Realm Mobile Database

Knoxville CocoaHeads

April 2017 by Gavin Wiggins

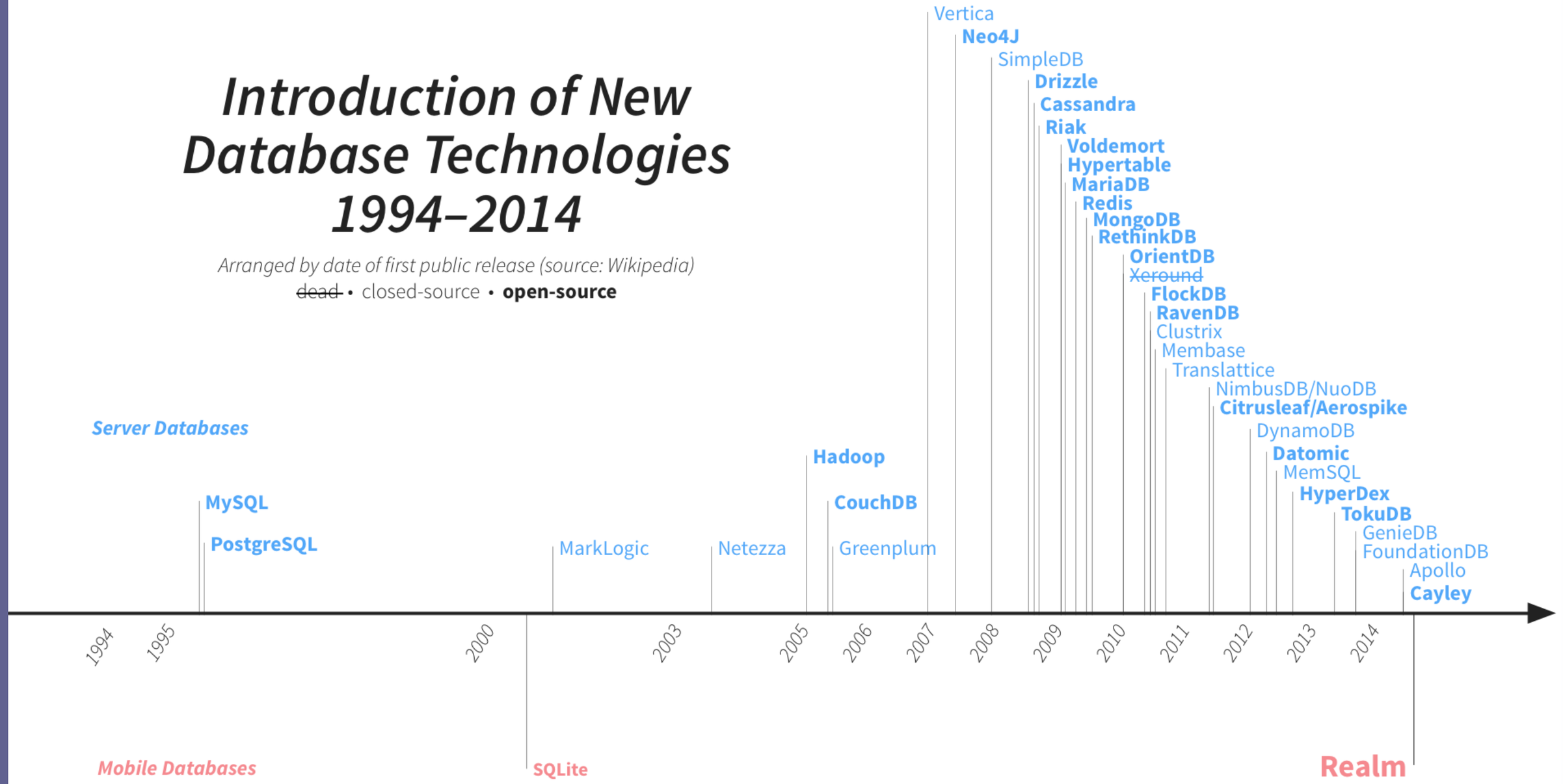
What is the Realm Mobile Database?

