

# Desarrollo de Aplicaciones iOS

PARTE II  
Sergio Padrino Recio

# Temario

- Core Data
- Localización, manejo de mapas
- Sensores
- Notificaciones locales y notificaciones push
- Compras In-App

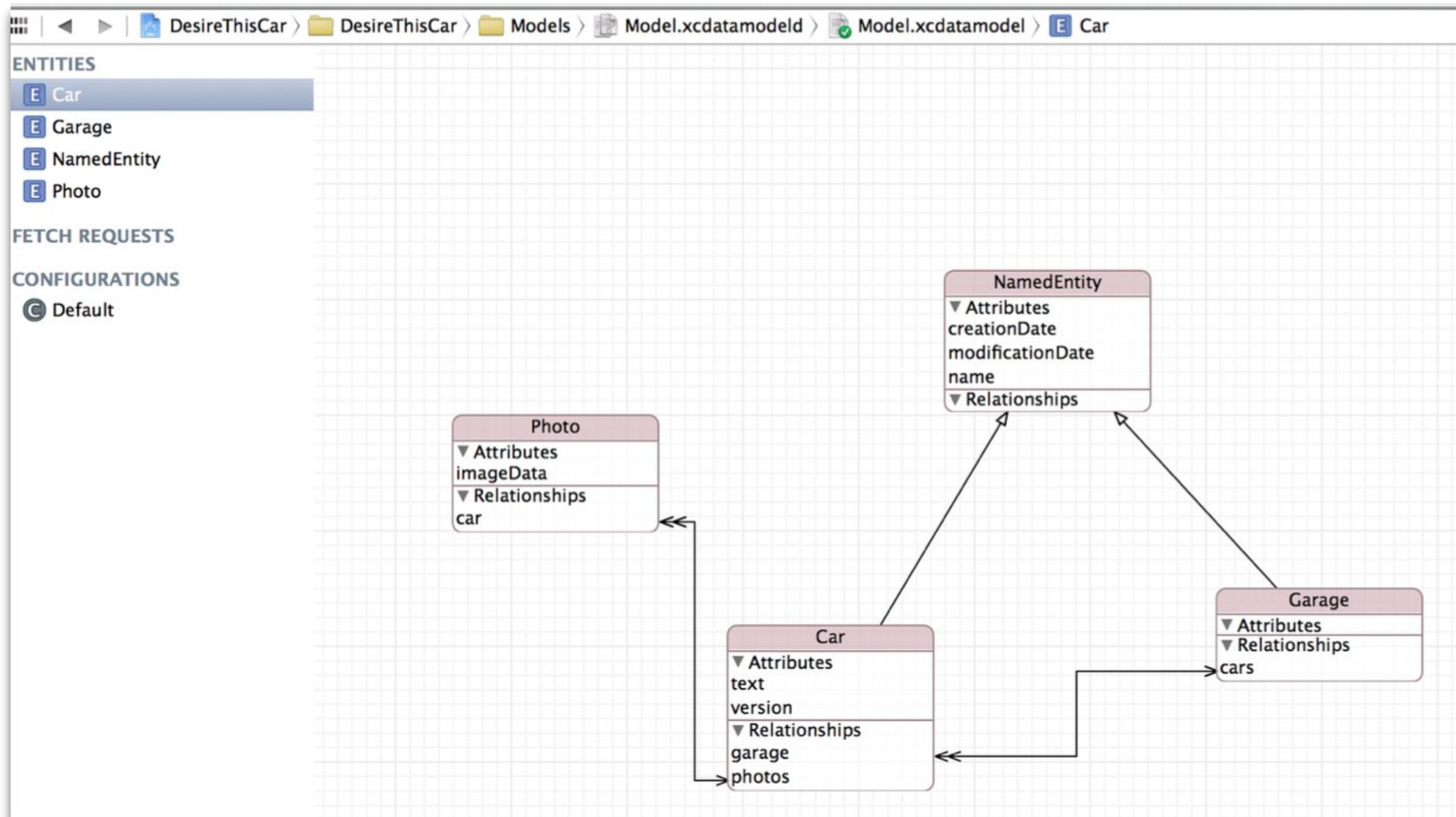
# Core Data

# ¿Qué es Core Data?

- Framework de Apple para manejo de datos.
- Gestión de ciclo de vida y grafo de objetos.
- Mapeo relacional de datos (principalmente persistencia en una base de datos).
- Extremadamente potente.

