

#WWDC18

# Core Data Best Practices

Session 224

Scott Perry, Engineer  
Nick Gillett, Engineer

Modernizing Core Data

Evolving containers

Matching models with views

Managing growth

Potpourri

# Modernizing Core Data

Evolving containers

Matching models with views

Managing growth

Potpourri

Modernizing Core Data

**Evolving containers**

Matching models with views

Managing growth

Potpourri

Modernizing Core Data

Evolving containers

Matching models with views

Managing growth

Potpourri

Modernizing Core Data

Evolving containers

Matching models with views

**Managing growth**

Potpourri

Modernizing Core Data

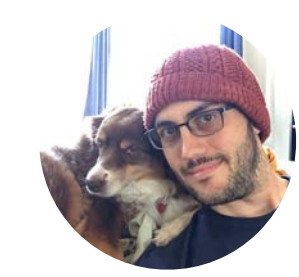
Evolving containers

Matching models with views

Managing growth

Potpourri

# Let's Build an App



Scott Perry posted "*Misty Morning*"



No such thing as a bad day in the  
Santa Cruz mountains!

1 Comment

6:34 AM on Friday, June 1, 2018



# Let's Manage Some Data

# Let's Manage Some Data

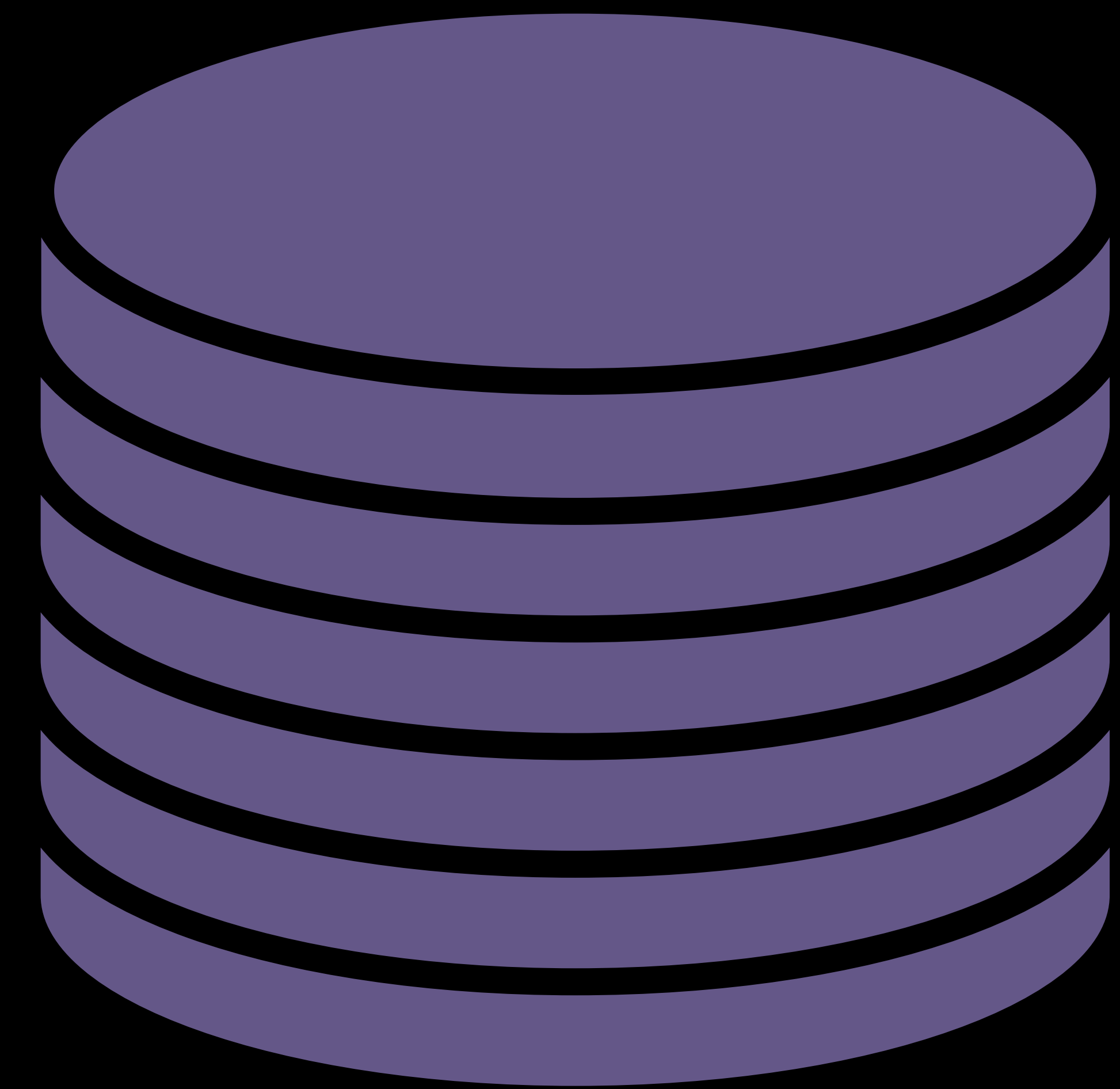
Where does the data live?

- Online

# Let's Manage Some Data

Where does the data live?

- Online
- On device



# Object Graph

# Object Graph Persistence

Core Data

# Core Data Manages Persistence



Scott Perry posted "*Misty Morning*"



No such thing as a bad day in the  
Santa Cruz mountains!

1 Comment

6:34 AM on Friday, June 1, 2018

# Core Data Manages Persistence



Scott Perry posted "*Misty Morning*"



No such thing as a bad day in the Santa Cruz mountains!

1 Comment

6:34 AM on Friday, June 1, 2018

NSManagedObjectContext

Entity: Post

Attribute: image (Binary Data)

Attribute: timestamp (Date)

Relationship: comments (→ Comment)

...

Entity: Comment

Attribute: author (String)

Relationship: post (→ Post)

...



# Core Data Manages Persistence

NSManagedObjectContext

Entity: Post  
Attribute: image (Binary Data)  
Attribute: timestamp (Date)  
Relationship: comments (→ Comment)  
...

Entity: Comment  
Attribute: author (String)  
Relationship: post (→ Post)  
...

# Core Data Manages Persistence



NSManagedObjectModel

Entity: Post  
Attribute: image (Binary Data)  
Attribute: timestamp (Date)  
Relationship: comments (→ Comment)  
...

Entity: Comment  
Attribute: author (String)  
Relationship: post (→ Post)  
...

# Core Data Manages Persistence

NSPersistentStoreCoordinator

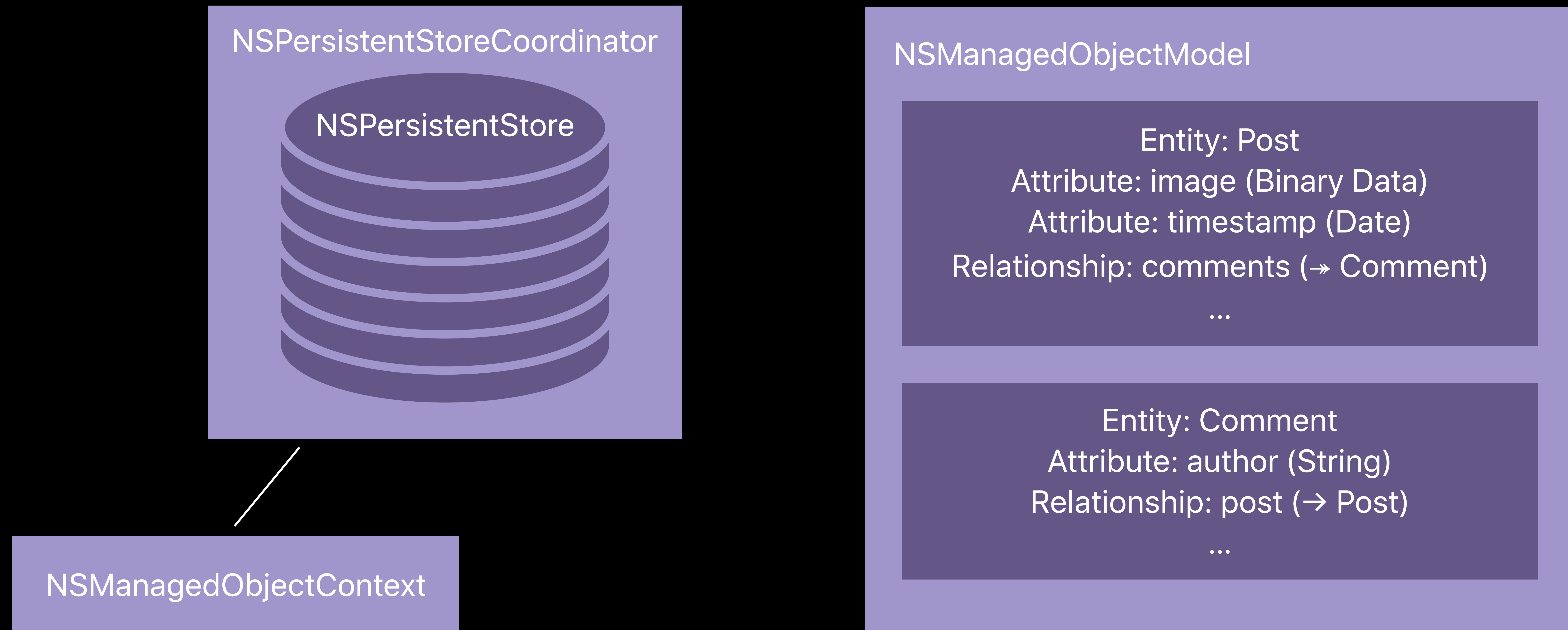


NSManagedObjectModel

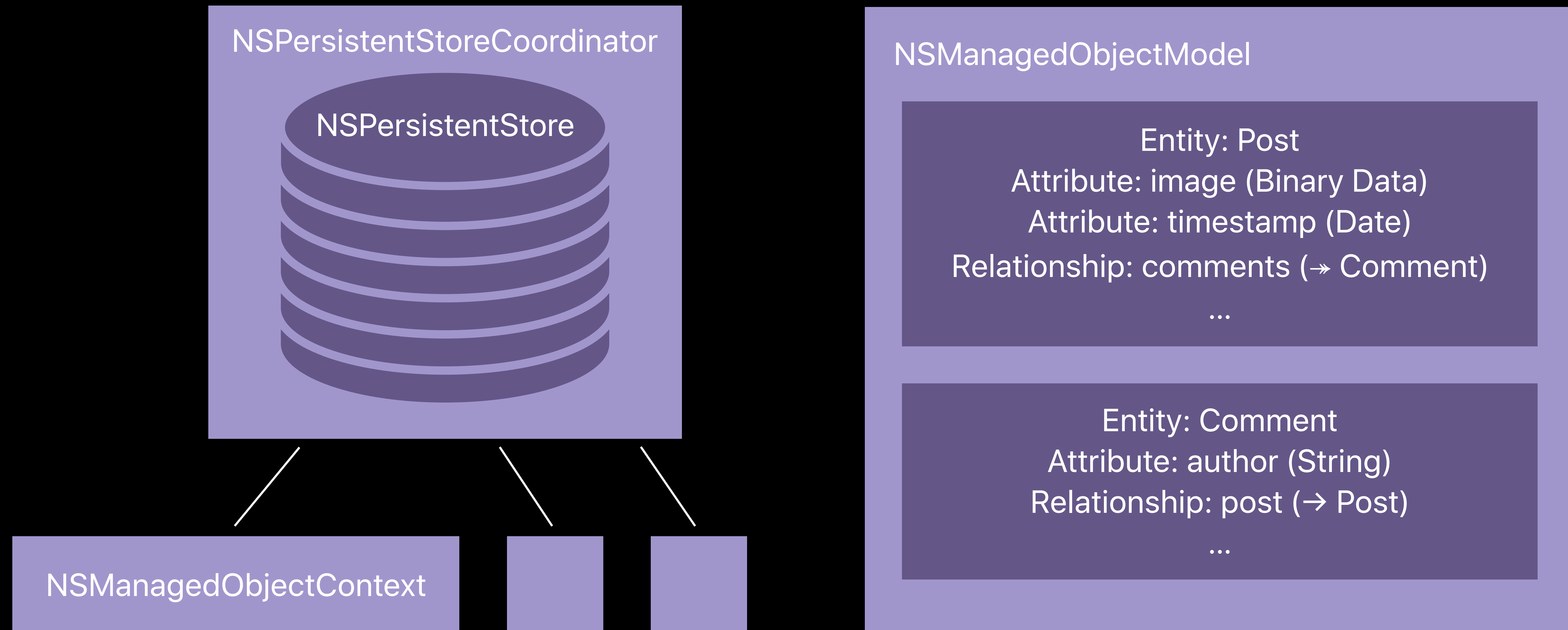
Entity: Post  
Attribute: image (Binary Data)  
Attribute: timestamp (Date)  
Relationship: comments (→ Comment)  
...

Entity: Comment  
Attribute: author (String)  
Relationship: post (→ Post)  
...

# Core Data Manages Persistence



# Core Data Manages Persistence



```
let storeURL = FileManager.default
    .urls(for: .documentDirectory, in: .userDomainMask)[0]
    .appendingPathComponent("Postings")

guard let modelURL = Bundle.main.url(forResource: "Postings", withExtension: "momd"),
    let model = NSManagedObjectModel(contentsOf: modelURL) else {
    print("error")
    return
}

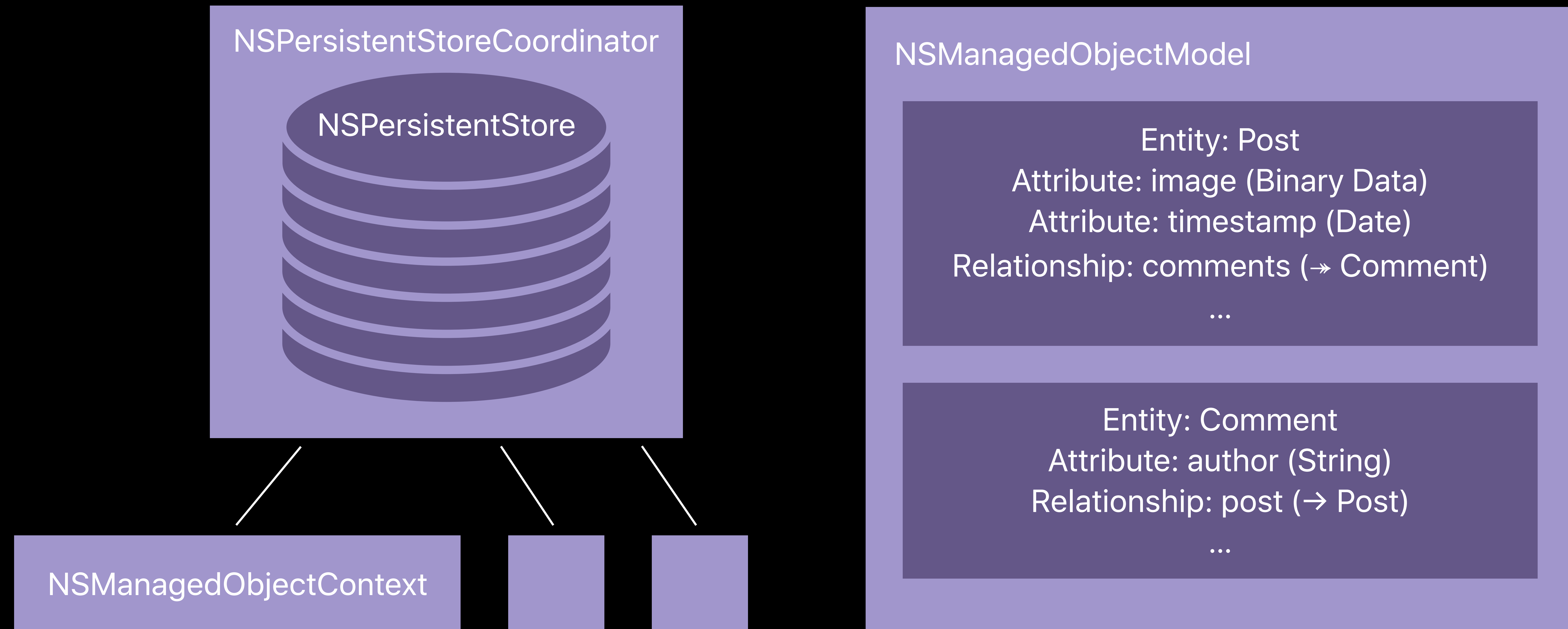
let psc = NSPersistentStoreCoordinator(managedObjectModel: model)
var success = false
do {
    try psc.addPersistentStore(ofType: NSSQLiteStoreType, configurationName: nil,
        at: storeURL,
        options: [NSInferMappingModelAutomaticallyOption: true,
            NSMigratePersistentStoresAutomaticallyOption: true])

    success = true
} catch {
    print("error loading store: \(error)")
    return
}

if success {
    print("ready!")
}
```

```
let container = NSPersistentContainer(name: "Postings")
container.loadPersistentStores { (desc, err) in
    if let err = err {
        print("error loading store \(desc): \(err)")
        return
    }
    print("ready!")
}
```

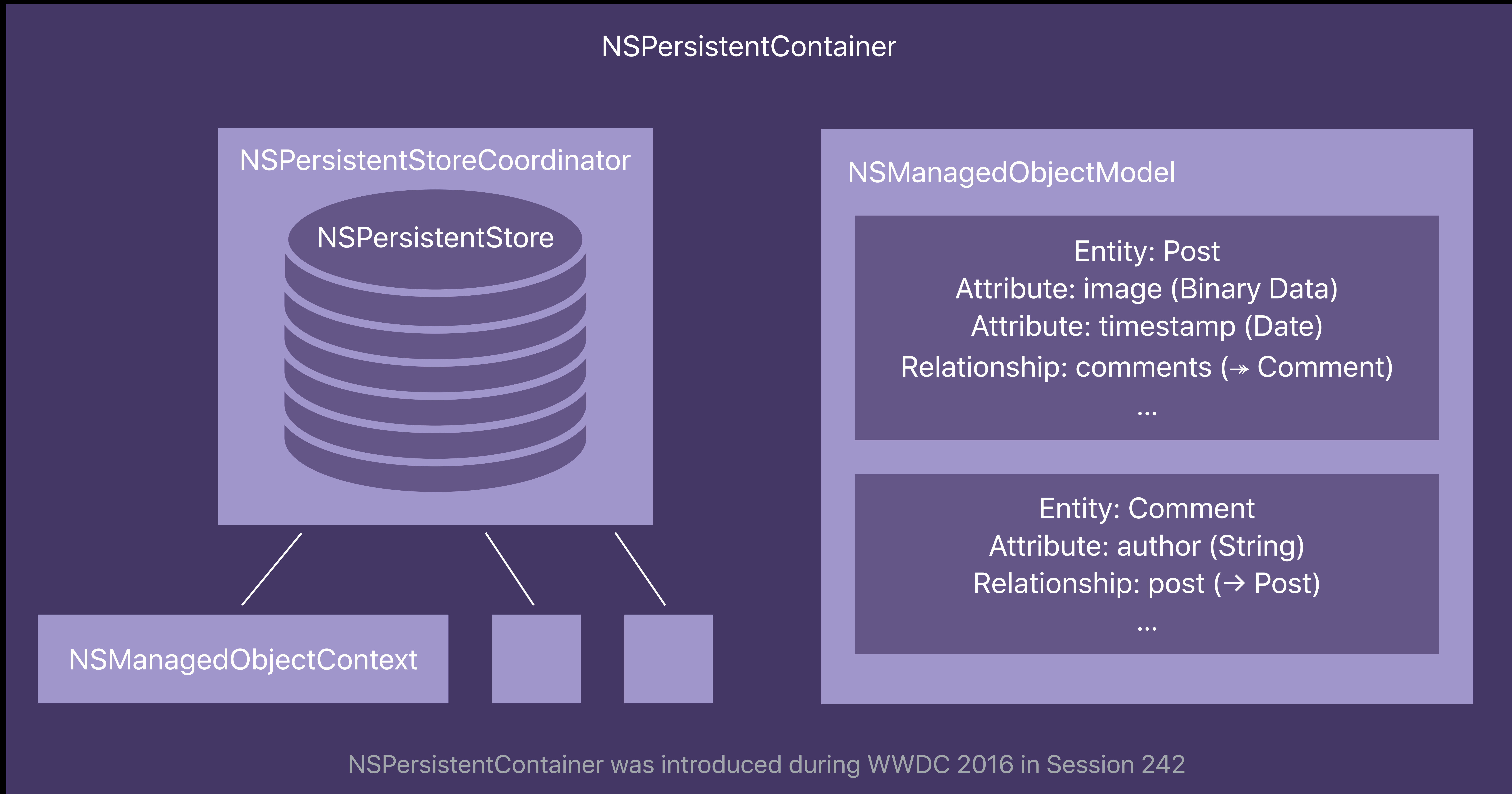
# Core Data Manages Persistence

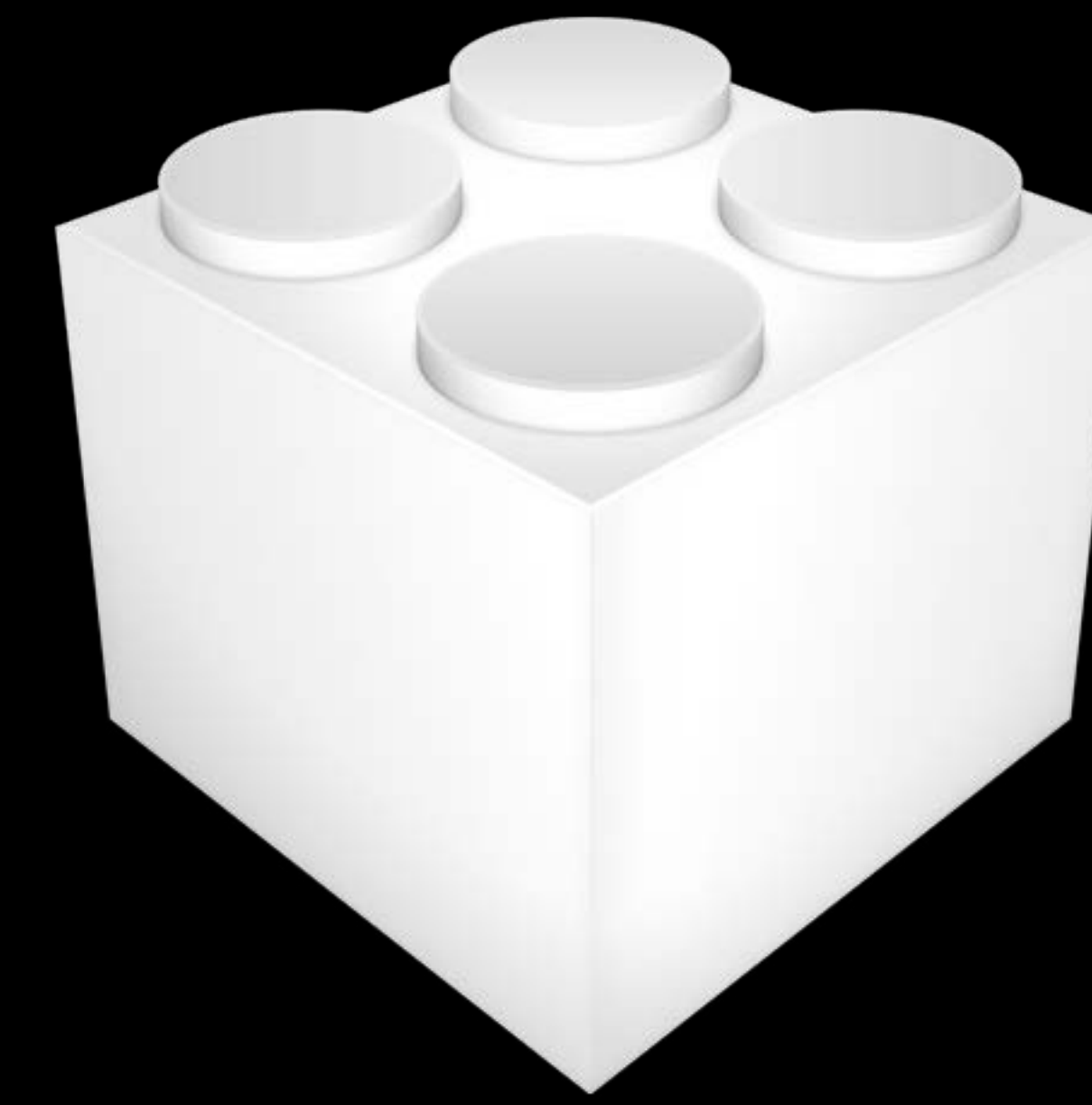






NSPersistentContainer was introduced during WWDC 2016 in Session 242



# Core Data Manages Persistence





 Postings.app  
  Contents  
  Resources  
  Postings.xcdatamodeld

- 📄 Postings.app
  - 📁 Contents
    - 📁 Frameworks
      - 📦 PostingsModel.framework
        - 📁 Contents
          - 📁 Resources
            - 📄 Postings.xcdatamodeld

