**Realm**

MVCC (multiversion concurrency Control)

**Part 1:**

* Cơ chế lưu trữ: lưu object dạng JSON

* Overriding primarykey function để set primary key cho object
* Dynamic nghĩa là sử dụng Objective-C dynamic dispath

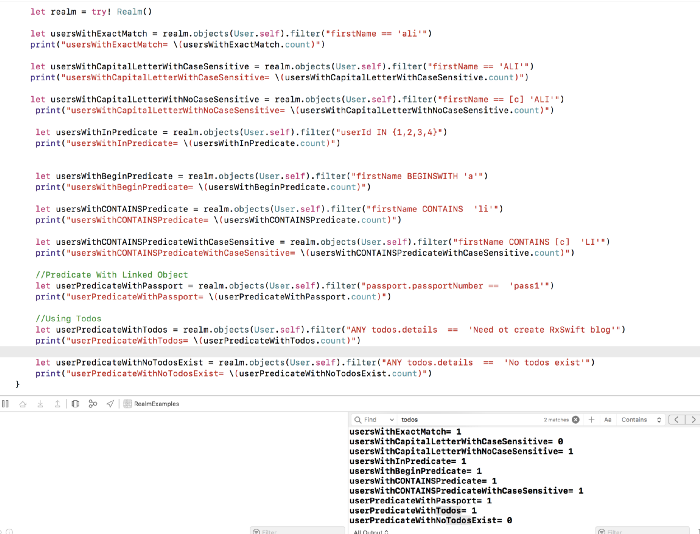
Property được khai báo **dynamic** khi được set giá trị sẽ tự động gọi các hàm **willChangeValue** và **didChangeValue**. 1 property không khai báo **dynamic** mà muốn **observe** ta có thể tự gọi hàm **willChangeValue** và **didChangeValue** khi thay đổi giá trị của property hoặc dùng hàm **setValue** để set giá trị mới cho property

Khi khai báo dynamic thì nó sẽ được gán message dispatch

* Các kiểu quan hệ trong realm: to-one, to-many
* To-one: mặc định nên để là optional
* To-many: mặc định nên để là empty list
* Có một sổ kiểu dữ liệu không được objective c hỗ trợ sẽ không lưu trực tiếp lên realm được (vd: enum, struct)
* String, Nsdata, nsdate có thể khai báo optional.
* Các biến optional khai báo là RealmOptional<>()
* Do Objective C Realm viết trên Objective C nên không có kiểu optional int trong Objective C.
* Các biến khai báo bằng RealmOptional<>() nên được khai báo let
* Khi khai báo realmoptional<>() thì không khai báo @objc dynamic vì trong objectiveC không có generic
* Để khai báo mảng ta dùng List. List không thể khai báo @objc dynamic giống RealmOptional
* Các đối tượng có trong list sẽ có cùng kiểu dữ liệu (Và đều phải implement RealmCollectionValue)
* override indexedProperties để cài thiện thời gian query và filter
* **LinkingObjects** để tạo backlinks đến các object đang trỏ tới nó (có hiệu lực sau khi object được add vào DB hay managed)
* Realm có thẻ lưu dữ liệu trên ổ cứng hoặc trên ram (khai báo qua configuration)
* Realm không phải thread safe và không thể được chia sẻ qua các thread (khi sử dụng realm trong thread khác cần tạo 1 instance mới, vd: instance realm tạo ở main thread không được truyền qua các thread ở background mà phải tạo 1 instance mới ở background)
* Các object, collection, result muốn truyền qua thread khác phải thông qua ThreadSafeReferance

Filter (Predicate):

* **[==] filter →**matches values equal to
* **[==] [c] filter case** **insensitive →**matches values equal to ignore case
* **IN {1,2,3} filter →**matches value from a list of values.
* **[BEGINSWITH] filter →**matches if the firstName value starts with a.
* **[CONTAINS] filter →**matches if the firstName value conatins with li.



