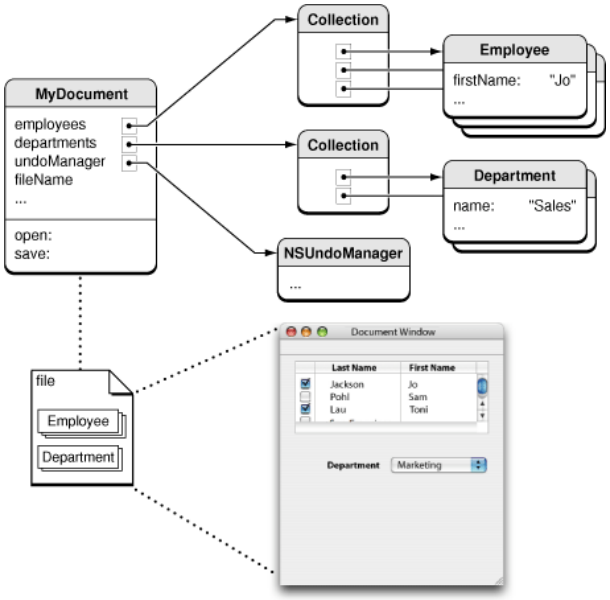
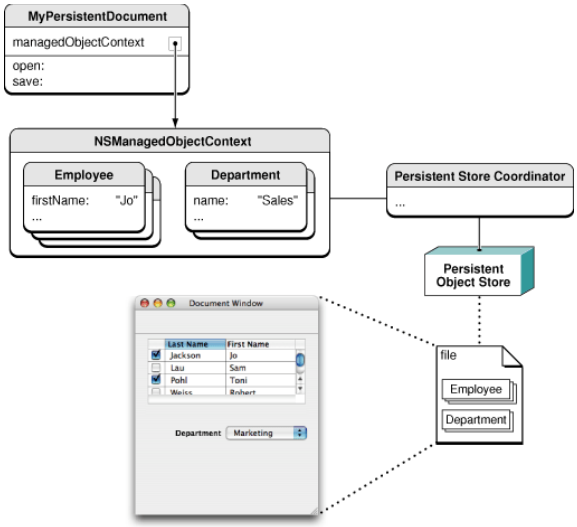
La persistance des données
Le modèle Core Data

Présentation de Core Data[1]

Dans la plupart des applications il est nécessaire d'archiver des objets dans un fichier, et si vous voulez prendre en charge l'annulation de modifications apportées, il faut pouvoir suivre les modifications apportées aux objets.
Par exemple, dans une application de gestion des employés, vous avez besoin d'un moyen d'ouvrir un fichier contenant un archivage des objets représentatifs des employés et de leurs services.



En utilisant le **Framework Core Data**, la plupart de ces fonctionnalités vous sont fournies, principalement par l'intermédiaire d'un objet nommé « objet de gestion du contexte », instance de la classe **NSManagedObjectContext**.
Cet objet sert de passerelle vers une collection sous-jacente des objets à gérer et à rendre persistants (pile des objets persistants). Il sert d'intermédiaire entre les objets de données de votre application et leur stockage externe, généralement dans une base de données fichier **SQLite**.



[1] Depuis le « Core Data Programming guide »
http://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/CoreData/Articles/cdBasics.html

Le contexte de gestion des objets « Managed Object Context »

Apple présente son « contexte de gestion des objets » comme un « bloc-notes intelligent » :

- ✓ lorsque vous chargez des objets issus d'un magasin persistant, cela revient à vous apporter des copies temporaires de ceux-ci sur le bloc-notes où elles forment un graphique d'objet,
- ✓ vous pouvez ensuite modifier ces objets sans que cela ne modifie le magasin persistant,
- ✓ la répercussion des changements dans le magasin n'aura lieu qu'au moment de la sauvegarde des objets du bloc note.