# How the Container Finds Models

```
guard let modelURL = Bundle.main.url(forResource: "Postings", withExtension: "momd"),
    let model = NSManagedObjectModel(contentsOf: modelURL) else {
    print("error")
    return
}

let container = NSPersistentContainer(name: "Postings", managedObjectModel: model)
container.loadPersistentStores { (desc, err) in
    if let err = err {
        print("error loading store \(desc): \(err)")
        return
    }
    print("ready!")
}
```

# How the Container Finds Models

# How the Container Finds Models

```
class PMPersistentContainer: NSPersistentContainer {}
```

# How the Container Finds Models

```
class PMPersistentContainer: NSPersistentContainer {}
```

```
let container = NSPersistentContainer(name: "Postings")
```

```
container.loadPersistentStores { (desc, err) in
    if let err = err {
        print("error loading store \(desc): \(err)")
        return
    }
    print("ready!")
}
```



# How the Container Finds Models

```
class PMPersistentContainer: NSPersistentContainer {}
```

```
let container = PMPersistentContainer(name: "Postings")

container.loadPersistentStores { (desc, err) in
    if let err = err {
        print("error loading store \(desc): \(err)")
        return
    }
    print("ready!")
}
```

 Postings.sqlite

 Documents

 Postings.sqlite

- 📁 Documents
  - 📁 PostingsModel
    - 📄 Postings.sqlite

# More Container Tricks

Customizing where stores are—stored

```
let storeName = container.persistentStoreDescriptions[0].url?.lastPathComponent
    ?? container.name

let container = PMPersistentContainer(name: "Postings")

container.persistentStoreDescriptions[0].url = storeStorage.appendingPathComponent(storeName)

container.loadPersistentStores { (desc, err) in
    if let err = err {
        print("error loading store \(desc): \(err)")
        return
    }
    print("ready!")
}
```

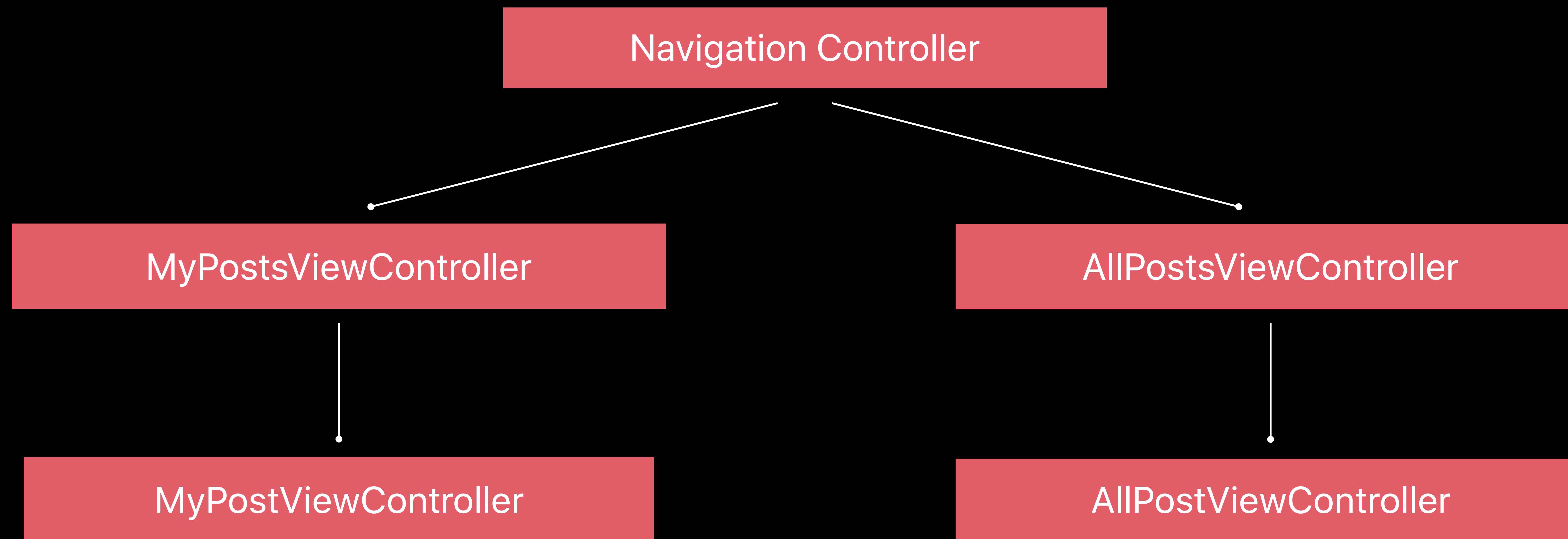
# More Container Tricks

Customizing where stores are—stored

```
class PMPersistentContainer: NSPersistentContainer {
    override class func defaultDirectoryURL() -> URL {
        return super.defaultDirectoryURL().appendingPathComponent("PostingsModel")
    }
}
```

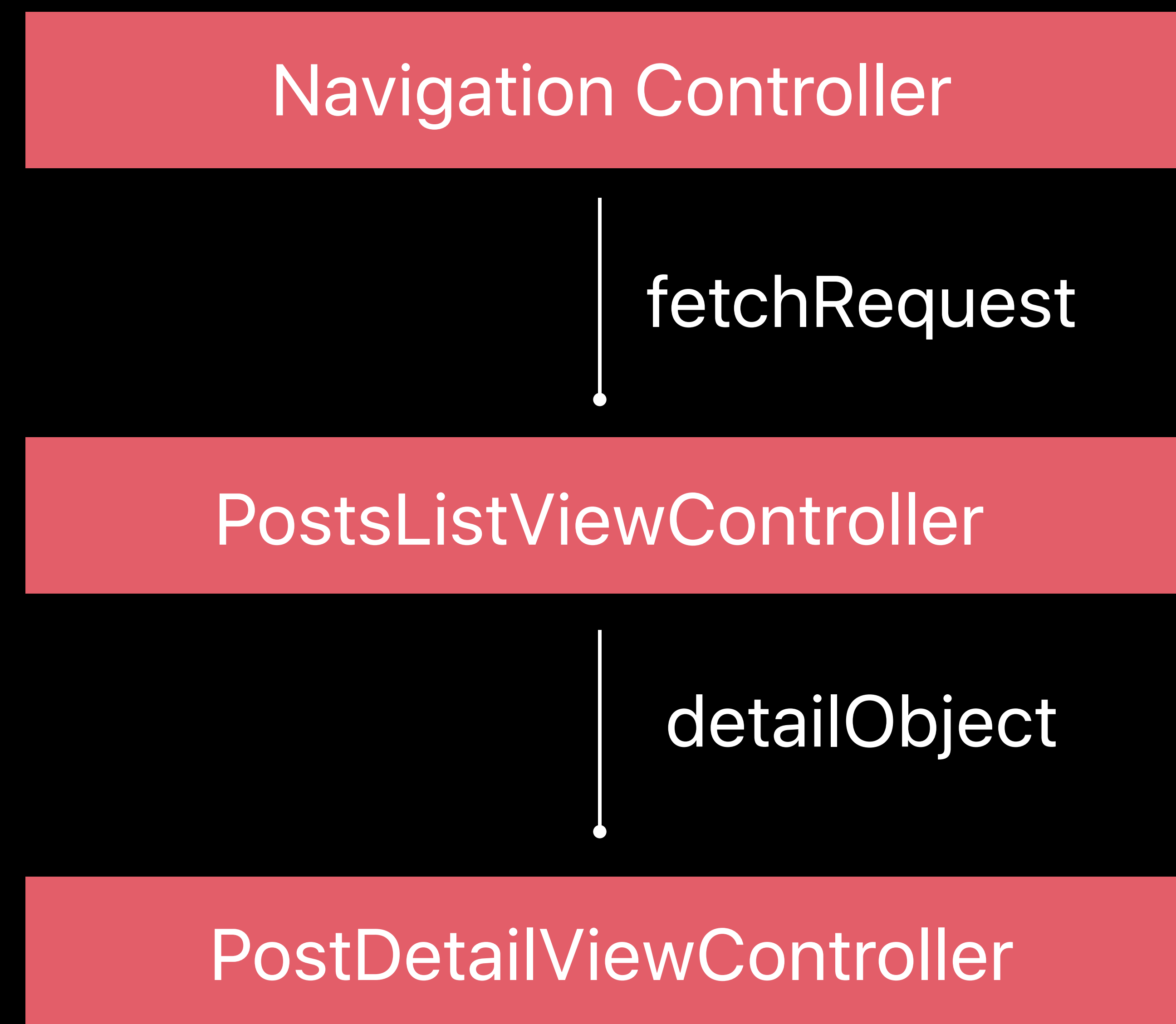
# Generalizing Controllers

# Generalizing Controllers





# Generalizing Controllers



# Generalizing Controllers

Controllers using Core Data should have

- Data
  - Managed object
  - Fetch request

# Generalizing Controllers

Controllers using Core Data should have

- Data
  - Managed object
  - Fetch request
- Managed object context

# Generalizing Workers

Workers using Core Data should have

- Data
  - URL
  - Property list
- Managed object context

# Generalizing Controllers

Getting view controllers what they need when using

# Generalizing Controllers

Getting view controllers what they need when using

- Segues

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    let fetchRequest = NSFetchRequest<Post>(entityName: "Post")
    // ...
    if let destinationViewController = segue.destination as? PostsListViewController {
        destinationViewController.context = self.context
        destinationViewController.fetchRequest = fetchRequest
    }
}
```

# Generalizing Controllers

Getting view controllers what they need when using

- Segues
- Storyboards

```
let fetchRequest = NSFetchRequest<Post>(entityName: "Post")
// ...
let destinationViewController = PostsListViewController(nibName: "...", bundle: nil)
destinationViewController.context = self.context
destinationViewController.fetchRequest = fetchRequest
present(destinationViewController, animated: true, completion: nil)
```

# Generalizing Controllers

Getting view controllers what they need when using

- Segues
- Storyboards
- Code

```
let fetchRequest = NSFetchRequest<Post>(entityName: "Post")
// ...
let destinationViewController = PostsListViewController(context: self.context,
                                                       fetchRequest: fetchRequest)
present(destinationViewController, animated: true, completion: nil)
```



# Turning Fetch Requests Into List Views

Configure the results' behaviour

# Turning Fetch Requests Into List Views

Configure the results' behaviour

- Fetch limit
- Batch size

# Turning Fetch Requests Into List Views

Configure the results' behaviour

- Fetch limit
- Batch size

Use the fetched results controller!

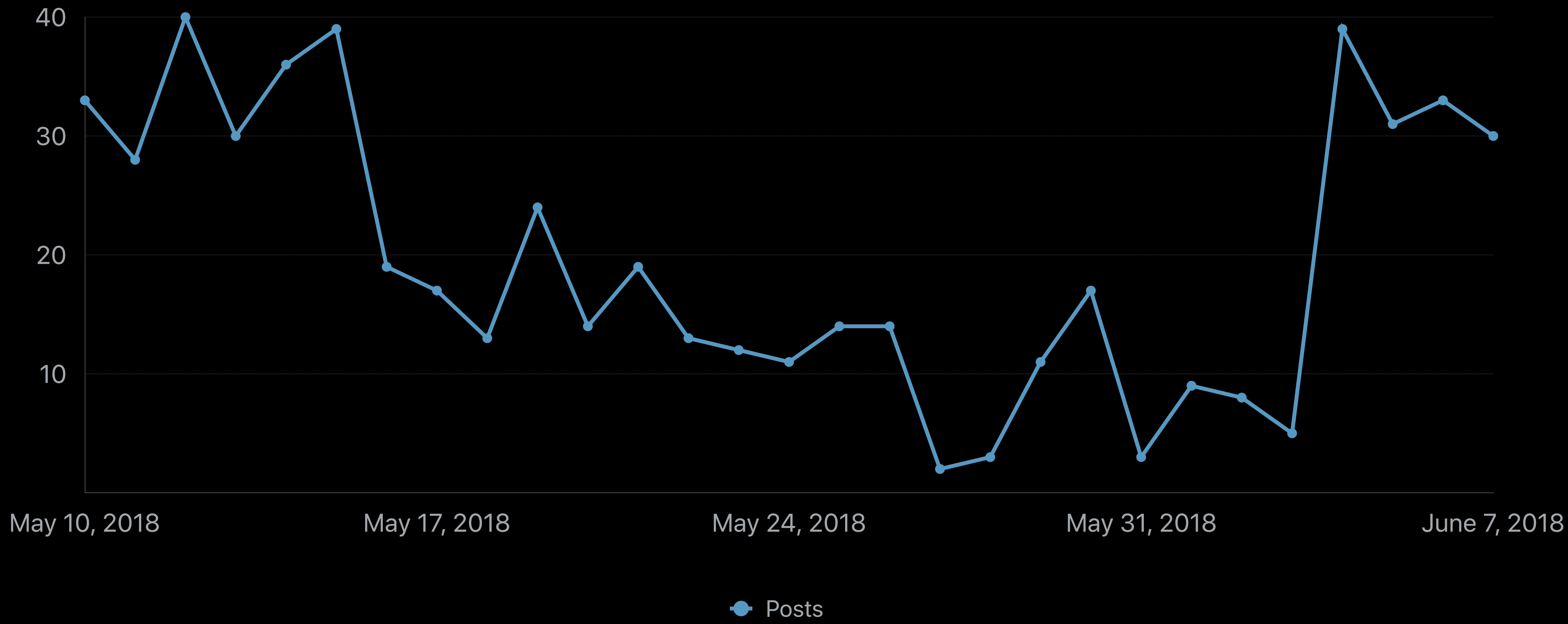
# Turning Fetch Requests Into List Views

Use the fetched results controller!

```
extension Post {  
    @objc var day: String {  
        let components = Calendar.current.dateComponents([Calendar.Component.year,  
                                                         Calendar.Component.month,  
                                                         Calendar.Component.day],  
                                                         from: self.timestamp!)  
        return "\(components.year!)-\(components.month!)-\(components.day!)"  
    }  
}
```

# Matching UIs to the Model

## ChartViewController



# Matching UIs to the Model

## Complex fetch requests

```
let fr = NSFetchRequest<NSDictionary>(entityName: "Post")

let end = Date()
let start = Calendar.current.date(byAdding: DateComponents(day: -30), to: end)!
fr.predicate = NSPredicate(format: "timestamp > %@ AND timestamp <= %@",
                             argumentArray: [start, end])

fr.propertiesToGroupBy = ["day"]
fr.resultType = .dictionaryResultType

let ced = NSEntityDescription()
ced.expression = NSEntityDescription.expression(forKeyPath: "count:",
                                                arguments: [NSEntityDescription.expression(forKeyPath: "day")])

ced.name = "count"
ced.expressionResultType = .integer64AttributeType
fr.propertiesToFetch = ["day", ced]
```

# Matching UIs to the Model

## Complex fetch requests

```
let fr = NSFetchRequest<NSDictionary>(entityName: "Post")

let end = Date()
let start = Calendar.current.date(byAdding: DateComponents(day: -30), to: end)!
fr.predicate = NSPredicate(format: "timestamp > %@ AND timestamp <= %@",
                             argumentArray: [start, end])

fr.propertiesToGroupBy = ["day"]
fr.resultType = .dictionaryResultType

let ced = NSEExpressionDescription()
ced.expression = NSEExpression(forFunction: "count:",
                               arguments: [NSEExpression(forKeyPath: "day")])

ced.name = "count"
ced.expressionResultType = .integer64AttributeType
fr.propertiesToFetch = ["day", ced]
```

# Matching UIs to the Model

## Complex fetch requests

```
let fr = NSFetchRequest<NSDictionary>(entityName: "Post")

let end = Date()
let start = Calendar.current.date(byAdding: DateComponents(day: -30), to: end)!
fr.predicate = NSPredicate(format: "timestamp > %@ AND timestamp <= %@",
                             argumentArray: [start, end])

fr.propertiesToGroupBy = ["day"]
fr.resultType = .dictionaryResultType

let ced = NSEExpressionDescription()
ced.expression = NSEExpression(forFunction: "count:",
                               arguments: [NSEExpression(forKeyPath: "day")])

ced.name = "count"
ced.expressionResultType = .integer64AttributeType
fr.propertiesToFetch = ["day", ced]
```



# Matching UIs to the Model

## Complex fetch requests

```
let fr = NSFetchRequest<NSDictionary>(entityName: "Post")

let end = Date()
let start = Calendar.current.date(byAdding: DateComponents(day: -30), to: end)!
fr.predicate = NSPredicate(format: "timestamp > %@ AND timestamp <= %@",
                             argumentArray: [start, end])
```

```
fr.propertiesToGroupBy = ["day"]
fr.resultType = .dictionaryResultType
```

```
let ced = NSEExpressionDescription()
ced.expression = NSEExpression(forFunction: "count:",
                               arguments: [NSEExpression(forKeyPath: "day")])

ced.name = "count"
ced.expressionResultType = .integer64AttributeType
fr.propertiesToFetch = ["day", ced]
```

# Matching UIs to the Model

## Complex fetch requests

```
let fr = NSFetchRequest<NSDictionary>(entityName: "Post")

let end = Date()
let start = Calendar.current.date(byAdding: DateComponents(day: -30), to: end)!
fr.predicate = NSPredicate(format: "timestamp > %@ AND timestamp <= %@",
                             argumentArray: [start, end])

fr.propertiesToGroupBy = ["day"]
fr.resultType = .dictionaryResultType

let ced = NSEExpressionDescription()
ced.expression = NSEExpression(forFunction: "count:",
                                arguments: [NSEExpression(forKeyPath: "day")])

ced.name = "count"
ced.expressionResultType = .integer64AttributeType
fr.propertiesToFetch = ["day", ced]
```

# Matching UIs to the Model

## Complex fetch requests

```
let fr = NSFetchRequest<NSDictionary>(entityName: "Post")

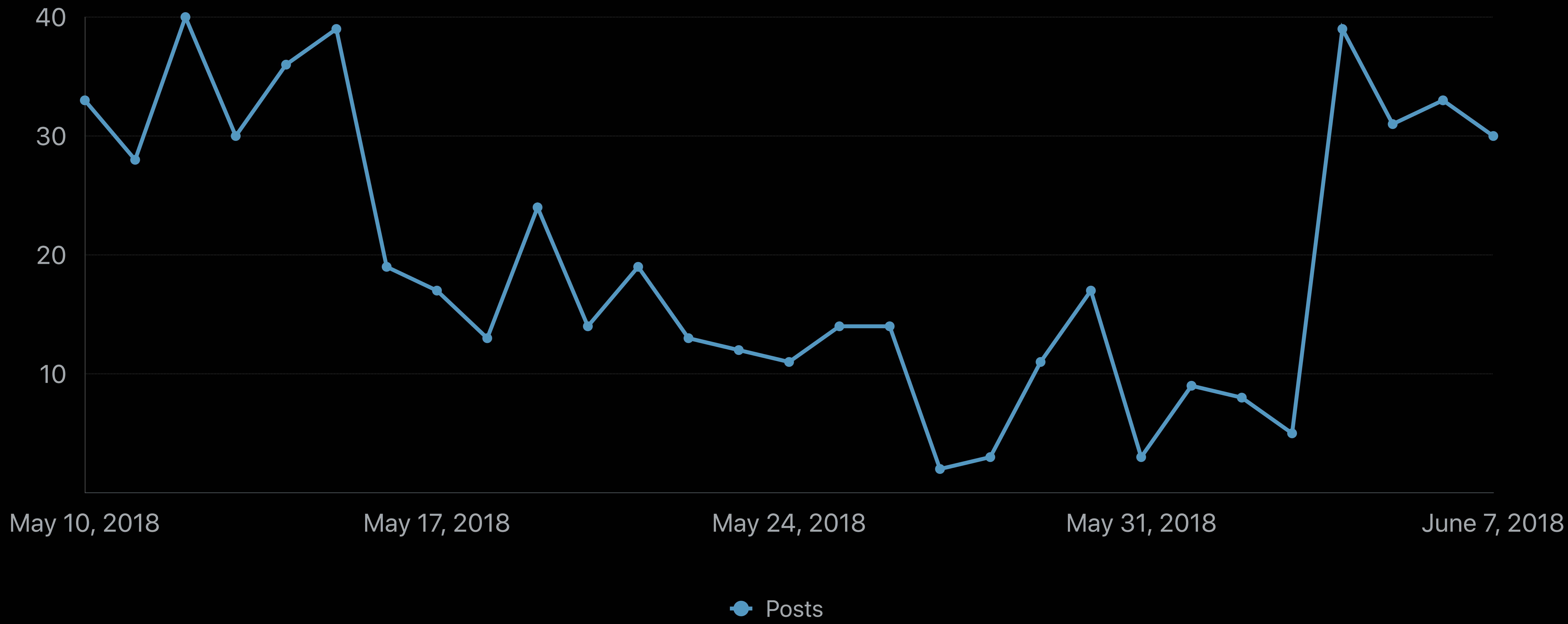
let end = Date()
let start = Calendar.current.date(byAdding: DateComponents(day: -30), to: end)!
fr.predicate = NSPredicate(format: "timestamp > %@ AND timestamp <= %@",
                             argumentArray: [start, end])

fr.propertiesToGroupBy = ["day"]
fr.resultType = .dictionaryResultType

let ced = NSEExpressionDescription()
ced.expression = NSEExpression(forFunction: "count:",
                               arguments: [NSEExpression(forKeyPath: "day")])
ced.name = "count"
ced.expressionResultType = .integer64AttributeType
fr.propertiesToFetch = ["day", ced]
```

# Matching UIs to the Model

## ChartViewController



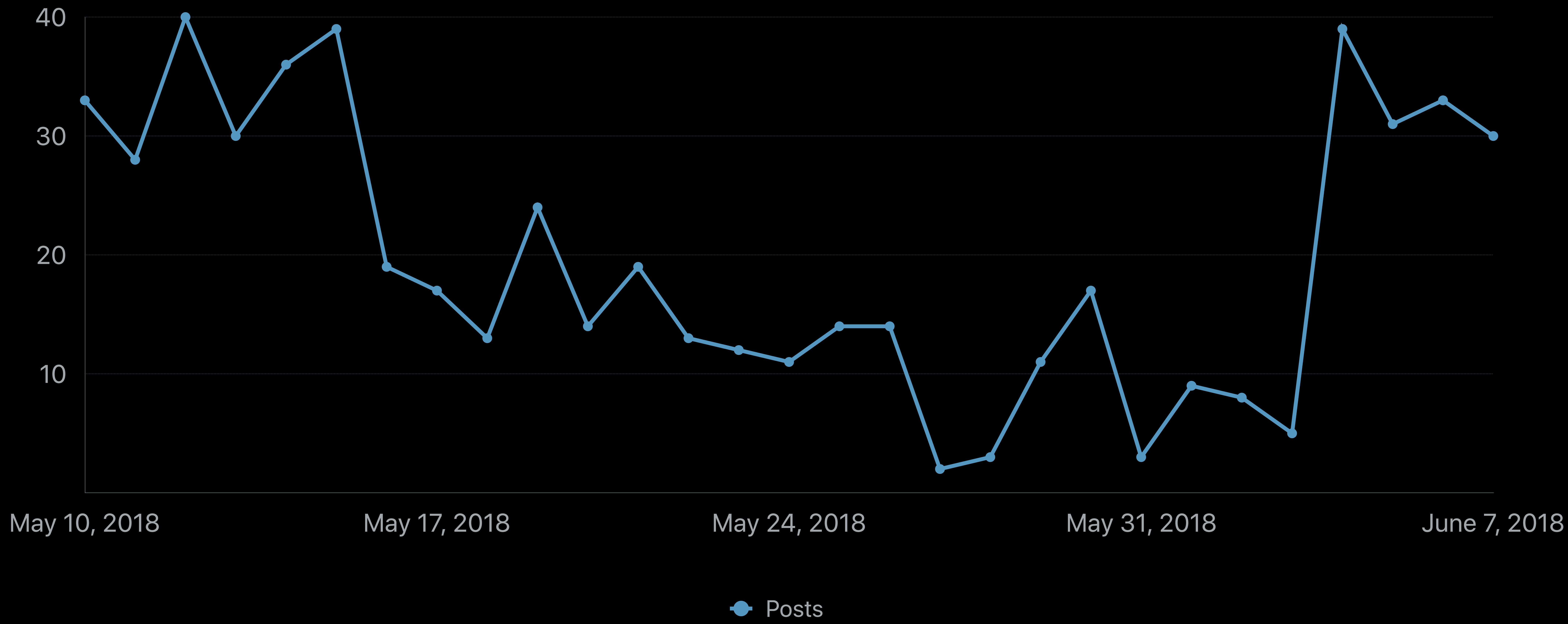
# Matching UIs to the Model

## Complex fetch requests