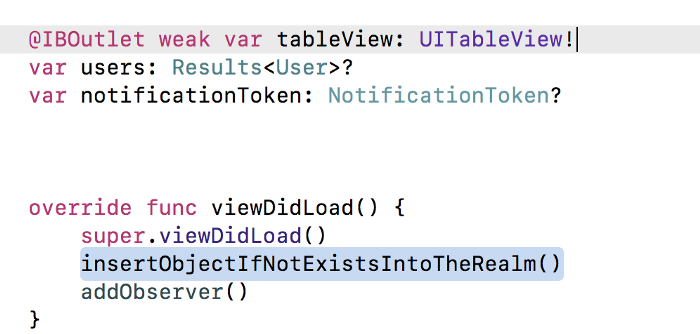
* Live result: results<T> (xem kỹ hơn ở phần 4)

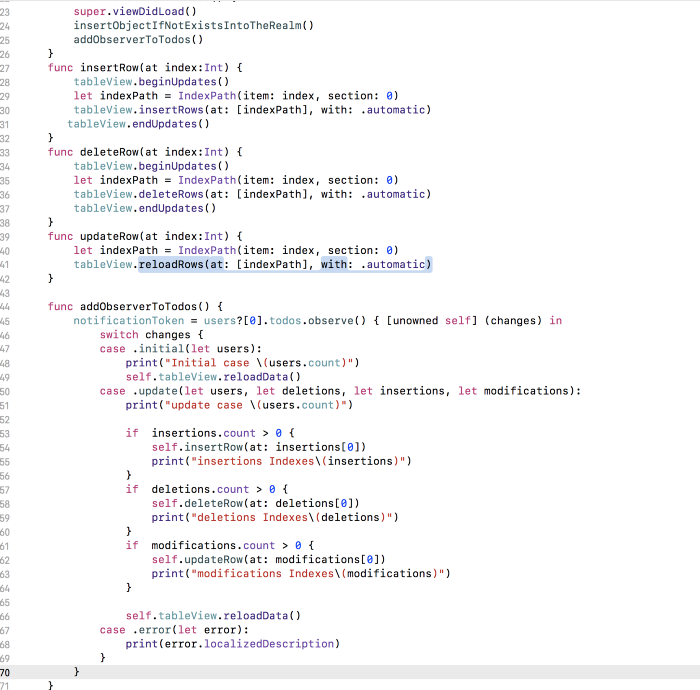
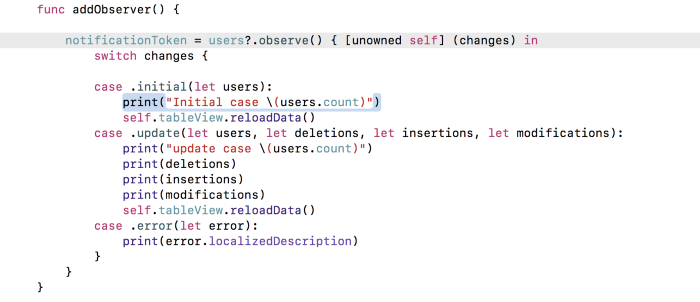
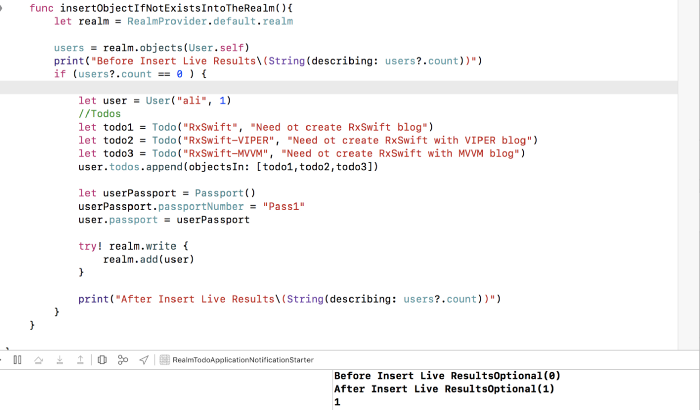
Part 2

**Types of Notification**

* **Object Level →**Observe Realm object for any property change
* **Collection Level →**Observe Realm collections which includes List, Results and linking object properties
* **Realm Level →**Observe changes in whole realm file

