Analysis and quantitative comparison of storage, management, and scalability of data in Core Data system in relation to Realm.

Tobias Andersson

Högskoleingenjör, Datateknik
2018

Luleå tekniska universitet
Institutionen för system- och rymdteknik
Abstract

When developing applications for iOS devices it is very common to use Apple’s own Core Data database system which is a framework handling database persistence for iOS devices among other things, but since there so many different kinds of applications it might not be the best option to use the same database system every time. Realm is another database system for iOS devices, it is very lightweight and a big rival to Core Data. This work was conducted with the goal of finding differences between the database systems Core Data and Realm that might show that one or the other is better used in some cases. The comparison between the systems was divided into two different parts, one theoretical comparison focused on reading and analyzing documentation and development of a test application. The test application tested time of create, read, update and delete operations in relation to increasing number of objects and increased number of properties in each object. The tests on Core Data were made with two different implementations to get the aspect of time difference based on implementation included. The results were fairly similar on the different operations with a slight advantage to Core Data. The big difference was seen in implementation difficulty and usability. The included features in the database system were also considered. Realm included more of commonly used and important features but Core Data gives the user the ability to add most of these in the implementation, this results in a question of user case.
Abstract  

Introduction  4  
Object-oriented database management systems (OODBMS)  5  
Core Data  5  
Realm  6  
Purpose  7  
Planning  7  

Preparatory study  8  
Realm concept  8  
Core Data concept  8  
Encryption  9  

Test program and method  10  
Test cases and strategy  10  
  Test cases  10  
  Strategy  10  
Implementation  11  
  Core Data objects  12  
  Realm objects  12  
  Measuring time  13  
  Individual and Grouped context save  13  
  Test run  13  
  Create  14  
  Read  14  
  Update  14  
  Delete  14  

Results  15  
Raw data  15  
  Core Data tests  15  
  Realm tests  18  
Graphs  20  
  Create  20  
  Read  21  
  Update  22  
  Delete  23  

Discussion and Conclusion  24  
  iPhone simulator  24  
  2
Introduction

Behind most applications today there is some sort of database where the information that is handled in the application is being persisted. There are many different databases that can be used in applications to achieve unique sets of advantages. This is since databases can use different ways of persisting data with algorithms designed for specific purpose. For example, a database can be designed so that it is optimized for reading data, but that design might prevent it from being fast when updating data in the database.

There is an increasing demand for good data management. SINTEF wrote in an article 2013 that 90% of all data in the world had been created the last two years, by those numbers it is clear how important it is that the data is being managed in a safe and effective way.¹

iOS applications often use Apple’s own Core Data system to persist their data since it is integrated into their development IDE, Xcode, and is sure to be compatible. Core Data is a storage system of its own kind, the system consists of parts that one will not find in most common storage systems. There are many other storage systems for iOS aside from Core Data. One of those is Realm, it is implemented in an entirely different way but can be used for the same applications.

This report consists of a theoretical comparison between the implementation of Core Data and Realm and how the differences might affect use-cases. The theoretical investigation will keep in mind use-cases of applications for multiple users. There are tests implemented and run on both databases to compare performance in CRUD operations when using a very simple implementation and local persistence on iOS devices. CRUD is the name of a standard set of operations which are ‘create’, ‘read’, ‘update’ and ‘delete’. This set of operations is the most basic and commonly used when testing databases. More complex test scenarios are often aimed at a specific use-case but CRUD operations are very general. The test app is developed to get an understanding of how an implementation of these systems is done and maybe some related issues with this and to get some understanding of the performance of CRUD operations. The goal is to pinpoint differences between the two databases and draw conclusions if one or the other is better suited to use in different scenarios.

An important task that databases have is to keep data accessible by multiple users, data stored locally is often very small and not that important. Losing a phone is easy and if the data on the phone has not been synchronized to another storage the data is lost too. There is also the aspect that all online applications that share data between users need a storage space for the data that multiple users can access. The central database is one holding all the database for the application and if new users start the application the data is synchronized from the central storage to the local database. When changes are made to the local database they are pushed

up to the central database so that the other users can collect those and in that way interact with other users in the application. All the users’ local databases are constantly synchronizing towards the central database to keep everything up to date.

This report will focus on the general features of the databases but the implementation of the databases are on a local level. This is to limit the work to go hand in hand with the time schedule. The synchronization to central database opens up too many options and require too much more testing.

Object-oriented database management systems (OODBMS)

Object-oriented database management systems are made to work as a supporting tool for persistent data storage when programming in an object-oriented language. The goal is to abstract the database specific queries from the code so that the user works with the objects as usual. It is then the OODBMS job to replicate the objects handled by the application and persisting them in the specific database. There are many systems like this that are implemented in very different ways and have different base systems of persisting the data.