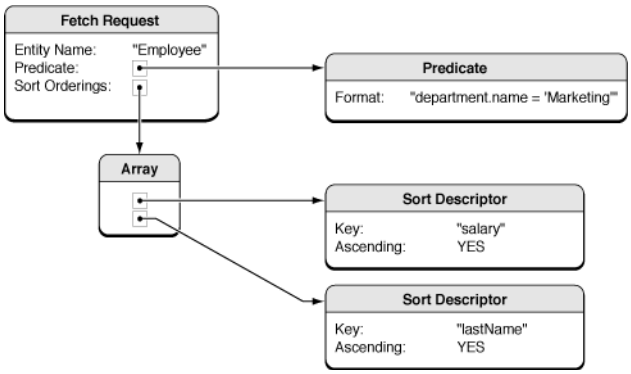
Les objets rendus persistants sont appelés « objets gérés », tous les objets gérés doivent être inscrits avec un « contexte de gestion des objets » ou simplement « contexte ».
Vous ajoutez ou supprimez des objets gérés au graphique d'objet en utilisant le contexte. Le contexte suit les modifications que vous apportez, tant aux attributs des objets individuels que pour les relations entre les objets.
Il assure le suivi des modifications, il est donc capable d'assurer l'annulation/rétablissement de modifications.
Il garantit également que si vous modifiez les relations entre les objets, l'intégrité du graphique d'objets sera maintenue.

Recherche des objets « Fetch Request »

Pour récupérer des données à l'aide d'un contexte, vous devez créer une requête de récupération. Cette requête utilise un objet de la classe **NSFetchRequest** qui spécifie quelles données vous souhaitez récupérer.
Par exemple, « tous les employés » ou « tous les employés du service de commercialisation ordonnés selon le salaire décroissant » (en vue d'un « dégraissage » par exemple).

Une demande d'extraction comporte généralement trois parties :

- ✓ le nom d'une entité d'objet, classe **NSEntityDescription**, correspondant au type du (ou des) objet(s) recherché(s),
- ✓ un objet « prédicat » optionnel qui spécifie les conditions de recherche, classe **NSPredicate**,
- ✓ un tableau optionnel de « consignes de rangement » des résultats obtenus, classe **NSSortDescriptor**.



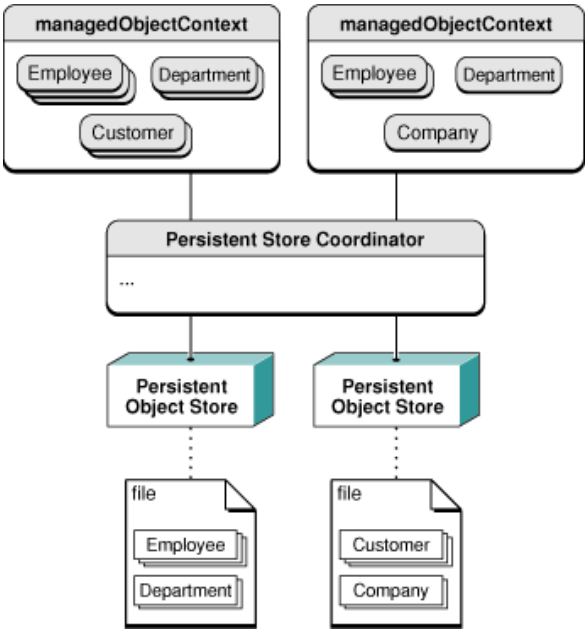
La requête de récupération est transmise à l'objet en charge du contexte pour exécution qui retourne le (ou les) objet(s) qui correspond(ent) à la demande (peut-être aucun).
Les objets retournés par une extraction sont automatiquement inscrits dans le contexte des objets gérés que vous avez utilisé pour l'extraction.

Si un contexte contient déjà un objet géré pour un objet compris dans une requête, alors l'objet géré existant est renvoyé dans les résultats de l'extraction.

Le coordinateur « Persistent Store Coordinator »

Entre les objets chargés et gérés par un contexte et les magasins des objets persistants (la base **SQLite**), se trouve un coordinateur de magasin persistant « **Persistent Store Coordinator** ».