Collection level

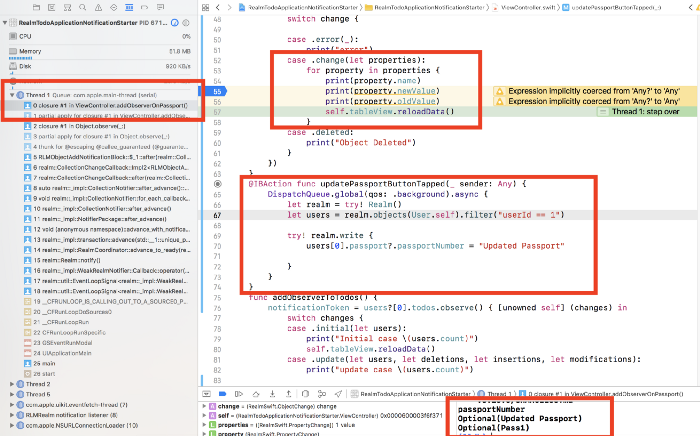




Deletions, insertations, modifications: là list các index

**Object level**



* Initial: Gọi khi lần đầu tiên observer được khởi tạo
* Update:
* Delete: 

Observer ở background thread (concurrency)

class Stuff {

var token: NotificationToken? = nil

var notificationRunLoop: CFRunLoop? = nil

func initNotificationToken() {

DispatchQueue.global(qos: .background).async {

// Capture a reference to the runloop so that we can stop running it later

notificationRunLoop = CFRunLoopGetCurrent()

CFRunLoopPerformBlock(notificationRunLoop, CFRunLoopMode.defaultMode.rawValue) {

let realm = try! Realm()

results = self.getRealmResults()

// Add the notification from within a block executed by the

// runloop so that Realm can verify that there is actually a

// runloop running on the current thread

token = results.addNotificationBlock { [weak self] (changes: RealmCollectionChange) in

// ...

}

}

// Run the runloop on this thread until we tell it to stop

CFRunLoopRun()

}

}

deinit {

token?.stop()

if let runloop = notificationRunLoop {

CFRunLoopStop(runloop)

}

}

}

Thay đổi được tạo ra ở background nhưng observer ở main thread thì observer vẫn được xử lý ở main thread

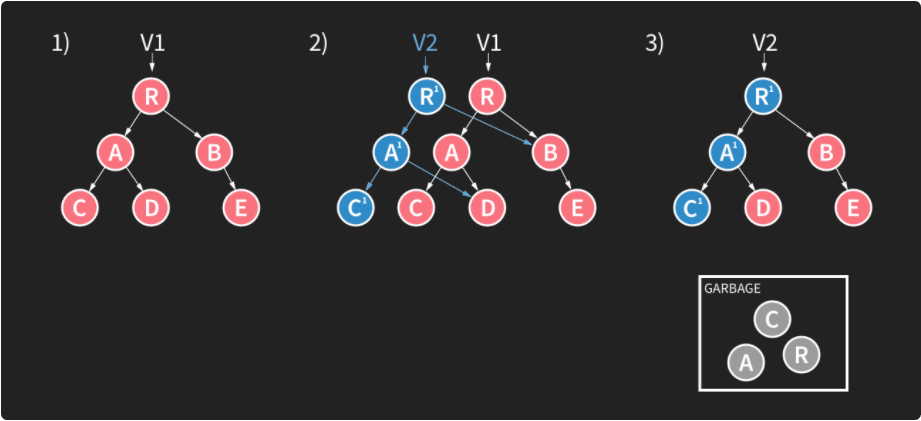
**Part 3: Pre-bundle, encryption**

Các file được lưu ở *<Application\_Data>/Library/Caches or <Application\_Data>/tmp*sẽ không được backup lên icloud

**Part 4: MVCC vs Autofetch**

Chú ý:

* MVCC của realm hoạt động gần giống Git
* Trong 1 lúc chỉ có 1 writer được hoạt động và dữ liệu sẽ được khi trên vistion mới nhất của realm (comitWrite cuối cùng)



* Autofetch:
* Tự động cập nhật lại realm object, results, collection khi bắt đầu run loop
* Đối với background thread, do không có active run loop nên ta phải gọi hàm refetch() để cập nhật lại kết quả.

