier "unFichier.txt".
* Une fonction onClickLire () qui lit dans le fichier "unFichier.txt" disponible uniquement depuis l'application. La fonction lit le contenu du fichier puis l'affiche dans un toast.
* Fichier externe
* Présentation

Les appareils Android peuvent enregistrer des données sur des espaces de stockage externes. Ces espaces de stockages sont appelés ainsi car ils sont amovibles et peuvent parfois être inaccessibles.

Les étapes pour accéder à un espace de stockage externe sont les suivantes :

* Avoir la permission de lecture ou/et d'écriture sur un espace de stockage externe.
* Vérifier la disponibilité de l'espace de stockage.
* Vérifier que le fichier/dossier sur lequel il y a un accès soit disponible.

Le système fournit des répertoires publics pour que certains fichiers tels que des photos, sonneries, musiques et autres soient disponibles pour toutes les applications.

Il est également possible d'enregistrer des fichiers sur un espace de stockage externe dans un répertoire spécifique à l'application que le système supprimera lorsque l'utilisateur désinstallera l'application. Cela permet d'avoir un plus grand espace de stockage.

Lorsque l'on parle d'espace de stockage externe on parle généralement de carte SD.

* Permissions

Écriture

Voici la balise de permission nécessaire pour écrire sur un espace de stockage externe :

AndroidManifest.xml

<manifest ...>   
    <uses-permission android:name="android.  
permission.WRITE\_EXTERNAL\_STORAGE" />   
    ...   
</manifest>

Le droit d'écriture donne implicitement le droit de lecture.

Lecture

Voici la balise de permission nécessaire pour lire sur un espace de stockage externe :

AndroidManifest.xml

<manifest ...>   
    <uses-permission android:name="android.  
permission.READ\_EXTERNAL\_STORAGE" />   
    ...   
</manifest>

* Vérifier la disponibilité d'un espace de stockage

Lecture/Écriture

Le code ci-dessous permet de tester si l'espace de stockage externe est disponible en lecture et écriture :

fun isExternalStorageWritable(): Boolean {   
    return Environment.getExternalStorageState() ==   
Environment.MEDIA\_MOUNTED   
}

Lecture

Le code ci-dessous permet de tester si l'espace de stockage externe est disponible en lecture :

fun isExternalStorageReadable(): Boolean {   
     return Environment.getExternalStorageState() in   
        setOf(Environment.MEDIA\_MOUNTED,   
Environment.MEDIA\_MOUNTED\_READ\_ONLY)   
}

* Écrire dans un fichier externe privé

Le code ci-dessous permet d'écrire dans un fichier externe nommé unFichier.txt. Le fichier sera enregistré dans le dossier documents :

val dossier =   
this.getExternalFilesDir(Environment.DIRECTORY\_DOCUMENTS)   
   
File(dossier, "unFichier.txt").outputStream().use {   
    it.write("Je rêve, rêve rêve".toByteArray())   
}

* Lire un fichier externe privé

Le code ci-dessous permet de lire dans un fichier externe nommé unFichier.txt. Le fichier se trouve dans le dossier documents :

val dossier = this.getExternalFilesDir(Environment.DIRECTORY\_DOCUMENTS)   
   
val recuperation = File(dossier, "unFichier.txt").  
bufferedReader().readText();

* Écrire un fichier externe public

En utilisant la fonction getExternalStoragePublicDirectory() l'écriture du fichier se fait dans un dossier public et disponible pour les autres applications :

val dossier =   
Environment.getExternalStoragePublicDirectory(Environment.  
DIRECTORY\_NOTIFICATIONS)   
val file = File(dossier, "unFichier.txt")   
file.outputStream().use {   
    it.write("Je voyage".toByteArray())   
}

Une requête de permission doit être effectuée.

* Lire un fichier externe public

Le code ci-dessous permet de lire un fichier externe nommé unFichier.txt. Le fichier se trouve dans le dossier public documents :

val dossier =   
Environment.getExternalStoragePublicDirectory(Environment.  
DIRECTORY\_NOTIFICATIONS)   
val recuperation = File(dossier, "unFichier.txt").  
bufferedReader().readText();

Une requête de permission doit être effectuée.