Le coordinateur est conçu pour présenter une façade aux contextes, afin qu'un groupe de magasins persistants apparaisse comme un seul magasin global vis-à-vis des contextes.
Un objet contexte peut alors créer un graphe d'objets basé sur l'union de tous les magasins de données connectés au coordinateur.
La classe **NSPersistentStoreCoordinator** représente la fonction de coordinateur.



Les magasins persistants « Persistent Stores »

Un magasin d'objets persistants est associé à un unique fichier de données. Normalement, la seule interaction que vous avez avec ce magasin intervient lorsque vous spécifiez son emplacement au coordinateur.

La classe coordinateur **NSPersistentStoreCoordinator** fournit le support natif de plusieurs formats de fichiers de données :

- ✓ fichier de base **SQLite**, type **NSSQLiteStoreType**,
- ✓ fichier binaire, type **NSBinaryStoreType**,
- ✓ fichier XML, type **NSXmlStoreType**,
- ✓ fichier en mémoire, type **NSInMemoryStoreType**.

Le modèle permet, si à un moment donné vous décidez de choisir un format de fichier de données différent, de maintenir votre application fonctionnelle sans avoir à modifier son architecture.

Remarque importante : bien que **SQLite** soit pris en charge et utilisé comme mode de stockage le plus courant avec **Core Data**, le modèle ne peut pas gérer toute base de données **SQLite** arbitraire. Afin d'utiliser une base de données **SQLite**, le modèle doit créer et gérer la base de données lui-même.

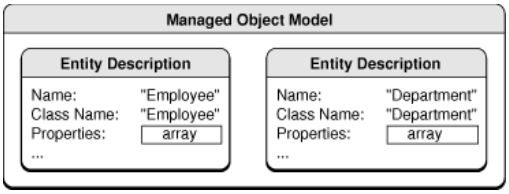
Les documents persistants

Pour gérer la persistance au sein d'une application il est nécessaire de créer et configurer la chaîne des objets nécessaires à celle-ci (**NSManagedObjectContext**, **NSPersistentStoreCoordinator**, **NSFetchRequest**, ...).

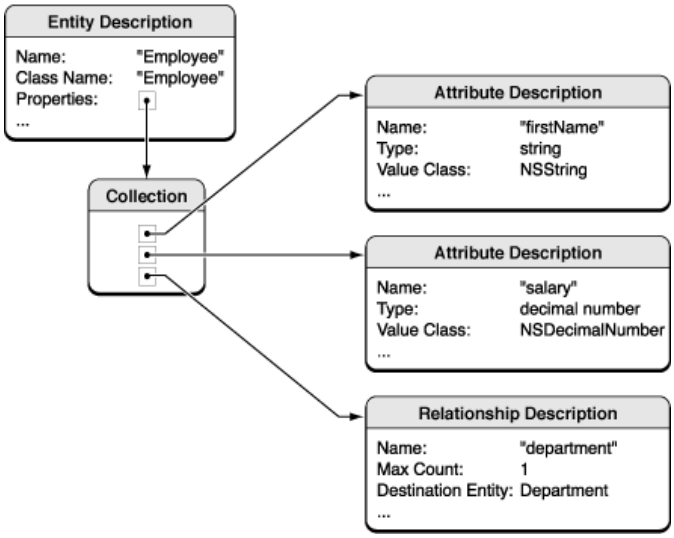
Dans certains cas, cependant, la persistance peut uniquement s'appliquer à des fichiers document. La classe **NSPersistentDocument** est une classe dérivée de **NSDocument** et complète celle-ci des fonctionnalités de persistance via **Core Data**.
Par défaut, une instance de **NSPersistentDocument** crée sa propre pile de persistance prête à l'emploi, y compris un contexte objet managé et un magasin unique objet persistant. Il y a en l'espèce un mappage biunivoque entre un document et un magasin de données externes.
Par défaut, vous n'avez pas à écrire de code supplémentaire pour gérer la persistance de l'objet.