- Free open-source mobile database with support for Java, Objective-C, Javascript, Swift, and Xamarin
- Data is persisted to disk as objects (not tables) using a zero-copy design
- Realm objects are full-fledged classes
- Core is written in C++
- Not an ORM database, not based on SQLite
- Support for complex queries and data encryption

Introduction of New Database Technologies 1994–2014

Arranged by date of first public release (source: Wikipedia)

~~dead~~ • closed-source • **open-source**

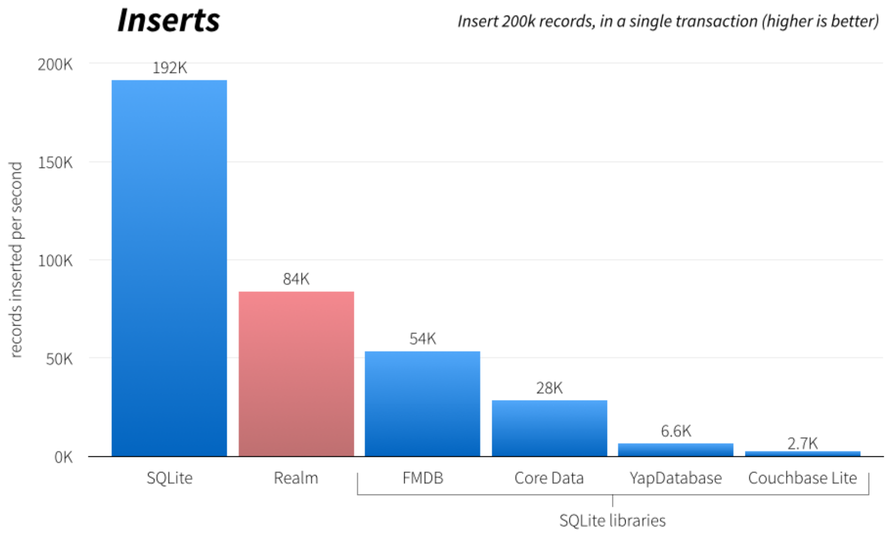


Source: Realm GitHub 2015

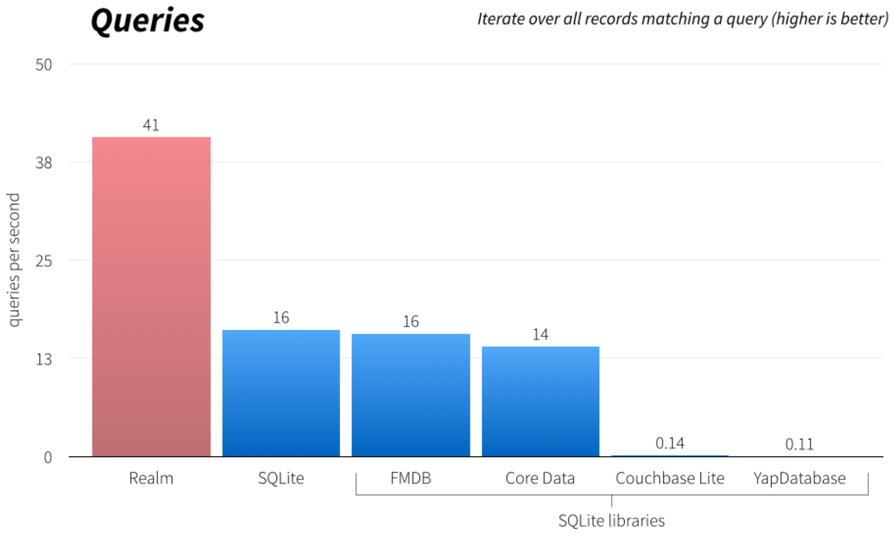
Who is using Realm?