```
SELECT t0.ZDAY, COUNT( t0.ZDAY)
FROM ZPOST t0
WHERE ( t0.ZTIMESTAMP > ? AND t0.ZTIMESTAMP <= ?)
GROUP BY t0.ZDAY
```

# Matching UIs to the Model

## ChartViewController



# Matching UIs to the Model

ChartViewController



# Denormalization



# Matching UIs to the Model

## ChartViewController

### New entity

- PublishedPostCountPerDay
  - day (Date)
  - count (UInt)



# Matching UIs to the Model

## ChartViewController

New entity

Maintained as needed

- When publishing a post
- When deleting a post

```
@objc func contextWillSave(_ notification: Notification) {
    guard let context = notification.object as? NSManagedObjectContext else { return }

    context.performAndWait {
        for case let post as Post in context.insertedObjects {
            let countObj = PublishedPostCountPerDay(dayOf: post.timestamp, context: context)
            countObj.count += 1
        }
        for case let post as Post in context.deletedObjects {
            let countObj = PublishedPostCountPerDay(dayOf: post.timestamp, context: context)
            countObj.count -= 1
        }
    }
}
```

# Managing Growth

Nick Gillett

# Managing Growth

# Managing Growth

We want your application to grow

# Managing Growth

We want your application to grow

- Specific to your app



# Managing Growth

We want your application to grow

- Specific to your app
- Dependent on customer experience

# Managing Growth

We want your application to grow

- Specific to your app
- Dependent on customer experience
- Tends toward chaos

# Managing Growth

# Managing Growth

Give structure to the chaos

# Managing Growth

Give structure to the chaos

- Predictable behaviors

# Managing Growth

Give structure to the chaos

- Predictable behaviors
- Build tunable containers

# Managing Growth

Give structure to the chaos

- Predictable behaviors
- Build tunable containers
- Align performance with experience

# Managing Growth



# Managing Growth

Customer Aligned Metrics

# Managing Growth

## Customer Aligned Metrics

- Consistent UI

# Managing Growth

## Customer Aligned Metrics

- Consistent UI
- Responsive scrolling

# Managing Growth

## Customer Aligned Metrics

- Consistent UI
- Responsive scrolling
- Customer Delight

# Managing Growth

# Managing Growth

Engineering Metrics

# Managing Growth

## Engineering Metrics

- Peak memory consumption

# Managing Growth

## Engineering Metrics

- Peak memory consumption
- Battery drain



# Managing Growth

## Engineering Metrics

- Peak memory consumption
- Battery drain
- CPU time

# Managing Growth

## Engineering Metrics

- Peak memory consumption
- Battery drain
- CPU time
- I/O

9:41



Edit

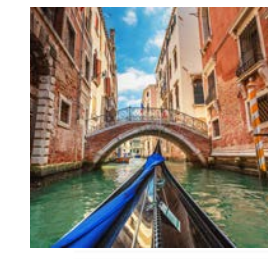
Library