Le modèle des objets gérés « Managed Object Model »

Le modèle d'objets gérés est un schéma qui fournit une description des objets gérés, ou entités, utilisées par votre application.
Ce modèle est généralement créé graphiquement depuis XCode à l'aide d'un fichier « Data Model » d'extension **xcdatamodeld**.



Les objets persistants ou entités correspondent à des tables contenant des attributs eux mêmes paramétrés.



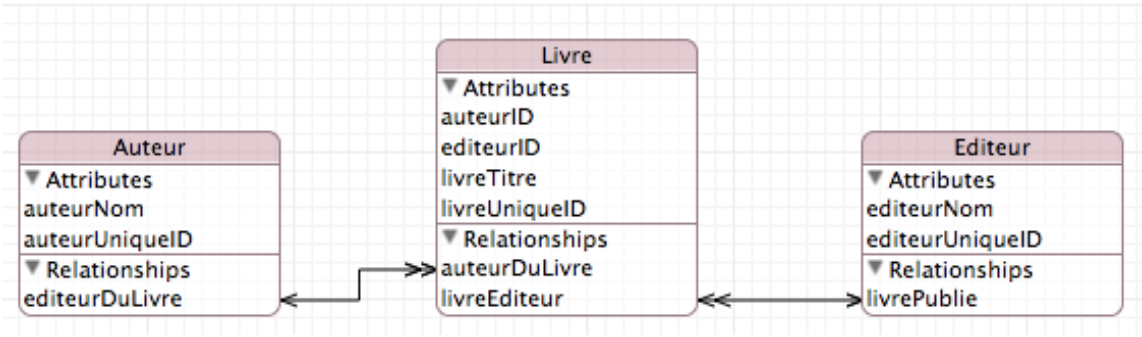
Chaque entité représente alors un objet géré persistant, le code des classes associées représentant les objets qui seront chargés via le contexte est généré par XCode, ces classes doivent hériter directement ou non de la classe **NSManagedObject**.

Exemple, enfin…

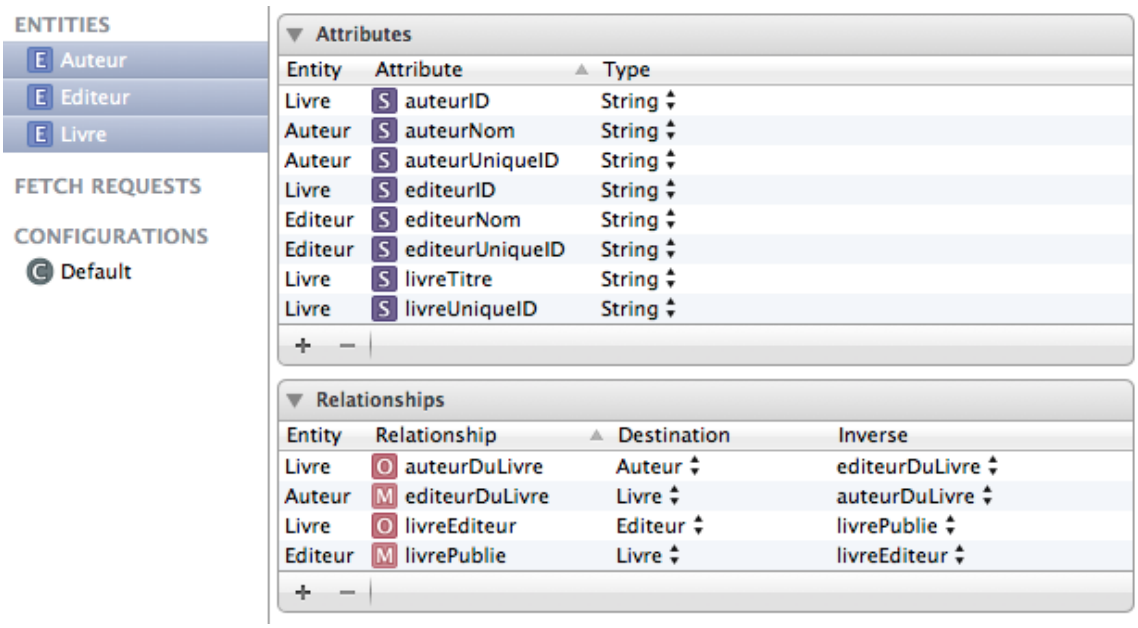
Pour cet exemple nous allons considérer une application de gestion d'un catalogue de livres. Ce catalogue simplifié fera appel à des objets **Livre**, **Editeur** et **Auteur**. Ces objets devront bien entendu être persistants.
On considère pour cet exemple :