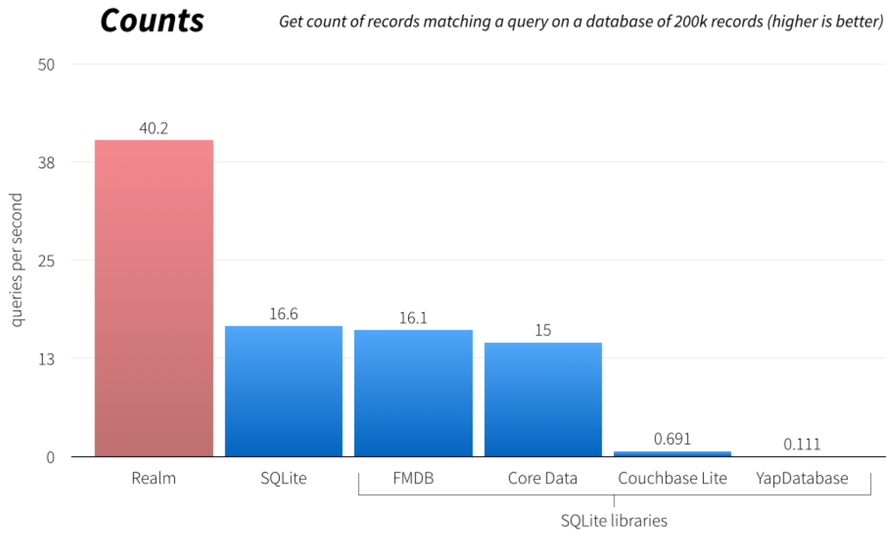
Source: Marin Todorov

Performance



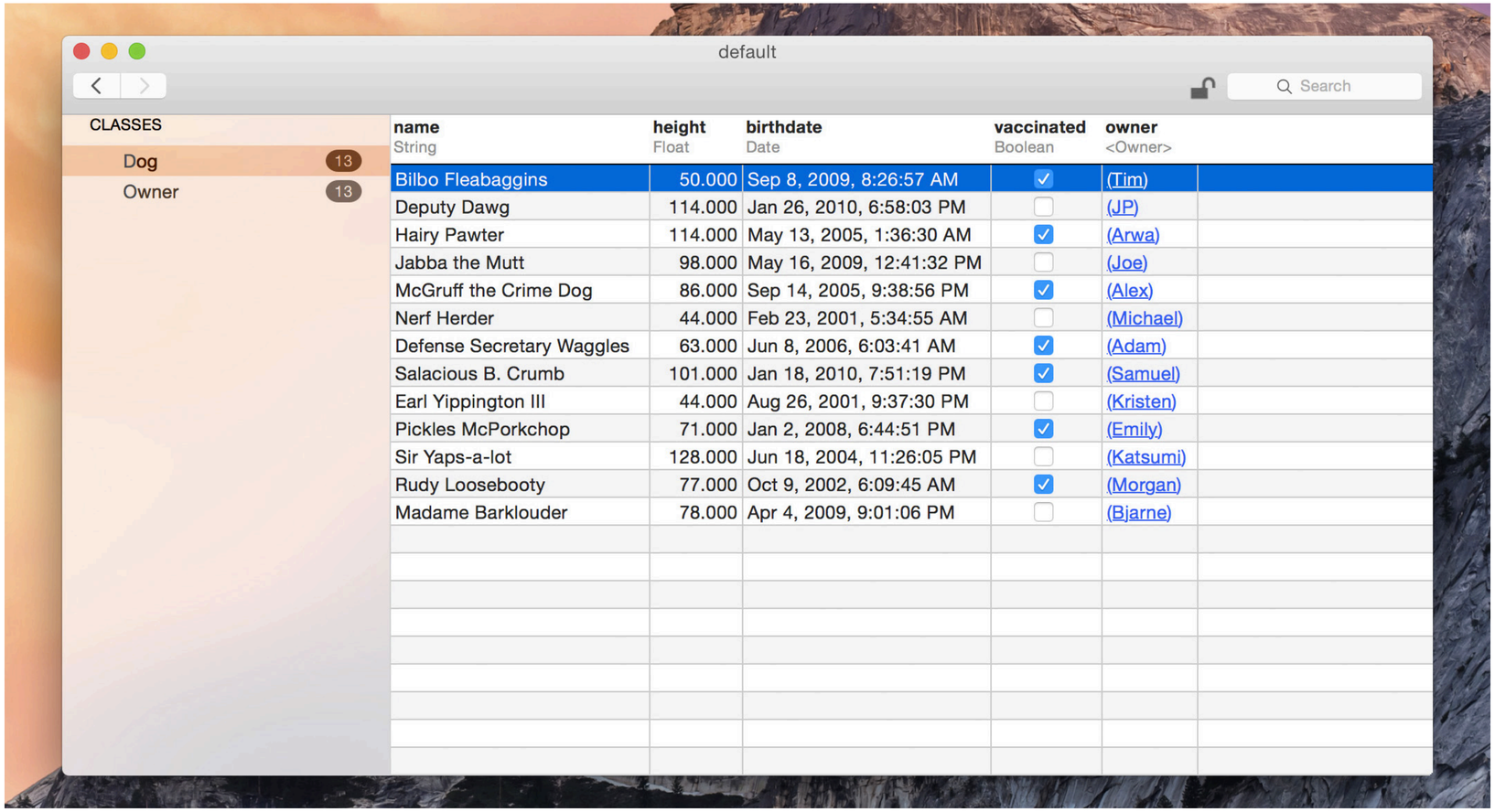
Source: Realm GitHub 2015





Other products - Realm Browser

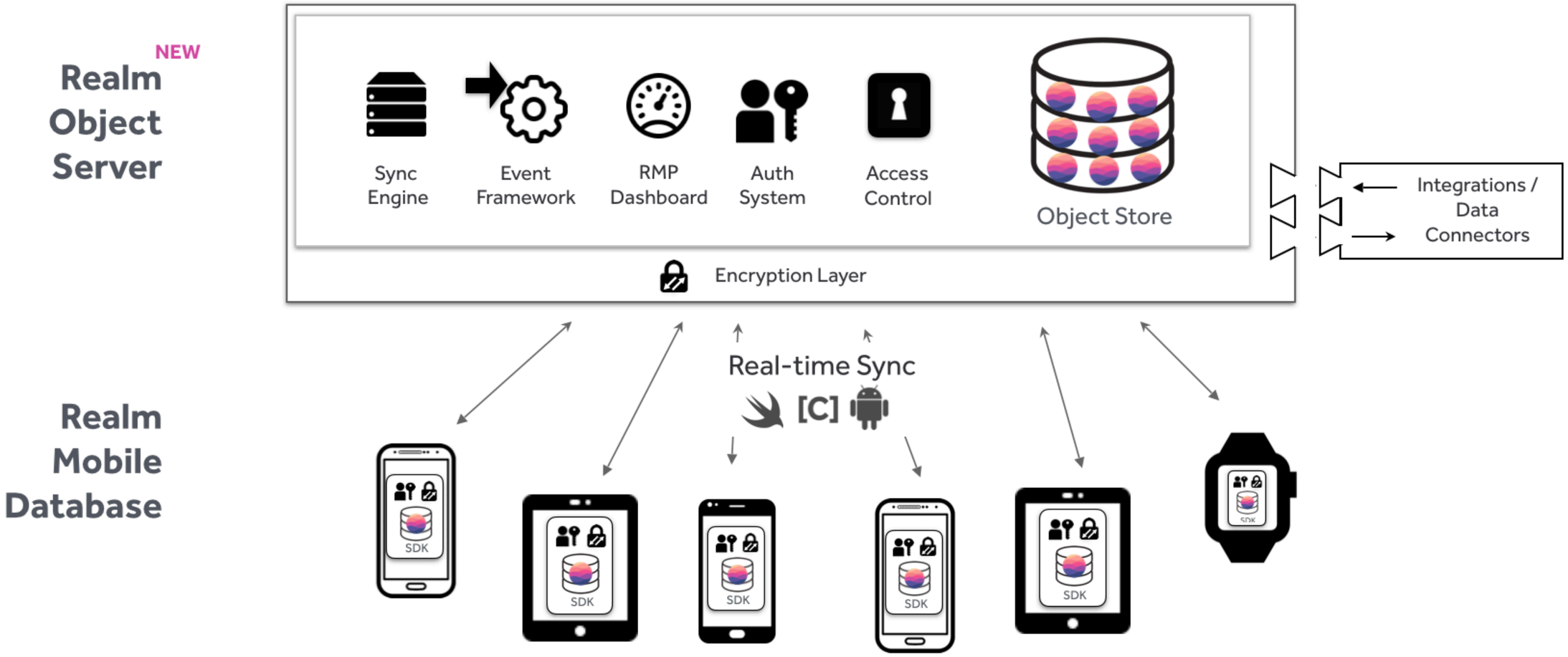
- A standalone Mac app to read and edit realm database files
- Easily view, filter, and debug contents of the realm file in an app
- Available on the Mac App Store



Source: App Store

Other products - Realm Mobile Platform

- Realtime data sync and event handling between server and devices
- Combines the Realm Object Server with the Realm Mobile Database



Source: Realm website

Installation options for Realm Mobile Database

Dynamic framework

CocoaPods

Carthage

Use `import RealmSwift` at the top of your Swift files.

See the Realm documentation at https://realm.io/docs/swift/latest/ for step-by-step instructions.

Define Realm models like regular Swift classes


```swift
import RealmSwift

class Dog: Object {
  dynamic var name = ""
  dynamic var age = 0
}

class Person: Object {
  dynamic var name = ""
  dynamic var picture: NSData? = nil // optionals supported
  let dogs = List<Dog>()
}
```

```swift
// Use them like regular Swift objects
let myDog = Dog()
myDog.name = "Rex"
myDog.age = 1
print("name of dog: \(myDog.name)")
```

The “dynamic” keyword allows Realm to map properties to underlying C++ data structure.