The result is that the developer can be sure that when writing code and working with the different objects that are created, manipulated or deleted, it is replicated by the object database and persisted in the underlying database. This makes development easier, faster and safer since the developer does not have to worry about writing database queries to handle the persistent storage which is often a bit complex and a source of errors.

Core Data

Core Data is Apple’s own solution for database management system for applications using OSX and iOS. Core Data is described as an assisting framework for the model layer, it manages object graphs and can handle persistence. Persisting data is not its main task but when it does, Core Data works with an underlying database but abstracts the operations towards the database for the user. The user works with the objects and the system Core Data manipulate these accordingly in the persistent database. The most used persistence database for Core Data is SQLite but something like XML would also work. This work will focus on Core Data used with SQLite. Core Data was not designed to run on more than one thread at the same time but there is support for the implementation of concurrency control, although when using other app-related technologies that are typically used, this is described as a bit complex.²

With Core Data the user can choose what underlying database to use, SQLite, XML or a custom storage type. Core Data itself is just an assisting framework and does not really conform to the

² https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/CoreData/Concurrency.html
ACID term but when using SQLite as an underlying database which is ACID compliant it can be viewed as if Core Data is ACID compliant. ACID is a collection of four properties related to operations in the database, these properties combined ensures the safety and validity of the data in any case of errors or failure during the transaction. The four properties are Atomicity, Consistency, Isolation, and Durability. Atomicity tells the user that the transaction is either fully completed or not begun at all. Consistency ensure that when all operations have been made the database is still in a valid state. Isolation means that every transaction run as if it is the only user on the database, this is to ensures that the transaction is only working on consistent data. Durability means that completed transactions are always permanent and cannot be lost in case of any failure, completed transactions can be recreated even in case of any problems. This report will focus on the most common use which is with the relational database SQLite, in a way it might seem as an object-relational mapper (ORM) but that is not entirely true.

The underlying structure of Core Data looks is presented in figure 1, with the necessary components to save and manipulate to the persistent storage. The context is like a workspace where the user works with the managed objects, when the user wants to save these changes the context is saved and then the changes are merged with the persistent data in the persistent object store with help from store coordinator. This could as said earlier be stored in a number of different ways. The managed object model represent the models of the objects handled in the application that the user wants to persist.

![Diagram of Core Data structure](image)

**Figure 1.** An illustration of how the full stack of Core Data looks.

**Realm**

Realm is an open source NoSQL database management system with zero-copy, ACID and multi-version concurrency control (MVCC) which enables multiple threads to work concurrently with the database. These are all good and in some cases critical features to have in a database.
Zero-copy speeds up the database in the way that the data is not copied from one memory space to another just so that the application can read it, saved CPU cycles result in faster access to the data. Realm supports the creation of transactions which fulfill the terms of ACID.

Realm supports both Android and iOS along with some other platforms. Realm is an embedded, object-oriented database system that is focused on replacing Core Data and similar systems in the area of mobile storage. When using Realm the user does not have to work with any database queries or similar operations but what is different from most database management systems like it is that Realm does not have any underlying relational database. Realm uses a data container model which holds the actual objects of the application. When the user manipulates the objects in the application, it is directly working on the object in the Realm.

Purpose

The purpose of this report is to compile enough quantitative statistics to be able to analyze and draw conclusions about what data management system is better in different regards. A theoretical study will be conducted and a test program will be made to get some simple test running on Core Data and Realm. The aim of this is to see if there is any difference in performance in any aspect of storing data on mobile units using Core Data and Realm.

The goal is to conduct a theoretical study and to build a test program that generates data and do CRUD operations to compare performance. This is to be able to get enough statistics to compare the two database systems and draw conclusions if one or the other is better in any regard to storing or managing data.

The test program will test the basic CRUD operations with two changing parameters, the amount of data in each object and the total amount of objects in the database.

Planning

The writing of the report will be conducted during the entire period of the work to collect enough theory both to support the claims being made but also to document the things that are happening regarding test program development, results and the discussion of the report.
Preparatory study

The two database management systems that are being compared both fall under the category of OODBMS, but they are built in two very different ways. This chapter will display some high-level information of how to interact with these databases, theoretic information regarding how these systems are implemented and key differences that set them apart.

Realm concept

When using Realm for persistent storage the objects that are being managed by the system are all saved in a space called the Realm. As described by the creators, it is not meant to keep everything in the same Realm, different Realms are used to organize the data and to more easily keep control over storage accessing.

When creating a Realm it can be set up in three different ways, as a local, synchronized or in-memory Realm. These are be meant to be used as it sounds, the local Realm is used for local persistent storage on one device, synchronized Realm is used when more than one user is working towards the database and in-memory Realms are used for local temporary storage.

As mentioned the Realm system stores the data as objects in what you could view as a container of space, called a Realm. Object properties are defined by keys and values and might resemble a document store in that way but the Realm schema gives the user more customization to the values as they can be marked as required or optional and be assigned with default values.