- ✓ qu'à un livre est lié un unique auteur et un unique éditeur,
- ✓ qu'un auteur peut avoir écrit un nombre non limité de livres,
- ✓ qu'un éditeur peut avoir édité un nombre non limité de livres.

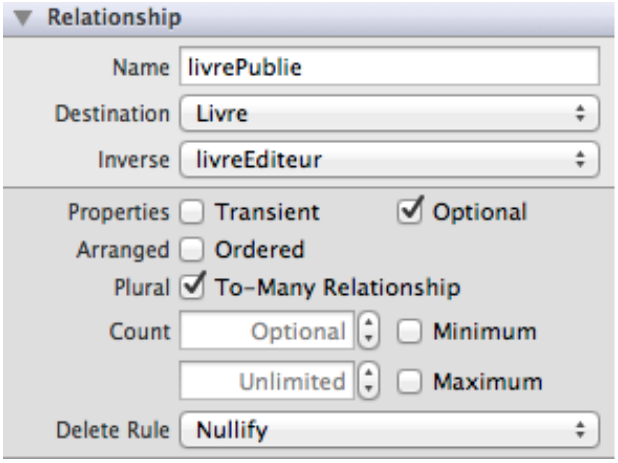
La première chose à faire depuis **XCode** consiste à établir le schéma des objets gérés à l'aide de l'outil graphique donnant lieu à un fichier de modèle d'extension **xcdatamodeld**. La vue suivante représente le schéma de notre exemple :



La définition des entités quant à leurs attributs et types associés ainsi que les relations entre entités :



Et le détail de la configuration d'une relation, en l'occurrence la relation « **livrePublie** » :



Une fois le modèle de données établi, il suffit de sélectionner le menu **Editor->Create NSManagedObject SubClass** pour faire générer les classes associées aux entités du modèles.
Ces classes nous serviront à charger, modifier, sauvegarder les données via un contexte qui reste à créer.

L'exemple ci-dessous montre le fichier **Auteur.h** généré, modèle de l'entité ou objet géré « **Auteur** » qui hérite de la classe **NSManagedObject**.

```
#import <Foundation/Foundation.h>
#import <CoreData/CoreData.h>

@class Livre;

@interface Auteur : NSManagedObject

@property (nonatomic, retain) NSString * auteurNom;
@property (nonatomic, retain) NSString * auteurUniqueID;
@property (nonatomic, retain) NSSet *editeurDuLivre;
@end

@interface Auteur (CoreDataGeneratedAccessors)

- (void)addEditeurDuLivreObject:(Livre *)value;
- (void)removeEditeurDuLivreObject:(Livre *)value;
- (void)addEditeurDuLivre:(NSSet *)values;
- (void)removeEditeurDuLivre:(NSSet *)values;

@end
```

On remarquera les propriétés associées aux attributs d'un livre et les méthodes permettant d'ajouter/supprimer un livre et un éditeur du livre.

La sauvegarde des données se fera dans un magasin persistant du type **SQLite** et le fichier correspondant sera, pour cet exemple, nommé **MM_DataBase_Livres** et sera stocké dans le dossier **/Documents** de l'application.