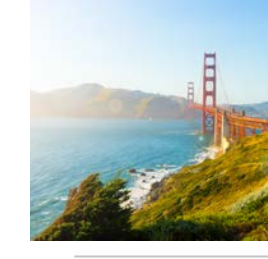
Clear

Download

Post All



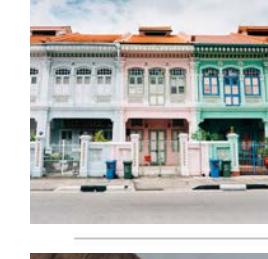
Love It!



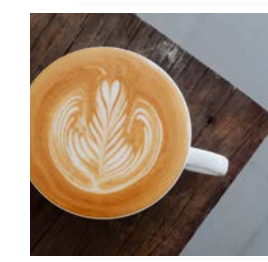
Can you believe it?



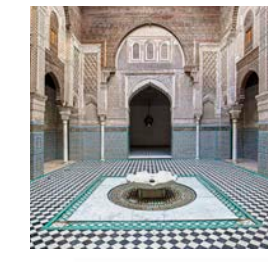
Woah!



Love It!



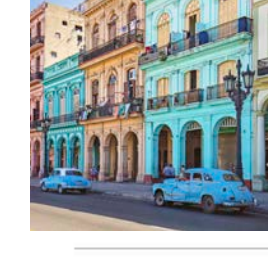
This demo is amazing!



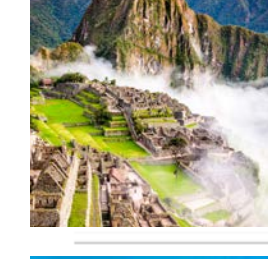
And voila it came across



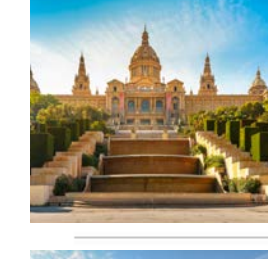
And voila it came across



I liked this over there



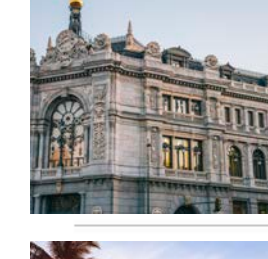
Woah!



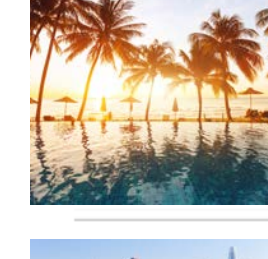
Can you believe it?



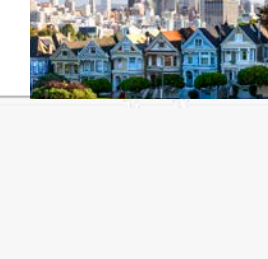
This demo is amazing!



Can you believe it?



Posting from my other device



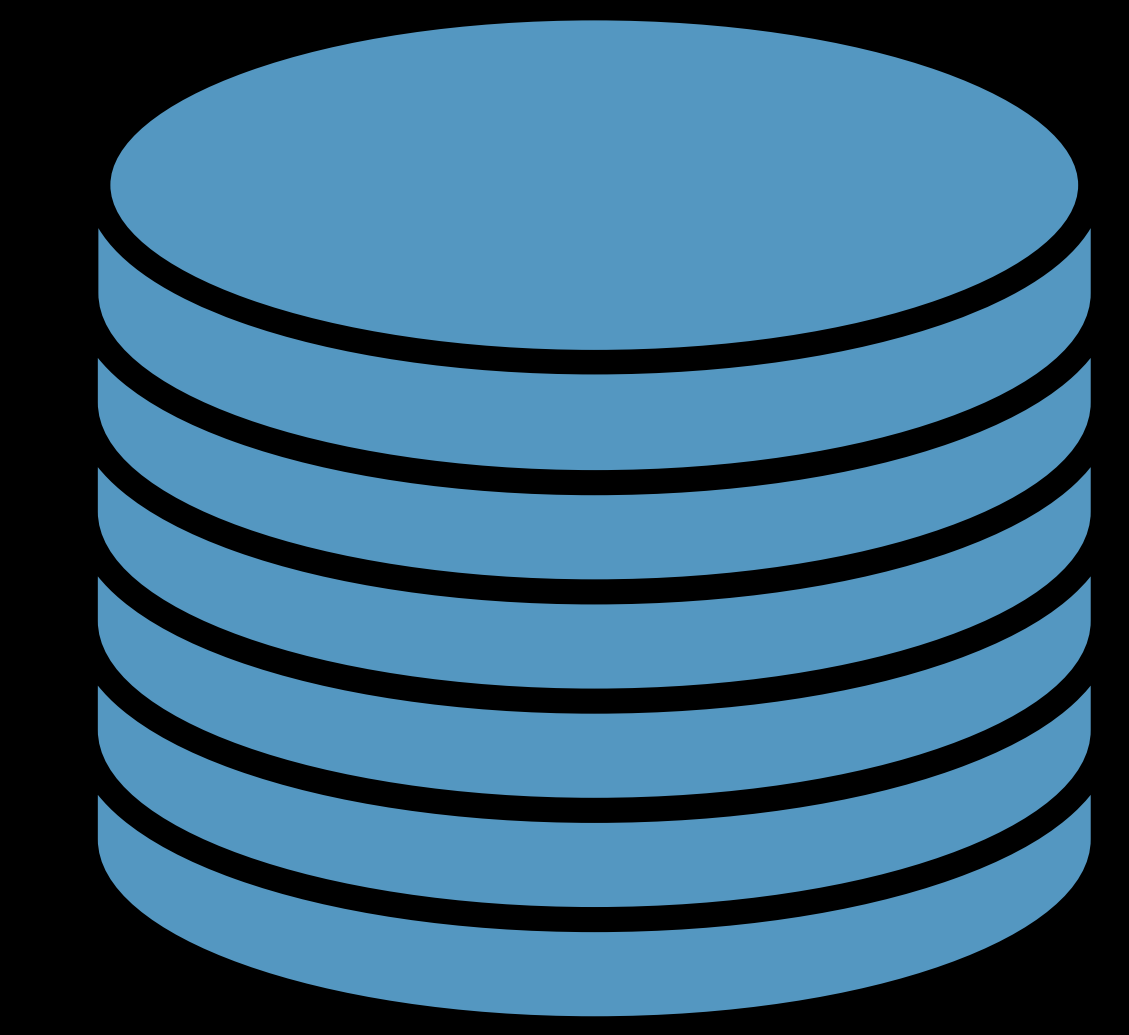
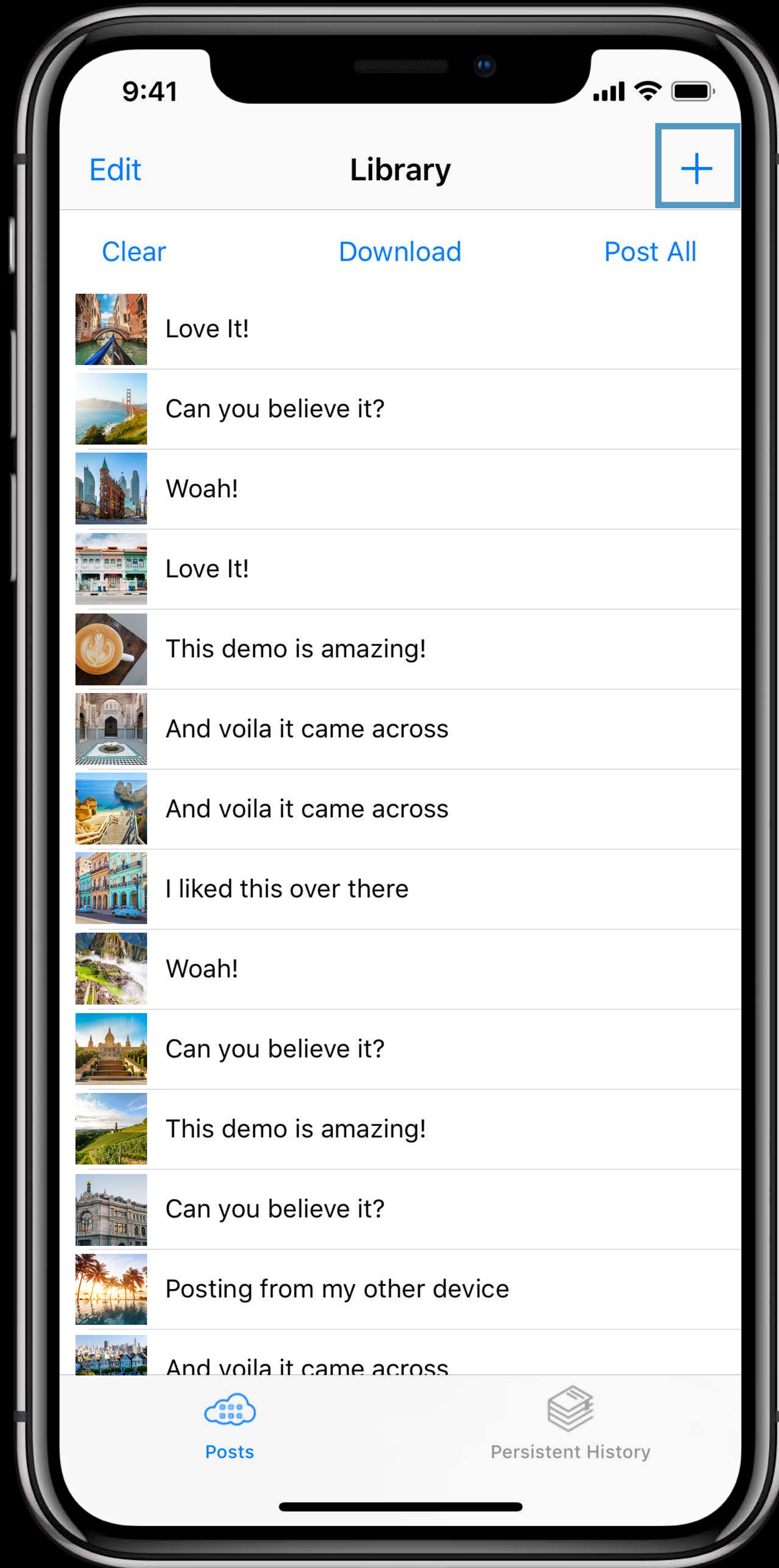
And voila it came across

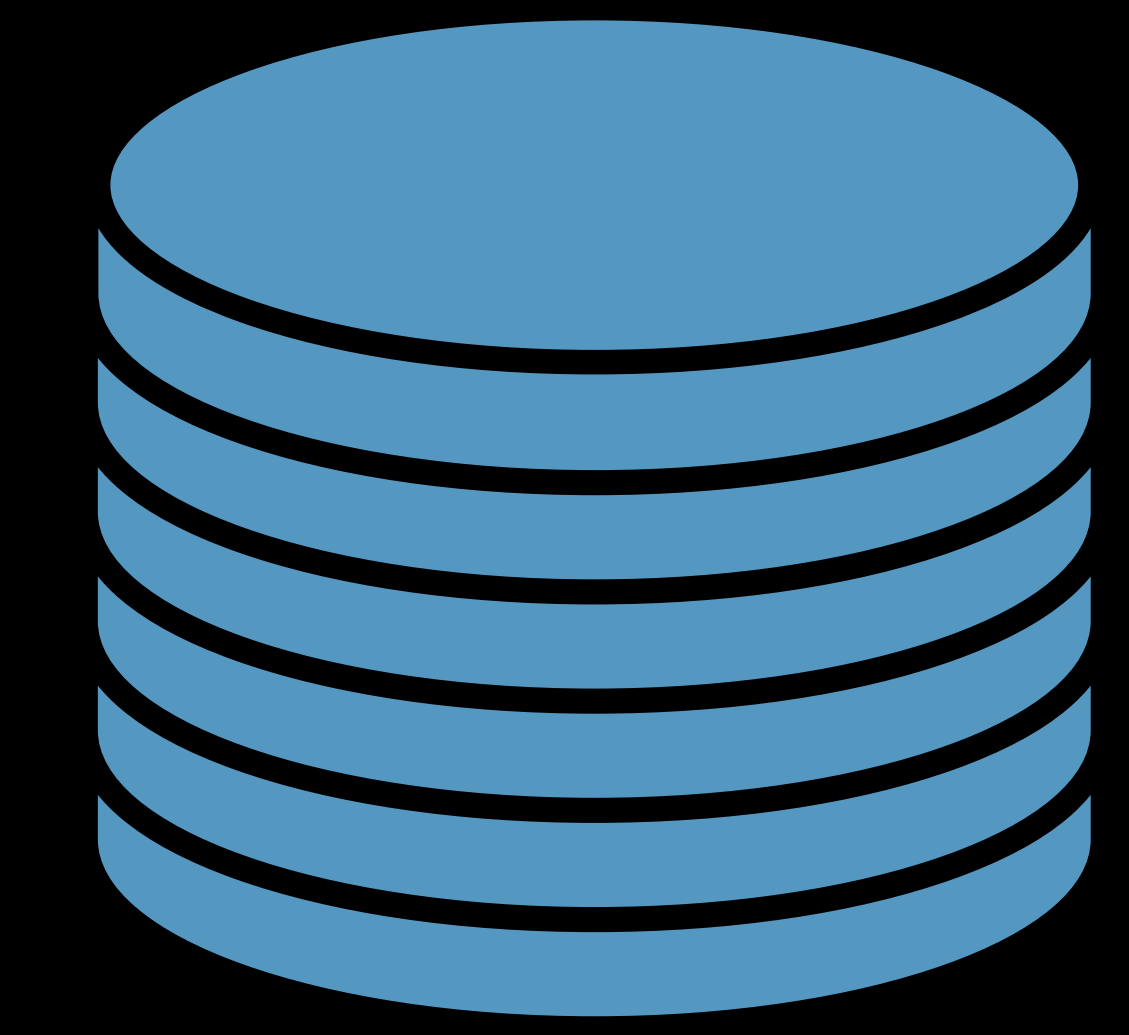
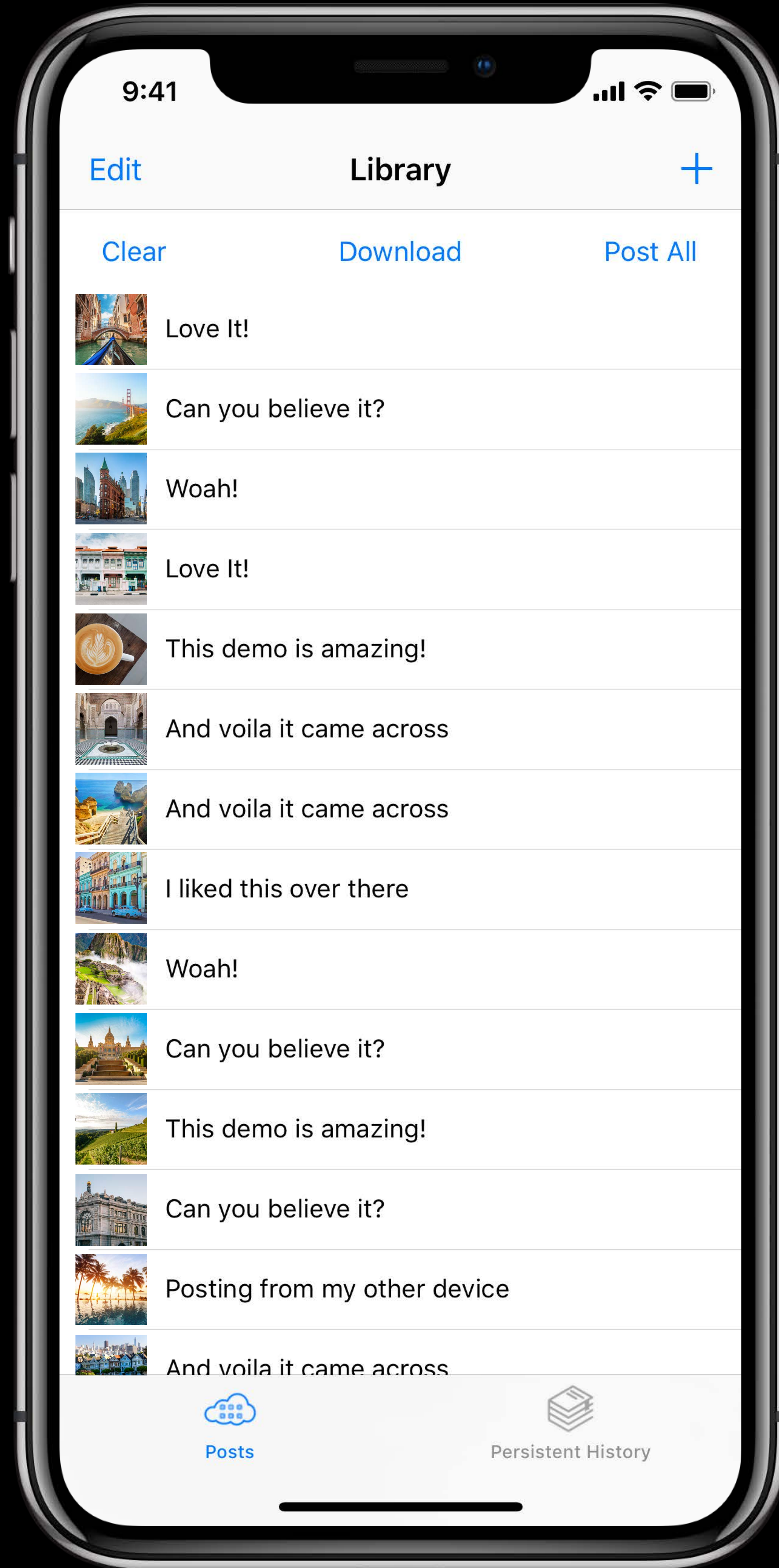


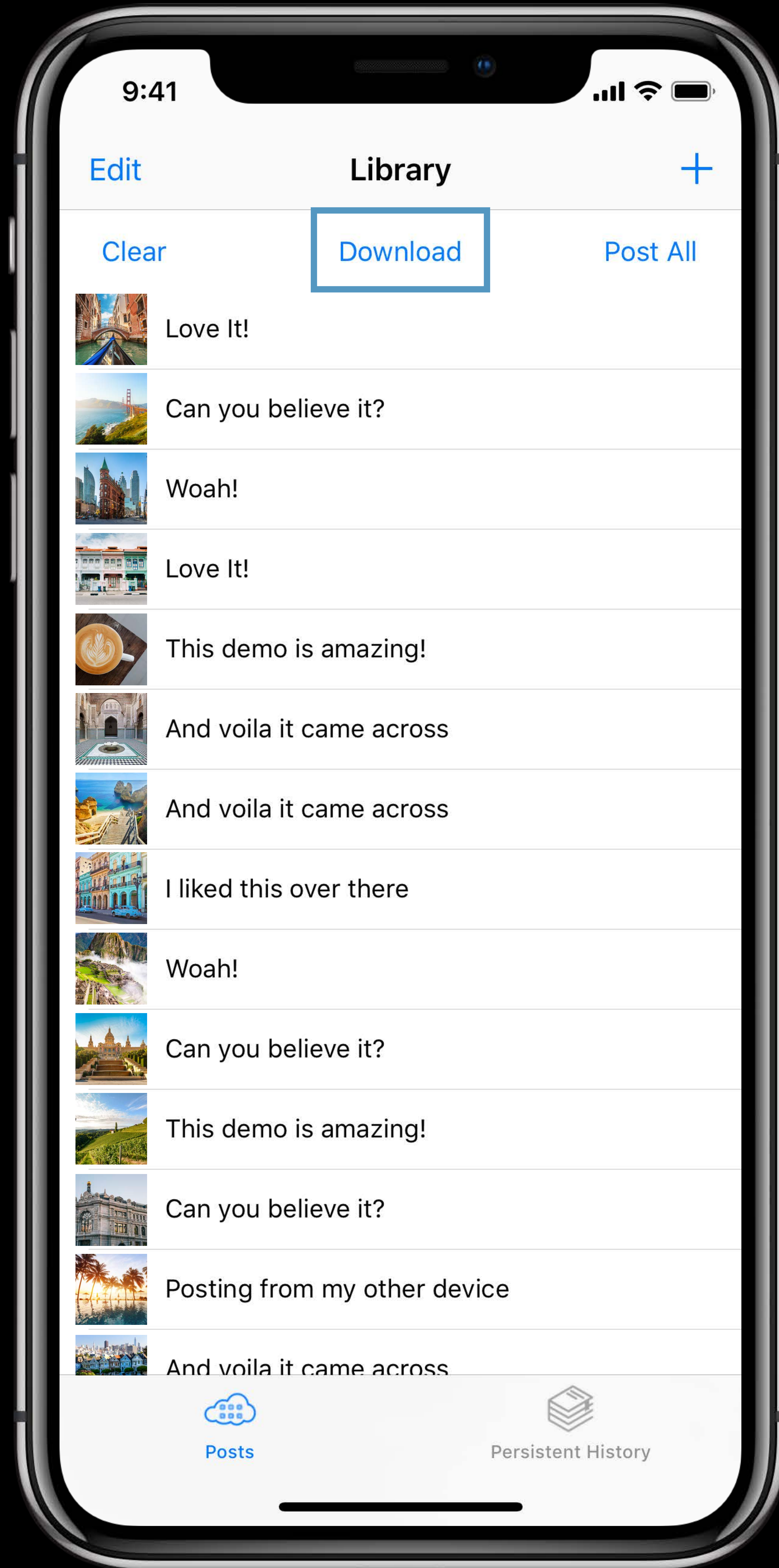
Posts

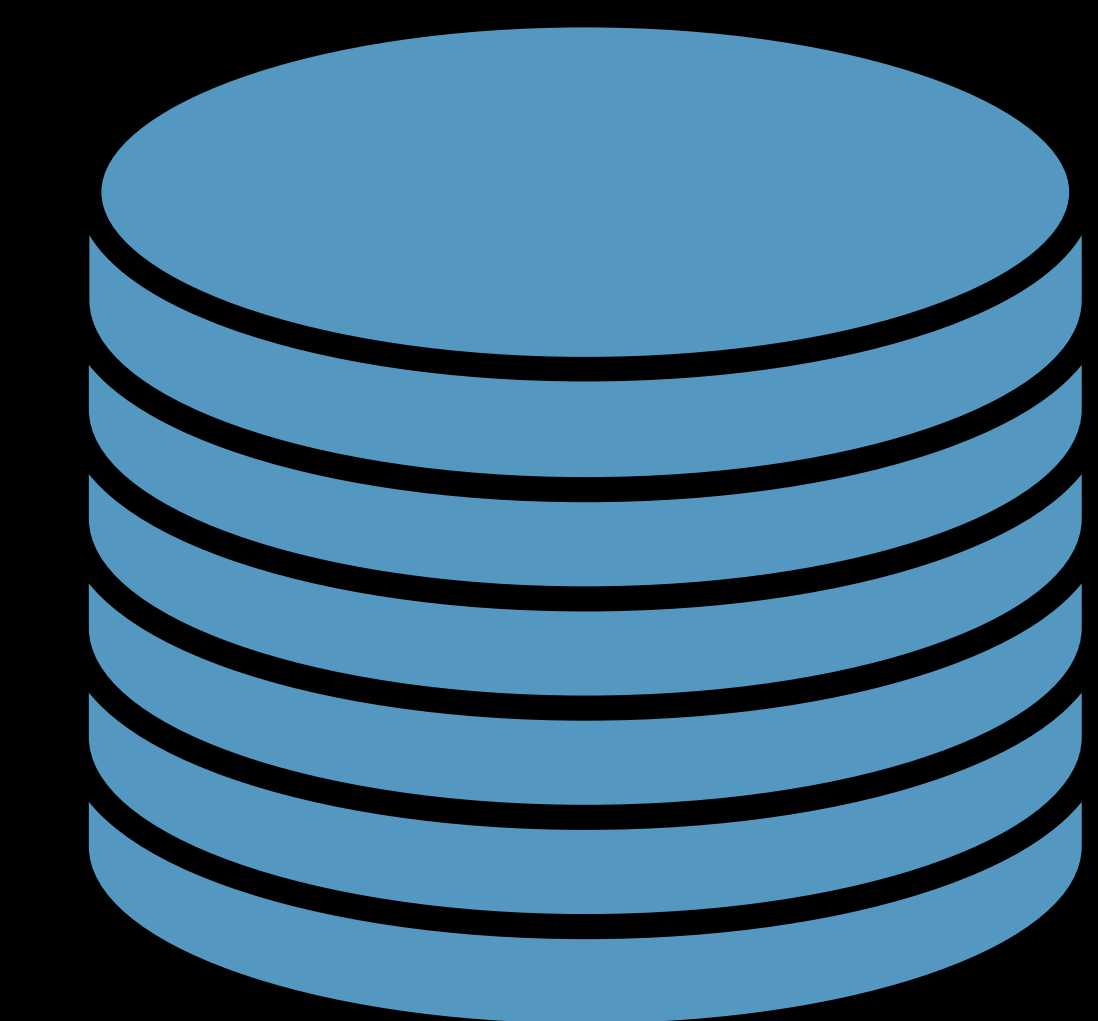
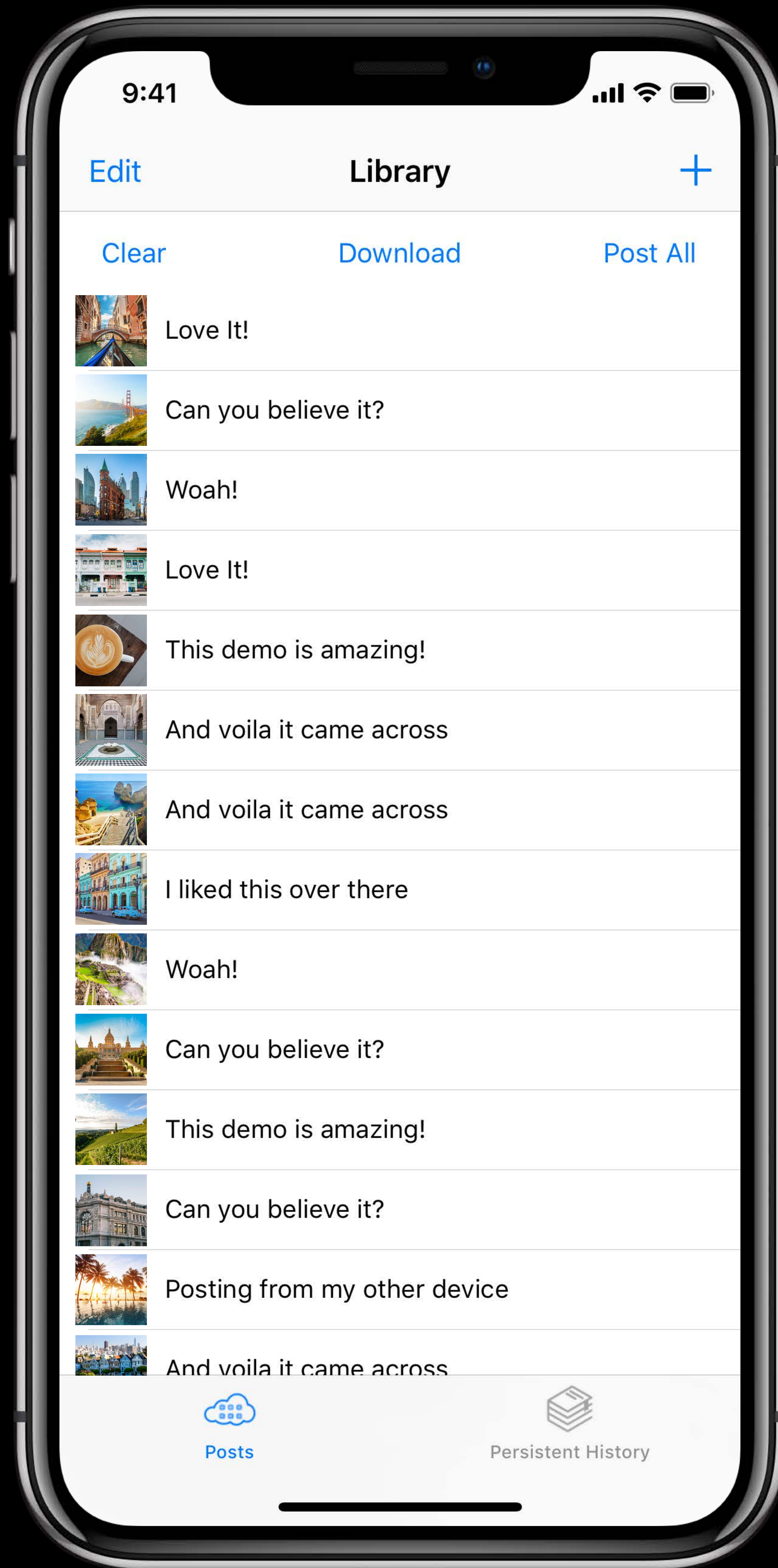


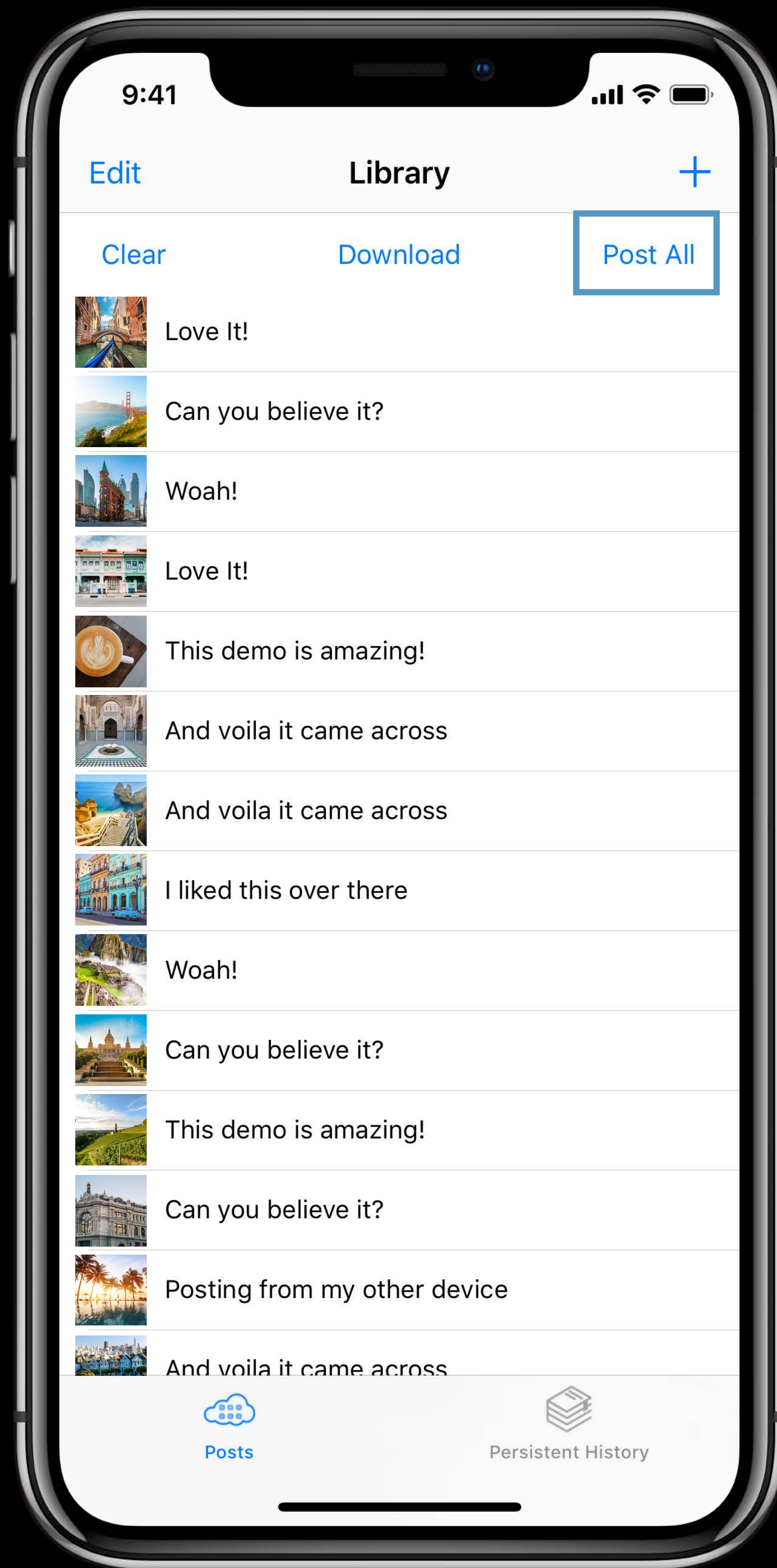
Persistent History



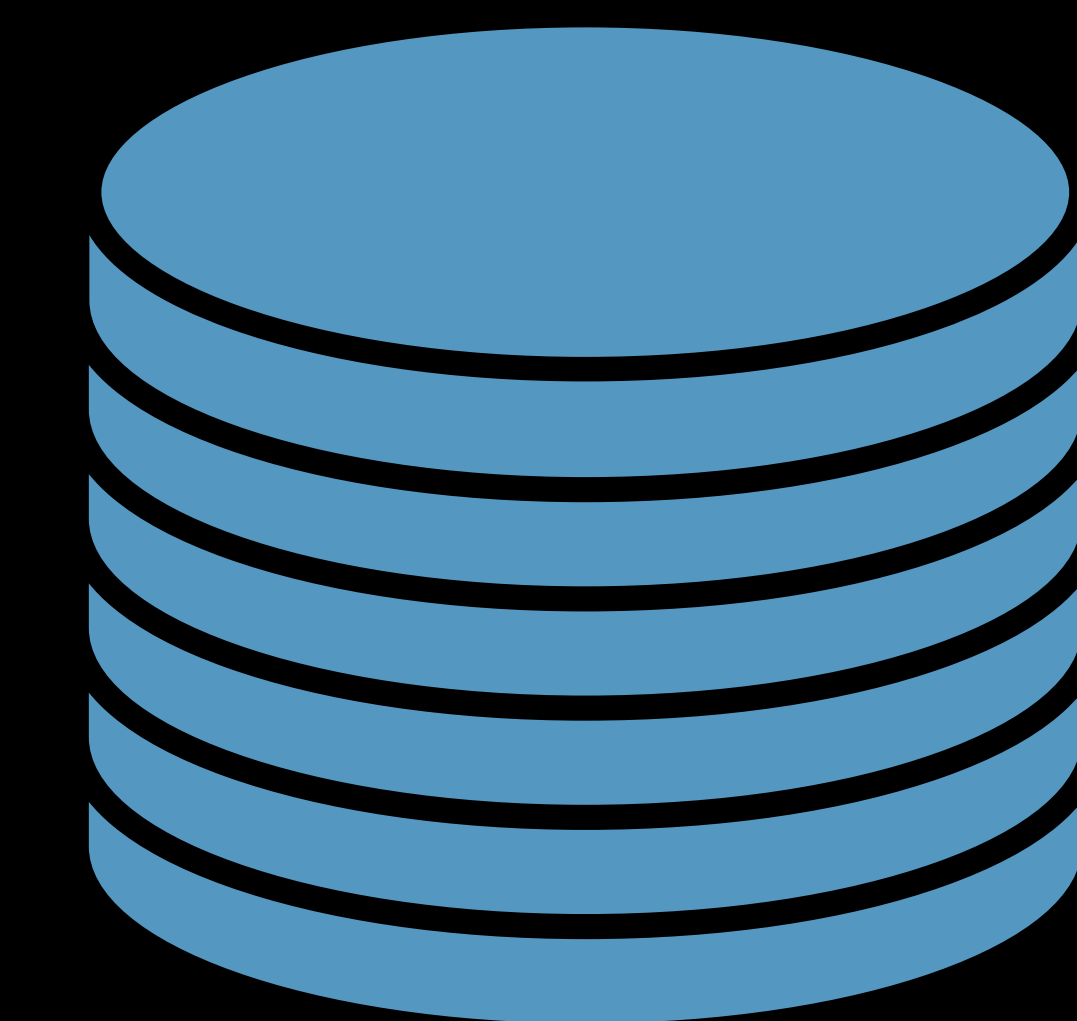
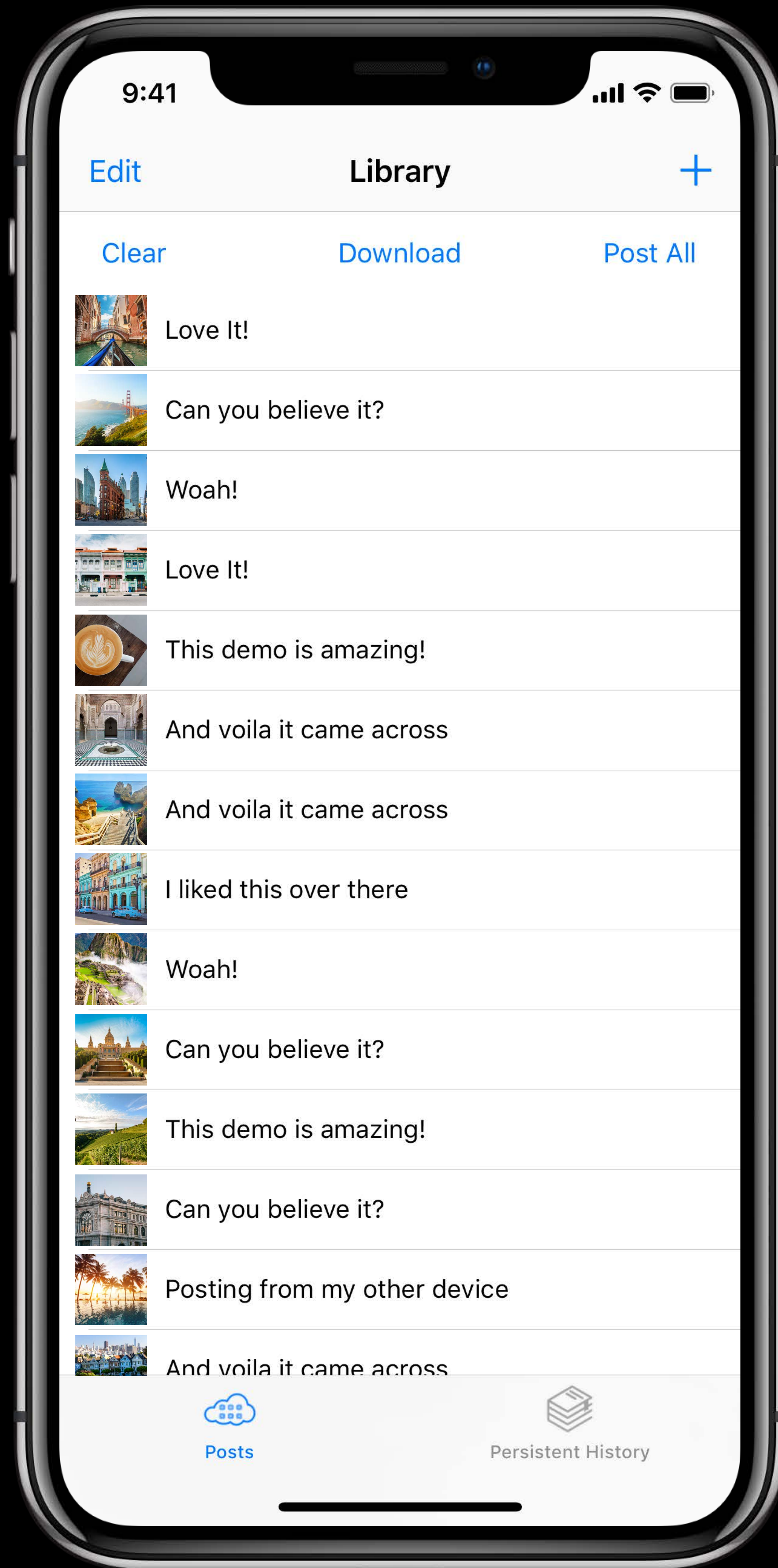


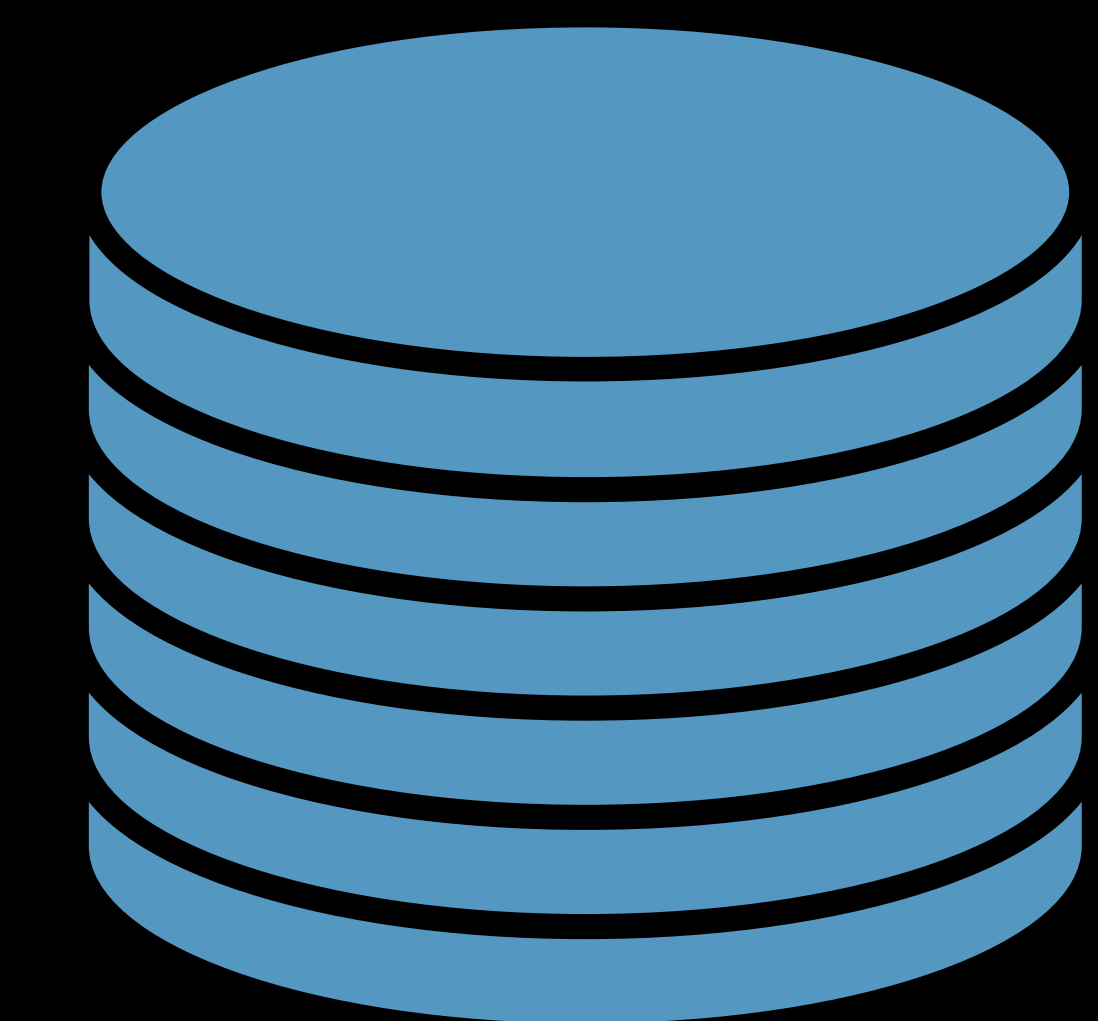
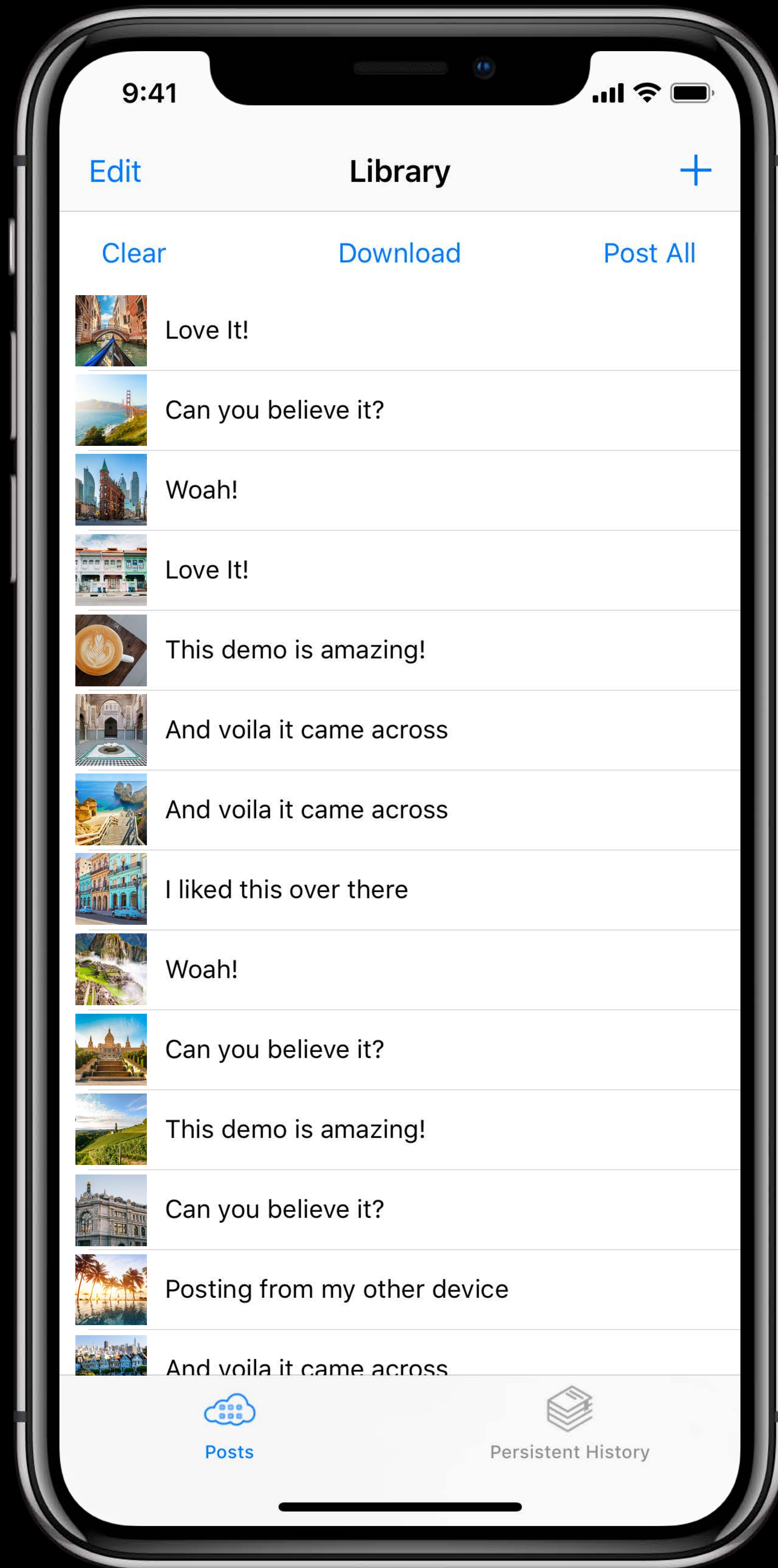


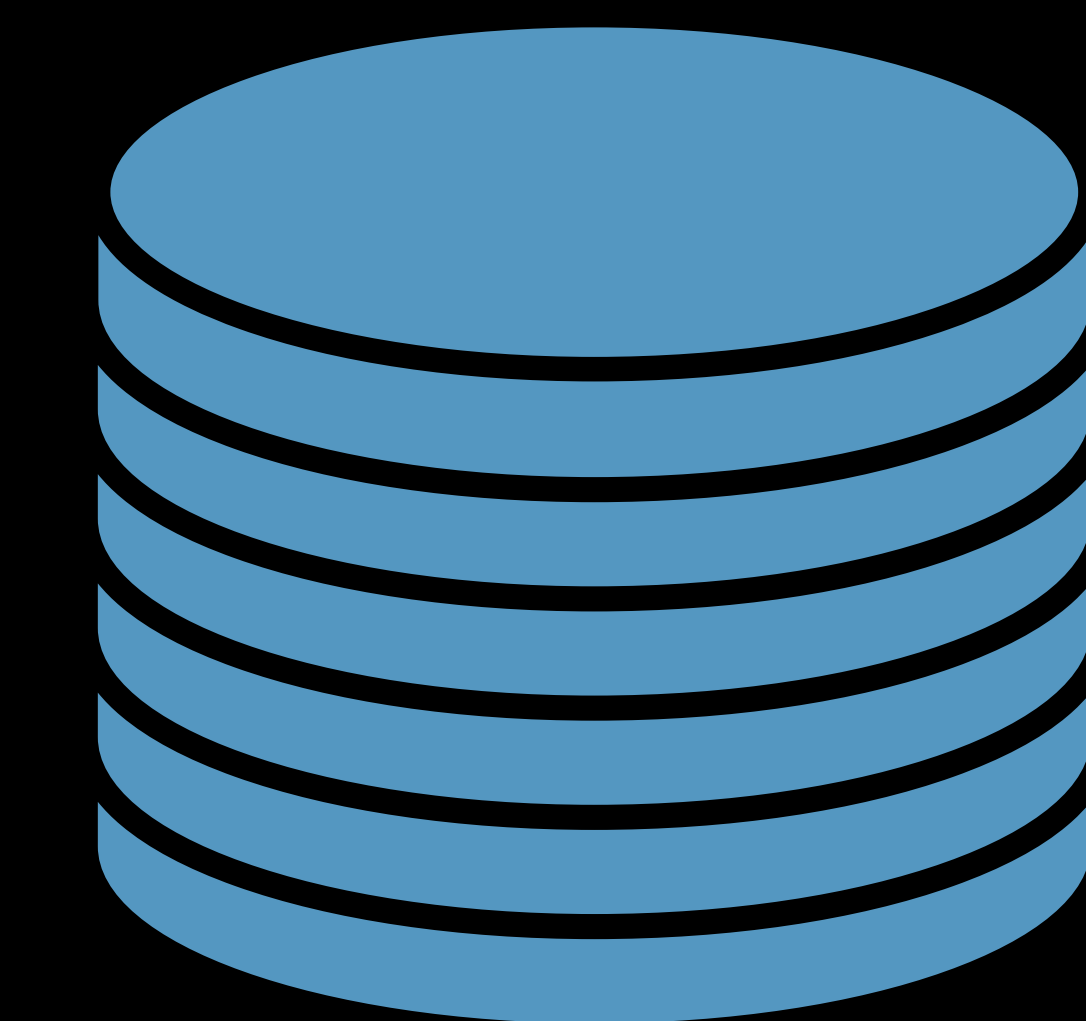
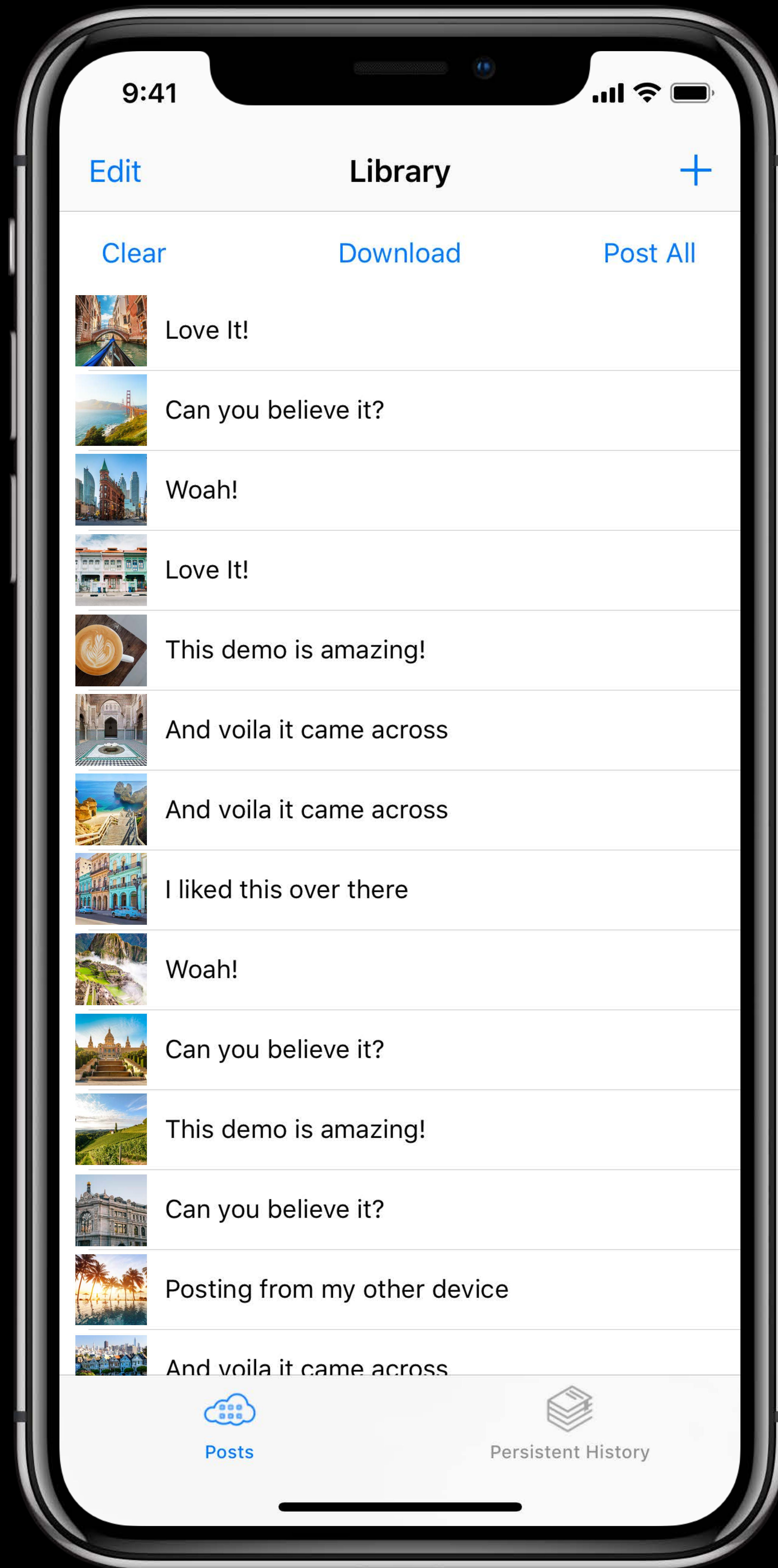












9:41



Edit

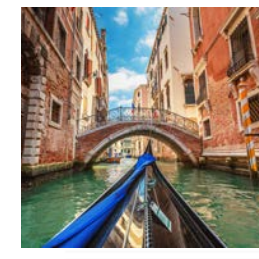
Library



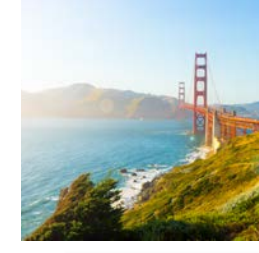
Clear

Download

Post All

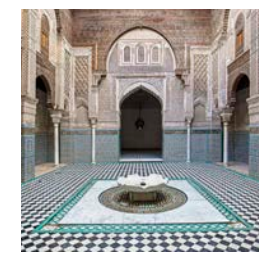
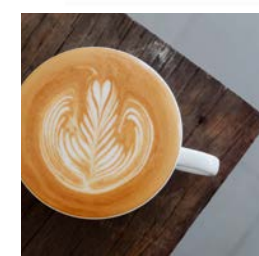


Love It!

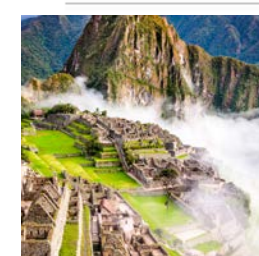


Can you believe it?

Love It!

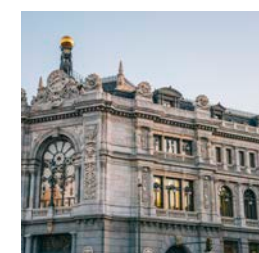


And voila it came across

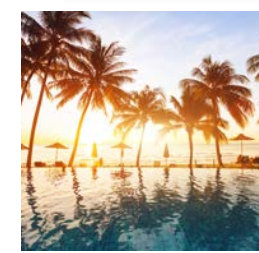


Woah!

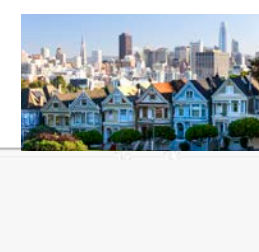
This demo is amazing!



Can you believe it?



Posting from my other device



And voila it came across



Posts



Persistent History

# Consistent Experiences with Query Generations

# Consistent Experiences with Query Generations

CoreData is here to help

# Consistent Experiences with Query Generations

CoreData is here to help

- Query generations

# Consistent Experiences with Query Generations

CoreData is here to help

- Query generations
  - Isolate contexts from competing work



# Consistent Experiences with Query Generations

CoreData is here to help

- Query generations
  - Isolate contexts from competing work
  - Provide a consistent, durable view of the database

# Consistent Experiences with Query Generations

CoreData is here to help

- Query generations
  - Isolate contexts from competing work
  - Provide a consistent, durable view of the database
  - Requires WAL journal mode