The objects that are used and worked on in the application are objects of type ‘Object’ which is basically a Realm type, this makes Realm able to handle the object in question and that also means that the object class needs to be defined in a way that is compatible with Realm. Fortunately, this is not much different from how it would look if it was not a Realm compatible object class.

Core Data concept

Core Data is a very comprehensive framework which manages the model layer objects of an application. As mentioned before there is an underlying database, which in this project is SQLite, and Core Data works as a big abstraction layer on top of that database so the user does not need to write any real SQL queries. The data models that the user created for the Core Data system are the models which the relation tables are based on. But Core Data does more than abstract the SQL queries, it adds very much functionality for the user and enables the user to build very big and complex databases and make very complicated queries in an easier way.
Encryption

This subject was not planned to be a focus point in this study but in a world of digitalization, it becomes a more and more vital key for database storage, especially when working with sensitive or personal data, so it is worth mentioning.

Core Data does not come with built-in encryption secure enough for any real production deployments. Encryption can be and is recommended to be added on by the user.³

Realm on the other hand comes with built-in encryption that uses a 64 byte encryption key along with AES-256, which stands for advanced encryption standard (AES) and is an encryption method used by NSA along with countless others⁴, and SHA-2 HMAC, which is a secure hash algorithm (SHA) designed by the NSA used together with a hash-based message authentication code (HMAC)⁵. This technique is apparently secure enough to be entrusted by bank apps.⁶

³ https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/CoreData/PersistentStoreFeatures.html
⁴ http://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard
Test program and method

The test bench program will be developed in Swift for iOS devices. It will be programmed and designed only to test the performance when using the two different data management systems in the simplest way possible. The software will be implemented to share as much code as possible, the goal is to only separate the software when it is time to persist the data in the two different database systems. All the tests will run on a simulated iPhone 6S.

Test cases and strategy

The queries being made are the CRUD operations, as they are usually called, consist of the listed actions ‘create’ (C), ‘read’ (R), ‘update’ (U) and ‘delete’ (D). These are the most basic and fundamental actions that are used when handling any data in databases. Most functions and operations in applications end up in one of these functions being used on the database. Since they are so fundamental it was a clear choice that these were the ones to test.

Test cases

Queries that will be made to the database in the test program:
- Create
- Read
- Update
- Delete

Corresponding queries will be made to objects both in Realm and Core Data and will be made as equal as can be achieved even though there might be minor differences one has to make when implementing the tests. The queries will be made with a variation of:
- Amount of data in each object
- Number of objects

Strategy

The persistence methods, create, read, update and delete, of each database system will be separated into one service for each database. This is to keep as much code shared as possible so that only the service need to be swapped when using a different database. The goal is to put as much of the database specific code in a separate file/service and exposing the crud operations to make the two services interchangeable from the outside.

These parameters are tested to see how these two databases can handle scaling. This delimitation is made since other more advanced queries can easily become very subjective, for example, the complexity of implementation.
The parameters being used are set to reenact the real use of databases where there is a lot of data being handled and different users store their data in different ways. Some choose many small objects and others might have another approach with fewer objects that hold more data. That is why there is an interest to see if there might be an advantage in performance when storing the data in a specific way.

The object properties will always have their value set to a string value “test”, the variation when talking about “amount of data in each object” is in regards to how many properties each object has.

**Implementation**

The program was written in Swift. For a simple overview of the system design and components see figure 2. The program has a simple GUI to make it easier to run a wider variety of tests without having to configure the source code. There is one test class written for each database tested which is called when running the test.

For the integration with the database and everything regarding that code, a guide was used to get help and influence with the implementation.\(^7\) The person with the guides had implementation suggestions for each database which helped to use the same technique and style in the separate implementations, in the way that the CRUD operations are embedded in a service class.

Implementation was made in the simplest way possible and no special modifications, like indexing or similar, were made to improve performance in any way.

---

\(^7\) [https://www.youtube.com/watch?v=tP4OGvIUC4](https://www.youtube.com/watch?v=tP4OGvIUC4)

\(^8\) [https://www.youtube.com/watch?v=hC6dLLbfUXc](https://www.youtube.com/watch?v=hC6dLLbfUXc)
All the test code is located in the separate test classes, this includes the calls for CRUD operations and the time measuring.

Core Data objects

The Core Data test class communicate with a persistence service when accessing the Core Data data model. Since Core Data has an underlying SQLite database, all the objects that are going to be persisted need predefined data models to map against the table representations in the database. The data model is the model of the managed object that is going to be persisted with all the properties and the different relationships to other managed objects. When adding a managed object the user need to add an entity in the data model and then specify the properties of the object with their type, this is also where the user can add additional conditions to this entity.