Le modèle de données étant créé, il faut à présent construire le contexte apte à charger les entités.
Pour rappel :

- ✓ l'objet contexte est du type **NSManagedObjectContext**,
- ✓ un objet coordinateur de type **NSPersistentStoreCoordinator** doit servir d'interface entre le contexte et le magasin d'objets persistants.

```
 // Déclaration d'un objet contexte
 NSManagedObjectContext* managedObjectContext;

 // Méthode retournant le chemin du magasin persistant
-(NSURL*)getCoreDataBaseUrl{
NSURL *url = [[[NSFileManager defaultManager]
                URLsForDirectory:NSDocumentDirectory
                inDomains:NSUserDomainMask] lastObject];
NSURL *storeDatabasedUrl= [url
                URLByAppendingPathComponent:@"MM_Database_Livres"];
return storeDatabasedUrl;
}

// Méthode retournant l'objet contexte initialize
- (NSManagedObjectContext *) managedObjectContext {
      if(managedObjectContext == nil)
          {
           // Lecture du chemin du magasin persistant
           NSURL *storeDatabasedUrl = [self getCoreDataBaseUrl];
           NSError *error = nil;
           // Création du coordinateur a partir du modèle recherché dans
           // le paquetage de l'application
           NSPersistentStoreCoordinator *persistentStoreCoordinator
           = [[NSPersistentStoreCoordinator alloc]
                initWithManagedObjectModel:[NSManagedObjectModel
                mergedModelFromBundles:nil]];
           // Choix du type de magasin SQLite
           if (![persistentStoreCoordinator
                addPersistentStoreWithType:NSSQLiteStoreType
                configuration:nil URL:storeDatabasedUrl options:nil
                error:&error])
              {
                 NSLog(@"Erreur de chargement du magasin SQLite.");
              }
          // Création du contexte et association au coordinateur
          managedObjectContext=[[NSManagedObjectContext alloc]init];
          [managedObjectContext
          setPersistentStoreCoordinator:persistentStoreCoordinator];
          }
        return managedObjectContext;
      }
```

Le contexte étant créé il ne reste plus qu'à utiliser les entités pour lire, modifier, sauvegarder les données.

Exemple de création des objets Auteur, Editeur et Livre pour sauvegarde dans le magasin :

```
      Auteur *nouvelAuteur = (Auteur *) [NSEntityDescription
           insertNewObjectForEntityForName:@"Auteur"
           inManagedObjectContext:self.managedObjectContext];
      nouvelAuteur.auteurNom = @"M.M.Corporation";
      nouvelAuteur.AuteurUniqueID = @"1";

      Editeur *nouvelEditeur = (Editeur *) [NSEntityDescription
           insertNewObjectForEntityForName:@"Editeur"
           inManagedObjectContext:self.managedObjectContext];
      nouvelEditeur.editeurNom = @"@Apple Inc.";
      nouvelEditeur.editeurUniqueID = @"1";

      Livre *nouveauLivre1 = (Livre *) [NSEntityDescription
```

```
        insertNewObjectForEntityForName:@"Livre"
        inManagedObjectContext:self.managedObjectContext];
    nouveauLivre1.livreUniqueID = @"1001";
    nouveauLivre1.livreTitre = @"La persistance des données avec
    Core Data";
    nouveauLivre1.AuteurID = @"1";
    nouveauLivre1.EditeurID = @"1";
    if ([self.managedObjectContext hasChanges])
        {
            [self.managedObjectContext save:nil];
        }
```

Exemple de lecture des objets Auteur, puis les livres et éditeurs associés pour chaque auteur du magasin :