* Démonstration

Cette démonstration montre concrètement comment, depuis une IHM, il est possible de lire et d’écrire dans un fichier externe.

AndroidManifest.xml

<?xml version="1.0" encoding="utf-8"?>   
<manifest   
xmlns:android="http://schemas.android.com/apk/res/android"   
    package="fr.acos.externalstoragewithkoltin">   
   
    <uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE"  />   
   
    <application   
        android:allowBackup="true"   
        android:icon="@mipmap/ic\_launcher"   
        android:label="@string/app\_name"   
        android:roundIcon="@mipmap/ic\_launcher\_round"   
        android:supportsRtl="true"   
        android:theme="@style/AppTheme">   
        <activity android:name=".MainActivity">   
            <intent-filter>   
                <action android:name="android.intent.action.MAIN" />   
                <category android:name="android.intent.category.LAUNCHER" />   
            </intent-filter>   
        </activity>   
    </application>   
   
</manifest>

Dans cet exemple, la permission en lecture et écriture sur un fichier est définie.

Activity\_main.xml

<?xml version="1.0" encoding="utf-8"?>   
<android.support.constraint.ConstraintLayout   
xmlns:android="http://schemas.android.com/apk/res/android"   
    xmlns:app="http://schemas.android.com/apk/res-auto"   
    xmlns:tools="http://schemas.android.com/tools"   
    android:layout\_width="match\_parent"   
    android:layout\_height="match\_parent"   
    tools:context="fr.acos.externalstoragewithkoltin.MainActivity">   
   
    <Button   
        android:id="@+id/btn\_enregistrer"   
        android:layout\_width="wrap\_content"   
        android:layout\_height="wrap\_content"   
        android:text="Enregistrer"   
        app:layout\_constraintBottom\_toBottomOf="parent"   
        app:layout\_constraintLeft\_toLeftOf="parent"   
        app:layout\_constraintRight\_toRightOf="parent"   
        app:layout\_constraintTop\_toTopOf="parent"   
        android:onClick="onClickEcrire"   
        />   
   
    <Button   
        android:id="@+id/btn\_lire"   
        android:layout\_width="wrap\_content"   
        android:layout\_height="wrap\_content"   
        android:text="Lire"   
        app:layout\_constraintLeft\_toLeftOf="parent"   
        app:layout\_constraintRight\_toRightOf="parent"   
        app:layout\_constraintTop\_toBottomOf="@id/btn\_enregistrer"   
        android:onClick="onClickLire"   
        />   
   
</android.support.constraint.ConstraintLayout>

Dans cet exemple, l'IHM contient :

* Un bouton "Ecrire" qui exécute des instructions permettant d'écrire dans un fichier externe.
* Un bouton "Lire" qui exécute des instructions permettant de lire dans un fichier externe.

MainActivity.kt

package fr.acos.externalstoragewithkoltin   
   
import android.Manifest   
import android.support.v7.app.AppCompatActivity   
import android.os.Bundle   
import android.os.Environment   
import android.view.View   
import android.widget.Toast   
import java.io.File   
import android.content.pm.PackageManager   
import android.support.v4.app.ActivityCompat   
   
   
class MainActivity : AppCompatActivity() {   
   
    override fun onCreate(savedInstanceState: Bundle?) {   
        super.onCreate(savedInstanceState)   
        setContentView(R.layout.activity\_main)   
    }   
   
    fun onClickEcrire(view: View)   
    {   
        ActivityCompat.requestPermissions(this,   
arrayOf(Manifest.permission.WRITE\_EXTERNAL\_STORAGE), 14540)   
    }   
   
    override fun onRequestPermissionsResult(requestCode: Int,   
permissions: Array<out String>, grantResults: IntArray) {   
        super.onRequestPermissionsResult(requestCode, permissions,   
grantResults)   
        if (requestCode == 14540) {   
            if (grantResults.size == 1   
                    && grantResults[0] ==   
PackageManager.PERMISSION\_GRANTED) {   
                val dossier = Environment.getExternalStorage  
PublicDirectory(Environment.DIRECTORY\_NOTIFICATIONS)   
                val file = File(dossier, "unFichier.txt")   
                file.outputStream().use {   
                    it.write("Je voyage".toByteArray())   
                }   
            }   
        }   
    }   
   