Realm objects

The Realm test class also communicate with a persistence service that exposes CRUD methods and handles the persistence. The test class sends the object it wants to read or modify as a parameter to the persistence service method. The Realm system enters a state of writing to the database when modifying any data, either a create, update or delete operation. Since the database is synchronous it acquires a lock on this data during the time it is in the writing state.
The object is a class file as any other object in Swift but is defined as an object of type “Object” which indicates that it is a Realm object. This makes the object able to be persisted and managed by the Realm system, the object can be used like any other object in Swift and still have added methods in the class. The object properties are defined in the class file, almost like any other, and this works as the data model for Realm. Minor differences in Realm since it is based on C, properties need to be marked as an objective C variable. The variables also have to be defined with a valid value because Realm parses all objects that subclass the “Object” type since these are used as the data models for Realm. This means that if an object has an empty value the Realm database cannot create a model of that object since it cannot define the needed memory space for that undefined value.

Measuring time

CACurrentMediaTime will be used to measure the time of the operations. This specific method of measuring time was used since it is said to be the most accurate of the easily accessible timestamp methods since it is using mach absolute time. The common alternative to this method is CFAbsoluteTimeGetCurrent but there is a warning in the documentation that “Repeated calls to this function do not guarantee monotonically increasing results. The system time may decrease due to synchronization with external time references or due to an explicit user change of the clock.”, with this in mind CACurrentMediaTime seem like a better choice.

Individual and Grouped context save

When implementing the Core Data test one of two key differences is chosen which heavily impact the performance and characteristics of the database interaction. When working with the objects the changes are kept in a figurative container. To persist the data to the database the context needs to be saved, all the changes made in the figurative container are then persisted to the database.

The choice is whether to save the context after each individual change made on the data in the database or to save the context after finishing all the consecutive operations of the same type for instance. Another choice could be to save the context after all the different CRUD operations but then there would basically be no interaction with the database.

Tests were run with first two mentioned implementations to show how big of a difference these two approaches can have.

Test run

When the test method was called the tests were initiated, the test calls the operations in the order, create, read, update and lastly delete. Each operation is repeated the number of

iterations specified when starting the test before next operation is initiated. The time is only measured during the calls towards the database.

Create
Core Data is implemented to save each property separately after setting up object model while in Realm properties are sent into the constructor of the object as when creating any other object. As previously mentioned, Core Data was tested with both individual and grouped context save.

Read
Both databases have built-in methods that were used to read all the objects of the same type from the database.

Update
To update a property of a Core Data or Realm object the object is accessed and the setValue method is called on the object with two parameters, the name of the property which is to be updated along with the value to write for that property.

Delete
For both Core Data and Realm, there is a built-in method for deleting the object which does not need anything more than the object which is to be deleted.
Results
Here is where the test results generated from the test program is presented.

Raw data

Core Data tests
All the raw data results from the tests performed using the Core Data system will be presented below.

Constants:
Number of objects: 1000
Property value: “test”
The context was saved after all operations of the same type (Grouped context saving)

<table>
<thead>
<tr>
<th>Number of properties</th>
<th>Create</th>
<th>Read</th>
<th>Update</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.01270402991</td>
<td>0.001626385958</td>
<td>0.01311401802</td>
<td>0.008469008026</td>
</tr>
<tr>
<td>2</td>
<td>1.36E-02</td>
<td>0.001762711094</td>
<td>0.01566659205</td>
<td>0.009565884946</td>
</tr>
<tr>
<td>4</td>
<td>0.01657719095</td>
<td>0.001696998952</td>
<td>0.01667425199</td>
<td>0.009057443938</td>
</tr>
<tr>
<td>8</td>
<td>0.02041270793</td>
<td>0.002852778067</td>
<td>0.02551801701</td>
<td>0.009295111988</td>
</tr>
<tr>
<td>16</td>
<td>0.03855578601</td>
<td>0.002582176006</td>
<td>0.03900691401</td>
<td>0.01108771202</td>
</tr>
</tbody>
</table>

Table 1.
CRUD test results in relation to the number of properties in each object when keeping the number of objects at a constant value of 1000 and each property value set to “test”.
## Constants:
Number of properties: 1
Property value: “test”
The context was saved after all operations of the same type (Grouped context saving)