# Conceptos básicos

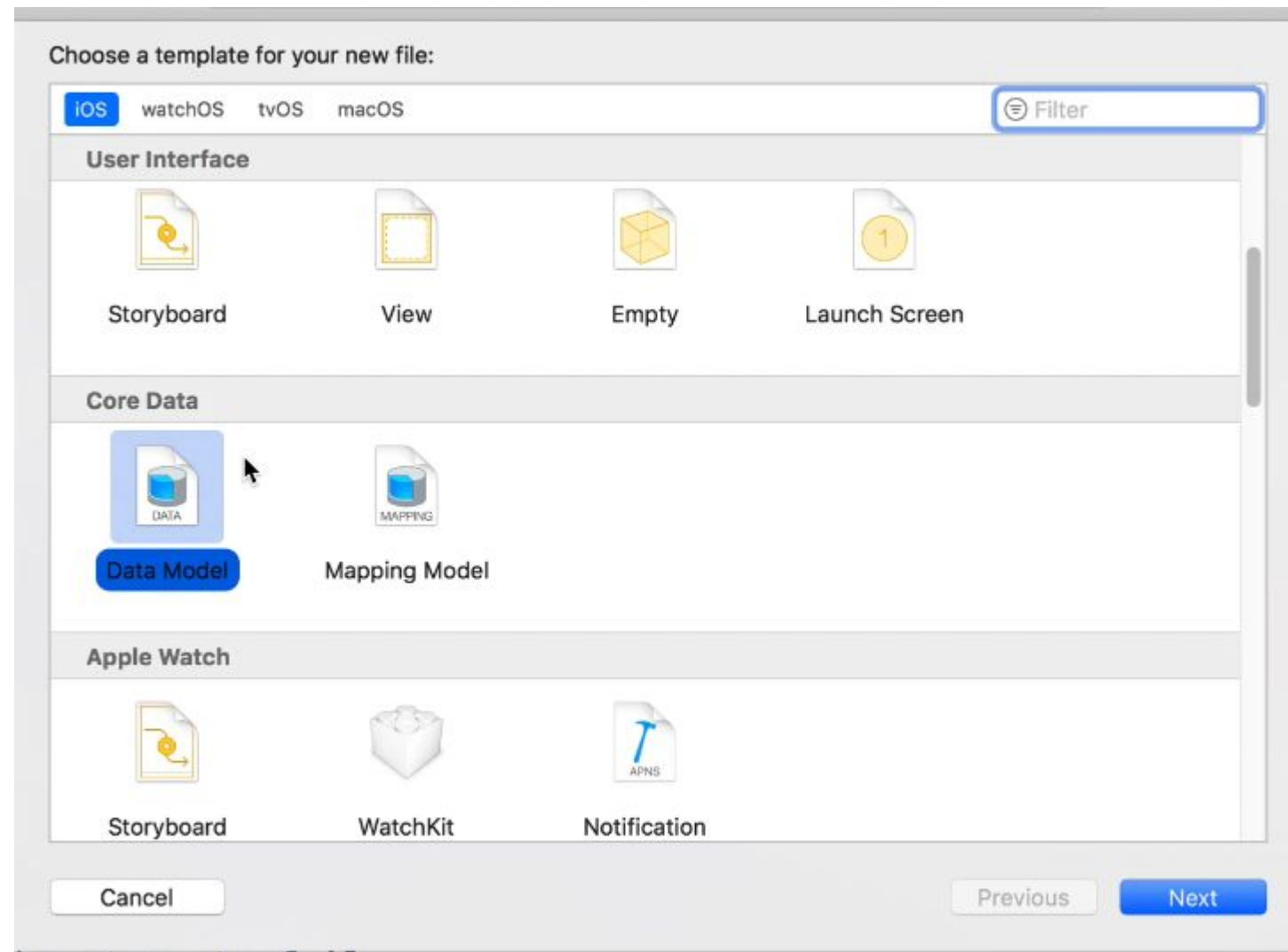
- **Entities:** clases del modelo (tablas de una base de datos).
- **Attributes:** propiedades de una entidad.
- **Relationships:** relaciones entre las entidades.