```
let context = container.viewContext
context.performAndWait {
    try context.setQueryGenerationFrom(NSQueryGenerationToken.current)
    try self.fetchedResultsController.performFetch()
}
self.tableView.reloadData()
```

```
let context = container.viewContext
context.performAndWait {
    try context.setQueryGenerationFrom(NSQueryGenerationToken.current)
    try self.fetchedResultsController.performFetch()
}
self.tableView.reloadData()
```

```
@objc func contextDidSave(notification: Notification) {  
    let context = self.viewContext  
    context.perform {  
        context.mergeChanges(fromContextDidSave: notification)  
    }  
}
```

```
@objc func contextDidSave(notification: Notification) {  
    let context = self.viewContext  
    context.perform {  
        context.mergeChanges(fromContextDidSave: notification)  
    }  
}
```

9:41



Edit

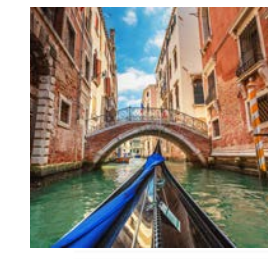
Library



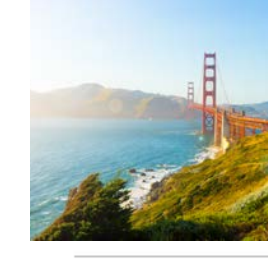
Clear

Download

Post All



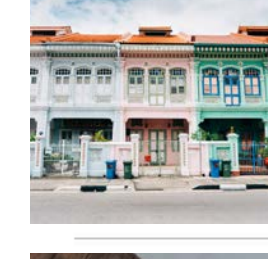
Love It!



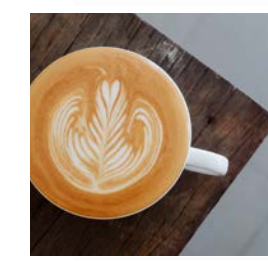
Can you believe it?



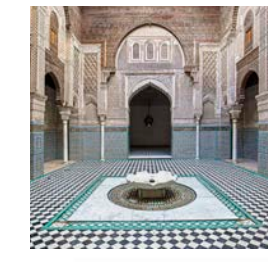
Woah!



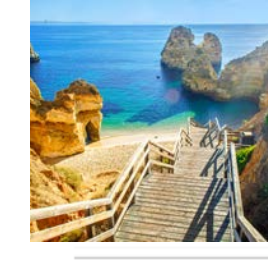
Love It!



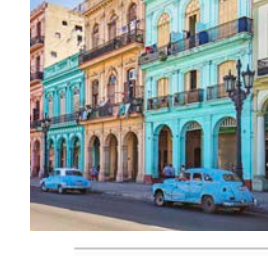
This demo is amazing!



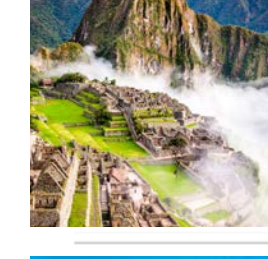
And voila it came across



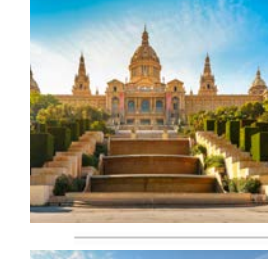
And voila it came across



I liked this over there



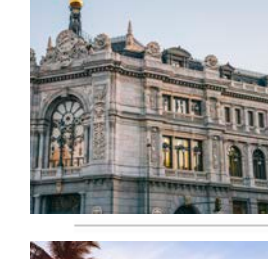
Woah!



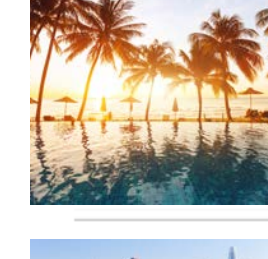
Can you believe it?



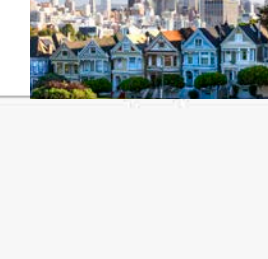
This demo is amazing!



Can you believe it?



Posting from my other device



And voila it came across



Posts



Persistent History

9:41



Edit

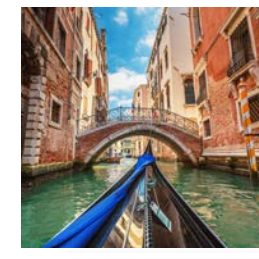
Library



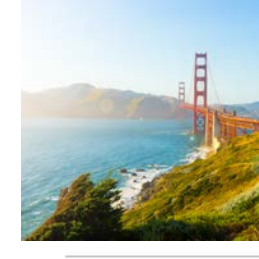
Clear

Download

Post All



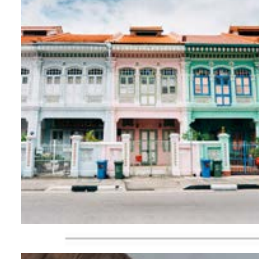
Love It!



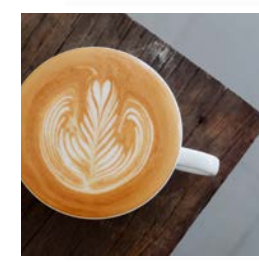
Can you believe it?



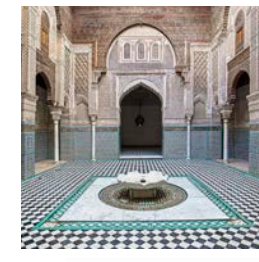
Woah!



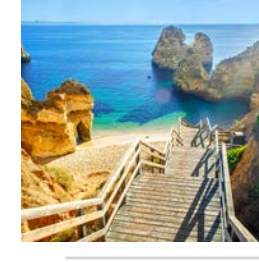
Love It!



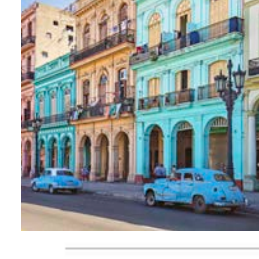
This demo is amazing!



And voila it came across



And voila it came across



I liked this over there



Woah!

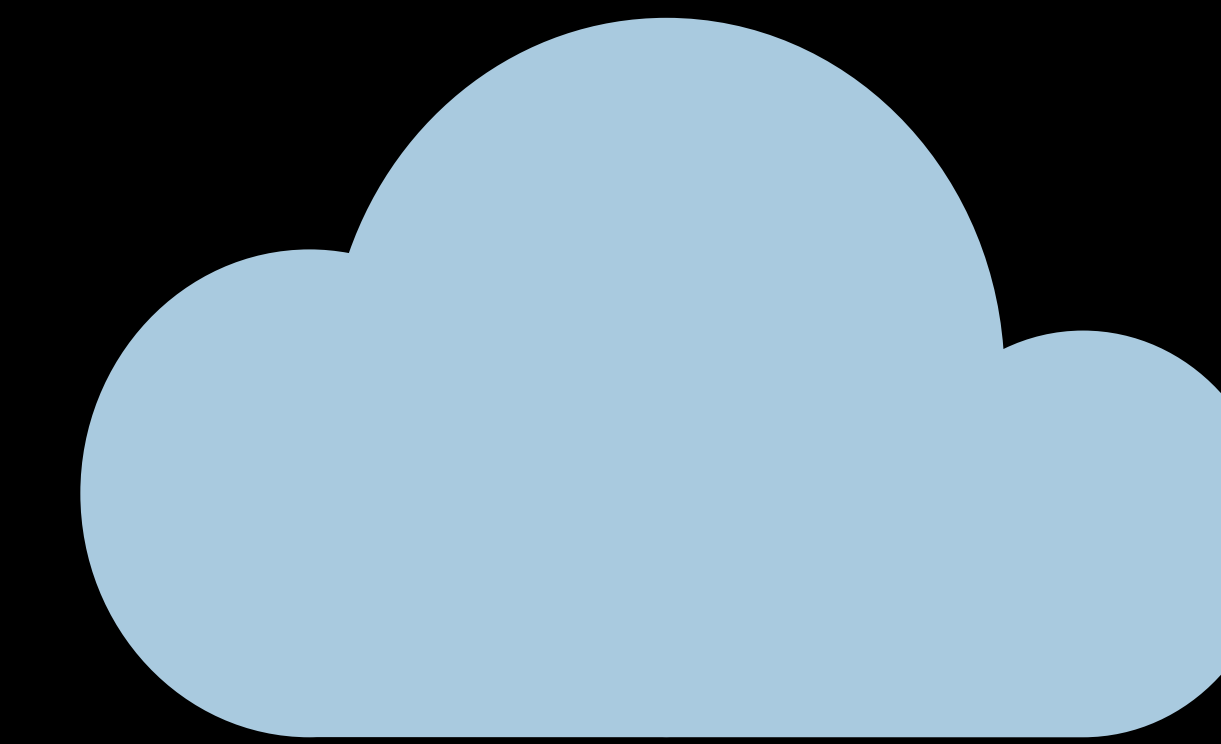
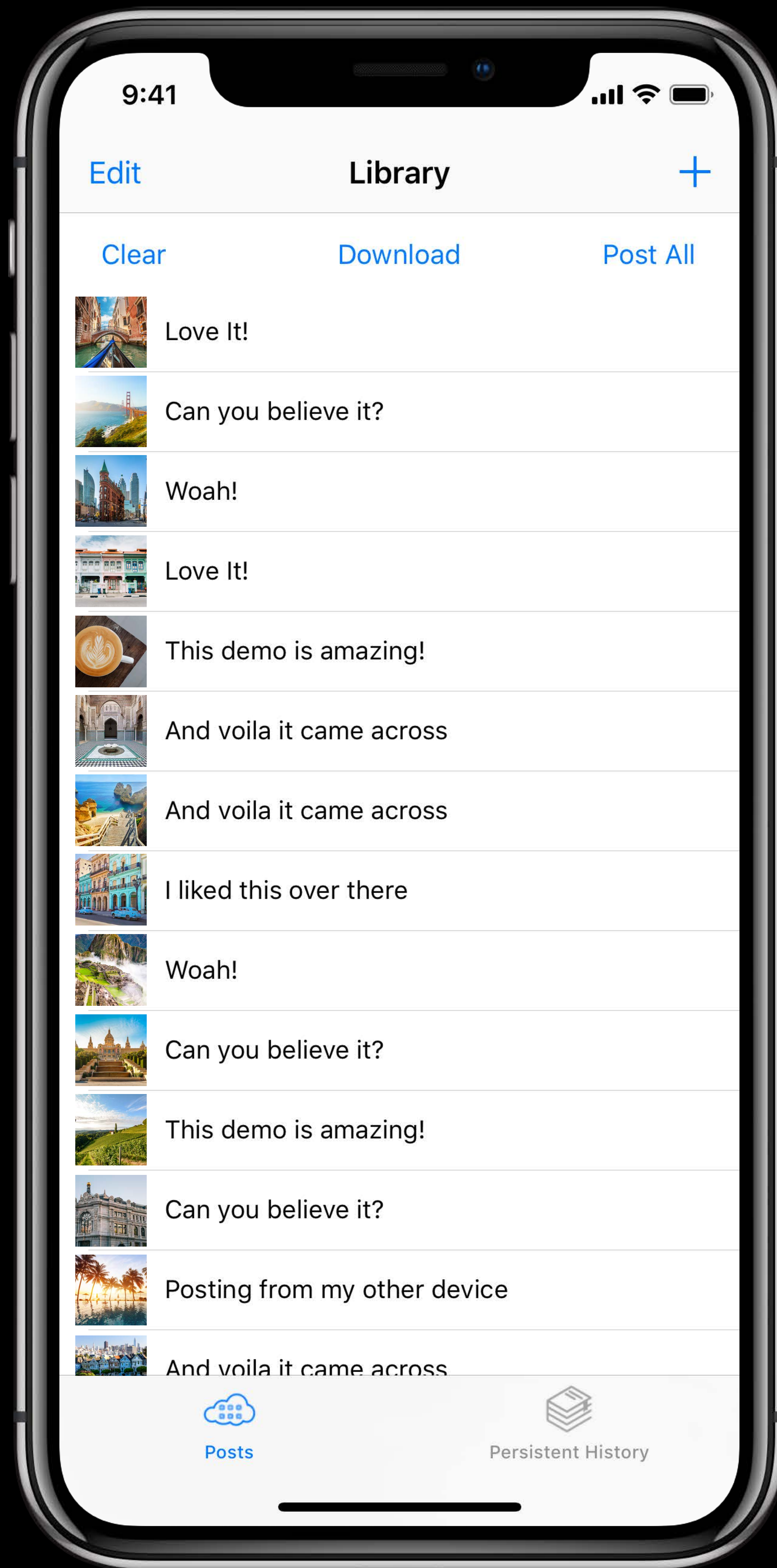


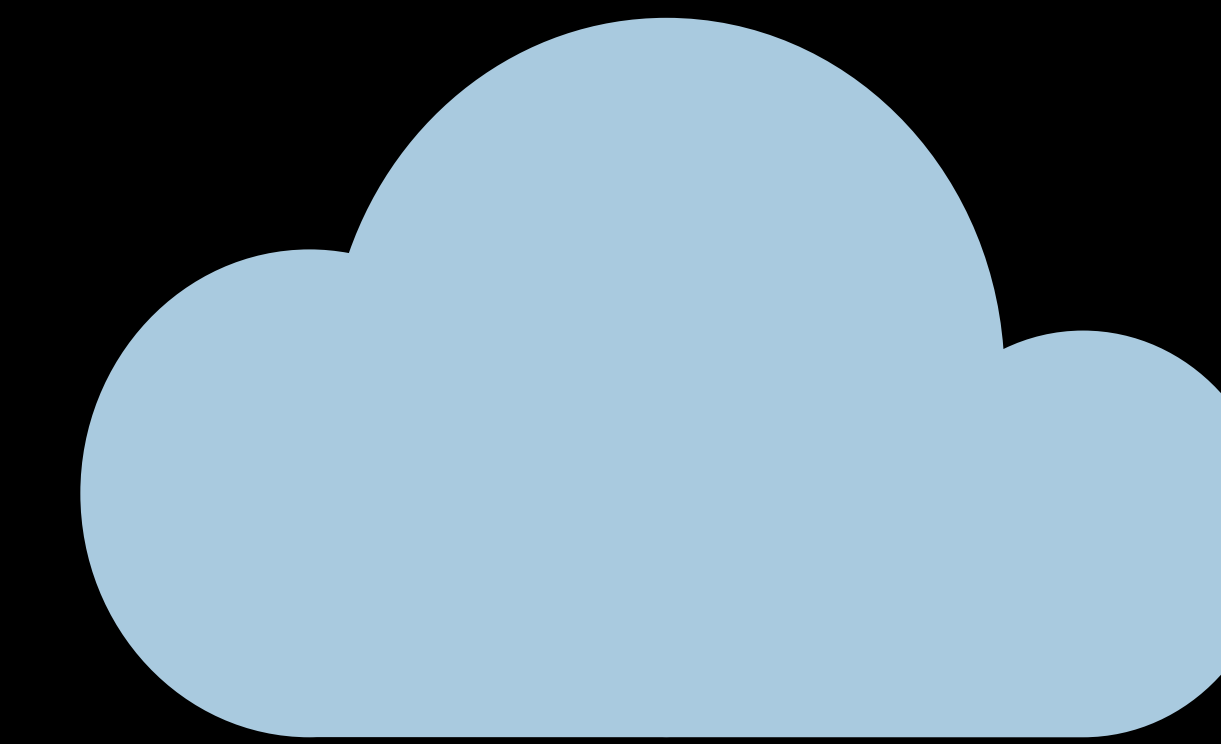
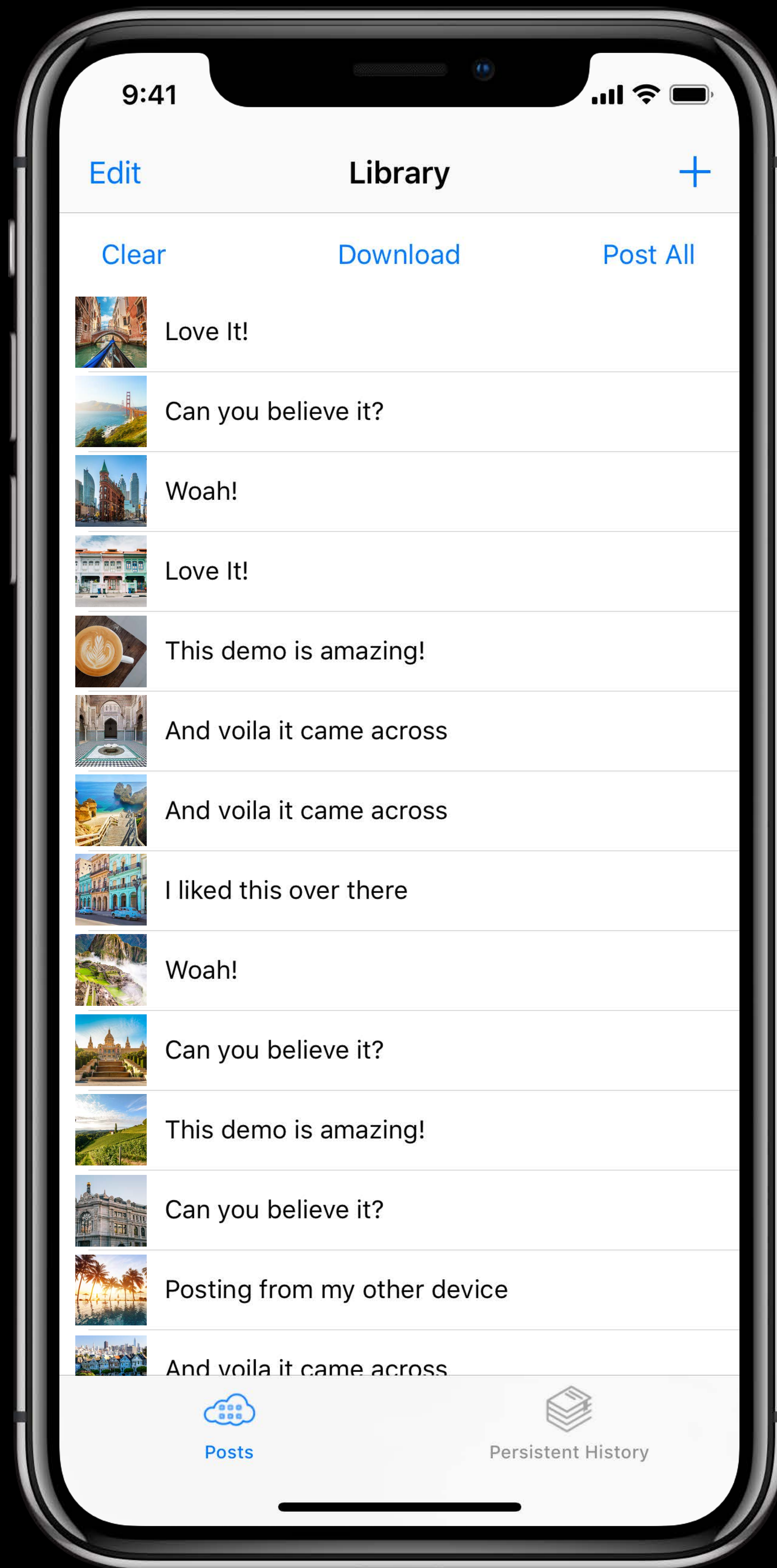
Posts



Persistent History







# Filtering Updates with Persistent History Tracking

# Filtering Updates with Persistent History Tracking

New in iOS 11 and macOS 10.13

# Filtering Updates with Persistent History Tracking

New in iOS 11 and macOS 10.13

Persisted record of each transaction