<table>
<thead>
<tr>
<th>Number of objects</th>
<th>Create</th>
<th>Read</th>
<th>Update</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.003709979006</td>
<td>0.000297746039</td>
<td>0.00344079698</td>
<td>0.003072137013</td>
</tr>
<tr>
<td>10</td>
<td>4.41E-03</td>
<td>0.0006036999403</td>
<td>0.004462395911</td>
<td>0.003802891937</td>
</tr>
<tr>
<td>50</td>
<td>0.005085003097</td>
<td>0.0007836869916</td>
<td>0.00459376693</td>
<td>0.004175796057</td>
</tr>
<tr>
<td>100</td>
<td>0.005899168085</td>
<td>0.0005145629402</td>
<td>0.004458790994</td>
<td>0.005609882995</td>
</tr>
<tr>
<td>200</td>
<td>0.005899173091</td>
<td>0.0008926350856</td>
<td>0.004049725016</td>
<td>0.004304729053</td>
</tr>
<tr>
<td>300</td>
<td>0.006191047025</td>
<td>0.00109019794</td>
<td>0.004909292911</td>
<td>0.004770086962</td>
</tr>
<tr>
<td>400</td>
<td>0.005824189982</td>
<td>0.001672843006</td>
<td>0.006010676967</td>
<td>0.004595722072</td>
</tr>
<tr>
<td>500</td>
<td>0.008537436021</td>
<td>0.00130042294</td>
<td>0.006293048034</td>
<td>0.006186792045</td>
</tr>
<tr>
<td>600</td>
<td>0.009317430086</td>
<td>0.00127836701</td>
<td>0.008722750004</td>
<td>0.006040254026</td>
</tr>
<tr>
<td>700</td>
<td>0.01151382003</td>
<td>0.001555581926</td>
<td>0.00904557202</td>
<td>0.006934321951</td>
</tr>
<tr>
<td>800</td>
<td>0.01174502599</td>
<td>0.001576868002</td>
<td>0.01097766601</td>
<td>0.007522650994</td>
</tr>
<tr>
<td>900</td>
<td>0.01366642409</td>
<td>0.002005557995</td>
<td>0.01159498398</td>
<td>0.008020460955</td>
</tr>
<tr>
<td>1000</td>
<td>0.01379005099</td>
<td>0.001651165076</td>
<td>0.01271405094</td>
<td>0.008563645999</td>
</tr>
<tr>
<td>1500</td>
<td>0.01864368492</td>
<td>0.002505579032</td>
<td>0.01833330397</td>
<td>0.01195948105</td>
</tr>
<tr>
<td>2000</td>
<td>0.02233381197</td>
<td>0.002724909922</td>
<td>0.02405094495</td>
<td>0.01462132297</td>
</tr>
<tr>
<td>3000</td>
<td>0.03154083202</td>
<td>0.004080470069</td>
<td>0.03565213911</td>
<td>0.02111044608</td>
</tr>
<tr>
<td>4000</td>
<td>0.04631204205</td>
<td>0.005462068948</td>
<td>0.04682039097</td>
<td>0.02948840603</td>
</tr>
<tr>
<td>5000</td>
<td>0.05177436396</td>
<td>0.006821093033</td>
<td>0.05805148301</td>
<td>0.03474181402</td>
</tr>
<tr>
<td>7500</td>
<td>0.07711441605</td>
<td>0.01037115208</td>
<td>0.08820761996</td>
<td>0.0643912399</td>
</tr>
<tr>
<td>10000</td>
<td>0.1016424251</td>
<td>0.012795093</td>
<td>0.1210018899</td>
<td>0.08073495503</td>
</tr>
</tbody>
</table>

Table 2.

CRUD test results in relation to the number of objects kept in the database in each object when each object only have one property and having that value in each object set to “test”. 
### Constants:
- Number of properties: 1
- Property value: “test”

The context was saved after each separate operation on the database (Individual context saving)

<table>
<thead>
<tr>
<th>Number of objects</th>
<th>Create</th>
<th>Read</th>
<th>Update</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.028721179</td>
<td>0.004752528999</td>
<td>0.001829624991</td>
<td>0.002465665981</td>
</tr>
<tr>
<td>10</td>
<td>2.85E-02</td>
<td>0.000284949987</td>
<td>0.025877034999</td>
<td>0.02520912199</td>
</tr>
<tr>
<td>50</td>
<td>0.149278286</td>
<td>0.0003107719822</td>
<td>0.122480364</td>
<td>0.126999766</td>
</tr>
<tr>
<td>100</td>
<td>0.256486513</td>
<td>0.0004187730083</td>
<td>0.278228202</td>
<td>0.265594664</td>
</tr>
<tr>
<td>200</td>
<td>0.490304704</td>
<td>0.0005909779866</td>
<td>0.538942651</td>
<td>0.491174384</td>
</tr>
<tr>
<td>300</td>
<td>0.806417333</td>
<td>0.0006985259824</td>
<td>0.771090014</td>
<td>0.744577235</td>
</tr>
<tr>
<td>400</td>
<td>1.094600717</td>
<td>0.001002475008</td>
<td>0.965839892</td>
<td>0.980862428</td>
</tr>
<tr>
<td>500</td>
<td>1.323231899</td>
<td>0.001147784002</td>
<td>1.291495051</td>
<td>1.264137859</td>
</tr>
<tr>
<td>600</td>
<td>1.513369926</td>
<td>0.001047353988</td>
<td>1.539514403</td>
<td>1.533691061</td>
</tr>
<tr>
<td>700</td>
<td>1.747403166</td>
<td>0.001182963984</td>
<td>1.676346998</td>
<td>1.744253981</td>
</tr>
<tr>
<td>800</td>
<td>1.991931649</td>
<td>0.001232273004</td>
<td>1.914805476</td>
<td>2.007535559</td>
</tr>
<tr>
<td>900</td>
<td>2.172877828</td>
<td>0.001428249991</td>
<td>2.207071257</td>
<td>2.194109818</td>
</tr>
<tr>
<td>1000</td>
<td>2.360498757</td>
<td>0.001466310991</td>
<td>2.493499341</td>
<td>2.390119033</td>
</tr>
<tr>
<td>1500</td>
<td>3.688112674</td>
<td>0.001849010994</td>
<td>3.596204259</td>
<td>3.743479447</td>
</tr>
<tr>
<td>2000</td>
<td>4.920156181</td>
<td>0.002350032009</td>
<td>4.720991345</td>
<td>5.038808991</td>
</tr>
<tr>
<td>3000</td>
<td>7.134737489</td>
<td>0.002945009997</td>
<td>7.397558932</td>
<td>8.319498693</td>
</tr>
<tr>
<td>4000</td>
<td>9.674394569</td>
<td>0.003745058988</td>
<td>9.96703559</td>
<td>13.59830883</td>
</tr>
<tr>
<td>5000</td>
<td>12.23584633</td>
<td>0.004483986695</td>
<td>13.06457171</td>
<td>20.21921955</td>
</tr>
<tr>
<td>7500</td>
<td>19.11748965</td>
<td>0.006439591001</td>
<td>25.58460837</td>
<td>46.54443883</td>
</tr>
<tr>
<td>10000</td>
<td>26.45418867</td>
<td>0.008677995997</td>
<td>46.09853161</td>
<td>97.94772476</td>
</tr>
</tbody>
</table>