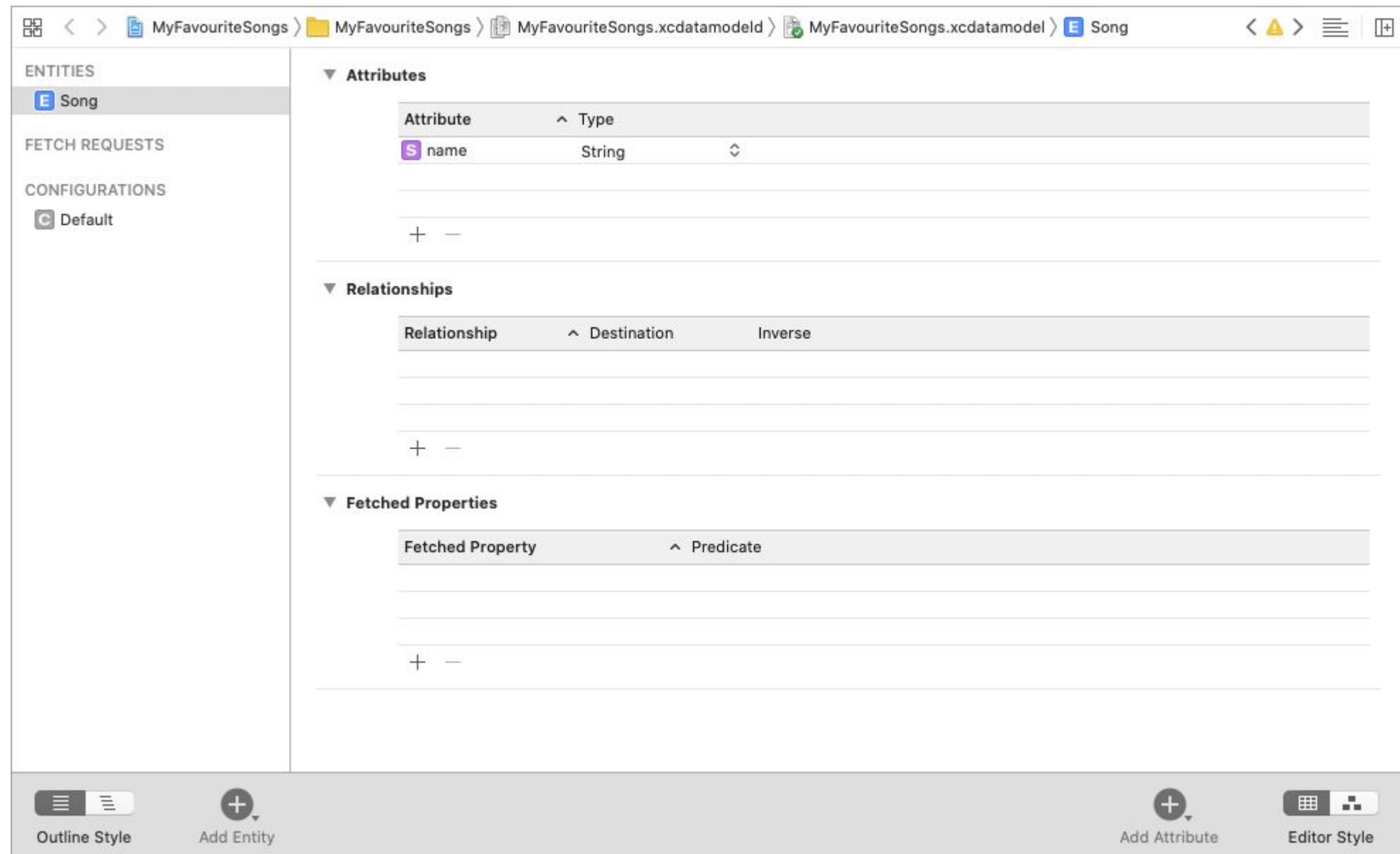
# Modelo en Core Data

DIAGRAMA ENTIDAD-RELACIÓN EN XCODE

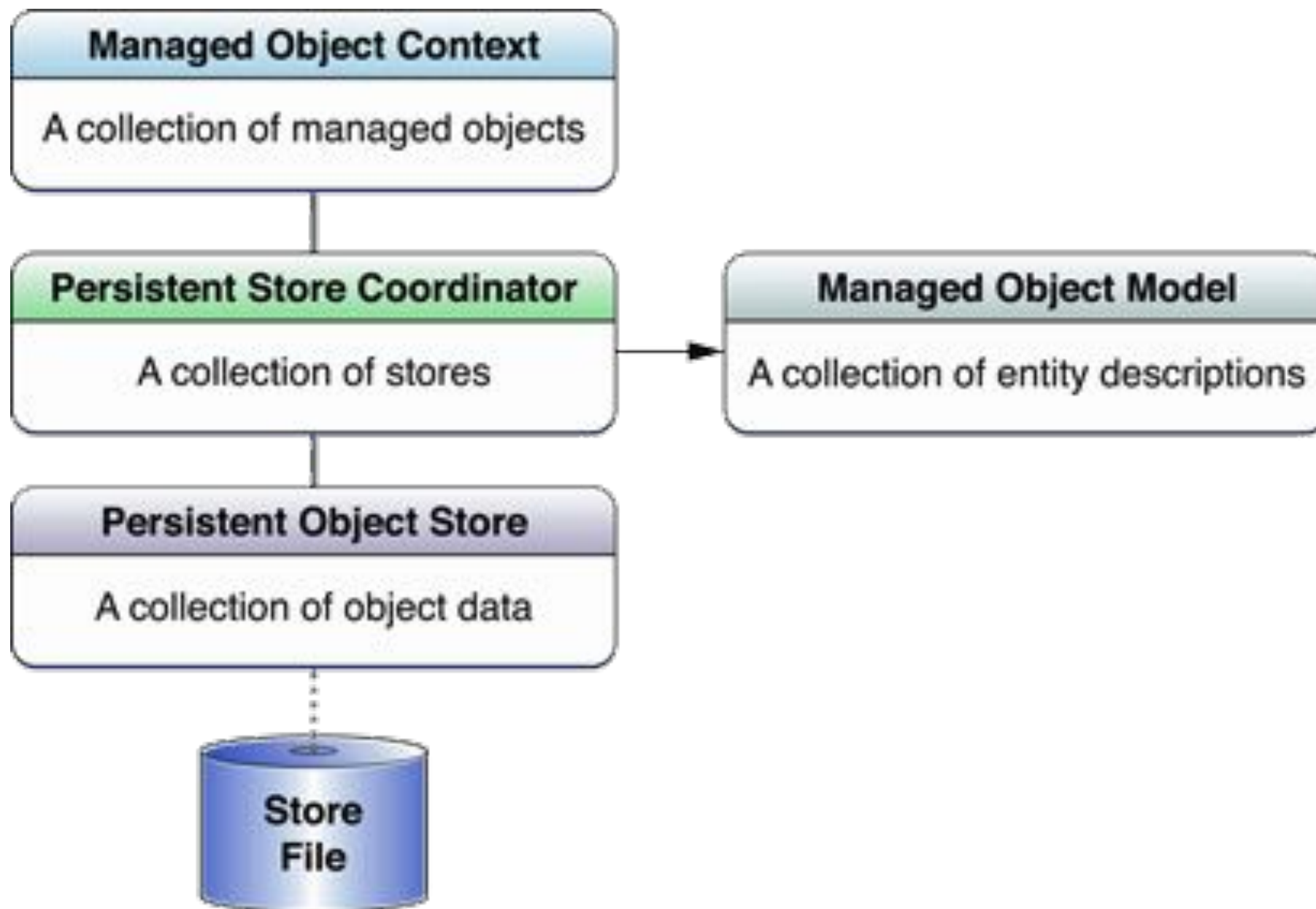
# Cómo crear una base de datos



# Cómo crear una base de datos







# Arquitectura de Core Data

# Managed Object Model (MOM)

Es el modelo de la app. Aquí es donde se definen las entidades, atributos y relaciones mencionadas anteriormente. Se define en un archivo de extensión .xcdatamodeld.

Se suele crear de forma gráfica (como un xib), pero también se puede hacer por código.