```
@available(iOS 11.0, *)
open class NSPersistentHistoryChange : NSObject, NSCopying {
    @NSCopying open var changedObjectID: NSManagedObjectID { get }
    open var updatedProperties: Set<NSPropertyDescription>? { get }
    //...
}
```

```
@available(iOS 11.0, *)
open class NSPersistentHistoryChange : NSObject, NSCopying {
    @NSCopying open var changedObjectID: NSManagedObjectID { get }
    open var updatedProperties: Set<NSPropertyDescription>? { get }
    //...
}
```

```
@available(iOS 11.0, *)
open class NSPersistentHistoryTransaction : NSObject, NSCopying {
    open var changes: [NSPersistentHistoryChange]? { get }
    //...
    open func objectIDNotification() -> Notification
}
```

# Filtering Updates with Persistent History Tracking



# Filtering Updates with Persistent History Tracking

changedObjectID	type
Post/p1	insert
Post/p2	insert
Post/p3	insert
Post/p4	insert
Post/p5	insert

```
let context = self.viewContext
let transaction: NSPersistentHistoryTransaction = //...
context.perform {
    context.mergeChanges(fromContextDidSave: transaction.objectIDNotification())
}
```

```
let context = self.viewContext
let transaction: NSPersistentHistoryTransaction = //...
context.perform {
    context.mergeChanges(fromContextDidSave: transaction.objectIDNotification())
}
```

# Filtering Updates with Persistent History Tracking

changedObjectID	type
Comment/p1	insert
Comment/p2	insert
Comment/p3	insert
Comment/p4	insert
Comment/p5	insert

```
var changes = [] as Array<NSPersistentHistoryChange>
//...
for transaction in transactions {
    let filteredChanges = transaction.changes!.filter({ (change) -> Bool in
        return Post.entity().name == change.changedObjectID.entity.name
    })
    changes.append(contentsOf: filteredChanges)
}
```

```
var changes = [] as Array<NSPersistentHistoryChange>
//...
for transaction in transactions {
    let filteredChanges = transaction.changes!.filter({ (change) -> Bool in
        return Post.entity().name == change.changedObjectID.entity.name
    })
    changes.append(contentsOf: filteredChanges)
}
```

9:41



Edit

Library



Clear

Download

Post All



Love It!



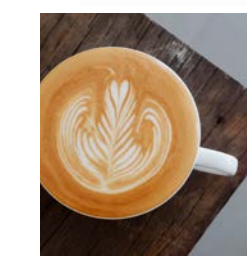
Can you believe it?



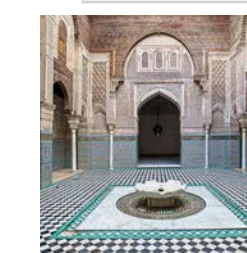
Woah!



Love It!



This demo is amazing!



And voila it came across



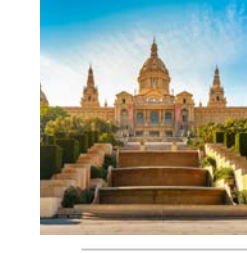
And voila it came across



I liked this over there



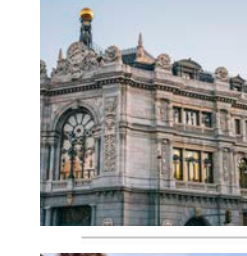
Woah!



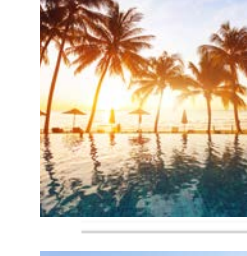
Can you believe it?



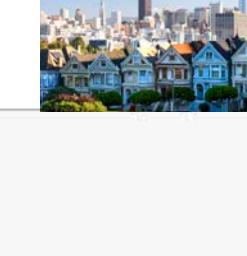
This demo is amazing!



Can you believe it?



Posting from my other device



And voila it came across

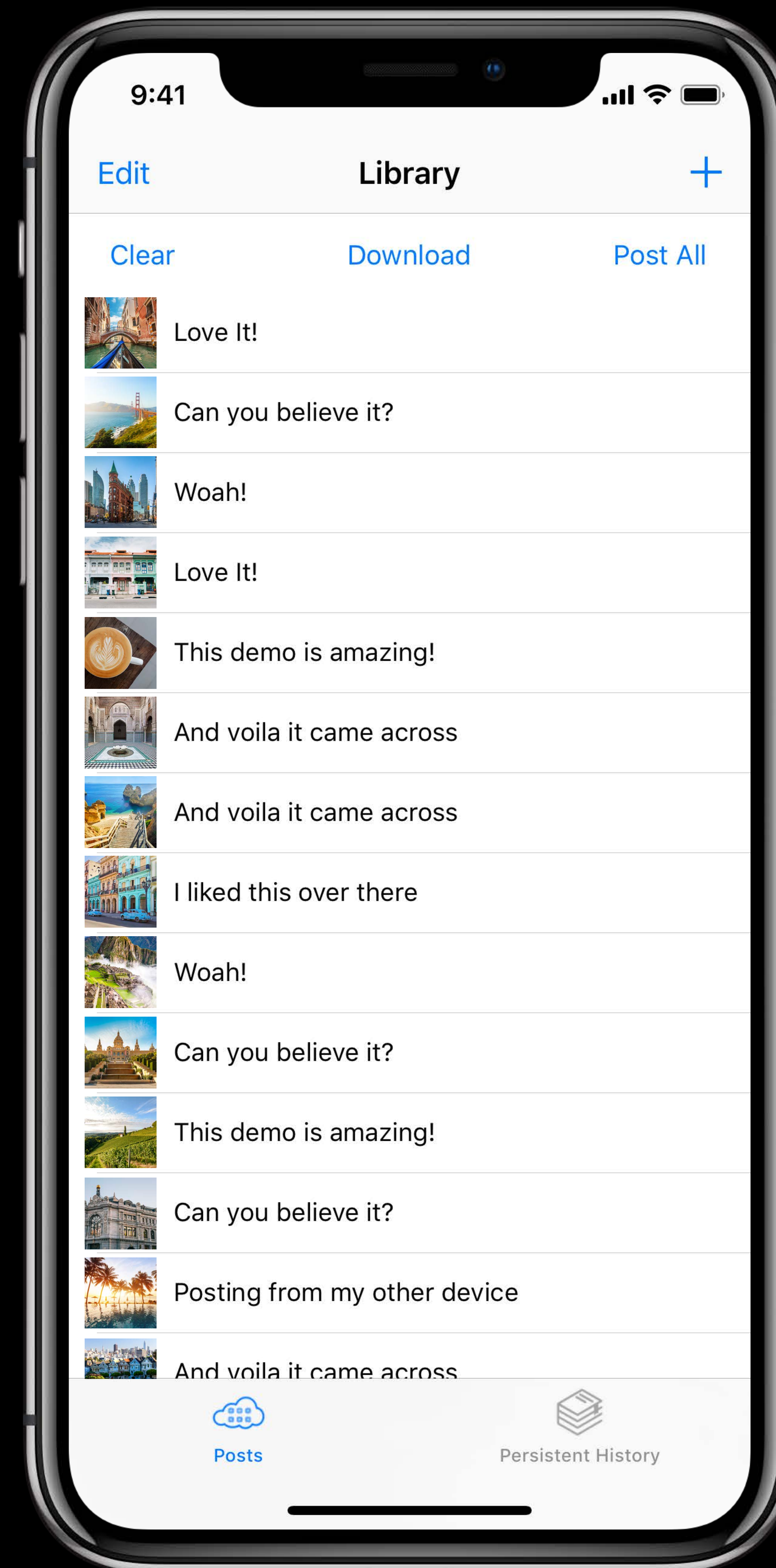


Posts



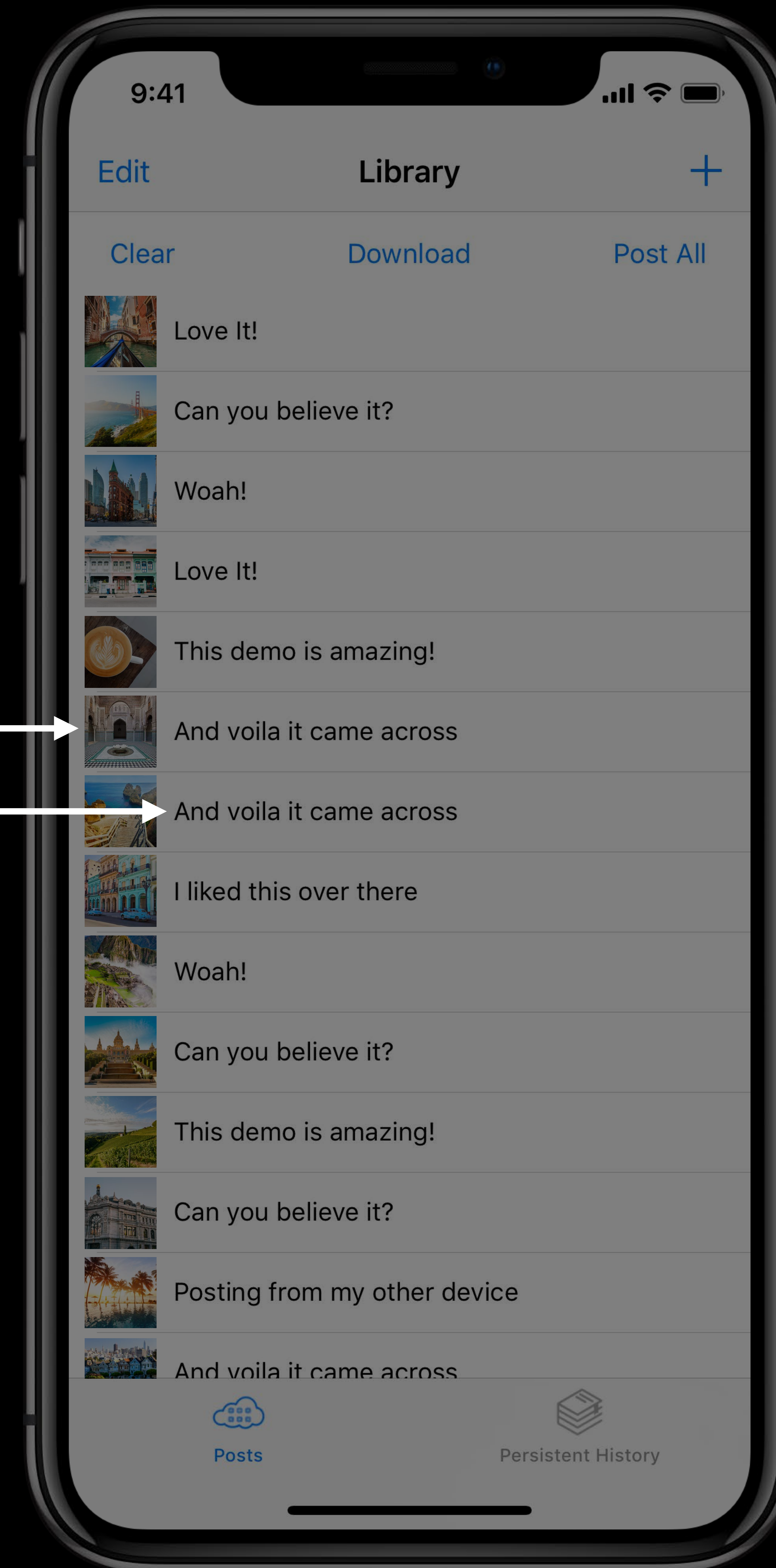
Persistent History

```
extension Post {  
    @NSManaged public var image: Data?  
    @NSManaged public var title: String?  
    // ...  
}
```





```
extension Post {  
    @Managed public var image: Data?  
    @Managed public var title: String?  
    // ...  
}
```



```
var changes = [] as Array<NSPersistentHistoryChange>
//...
for transaction in transactions {
    let filteredChanges = transaction.changes!.filter({ (change) -> Bool in
        if let updatedProperties = change.updatedProperties {
            return updatedProperties.contains(where: { (property) -> Bool in
                return property.name == "image" ||
                    property.name == "title"
            })
        } else {
            return false;
        }
    })
    changes.append(contentsOf: filteredChanges)
}
```

```
var changes = [] as Array<NSPersistentHistoryChange>
//...
for transaction in transactions {
    let filteredChanges = transaction.changes!.filter({ (change) -> Bool in
        if let updatedProperties = change.updatedProperties {
            return updatedProperties.contains(where: { (property) -> Bool in
                return property.name == "image" ||
                    property.name == "title"
            })
        } else {
            return false;
        }
    })
    changes.append(contentsOf: filteredChanges)
}
```

# Bulk Editing with Batch Operations

9:41



5 Photos Selected

Cancel

**Point Reyes National Seashore**

Mar 12, 2011 · Inverness, CA

Deselect



**Pingeyjarsveit, Northeast Icela...**

Aug 8, 2012 · Goðafossvegur

Deselect



**Djúpavogshreppur**

Aug 8, 2012 · East Iceland

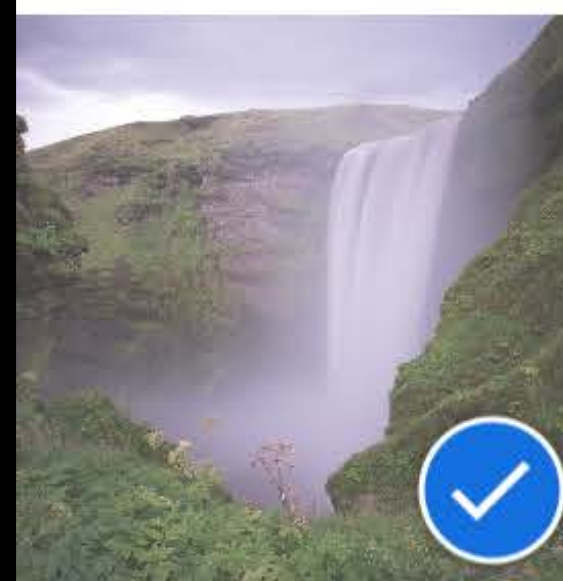
Deselect



**Rangárbing eystra**

Aug 8, 2012 · South Iceland

Deselect



5 Photos



Add To



```
let context = self.container.newBackgroundContext()
context.perform {
    let bur = NSBatchUpdateRequest(entity: Photo.entity())
    bur.propertiesToUpdate = ["favorite": NSEExpression(forConstantValue: 1)]
    try context.execute(bur)
}
```

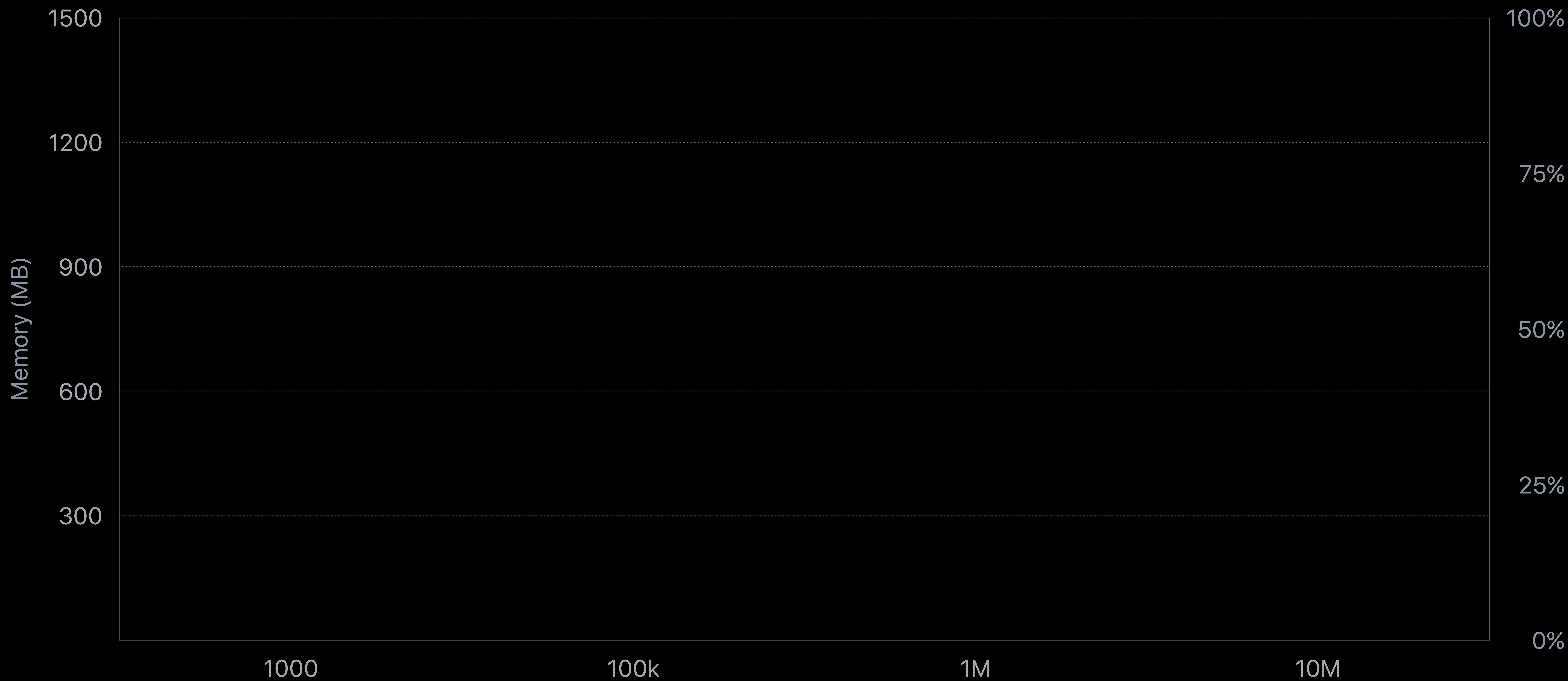
```
let context = self.container.newBackgroundContext()
context.perform {
    let bur = NSBatchUpdateRequest(entity: Photo.entity())
    bur.propertiesToUpdate = ["favorite": NSEExpression(forConstantValue: 1)]
    try context.execute(bur)
}
```