To-One Relationships

For many-to-one or one-to-one relationships, simply declare a property with the type of your **`Object`** subclass.

```
class Dog: Object {
    // ... other property declarations
    dynamic var owner: Person? // to-one relationships must be optional
}
```

```
let jim = Person()
let rex = Dog()
rex.owner = jim
```

To-Many Relationships

You can define a to-many relationship using **`List`** properties.

```
class Person: Object {
    // ... other property declarations
    let dogs = List<Dog>()
}
```

```
let someDogs = realm.objects(Dog.self).filter("name contains 'Fido'")
jim.dogs.append(objectsIn: someDogs)
jim.dogs.append(rex)  // list properties preserve their order
```

Inverse Relationships

Links to other objects are unidirectional.

With **`LinkingObjects`** properties, you can obtain all objects that link to a given object from a specific property.

```
class Dog: Object {
    dynamic var name = ""
    dynamic var age = 0
    let owners = LinkingObjects(fromType: Person.self, property: "dogs")
}
```

Realm property types

Type	Non-optional	Optional
Bool	`dynamic var value = false`	`let value = RealmOptional<Bool>()`
Int	`dynamic var value = 0`	`let value = RealmOptional<Int>()`
Float	`dynamic var value: Float = 0.0`	`let value = RealmOptional<Float>()`
Double	`dynamic var value: Double = 0.0`	`let value = RealmOptional<Double>()`
String	`dynamic var value = ""`	`dynamic var value: String? = nil`
Data	`dynamic var value = NSData()`	`dynamic var value: NSData? = nil`
Date	`dynamic var value = NSDate()`	`dynamic var value: NSDate? = nil`
Object	n/a	`dynamic var value: Class?`
List	`let value = List<Class>()`	n/a
LinkingObjects	`let value = LinkingObjects(fromType: Class.self, property: "property")`	n/a

Adding objects to a Realm


```
// Create a Person object
let author = Person()
author.name = "David Foster Wallace"

// Get the default Realm
let realm = try! Realm()

// Add to the Realm inside a transaction
try! realm.write {
  realm.add(author)
}
```

Writes will block the thread they are made on.

Reads are not blocked while a write is in progress.

Updating typed objects and objects with primary key

Typed updates

```
// Update an object with a transaction
try! realm.write {
  author.name = "Thomas Pynchon"
}
```

Creating/updating objects with primary keys

```
// Creating a book with the same primary
// key as a previously saved book
let cheeseBook = Book()
cheeseBook.title = "Cheese recipes"
cheeseBook.price = 9000
cheeseBook.id = 1

// Updating book with id = 1
try! realm.write {
  realm.add(cheeseBook, update: true)
}
```

Deleting objects in a Realm database


```
// let cheeseBook = ... Book stored in Realm

// Delete an object with a transaction
try! realm.write {
  realm.delete(cheeseBook)
}
```

```
// Delete all objects from the realm
try! realm.write {
  realm.deleteAll()
}
```

Queries

- Queries return a **`Results`** instance which contains a collection of Objects
- Results of a query are not copies of your data
- Execution of query is deferred until results are used
- Results of a query are kept up-to-date with changes in the Realm

```
let dogs = realm.objects(Dog.self) // retrieves all Dogs from the default Realm
```

Filtering

Use the **`filter`** method to query for specific objects by passing

- an **`NSPredicate`** instance
- predicate string
- predicate format string

```
// Query using a predicate string
var tanDogs = realm.objects(Dog.self).filter("color = 'tan' AND name BEGINSWITH 'B'")

// Query using an NSPredicate
let predicate = NSPredicate(format: "color = %@ AND name BEGINSWITH %@", "tan", "B")
tanDogs = realm.objects(Dog.self).filter(predicate)
```

Sorting

Sort criteria and order can be based on

- a key path
- a property
- one or more sort descriptors

```
// Sort tan dogs with names starting with "B" by name
let sortedDogs = realm.objects(Dog.self).filter("color = 'tan' AND name BEGINSWITH
'B'").sorted(byKeyPath: "name")
```

Other features and limitations

Other features

- Configure single or multiple Realm databases for a single app
- Threading
- Add Realm objects from JSON data
- Register for notifications when a Realm is modified
- Migrations
- Encrypting the database file on disk with AES-256 + SHA2

Limitations

- `NSData` and `String` properties limited to 16 MB in size
- No auto-incrementing properties

Demo

Resources for learning more about Realm

- Realm.io - official website and documentation
- Raywenderlich.com - beginner and intermediate video tutorials
- github.com/realm
- StackOverflow - use the [realm] tag to filter search
- A six part series by Marin Todorov titled Realm is an Object-Centric, Present-Day Database for Mobile Applications