# Managed Object Context (MOC)

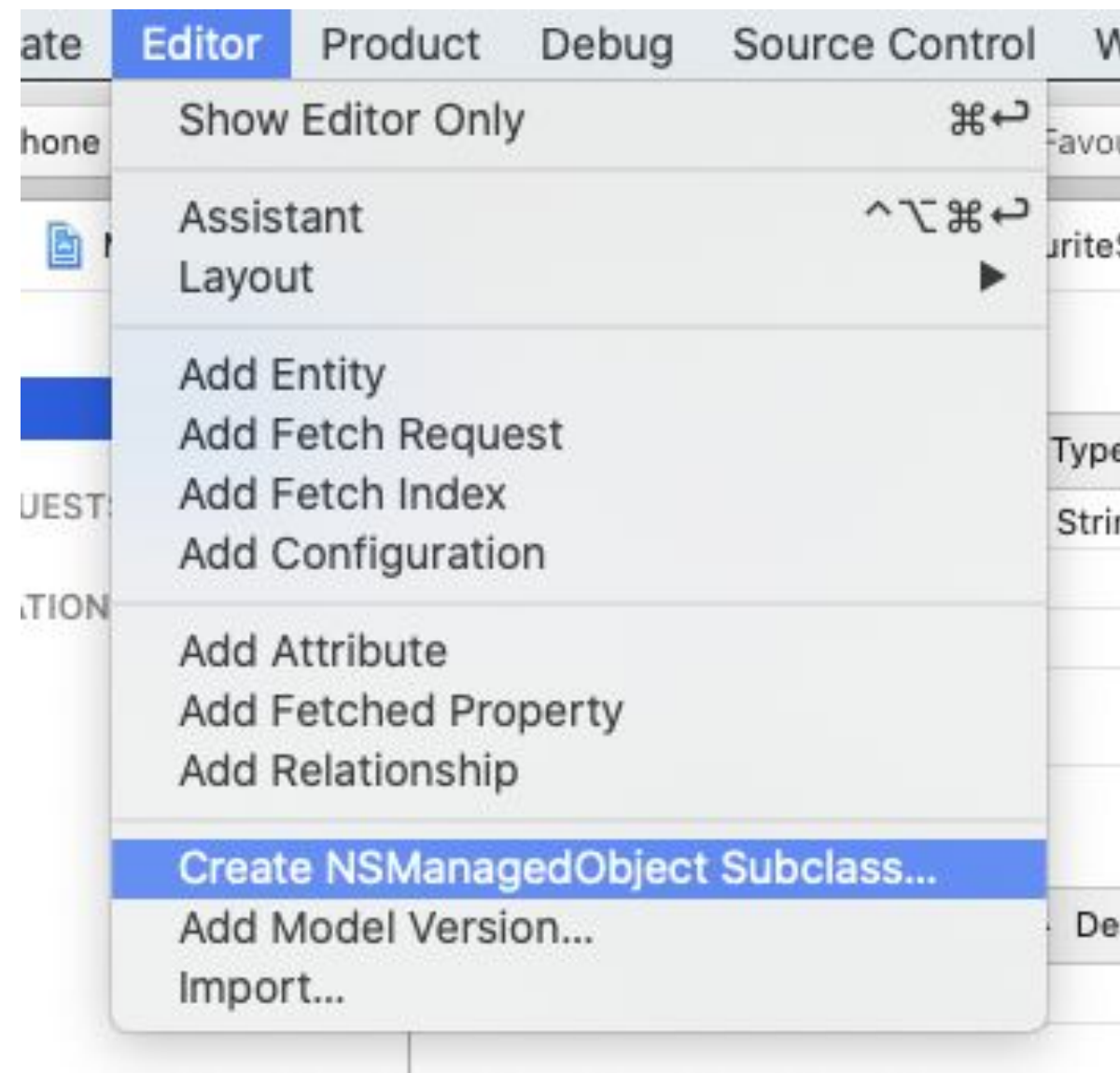
Es un “área” donde podemos modificar objetos antes de guardarlos a disco. Carga en memoria los objetos que necesitamos y nos permite interactuar con ellos (leerlos, guardarlos, eliminarlos...).

Se encarga de la integridad de los datos.

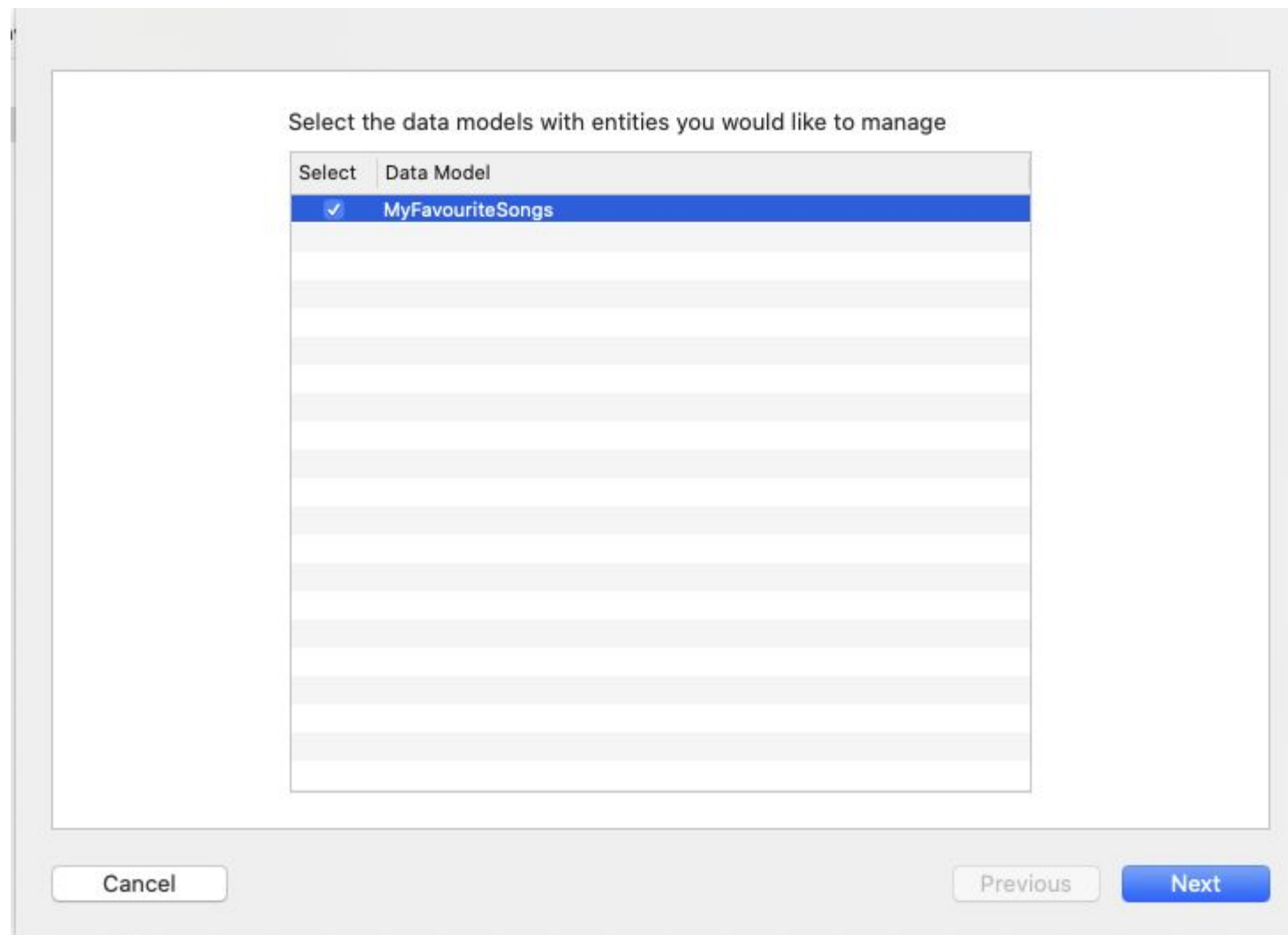
# NSManagedObject

- Es la clase base de los objetos almacenados con Core Data.
- Podemos acceder a sus datos utilizando clave-valor o creando nuestras propias subclasses.
- Nuestras subclasses tendrán propiedades y se podrán utilizar como objetos normales... ¡pero no lo son!