    fun onClickLire(view: View)   
    {   
        val dossier =   
Environment.getExternalStoragePublicDirectory(Environment.  
DIRECTORY\_NOTIFICATIONS)   
        val recuperation = File(dossier,   
"unFichier.txt").bufferedReader().readText();   
        Toast.makeText(this,recuperation,   
Toast.LENGTH\_LONG).show()   
    }   
}

Dans cet exemple, le contrôleur contient :

* Une fonction onClickEcrire() qui écrit dans un fichier interne disponible uniquement depuis l'application. La fonction écrit la valeur "Bonjour !!!" dans le fichier "unFichier.txt".
* Une fonction onClickLire () qui fait une demande d'autorisation.
* Une fonction onRequestPermissionsResult() qui exécutera le code pour lire dans le fichier externe au moment où l'utilisateur aura donné son autorisation de lecture.
* Base de données avec SQLiteOpenHelper
* Présentation

Il existe plusieurs méthodes pour accéder à une base de données. La méthode présentée ci-dessous utilise les fonctionnalités du package android.database.sqlite, c'est la manière historique, elle est fastidieuse, elle peut être simplifiée grâce à ANKO (cf. chapitre Android : quelques librairies incontournables - Anko). Le site Android Developer préconise d'utiliser l'ORM Room pour accéder aux données de vos applications. Malgré ces recommandations il a été choisi de présenter cette manière de faire, car elle propose des fonctionnalités bas niveau et peut s'avérer puissante pour certains cas d'utilisations. De plus, les développeurs Java pourront être curieux de voir comment cette façon de faire est mise en œuvre avec Kotlin.

* Procédure

Pour mettre en place une base de données :

* Créer une classe modèle pour chaque table de la bdd.
* Créer une classe de type Contract pour chaque table de la bdd.
* Créer une classe BddHelper héritant de SQLiteOpenHelper, cette classe va permettre de créer et mettre à jour la structure de la base de données et va aussi permettre de fournir des objets représentant la base de données.
* Créer une DAO pour chaque table.
* Mise en place
* Schéma de la base de données

Pour présenter la mise en place d’une base de données, nous allons travailler sur une base de données nommée "BddExemple" contenant une table personnes avec un identifiant de type INTEGER et un nom et un prénom de type TEXT.

* Modèle

Le code ci-dessous contient une classe objet représentant une personne :

Personne.kt

package fr.acos.sqliteopenhelperwithkotlin.bo   
   
data class Personne constructor(val id:Int?,val nom:String, val  
 prenom:String)

* Contract

Le code ci-dessous contient une classe avec les valeurs utiles pour construire et manipuler la table personnes en SQL :

Contracts.kt

package fr.acos.sqliteopenhelperwithkotlin.contracts   
   
import android.provider.BaseColumns   
   
object Contracts {   
   
    object Personnes : BaseColumns {   
        const val TABLE\_NAME = "personnes"   
        const val COLUMN\_NAME\_ID = "id"   
        const val COLUMN\_NAME\_NOM = "nom"   
        const val COLUMN\_NAME\_PRENOM = "prenom"   
   
        const val SQL\_CREATE\_TABLE =   
                "CREATE TABLE ${TABLE\_NAME} (" +   
                        "${COLUMN\_NAME\_ID} INTEGER PRIMARY KEY   
AUTOINCREMENT," +   
                        "${COLUMN\_NAME\_NOM} TEXT," +   
                        "${COLUMN\_NAME\_PRENOM} TEXT)"   
   
        const val SQL\_DELETE\_TABLE = "DROP TABLE IF EXISTS   
${Personnes.TABLE\_NAME}"   
    }   
}

* BddHelper

Le code ci-dessous contient une classe permettant de créer/mettre à jour la base de données et permet aussi de récupérer un objet représentant une connexion à la base de données :