```
let context = self.container.newBackgroundContext()
context.perform {
    let bdr = NSBatchDeleteRequest(entity: Photo.entity())
    try context.execute(bdr)
}
```

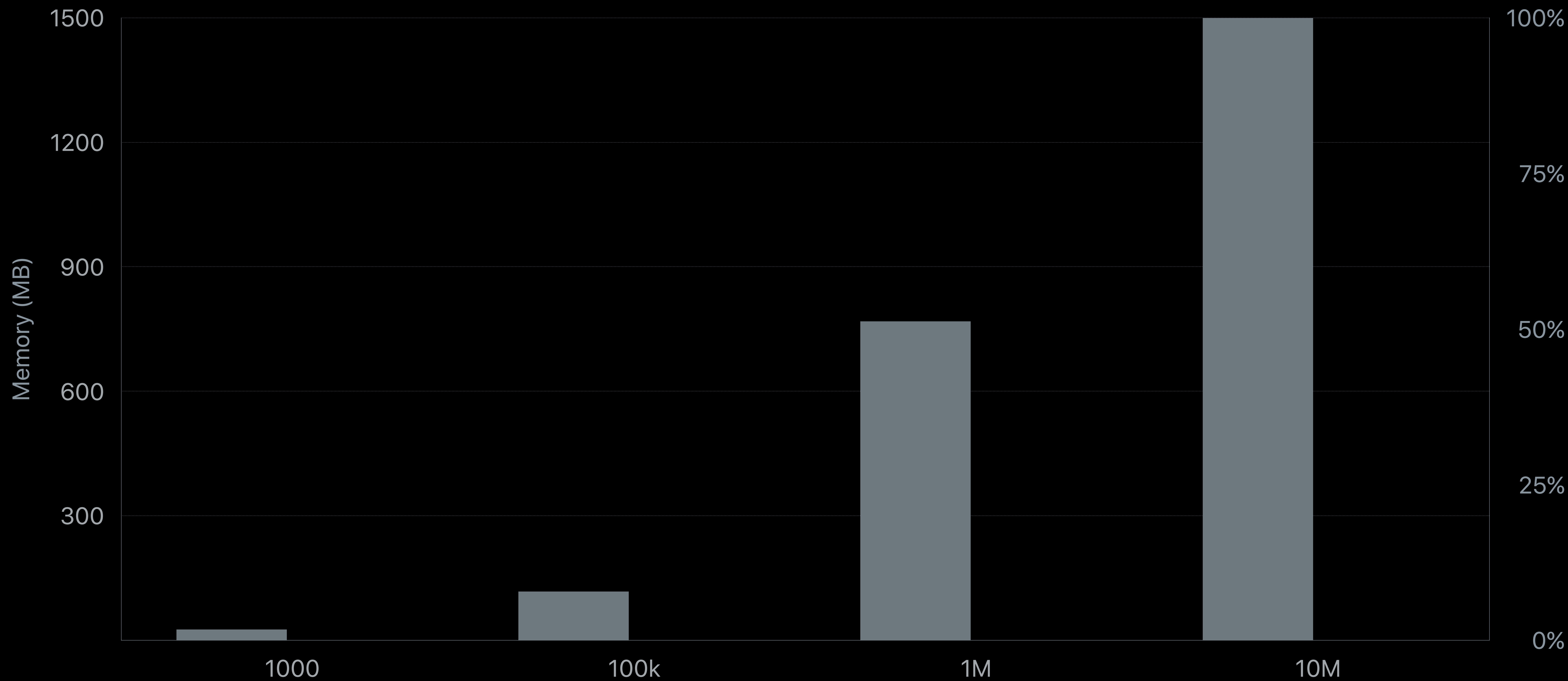


# **NSManagedObject.delete vs NSBatchDeleteRequest**

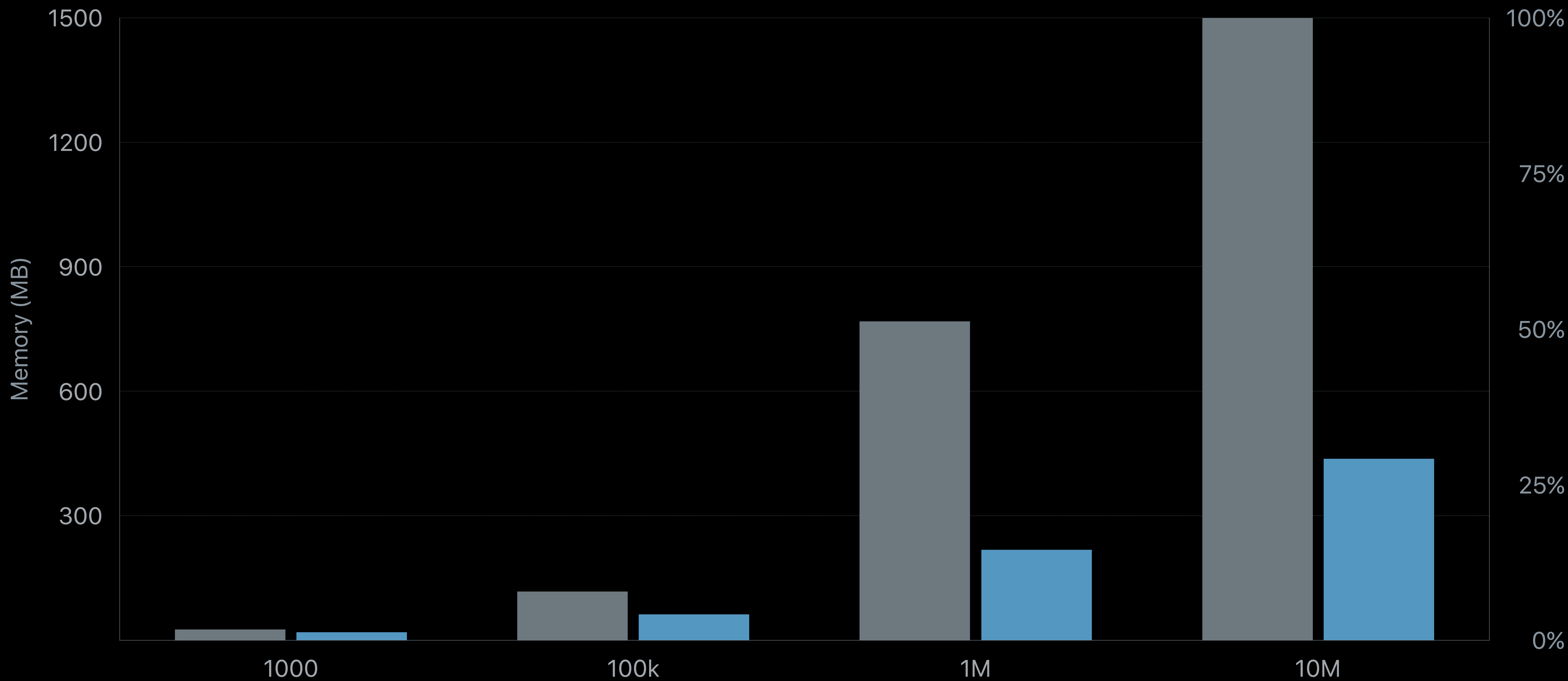
# NSManagedObject.delete vs NSBatchDeleteRequest



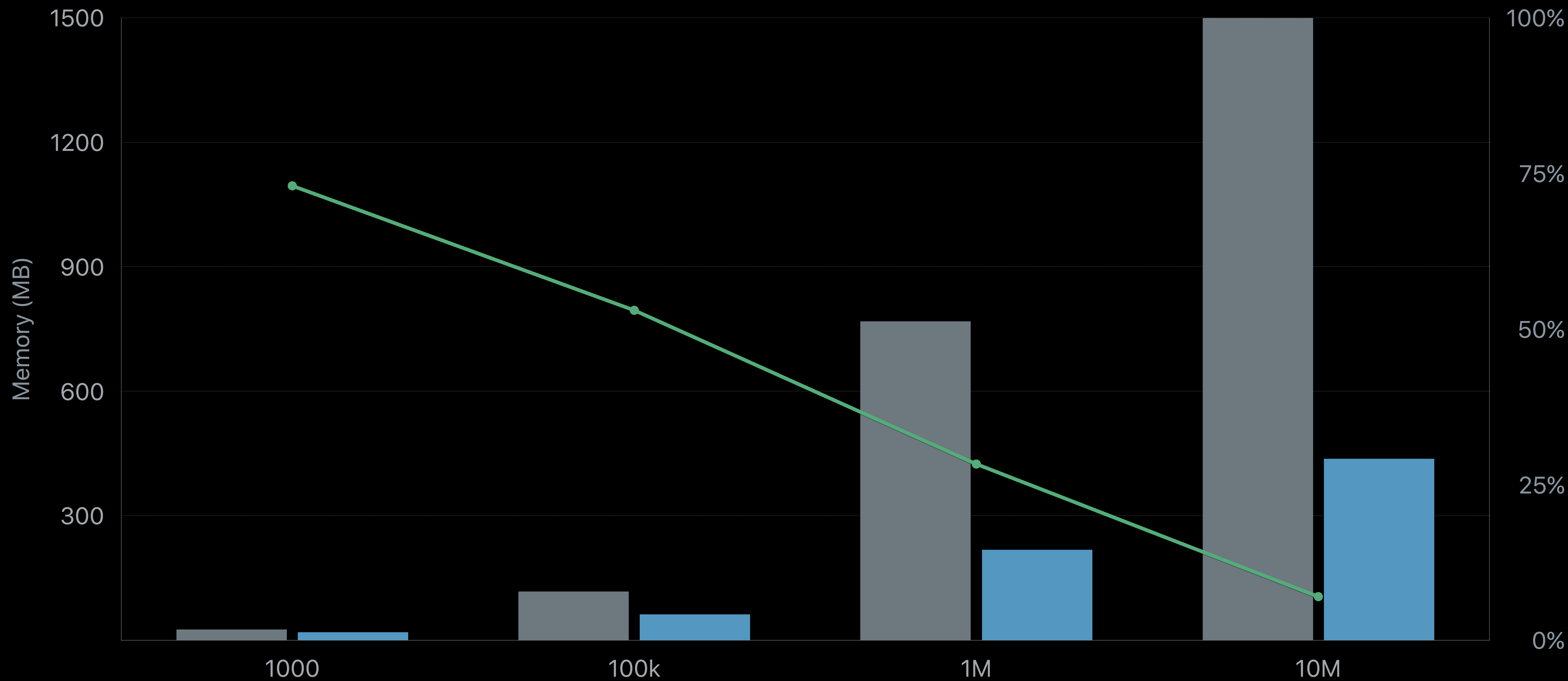
# NSManagedObject.delete vs NSBatchDeleteRequest



# NSManagedObject.delete vs NSBatchDeleteRequest



# NSManagedObject.delete vs NSBatchDeleteRequest



```
let context = container.viewContext
context.performAndWait {
    let request = NSPersistentHistoryChangeRequest()
    request.resultType = .transactionsAndChanges
    do {
        let result = try context.execute(request) as! NSPersistentHistoryResult
        let transactions = result.result as! Array<NSPersistentHistoryTransaction>
        for transaction in transactions {
            context.mergeChanges(fromContextDidSave: transaction.objectIDNotification())
        }
    } catch {
        //...
    }
}
```

```
let context = container.viewContext
context.performAndWait {
    let request = NSPersistentHistoryChangeRequest()
    request.resultType = .transactionsAndChanges
    do {
        let result = try context.execute(request) as! NSPersistentHistoryResult
        let transactions = result.result as! Array<NSPersistentHistoryTransaction>
        for transaction in transactions {
            context.mergeChanges(fromContextDidSave: transaction.objectIDNotification())
        }
    } catch {
        //...
    }
}
```

```
let context = container.viewContext
context.performAndWait {
    let request = NSPersistentHistoryChangeRequest()
    request.resultType = .transactionsAndChanges
    do {
        let result = try context.execute(request) as! NSPersistentHistoryResult
        let transactions = result.result as! Array<NSPersistentHistoryTransaction>
        for transaction in transactions {
            context.mergeChanges(fromContextDidSave: transaction.objectIDNotification())
        }
    } catch {
        //...
    }
}
```



# Working Efficiently with CoreData

**Help Future You**

# Help Future You

NSKeyedArchiver API is changing

# Help Future You

NSKeyedArchiver API is changing

Default value transformer

# Help Future You

NSKeyedArchiver API is changing

Default value transformer

- Old—nil or NSKeyedUnarchiveFromDataTransformerName
- New—NSSecureUnarchiveFromDataTransformerName

# Adopt NSSecureUnarchiveFromDataTransformerName

The screenshot shows the Xcode Core Data model editor for a project named 'HistoryDemo'. The main window displays the 'aTransformable' attribute configuration for the 'Post' entity. The attribute is of type 'Transformable' and is optional. The 'Value Transformer Name' is set to 'Value Transformer Name', and the 'Custom Class' is 'NSObject' in the 'Global namespace'. The 'Advanced' section includes options for 'Index in Spotlight' and 'Preserve After Deletion', both of which are unchecked. The 'Deprecated' section includes the option 'Store in External Record File', which is also unchecked. The 'User Info' section is currently empty. The 'Versioning' section includes 'Hash Modifier' set to 'Version Hash Modifier' and 'Renaming ID' set to 'Renaming Identifier'. The left sidebar shows the 'ENTITIES' section with 'Post' selected, and the 'CONFIGURATIONS' section with 'Default' selected. The bottom toolbar includes 'Outline Style', 'Add Entity', 'Add Attribute', and 'Editor Style' buttons.

HistoryDemo.xcdatamodel

HistoryDemo > HistoryDemo > Histor...model > Histor...amodel > E Post > T aTransformable

ENTITIES

- Post
  - timestampIndex
  - PostsPerDay

FETCH REQUESTS

CONFIGURATIONS

- Default

Attributes

Attribute	Type
aTransformable	Transformable
identifier	UUID
image	Binary Data
posted	Integer 16
source	String
timestamp	Date
title	String
url	String

Relationships

Relationship	Destination	Inverse
--------------	-------------	---------

Fetches Properties

Fetches Property	Predicate
------------------	-----------

Attribute

Name: aTransformable

Properties:  Transient  Optional

Attribute Type: Transformable

Value Transfo...: Value Transformer Name

Custom Class: NSObject

Module: Global namespace

Advanced:  Index in Spotlight  Preserve After Deletion

Deprecated:  Store in External Record File

User Info

Key	Value
-----	-------

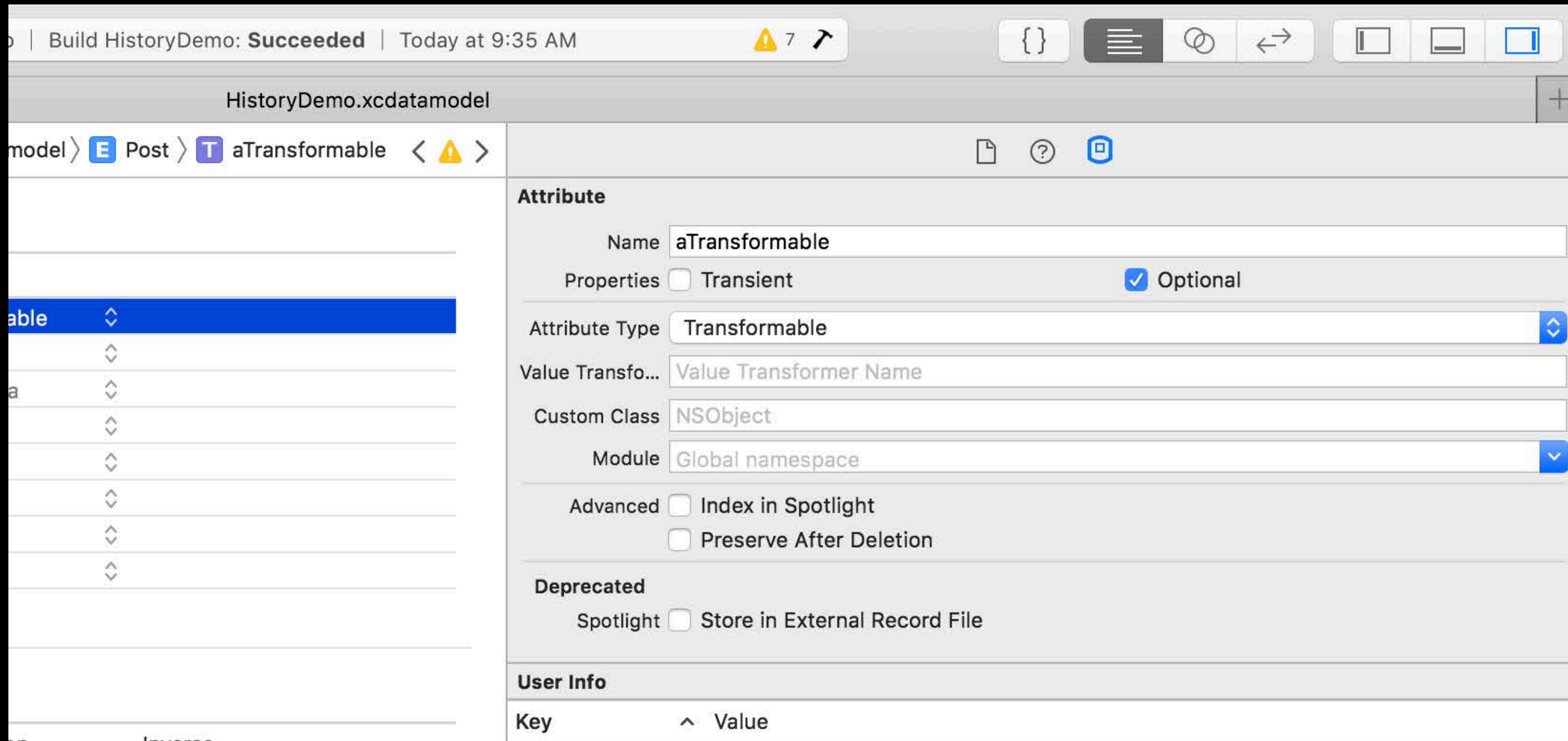
Versioning

Hash Modifier: Version Hash Modifier

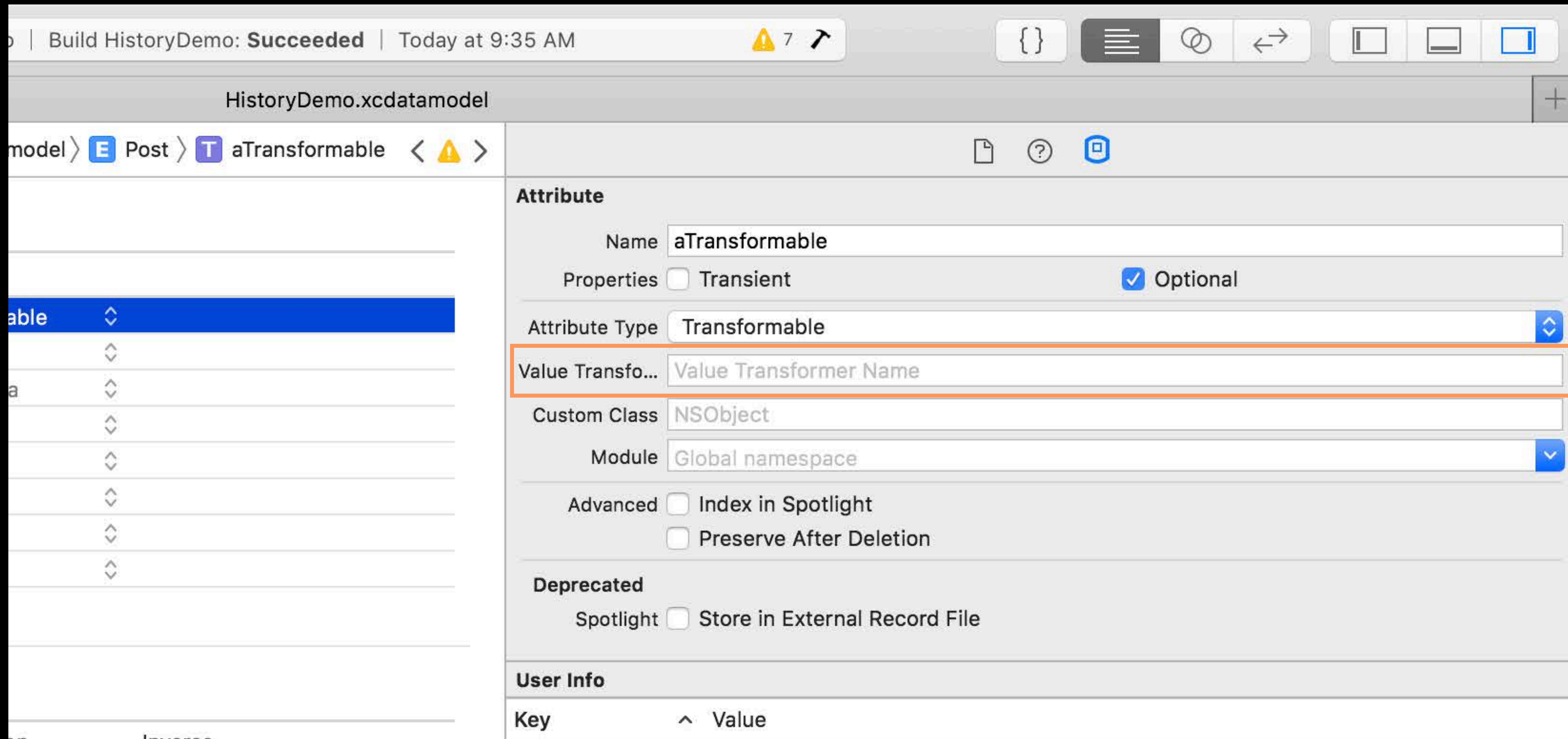
Renaming ID: Renaming Identifier

Outline Style Add Entity Add Attribute Editor Style

# Adopt NSSecureUnarchiveFromDataTransformerName

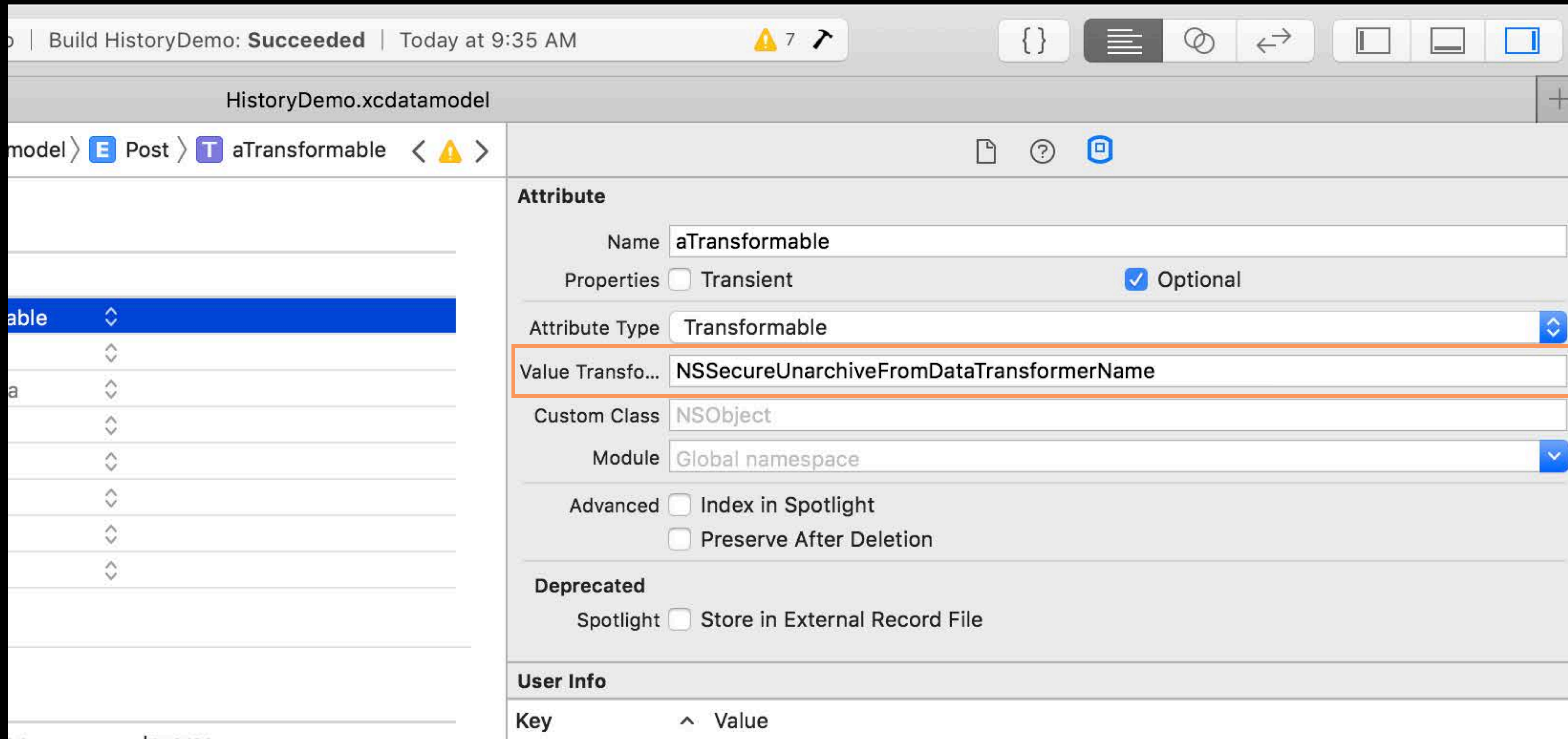


# Adopt NSSecureUnarchiveFromDataTransformerName





# Adopt NSSecureUnarchiveFromDataTransformerName