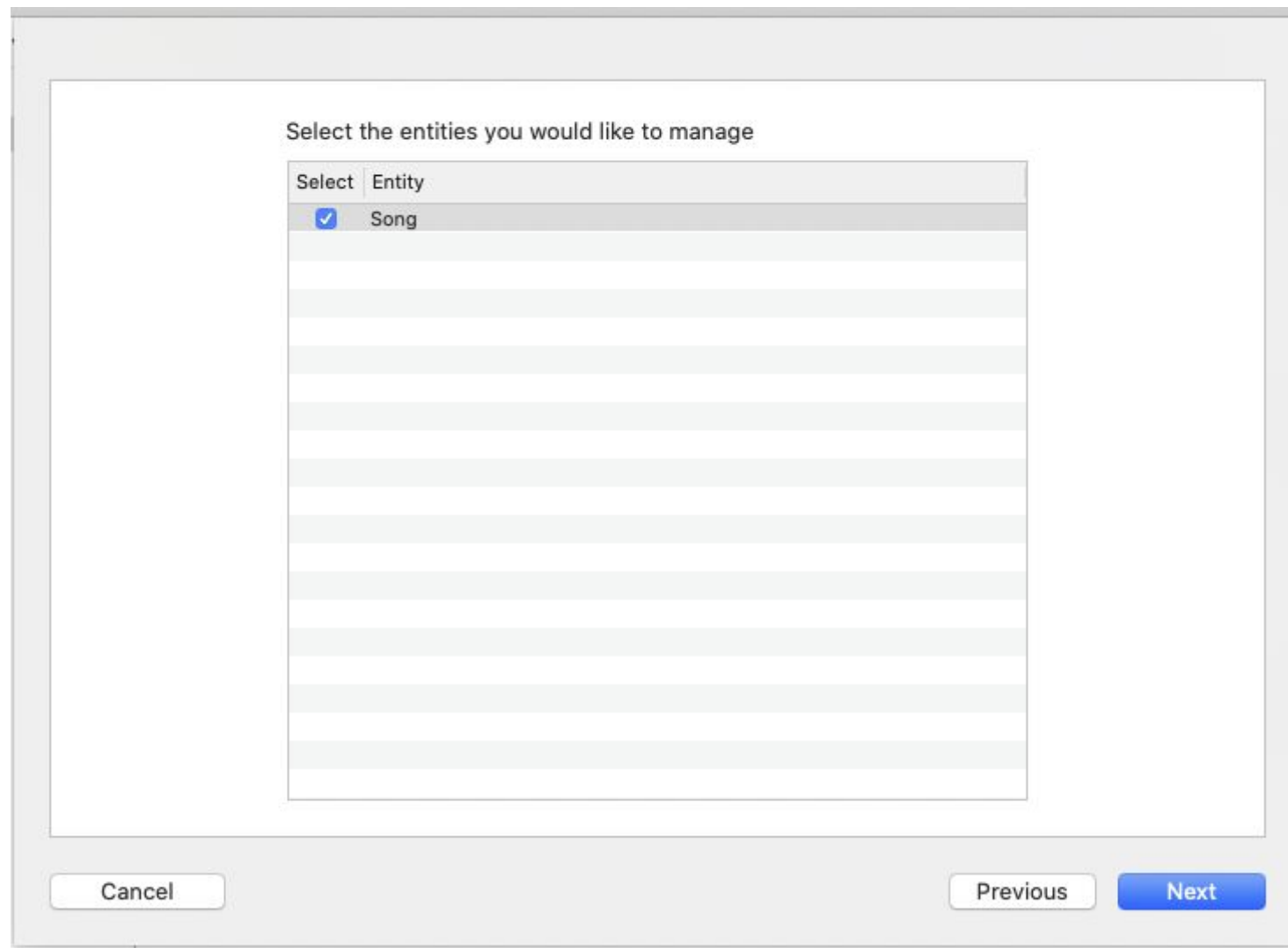
# Generador de NSManagedObject



# Generador de NSObject



# Generador de NSObject



# Generador de NSManagedObject

```
MyFavouriteSongs > Song+CoreDataProperties.h > No Selection
8 //
9
10 #import "Song+CoreDataClass.h"
11
12
13 NS_ASSUME_NONNULL_BEGIN
14
15 @interface Song (CoreDataProperties)
16
17 + (NSFetchRequest<Song *> *)fetchRequest;
18
19 @property (nullable, nonatomic, copy) NSString *name;
20
21 @end
22
23 NS_ASSUME_NONNULL_END
24
```



# Persistent Store (PS)

Los store son los almacenes, es decir, el lugar donde se guardan los datos de Core Data. Tipos:

- SQLite (NSQLiteStoreType)
- XML (NSXMLStoreType)
- Datos Binarios (NSBinaryStoreType)
- Memoria (NSInMemoryStoreType)

# Persistent Store Coordinator (PSC)

Se comunica con el MOC, y su labor es serializar (es decir, eliminar la concurrencia) las operaciones sobre nuestro modelo de datos.

Gracias a él no tenemos que preocuparnos de no corromper los datos. Es la pieza clave que relaciona los datos cargados en memoria (MOC), el acceso a disco, y el modelo que hemos creado (MOM).