BddHelper.kt

package fr.acos.sqliteopenhelperwithkotlin.dao   
   
import android.database.sqlite.SQLiteDatabase   
import android.database.sqlite.SQLiteOpenHelper   
import fr.acos.sqliteopenhelperwithkotlin.contracts.Contracts   
import android.content.Context   
   
class BddHelper(context: Context) : SQLiteOpenHelper(context,   
DATABASE\_NAME, null, DATABASE\_VERSION) {   
   
//Permet de créer la table personnes   
    override fun onCreate(db: SQLiteDatabase) {   
        db.execSQL(Contracts.Personnes.SQL\_CREATE\_TABLE)   
    }   
   
//Permet de mettre à jour la table personnes   
    override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int,   
newVersion: Int) {   
        db.execSQL(Contracts.Personnes.SQL\_DELETE\_TABLE)   
        onCreate(db)   
    }   
   
//Contient les informations relatives à la bdd   
 companion object {   
        const val DATABASE\_VERSION = 1   
        const val DATABASE\_NAME = "BddExemple.db"   
    }   
}

* DAO

Le code ci-dessous permet de définir la DAO qui contient les fonctions d'accès à la table personnes.

PersonneDao.kt

package fr.acos.sqliteopenhelperwithkotlin.dao   
   
import android.content.ContentValues   
import android.content.Context   
import fr.acos.sqliteopenhelperwithkotlin.bo.Personne   
import fr.acos.sqliteopenhelperwithkotlin.contracts.Contracts   
   
class PersonneDao(context: Context)   
{   
    //Récupération d'un objet BddHelper permettant de récupérer des   
    //connexions à la base de données   
    private val dbHelper= BddHelper(context)   
   
    /\*\*   
     \* Fonction permettant d'enregistrer une personne en base de données   
     \* Une requête INSERT sera envoyée à la bdd.   
     \*/   
    fun insert(personne:Personne):Long?   
    {   
        //Récupération d'un objet représentant une connexion à la   
        //base de données.   
        //Si la base de données n'existe pas alors elle est créée.   
        val db = dbHelper.writableDatabase   
        //Objet permettant d'enregister des données sous forme de   
        //clé/valeur   
        //La clé est le nom d'une colonne   
        //La valeur est la valeur à insérer dans la colonne de   
        //l'enregistrement.   
        val values = ContentValues().apply {   
            put(Contracts.Personnes.COLUMN\_NAME\_NOM, personne.nom)   
            put(Contracts.Personnes.COLUMN\_NAME\_PRENOM, personne.prenom)   
        }   
        //Appel de la fonction insert qui enregistre les données en base.   
        return db?.insert(Contracts.Personnes.TABLE\_NAME, null, values)   
    }   
   
    /\*\*   
     \* Fonction permettant de récupérer un objet de type Personne   
     \* grâce à son identifiant   
     \* Une requête SELECT sera envoyée à la bdd.   
     \*/   
    fun get(id:Int):Personne?   
    {   
        //Objet de type Personne permettant d'enregistrer les   
        //informations à retourner   
        var resultat:Personne?= null;   
        //Récupération d'un objet représentant une connexion en   
        //lecture sur la base de données.   
        //Si la base de données n'existe pas alors elle est créée.   
        val db = dbHelper.readableDatabase   
   
        //Définition de la projection de la requête SELECT   
        val projection =   
arrayOf(Contracts.Personnes.COLUMN\_NAME\_ID,   
Contracts.Personnes.COLUMN\_NAME\_NOM,   
Contracts.Personnes.COLUMN\_NAME\_PRENOM)   
   
        // Définition du WHERE de la requête SELECT   
        //Le WHERE contient un paramètre représenté par le "?"   
        val selection = "${Contracts.Personnes.COLUMN\_NAME\_ID} = ?"   
        //Valeur qui sera fournie au paramètre de la restriction WHERE   
        val selectionArgs = arrayOf("$id")   
   