**Part 5: LightWeight Persistence layer**

Sử dụng 2 lớp object: DAO và DTO

So sánh mô hình 1 model vs mô hình xây dựng kiểu DAO DTO

|  |  |
| --- | --- |
| **Mô mình 1 lớp model** | |
| **Ưu điểm (Pros)** | **Nhược điểm (Cons)** |
| Truy cập trực tiếp đến DB không cần qua mapping | Business layer bị phụ thuộc vào phần DB model |
| Undo và Redo (Core data) | Không thể thay đổi framework (ví dụ từ realm nhảy qua coredata) |
| Live results (realm) | Business layer phải đảm nhiệm thêm xử lý luồng |
|  | Faulted data crashes (Core Data) |
|  | Business layer dealing with heavy models |
|  | Business layer is doing data layer work |
|  | Heavy Models (Models only be Class) |

DTO (data transfer object): model có các thông tin mà view cần (không nhất thiết phải có toàn bộ thông tin về object gốc)

DAO (data access object): nó bao gồm các logic có liên quan đến database như thêm sửa xoá

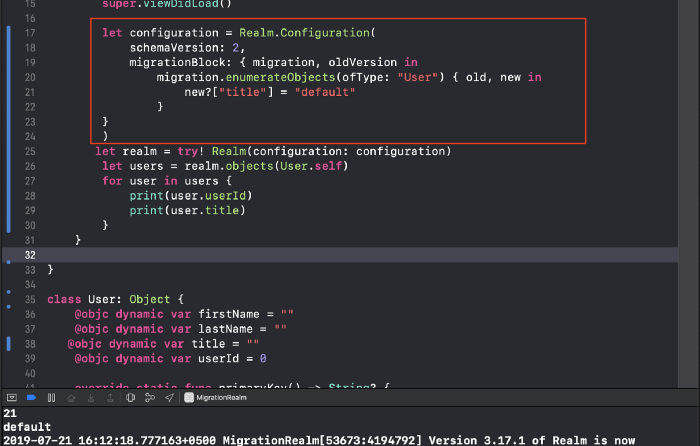
|  |  |
| --- | --- |
| Mô hình DAO DTO | |
| Business layer làm việc với lightweight model | Manually manage object graph |
| Có thể tạo các model bằng struct | Không có các chức năng như live results,… |
| UI tương tác với các lightweight model | Phải xây 2 lớp model |
| No faulted data (core data) | Phải mapping giữa các lớp model |
| No threading Issues in business layer |  |
| Dễ scale |  |

**Part 6:** **Migration**

* Cách 1: Configuration: (deleteRealmIfMigrationNeeded: true) khi cần phải migration thì sẽ xoá DB và tạo db mới, dữ liệu sẽ bị mất
* Cách 2: (khi migration: propreties mới được thêm vào sẽ lấy giá trị mặc định nhưng đối với nhà phát triển realm thì đây là bug)
* Configuration: cài schemaVersion cho realm
* Cách 3: custom migtation

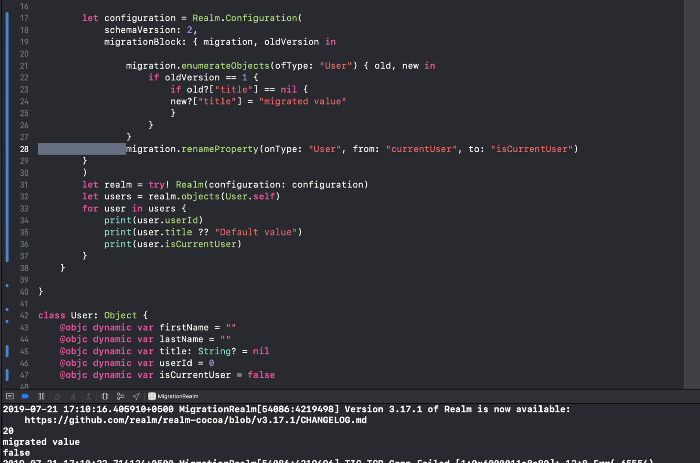
MigrationBlock cung cấp 2 properties: 1 là migration cho phép access đến old và new object schemas và cung cấp các methods để access đến file realm mới và cũ. 2 là oldVersion cung cấp phiên bản cũ của schema.