# Fetch Request

- `NSFetchRequest`
- Es la clase que se encarga de representar el resultado de una búsqueda de objetos. Lo usaremos para buscar en la base de datos el conjunto de objetos que concuerden con un criterio particular.
- Ejemplo: “Dame todos los empleados que viven en Alabama y han estado en la empresa al menos 3 años”

# Ejemplos de búsquedas

- Obtener todos los objetos de una determinada entidad:

```
let fetchRequest = NSFetchRequest()  
let entity = NSEntityDescription.entity(forEntityName:  
"Person", in: managedObjectContext)  
fetchRequest.entity = entity
```

# Ejemplos de búsquedas

- Obtener todos los objetos de una determinada entidad (versión abreviada):

```
let fetchRequest = NSFetchRequest(entityName: "Person")
```

# Fetch request

- La propiedad `predicate` nos permite pasar un `NSPredicate` para filtrar el resultado de la consulta (equivalente a la cláusula `WHERE`).

```
fetchRequest.predicate = NSPredicate(format: "name == %@", "Ray")
```

# Ejemplos de búsquedas

- Usar búsqueda definida con nombre “peopleFR” en el MOM:

```
let fetchRequest = mom.fetchRequestTemplate(forName:  
“peopleFR”)
```

# Ejemplos de búsquedas

- Usar búsqueda definida con nombre “peopleFR” en el MOM, acotando la búsqueda con NAME=Ray:

```
let fetchRequest = mom.fetchRequestFromTemplate(withName:  
“peopleFR”, substitutionVariables: [“NAME” :“Ray”])
```



# Fetch request

- Con la propiedad `sortDescriptors` escogeremos el orden en que se nos devolverán los resultados (equivalente a `ORDER BY`).

```
let ageSortDescriptor = NSSortDescriptor(key: "age", ascending: true)
let nameSortDescriptor = NSSortDescriptor(
    key: "name",
    ascending: true
    selector: #selector(NSString.localizedStandardCompare(_:)))

fetchRequest.sortDescriptors = [ageSortDescriptor, nameSortDescriptor]
```

# Fetch request

- La propiedad `resultType` nos permite elegir el tipo de resultados:
  - `NSManagedObjectResultType`: Devuelve Managed Objects.
  - `NSCountResultType`: Devuelve el número de objetos que coinciden con nuestra búsqueda.
  - `NSDictionaryResultType`: Devuelve un diccionario formado por una o más propiedades que indiquemos de un objeto concreto.
  - `NSManagedObjectIDResultType`: Devuelve identificadores únicos, en vez de objetos completos.

# Fetch request

- Por defecto son síncronas / bloqueantes.
- Podemos usar `NSAsynchronousFetchRequest` para consultas asíncronas pasando un bloque que se ejecutará al finalizar la consulta.

```
asyncFetchRequest = NSAsynchronousFetchRequest(fetchRequest: fetchRequest)
{
    [unowned self] (result: NSAsynchronousFetchResult! ) -> Void in
        self.venues = result.finalResult
        self.tableView.reloadData()
}
```

# NSFetchedResultsController

Configurar nuestro fetch request para obtener el array de objetos y mostrarlos en una tabla es un escenario común.

Core Data nos proporciona controlador llamado NSFetchedResultsController que relaciona nuestros fetch request con un UITableView.

No es un ViewController (no existe interfaz de usuario).

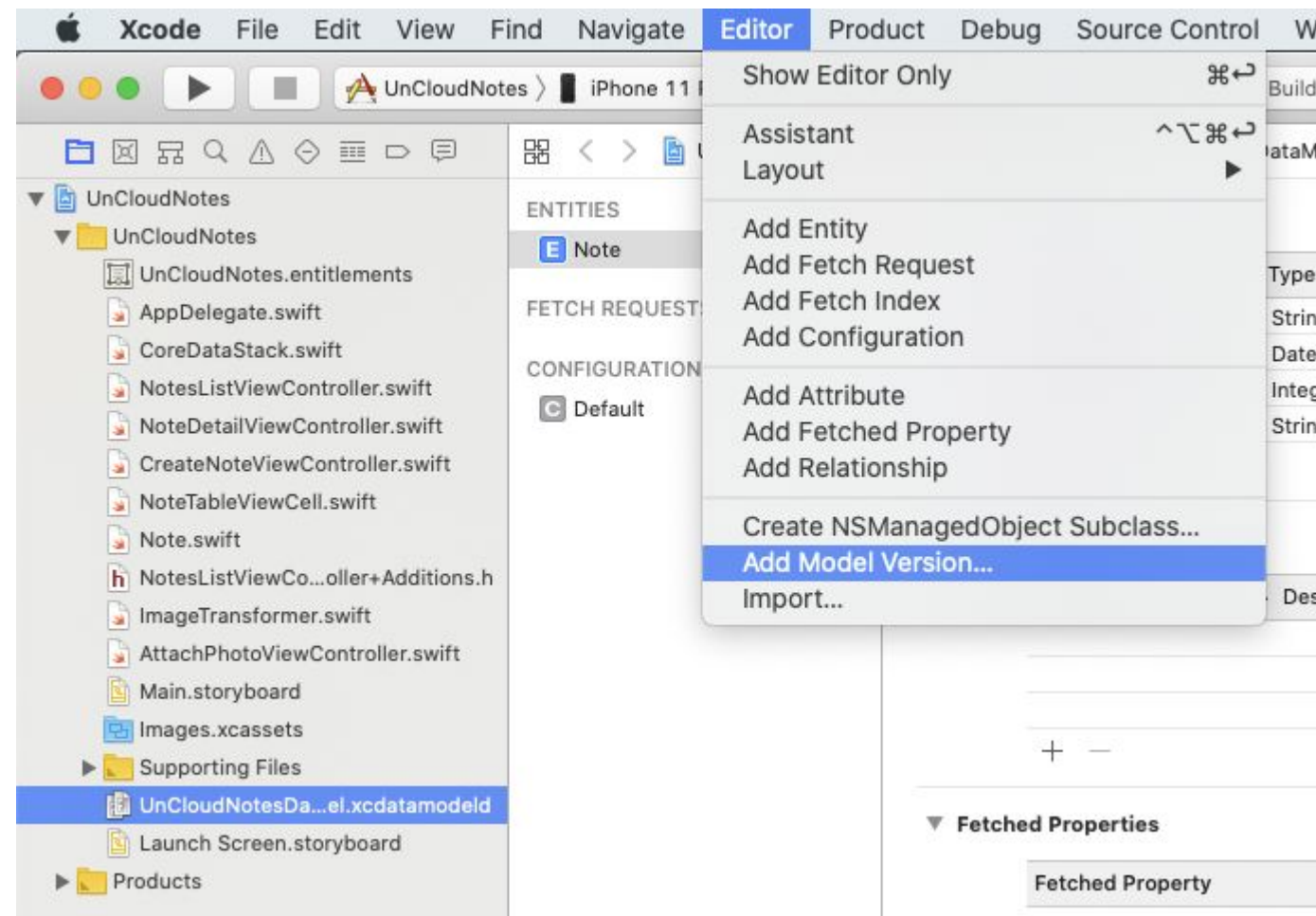
# NSFetchedResultsControllerDelegate

- `controllerWillChangeContent`: Notifica al delegado que se van a producir cambios en el resultado.
- `controller:didChange:at:for:newIndexPath`: Notifica al delegado qué objeto cambió exactamente, qué tipo de cambio se produjo y en qué índice.
- `controller:didChange:atSectionIndex:for:`: Similar al anterior, pero informa sobre cambios en las secciones, no en los objetos.
- `controllerDidChangeContent`: Notifica al delegado que se han producido cambios en el resultado.