```
if let transformableAttribute = model.entitiesByName["Post"]?.attributesByName["aTransformable"] {  
    transformableAttribute.valueTransformerName = NSSecureUnarchiveFromDataTransformerName  
}
```

```
if let transformableAttribute = model.entitiesByName["Post"]?.attributesByName["aTransformable"] {  
    transformableAttribute.valueTransformerName = NSSecureUnarchiveFromDataTransformerName  
}
```

# Help Future You

NSKeyedArchiver API is changing

Default Value Transformer

- Old—nil or NSKeyedUnarchiveFromDataTransformerName
- New—NSSecureUnarchiveFromDataTransformerName

# Help Future You

NSKeyedArchiver API is changing

Default Value Transformer

- Old—nil or NSKeyedUnarchiveFromDataTransformerName
- New—NSSecureUnarchiveFromDataTransformerName

Transparent for plist types

# Help Future You

NSKeyedArchiver API is changing

Default Value Transformer

- Old—nil or NSKeyedUnarchiveFromDataTransformerName
- New—NSSecureUnarchiveFromDataTransformerName

Transparent for plist types

- Custom classes need a custom transformer
- Come see us in lab!

# Help Us Help You

HistoryDemo > iPhone X

Build HistoryDemo: **Succeeded** | Today at 4:35 PM

HDPPostGenerator.swift

HistoryDemo > iPhone X

Build 3 targets

Run Debug

Test Debug

Profile Release

Analyze Debug

Archive Release

Install Debug

Info Arguments Options Diagnostics

Arguments Passed On Launch

- com.apple.CoreData.SQLDebug 4
- com.apple.CoreData.ConcurrencyDebug 1

+ -

Environment Variables

Name	Value
<input checked="" type="checkbox"/> SQLITE_ENABLE_THREAD_ASSERTIONS	1
<input type="checkbox"/> SQLITE_AUTO_TRACE	1
<input checked="" type="checkbox"/> SQLITE_ENABLE_FILE_ASSERTIONS	1

+ -

Expand Variables Based On HistoryDemo

Duplicate Scheme Manage Schemes...  Shared Close

CoreData: annotation: Core Data multi-threading assertions enabled.

CoreData: annotation: Connecting to sqlite database file at "/path/to/db.sqlite"



```
CoreData: annotation: Core Data multi-threading assertions enabled.
CoreData: annotation: Connecting to sqlite database file at "/path/to/db.sqlite"
//...
CoreData: annotation: fetch using NSSQLiteStatement <0x600003aae0d0> on entity 'Post' with sql
text 'SELECT 0, t0.Z_PK FROM ZPOST t0 ORDER BY t0.ZTIMESTAMP DESC' returned 100000 rows with
values: (
//...
)
CoreData: annotation: total fetch execution time: 17.6644s for 100000 rows.
```

```
CoreData: annotation: Core Data multi-threading assertions enabled.
CoreData: annotation: Connecting to sqlite database file at "/path/to/db.sqlite"
//...
CoreData: annotation: fetch using NSSQLiteStatement <0x600003aae0d0> on entity 'Post' with sql
text 'SELECT 0, t0.Z_PK FROM ZPOST t0 ORDER BY t0.ZTIMESTAMP DESC' returned 100000 rows with
values: (
//...
)
CoreData: annotation: total fetch execution time: 17.6644s for 100000 rows.
CoreData: details: SQLite: EXPLAIN QUERY PLAN SELECT 0, t0.Z_PK FROM ZPOST t0 ORDER BY
t0.ZTIMESTAMP DESC
  0 0 0 SCAN TABLE ZPOST AS t0
  0 0 0 USE TEMP B-TREE FOR ORDER BY
```

```
$ sqlite3 /path/to/db.sqlite
sqlite> EXPLAIN QUERY PLAN SELECT 0, t0.Z_PK FROM ZPOST t0 ORDER BY t0.ZTIMESTAMP DESC;
QUERY PLAN
|--SCAN TABLE ZPOST AS t0
`--USE TEMP B-TREE FOR ORDER BY

sqlite> .expert
sqlite> SELECT 0, t0.Z_PK FROM ZPOST t0 ORDER BY t0.ZTIMESTAMP DESC;
CREATE INDEX ZPOST_idx_43b4963d ON ZPOST(ZTIMESTAMP DESC);

SCAN TABLE ZPOST AS t0 USING COVERING INDEX ZPOST_idx_43b4963d
```

```
$ sqlite3 /path/to/db.sqlite
```

```
sqlite> EXPLAIN QUERY PLAN SELECT 0, t0.Z_PK FROM ZPOST t0 ORDER BY t0.ZTIMESTAMP DESC;
```

```
QUERY PLAN
```

```
|--SCAN TABLE ZPOST AS t0
```

```
`--USE TEMP B-TREE FOR ORDER BY
```

```
sqlite> .expert
```

```
sqlite> SELECT 0, t0.Z_PK FROM ZPOST t0 ORDER BY t0.ZTIMESTAMP DESC;
```

```
CREATE INDEX ZPOST_idx_43b4963d ON ZPOST(ZTIMESTAMP DESC);
```

```
SCAN TABLE ZPOST AS t0 USING COVERING INDEX ZPOST_idx_43b4963d
```

```
$ sqlite3 /path/to/db.sqlite
```

```
sqlite> EXPLAIN QUERY PLAN SELECT 0, t0.Z_PK FROM ZPOST t0 ORDER BY t0.ZTIMESTAMP DESC;
```

```
QUERY PLAN
```

```
|--SCAN TABLE ZPOST AS t0
```

```
`--USE TEMP B-TREE FOR ORDER BY
```

```
sqlite> .expert
```

```
sqlite> SELECT 0, t0.Z_PK FROM ZPOST t0 ORDER BY t0.ZTIMESTAMP DESC;
```

```
CREATE INDEX ZPOST_idx_43b4963d ON ZPOST(ZTIMESTAMP DESC);
```

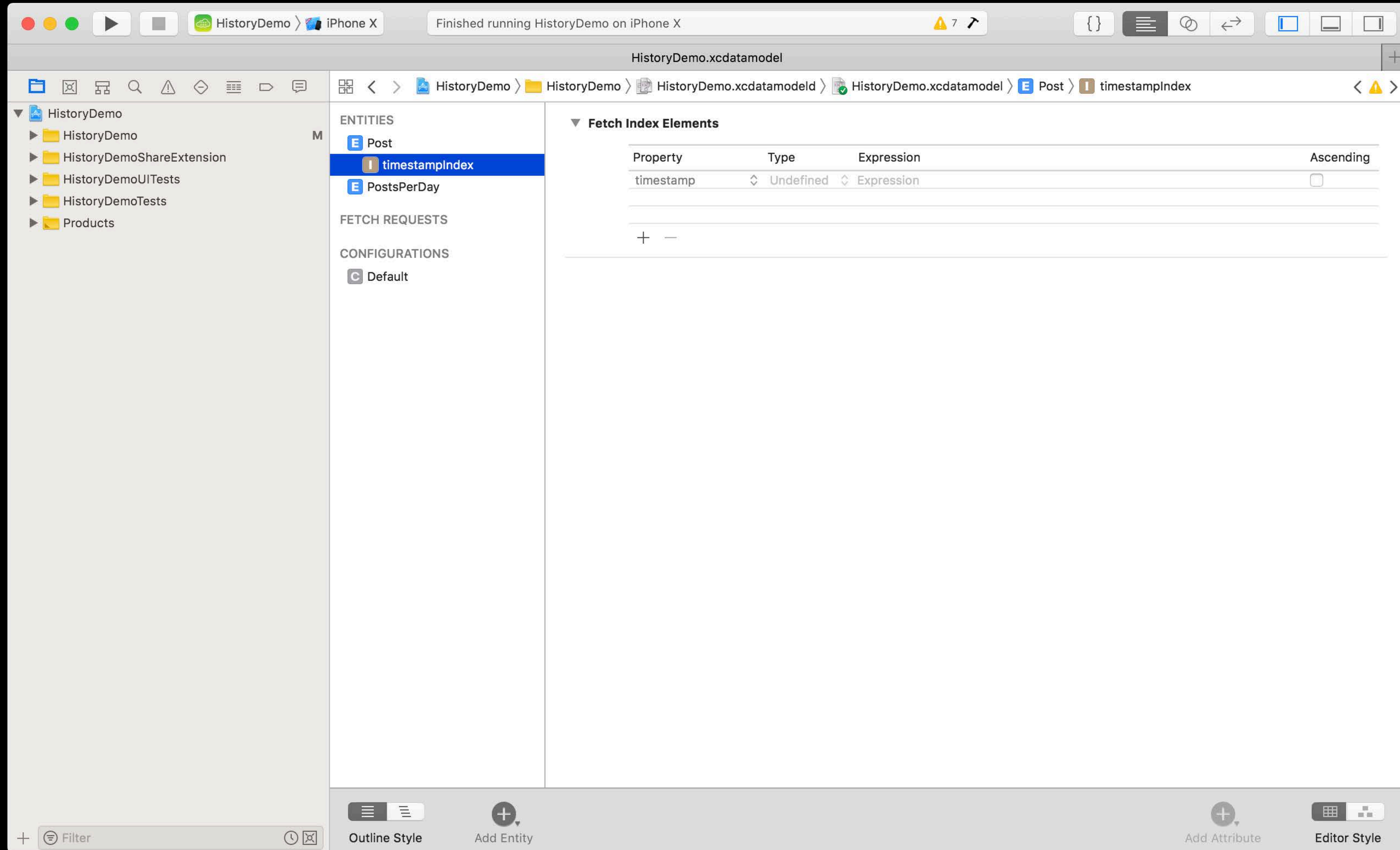
```
SCAN TABLE ZPOST AS t0 USING COVERING INDEX ZPOST_idx_43b4963d
```

```
$ sqlite3 /path/to/db.sqlite
sqlite> EXPLAIN QUERY PLAN SELECT 0, t0.Z_PK FROM ZPOST t0 ORDER BY t0.ZTIMESTAMP DESC;
QUERY PLAN
|--SCAN TABLE ZPOST AS t0
`--USE TEMP B-TREE FOR ORDER BY
```

```
sqlite> .expert
sqlite> SELECT 0, t0.Z_PK FROM ZPOST t0 ORDER BY t0.ZTIMESTAMP DESC;
CREATE INDEX ZPOST_idx_43b4963d ON ZPOST(ZTIMESTAMP DESC);
```

```
SCAN TABLE ZPOST AS t0 USING COVERING INDEX ZPOST_idx_43b4963d
```

# Performance Analysis with sqlite3



CoreData: sql: SELECT 0, t0.Z\_PK FROM ZPOST t0 ORDER BY t0.ZTIMESTAMP DESC

TraceSQL(0x7fac11f0c200): SELECT 0, t0.Z\_PK FROM ZPOST t0 ORDER BY t0.ZTIMESTAMP DESC



CoreData: sql: SELECT 0, t0.Z\_PK FROM ZPOST t0 ORDER BY t0.ZTIMESTAMP DESC

TraceSQL(0x7fac11f0c200): SELECT 0, t0.Z\_PK FROM ZPOST t0 ORDER BY t0.ZTIMESTAMP DESC

//...

CoreData: details: SQLite: EXPLAIN QUERY PLAN SELECT 0, t0.Z\_PK FROM ZPOST t0 ORDER BY t0.ZTIMESTAMP DESC

```
0 0 0 SCAN TABLE ZPOST AS t0 USING COVERING INDEX Z_Post_timestampIndex
```

# Better Indexing

The screenshot shows the Xcode interface for configuring a CoreData model. The breadcrumb path is HistoryDemo > HistoryDemo > History...amodel > History...tamodel > E Post > I byLocation. The left sidebar shows a project tree with folders for HistoryDemo, HistoryDemoShareExtension, HistoryDemoUITests, HistoryDemoTests, and Products. The main area is divided into three panes:

- ENTITIES:** Lists 'Post' with sub-entities 'timestampIndex' and 'byLocation' (selected), and 'PostsPerDay'.
- Fetch Index Elements:** A table defining the index structure for the 'byLocation' fetch request.

Property	Type	Expression	Ascending
latitude	Undefined	Expression	<input checked="" type="checkbox"/>
longitude	Undefined	Expression	<input checked="" type="checkbox"/>
- Fetch Index:** Configuration for the 'byLocation' fetch request, showing Name: 'byLocation' and Type: 'R-Tree'. Below it, the 'Partial Index' section shows a Predicate field.

At the bottom, there are navigation buttons: Filter, Outline Style, Add Entity, Add Attribute, and Editor Style.

# Better Indexing

The screenshot shows the Xcode interface for configuring a CoreData model. The breadcrumb path is HistoryDemo > HistoryDemo > History...amodeld > History...tamodel > Post > byLocation. The left sidebar shows the 'ENTITIES' section with 'Post' selected, and the 'byLocation' index highlighted. The main area displays the 'Fetch Index Elements' table, and the right sidebar shows the 'Fetch Index' configuration with 'Name' set to 'byLocation' and 'Type' set to 'R-Tree'.

HistoryDemo | Build HistoryDemo: **Succeeded** | Today at 12:42 PM

HistoryDemo.xcdatamodel

HistoryDemo > HistoryDemo > History...amodeld > History...tamodel > Post > byLocation

**ENTITIES**

- Post
- timestampIndex
- byLocation**
- PostsPerDay

**FETCH REQUESTS**

**CONFIGURATIONS**

- Default

**Fetch Index Elements**

Property	Type	Expression	Ascending
latitude	Undefined	Expression	<input checked="" type="checkbox"/>
longitude	Undefined	Expression	<input checked="" type="checkbox"/>

**Fetch Index**

Name: byLocation

Type: R-Tree

**Partial Index**

Predicate: Predicate

# Better Indexing

iPhone X | HistoryDemo | Build HistoryDemo: **Succeeded** | Today at 12:42 PM

HistoryDemo.xcdatamodel

HistoryDemo > HistoryDemo > History...amodeld > History...tamodel > E Post > I byLocation

**ENTITIES**

- E Post
  - I timestampIndex
  - I byLocation**
  - E PostsPerDay

**FETCH REQUESTS**

**CONFIGURATIONS**

- C Default

**Fetch Index Elements**

Property	Type	Expression	Ascending
latitude	Undefined	Expression	<input checked="" type="checkbox"/>
longitude	Undefined	Expression	<input checked="" type="checkbox"/>

**Fetch Index**

Name: byLocation

Type: R-Tree

**Partial Index**

Predicate: Predicate

```
let longPredicate = NSPredicate(format: "indexed:by:(longitude, \"byLocation\") between { %@, %@ }",
                                argumentArray: [ 109.93333333333334, 134.75 ])
let latPredicate = NSPredicate(format: "indexed:by:(latitude, \"byLocation\") between { %@, %@ }",
                                argumentArray: [ 20.233333333333334, 53.55 ])

fetchRequest.predicate = NSCompoundPredicate(type: NSCompoundPredicate.LogicalType.and,
                                             subpredicates: [ latPredicate, longPredicate])
```

```
CoreData: details: SQLite: EXPLAIN QUERY PLAN SELECT 0, t0.Z_PK FROM ZPOST t0 WHERE  
(( t0.ZLATITUDE BETWEEN ? AND ?) AND ( t0.ZLONGITUDE BETWEEN ? AND ?)) ORDER BY t0.ZTIMESTAMP  
DESC  
0 0 0 SCAN TABLE ZPOST AS t0 USING INDEX Z_Post_timestampIndex
```



```
CoreData: details: SQLite: EXPLAIN QUERY PLAN SELECT 0, t0.Z_PK FROM ZPOST t0 WHERE ( t0.Z_PK
IN (SELECT n1_t0.Z_PK FROM Z_Post_byLocation n1_t0 WHERE (? <= n1_t0.ZLONGITUDE_MIN AND
n1_t0.ZLONGITUDE_MAX <= ?)) AND t0.Z_PK IN (SELECT n1_t0.Z_PK FROM Z_Post_byLocation n1_t0
WHERE (? <= n1_t0.ZLATITUDE_MIN AND n1_t0.ZLATITUDE_MAX <= ?))) ORDER BY t0.ZTIMESTAMP DESC
0 0 0 SEARCH TABLE ZPOST AS t0 USING INTEGER PRIMARY KEY (rowid=?)
0 0 0 EXECUTE LIST SUBQUERY 1
1 0 0 SCAN TABLE Z_Post_byLocation AS n1_t0 VIRTUAL TABLE INDEX 2:D2B3
0 0 0 EXECUTE LIST SUBQUERY 2
2 0 0 SCAN TABLE Z_Post_byLocation AS n1_t0 VIRTUAL TABLE INDEX 2:D0B1
0 0 0 USE TEMP B-TREE FOR ORDER BY
```





Test Case '-[HistoryDemoTests.TestFetchPerformance testLibraryFetchWithLocation]' measured  
[Time, seconds] average: 0.434, relative standard deviation: 2.792%

Test Case '-[HistoryDemoTests.TestFetchPerformance testLibraryFetchWithLocation]' measured  
[Time, seconds] average: 0.434, relative standard deviation: 2.792%

Test Case '-[HistoryDemoTests.TestFetchPerformance testLibraryFetchWithLocationIndex]' measured  
[Time, seconds] average: 0.306, relative standard deviation: 3.255%

# Testing With CoreData

# Testing With CoreData

Tests are great!

# Testing With CoreData

Tests are great!

- Learn CoreData

# Testing With CoreData

Tests are great!

- Learn CoreData
- Verify assumptions

# Testing With CoreData

Tests are great!

- Learn CoreData
- Verify assumptions
- Capture product requirements



# Testing With CoreData

Tests are great!

- Learn CoreData
- Verify assumptions
- Capture product requirements
- Communicate your expectations

```
class CoreDataTestCase: XCTestCase {
    override func setUp() {
        super.setUp()
        container = NSPersistentContainer(name: "HistoryDemo")
        container.persistentStoreDescriptions[0].url = URL(fileURLWithPath: "/dev/null")
        container.loadPersistentStores { (description, error) in
            XCTAssertNil(error)
        }
    }

    override func tearDown() {
        container = nil
        super.tearDown()
    }
}
```

```
class CoreDataTestCase: XCTestCase {
    override func setUp() {
        super.setUp()
        container = NSPersistentContainer(name: "HistoryDemo")
        container.persistentStoreDescriptions[0].url = URL(fileURLWithPath: "/dev/null")
        container.loadPersistentStores { (description, error) in
            XCTAssertNil(error)
        }
    }

    override func tearDown() {
        container = nil
        super.tearDown()
    }
}
```

```
class CoreDataTestCase: XCTestCase {
    override func setUp() {
        super.setUp()
        container = NSPersistentContainer(name: "HistoryDemo")
        container.persistentStoreDescriptions[0].url = URL(fileURLWithPath: "/path/to/db.sqlite")
        container.loadPersistentStores { (description, error) in
            XCTAssertNil(error)
        }
    }

    override func tearDown() {
        container = nil
        super.tearDown()
    }
}
```



```
class AppDatabaseTestCase: XCTestCase {
    override func setUp() {
        super.setUp()
        let delegate = UIApplication.shared.delegate
        XCTAssertNotNil(delegate)
        container = (delegate as! AppDelegate).persistentContainer
    }
}
```

```
class AppDatabaseTestCase: XCTestCase {
    override func setUp() {
        super.setUp()
        let delegate = UIApplication.shared.delegate
        XCTAssertNotNil(delegate)
        container = (delegate as! AppDelegate).persistentContainer
    }
}
```

```
public func insertSamplePosts(into managedObjectContext: NSManagedObjectContext) {  
    for _ in 0 ..< 100000 {  
        self.insertSamplePost(into: managedObjectContext)  
    }  
    XCTAssertNoThrow(try managedObjectContext.save())  
}
```



```
public func insertSamplePosts(into managedObjectContext: NSManagedObjectContext) {
    for _ in 0 ..< 100000 {
        self.insertSamplePost(into: managedObjectContext)
    }
    XCTAssertNoThrow(try managedObjectContext.save())
}
```

```
public func insertSamplePost(into managedObjectContext: NSManagedObjectContext) {
    let newPost = Post(context: managedObjectContext)
    newPost.timestamp = Date()
    newPost.identifier = UUID()
    newPost.url = URL("http://store.apple.com")
    newPost.title = self.sampleTitles[arc4random() % self.sampleTitles.count]
    newPost.source = "User"
}
```

```
func testLibraryFetchPerformance() {
    let fetchRequest: NSFetchRequest<Post> = Post.fetchRequest()
    fetchRequest.fetchBatchSize = 200
    fetchRequest.sortDescriptors = [ NSSortDescriptor(key: "timestamp", ascending: false) ]

    self.measure {
        do {
            let context = self.newContainer().newBackgroundContext()
            let frc = self.newFetchedResultsController(with: fetchRequest, context: context)
            try frc.performFetch()
        } catch {
            XCTFail("performFetch threw: \(error)")
        }
    }
}
```

```
func testLibraryFetchPerformance() {
    let fetchRequest: NSFetchRequest<Post> = Post.fetchRequest()
    fetchRequest.fetchBatchSize = 200
    fetchRequest.sortDescriptors = [ NSSortDescriptor(key: "timestamp", ascending: false) ]

    self.measure {
        do {
            let context = self.newContainer().newBackgroundContext()
            let frc = self.newFetchedResultsController(with: fetchRequest, context: context)
            try frc.performFetch()
        } catch {
            XCTFail("performFetch threw: \(error)")
        }
    }
}
```

[bugreport.apple.com](https://bugreport.apple.com)

# More Information

<https://developer.apple.com/wwdc18/224>

---

CoreData Lab

Technology Lab 7

Friday 1:30PM

---

Testing Tips and Tricks

Hall 2

Friday 3:20PM

---

 **WWDC18**