Table 3.

CRUD test results in relation to the number of objects kept in the database in each object when each object only have one property and having that value in each object set to “test”.
Realm tests

All the raw data results from the tests performed using the Realm system will be presented below.

Constants:
Number of objects: 1000
Property value: “test”

<table>
<thead>
<tr>
<th>Amount of data</th>
<th>Create</th>
<th>Read</th>
<th>Update</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.273397754</td>
<td>1.91E-05</td>
<td>0.278735558</td>
<td>0.231753497</td>
</tr>
<tr>
<td>2</td>
<td>0.350518357</td>
<td>2.12E-05</td>
<td>0.366340496</td>
<td>0.2409641311</td>
</tr>
<tr>
<td>4</td>
<td>0.4596965411</td>
<td>2.35E-05</td>
<td>0.454418426</td>
<td>0.2756584239</td>
</tr>
<tr>
<td>8</td>
<td>0.49837363</td>
<td>2.29E-05</td>
<td>0.596962107</td>
<td>0.344280329</td>
</tr>
<tr>
<td>16</td>
<td>0.6264997671</td>
<td>1.90E-05</td>
<td>0.7864768869</td>
<td>0.477629637</td>
</tr>
</tbody>
</table>

Table 4.
CRUD test results in relation to the number of properties in each object when keeping the number of objects at a constant value of 1000 and each property value set to “test”.
### Constants:
- Number of properties: 1
- Property value: “test”

<table>
<thead>
<tr>
<th>Number of objects</th>
<th>Create</th>
<th>Read</th>
<th>Update</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.143177305</td>
<td>0.001862036966</td>
<td>0.003527251014</td>
<td>0.0009073770198</td>
</tr>
<tr>
<td>10</td>
<td>0.003857128002</td>
<td>2.31E-05</td>
<td>0.001585683</td>
<td>0.001301490003</td>
</tr>
<tr>
<td>50</td>
<td>0.01008416098</td>
<td>1.95E-05</td>
<td>0.008308193996</td>
<td>0.00590882299</td>
</tr>
<tr>
<td>100</td>
<td>0.014955636</td>
<td>2.70E-05</td>
<td>0.01358231402</td>
<td>0.01151998699</td>
</tr>
<tr>
<td>200</td>
<td>0.04163465</td>
<td>2.04E-05</td>
<td>0.054078452</td>
<td>0.03465026201</td>
</tr>
<tr>
<td>300</td>
<td>0.06037916601</td>
<td>2.21E-05</td>
<td>0.059424198</td>
<td>0.03959211099</td>
</tr>
<tr>
<td>400</td>
<td>0.06210034798</td>
<td>2.02E-05</td>
<td>0.06610507201</td>
<td>0.05387296699</td>
</tr>
<tr>
<td>500</td>
<td>0.087645483</td>
<td>2.06E-05</td>
<td>0.08295472502</td>
<td>0.07245885697</td>
</tr>
<tr>
<td>600</td>
<td>0.114939479</td>
<td>2.32E-05</td>
<td>0.113505758</td>
<td>0.09219753099</td>
</tr>
<tr>
<td>700</td>
<td>0.127782555</td>
<td>2.20E-05</td>
<td>0.140193956</td>
<td>0.111828761</td>
</tr>
<tr>
<td>800</td>
<td>0.162746064</td>
<td>2.61E-05</td>
<td>0.16463856</td>
<td>0.13606763</td>
</tr>
<tr>
<td>900</td>
<td>0.160481618</td>
<td>2.54E-05</td>
<td>0.171929259</td>
<td>0.146595624</td>
</tr>
<tr>
<td>1000</td>
<td>0.181013863</td>
<td>2.06E-05</td>
<td>0.191119187</td>
<td>0.161082944</td>
</tr>
<tr>
<td>1500</td>
<td>0.268491157</td>
<td>2.53E-05</td>
<td>0.293984009</td>
<td>0.243619338</td>
</tr>
<tr>
<td>2000</td>
<td>0.458148063</td>
<td>2.50E-05</td>
<td>0.399151631</td>
<td>0.343791503</td>
</tr>
<tr>
<td>3000</td>
<td>0.573705296</td>
<td>2.46E-05</td>
<td>0.605278099</td>
<td>0.526366717</td>
</tr>
<tr>
<td>4000</td>
<td>0.735401489</td>
<td>2.12E-05</td>
<td>0.825333372</td>
<td>0.74548711</td>
</tr>
<tr>
<td>5000</td>
<td>1.066577012</td>
<td>2.24E-05</td>
<td>1.004627413</td>
<td>0.890347537</td>
</tr>
<tr>
<td>7500</td>
<td>1.470144983</td>
<td>2.98E-05</td>
<td>1.77182211</td>
<td>1.672800563</td>
</tr>
<tr>
<td>10000</td>
<td>1.88961745</td>
<td>2.21E-05</td>
<td>2.115588376</td>
<td>1.916093804</td>
</tr>
</tbody>
</table>