Rappel : la recherche dans le magasin nécessite l'utilisation de la classe **NSFetchRequest**. Les auteurs sont rangés dans l'ordre alphabétique à l'aide d'un objet de la classe **NSSortDescriptor**.

```
NSFetchRequest * request = [[NSFetchRequest alloc] init];
NSEntityDescription *myEntityQuery =
        [NSEntityDescription entityForName:@"Auteur"
        inManagedObjectContext:[self
              getAppDelegateRef].managedObjectContext];
NSSortDescriptor* sortAuteur = [[NSSortDescriptor alloc]
              initWithKey:@"auteurNom" ascending:YES];
NSArray *sortArray = [[NSArray alloc]initWithObjects:sortAuteur, nil];
[request setEntity:myEntityQuery];
[request setSortDescriptors:sortArray];

NSError *error = nil;
NSArray *auteurs = [[self getAppDelegateRef].managedObjectContext
        executeFetchRequest:request error:&error];

// Selection des auteurs
NSInteger nbrAuteurs = [auteurs count];
for (int loop = 0; loop < nbrAuteurs; loop++)
{
    Auteur *auteur = (Auteur *)[auteurs objectAtIndex:loop];
    NSLog(@"Le nom de l'auteur est : %@", auteur.auteurNom);
    NSLog(@"L'identifiant unique de l'auteur est : %@",
    auteur.auteurUniqueID);

    // Recherche des livres de l'auteur
    NSArray *livres = [auteur.editeurDuLivre allObjects];
    NSInteger nbrLivres = [livres count];
    for (int innerLoop = 0; innerLoop < nbrLivres; innerLoop++)
       {
        Livre *livre = (Livre *)[livres objectAtIndex:innerLoop];
        NSLog(@"L'identifiant unique du livre est   : %@",
           livre.livreUniqueID);
        NSLog(@"Le titre du livre est               : %@",
           livre.livreTitre);
        NSLog(@"L'auteur du livre est               : %@",
           livre.auteurID);
        NSLog(@"L'éditeur du livre est              : %@\n\n",
           livre.editeurID);
       }
       if(loop == 0)
NSLog(@"#########################################################\n\n;");
}
```

L'exemple précédent mais avec un predicat pour la recherche des livres par auteur :

```
// Selection des auteurs
NSInteger nbrAuteurs = [auteurs count];
for (int loop = 0; loop < nbrAuteurs; loop++)
{
      Auteur *auteur = (Auteur *)[auteurs objectAtIndex:loop];
      NSLog(@"Le nom de l'auteur est : %@", auteur.auteurNom);
      NSLog(@"L'identifiant unique de l'auteur est : %@",
      auteur.auteurUniqueID);

      // Recherche des livres de l'auteur
      NSEntityDescription *myEntityQuery = [NSEntityDescription
          entityForName:@"Livre" inManagedObjectContext:[self
                 getAppDelegateRef].managedObjectContext];
      NSPredicate* myPredicateQuery = [NSPredicate
           predicateWithFormat:@"auteurID like %@",
      auteur.auteurUniqueID];
      NSSortDescriptor* sortTitre = [[NSSortDescriptor alloc]
           initWithKey:@"livreTitre" ascending:YES];
      NSArray *sortArray = [[NSArray alloc]initWithObjects:sortTitre,
      nil];
      [request setEntity:myEntityQuery];
      [request setSortDescriptors:sortArray];
      [request setPredicate:myPredicateQuery];

      NSError *error = nil;
      NSArray *livres = [[self getAppDelegateRef].managedObjectContext
          executeFetchRequest:request error:&error];

      NSInteger nbrLivres = [livres count];
      for (int innerLoop = 0; innerLoop < nbrLivres; innerLoop++)
         {
          Livre *livre = (Livre *)[livres objectAtIndex:innerLoop];
          NSLog(@"L'identifiant unique du livre est   : %@",
             livre.livreUniqueID);
          NSLog(@"Le titre du livre est               : %@",
             livre.livreTitre);
          NSLog(@"L'auteur du livre est               : %@",
             livre.auteurID);
          NSLog(@"L'éditeur du livre est              : %@\n\n",
             livre.editeurID);
         }
         if(loop == 0)
NSLog(@"##########################################################\n\n;");
}
```