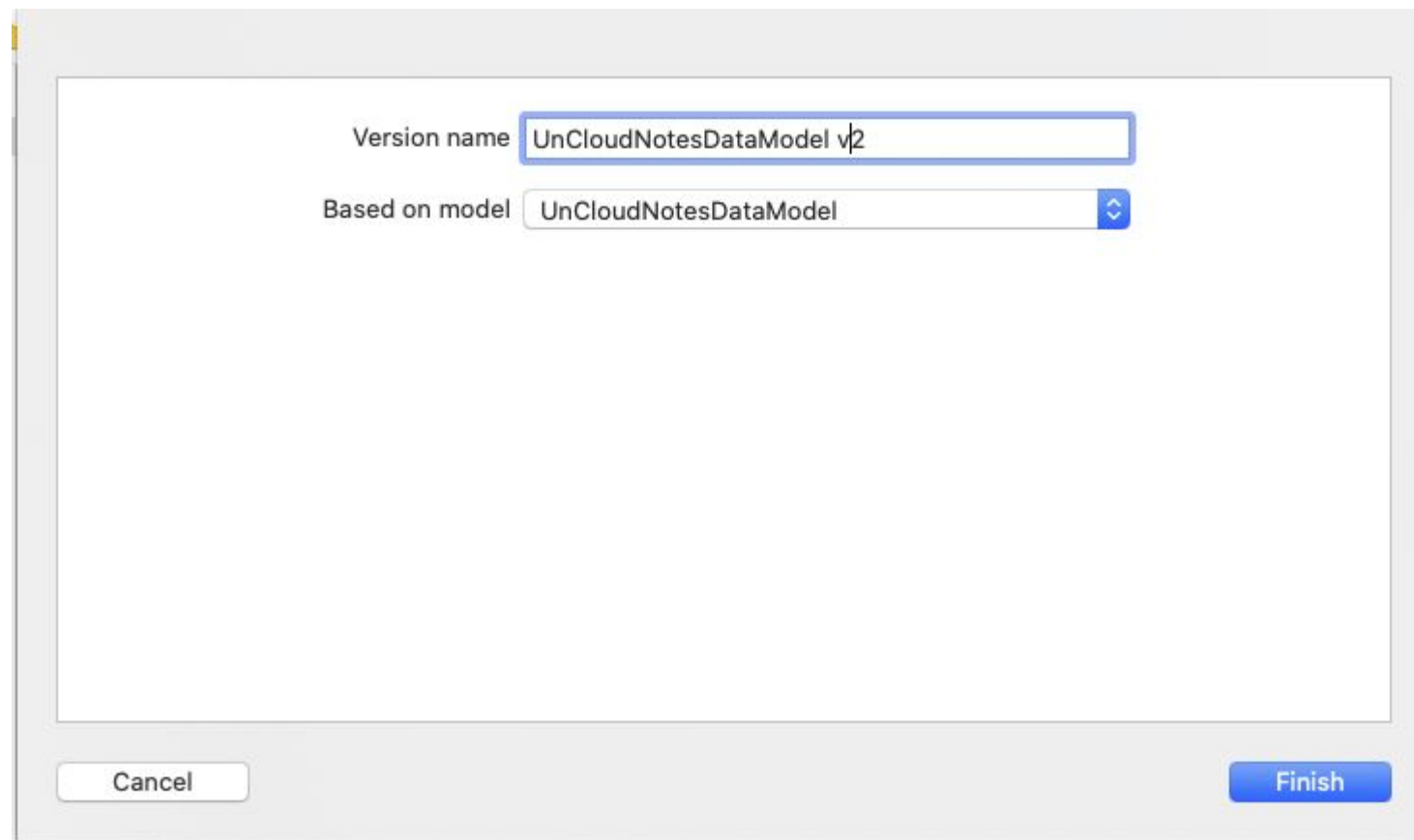
# Versionado del modelo

- A medida que desarrollamos funcionalidades, variaremos nuestro modelo de datos.
- Las variaciones en el modelo de datos son versionadas, y en cada versión haremos solo los cambios necesarios con respecto a la anterior.

# Versionado del modelo: pasos



# Versionado del modelo: pasos



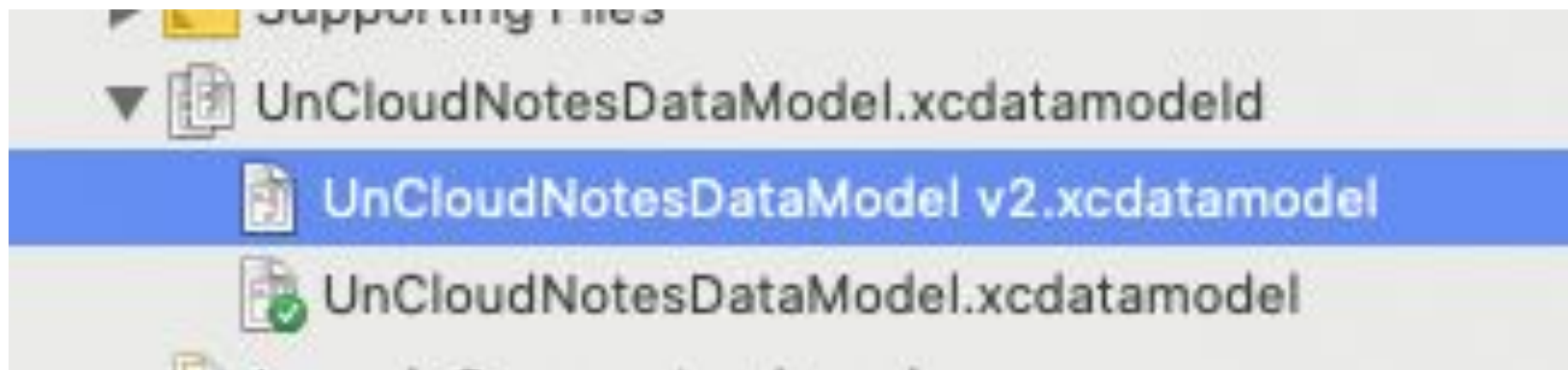
A screenshot of a software dialog box for creating a new version of a model. The dialog has a light gray border and a white background. It contains two input fields: "Version name" with the text "UnCloudNotesDataModel v2" and "Based on model" with the text "UnCloudNotesDataModel". The "Based on model" field has a blue dropdown arrow on its right. At the bottom, there are two buttons: "Cancel" on the left and "Finish" on the right.