**Table 5.**

CRUD test results in relation to the number of objects kept in the database in each object when each object only have one property and having that value in each object set to “test”. 
Graphs

Create

Graph 1.
Showing graphical representation of the relation between Core Data and Realm results from Table 1 and 3.

Create

Graph 2.
Showing graphical representation of the relation between Core Data and Realm results from Table 2 and 4.
Read

Graph 3.
Showing graphical representation of the relation between Core Data and Realm results from Table 1 and 3.

Graph 4.
Showing graphical representation of the relation between Core Data and Realm results from Table 2 and 4.
Graph 5.
Showing graphical representation of the relation between Core Data and Realm results from Table 1 and 3.

Graph 6.
Showing graphical representation of the relation between Core Data and Realm results from Table 2 and 4.
Delete

**Graph 7.**
Showing graphical representation of the relation between Core Data and Realm results from Table 1 and 3.

**Graph 8.**
Showing graphical representation of the relation between Core Data and Realm results from Table 2 and 4.
Discussion and Conclusion

iPhone simulator

To start out I could discuss the fact that a simulator of an iPhone 6S was used to run all the tests, this was done for convenience reasons since some tests were dependent on source code being changed, for example, the tests in relation to the number of properties. This is also the reason why there were no tests run with more than 16 properties, there was too much work in the source code to be worth to go higher than what was needed to see some trend in the relation. The simulator in Xcode is very sophisticated and as long as all the tests were run in the same environment you can still see the trends in performance changes.

Results discussion

As can be seen in the graphs 1 to 8, Core Data is faster than Realm in everything other than reading operations. Important to consider is that this is when the mentioned grouped context save. When viewing the raw results from the tests with individual context save it is not even comparable to the Realm results since it is so very ineffective, this is why no graphs were displayed of those test results.

The fact that reading was faster on Realm could be changed with something like indexing that can be added to Core Data when adding data to the database. This slows down the process of adding data to the database but improved performance when reading the data. This is very common that you optimize your database in one way or the other by improving one behavior at the cost of another, it all depends on what application is using the database and what operations are most prioritized in that application.

The tests with relation to the number of properties have a more stable relation between Core Data and Realm. Realm seems to be the slower option and increasing the number of properties seems to impact the performance more than the Core Data implementation but the increase in time seems to be very slow growing linear line. The difference between the two is not big when you considering that the time difference is about 0,2 - 0,4 seconds and the time increase is as mentioned a very slowly growing linear increase, the most troubling of the four operations is the update operations which seems to have the biggest growing linear development.

Moving on to the tests in relation to the number of objects handled it is clear to see that the Realm database is heavily impacted in comparison to the Core Data implementation. The increasing time of the Realm operations seem to be linear and has increased from 0 to about 1,5 seconds as the number of handled objects were set at 1 to 10 000. The only case when Realm outperformed the Core Data implementation was in reading objects from the database.
The Core Data operations seem to have a linear increase in time as the Realm database but of a much lower degree.