* TH1: thêm biến mới



new?["title"] = "default"

* TH2: gán giá trị mặc định mới cho biến của phiên bản cũ và đổi tên biến



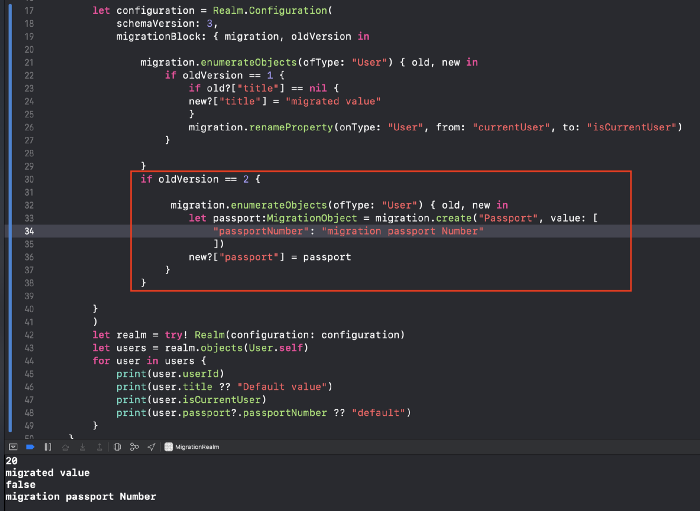
**if** old?["title"] == **nil** {

new?["title"] = "migrated value"

}

migration.renameProperty(onType: "User", from: "currentUser", to: "isCurrentUser")

* TH3: thêm reference đến 1 object mới



migration.enumerateObjects(ofType: "User") { old, new **in**

**let** passport:MigrationObject = migration.create("Passport", value: [

"passportNumber": "migration passport Number"

])

new?["passport"] = passport

}

**Part 8: Problem**

* Việc migration quá lâu có thể khiến watchdog của apple kill app (watchdog timeout: 20s)

Giải pháp:

1. Migration tại Viewcontroller đầu tiên của app
2. Tạo 1 màn hình để show quá trình migrate cho user biết
3. Chuyển việc migration qua thread khác tại didfinishlaunching

Lưu ý: bên trong block migration không đc chuyển qua thread khác

Ghi chú:

* Realm không phải thread safe
* Unmanaged object là object chưa được thêm vào DB
* Đối với các unmanaged object thì việc thay đổi dữ liệu của object không cần writetransaction
* linkingObject chỉ hoạt đồng với managed object
* Realm chạy đồng bộ và blocking thread
* Không thể thay đổi primarykey của object
* Khi xoá object thì property của các object link đến object đó sẽ bị set về nil hoặc remove khỏi list
* Live data: khi dữ liệu của managed object thay đổi sẽ làm cho dữ liệu trong db thay đổi
* (phục vụ cho concurrency)1 biến realm sẽ được refresh tự động tại đầu run loop trừ khi autorefresh bị tắt. Nếu không có run loop thì phải gọi Realm.refresh() để lấy được realm mới nhất Khi gọi Realm.commitWrite() thì realm cũng đc làm mới (đối với realm chạy trên các thread khác nhau, còn đối với cùng 1 thread thì là dữ liệu real time).
* Size của file realm luôn lớn hơn nội dung bên trong nó do realm sử dụng MVCC để quản lý dữ liệu – nó hoạt động gần giống git, dữ liệu sẽ được lưu trên bộ dữ liệu mới nhất.
* Có 2 cách compact file
* Copy dữ liệu ra file realm mới rồi copy file đó ngược lại thay thế file gốc (phải sử dụng autoreleasepool để release reference đến Realm, do compact sẽ bị skip khi vẫn còn có kết nối đến Realm)
* Sử dụng shouldCompactOnLaunch của realm

**let** config **=** **Realm.Configuration(**shouldCompactOnLaunch**:** **{** totalBytes**,** usedBytes **in**

*// totalBytes refers to the size of the file on disk in bytes (data + free space)*

*// usedBytes refers to the number of bytes used by data in the file*

*// Compact if the file is over 100MB in size and less than 50% 'used'*

**let** oneHundredMB **=** 100 **\*** 1024 **\*** 1024

**return** **(**totalBytes **>** oneHundredMB**)** **&&** **(Double(**usedBytes**)** **/** **Double(**totalBytes**))** **<** 0.5