        //Appel de la fonction query qui exécute le SELECT et   
        //retourne le résultat de la requête sous forme de cursor   
        val cursor = db.query(   
                Contracts.Personnes.TABLE\_NAME,   // Definition de la table   
dans laquelle a lieu la recherche (Définition du FROM)   
                projection,             // Définition du SELECT   
                selection,              // Définition du WHERE   
                selectionArgs,          // Valeur pour la clause WHERE   
                null,           // Pas de GROUP BY dans cette requête   
                null,            // Pas de HAVING dans cette requête   
                null             // Pas de tri dans cette requête   
        )   
   
        //Travail sur le curseur   
        with(cursor) {   
            //fonction moveToNext de l'objet cursor   
            if (moveToNext()) {   
                //Récupération de l'identifiant dans le résultat de la requête   
                val itemId =   
getInt(getColumnIndexOrThrow(Contracts.Personnes.COLUMN\_NAME\_ID))   
                //Récupération du nom dans le resultat de la requête   
                val itemNom =   
getString(getColumnIndexOrThrow(Contracts.Personnes.COLUMN\_NAME\_NOM))   
                //Récupération du prénom dans le résultat de la requête   
                val itemPrenom =   
getString(getColumnIndexOrThrow(Contracts.Personnes.COLUMN\_NAME\_PRENOM))   
                //Construction de l'objet de type Personne qui sera la   
                //valeur retournée.   
                resultat = Personne(itemId,itemNom,itemPrenom)   
            }   
        }   
        //Retourne le résultat   
        return résultat;   
    }   
   
    /\*\*   
     \* Fonction permettant de mettre à jour un enregistrement dans   
     \* la table personnes   
     \* Une requête UPDATE sera envoyée à la bdd.   
     \*/   
    fun update(item:Personne)   
    {   
        //Récupération d'un objet représentant une connexion en écriture   
        //sur la base de données.   
        //Si la base de données n'existe pas alors elle est créée.   
        val db = dbHelper.writableDatabase   
        //Objet permettant d'enregister des données sous forme de clé/valeur   
        //La clé est le nom d'une colonne   
        //La valeur est la valeur à mettre à jour dans la colonne   
        //de l'enregistrement.   
        val values = ContentValues().apply {   
            put(Contracts.Personnes.COLUMN\_NAME\_ID, item.id)   
            put(Contracts.Personnes.COLUMN\_NAME\_NOM, item.nom)   
            put(Contracts.Personnes.COLUMN\_NAME\_PRENOM, item.prenom)   
        }   
   
        // Définition du WHERE de la reqûete UPDATE   
        //Le WHERE contient un paramètre représenté par le "?"   
        val selection = "${Contracts.Personnes.COLUMN\_NAME\_ID} = ?"   
        //Valeur qui sera fournie au paramètre de la restriction WHERE   
        val selectionArgs = arrayOf("${item.id}")   
        //Appel de la fonction query qui exécute la requête UPDATE et   
        //retourne le nombre de lignes mises à jour   
        val count = db.update(   
                Contracts.Personnes.TABLE\_NAME,//Nom de la table à mettre   
à jour   
                values, //Valeurs à mettre à jour   
                selection, //Définition du WHERE   
                selectionArgs) //Valeurs à mettre dans le WHERE   
    }   
   
    /\*\*   
     \* Fonction permettant de supprimer un enregistrement dans la table   
     \* personnes   
     \* Une requête DELETE sera envoyée à la bdd.   
     \*/   
    fun delete(id:Int)   
    {   
        //Récupération d'un objet représentant une connexion en écriture   
        //sur la base de données.   
        //Si la base de données n'existe pas alors elle est créée.   
        val db = dbHelper.writableDatabase   
        // Définition du WHERE de la requête DELETE   
        //Le WHERE contient un paramètre représenté par le "?"   
        val selection = "${Contracts.Personnes.COLUMN\_NAME\_ID} = ?"   
        //Valeur qui sera fournie au paramètre de la restriction WHERE   
        val selectionArgs = arrayOf("$id")   
        //Appel de la fonction query qui exécute la requête DELETE   
        db.delete(Contracts.Personnes.TABLE\_NAME, selection, selectionArgs)   
    }   
}