Something to mention is the fact that these tests focus on the local storage of applications. Most applications have a central database that communicates with all the clients/users of the applications. So when a user makes changes that affect the database, the client sends the changes up to the central database which updates the central database and sends out updates to the other clients. The central database system is the one that handles all the incoming transactions and makes sure that nothing collides and that the database stays intact after each transaction. This communication with a central database adds both time for operations and implementation of the central node and how to handle all incoming transactions. But it would be too much to handle in a work of this scale so that is something for a whole other work.

Core Data discussion

Something that is very important to consider is that the Core Data results used in the graphs and that are even comparable to the Realm results is with the grouped context save implementation. When viewing the results in table 3 it gives a whole other picture of the performance difference between these two. The context saving is not a unique feature for Core Data but is a complex feature but I will get into it a bit later. The thing to think about is that the two tested implementations are just two of many different ways to make it work. This is just one of the things that show how much can be customized and tweaked in the Core Data framework since it is such a comprehensive abstraction framework.

When working on the objects in Core Data it makes all the operations on temporary objects in a sort of container and then when you want to persist these changes made to the database the context is called to be saved. The Core Data framework then merges the changes with the current database content. When working as a single user on a local database this is no problem but imagine multiple people working towards a central database and all users build up a huge container of changes before saving context, this gives the framework a lot more overhead work to be done when persisting the changes with the current data in the database. You can see in figure 3 how the data travels from the code operation to the actual database. Core Data has control from the Managed Object Context and down to the database.
This may not be clear to everyone but real implementations probably do not use either one or
the other way regarding grouped or single context saving. The context can be implemented to
be saved in groups of 10 or 100 objects or scheduled to be saved every 10 seconds. This is
something that the user can choose based on the needs of the application. If the application
does not need real-time data in the database or if the application is used in environments with a
bad internet connection it can be implemented to save context more seldom to minimize data
traffic and overhead work.

The more sporadic approach of updates may mean that the database receives updates from
two sources, user one updates an object and waits for something before saving context and
user two who update the same object, later than user one, but saves context instantly. This
means that when the database receives the updates from user one, the updates on the object
are older and thereby not as relevant as the updates currently persisted. In Core Data this
creates a conflict which is detected by a mechanism called optimistic locking, how the conflict
should be resolved is defined by the user.

Realm discussion

Realm, on the other hand, works directly on the persisted objects, in the sense that when the
CRUD operations are called upon and Realm is at that same moment communicating with the
database. This makes things very easy for the user and by the looks of the results it work really
well. The fact that the Realm objects are basically just regular objects of type Object makes
things very easy when building an application. Though this might just make things confusing
when working on bigger systems where more sophisticated models and such is needed when
planning and developing a big system.
Choosing database

The hard part of comparing these two databases is the fact that they are very different in everything from how they are designed, built and the way they are and can be used. With these things in mind, that each database can be customized in very many ways and that there is such a big difference in performance when implementing the database in a specific way, as seen from the tests in Core Data. I think it becomes a big question about the use-case.

Worth to mention is that Core Data is very compatible and supported when developing iOS apps and especially when working in the IDE Xcode that is Apple’s own code editor. This might push some people towards just using Core Data since it is seen as a bigger hassle to start comparing other alternatives.

Use-case

When I say that the choice is about use-case I mostly think about the size of the app that is being developed. The type of the application also determines how much of a need there is for a persisting database and the complexity of it. Simpler applications might just need a few objects with a couple of simple relations between the objects which might push the developer to look for a more lightweight database system which is fast and easy to implement. But if the application has a lot of objects with more complex relations between them it might be smarter to look for a database system which has better support for more complex data models.

From my experience with getting started with these databases and making this small application, I would say that Realm is the easier and more straightforward system to get started and work with. This seems to be a common opinion when looking at comparisons between Core Data and Realm on the web, people like how easy Realm is to get started with and use since it requires less overhead work and so little code to get started. But when managed complex data models and growing applications with new requirements on the database and models it is commonly recommended to use Core Data.\(^\text{11}\)

But from my experience in working with bigger applications, I would say that if the goal is to make a big and constantly growing application Core Data is a smarter choice. Core Data basically forces the user to have an idea behind the data models and that is often good in a big application to limit users from destroying the database scheme. The importance of database design is something most developers can agree on\(^\text{12}\), the bigger and more complex application the more need for a well thought out database design is needed and in my opinion, Core Data has better support for work of this type.

\(^{11}\) https://medium.com/@hiddenbrains/sqlite-core-data-and-realm-which-one-to-choose-for-ios-database-b12c0cd424df

When looking at the features and technical capabilities of these two databases I can not find any big shortcoming in regards to missing any key features. Both systems have support for key features like ACID transactions and concurrency. But as mentioned numerous times, especially with encryption, there is a common theme that it is easier and more straightforward with Realm but since much is up to the user to activate and manage in Core Data it also gives the user more ability to customize the feature. This brings it back to the discussion about the use-case and from the understanding I have gotten by this work I can say that Realm is to recommend and Core Data is more for the ones that really want to get involved in the database and is willing to wrap their head around a bigger system.