**})**

**do** **{**

*// Realm is compacted on the first open if the configuration block conditions were met.*

**let** realm **=** **try** **Realm(**configuration**:** config**)**

**}** **catch** **{**

*// handle error compacting or opening Realm*

**}**

* Khi write dữ liệu vào DB ví dụ ghi 40000 bản ghi
* Khi tạo write transaction 40000 lần thì thời gian chạy sẽ lâu do realm phải tạo lại transaction nhiều lần cùng với đó sẽ thực hiện các tác vụ khác như notification,… Nhưng về file size sẽ được cải thiện hơn so với ghi 1 lần 40000 bản ghi do các transaction đã được sử dụng lại
* Khi tạo write transaction 1 lần cho 40000 bản ghi thì thời gian chạy sẽ ngắn hơn nhưng file size sẽ lớn hơn cho nó tạo 1 cổng ghi duy nhất cho 40000 bản ghi. File size sẽ lớn hơn ít nhất 8 lần so với dữ liệu thực
* Nên ghi 1000 bản ghi cho 1 transaction để tối ưu
* Nên custom transaction của realm. Có 2 cách custom:

1. Check realm có đang trong transaction hay không (realm.isInTransaction). nếu có thì không khai báo beginWrite nữa.
2. Đưa hết các transaction vào 1 thread để quản lý

Câu hỏi:

* Tại sao String là struct mà vẫn lưu đc vào realm

Do nó implement \_ManagedPropertyType

1 struct chỉ được lưu gián tiếp bằng cách convert qua 1 class và custom lại writetransaction

* linkingObject có hoạt động khi 1 đối tượng unmanage liên kết với 1 đối tượng managed không?

Không

So sánh:

* add vs create:

|  |  |
| --- | --- |
| Add | Create |
| Thêm 1 đối tượng unmanaged vào db | Thêm 1 đối tượng managed ở realm này vào realm khác |
| Thêm được nhiều đối tượng | Chỉ thêm được 1 đối tượng 1 lần |
|  | Cho phép create hoặc update bằng dictionary. Khi update bằng dictionary chỉ những properties có trong dictionary là được update. |
| Chú ý: | Chú ý:   * không đc truyền 1 biến linkingObjects vào * khi create bằng object thì nó hoạt động giống với add |

Giải pháp:

1. phân tầng db hoặc phân quyền db:

vd: khi app lưu db của nhiều người dùng khác nhau có 2 cách để xử lý:

* phân tầng db: đối với mỗi người dùng sẽ tạo 1 file realm khác nhau (ưu điểm: dễ xử lý query, nhược điểm: khó đồng bộ các db của người dùng)
* phân quyền db: tạo thêm bảng user để xử lý query xem data nào là của người dùng (ưu điểm: đồng bộ dữ liệu của các người dùng dùng chung điện thoại, nhược điểm: phải xử lý query)
* tuỳ vào mục đích để chọn phương pháp hợp lý

Link:

Part 1: <https://medium.com/@aliakhtar_16369/crud-operation-using-realmswift-part-1-17a99de83cc1>

Part 2: <https://medium.com/@aliakhtar_16369/realm-notifications-realmswift-part-2-60c66ab99ea9>

Part 3: <https://medium.com/@aliakhtar_16369/realm-custom-configuration-and-encryption-realmswift-part-3-f991f090ae22>

Part 4: <https://medium.com/@aliakhtar_16369/concurrency-multi-threading-in-realm-realmswift-part-4-2345deabe512>

Part 5: <https://medium.com/@aliakhtar_16369/lightweight-persistence-layer-with-realm-realmswift-part-5-7f5a707e35e8>

Part 6: <https://medium.com/@aliakhtar_16369/migration-with-realm-realmswift-part-6-11c3a7b24955>

Part 8: <https://medium.com/@aliakhtar_16369/challenges-problems-in-real-application-realmswift-part-8-6832049bd7de>

Mechanism (thread and notification): <https://academy.realm.io/posts/threading-deep-dive/>

# Break transaction history into smaller pieces: <https://github.com/realm/realm-core/issues/2343>

Tips: <https://blog.usejournal.com/realm-tips-of-an-ios-developer-26ca1654adaf>

What next?

* Tự viết transaction