Version name

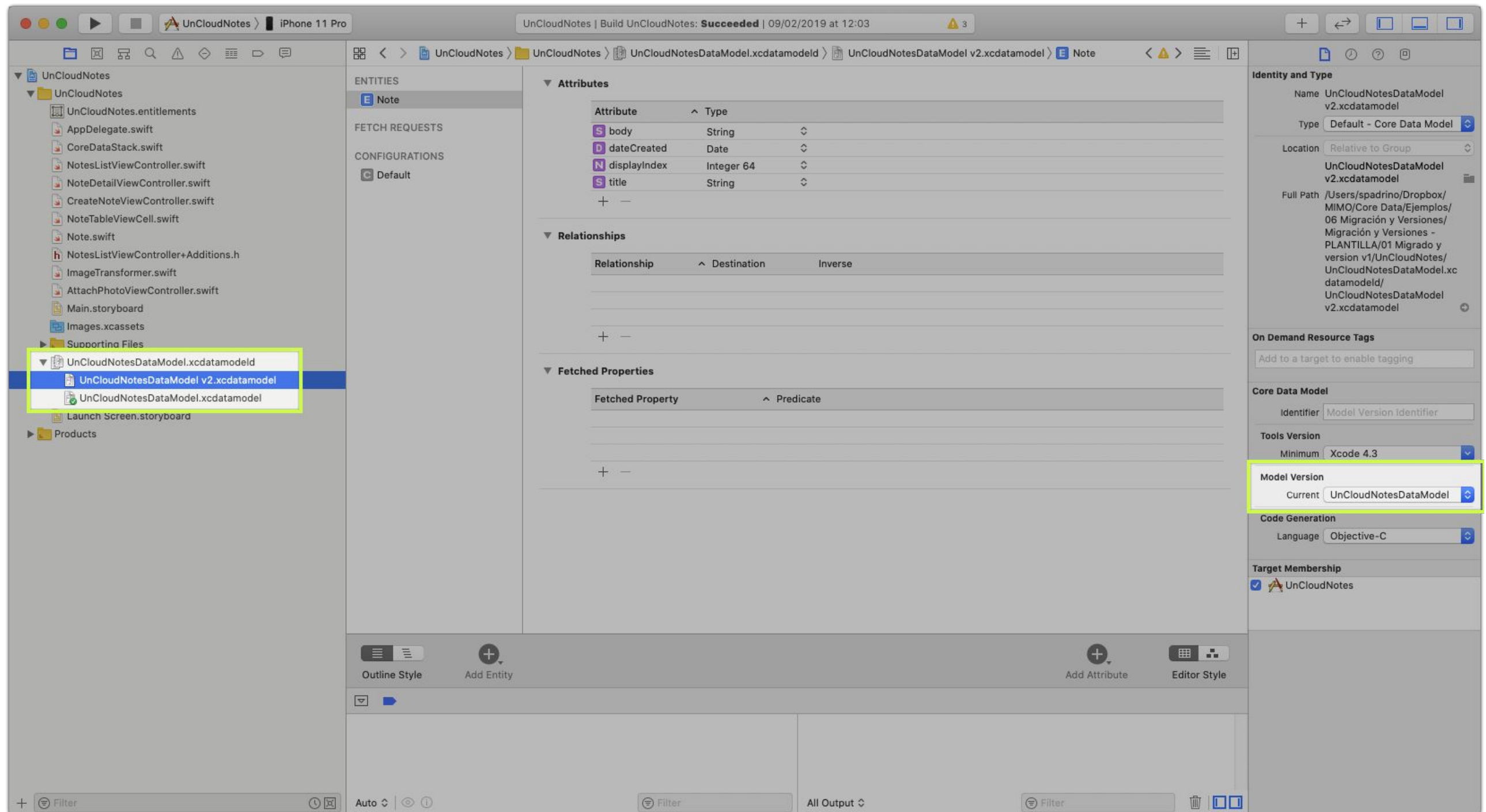
Based on model



# Versionado del modelo: pasos



# Versionado del modelo: pasos



# Migración de datos

- Si nuestro modelo cambia y tenemos datos guardados en nuestro Persistent Store, la aplicación crashearán.
- Core Data permite hacer migraciones automáticas entre diferentes versiones de nuestro modelo.
- Para aquellos cambios donde la complejidad sea muy elevada, serán necesarios procesos de migración manual.

# Migración Automática

- Conocida como Lightweight migrations (migración ligera)
- Requiere que el modelo modificado tan solo haya sufrido sencillos cambios en su estructura:
  1. Añadir o eliminar un atributo
  2. Cambiar la propiedad optional de los atributos.
  3. Asignar un valor por defecto a un atributo.
  4. Renombrar entidades o atributos usando el campo Renaming ID.

# Migración Manual

- Proceso más costoso, ya que requiere trabajo por parte del desarrollador.
- Core Data no puede entender por sí solo la transformación de un modelo a otro.
- Es necesario definir un modelo de transformación (Mapping Model).
- En este tipo de migración se utiliza una instancia de `NSMigrationManager`.